Galaxy Kings

(Detailed Player's manual)

Recent Change History

Date	Change
17.03.2021	Laser hit probability formula
15.02.2021	Fixing typos and some doc management
11.02.2021	Added description of command “set player mode permanent”
03.02.2021	Mini games description and commands
03.11.2020	New rating description
28.10.2020	New fields in GetShips(), AssignmentPart filter, commands attach/detach/tow, GetDistanceToNotColony
04.09.2020	Maximum years in space corrected again
19.08.2020	Little corrections and editions
02.04.2020	Document review and corrections, Game story added
09.02.2020	New fleet payment rule; change user command parameter --code
04.01.2020	New way of joining the game documented
10.10.2019	Missiles launch every battle cycle as lasers (removed description of reloading)
04.10.2019	Description of command ‘set rating for’: added/lost fields
04.09.2019	Removed description for command turn. Explained new way of scheduling
31.08.2019	Removed notion of admin/turn commands
31.07.2019	Information about attacker when your colony has been bombed out
30.07.2019	Updated description for command ‘get report’
18.07.2019	A way of joining the game has changed
10.07.2019	Updated parameter –gender in ‘join the game’ command family
25.06.2019	Updated naming convention for objects: can use ‘_’ and ’-’
23.06.2019	Description of triggers and trigger supporting commands
06.01.2019	Added description for commands ‘guard’ and ‘disband fleet’
02.08.2018	Added description of range generator in command senhaved
…	…

Table of Contents

A few words about the game

An idea of this project has been originated from the old PBEM (Play-by-emal) Galaxy games taking place a while ago (google 'PBEM Galaxy' to find out more). Those who have played these games will find some similarity, but the game's layout, goals, and scale have changed dramatically.

Although, the ability to send orders via emails still supported, the new Galaxy Viewer can talk directly to the server via internet.

The game is turn based, i.e. the game server makes a turn, sends you a report, you analyse the report using an existing Galaxy Viewer application, create your orders and send them back to the server. It reminds a chess game, but you play against many other players at the same time. The game server simulates an expanding universe with already thousands of star systems and a number of races. Some of the races are ruled by the real human players, some - by the server (AI nations). You may join as a new race (or nation). In this case you will be placed in some sector of an existing galaxy, having just one planet and some initial resources to start. Then you will try to research and occupy the nearby systems, design and build various ships, negotiate with your neighbours... You also may join as an existing race which has no owner.

Your goal in the game is to dominate in your galaxy, become a Master or even an Emperor of the Galaxy. To achieve that you need to defeat your neighbours and expand colonizing more and more star systems. You need to build up a strong economy and powerful fleet. Even if you have succeeded in your goal and eliminated all the opponents in your galaxy, the game isn't over for you if you want to continue. You can explore the space, find another galaxy and invade it… Each can use own strategy to succeed.

Because all the players are anonymous, each player can create a specific image of its nation, with its own moral and laws. Players may create alliances, make treaties, use intrigue and treachery.

The Universe is unlimited. It has several galaxies and more than 4500 star systems. The Universe is expanded when necessary to fit new races. Every turn thousands of ships are being built, traveling in different destinations, clashing in hundreds battles and being destroyed. Every turn tens systems get colonized, or invaded or bombed out. It's always something going on...

This game is the best suitable for IT people, especially the programmers, who like writing commands and some scripting code.

The Story

More than 700 years ago started a new age for Humanity – a hyper-drive capable to move space ships between stars was created. People started to research the remote star systems and found many worlds suitable for live. A huge migration to the new worlds begun from the overcrowded Sol and hundreds of colonies were created far in the space. Gradually, many colonies decided to become independent from Sol and declared their own states... Gradually, the states expanded, colonizing more and more star systems... Space aristocracy was established... New baked barons, counts and kings were greedy to extend their territories... Gradually, interests and living spaces of the growing states started to cross and conflicts begun leading to wars... Many kingdoms were destroyed, some others grew up immensely trying to dominate whole galaxies... Gradually, moving into deep space the human space ships run into hostile alien races...

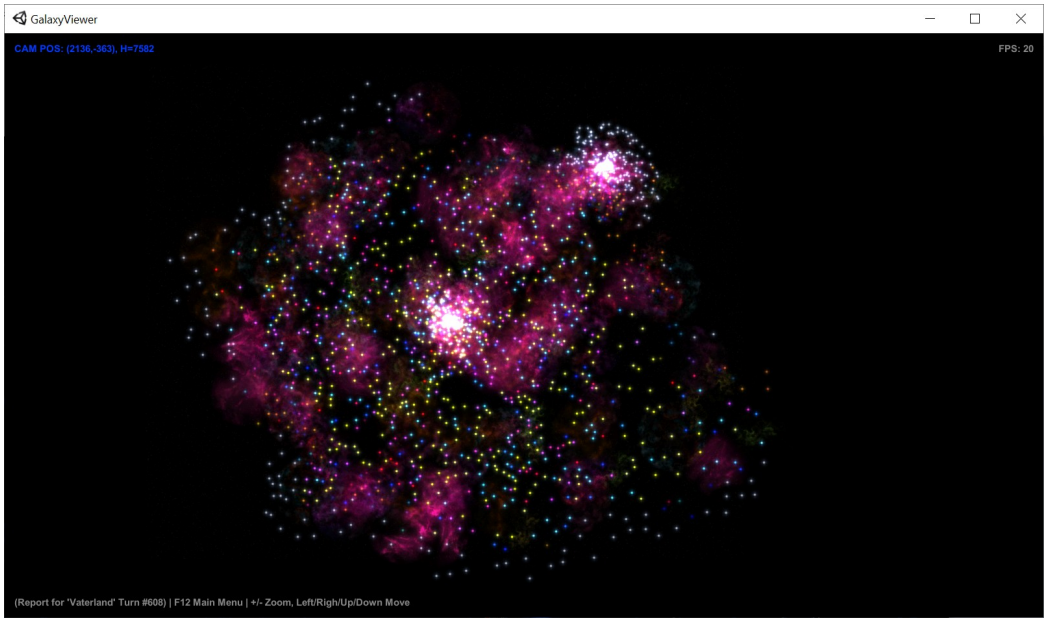
The Simulated World (Universe)

In the very beginning (the Day One, the Turn Zero, or whatever we call it when the world begins...) the Universe is created and it contains just one galaxy. The galaxy consists of a few hundreds stars or Star Systems (a star and a number of planets on its orbit). It is considered, that each star system has planets suitable for colonization. To make things easy, the number of planets is always 1. That is, one star can have only one planet. If this planet is inhabited, the star system is called a 'colony'. So, when we talk about a 'colony' it's a synonym of a star or an inhabited planet the star has on it's orbit.

The stars have following attributes:

- **size** (range 1-10000): rather than physical size of the star it specifies maximum possible population a potential colony may have. The bigger a star is, the bigger it's shown in the report viewer. It is not scientific, but convenient for the game (in the real world bigger stars don't necessary mean they are more suitable for live than smaller stars!). Bigger stars always have more potential for development than smaller ones – choose them for colonization first if there is a choice, although there is always a compromise with 'resource factor' and 'live conditions'.
- **resource factor** (range 0.01 – 1): specifies how rich or poor with resources a potential colony is. Resources are used in production. The bigger resource factor is, the more productive is the colony. Always choose the richer stars for colonization first.
- **live conditions** (range 0 – 1): specifies how suitable is the colony for natural growing of the population. The population grows on 0.1*CurrentPopulation*LiveConditions each turn. That is, each turn a colony may have new-born population from 0 to 10% of original population. Always choose the stars with higher live conditions for colonization first.

The distances are measured in the Light Years (LY). The galaxy stars are generated randomly with sizes between 1 and 2500, filling a circle of a size depending on a number of stars. A minimum distance between two stars is 2LY. A galaxy tends to have the highest density of the stars in the centre, gradually reducing to the edges:



Each new player becomes an owner of just one star, called the Home World (HW) with the following parameters:

1000 <= size <= 2500
0.5 <= resource factor <= 1
0.5 <= live conditions <= 1

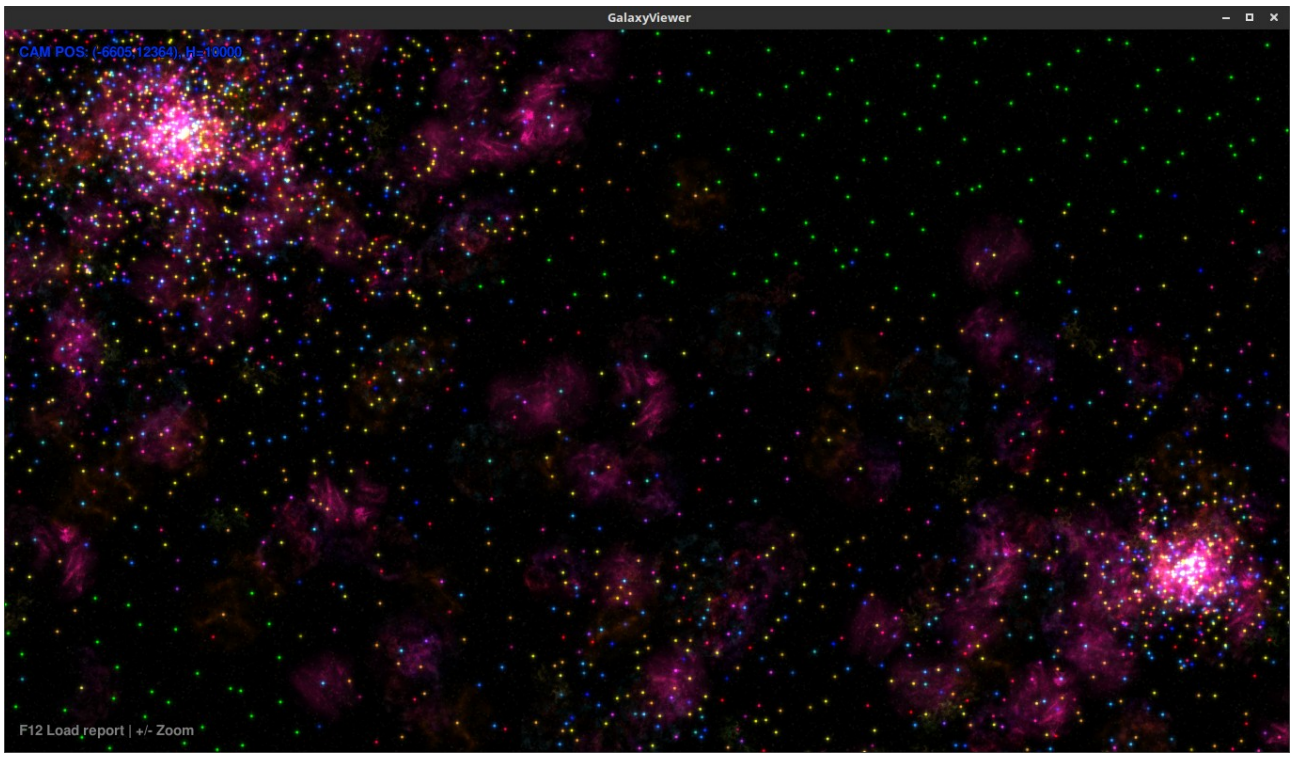
The HW suitable stars are separated from each other by minimum distance of 20LY.

One can see that the start conditions are not equal – some can have bigger, richer and more suitable for life start up colonies then others, but this reflects the reality of a cruel world of the game. The game is an open end – players join and quit. When you join it might be some players with hundreds colonies somewhere “in the galaxy far away...”. So “there is always a bigger fish in the pond...”.

When a new player joins not specifying a HW, the server is trying itself to find a suitable HW. The HW should not just satisfy the parameters mentioned above, it also should not belong to any already existing nation and no ships should be on the colony orbit. In an event when no HW can be found for a new player, the server generates a brand new galaxy of a random size (usually between 100 and 500 stars) and the player is placed there. A new galaxy is placed not closer than 20LY from any existing star.

So, the Universe can expand without limitations depending on a number of playing nations. It never shrinks back, if a number of nations reduces.

A space between galaxies has a low density of stars and is called 'Void'. The Void makes difficult to travel because the distances between stars are big, especially, if TechDrive (your level of DRIVE technology) is not high enough. A positive side of the Void is it's an open space to explore and colonize. Also, Void may contain super big stars, the Giants, with sizes between 2500 and 10000. A galaxy has stars with sizes only between 1 and 2500. The giants always remote from any other star for at least 20LY.



On the picture one can see two galaxies (in top-left and bottom-right corners) are separated by the space with low stars density – that's 'Void'. The long running server tests have shown that nations developed on the galaxy edges had sometimes great success – they had less concurrency and more space to propagate.

However, Void can bring you unpleasant surprises in a shape of hostile alien civilizations, created and controlled by the game server...

Void can be big, stretching for hundreds light years – your ships may need to travel far to discover another galaxy.

Apart from size, resource factor and live conditions, each star has:

- **unique id**: assigned by the server during creation. It never changes
- **name**: when created a star has no name, but it can be set/changed by a player.
- **galaxy name**: each star added during galaxy creation has a specific galaxy name consisting of a letter and a following number, like 'q1','w1', … 'h12'. If the star belongs to the Void, the galaxy name is empty

Measurements and numerical values

The measurements in the game try to simulate the real values. The distances are measured in LY. Each unit of population means 10 million people, unit of materials is 10 million tons,... and so on. Further we will use the term 'unit' to address any measurements in the game. For example: '5 units of population' or '78.89 units of industry'.

All numerical values in reports and orders are represented with 2 decimal places, although the server, internally, uses the high accuracy calculations.

Each game turn corresponds to 1 year.

Money are counted in Galaxy Credits, abbreviated as CREG. That is, one money unit is 1 CREG.

Nations

Nations are treated in the game like countries in the modern world. From the server prospective nations are the players in the Galaxy Kings. Some nations are ruled by real human players, but others have no ruler and controlled by the server itself. They are called AI (Artificial Intelligence) nations. Such AI nations may appear because:

- The server has created them randomly
- Some players have left the game

That is, if a player is leaving the game, it doesn't mean that the nation he has ruled will disappear. The server takes control of such a nation, trying to do its best to develop it further. All AI nations are listed in 'Available Nations' list, so new joiners can pick them up and continue to rule.

Each Nation has:

- Unique name (see Names)
- State flag (see Flags)
- Treasury (specifies how much money the nation has – see Money)
- Technology levels for DRIVE, WEAPON, SHIELD and CARGO (see Technologies)
- List of sciences
- List of ship types
- A number of ships
- A number of colonies

Visibility

The Universe can be big, consisting of many galaxies and thousands of stars. Each player can observe only a visible part of the Universe, calculated as 50*TechDrive from any player's colony or a ship, located on any star. This also means, that your visibility changes nearly every turn depending on developing TechDrive and moving your ships around. It can expand and shrink.

The game flow

The Galaxy Kings is a turned based strategy game. All the turns have unique incremented numbers, starting from 0. The turn Zero (0) is the turn of creation.

The turns take place regularly in known time and can be considered as happening instantly. Each turn ends with sending reports to all the players. Between the turns the players can make an influence on the game process by sending orders, doing diplomacy correspondence with other nations, and so on. Time between the turns can be treated as a preparation for the next turn.

The only turn having no 'preparation phase' is turn Zero.
A client's front-end, such as the Galaxy Viewer, helps to analyse the reports and prepare orders.

Names

In the Galaxy Kings all players, nations, ship types, ships, fleets, stars, sciences have names – mandatory or optional. The names should obey the following rules, otherwise they'll be rejected by the server:

- name length should not exceed 30 characters
- spaces are allowed, but cannot be repeated. Examples:
  - “DeathStar” : no spaces - fine
  - “Death Star” : one space inside - fine
  - “ Bad Star” : two or more spaces – bad

  Heading and trailing spaces are trimmed automatically
- Players and nation names allow only Latin letters. Examples:
  - “Gremlins” : Latin letters - fine
  - “Alex The Great” : Latin letters and spaces - fine
  - “Terminator 2” : has digits - bad
  - “Köln” : has not Latin character 'ö' – bad
  - “Bad_nation”: has not Latin characher '_' - bad
- All other objects such as ship types, ships, fleets, stars and so on allow only Latin letters and/or digits. Also underscores and dashes are allowed the same way as spaces. Examples:
  - “Death Star” : Latin letters – fine
  - “Death_star-2”: fine, using dashes and undescores
  - “Death Star 2” : Latin letters and digits - fine
  - “2nd Fleet” : digits and Latin letters - fine
  - “1234” : just digits – fine
- Names cannot be empty or consist only from spaces:
  - “” : empty name – bad
  - “ “ : only spaces - bad

A player can assign a name to a star, but only if the star is a player's colony. A new star name is accepted only if there is no other colony with the same name belonging to the player. It means, that in the Universe may already exist others stars with the same name belonging to other nations. That is – a star name is unique in a scope of a nation. The name of a star belongs to the nation owner, not to the star itself. It guaranties, that a star can have only one name at a time. The Report Viewer shows star names (if specified) for other players properly, figuring it out from the Universe.

It is wise to refer the stars by unique Id in your orders, because more than one star with the same name may appear in your visibility.

All other objects (ships, fleets,...) are local for the nation and not shared by other nations in the

same way as stars. Each object has an unique name in it's kind.

Population, Colonists and Industry

Each colony has population > 0, otherwise it would not be an inhabited planet! Each turn the population of a colony grows up by 0.1*CurrentPopulation*LiveConditions. Colonies with 0 live conditions have no natural population growing – it must be very hazardous environments where difficult to survive. Population for such colonies may be only transported from other colonies.

The maximum level of population cannot exceed the colony size. Example: colony with size 1000 can have maximum 1000 units of population. The colony size means that the colony cannot feed and employ any more population – no more room to live and work.

What happens to those, who was added during a turn or arrived from other places? Every 10 units of population exceeding a size of the colony are automatically transferred to 1 unit of colonists. The colonists are kept in special containers under freezing temperature. The colonists are supposed to be used to colonize new star systems. When unloaded to a planet, each unit of colonists automatically converted into 10 units of population.

Each planet also has industry >= 0. Industry can be built by your nation or captured by your colonists from the previous planet owners. The level of industry is important in production.

Colonizing star systems

There are only 3 ways of colonizing a star system:

1. Send a colony ship (any ship with a colony unit component) to colonize an uninhabited planet

2. Capture enemy planet by unloading COL from your ships

3. Buy an uninhabited colony (see below)

It is not possible to colonize a planet by unloading COL there. More than that – it is not possible to unload anything on an uninhabited planet.

To create a colony you need a colony unit – it contains all the necessary equipment to build the dwelling, soil cultivation, means of protection and so on. Without all this it’s impossible to create a settlement. It is considered, that a planet captured from the enemy already has all the necessary for live, therefore its colonisation doesn’t require a colony ship.

Production

Production takes place on your colonies, i.e. inhabited stars (planets). Each colony has a production potential (PP), measured in Production Units (PU). The PP depends on a level of population and industry the colony has. The exact formula for calculating the PP looks so:

PP = (Industry > Population ? Population : Industry) + (Population > Industry ? Population – Industry : 0)/4

Only Industry units <= Population can be used in production. This is logical – if there is no population to handle the industry, it will not produce, like a factory without workers. But Population can be bigger than Industry. This part of population can produce nevertheless, but less effectively, giving just 25% of PU they would deliver having industry available.

Examples of calculating the PP:

Colony Size	Population	Industry	Comment
1000	1000	1000	PP = 1000: all 1000 Population units handle all Industry units
1000	400	1000	PP = 400: all 400 Population units handle all available 400 Industry units; 600 remain unused
1000	1000	400	PP = 400 + (1000-400)/4 = 400 + 600/4 = 550: 400 Industry units are handled by 400 Population units, the rest of the population produces just 25% PUs giving 150 from 600
1000	1000	0	PP = 1000/4 = 250: no Industry at all, the rest of the population produces just 25% PU giving 250 from 1000
1000	0	1000	PP = 0: no Population to handle the industry – no production

A colony is considered as 100% developed (reached MAX), when it has Population >= Colony Size and Industry >= Colony Size. Only in this case a colony gives a maximum possible PP.

Production Orders

Whole nation production is controlled by the Production Orders (PO). That is, to produce something you need to order it. There are two types of PO:

- Colony orders (or Star orders) – orders to produce something on a specific colony
- Common orders – orders to produce something without specifying which colony should produce it.

You can place orders to produce the following:

- industry
- materials
- clones
- technology
- science
- ships
- money (sounds strange, but it’s a way to avoid spending some money)

There are also two special orders, consuming PU, but not producing anything:

- ship repair
- ship upgrade

Only orders to produce ships, technology and science can be Common Orders, i.e. orders placed without specifying a colony. Producing industry, materials and clones needs some 'ground' – where the production is to be placed, when technologies are something not-weight – they are just added to your nation properties. The ships after production can be sent to an assembly point you can specify.

Each PO has a unique ID you can use to cancel it later if necessary.

Orders also have a priority – an integer value, by default 0. All orders, when placed, are stored in a priority queue. During production phase of a turn, the orders are executed according their priority: orders with higher priority are executed first.

Obviously, each colony can produce different kind of production during the same turn – it's just a subject of availability of PUs and money. To produce anything every PU needs one money unit (1CREG). As soon as one order has been finished and there are still available PUs and money, next order is taken from the queue for execution. The turn production phase finishes either because there are no more orders to execute, or there are no more production units or money available. In the latter case, the rest of PUs 'produces' money to the National Treasury.

One can look at it this way. All orders you place for production are the 'Government Orders'. You want to produce something for the state: industry, transport or ships, advanced technologies and so on. To produce all this you need to spend money. You order something from the Industry sector and you pay for it. If there are no government orders (or not enough budget to subside it), the Industry sector produces some sort of 'civilian' production (food, entertainment, cars, clothing...) and pays

taxes to the government. So, the less you burden your Industry sector by government orders, the less you spend government money, the more income in taxes comes back to your Treasury. So, money is another implicit type of production, made by production units not executing any orders.

Apparently, you should try to develop your industry very intensively to reach maximum possible production potential. A high PP gives you not only a possibility to deal with the orders queue fast, but also gives you self-generating money to your Treasury. And these money could be used in many different ways (financing the the production again, buying, bribing, helping someone by money and so on).

To produce industry, clones or ships one need to spend equivalent mass of materials. Each planet has some volume of materials >= 0. If materials already present on the colony, they must have been produced earlier, or it might be destroyed industry as result of previous bombing by combat ships. During production phase of a turn, when the server is trying to produce a unit of goods (industry, clone or a part of a ship), an equivalent mass of materials is sought on the planet. If it's present, this mass is deduced from the colony and 'used' for production. If there is not enough materials to produce a unit, the server starts spending some production units to produce necessary materials. Depending on the colony resource level, a number of PUs, spend to produce materials can be different. As soon as enough materials have been produced, they are added to the colony and the server returns to the production of the unit of goods, postponed by materials production.

All this means, that colonies with poor resource level are not good for production. They ether need materials to be transported from other richer colonies, or they should just focus on production of something, not requiring materials at all, such as technologies/sciences, or... money (just do civilian production and pay taxes to the government!)

Production orders can be cancelled using command '`cancel prod order`' (see Commands). If cancelled order has something 'half done' (for example not finished ship), the spent materials are returned to the planet where the order has been executed, however, spent PUs and money are lost.

Production Loop

During production phase of a turn the server processes the orders priority queue in the following way:

1. It checks an order with the highest priority assigned to any colony. If there is such an order the following is happening. While there is enough PUs available in the colony and enough money in the Treasure to finance the order production, server spends PUs on production until the order is finished or PUs or money are over. A current order might be only partly done, but the ready part is delivered. For example, you have an order to produce 100 units of industry, but only 3.62 have been produced. These 3.62 production units are added to the colony and will start working in the next turn. If money are over, whole production phase stops. If PUs on the colony are over but there is still money available, production stops for this colony and another order with the highest priority is looked for next available colony

2. If there are no more orders assigned to colonies which can be executed are found, but still there are colonies with PUs available and there are still money in the Treasury, the server checks the common orders. It takes the one with the highest priority and distributes it to the colonies still capable to produce. The big orders can be split on smaller ones and assigned to several colonies. If that happened item 1. is repeated again until all is done or PUs or money are over. Then again item 2. is repeated.

The production loop can stop only by 2 reasons:

- No more PUs available on any colony or no more money in the Treasury
- All orders are done

When the production loop stops, all not used PUs (if there are such) implicitly 'produce' money for the Treasury.

Materials

Materials are necessary to produce Industry, Clones and Ships. Sometimes you may need to produce materials themselves. The most obvious reason for that is to produce a volume of materials enough to build Industry on a neighbour colony with very poor resource factor. You might want to transport then the produced materials to that colony and build Industry there to increase the PP of the colony.

To produce 1 materials unit required 1PU/ResorceLevel. That is, on a planet with resource level 1 you need just 1PU to produce 1 unit of materials. If resource level is 0.1, the number of PU rises to 10; with 0.01 (the lowest!), you would need to spend 100 PUs to produce 1 material unit. Such a waste...

Server implicitly spends required number of PUs to produce materials to cover production orders during production phase of a turn.

Material orders should be always associated with a colony. Cannot be 'common' material orders.

Industry

Industry is a stone corner of your national production potential. It's a good idea to maximise the PP of your colonies before starting to produce anything else.

To produce 1 industry unit required 5 PUs and 1 unit of materials.
Industry orders should always be associated with a colony. Cannot be 'common' industry orders.

Clones

You may want to produce clones to create additional population on your colony. The most obvious reason for doing this is your colony has industry, but not enough population to reach the maximum PP. Also, this colony may have very low live conditions making the natural grow of population extremely slow.

Solution: you can 'produce' population by making 'clones'. The produced clones don't differ from normal population and simply added to the colony population.

To produce 1 clone unit required 3 PU and 1 unit of materials.

Clone orders should always be associated with a colony. Cannot be 'common' clone orders.

Technologies

When you join the game with a new created nation you start with technology level 1 in following areas:

- DRIVE
- WEAPON
- SHIELD
- CARGO

To increase a technology level one need to 'invest' a number of PUs for developing the specified technology area.

Depending on a number of PUs spent for the technology, the level is calculated as:

$$ln(1 + PU(10000 + PU/55))$$

where:

PU – number of spent PUs

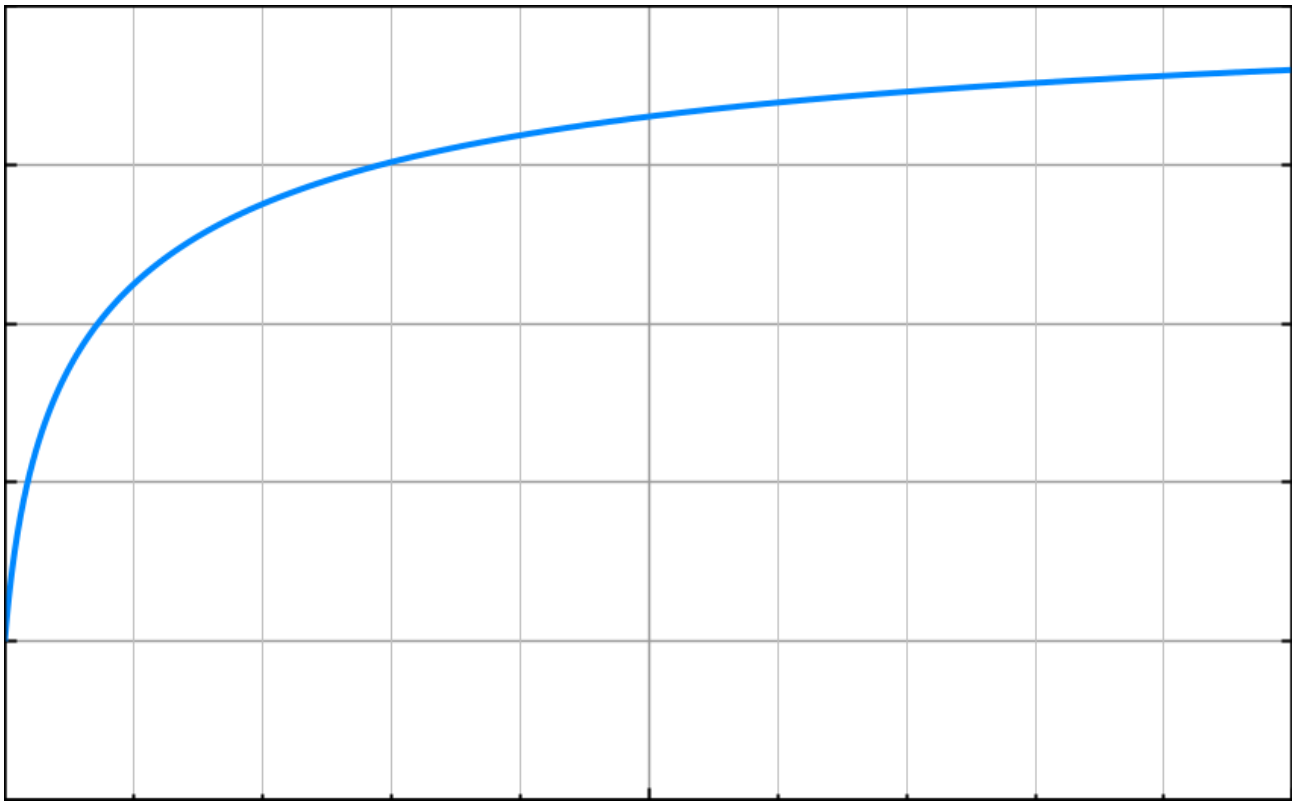
One can see, that a 'speed' of increasing technologies is slowing down gradually: the same increase needs more and more PUs 'invested'.

Example of investing 1000 PUs every turn for a selected technology, started from level 1:

PUs spent a turn	Totally PUs spent	Technology increased on	Absolute Technology level
0	0	0	1
1000	1,000	0.095145177	1.095145177
1000	2,000	0.086572333	1.18171751
1000	3,000	0.079394058	1.261111568
1000	4,000	0.073295619	1.334407187
1000	5,000	0.0680504	1.402457587
...	...	...	...
...	10,000	0.281720924	1.684178511
...	100,000	1.563056487	3.247234998
...	1,000,000	1.349635274	4.596870272
	10,000,000	0.375922313	4.972792585
	100,000,000	0.047172385	5.01996497
	1,000,000,000	0.004846693	5.024811663

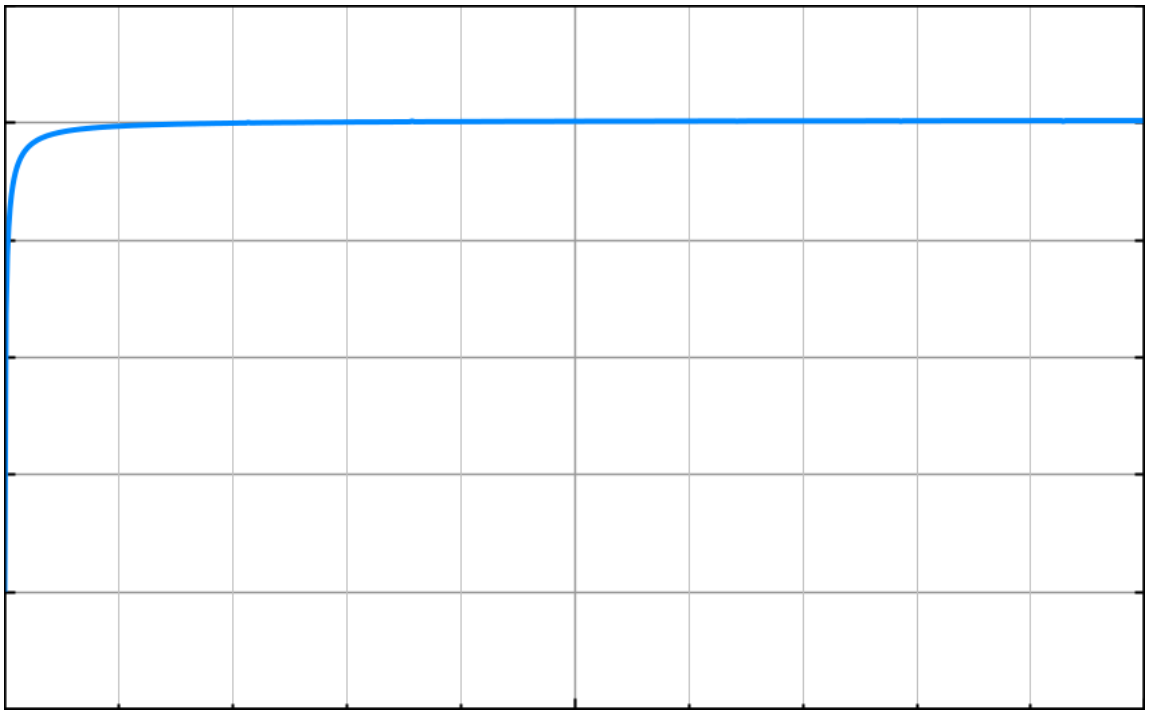
In the beginning the tech level grows fast, than slower and slower. You need to spend 1,000,000 PUs to reach the tech level 4.59

The tech formula graph can be seen on the picture below:



Each level on X is 100,000 of PU invested. Each level on Y is an actual value of technology you get. The last value of X in this graph is 1,000,000.

Let’s have a look at the next picture where each level of X is 10,000,000:



You may notice, that after investing 10,000,000 in a technology it doesn’t really grow any further. Such logarithmic grow of technologies has been done to avoid of appearing super developed, absolutely invincible nations (which would appear had we linear growth). So, after reaching level 5-ish it’s better stop developing the tech – you have reached the maximum.

The technology level **FOREVER** remains with your nation – it never decreases. Your ships can be shot down, your industry can be bombed or captured, your population can be killed, but the techs will never become lower.

It's always a good idea to develop DRIVE first to make your ships faster. It's crucial for your nation development and defence. The transport ships will deliver cargo faster, the combat ships will assembly faster in required location.

Then you would probably think about developing WEAPON and SHIELD technologies. It's a good approach to invest in both of them approximately equal parts of production units. The WEAPON technology makes your ship weapons more destructive, and the ships with higher SHIELD tech level are better protected against hits.

CARGO technology increases ability of your transport ships to move cargo from one location to another. You need less transport ships to deliver the same amount of goods with higher CARGO technology.

Production of technologies doesn't require any materials. It is wise to place the technology orders for the colonies with low resource factor.

Technology production orders can be placed without specifying a colony. The server will find automatically where the best to place the star orders – it will try to find the poorest (by resources) colonies first, then, any available colonies. The Common Orders are executed by the server not at once, but by reasonable parts, depending on available PUs and money. It is wise to trust the server here, especially if a number of colonies you have is large.

Sciences

It is possible to create a science to develop several technologies at the same time (in the same production order). Science is a named record of 'parts', specified for each technology:

<Science Name> <DRIVE>,<WEAPON>,<SHIELD>,<CARGO>

Examples:

Drive 2,1,1,0 means science 'Drive' with 2 parts for DRIVE, 1 part for WEAPON, 1 part for SHIELD and 0 parts for CARGO what gives you percentage of production order PUs used as 50% for DRIVE, 25% for WEAPON and 25% for SHIELD, and 0% for CARGO.

Defence 1,2,5,0 means science 'Defence' with 1 part for DRIVE, 2 parts for WEAPON, 5 part for SHIELD and 0 parts for CARGO what gives you percentage of production order PUs used as 12.5% for DRIVE, 25% for WEAPON and 62.5% for SHIELD, and 0% for CARGO.

Transportation 1,1,1,7 means science 'Transportation' with 1 part for DRIVE, 1 parts for WEAPON, 1 part for SHIELD and 7 parts for CARGO what gives you percentage of production order PUs used as 10% for DRIVE, 10% for WEAPON and 10% for SHIELD, and 70% for CARGO.

Having a science created you may always place an order to develop it.

It is also possible to place an order to develop an 'anonymous' science, which exists only for this particular order, something like:

```
dev science --prop 10,15,25,50 --amount 100000
```

(develop a science with proportions 10 for DRIVE, 15 for WEAPON, 25 for SHIELD, and 50 for CARGO and spending 100000 PUs)

Money

Money production is not really a production. A money production order just directly transfers a given amount of PUs to the same amount of money. It's a kind of useful trick to make the production system generating money not after the production turn, when all not used PUs are transferred to money, but during the turn.

Also, high priority money orders can be placed deliberately to temporary delay other production orders or avoid any production on selected star systems.

Ships design

You can design your own types of the space ships. To create a new ship type you need to give it a name and specify components the ship will consists of. There are following ship components available:

- **Drive**: specifies a power of the ship's hyper-drive. A ship cannot have more than one drive.
- **Shield**: specifies a power of the shields generator. A ship cannot have more than one shield generator.
- **Laser cannons**: specifies a power and number of the ship's laser cannons. A ship can have many laser cannons with different power and number of barrels.
- **Missiles**: specifies a power and number of the ship's missiles launchers, and a number of 'shots' for each launcher. A ship can have many missile launchers with different power and number of missiles.
- **Cargo**: specifies a size of the ship's cargo bay. A ship cannot have more than one cargo bay.
- **Repair Unit**: specifies a size of the ship's repair facility. A ship cannot have more than one repair unit.
- **Colony Unit**: having this component, a ship can be used to create a new colony on an uninhabited planet. A ship cannot have more than one colony unit.
- **Tow Unit**: specifies a size of a tow component. With this component the ship can tow another ship.

A ship type must have at least one of the components listed above, but not more than 50 components altogether. Maximum ship component mass cannot exceed 30000. Full ship mass is a sum of its components masses + mass of cargo in the cargo hold.

Each ship component has Efficiency – a float value from 0 to 1 (meaning from 0 to 100%). Efficiency determines how healthy the component is or how bad the damages are. A component can be damaged or destroyed completely as result of enemy fire. Zero efficiency means the component is destroyed, 100% - component not damaged at all.

Drive

A ship can move only if it has a Drive component. Ship speed (a distance in LY a ship can make during a turn) is calculated by following formula:

*Speed = 20 * DriveEfficiency * DriveMass * TechDrive/FullShipMass*

Without drive (*DriveMass = 0*) a ship will always remain on a planet orbit.

A ship has a maximum possible speed when it has a drive component only, no matter how big the drive is.

*MaxShipSpeed = 20 * DriveEfficiency * TechDrive*

What gives you 20 with 100% efficiency and TechDrive=1. Obviously, if DriveEfficiency is 0.5 (50%), the maximum speed will be only 10LY.

Ships containing only a drive component with mass = 1 are perfect scout ships. They are the fastest possible and the cheapest to produce.

Drive power is a real value >= 1.0. It makes no sense to build ships having only drive with mass > 1, although possible.

Drive component mass is equal to the Drive power.

In the commands to construct ships, drive is specified by

D<Power>

Examples: D1 (drive power=1), D50 (drive power=50)

Shield

The Shield component generates a protective field around the ship. A power of the protective field depends on the Shield component mass and full ship mass:

*ShieldPower = ShieldMass * ShieldEfficiency * TechShield/(FullShipMass^(1/3)) * 30^(1/3)*

Examples:

A ship having Shield = 20 and FullShipMass = 100 with TechShield = 1 and Efficiency = 1 has ShieldPower = 13.39. Adding 20 more to the shield in this example (what also increases full ship mass to 120) would give us ShieldPower = 25.2

ShieldPower here is a value decreased from a hit power in an event, when a ship has been hit by enemy fire. A result DamagePower of a hit is calculated so:

DamagePower = HitPower - ShieldPower

It makes any that HitPower <= ShieldPower doesn't penetrate the shield.

The remaining value (DamagePower) makes some destruction. See details in Battles.

Shield power is a float value >= 1.0. It makes no sense to build ships having only Shield or Shield and Drive (although possible). It would be either a static or moving target for enemy ships, not really doing anything useful.

Shield component mass is equal to the Shield power.

In the commands to construct ships, shield is specified by

S<Power>

Examples: S1 (shield power=1), S50 (shield power=50)

Lasers

Ship can have several laser cannons. Each cannon is specified by Power and a number of Barrels from 1 up to 20.

Laser power as well as a number of barrels are specified by a value >= 1. The power can be a float value (like 1.56), the number of barrels must always be an integer. In the commands to construct ships, lasers are specified by code

L<Power>x<Number of barrels>.

Examples: L1x1 (1 barrel with power 1), L10x5 (5 barrels 10 power each), L200.55x7 (7 barrels 200.55 power each). You can't have laser component, described as L5x8.6 (8.6 barrels 5 power each – incorrect: number of barrels must be integer).

The total mass of the laser cannon is calculated by this formula:

*LaserComponentMass = Power * (1 + (NumberOfBarrels – 1)/2)*

The lasers components in the example above have following masses:

L1x1 → 1 * (1 + (1-1)/2) = 1

L10x5 →10 * (1 + (5-1)/2) = 30

L200x7 → 200 * (1 + (7-1)/2) = 800

The more barrels a laser component has, the 'cheaper' each barrel is estimated by contributed mass. It's definitely better to have one laser with 10 barrels than 10 separate lasers with one barrel each. It would give you the same fire power, but much more 'heavier' ship. Obviously, the bigger ships are more expensive to build and keep.

From another hand, whole component may malfunction (skip firing loop), if it is damaged (see Battles for more details).

The laser cannons have limited fire range, depending on power.

*MaxRange = sqrt(Power) * 40;*

That is, laser with power 1 has maximum range 40, with power 10: 126.49

The range is measured in 'battle field distance units' (see Battles for details).

Also, the hit power of a laser canon depends on a distance to the target. The hit power is 0 if a distance to the target >= MaxRange, otherwise it's calculated by formula:

*HitPower = Power * LaserEfficiency * TechWeapon* (1 – Distance/MaxRange)*

Examples: (let's assume the component has efficiency 100%, i.e. 1 and TechWeapon = 1)

Laser with power 1 has MaxRange = 40. The HitPower on distance 0 is 1; on distance 20 it is 0.5; on distance 40 – 0 accordingly.

During the battle (see Battles for details) the combat ships try to move closer to the enemy ships to use full power of their laser cannons, or at least to enter the firing range.

Lasers also may miss the target. A hit probability depends on the laser power, a distance to the target, its size and a combat factor:

*HitProbability = CF * TargetDiameter * Power^(1/3)/Distance*

Where:

- CF – a combat factor (see Combat Factor): the higher CF (more experienced ship), the higher is a probability to hit the target
- TargetDiameter – is literally a calculated target diameter

  *TargetDiameter = 2 * ((Mass * 3)/(4.0 * pi * 0.5)^1/3)/TargetCF*

  where:

  pi = 3.14159265,

  TargetCF – combat factor of the target – this simulates ability of more experienced target ships to avoid hitting (they perform better manoeuvres and so on).

Basically, all this mathematics is presented here just for note – don't plunge too much in it.

Increasing the HitProbability is another reason why combat ships try to get closer to the enemy ships.

Missiles

Ship can have several missile launcher components. A missile launcher component is defined by:

- a **number of launch rails** – specifies how many missiles can be launched in one battle cycle: from 1 to 20
- a **number of launches** it can do during the battle. Let's say we have 3 launch rails and only one launch. It means during the battle such a missile component will launch only 3 missiles in one battle cycle, and then stop working. If it had, say 5 launches, such a component will do 5 battle cycles totally firing 15 missiles, and then stop working. The bigger this number, the longer 'lasts' firing ability of the missile component. This comes for a price though – the bigger number of launches is, the bigger the mass of the component.
- **power** – specifies a power of the missile the same way as for lasers
- **war head percentage** (WHP): specifies what part of the missile is used for 'the war head' and the rest – for missile drive. Default value is 0.3 (30%). The HitPower of the missile entirely depends on the WHP:

  $$HitPower = Power * TechWeapon * (1 + WHP * 9)$$

  So, the war-head makes 10 times more destruction, then the rest of the missile body.

  The bigger a war-head is, the smaller is the drive. The fastest missile has no war-head at all – just a drive-body. It's HitPower is equal to the missile Power. A missile speed on the battlefield is calculated by the same formula as for ships, but with factor 40 (not 20):

  $$Speed = 40 * MissileEfficiency * MissileDriveMass * TechDrive/FullMissileMass$$

  Missile can be damaged when moving to the target, so its efficiency can be reduced, reducing the speed. Missile drive mass is the rest of the full mass without the war-head.

  Examples (for TechDrive = TechWeapon = 1, and MissileEfficiency = 1):

Power	WHP	HitPower	Speed
1	0	1	40
1	0.3	3.7	28
1	0.5	5.5	20

In difference to the lasers, missiles always hit the target sooner or later, unless shot down. So, the HitProbability is 100%. Yes, because it moves to the target, it can be shot down by the enemy fire. Also, the missiles hit the target always with full power. There is no decreasing of power depending on distance, like for lasers. All this makes missile quite powerful weapon: even having a low TechWeapon it’s possible to create missiles capable to penetrate nearly any enemy ship with a high TechShield.

When a missile just launched it cannot hit a target in this battle cycle. This is to give an opponent a chance to intercept it. It just covers ½ distance to the target, if the target is in immediate reach of the missile. An exception from this rule is another missile.

In the construct ship commands, missiles are specified by code

M<Power>x<Number of rails>/<Number Of Launches>/0/<War Head %>.

All parameters here must be integer values, except the War-Head (it must be from 0 to 1) and the Power (>= 1). A war-head with value 1 (100%) makes no sense – such a missile will never fly.

Note '**0**' between the Number Of Launches and the War Head – this value is left for compatibility with legacy old data. It's just a place-holder for a not used value.

Examples:

M10x5/3/0/0.3 – a missile component of power 10 with 5 launching rails; number of launches is 3; war-head 30%.

M1x1/1/0/0.5 – a missile component of power 1 with 1 launching rail; number of launches is 1; war-head 50%.

The total mass of the missile component is calculated by this formula:

*MissileComponentMass = (Power * (1 + (NumberOfRails – 1)/2))*(1 + (LaunchCount - 1)/2);*

The first part of the formula looks the same as for lasers, but for missiles there is additional factor – a number of launches. Additional mass is calculated the same way as a number of rails. WHP doesn't affect the mass.

Examples:

M1x1/1/0/0.3 → 1 * (1 + (1-1)/2) * (1 + (1-1)/2) = 1 (missile power 1 with 1 rail and 1 launch)

M1x1/2/0/0.3 → 1 * (1 + (1-1)/2) * (1 + (2-1)/2) = 1.5 (missile power 1 with 1 rail and 2 launches)

M1x1/3/0/0.3 → 1 * (1 + (1-1)/2) * (1 + (3-1)/2) = 2 (missile power 1 with 1 rail and 3 launches)

M1x1/5/0/0.3 → 1 * (1 + (1-1)/2) * (1 + (5-1)/2) = 3 (missile power 1 with 1 rail and 5 launches)

So, there is an additional cost for additional launches.

Target types

Lasers and missiles may have additional option – a sort of targets they are supposed to be used against:

- Anti-ship (@as)
- Anti-missile (@am)

The default target type for both is 'anti-ship'.

During the battle when a laser/missile component is going to fire, it is looking for an appropriate target in the range. For @as weapons only ships are selected, for @am, accordingly – only missiles. It does mean that the component fires only the targets of its kind. If there is no appropriate target in the range – the weapon does not fire at all. The anti-missiles lasers and missiles are designed to destroy the fast moving small targets – missiles. They do not do any harm to ships.

It makes sense, when constructing a big ship with loads of heavy weaponry, adding some small lasers or missiles targeting the enemy missiles.

Examples of specifying the target types:

L10x5@as – laser with 5 barrels 10 power each, anti-ship

L10x5 – laser with 5 barrels 10 power each, anti-ship (default)

L1x10@am – laser with 10 barrels 1 power each, anti-missile

M10x5@as/3/0/0.3 – a missile component of power 10 with 5 launching rails; number of launches is 3; war-head 30%; anti-ship

M10x5/3/0/0.3 – a missile component of power 10 with 5 launching rails; number of launches is 3; war-head 30%; anti-ship (default)

M1x7@am/3/0/0.3 – a missile component of power 1 with 7 launching rails; number of launches is 3; war-head 30%; anti-missile

Cargo

Cargo component is a cargo bay of the ship.

Amount of cargo which can be loaded into the bay is calculated by formula:

*AmountOfCargo = CargoEfficiency * TechCargo * (Size + (Size^2)/20)*

Examples: (assuming that CargoEfficiency = 1, and TechCargo = 1)

Cargo Bay size	Amount of cargo
1	1.05
2	2.2
3	3.45
5	6.25
10	15
20	40

…

Amount of cargo is multiplied by CargoTech, what makes developing this technology quite important for speeding up the cargo delivery and reducing a number or required transport ships.

Cargo bay can be loaded by following cargo types:

- **IND**: industry
- **COL**: colonists
- **POP**: population (see `load/unload` command description for differences between COL and POP)
- **MAT**: materials

It is possible to load different cargoes to the same ship, and unload either everything or any amount of any cargo.

Cargo bay mass is a float value >= 1.0.

Please keep in mind, that amount of cargo loaded in the cargo bay is added to the FullShipMass when calculating a ship speed. The ship speed can be reduced considerably by large amount of cargo.

In the commands to construct ships, cargo bay is specified by

C<Size>

Examples: C1 (cargo bay size=1), C50.77 (cargo bay size=50.77)

Repair Unit

This component can repair any damages the ship has got during a battle. It also repairs damages of other ships which are on the same planet orbit. See Ship Repairs for details.

Amount of damaged mass the component can 'fix' during a turn is calculated by formula:

*RepairedMass = RepairComponent_TechDrive * Efficiency * Mass/RepairedComponent_Tech*

The TechDrive is the one used to increase a repair ability of the Repair Unit – another good reason to develop this technology.

One can see that 'repaired mass' depends on the tech of the repair unit itself and a tech of the repaired component. It makes sense: if repaired component is more advanced, it is more difficult to repair it with less advanced repair unit.

Considering Efficiency = 1, if tech levels of both are the same, the repaired mass will be equal to the repair unit mass. If RepairUnit has tech level 2 and the repaired component has level 4, then repaired mass = Mass/2, and so on.

Having such a component, you have an advantage to repair your ships everywhere in the Universe – on any orbit, not only in your colony. Also, no PUs (and money accordingly) are spent for repair – you get it for 'free'. The only cost is you need to have such ships equipped by the Repair units. It makes sense to have the purpose built repair ships in your fleets.

In the commands to construct ships, the repair unit is specified by

R<Size>

Examples: R1 (repair unit size=1), R52.45 (repair unit size=52.45)

Colony Unit

Add this component to make a colony ship. Just a presence of the colony unit in a ship make this ship a “colony ship”. The colony ships are the primary way to colonize uninhabited star systems.

A size of this component is fixed and equal 100.

In the commands to construct ships, the colony unit is specified by

K

It’s possible to specify a size, but it’s be ignored.

Examples:

K100 = K

K45 = K (silently uses mass 100)

K0 – wrong, error will be displayed

Tow Unit

A ship with tow unit can tow another ship. A maximum mass of the towed ship depends on the component’s mass and can be calculated by the following formula:

*MaxTowedMass = TowUnit_TechCargo * Efficiency * Mass * 10*

The component uses TechCargo as its technology level.

Examples:

Tow Unit mass=1, Efficiency=1, TechCargo=1: max towed mass will be 10

Tow Unit mass=5, Efficiency=1, TechCargo=3: max towed mass will be 150

In the commands to construct ships, the tow unit is specified by

T

Examples:

Examples: T1 (tow unit size=1), T52.45 (tow unit size=52.45)

Advices about ship design

Having 7 different ship components you have a wide flexibility to build different ships. Here are just a few advices you may follow or just ignore:

- Don't build useless ships. Examples of such:
    - Ships with only drive and mass > 1: you will not gain any additional speed, but only spend more resources to build and keep
    - Ships with only drive and shield
    - Ships with cargo bay without drive: such ships will not move, what the cargo bay for?
- Don't' build too slow ships. Make sure the result speed of the ship is enough to reach at least your neighbour star systems just several LY. The server will refuse to move a ship to a destination if the travelling time is > 10LY. This is a sort of protection to avoid some ships travelling endlessly in the space (imagine a ship with speed 0.01 travelling to a star 5LY away – it would take 5/0.01 = 500 years. Such slow ship may appear after battles, if a drive component has been heavily damaged, but not entirely destroyed, or, because of a stupid design)

  Normally, a drive should have about 40-50% of a ship mass, i.e. nearly a half, unless you have a well developed TechDrive.
- Prefer a smaller number of bigger ships to a great number of smaller ships. Big ships can have more stuff: more weapons, more powerful shield and so on. Because of that they have more probability to survive battles and gain experience (see Combat Factor). They are also cheaper to keep (see Ships Payments). Because ship weapons are 'smart' (see Battles), there is no advantage in having a huge number of small ships in a fleet during a space battle. The weapon components (lasers and missiles) try to find the 'most appropriate' target amongst those in the range. Ships can detect sizes of the enemy ships and do the most destructive fire (In traditional Galaxy games a decade ago, the targets were selected randomly what made sense having clouds of small ships 'drones', playing a role of false targets. It is not the case in Galaxy Kings).

  A number of small purpose built ships is always necessary. These are scouts, small/middle transports, small/middle combat ships for doing small tasks. But your main attack combat fleet is better of having big powerful line ships.
- During the years of the game flow, an alternative approach to build the combat fleets has been developed and proved its usability. Instead of creating huge individual ships, a large number of relatively cheap missile ships are formed into fleets. These ships should be capable to penetrate a shield of any potential enemy ship. The swarm of such ships mostly wins a battle against an equivalent by mass (or even much larger) individual enemy ships.
- Some ship types you might want to have:
    - 'Scout': **D1** – the fastest and cheapest possible useful ships to explore unknown stars and keep an eye on your neighbours
    - 'Scout Killer': **D1,L1x1** – a scout armed by one laser. It is also fast, but can shot down enemies scouts. But be careful when using these 'killers' everywhere instead of scouts – they may upset your neighbours

- 'Transport Ship': **D10,C10** – a small not armed transport ship.
- 'Root Master': **D25,C20,L3x3@as** – a bigger transport ship with lasers
- 'Defender': **S100,L40x3,L10x3@am** – an orbital defence station (mind and absence of drive) with strong shield and powerful lasers (40) with 10 anti-missile lasers. Note, ships without drive are perfect for protection of your colonies. Instead of a drive you can build more shield and heavy weaponry.
- 'Line Ship': **D250,S80,L60x3,M10x5@as/5/0/0.3** – a well armed line ship

Service and Combat ships

Note, all the ship type belong to one of two categories:

- Combat ships: basically, these are ships designed for combat. By default, if not specified, any ship with weapons is set up as a combat ship.
- Service ships: these ships have other purposes than combat. Example: scout ships, transport ships, tow ships, repair ships.

The main difference between the combat and service ships is in their behaviour during the battle. Combat ships always try to advance and attack the enemy using all their weapons. The service ships, in opposite, are trying not to be involved and stay aside. The armed service ships have weapons mostly for 'self-protection', not to engage.

It's a good idea to explicitly specify a ship type for service ships during design, if such a ship has any weapons. If you forget to do that, your, say, armed transport ship with a few weak lasers, will try to attack enemy combat ships, instead of staying away and may be retreating...

Ships Production

You can place an order to build a ship of any ship types available in your list. An order can be placed with specifying one of your colonies or without it, i.e. 'common' orders can be used. Necessary costs to produce a ship depend on it's mass. To produce one mass unit required 10 PUs and one unit of materials. This makes ships the most expensive type of production.

Build time depends on a ship mass and can be long, if your nation is short of money or industry. Big ships is better to build on big well developed colonies.

In ship production orders you can specify an assembly point – a star, where the ready ships will be sent automatically. It is very convenient when you need to gather a fleet in certain location.

Instead of assembly location you can specify a fleet which ships will join automatically as soon as they are ready. The ships will be sent to fleet location, even if the fleet moves. In this case the new built ships will follow the fleet trying to join it. This is a sort of automatic reinforcement of fleets, acting somewhere. Such automation is extremely useful when your nation has many colonies and it's becoming difficult to maintain new-built ships assignments manually.

Ships Costs

After any ship has been built, it becomes an active member of your National Fleet. Some cost is being taken from your National Treasury account to maintain the National Fleet. It includes salaries to the crews and service personnel, fuel, ammo, management and so on. The cost amount is calculated by the following formula:

FleetPayment = CalcMassCosts() + AdminCosts

Where *AdminCosts* equals to the total number of ships you have.

CalcMassCosts() is a formula, calculating the costs depending on summarized mass of all ships. The algorithm is progressive: the more ships you have, the more expensive to keep them:

Number of Ships	Payment Factor
1..10	0
11..100	1
101..200	2
201..500	3
501..1000	4
1001..2500	5
2501..5000	6
5001..10000	7
10001..20000	8
>20000	9

Examples:

Let's say you have 10 ships, all have the same mass 100 (to simplify calculation). The fleet payment would be

10 + 10*100/(10 - 0) = 110

If you have 15 ships, the cost would be:

10 + 10*100/(10 - 0) + 5 + 5*100/(10 - 1) = 110 + 60.55 = 170.55

If you have 250 ships:

10 + 10*100/(10 – 0) + 90 + 90*100/(10 - 1) + 100 + 100*100/(10 - 2) + 50 + 50*100/(10 – 3) =

110 + 1090 + 1350 + 764.29 = 3314.28

… and so on

Note, that AdminCosts (shown as **green**) doesn't depend on a ship mass – it's same 1 Credit for a 'Scout' with mass 1 and, say, a 'Death Star' with mass 50000.

The fleet payments automatically deduced from your Treasury account in the beginning of each

turn.

What happens if you have not enough money to pay for the fleet? Such situation may happen if you have a big fleet, but lost most of your population and industry as result of a war. In this case the money you owe are recorded and accumulated. All your remaining production will work to pay the debt until it's paid off. The fleet will remain “in tact”, however there are plans to change this behaviour soon, simulating the “fleet unhappiness” by the lack of financing (ships may refuse to follow the orders, fight or even betray you).

Ships Repair

Ships can be damaged during a battle. A level of damages can be judged by Ship Efficiency. If Efficiency is less than 1 – the ship needs repair – one or more of its components have been damaged or destroyed.

Ships can be repaired in two ways:

- By Repair Unit component in the ship itself or in any ship located on the same planet orbit
- By directly placed Production Order to repair the ship, if it is located on one of your colonies.

Repair Unit component doesn't spend any PUs and money to repair the ship and it repairs automatically without your involvement. The production order for repair, however, has to be placed explicitly. A cost of repair of 1 damaged mass unit is the same as a production of this unit, i.e. 10PU. No materials required, so ships can be repaired without additional production costs for materials.

Ships Upgrade

A newly built ship has technologies for all its component for the turn when it's been built. After a while, when your technologies has been developed, some old ships become 'outdated'. You can modernize the ships brining them to the level of your latest technologies. You may place a PO for ship upgrade and specify which technology is to be upgraded.

Upgrade basically means 'replacing' a component, associated with the specified technology by the new one with current level of technology. Technically, the old component is destroyed, and new component is built on its place. A level of technology for a upgraded component is the one, when this replacement has started. Upgrade may take more than one turn, depending on a size of the component and available PUs and money (like in normal production). For example, let's assume that upgrade of ship's Drive has started on turn N with current DriveTech 3.67 and then lasted 5 turns. On turn N+5 the DriveTech already became 4.12 (you might have a long lasting order to developed DriveTech). Nevertheless, your upgraded Drive will have only 3.67 when upgrade has finished.

Upgrade can happen only on one of your colonies. Associated costs are the same as during repair: 10PUs for each upgraded mass unit. Materials are not required as well.

Ships Combat Factor

In the Galaxy Kings all the ships are treated individually. Each ship has:

- Own efficiency
- May have individual name
- Own automatically calculated Combat Factor.

The Combat Factor (CF) is accumulation of the ship 'experience' during its live time. The CF is used during the battles and mostly gained during the battles. Ships with higher CF fight better – they are more difficult to hit, they fire more precisely, they are more reliable (damaged component fail with less probability).

After being built a ship has CF = 1. Then, the CF grows each turn the following way:

- 0.01 added for each 'year of service'. It is considered, the ship's crew is getting more and more experienced. 0.01 is added each turn to CF even if the ship does nothing and just staying on some orbit for years.
- 0.0001 added for each travelled LY. Acting ships gain more experience. For example if a ship has moved on 20LY during a turn, its CF will be increased on 0.01 + 20*0.0001 = 0.012.
- 0.01 for participating any battle even passively. 'Passively' means the ship has just witnessed the battle, not even firing a shot.
- 0.005 for each damaged enemy ship. This is added if any enemy ship damage has happened as a result of the ship firing.
- 0.01 + 0.1*(MassOfDestroyedShip/*MassOfShip*) for each destroyed ship. This is added to the ship's CF if a result of its shot has finally destroyed an enemy ship. This can be sometimes considered unfair to other ships, which have contributed their efforts to damage this ship, but the final shot is very important – the enemy ship blows up boosting crew moral and CF. Also, one can see from the formula, that the bigger the mass of the destroyed ship in comparison to the shooting ship, the more CF grows. It’s obvious: a small ship, destroyed a big ship has a bigger moral boost, than a big ship destroyed a small one!
- 0.005 if a ship has been damaged during the battle.

Tests have shown, that the same group of ships with higher CF has more probability to win, than a group with lower CF. The higher the difference, the higher the probability to win.

The CF is another point why having big ships is an advantage. Statistically, the bigger ships have longer lives, so the may gain higher CF and can be even more effective in battles.

All ships statistics (used in calculating the CF) is available in the reports.

Some examples of ships statistics during simulation run of the game (412 turns):

Type	Age	Travelled (LY)	Been damaged	Destroyed ships	Destroyed mass	CF
AttackFregate (D25,S10,L10x2)	203	199	1	268	4421	15.78

MiddleDefender (S100,L40x3,L10x3)	403	0 (orbital station)	2	80	1951	8.46
AttackLineShip (D250,L60x3,L10x5,R20)	253	344.44	2	152	4835	8.17
CargoCarrier (D10,C6,L2x3)	143	1642	0	0	0	2.59

One can see that some ships have very long living and experience

Scrapping Ships

You can scrap unwanted ships only on your colonies. The mass of scrapped ships is added to materials of the planet, all cargo on board (if there is such) is unloaded.

Note: after scrapping, the ship is removed from your National Fleet with all its history and statistics. Think twice before scrapping ships with big CF.

Movement

Most of the spaceships are equipped with hyperspace drives. The drive power is equal to Drive value multiplied by the Drive tech level at which the ship was built. Ships with a Drive of zero remain forever in the system where they were built.

As has been mentioned above, a ship moves a number of light years per turn calculated by the following formula:

$$Speed = 20 * DriveEfficiency * DriveMass * TechDrive/FullShipMass$$

Remember, that unless your Drive tech level is very high, large ships should have correspondingly large drives or they will be very slow..

The mathematics of hyperspace travel dictates that ships can only travel from one concentration of mass to another, i.e. from one system to another; you can't send a ship out into empty space. A movement in hyperspace between two points called ‘jump’. Also, once a ship has started a jump to another system it cannot change course, turn back or even slow down. So for example, if you launch a massive attack on an enemy system and then discover that the system is heavily defended – its too late: your fleet is inevitably doomed to fight, and most probably sustain heavy losses or be wiped out completely.

The maximum allowed time for any ship to spend in space is 10 years. If calculation shows a ship is about to be in space more than that, the server automatically calculates a number of jumps to intermediate star systems.

You can send ships only to visible stars and only those, which have distance <= 50*TechDrive LY from any of your colonies.

Fleets

Fleet is a named group of ships located in one place. Fleet can have ships of different types. To create a fleet you need to choose a distinctive name and order some ships to join the fleet. A fleet is automatically created in a location where the 1st ship has joined. After that you may order other ships to join the same fleet even if these ships are located in other star systems. These ships will start moving to the fleet location and join the fleet sooner or later.

A speed of a fleet is the speed of the slowest ship in the fleet. It is impossible for a ship with zero speed to join a fleet. If a ship has it's drive badly damaged in a battle, so its speed reduced to value < 1 (minimum speed), it's automatically excluded from a fleet (if it belongs to any fleet). This is a protection for the fleets to be 'unacceptably' slow. You may manually order the ship to join the fleet back, if its speed is still > 0, but probably it's not what you want. After being repaired (when its speed becomes >= 1), a ship excluded from fleet will automatically re-join it's fleet again.

There are a number of so called System Fleets, controlled by the Automated Managers (see below). There are 4 System Fleets in the Galaxy Kings:

- SCOUT_FLEET – used by the Exploration Manager. Ships belonging to this fleet are used by the server to explore new stars and keep all your discovered stars with at least one ship on the orbit, to have up-to-date information about what's going on there.
- TRANSPORT_FLEET – used by the Transport Manager responsible for automated delivering different cargo from one point to another
- DEFENCE_FLEET – used by the Defence Manager responsible for building orbital battle stations on your colonies to protect them against enemy ships
- ATTACK_FLEET – used by the Attack Manager, responsible for destroying any detected enemy colonies

The System Fleets do not have any particular location. You cannot send them anywhere. The only thing you can do is to order some ships to join these fleets to perform automated operations, and, order some ships to leave these fleets, if you need them somewhere else. Ships, assigned to the System Fleets are controlled by the server on your behalf.

Transporting Cargo

Any ship with Cargo component can transport the following cargo from one star system to another:

- Industry
- Colonists (population)
- Materials

Loading cargo

Colonists can be loaded only from your colonies, Industry and Materials can be loaded from your colonies and from any uninhabited planets – just a subject of availability.

Unloading cargo

Colonists can be unloaded:

- on your colonies
- on enemies colonies. In this case the colonists are considered as an invasion force. The landing causes a 'ground battle' (see Ground Battles). If your colonists win, the planet automatically becomes your new colony. Otherwise, your landed party gets destroyed.

It is impossible to unload colonists on the colonies of your Friends (see War And Peace)

Materials and Industry can be unloaded on your colonies only.

Buying

If you have enough money, instead of producing something, you can buy something instantly from 'dark market dealers'. It comes at significant cost, but you get it immediately.

You can buy ships, technologies, industry, materials, and clones.

To buy something you place a 'buying order' specifying a colony (where to buy), what to buy and how much. A buying price is calculated as amount of money would be spent for production of the ordered goods multiplied by 10. Because production can cost differently depending on a specific colony (mind resource factor), it is important to buy on colonies with rich with natural resources. Otherwise it can be very expensive. The 'dark dealers' take their price for smuggling on poor planets!

Example: You'd like to buy a ship of mass 100 on a colony with resource factor 0.5. Production of such a ship, would take 100*10 PUs + producing materials 100/0.5 PUs = 1000 + 200 = 1200PUs. Because each PU needs 1 CREG (money unit), total cost of production is 1200 CREG. A price of the 'dark dealers' is 10 times higher, i.e. 12000 CREG. If resource factor of this colony were 0.1, the final price would be (1000 + 100/0.1)*10 = 20000 CREG – mark the difference!

You need to buy whole ordered amount of goods at once. The buy orders cannot be spread over several turns like production orders. If you have not enough money in your treasury, the buy order will be rejected.

The goods will appear on specified by buying orders colonies on the next turn.

Buying is a waste of money. But sometimes, if you have money you may consider it. Typical example – preparation to a big war against someone – you may get a big fleet instantly and throw it in action, not waiting till it's been produced.

Buying Colonies

Instead of colonizing a star system you can buy it from the same 'dark market dealers', if you can afford it. A colony price is equal to *ColonySize * 10*, i.e. star system with size 1000 would cost 10000 CREG. You can by if all of the following conditions are true:

- you have enough money
- the star system is uninhabited
- you have one of your ships on the planet orbit

A bought star system has 1 population unit added automatically.

War and Peace

At the start of the game you are assumed to be at war with all the other nations. You can change this status if you know a nation's name you'd like to be in peace.

Information about other nations in the Universe can be found from several sources:

- from a list of your 'Known Nations'. An alien nation is recorded in this list if:
  - your ship meets a ship of this nation in any orbit
  - your ship visits a colony of alien nation
  - alien ship visits your colony

  An alien nation is removed from this list automatically only if it has been completely destroyed.
- From a rating list in your reports (see Rating for details). This list contains all nations which have colonies in any of your visible stars. You may have not even met some of these nations, but they will be in the rating list
- From the web page (see List Of All Nations http://galaxy.magix.net/public/AllNations.html). Currently all existing nations are listed there not depending on their wish, but this may change in the future.

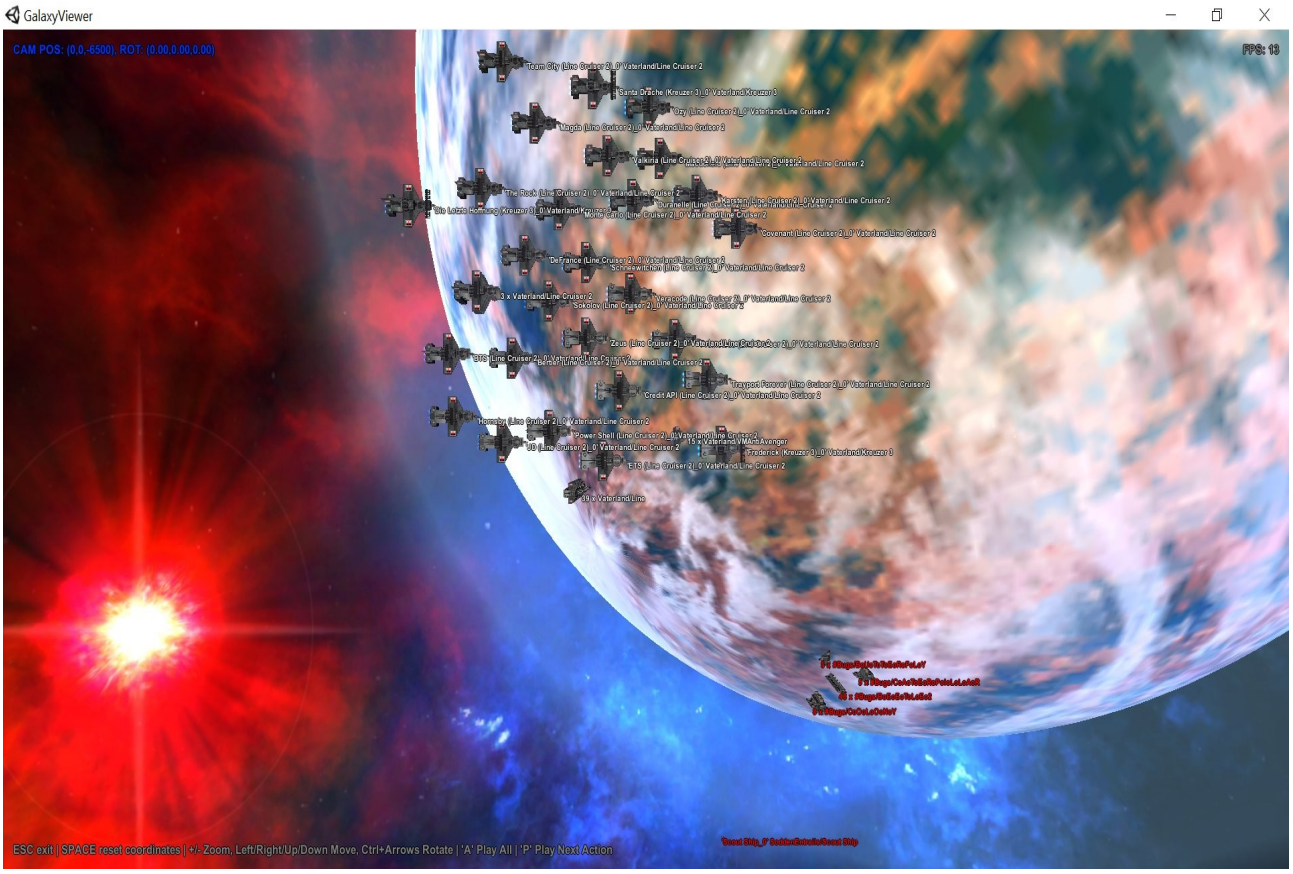
Your state of War or Peace to another nation doesn't automatically mean that this nation has the same relationship to you. If you'd like to make a peace with someone, it's a good idea to use Diplomatic Mail to negotiate your intention with other side. You will never know the real attitude to you from another nation until combat ships of your nations meat each other or appear on the colonies orbits. If both have state peace nothing will happen, otherwise, the ship start fire at each other or bomb the colonies.

There is no possibility to set a default peace to all nations.

Space Battles

Ships in hyperspace cannot be attacked, but whenever hostile warships are present at the same star system a battle will take place.

The battle is happening on a virtual 2-dimensional area, measured by 'distance units'. We may think, that one unit is about 500 yards. All the ships belonging to the same nation are grouped together, creating virtual 'nation fleets'. Also, ships of the same type are placed at the same coordinates,



creating virtual 'groups of ships' (This has been done for practical reasons – making battle viewing faster and simpler. Original version where every ship was shown separately was a disaster when a number of ships was too big (thousands and more...)).

There is no limitation on a number of participants in the space battles.

The minimum distance between ships groups inside a nation fleet is 8 distance units. The minimum distance between the nation fleets is 15 units, and the maximum distance is 40 units. All the groups inside fleets and fleets inside the virtual battle area are placed randomly, just obeying these min/max rules.

After the fleet formations for each battle participant have been created, the battle begins.

A battle consists of battle cycles. Each cycle has the following steps:

- Collection of all targets for each nation fleet. A target – is an enemy ship or an enemy missile. If there are no targets found (all shot down or all sides are in peace), the battle ends.
- Collection of all weapons each ship has which. If there are no weapons left (ships are not armed, or weapon components have been destroyed), the battle ends.

- Firing loop stars:
    - Randomly selected a not fired weapon. The algorithm is trying to find the most suitable target in the range of this weapon. If target found – the weapon fires a single shot (it can be a laser shot or a missile launch), if a total number of shots during the firing loop has not exceeded a number of gun barrels. If exceeded, the weapon is marked as 'fired', otherwise, a number of fired shots is incremented for this weapon. If no targets has been found for this weapon, it is marked as 'fired'. If a missile is launched and it reaches the target straight away because it is close (this depends on distance and missile speed), it moves just a half way to make possible the anti-missile weapons of the opposite sides to fire.
    - All this continues until no more not fired weapons left.
- Moving launched missiles. Every missile launched in previous firing loops is moved towards their targets. If a target is in reach – it hits damaging or destroying components or ships.

Note: all the hits are checked against a ship's shield. Depending on that the hits may do more or less damage

- Moving all the ships groups. All the combat ships, if they are not heavily damaged, advance against selected enemy group of ships. The target group is selected as the most appropriate target of the combat ship group's primary weapon. The primary weapon is the most powerful laser. Missiles are not counted here, because they don't have range – missiles can be conveniently launched from a safe distance, without proximity to the enemy.

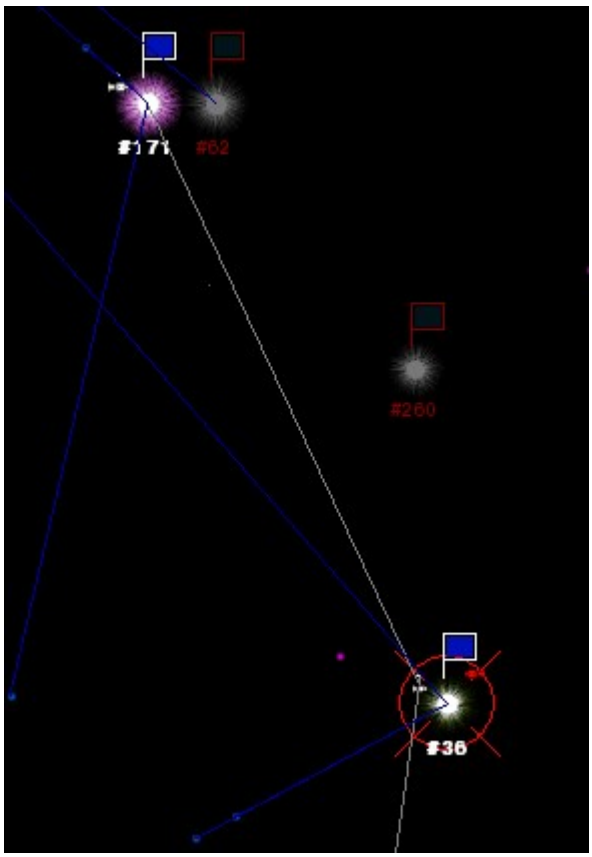
The service ships do not advance in the direction of enemy: they remain on the same place.

If the nation fleet is retreating (see below), all the ships capable to move fly in the direction away from the enemy ships.

- Checking possible retreat. Each battle participant fleet has internal 'retreat counter'. If during the check it is discovered, that overall damages of the fleet during current battle cycle are 4 times (or more) bigger than enemy damages inflicted by the fleet's fire, the retreat counter is incremented. This is a sign, that enemy is has a great superiority in numbers or fire power or shield. If retreat counter for a fleet is > 0, the fleet will be retreating during the next battle cycle. From other hand, if the check shows retreat condition is not fulfilled, but the retreat counter is > 0, the retreat counter is reset to 0, and during the next battle cycle the fleet will advance again.

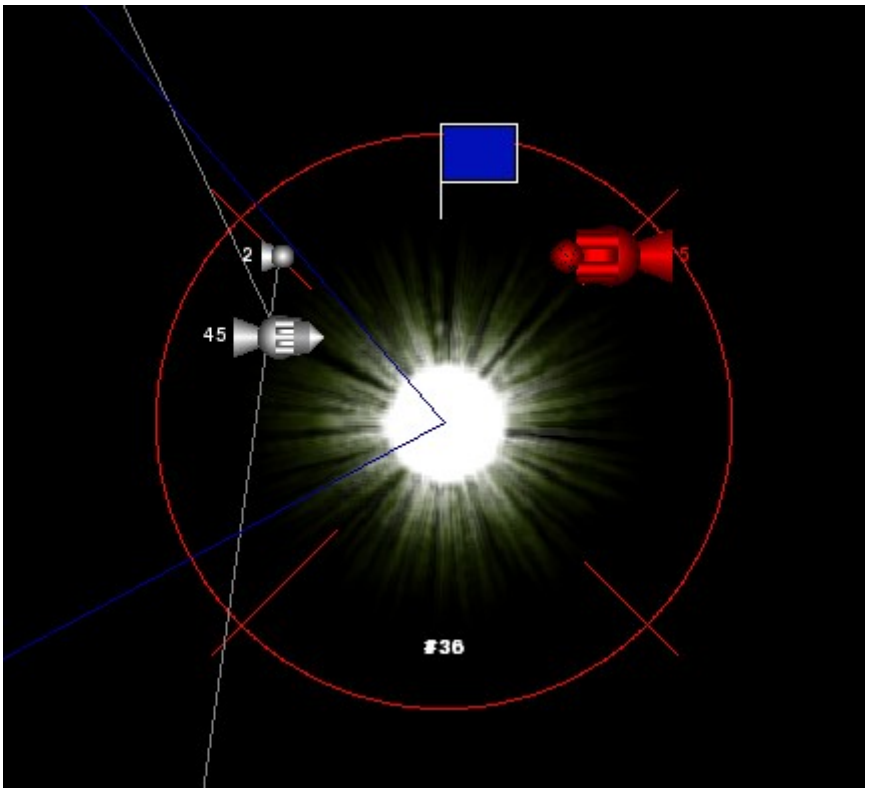
If the retreat counter has reached 3, the 'general retreat' command is issued for the fleet, and it leaves the battle making a hyperspace jump to the closest star. All the ships which cannot make a jump (drive component is badly damaged or destroyed, making speed very slow) are left behind, so they continue the battle with most possible result – complete destruction.

The escaping ships do not move until next turn. They are just 'launched'. The destination of retreat can be seen in the Report Viewer:



Here a group is escaping from the battle happened on star system #36 to the star system #171 (see white line).

A bit closer look shows the group of 45 transport ships is escaping:



After that starts a new battle cycle.

A battle ends if:

- There are no more targets found
- There are no more firing weapons available
- If during more than 100 battle cycles no damages have been made by either side (possible reason: all sides have very weak weapon and very strong shield – deadend)
- 3000th battle cycle ended. i.e. a battle can go not more than 3000 cycles. This may happen very rare, for example, because the sides inflict each other miserable, but persisting damages over the cycles. Such a battle may go on very long. This is a defence against potential long processing.

Damages

If a ship is hit by laser shot or a missile then:

- Calculated amount of damages:

  $$Damages = HitPower - Shield$$

- Obviously, if Damages <= 0 there are no damages
- If Damages >= ShipMass, the ship is destroyed
- If Damages < ShipMass, the ship is *damaged*. But what component/components? The components to be 'damaged' are selected randomly, depending on their size, so a bigger component has more probability to be hit first. So, the first component is 'damaged' by full hit power. If HitPower >= ComponentMass, the component is destroyed. If there is still some power left – a next component is selected from the 'queue'. It lasts till the HitPower becomes 0. That is, one hit can destroy/damage one or more components in the ship.

  Example:

  Let's say we have a ship D10C10 (Drive=10, Cargo=10), total mass=20. The ship has been hit by power 3.

  Let's assume the drive was hit first. Because HitPower (3) < ComponentMass (10), the component (Drive) is damaged. No more HitPower left. After the hit, drive will have efficiency 70% (remains 7 not destroyed mass), cargo - 100%.

  Another hit: HitPower = 15. Let's assume this time cargo hit first. The HitPower(15) > ComponentMass(10): component Cargo has been destroyed. There is still HitPower 5 left – it goes for drive destroying other 5 mass units, leaving in drive just 2. After the hit, the drive efficiency is 20%, cargo efficiency is 0% (destroyed). But the ship is still alive. With 10% full efficiency.

When calculating damages a cargo mass in the cargo hold is not counted. The cargo is destroyed according to percentage of damages of the cargo hold.

Example: Cargo hold has size 10, loaded by cargo COL = 4. If the cargo hold has been hit by power 3, it makes 30% damage. The cargo COL will be destroyed on 30%, i.e. after the hit we have only COL = 4 - 4*.03 = 2.8.

Bombing

If the armed ships are left at an enemy nation's system after all fighting has been done, they will bomb the system. On the system they destroy the population, colonists and industry in the same quantity of bombing power of all the ships. The bombing power is calculated the following way:

*BombingPower = ((AllLasersShots^0.5)/10 + 1) * AllLasersShots*

Where *AllLasersShots* is a sum of all the shots the ships lasers with ANTI-SHIP target type would fire. Each laser shot is calculated as LaserPower*TechWeapons*LaserComponentEfficiency (Missiles do not 'shell' the planet). Please note, that ANTI-MISSILE lasers do not shell the planet surface. They designed to hit only fast moving small targets.

Example. Let's consider a colony with 3000 Population, 800 Colonists and 2500 Industry. There are 2 enemy 'Line Ship' (D250.0,S80.0,L60.0x3@as,L10.0x5@as,M5x4@as/5/1/0.3,R20.0) on the orbit. Ship 1 has TechWeapon 1.5, all components not damaged, Ship 2 has TechWeapon 2.3, but damaged (L60x3 has efficiency 0.7, L10x5 has efficiency 0.3)

Ship 1 AllLaserShots = 60*1.5*3 + 10*1.5*5 = 345

Ship 2 AllLaserShots = 60*2.3*3*0.7 + 10*2.3*5*0.3 = 324.3

Total fleet power = 345 + 324.3 = 669.3 (Note: missiles M5x4 are not counted)

BombingPower = ((669.3^0.5)/10 + 1) * 669.3 = 2400.834

After the bombing we'll have:

Population = 3000 – 2400.834 = 599.166

Colonists = 800 - 2400.834 = 0 (totally wiped out)

Industry = 2500 - 2400.834 = 99.166

If after bombing some colonists left and population is below the colony size, then part or all the colonists converted to population.

Example: Planet Size = 1000, Population = 1000, Colonists = 400. After bombing with power 300 we'll have:

Population = 1000 – 300 = 700

Colonists = 400 – 300 = 100

Automatically 300 Population is added from colonists making final figures:

Population = 1000

Colonists = 300 – 30 = 270 (30 were converted to 300 Population)

The enemy star systems can be saved from bombing by command '*prevent bombing*'. To resume bombing use command '*allow bombing*'.

Ground Battles

Ground battles take place when one nation unloads their transport ships with colonists on the enemy planets. How it goes:

- Calculated InvationForce = Unloaded COL * 10
- Calculated ColonyDefenceForce = ActivePopulation + COL * 10
- Virtual battle happens the following way. Randomly selected a side who fires. One unit from selected side fires. Probability to hit an enemy unit depends on weapon technology of firing side and shield technology of other side:

  $$100 * n1tech\_w/(n1tech\_w + n2tech\_s)$$

  where:

  n1tech_w – weapon tech of firing side

  n2tech_s – shield tech of another side

  Then generated a number from 0 to 100. If this number is <= probability – an enemy unit is destroyed, otherwise it's considered as missed or shielded.

  The firing is repeating until one side it completely wiped out. If the Invasion Force has won, the invaded nation becomes an owner on this planet. All industry and materials (if there are such) remain as a prize.

A ship can unload COL on enemy planet only if there are no armed enemy ships on the orbit. Otherwise, a Space Battle will happen first.

Restricted Areas

If you have a peace with someone, their ships can be on your colonies orbits and gather up-to-date information about this colony and all your ships on the orbit. Sometimes you'd like to avoid sharing some important secrets even with your allies: you may design secret ship types or concentrate some forces...

It is possible to restrict access to some of your colonies even for friends to avoid them 'spying' your secrets. To do so you can use commands 'restrict access to'/'allow access to' (see below). If a ship of a nation you are in peace comes to a restricted colony (or 'is' on the orbit when you restricted the access), it'll be involved in a space battle with all your ships currently on the colony orbit.

Make sure the colonies you are restricting an access to are well protected by your combat ships. Otherwise an effect can be surprising: an ally war ship may come to your colony where your ships are just scouts or unarmed transports, and they all be shot down and the colony will be bombed!

It's also a good idea to warn your friends to keep away from the restricted star systems to avoid any 'blue-on-blue' fire.

Aristocracy

As a player you may be awarded a noble title for ruling a number of colonies.

Straight after joining you become a 'Governor' of your nation. To become a member of the 'Aristocracy Club' you need to gain the initial noble title of 'Baron' or 'Baroness'. To be awarded this title, you have to rule at least 20 colonies for not less than 5 turns in a row. See the table with all noble titles available in the Galaxy Kings:

Title	Number of Colonies	Number of years (turns) to keep the colonies
Baron/Baroness	20	5
Marquis/Marchioness	50	10
Count/Countess	75	10
Duke/Duchess	100	15
Archduke/Archduchess	150	15
King/Queen	200	20

A noble title can only by 'upgraded', not downgraded. I.e. after becoming a 'Baron/Baroness', you'll never be downgraded to 'Governor' again, not depending on how many colonies you may have later. A noble title is awarded for 'live' to a player, not to a nation. If the player leaves the game, his/her title will be lost. But the nation, in this case will be ruled by the server, and someone else can join the game with this nation at some point. The new player will not be immediately awarded by the title the previous owner had. The player will start from the 'Governor' again, but a promotion will be relatively quick, if the conditions fulfil. If, for example, a nation the player has joined with, already had 123 colonies, it would take 5 turns to become a 'Baron/Baroness', then 5 turns more to be a 'Marquis/Marchioness', then 1 turn to become a 'Count/Countess', and 4 more turns to be a 'Duke/Duchess'.

Galaxy Master and Galaxy Emperor

These are temporary titles co-existing with the noble titles.

A player can become a Galaxy Emperor of a certain galaxy if he/she rules all the star systems of this galaxy for at least 5 turns. To become a Galaxy Master you need to rule at least a half of star systems in this galaxy.

There is a money bonus for holding such high degree honourable titles. The Galaxy Emperor receives each turn in it's Treasury amount of money equal to half of entire population of the galaxy. The Galaxy Master receives less money, equal to a quarter of entire population of the galaxy.

Once obtained, these titles can be downgraded, if ownership conditions do not fulfil any more. So, a Galaxy Emperor can become a Galaxy Master if during 5 years he/she doesn't own all the star systems in the galaxy; and, correspondingly, a Galaxy Master title can be revoked, if during 5 years he/she doesn't own a half of all the star systems in the galaxy.

The same high player can theoretically be at the same time an Emperor and Master of several galaxies.

Rating

Each turn rating is calculated for each nation, so the players can estimate how well they are progressing in development and expansion. In the list of All Nations (see List Of All Nations http://galaxy.magix.net/public/AllNations.html), all the nations are printed in order of their Rating Index. Rating Index 1 corresponds to the highest Rating currently present in the list. The Rating is calculated as the Production Potential of all the colonies if of the nation divided by 1000.

One can see that the nation’s rating doesn’t depend on a number of ships, colonies and technology levels. Yes, it doesn’t. These all are just means to achieve the higher production potential. Use ships with high techs to conquer new colonies and transport population there, to protect existing colonies. If you don’t do it, you ships and techs are useless.

Information about other Nations

You can get pretty much information about your neighbours in the Universe from Rating table (in Reports or from the web page, but other players can hide some details. A player can hide any or all of following details from publishing:

- Rating
- Population (actual, added and lost)
- Industry (actual, added and lost)
- Ships number (actual, added and lost)
- Ships mass (actual, added and lost)
- Technological levels individually and avreage technology
- Colonies number (actual, added and lost)
- Colonies total size (actual, added and lost)
- Money
- Debt (how much fleet payments has not been paid)

What cannot be hidden is Rating Index and Age (or Year of Creation)

In addition to the 'public sources' each nation gathers its own intelligence data about other nations. In the turn reports it can be seen in 'Known Nations' table. This information can not necessary be up-to-date.

How it works.

- If alien ships (friendly or hostile – doesn't matter) are on the orbit of your colony by the end of a turn: a nation of the alien ship obtains a fresh data about your technological levels, entire population and industry. Your nation remembers the ship types of the alien ships
- If your ships and alien ships are on the orbit of the same planet by the end of a turn: your nation remembers the ship types of the alien ships, and alien nation remembers the ship types of your ships.
- Before any space battle begins, all nations of the ships met are informed about each other existence without any details
- If alien ships bomb your colony out (destroying all population and industry), you are still informed about the enemy ship types, participated in the bombardment, and general info about the attacker (name and flag). This is because the enemy ships have been on the orbit for a while. It’s considered enough to gather information about the enemy ships and send it out to the ‘information centre’. From their side, the enemy just obtains an information about your existence and flag.

Money Transfer

You can transfer money to another nation. Two conditions should fulfil:

- A specified amount of money should not exceed what you have in the Treasury
- A nation recipient should be 'known nation', i.e. you should have had a contact with this nation

Flags

Every nation has its own state flag – a rectangle image 130x100 pixels. When created, a flag is assigned automatically. This flag is a randomly generated by the server plain colour rectangle like this:

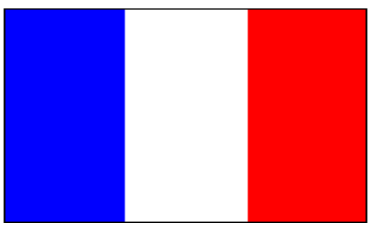
The flags are coded into the turn reports, so the report viewers should show them where appropriate. The standard Report Viewer shows the flags above each colonized star system (if information available for your visibility and intelligence data (known nations)).

Such 'plain' flags are a good indication of AI nations, although some players can deliberately use this style. Usually, human players tend to change the default flag to something more sophisticated. It is possible to design a flag using a set of templates, supported by the game server. You can't just post a random image and use it as your flag. But you can select one of the templates and set colours of your choice.

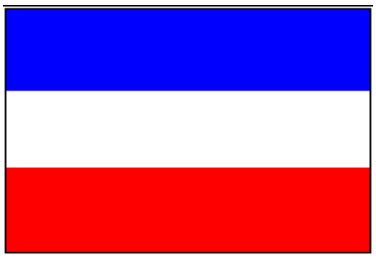
The templates are a pre-created text files with exact size 130x100 characters where each character corresponds to a colour (an RGB tuple where each colour is a value from 0 to 255, for example red =(255,0,0), green=(0,255,0), blue=(0,0,255), black=(0,0,0), white=(255,255,255), and so on). As a simple example let's have a look at the template '3x3', designed to create tri-colour style flags (like France or Holland). The template looks approximately so:

```
111111222222333333
111111222222333333
111111222222333333
111111222222333333
111111222222333333
111111222222333333
444444555555666666
444444555555666666
444444555555666666
444444555555666666
444444555555666666
444444555555666666
777777888888999999
777777888888999999
777777888888999999
777777888888999999
777777888888999999
777777888888999999
```

where each section of '1'...'9' characters is in reality 43x33 (with small corrections to fit exactly 130x100). If, for example, you' wanted to have a French flag, you'd assigned characters '1','4','7' to Red colour, '2','5','8' to White colour, and '3','6','9' to Blue colour. As result you'd have:



Changing the colour as '1','2','3' to Blue, '4','5','6' to White and '7','8','9' to Red will give you:



Currently supported templates are:

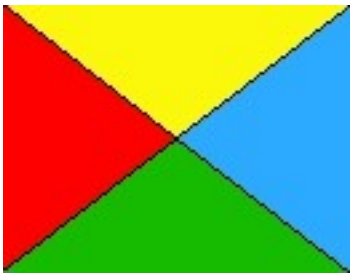
Template Name	**Picture or Schema**	**Comments**

3x3	`1 2 3`
	`4 5 6`
	`7 8 9`
		It's the simplest template available. Perfect for creating tri-colors vertical or horizontal. All sections are equal, approx 43x33

4x4	`1 2 3 4`
	`5 6 7 8`
	`9 0 A B`
	`C D E F`
		Ideal for two-colors. All sections are equal, approx 32x25

5x5	`0 1 2 3 4`
	`5 6 7 8 9`
	`a b c d e`
	`f g h i j`
	`m n o p q`
		Extended version of 3x3. All sections are equal, approx 26x20

1800		Correspondence of colours to section character: Red: Z Yellow: A Blue: X Green: O White: . (dot) Could be a flag used by French army at around 1800 with Z,X=Blue, A,O=Red and .=White
circle		Correspondence of colours to section character: Yellow: A Blue: B Black: Y Green: X Red: . (dot)

		Could be Japanese flag with .=Red and A,B,Y,X=White
cross		Correspondence of colours to section character: Red: A Blue: B Yellow: D Green: C Black: . (dot) Could be the St.George flag with .=Red and A,B,C,D=White
scots_cross		Correspondence of colours to section character: Red: A Blue: B Green: C Yellow: D Black: . (dot) Could be Scotland flag with .=White and A,B,C,D=Blue
dia_cross		Correspondence of colours to section character: Red: B Yellow: A Blue: D Green: C

New templates can be added later. There is also an open question to allow the players to design their own templates – this may come at some point.

Currently there is no server side checks for several players using the same flag. I've been thinking about that, but it's not easy to implement. It's possible to catch using the same template and the same colour, but really some colours look the same even they differ a little. You would not be able to differentiate on a screen the colours like these (255,0,0) and (254,0,0) – both look perfectly red. So, it's been decided to loose such a control. You can always check on the web page which flags are in use to avoid confusing collisions.

Diplomatic Mail

All the players in the Galaxy Kings are anonymous. No-one knows real names and emails of other players (except the GM, of course).

During the game all the playing nations can negotiate to each other using Diplomatic Mail via the game server. You may want to send a letter to an alien nation in order to make a peace or alliance, plan joint operations, demand something and so on.

There are two types of messages you can send:

- A personal message to any nation in the Universe if you know the nation name (see http://galaxy.magix.net/public/AllNations.html where you can find other nations names).
- A broadcast message to all the nations in your visibility area (the races with colonies visible by your nation).

Use English language for broadcast messages. You may try using any language of choice when sending personal messages, but it is not guaranteed that other than standard ASCII characters will be properly delivered.

You can send mail only to the nations ruled by real players, not to AI nations.

Donations

The game is absolutely free. But if you feel the game is enjoyable and you'd like to donate any amount, please do so. Currently, the donations are accepted by PayPal only. Please use email alexrzgm@gmail.com

The donations are rewarded. You can claim a 'bonus' in Galaxy Credits for each transaction you have made via PayPal. The amount of rewards depends on a sum of donation:

Donation in GBP (Great Britain Pounds)	**Reward in Galaxy credits for each GBP**
sum < 2	x100000
2 <= sum < 5	x150000
5 <= sum < 10	x200000
10 <= sum < 20	x300000
20 <= sum < 50	x500000
50 <= sum < 100	x700000
100 <= sum	x1000000

Examples:

£1 → 100,000 CREG (x100000)

£1.5 → 150,000 CREG (x150000)

£2 → 300,000 CREG (x150000)

£15 → 4,500,000 CREG (x300000)

£25 → 12,500,000 CREG (x500000)

£50 → 35,000,000 CREG (x700000)

£80 → 56,000,000 CREG (x700000)

£100 → 100,000,000 CREG (x100000)

£200 → 200,000,000 CREG (x100000)

…

Actually, a few GBP can make you quite a money to boost your economy. Or, if you need a tidy sum to buy a fleet from the 'dark dealers' before a big war, you can donate a bit more.

If you donate before joining the game, each GBP is multiplied by 2 making the reward much bigger.

Automated Managers

When the game progresses, your nation will have more and more colonies and ships and it will become more and more difficult to control all this stuff. You may spend hours to create orders for hundreds of ships and look after many tens or even hundreds colonies. You may loose control or even be fed up with creating too complicated orders like:

- manually building and sending transport ships from one location to another, loading them by some cargo, unloading them when they arrive
- manually building and sending scouts to investigate unknown star systems
- manually creating production orders to develop your colonies
- manually sending colonists where they needed to populate your colonies lacking population
- …

These tasks are quite manageable in the beginning when you have just a few colonies and ten or so ships. It's even advisable to control everything manually in the very beginning. But later, from turn to turn handling the nation becomes more tedious and difficult.

What you really want when playing the game is to focus on military operations against your enemies, defending your borders against attackers, 'doing politics' (creating alliances against common enemy, making friends with strong neighbours, and so on). All 'low-level' operations of researching the outer space, colonization, logistic, ships upgrade and repair, development and support of your colonies should ideally go automatically. The solution is the 'Automated Managers' (AM) you can switch ON or OFF at any moment.

There are following AM available:

- Exploration Manager
- Transport Manager
- Colonization Manager
- Development Manager
- Upgrade Manager
- Repair Manager
- Attack Manager
- Defence Manager

If you turn all of them ON, you'll transfer your nation to a pretty much AI nation and can do nothing, just view periodically incoming reports. It's not the best you can do, of course, because human players will definitely defeat you sooner or later. Also, you may be excluded from the game due to inactivity.

Exploration Manager

The Exploration Manager tries to explore all the star systems in your visibility and range (50*TechDrive LY from any of your colonies). To do that it uses the SCOUT_FLEET. Every turn the manager checks how many stars are to be explored and sends ships from SCOUT_FLEET to these stars (if they have not been sent yet). It works smart, sending ships which are the closest to the destinations. If there are not enough ships in the SCOUT_FLEET, the manager automatically places production orders to produce them with priority '-60' (note: all AM place the production orders with negative priorities to not disturb players orders which have default priority '0'). As a ship type it uses a default type 'Scout' (D1) – the simplest ever ship. A player may change this type to something else, for example adding a laser gun, like 'MyScout' (D1,R1x1). But be careful here, because your scout ship will fire at any alien ship, if they meet somewhere, and because there is no default 'Peace' to every one, a space battle is inevitable. Worse, this scout may start 'bombing' someone's colony (not with large destructions, but anyway irritating). This may really upset other players, and you may loose possible friendship. Note: when changing a ship type for the Exploration Manager a ship speed should not be less than 10LY, otherwise it will not be accepted.

The manager also takes care, that every your colony has at least one ship on the orbit. If not – it sends a scout there.

When you first switch the manager ON, it can be a bit aggressive in its actions trying to build tens or may be hundreds scouts (depending on your visibility).

To limit number of stars the Exploration Manager should explore, use command 'add star to manager' (see Commands). In this case only added stars will be explored.

Transport Manager

The Transport Manager (TM) is responsible for automatic delivering cargo from one location to another. It works on the 'order basis', i.e. it does something only if there are transport orders. If there are no orders, nothing happens.

The TM is heavily used by other AM:

- Colonization Manager places transport orders to move colonists
- Development Manager may issue some transport orders to move produced materials from rich colonies to poor ones to speed up their development
- Attack Manager may use the TM to move invasion forces against the enemy colonies

A transport order contains following fields:

- A cargo type (COL, MAT or IND)
- Amount of cargo, say 10000 units
- Source colony (where to load)
- Destination star (where to unload)

To execute an order, the TM employs the ships belonging to the TRANSPORT_FLEET. It tries to find the closest to the source colony free transport ships and orders them to deliver maximum amount of cargo they can carry from the source to destination. The TM knows how many ships currently busy with each order, how much cargo already delivered and to deliver, and so on. As soon as full cargo has been delivered, the transport order is removed. If there are not enough transport ships to execute all the orders, the TM places production orders to produce them with priority '-40'. As a ship type it uses a default type 'Transport Ship' (D10,C10) . Possibly it makes sense to change this default type to something bigger and add some weaponry for self-protection (but don't forget to set SERVICE ship type when you do so, to avoid your transport ships attacking enemies combat ships). A transport ship type should have a cargo bay and speed >= 5LY, otherwise it will be rejected.

As a player, you also may find useful to employ the TM sometimes. For example, to deliver 3000 COL from star #1 to star #2 you would need manually use several ships (let's consider the default 'Transport Ship' with relatively small cargo bay). For each ship you need to do following:

- issue a command to send a ship to the source colony #1 (if it's not there already)
- issue a command to load maximum possible cargo (COL in this case)
- add another command to send the ship to star #2
- when it arrives, issue a command to unload the ship
- (possibly send it back to #1 for another go)

So, here you need to handle all these commands and bookkeeping for several ships to deliver exactly 3000 COL from source to destination. Not easy and boring.

Using the TM you can issue just one order like:

```
move cargo --type COL --amount 3000 --from #1 --to #2
```

...and that's it – the TM will do the rest, and you may focus on diplomatic and military affairs

instead. The TM will find suitable ships, or build them and then deliver the cargo. Obviously all this will require several turns.

List of current transport orders and their progress can be visible in the turn reports. Transport order can be cancelled by special command ‘cancel transport order’.

Colonization Manager

This manager is responsible for colonization all the discovered uninhabited star systems. Such stars may be found by the Exploration Manager or manually by sending ships there – doesn't matter.

The manager also keeps the population in your colonies to be not less than colony Size, i.e. takes care about keeping the active population on the maximum level (imagine, after enemy bombing of one of your colonies, the active population dropped down below the colony size. The manager will automatically send additional COL here to fix the situation, but only if natural population grow is not sufficient).

Every turn, the Colonization Manager checks your colonies and generates transport orders, if necessary, to send COL where its needed. Here is a subject of COL availability – the manager can help only if there is available COL.

If the Colonization Manager finds uninhabited star systems with a size not less than 50 (currently this value is hard-coded), then it sends the closest available colony ship to colonize the system. If there are no available colony ships, the manager creates a production order to build them.

To limit number of stars the Colonization Manager should colonize (and also control keeping the population level at max possible), use command '*add star to manager*' (see Commands). In this case only added stars will be considered for colonization.

There is another useful command to handle the Colonization Manager, it's '*keep current borders*'. This commands applies '*add star to manager*' to all your current colonies and the star systems where colony ships currently moving to in order to create colonies. This command "makes" the manager to stop further expanding and focus on controlling already existing colonies.

Development Manager

The Development Manager takes care about the industry of your colonies to be at maximum possible level, i.e. equal to colony size. Generally it simply issues production orders to build the necessary amount of industry units. Also, in an event of your colony has been bombed by the enemies and industry has been destroyed partly or fully, the manager will place a right order to rebuild the industry.

Sometimes the Development Manager can abuse the Transport system 'a bit'. It may happen, if one of your new colonies has resource factor < 0.07. The manager considers (correctly!), that such a star should not build industry itself. Then it issues a production order to one of the neighbour colonies to produce materials and then transport all ready materials to the poor colony. These might be quite expensive orders, but hopefully, probability of such poor systems is not that high.

The Development Manager has default production orders priority '-50'.

To limit number of stars the Development Manager should develop, use command 'add star to manager' (see Commands). In this case only added stars will be managed.

Upgrade Manager

The Upgrade Manager is responsible for keeping your ships up-to-date. Every turn it checks all the ships currently located on your colonies orbits if their components need to be upgraded. If any of the following conditions is true, the manager tries to upgrade the component, if there are available PUs and money:

- If current technological level minus component tech level >= 2 (Example: component drive has tech 1.67, current TechDrive is 3.67 what makes 3.67-1.67 = 2: condition is true)
- If component is badly damaged (has efficiency <= 0.25) and component tech level is lower than current tech level (Example: component drive has tech 1.67, efficiency 0.25 or lower, current TechDrive is 1.68 or higher: condition is true). The manager decides that it's better to replace the component by a new one with current higher tech level, than trying to repair it
- If component is fairly damaged (has 0.25 < efficiency <= 0.5) and current technological level minus component tech level >= 1 (Example: component drive has tech 1.67, current TechDrive is 2.67 what makes 3.67-2.67 = 1, and component's efficiency is 0.5 or lower: condition is true)

So, the Upgrade Manager it's trying its best to upgrade old components and avoid repairing fairly old components if they are damaged.

If a component has passed the test for upgrade, the manager places an upgrade production order.

As soon as the upgrade physically starts, the old component is destroyed and replaced by the new one with current level of techs.

The upgrade order can last more than one turn depending on a volume of upgrade and available resources (PUs and money). The orders ends if:

- The component has been upgraded
- The ship leaves the colony (sent somewhere). In this case, the ship has partly upgraded component(s). As soon as it arrives to another colony (or returns back), it'll be repaired by the Repair Manager which 'indirectly' continues the broken upgrade. From the Repair Manager prospective – it's just a component with efficiency less than 100% and it need to be repaired. It makes perfect sense to activate the Repair Manager if you have activated the Upgrade Manager – the latter will always finish the job started by the former.

The Upgrade Manager has default priority 10 what is higher than player's default priority – mind the importance to keep your fleet up-to-date.

Repair Manager

The Repair Manager is responsible for keeping your ships effective. Every turn it checks all the ships currently located on your colonies orbits if their components need to be repaired, i.e. if their efficiency is less than 100%. If such ships with damaged components have been found , the Repair Manager issues the production repair order for all the ships on the same orbit. During production phase of the turn, the order is executed while there are available PUs and money or there are ships to be repaired. The most damaged ships are repaired first; the most damaged component inside the ship is also repaired first.

The Repair Manager has default priority 10 what is higher than player's default priority – mind the importance to keep your fleet effective.

To limit number of stars the Repair Manager should control, use command 'add star to manager' (see Commands). In this case only added stars will be used to repair the ships.

Attack Manager

The Attack Manager is responsible for attacking discovered enemies colonies by combat ships or invading them by sending transport with COL. This manager has been designed for the AI nations and it's not recommended to activate it.

Just a few words about how it works. If an enemy colony has been detected there are two possible scenarios:

- If colony has ships on the orbit, the manager tries to assembly a reasonable mission fleet and send it against the colony as soon as all ships arrived to the assembly point. The Attack Manager uses ships belonging to the 'ATTACK_FLEET' system fleet.

  Depending on a scale of the mission the attacking fleet is specified by 'size', and then decided which ships are to be included. It's always tried to include the biggest ships possible and the rest is populated by smaller ships.

  When creating a mission fleet the manager takes available ships from the 'ATTACK_FLEET' or issues production orders (with default priority '-80') to build new ones if they are not available. The types of new ships are based on the Attack Ship Type. A suitable ship type is calculated depending on the required mass. If it doesn't exist, it is created. Theoretically, preparing an attack can take a long time. During such preparation the conditions may change, the target enemy colony may have bigger fleet. The manager adopts the mission accordingly requesting more force before the attack.
- If a colony has no ships – just unprotected, the Attack Manager tries to send a number of transport ships with COL there to invade the colony.

The Attack Manager is very greedy for resources. If you activate it, your economy will be struggling with orders to build ships, assigned to different missions. May be it's not what you want.

The base Attack Ship Type can be changed by 'set ship type' command. Only types with weapons and drive are accepted.

To limit number of stars the Attack Manager should target for attack, use command 'add star to manager' (see Commands). In this case only added stars will be targeted.

Defence Manager

The Defence Manager automatically builds orbital defence stations to protect your colonies against enemy attack. This manager has also been designed for the AI nations, but could be used by the players as well (although the smart players may prefer using triggers (see Triggers)).

How it works. Every turn it checks your colonies for fully developed ones. A colony is considered as fully developed if it has been reached maximum population and industry. For every found fully developed colony the manager checks a total mass of defence ships, i.e. ships belonging to the 'DEFENCE_FLEET' system fleet. The Defence Manager issues production orders (with default priority '-70') to build the defence ships with total mass equal to 0.2*ColonySize (for example, for colony with size 1000, it'll build 200 mass of defence ships). As result it can be one or more ships based on the Defence Ship Type. A suitable ship type is calculated depending on the required mass. If it doesn't exist, it is created.

Note, the defence ships have no drive because they are orbital stations. Instead of drive they have more powerful shield and laser cannons.

The base Defence Ship Type can be changed by 'set ship type' command. Only types with weapons and no drive are accepted.

The Defence Manager does not protect colonies with size < 100 or with resource level < 0.05.

To limit number of stars the Defence Manager should defend, use command 'add star to manager' (see Commands). In this case only added stars will build defences.

Triggers

(Note: this chapter is for very advanced players with good programming experience. Using triggers makes sense only if your empire has grown up quite big, you have passion to automate your control over the galaxy and… you are experienced programmer.

The description is very brief, just gives you some highlights. Ask questions, I'll extend the guide.)

Trigger is a program called by the server at known turn calculation steps or at specific event. Triggers are written on a special 'ZX scripting programming language' (see Triggers Brief Programming Guide). Triggers are another way to automate your nation when it becomes too big.

The idea is the following. There are certain points during the turn calculation, like 'before turn starts' or 'when battle happens' or whatever. The server looks for a trigger written by a player and if such trigger exists – calls it. The trigger may create commands to be executed immediately or return some data which may change the server logic in this particular place. This adds a great flexibility for advanced players with programming experience to customize their nations.

Following trigger types are currently supported:

- **OnStarBeforeTurn**: executed for a specific star before turn. The trigger is designed to do some operations on a star system before turn calculation begins. It may be adding a command to build industry, or send some ships to the star or whatever is necessary

- **BeforeTurn**: executed after triggers 'OnStarBeforeTurn', but before turn calculation begins. This trigger may be used to process data created during execution triggers OnStarBeforeTurn (see examples below) or do some general actions before the turn.

- **WhenRemovedFromObject**: executed immediately when a trigger has been removed from any object (see examples below).

- **WhenCannotContinueRoute**: executed if a ship or a fleet on a route cannot continue to the next leg.

More triggers are planned to add.

Some triggers can be 'attached' to an object. 'Object' can be a star, a ship, or a fleet.

Each trigger must have a name. Trigger names should obey the naming rules (see chapter 'Names'). You can have any number of triggers of the same type with different names.

Object related triggers (like *OnStarBeforeTurn*), are executed only if 'attached' to specific object. General triggers (like *BeforeTurn*) are always executed.

Currently Supported Trigger Types

OnStarBeforeTurn

Type: Object

Can be attached to: any visible star

When called for a star system the server creates a variable ‘_star’ (type branch) populated by star data with the following fields:

Name	Type	Description
Id	long	Star id
Name	string	Star name
Size	long	Star size
ResourceFactor	double	Star resource factor
LiveConditions	double	Live conditions
Population	double	Whole population (with colonists)
Col	double	Colonists only
Materials	double	Currently available materials
Industry	double	Industry
Owner	string	Nation name: star owner. Empty if not colonized
LastKnownAt	long	Turn number with last known information (applicable for ‘known’ stars)
GalaxyName	string	Name of a galaxy the star belongs to
IsOwn	bool	‘true’ if it’s your colony, ‘false’ if not
IsBannedToVisit	bool	‘true’ if banned to visit
IsBannedToBomb	bool	‘true’ if banned to bomb
IsReachedMax	bool	‘true’ if star has reached Max possible development

Example of dummy usage:

```
update trigger --type OnStarBeforeTurn --name MyTestTrigger --code
#{
if (_star.IsOwn && _star.IsReachedMax)
  trace << “star #” << _star.Id << “ has reached max!”;
#}
;
```

BeforeTurn

Type: General

Cannot be attached to objects.

Executed every turn after all triggers OnStarBeforeTurn.

Example of dummy usage:

```
update trigger --type BeforeTurn --name MyTestTrigger --code
#{
  // Prints a number of ships moving to some star
  branch ships = _app.GetShips();
  trace << "You have "<< ships.Size() << " ships";
  long counter;
  for (long i = 0; i < ships.Size(); ++i)
  {
    string a = ships[i].Assignment;
    if (a.Find("moving to star") != -1)
       ++counter;
  } // for
  trace << "Found " << counter << " ships moving to stars";
#}
;
```

WhenRemovedFromObject

Type: General

Cannot be attached to objects.

When called for an object the trigger is removed from, the server creates a variable ‘_trigger’ (type branch) with the following fields:

Name	Type	Description
Name	string	Trigger name
Type	string	Trigger type
Id	string	Object id (it can be a name or a string representation of numeric value depending on object type). For instance for stars it would be a star id, for fleets, it would be a fleet name.

Example of dummy usage:

```
update trigger --type WhenRemovedFromObject --name MyTestTrigger --code
#{
  trace << "Called for "
        << _trigger.Type << "@" << _trigger.Name
        << ", id=" << _trigger.Id;
#}
;
```

WhenCannotContinueRoute

Type: General

Cannot be attached to objects.

This trigger fires if a fleet or a ship cannot continue a route (see command route) by some reasons. The most obvious reason for that is the next leg of the route points to star out of reach of the nation.

If the trigger executed without error, the route is paused and waits till it can continue (appropriate report message is generated in this case). The trigger could, for example, automatically create a colony on the star where the fleet has stuck. Or the player seeing the situation can issue command to create a colony. After that the next route leg will presumably be able to continue.

When the trigger is called, the server provides two variables helping the player to deal with the situation:

Variable **_star** – the same branch type as above for trigger OnStarBeforeTurn. It contains a description of the star system where the ship or fleet has stuck on the route

Variable **_object** (type branch) – a short description of an object (obviously ship or fleet) which has stuck on the route. It has the following fields:

Name	**Type**	**Description**
Type	string	Object type (‘fleet’ or ‘ship’)
Id	string	Object id (it can be a fleet name or a ship Id)

Example of dummy usage:

```
update trigger --type WhenCannotContinueRoute --name WhenCannotContinueRoute --code
#{
trace << _object.type << " '" << _object.Id << "' cannot continue route on star #" << _star.Id;
#}
;
```

For a ship it will trace something like:
ship ‘123’ cannot continue route on star #567
For a fleet:
fleet ‘F VIII’ cannot continue route on star #567

Although it’s possible to create many triggers of type ‘WhenCannotContinueRoute’, it makes no sense, unless you have something specific in mind. All of them will be called at the same time. That’s why in the example above the trigger has the same name as the type because I do mean that just one trigger of this type should exist.

Live Example of using the triggers

May be the best way to demonstrate how could you use triggers is just giving a real example. The code below is a real code you can copy&paste and use for yourself or use as a reference or an example.

Creating a Static Defence for your colonies

It’s an alternative to the Defence Manager way of defence automation for your colonies. The Defence Manager generates a specific ship type for each colony it protects. A generated ship type is based the “Defence Ship Type” and the ship size depends on the colony size. As result one ship is being built for each colony. Myself I don’t really like it and prefer to have a number of smaller but powerful missile ships, capable to destroy large targets. A number of ships depends on the size of a colony. That what the trigger below does. Let’s see the code:

```
update trigger --type OnStarBeforeTurn --name BuildStaticDefence --code
#{

// Currently used defence ship type: it depends
// on a size of the colony
string DefenceShipType;

// This function is a 'filter' to select
// the defence ships below by CalcShips
bool IsDefenceShip(branch aShip)
{
  return aShip.Type == DefenceShipType;
};

// This script ignores star systems with size < 800 and
// resource factor < 0.1
if (_star.IsOwn && _star.Size >= 800 &&_star.ResourceFactor >= 0.1)
{
  //---------------
  // STATIC DEFENCE
  //---------------

  // Industry level to start defence production and defence ship type
  double industryLevel;
  long minNumberOfShips;
  long maxNumberOfShips;
  // by the rules: stars > 2500 are considered as giants
  bool isGiant = _star.Size > 2500;

  if (isGiant)
  {
    // For giants we start building defence if production level has
    // reached a half size of the star system
    industryLevel = _star.Size/2;
    DefenceShipType = "Large Ship Type 1";
    minNumberOfShips = 5;
    maxNumberOfShips = 15;
  }
  else
  {
    // For 'normal star system we build defence when the industry
    // is fully built
    industryLevel = _star.Size - 1;
    DefenceShipType = "Smaller Ship Type 2";
    minNumberOfShips = 3;
    maxNumberOfShips = 10;
  }

  if (_star.IsReachedMax || _star.Industry >= industryLevel)
  {
    // Calculate a number of defence ships already on the orbit
    // (Here we are selecting the ships of DefenceShipType located
    // in the star system)
    branch fastFilter;
    fastFilter.Add(_star.Id, "Location");
    long numberOfDefenceShipsOnOrbit = _app.CalcShips(fastFilter, "IsDefenceShip");
    long numberOfDefenceShipsInProduction =
      _app.CalcShipTypeInProd(_star.Id, DefenceShipType);
    long planned = numberOfDefenceShipsOnOrbit + numberOfDefenceShipsInProduction;
```

```
    branch shipTypeInfo = _app.GetShipTypeInfo(DefenceShipType);

    // Calculate a number of ships to be built
    // should be between minNumberOfShips and maxNumberOfShips
    long requeredNumberOfShips = _star.Size/shipTypeInfo.Mass;
    long adjustedNumberOfShips = requeredNumberOfShips;
    if (requeredNumberOfShips < minNumberOfShips)
      adjustedNumberOfShips = minNumberOfShips;
    if (requeredNumberOfShips > maxNumberOfShips)
      adjustedNumberOfShips = maxNumberOfShips;

    if (planned < adjustedNumberOfShips)
    {
      trace << "--------------------------------";
      trace << "*** BuildStaticDefence for star #" << _star.Id
            << ", industry=" << _star.Industry
            << ", industryLevel=" << industryLevel
            << ", star.ResourceFactor=" << _star.ResourceFactor;

      trace << "defenceShipType=" << DefenceShipType
            << " (mass=" << shipTypeInfo.Mass << ")"
            << ", numberOfDefenceShipsOnOrbit=" << numberOfDefenceShipsOnOrbit
            << ", numberOfDefenceShipsInProduction=" << numberOfDefenceShipsInProduction
            << ", planned=" << planned;
      trace << "requeredNumberOfShips=" << requeredNumberOfShips
            << ", adjustedNumberOfShips=" << adjustedNumberOfShips;

      long toProduce = adjustedNumberOfShips - planned;
      trace << "*** Adding order to produce " << toProduce
            << " number of ships of type " << DefenceShipType << "...";

      // Command to produce ships (note: we need to log in before any command)
      cmd login my_email@gmail.com/my_password;
      cmd produce ships --type &DefenceShipType
                        --number &toProduce
                        --where &_star.Id
                        --priority "-888"
                        --sendto fleet: DEFENCE_FLEET;

      trace << "...Ok";
    } // if (planned < adjustedNumberOfShips)
  } // if _star.IsReached...
} // if (_star.IsOwn)

#}
;
```

The code is pretty self-documented. See the brief programming guide for syntax details.

Set real names for *DefenceShipType* variable for Giants and normal stars and your credentials (marked blue) to customize the script for yourself. Note: the ship types have to be already existing types or the script will not work.

It makes sense to assign the trigger ‘OnStarBeforeTurn’ to all stars ‘*’ - the script is smart and protects only colonies:

```
add trigger to object --type OnStarBeforeTurn --name BuildStaticDefence --id *;
```

Also, assigning the trigger to “*” guarantees that new colonies will be automatically “protected” by the trigger.

Creating a Light Defence for your colonies

The static defence supplied by either Defence Manager or a trigger (like the one above) builds defence for already developed colonies.

What if you'd like to protect just created colony which has nothing yet? This colony is weak and can be bombed out by any enemy ship with very little weaponry.

The idea is to create a number of fast light ships which are automatically sent (by your trigger) to any new colony to supply some sort of protection against enemy light ships (armed scouts and alike). These ships will stay on the colonies orbits till the colony builds up it's own static defence.

The light defence system is implemented using two triggers. One trigger of type 'OnStarBeforeTurn' is called for each attached star to detect if it needs defence or not. It also cancels assignments for ships guarding the stars which don't need the light defence any more, so the ships can be used to guard other stars . If it needs the defence, the trigger 'sends a message' caught by another trigger of type 'BeforeTurn'. This one assigns available ships to protect requested stars and builds defence ships if necessary.

See the code below.

```
update trigger --type OnStarBeforeTurn --name BuildLightDefence --code
#{
  //---------------
  // LIGHT DEFENCE
  // The purpose is to protect new not developed colonies
  // from enemy light ships
  // This trigger is called for each star it attached to and if
  // a star needs a defence, it sends a message to another trigger
  // BeforeTurn/BuildLightDefence.
  // If the star does not need a defence any more and has ships
  // already assigned to guard it, the trigger cancels those ships
  // assignments to make them free for other stars
  //---------------

  // Helper function: prints header when the trigger
  // is called and doing some job
  void PrintHeader()
  {
    trace << "--------------------------------";
    trace << "*** BuildLightDefence for star #" << _star.Id;
  };

  // Put your ship type here:
  string DefenceShipType = "My Defence Ship Type";

  // Filter function for GetShips below
  bool IsDefenceShip(branch aShip)
  {
    return aShip.Type == DefenceShipType;
  };

  // Get the ships already assigned to guard the star
  // These ships may be anywhere, not necessary in the star system
  branch fastFilter;
  fastFilter.Add("Guarding star #" + _star.Id, "Assignment");
  branch shipsAssignedToGuard = _app.GetShips(fastFilter, "IsDefenceShip");

  if (_star.IsOwn ||
      _app.CheckStarBelongsToManager(_star.Id, "ColManager"))
  {
    // Do the job only for colonies or
    // star systems assigned to the ColManager for colonization

    // Check current defence level
```

```
    // (Note: this obscure logic exists because
    //   operator || evaluates all expressions)
    branch defence;
    if (_star.IsOwn)
      defence = _app.CalcDefenceForStar(_star.Id);
    else
      defence.Add(0, "CurrentDefenceShipsMass");
    if (defence.CurrentDefenceShipsMass == 0)
    {
      // Need to defend this star because it doesn't have
      // any defence ships in the star system

      // Check if we have any ships assigned to guard the star
      if (!shipsAssignedToGuard.Size())
      {
        // Ok, this star has no ships assigned to defend it
        // Collect it (by sending message) to 'regular' trigger
        // 'BuildLightDefence'
        PrintHeader();
        trace << "Star #" << _star.Id
              << " has been scheduled for the Light Defence";
        _app.SendTriggerMessage("StarForLightDefence", _star.Id);
      } // if

      // We are done - stop trigger execution
      quit;
    } // if defence.CurrentDefenceShipsMass
  } // if star.IsOwn

  // The star is not own or already has some static defence
  // Make sure any ship assigned to guard the planet cancelled
  // to be used for other star systems needing defence

  if (shipsAssignedToGuard.Size())
  {
    PrintHeader();
    trace << "The star has no need in light defence.";
    if (!_star.IsOwn)
      trace << "(does not belong to the nation)";

    trace << "Cancelling " << shipsAssignedToGuard.Size()
          << " ships assigned to guard the star:";
    cmd login my_email@gmail.com/my_password;
    for (long i = 0; i < shipsAssignedToGuard.Size(); ++i)
    {
      branch ship = shipsAssignedToGuard[i];
      if (!ship.AssignmentCancelled)
      {
        trace << "Ship #" << ship.Id
              << ", location=" << ship.Location << ":";
        cmd cancel assignment for --ship &ship.Id;
        trace << "...Ok";
      }
    }
  } // if

#}
;


update trigger --type BeforeTurn --name BuildLightDefence --code
#{
  //---------------
  // This trigger collects all messages sent by trigger
  // OnStarBeforeTurn/BuildLightDefence and tries to find available
  // defence ships. The found ships are sent to defend the stars.
  // If there are not enough available ships, the trigger sends a command
  // to produce necessary number of them
  //---------------


  trace << "--------------------------------";
```

```
  trace << "*** BeforeTurn@BuildLightDefence";

  // Defence ship type: has to be the same as in script
  // OnStarBeforeTurn/BuildLightDefence
  string DefenceShipType = "My Defence Ship Type";

  // Filter function used in GetShips below
  bool IsDefenceShip(branch aShip)
  {
    return aShip.Type == DefenceShipType;
  };

  // Get all stars scheduled for defence by trigger
  // OnStarBeforeTurn/BuildLightDefence
  branch stars = _app.ReceiveTriggerMessages("StarForLightDefence");
  trace << "Found " << stars.Size()
        << " stars scheduled for light defence";
  if (stars.Size())
  {
    // Find all available defence ships
    branch fastFilter;
    fastFilter.Add("", "Assignment"); // no assignment
    fastFilter.Add("", "Fleet"); // does not belong to fleet
    branch availableShips = _app.GetShips(fastFilter, "IsDefenceShip");
    if (availableShips.Size() < stars.Size())
    {
      long required = stars.Size() - availableShips.Size();
      trace << "To defend " << stars.Size()
            << " required " << required << " ships";
      // Number of defence ships already in production
      long alreadyInProd = _app.CalcShipTypeInProd(0, DefenceShipType);
      if (alreadyInProd)
        trace << "  already in production: " << alreadyInProd;
      // ...so we need to build just this number:
      long toBuild = required - alreadyInProd;
      if (toBuild > 0)
      {
        trace << "Need to build " << toBuild
              << " defence ships of type " << DefenceShipType;

        cmd login my_email@gmail.com/my_password;
        trace << "*** Adding order to produce " << toBuild
              << " number of ships of type "
              << DefenceShipType << "...";
        cmd produce ships
          --type &DefenceShipType
          --number &toBuild --priority "-666"
          --hedge sizes:1000.., res: 0.5..;
        trace << "...Ok";
      } // if
    } // if availableShips.Size() < stars.Size()

    // Match available ships to the stars
    // (this function selects the closest ships to the stars)
    branch matched =
      _app.FindBestMatchSendingShipsToStars(availableShips, stars);
    trace << "Found " << matched.Size() << " matches";

    // Assign ships to guard
    cmd login my_email@gmail.com/my_password;
    for (long i = 0; i < matched.Size(); i++)
    {
      branch info = matched[i];
      trace << "Assigning ship #" << info.ShipId
            << " to guard star #" << info.StarId << "...";
      cmd guard --star &info.StarId --ship &info.ShipId;
      trace << "...Ok";
    } // for
  } // if (messages.Size())

#}
;
```

It makes sense to assign the trigger ‘OnStarBeforeTurn’ to all stars ‘*’ - the script is smart and protects only colonies or the star systems assigned to the ColManager:

```
add trigger to object --type OnStarBeforeTurn --name BuildLightDefence --id *;
```

Creating Mobile Defence System

The Light Defence above supplies just very weak temporary defence against enemy light ships. It's not a long term solution. Before a star system reaches the level when the static defence can be built, many years (sometimes decades) can go. And after that the static defence ships need to be built as well. This can take more years.

The mobile defence designed to give the developing star system more appropriate defence before it can protect itself. The mobile defence ships are supposed to have hyper drive. So, like the light defence ships, they can protect one star system till it needs them, then move to another one. It makes them weaker than any static defence ship, which supposed to have no hyper drive, but more weapon and shield. That's why it's not wise (or at least expensive) to rely on mobile defence only. It's a temporary measure.

By design, the mobile defence system is similar to the light defence one. It's also built on two triggers: one trigger of type 'OnStarBeforeTurn' is called for each attached star to detect does it need defence or not. It also cancels assignments for ships guarding the stars which don't need the light defence any more . If it needs, the trigger 'sends a message' caught by another trigger of type 'BeforeTurn'. This trigger assigns available ships to protect requested stars and builds defence ships if necessary. The main difference here is the mobile defence gives 10 defence ships for each star, not just one as the light defence does. It makes the triggers somewhat more complicated, but not that dramatic:

```
update trigger --type OnStarBeforeTurn --name BuildMobileDefence --code
#{
  //---------------
  // MOBILE DEFENCE
  // The purpose is to protect new not developed colonies
  // from enemy ships till the static defence is ready
  //---------------

  void PrintHeader()
  {
    trace << "--------------------------------";
    trace << "*** BuildMobileDefence for star #" << _star.Id;
  };

  string DefenceShipType = "My Defence Ship Type";
  long requiredNumberOfShips = 10;

  // Get all ships assigned to guard the star
  bool IsDefenceShip(branch aShip)
  {
    return aShip.Type == DefenceShipType;
  };
  branch fastFilter;
  fastFilter.Add("Guarding star #" + _star.Id, "Assignment");
  branch shipsAssignedToGuard = _app.GetShips(fastFilter, "IsDefenceShip");

  // Only own or stars listed by the ColManager are to be protected
  if (_star.IsOwn ||
      _app.CheckStarBelongsToManager(_star.Id, "ColManager"))
  {
    // Check current defence level
    // (Note: this obscure logic exists because
    //   operator || evaluates all expressions)
    branch defence;
    if (_star.IsOwn)
      defence = _app.CalcDefenceForStar(_star.Id);
    else
```

```
      defence.Add(0, "CurrentDefenceShipsMass");
    if (defence.CurrentDefenceShipsMass == 0)
    {
      // Need to defend this star

      // Check if we have any ships assigned to guard the star
      if (shipsAssignedToGuard.Size() < requiredNumberOfShips)
      {
        PrintHeader();
        long numberOfShipsToAssign =
          requiredNumberOfShips - shipsAssignedToGuard.Size();
        trace << "Star #" << _star.Id
              << " needs " << numberOfShipsToAssign
              << " mobile defence ships";

        string msg = _star.Id;
        _app.SendTriggerMessage(
          "MobileDefenceRequest", msg + "," + numberOfShipsToAssign);
      } // if

      quit;
    } // if defence.CurrentDefenceShipsMass
  } // if star.IsOwn

  // The star is not own or already has finished static defence
  // Make sure any ship assigned to guard the planet cancelled
  // (The star may become not own or be removed from ColManager)

  if (shipsAssignedToGuard.Size())
  {
    PrintHeader();
    trace << "The star has no need in mobile defence.";
    if (!_star.IsOwn)
      trace << "(does not belong to the nation)";

    trace << "Cancelling " << shipsAssignedToGuard.Size()
          << " ships assigned to guard the star:";
    cmd login my_email@gmail.com/my_password;
    for (long i = 0; i < shipsAssignedToGuard.Size(); ++i)
    {
      branch ship = shipsAssignedToGuard[i];
      if (!ship.AssignmentCancelled)
      {
        trace << "Ship #" << ship.Id
              << ", location=" << ship.Location << ":";
        cmd cancel assignment for --ship &ship.Id;
        trace << "...Ok";
      }
    }
  } // if

#}
;


update trigger --type BeforeTurn --name BuildMobileDefence --code
#{

  trace << "--------------------------------";
  trace << "*** BeforeTurn@BuildMobileDefence";

  string DefenceShipType = "My Defence Ship Type";
  bool IsDefenceShip(branch aShip)
  {
    return aShip.Type == DefenceShipType;
  };

  // Get all stars scheduled for defence
  branch request = _app.ReceiveTriggerMessages("MobileDefenceRequest");
  trace << "Found " << request.Size()
        << " mobile defence requests";
```

```
  if (request.Size())
  {
    // Find total number of requested ships
    long requestedShipsNumber = 0;
    for (long i = 0; i < request.Size(); ++i)
    {
      string r = request[i];
      branch tok = r.Split(",");
      if (tok.Size() != 2)
        throw ("Invalid request format: '" + r + "'");
      requestedShipsNumber += tok[1];
    } // for

    // Find all available defence ships
    branch fastFilter;
    fastFilter.Add("", "Assignment"); // no assignment
    fastFilter.Add("", "Fleet"); // does not belong to fleet
    branch availableShips = _app.GetShips(fastFilter, "IsDefenceShip");
    if (availableShips.Size() < requestedShipsNumber)
    {
      long required = requestedShipsNumber - availableShips.Size();
      trace << "To build mobile defence for " << request.Size()
            << " stars required " << required << " ships";
      long alreadyInProd = _app.CalcShipTypeInProd(0, DefenceShipType);
      if (alreadyInProd)
        trace << "  already in production: " << alreadyInProd;
      long toBuild = required - alreadyInProd;
      if (toBuild > 0)
      {
        trace << "Need to build " << toBuild
              << " defence ships of type " << DefenceShipType;

        cmd login my_email@gmail.com/my_password;
        trace << "*** Adding order to produce " << toBuild
              << " number of ships of type "
              << DefenceShipType << "...";
        cmd produce ships
          --type &DefenceShipType
          --number &toBuild --priority "-1111"
          --hedge sizes:1000.., res: 0.5..;
        trace << "...Ok";
      } // if
    } // if availableShips.Size() < stars.Size()

    // Match available ships to the stars
    branch matched =
      _app.FindBestMatchSendingShipsToStarsExt(availableShips, request);
    trace << "Found " << matched.Size() << " matches";

    // Assign ships to guard
    cmd login my_email@gmail.com/my_password;
    for (long i = 0; i < matched.Size(); i++)
    {
      branch info = matched[i];
      trace << "Assigning ship #" << info.ShipId
            << " to guard star #" << info.StarId << "...";
      cmd guard --star &info.StarId --ship &info.ShipId;
      trace << "...Ok";
    } // for
  } // if (messages.Size())

#}
;
```

It makes sense to assign the trigger ‘OnStarBeforeTurn’ to all stars ‘*’ - the script is smart and protects only colonies or the systems assigned to the ColManager:

```
add trigger to object --type OnStarBeforeTurn --name BuildMobileDefence --id *;
```

Star Wall defence system

This makes sense if your empire has grown really big, having hundreds of stars. Coming to such a state you may find that your frontier star systems are under constant attacks from other ‘barbarian’ AI nations. You definitely can arrange a peace with a human races, but can’t do anything about AI (at least for now).

These hostile AI nations usually sporadically attack your systems, causing heavy damage and losses, especially if your systems are not well protected. Why not to build a sort of a wall or a fence, surrounding your empire and not letting the enemy ships in? Like Chinese Wall?

The idea is to surround your territory by heavy fortified colonies, selected so that no enemy ships can fly in without landing on one of them. Here you’d get the intruders.

The proposed system has been built in a way similar the Mobile defence system above. But the defence ships are supposed to be heavier and be capable to destroy any known enemy ship – make sure of it, otherwise your wall would be broken.

In the example I have used a number (15) of heavy missile ships.

The star wall system has been built using 3 triggers. Trigger of type ‘OnStarBeforeTurn’ makes sure the star has enough defence ships. If not, it sends a message caught by trigger of type ‘BeforeTurn’ requesting the necessary ships. Trigger ‘BeforeTurn’ assigns requested ships to the star systems or builds them if necessary. The last trigger of type ‘WhenRemovedFromObject’ cancels assignments for any ships guarding the star system removed from the star wall. So, these ships can be re-used and sent to protect another star belonging to the wall.

The code is pretty much similar the examples above.

```
update trigger --type OnStarBeforeTurn --name STAR_WALL --code
#{
  //---------------
  // STAR WALL
  // The purpose is to protect the borders
  //---------------

  void PrintHeader()
  {
    trace << "--------------------------------";
    trace << "*** BuildStarWall for star #" << _star.Id;
  };

  string defenceShipType = "My ship type";
  long requiredNumberOfShips = 15;
  bool IsDefenceShip(branch aShip)
  {
    return aShip.Type == defenceShipType;
  };

  branch fastFilter;
  fastFilter.Add("Guarding star #" + _star.Id, "Assignment");
  long shipsAssignedToGuard = _app.CalcShips(fastFilter, "IsDefenceShip");

  // Check if we have any ships assigned to guard the star
  if (shipsAssignedToGuard < requiredNumberOfShips)
  {
    PrintHeader();
    long numberOfShipsToAssign =
      requiredNumberOfShips - shipsAssignedToGuard;
```

```
    if (numberOfShipsToAssign > 0)
    {
      trace << "Star #" << _star.Id
            << " needs " << numberOfShipsToAssign
            << " star wall defence ships";
      string msg = _star.Id;
      _app.SendTriggerMessage(
        "StarWallRequest", msg + "," + numberOfShipsToAssign);
    } // if
  } // if

#}
;


update trigger --type BeforeTurn --name BuildStarWall --code
#{

  trace << "--------------------------------";
  trace << "*** BeforeTurn@BuildStarWall";

  string defenceShipType = "My ship type";
  bool IsDefenceShip(branch aShip)
  {
    return aShip.Type == defenceShipType;
  };

  // Get all stars scheduled for defence
  branch request = _app.ReceiveTriggerMessages("StarWallRequest");
  trace << "Found " << request.Size()
        << " star wall requests";

  if (request.Size())
  {
    // Find total number of requested ships
    long requestedShipsNumber = 0;
    for (long i = 0; i < request.Size(); ++i)
    {
      string r = request[i];
      branch tok = r.Split(",");
      if (tok.Size() != 2)
        throw ("Invalid request format: '" + r + "'");
      requestedShipsNumber += tok[1];
    } // for

    // Find all available defence ships
    branch fastFilter;
    fastFilter.Add("", "Assignment"); // no assignment
    fastFilter.Add("", "Fleet"); // does not belong to fleet
    branch availableShips = _app.GetShips(fastFilter, "IsDefenceShip");
    if (availableShips.Size() < requestedShipsNumber)
    {
      long required = requestedShipsNumber - availableShips.Size();
      trace << "To build star wall for " << request.Size()
            << " stars required " << required << " ships";
      long alreadyInProd = _app.CalcShipTypeInProd(0, defenceShipType);
      if (alreadyInProd)
        trace << "  already in production: " << alreadyInProd;
      long toBuild = required - alreadyInProd;
      if (toBuild > 0)
      {
        trace << "Need to build " << toBuild
              << " defence ships of type " << defenceShipType;

        cmd login my_email@gmail.com/my_password;
        trace << "*** Adding order to produce " << toBuild
              << " number of ships of type "
              << defenceShipType << "...";
        cmd produce ships
          --type &defenceShipType
          --number &toBuild --priority "-2222"
          --hedge sizes:1000.., res: 0.5..;
```

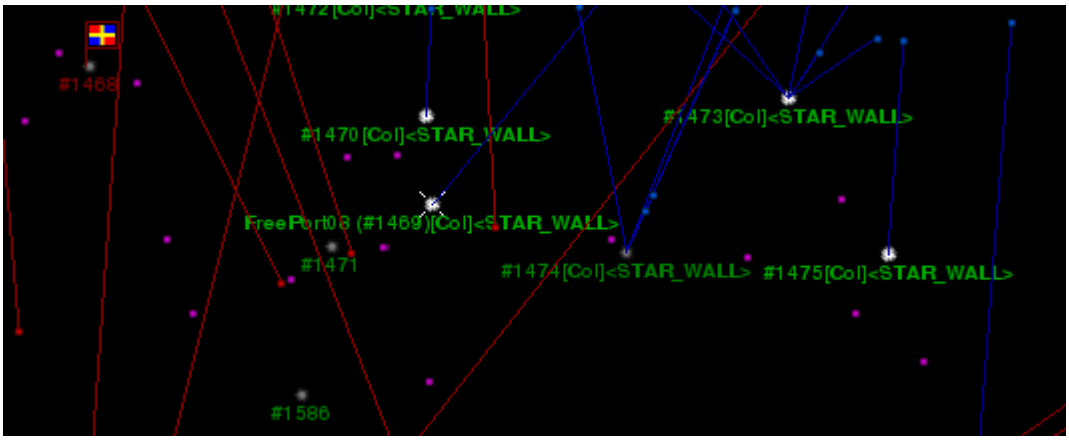
```
        trace << "...Ok";
      } // if
    } // if availableShips.Size() < stars.Size()

    // Match available ships to the stars
    branch matched =
      _app.FindBestMatchSendingShipsToStarsExt(availableShips, request);
    trace << "Found " << matched.Size() << " matches";

    // Assign ships to guard
    cmd login my_email@gmail.com/my_password;
    for (long i = 0; i < matched.Size(); i++)
    {
      branch info = matched[i];
      trace << "Assigning ship #" << info.ShipId
            << " to guard star #" << info.StarId << "...";
      cmd guard --star &info.StarId --ship &info.ShipId;
      trace << "...Ok";
    } // for
  } // if (messages.Size())

#}
;


update trigger --type WhenRemovedFromObject --name CancelStarWall --code
#{

  trace << "--------------------------------";
  trace << "*** WhenRemovedFromObject@CancelStarWall";
  trace << "Called for "
        << _trigger.Type << "@" << _trigger.Name
        << ", id=" << _trigger.Id;

  string defenceShipType = "My ship type";
  bool IsDefenceShip(branch aShip)
  {
    return aShip.Type == defenceShipType;
  };

  branch fastFilter;
  fastFilter.Add("Guarding star #" + _trigger.Id, "Assignment");
  branch shipsAssignedToGuard = _app.GetShips(fastFilter, "IsDefenceShip");
  trace << "shipsAssignedToGuard=" << shipsAssignedToGuard.Size();

  if (shipsAssignedToGuard.Size())
  {
    trace << "Cancelling " << shipsAssignedToGuard.Size()
          << " ships assigned to guard the star "
          << _trigger.Id << ":";
    cmd login my_email@gmail.com/my_password;
    for (long i = 0; i < shipsAssignedToGuard.Size(); ++i)
    {
      branch ship = shipsAssignedToGuard[i];
      if (!ship.AssignmentCancelled)
      {
        trace << "Ship #" << ship.Id
              << ", location=" << ship.Location << ":";
        cmd cancel assignment for --ship &ship.Id;
        trace << "...Ok";
      }
    }
  } // if

#}
;
```

You are supposed to assign the star wall trigger individually to selected stars like:

```
add trigger to object --type OnStarBeforeTurn --name STAR_WALL --id 6290;
add trigger to object --type OnStarBeforeTurn --name STAR_WALL --id 5600;
```

```
add trigger to object --type OnStarBeforeTurn --name STAR_WALL --id 5610;
//...
```

The stars with attached triggers will be marked in the Galaxy Viewer star map:



Triggers Brief Programming Guide

The triggers are written on a “zx script language” which has been created to write short configuration scripts or triggers, compiled on a fly into internal chain of executable commands and executed by server in real-time.

A few notes about the language:

- It’s not C++ or Java, do not try to find any commonalities – it’s just has similar syntax in some ways
- It’s a configuration language designed to write **short** scripts and handle tree-looking data (includes a multi-purpose container called ‘branch’). Trying to write large programs may be not an easy task
- There are no debugging facilities, apart from trace command you can use to display interesting data

A typical script is a text file consisting of operations (or commands) separated by semicolons:

command1;

command2;

command3; command4;

...

Each command is a keyword with or without string parameter which depends on command syntax.

Examples:

```
long a; // command ‘long’ declares variable ‘a’
for (long i = 0; i < 10; ++i); // command ‘for’ creates a for loop
if (a < 3) {}; // command ‘if’ creates an ‘if’ condition
a = 10; // implicit command – assignment 10 to variable ‘a’
```

‘yellow’ colour highlights commands arguments, **bold** – are keywords.

The language is easily extendible: new commands can be added without any difficulties. Some commands look like operators in C or Java (**for** or **if** for example), but in reality they are commands with their unique syntax. That’s why some compiler errors may look confusing. Example:

```
int a;
Execution trace:
       : 0000> int a;
       : !!! Compilation error ------->line 0: Unknown command "int"
```

There is no data type ’int’, but the compiler ‘thinks’ that ‘int’ is a command, not a declaration. The compiler doesn’t have a concept of declarations, only a concept of commands. All built-in data types automatically considered as ‘declarations’, but ‘int’ is not one of them.

Data types

long: represents an integer value (default initialization to 0)

bool: boolean value (can be *true* or *false*, defaulted to *false*)

double: represents a float value (default initialization to 0)

string: any string value (default initialization to empty string “”)

date: represents date/time value (default initialization to current date/time)

branch: container of pairs <name, value> where name is a string and value is any supported data type (including a branch). (default initialization to empty container)

Comments

```
// it’s a single line comment
/*
Multi
   line
      comment

*/

/*
  /*
   /*
Nested comments are fine.
Make sure a number of opening tags /* is equal to the number of closing ones */
  **/
 */
*/
```

Declarations

<datatype> <variable name>;

<datatype> <variable name> = <expression>;

Examples:

```
long a; // declares a variable ‘a’ of type long. Default value is 0
double d = 3.1415; // declares ‘d’ variable of type double and initializes it to
                   // 3.1415
branch a; // error! Variable ‘a’ already declared in this scope
branch myList; // creates an empty branch ‘myList’
```

Scopes and visibility

Scope is an area of code between the curved brackets {} (like in c/java). Any script starts from a ‘global scope’. Variables/functions declared in an inner scope are not visible from an outer scope, but variables/functions declared in outer scope are visible in inner scope. Possible conflicts resolved in a favour of inner scope.

Examples:

```
// Script starts – here is a global scope:
int a = 1;

// Start inner scope
{
  string a = “hello”; // hides int ‘a’ above
  trace << “string a=” << a;
  // Start another inner scope
  {
    double a = 1.23; // hides string ‘a’ above
    trace << “double a=” << a;
  }
}
trace << “int a=” << a;
```

This script will print the following:

string a=hello

double a=1.23

int a=1

Functions

It’s possible to write functions, the syntax is similar to C/Java. Functions can return any of existing data types or ‘void’ when returning nothing. Function must be declared before the first usage.

Examples:

```
// Declaration:
void Func1()
{
  trace << “I am Func1”;
};

// Call
Func1();
```

This will print: *I am a Func1*

```
double Mult(double a, double b)
{
  return a * b;
};
trace << Mult(3.15, 10);
```

This will print: *31.5*

Like any scope ‘{}’ functions obey the same rules. The can see variables and functions declared in the scope above:

```
long a = 3;
long Func(long n)
{
  return a + n; // ‘a’ is visible here
};
trace << Func(2) << “,” << Func(10);
```

This will print: *5,13*

Output

Use command **trace** or **cout** to output data:

```
trace << “Hello world”;
```

Prints: *Hello world*

```
trace << “a=” << 37 + 51.6 << “kg”;
```

Prints: *a=38.6kg*

```
cout << “wow!”;
```

Prints: *wow!*

Built-in objects

There is a built-in object called **_app** with a number of useful methods you can use in your triggers. In the triggers live examples one can see various applications of this object.

The call syntax is: *_app.<MethodName>([<parameter1, <parameter2>,...])*

The _app object contains the following methods:

long CalcShipTypeInProd(long id, string shipType);

Returns a number of ships of specified type currently in production.

Parameters:

id (long): id of the star system. If 0, function returns a number of produced ships on all your colonies.

shipType (string): ship type name.

Examples:

```
// Number of all ships ”Peace Maker 409” in production
long all = _app.CalcShipTypeInProd(0, “Peace Maker 409”);
// Number of ships ”Peace Maker 409” in production in star system 378
long in378 = _app.CalcShipTypeInProd(378, “Peace Maker 409”);
```

branch FindClosestAvailableShip(long id, string shipType);

Returns a descriptor of the closest available ship of type ‘shipType’ to star system ‘id’. The descriptor is a variable of type ‘branch’ with 2 fields: ‘Id’ (long) and ‘ETA’ (long). If there is no available ship of requested type, the function returns an empty descriptor with Id == 0 and ETA==0.

A ship of type ‘shipType’ is considered as ‘available’ if this ship has no assignment and it does not belong to any fleet.

Parameters:

id (long): id of the star system.

shipType (string): ship type name.

Examples:

```
branch res = _app.FindClosestAvailableShip(101, “Peace Maker 409”);
trace << “Found ship with id=“ << res.Id << “, ETA=” << res.ETA;
```

branch GetShipTypeInfo(string shipType);

Returns a descriptor of a ship of type ‘shipType’. The descriptor contains the following fields:

Mass (double): ship mass

LoadedMass (double): ship loaded mass

Currently that’s it… More fields describing the ship type will be added soon

Parameters:

shipType (string): ship type name.

Examples:

```
    branch shipTypeInfo = _app.GetShipTypeInfo(DefenceShipType);
    long requeredNumberOfShips = _star.Size/shipTypeInfo.Mass;
```

void SendTriggerMessage(string name, string value);

Sends a ‘message’ from one trigger to another. The ‘sender’ trigger should be executed **before** the ‘receiver’ trigger. The message mechanism is widely used in the trigger based defence systems presented as examples above. The triggers ‘OnStarBeforeTurn’ send messages with individual information to be executed later triggers of type ‘BeforeTurn’ to perform some actions

Parameters:

name (string): message name. The receiver trigger should use the same name when calling ‘ReceiveTriggerMessages’ to obtain the sent data.

value (string): message value – can be any string

Examples:

```
        string msg = _star.Id;
        _app.SendTriggerMessage(
          "MobileDefenceRequest", msg + "," + numberOfShipsToAssign);
```

branch ReceiveTriggerMessages(string name);

Returns all trigger messages with a name in parameter “name” sent before the calling time. The method returns a branch where each element has index-looking name like ‘0’, ‘1’,’3’,...’<number of elements in the branch - 1>’, so the messages can be iterated and processed.

Parameters:

name (string): message name

Examples:

```
// Get all stars scheduled for defence
branch request = _app.ReceiveTriggerMessages("MobileDefenceRequest");
trace << "Found " << request.Size()
      << " mobile defence requests";

if (request.Size())
{
  // Find total number of requested ships
  long requestedShipsNumber = 0;
  for (long i = 0; i < request.Size(); ++i)
  {
    string r = request[i];
    branch tok = r.Split(",");
    if (tok.Size() != 2)
      throw ("Invalid request format: '" + r + "'");
    requestedShipsNumber += tok[1];
  } // for
```

branch GetShips(branch fastFilter, string slowFilter);

Returns all ships satisfying criteria in ‘fastFilter’ and ‘slowFilter’. The ‘fastFilter’ is a branch with following fields:

Assignment (string): if specified returns ships with assignment string equal to this field

AssignmentPart (string): if specified returns ships with assignment string containing a substring stored in AssignmentPart

Fleet (string): if specified, returns ships belonging to the fleet

Location (string): if specified returns ships located at the star system id

Examples:

Here all ships with assignment string ‘Guarding star #<whatever _star.Id is>’ are returned (for example ‘Guarding star #123’):

```
branch fastFilter;
fastFilter.Add("Guarding star #" + _star.Id, "Assignment");
branch shipsAssignedToGuard = _app.GetShips(fastFilter);
```

Returns all ships located in the system _star.Id:

```
branch fastFilter;
fastFilter.Add(_star.Id, "Location");
branch shipsAssignedToGuard = _app.GetShips(fastFilter);
```

Returns all ships belonging to fleet ‘Fleet III’:

```
branch fastFilter;
fastFilter.Add(“Fleet III”, "Fleet");
branch shipsAssignedToGuard = _app.GetShips(fastFilter);
```

Returns all ships towing something (obviously another ship) to star 102. Using “AssignmentPart” field here is essential, because the assignment string for tow ship looks like “towing to star #<starId> ship #<shipId>”. We don’t know which ship is towed, but we know the star, that’s why we should use a substring specified in “AssignmentPart” for searching.

```
branch fastFilter;
fastFilter.Add(“towing to star #102 ship”, "AssignmentPart");
branch shipsTowingToStar = _app.GetShips(fastFilter);
```

Returns all ships without assignment and not belonging to any fleet (note: explicitly specifying a filter field with empty string):

```
   branch fastFilter;
   fastFilter.Add("", "Assignment"); // no assignment
   fastFilter.Add("", "Fleet"); // does not belong to fleet
   branch availableShips = _app.GetShips(fastFilter, "IsDefenceShip");
```

Returns all ships (note: no filter at all – all ships are returned):

```
branch shipsAssignedToGuard = _app.GetShips();
```

Be careful with returning all ships – there can be really many of them. The operation can take long time and be interrupted by the server. Normally your trigger of type «OnStarBeforeTurn» is expected to fit 1 second limit . The server breaks execution of a trigger if it takes more than 1

second.

The ‘fastFilter’ is not flexible, but it’s working fast (the server code just checks known 3 fields using hard-coded c++ procedure). Unfortunately, there are many cases when such inflexibility is not enough. Here we can use so named ‘slowFilter’. The ‘slowFilter’ is a name of a function inside the visibility scope of GetShip call site. The function accepts a ship as a parameter and must return bool (true or false) what means the ship has passed or not passed the filter.

A ‘slowFilter’ has to be used in combination with the ‘fastFilter’ due to performance reasons: the ‘fastFilter’ always called first, then, if it passed – the ‘slowFilter’ gets called.

Example:

This code selects all ships of type ‘my type’ without assignment and not belonging to any fleet:

```
  string DefenceShipType = "my type";
  bool IsDefenceShip(branch aShip)
  {
    return aShip.Type == DefenceShipType;
  };
...
    // Find all available defence ships
    branch fastFilter;
    fastFilter.Add("", "Assignment"); // no assignment
    fastFilter.Add("", "Fleet"); // does not belong to fleet
    branch availableShips = _app.GetShips(fastFilter, "IsDefenceShip");
```

There are plenty of similar examples above (see ‘live examples’).

The function returns a branch containing the descriptors of ships satisfying the filter criteria. The ship descriptor has the following fields:

Id (long): ship id

Type (string): ship type

Location (string): ship location (star id or “space”)

Assignment (string): assignment string

AssignmentCancelled (bool): ‘true’ if assignment is cancelled, ‘false’ if not

Fleet (string): fleet name

LoadedMass (double): ship’s loaded mass

Speed (double): ship’s speed

The fields can be accessed from the script using ‘.’. Example:

```
  if (shipsAssignedToGuard.Size())
  {
    for (long i = 0; i < shipsAssignedToGuard.Size(); ++i)
    {
      branch ship = shipsAssignedToGuard[i];
      if (!ship.AssignmentCancelled)
      {
        trace << "Ship #" << ship.Id
              << ", location=" << ship.Location << ":";
        cmd cancel assignment for --ship &ship.Id;
        trace << "...Ok";
      }
    }
```

long CalcShips(branch fastFilter, string slowFilter);

Returns a number of all ships satisfying criteria in ‘fastFilter’ and ‘slowFilter’. Parameters are absolutely the same as for GetShip function.

Prefer CalcShips to GetShips().size() if you need just a number of ships: the CalcShips works faster and consumes less resources.

branch FindBestMatchSendingShipsToStars(branch ships, branch stars);

This high-level functions helps to select which ships from ‘ships’ is the best to send to stars, specified in ‘stars’ (ids). Returns a branch of pairs ShipID/StarID.

There are plenty examples of using this functions in ‘live examples’ above. In a few words: for instance you have 5 ships to send to 3 stars. The function selects the closest ships to each star and returns pairs – which ship to be sent to which star. A number of pairs will be the smallest number (ships or stars), in this example it’s 3.

Example:

One can see here we’ve got a container of available ships to guard a list of stars. The function gives you the best selection of pairs ‘ship → star’:

```
// Match available ships to the stars
// (this function selects the closest ships to the stars)
branch matched =
  _app.FindBestMatchSendingShipsToStars(availableShips, stars);
trace << "Found " << matched.Size() << " matches";

// Assign ships to guard
cmd login my_email@gmail.com/my_password;
for (long i = 0; i < matched.Size(); i++)
{
  branch info = matched[i];
  trace << "Assigning ship #" << info.ShipId
        << " to guard star #" << info.StarId << "...";
  cmd guard --star &info.StarId --ship &info.ShipId;
  trace << "...Ok";
} // for
```

branch FindBestMatchSendingShipsToStarsExt(branch ships, branch stars);

This is an extended version of function ‘FindBestMatchSendingShipsToStars’ providing a wider flexibility. The difference is branch ‘stars’ contains not the star system ids, but strings in format “<star id>, <required number of ships>”, for example “343,2” (star #343 needs 2 ships).

This function is used in the live example of STAR WALL triggers above. Here are some fragments of code:

Trigger ‘OnStarBeforeTurn’ STAR_WALL sends a message ‘StarWallRequest’ to another trigger of type ‘BeforeTurn’ called BuildStarWall

```
update trigger --type OnStarBeforeTurn --name STAR_WALL --code
#{
...
      string msg = _star.Id;
      _app.SendTriggerMessage(
        "StarWallRequest", msg + "," + numberOfShipsToAssign);
...
```

One can see the message is constructed as _star.Id + “,” + numberOfShipsToAssign

Trigger BuildStarWall receives all the messages, figures out how many available ships and how many are to be built and calls FindBestMatchSendingShipsToStarsExt for available ships to find the best matches ship → star and sends the ships to the stars:

```
update trigger --type BeforeTurn --name BuildStarWall --code
#{
...
  if (request.Size())
  {
    // Find total number of requested ships
    long requestedShipsNumber = 0;
    for (long i = 0; i < request.Size(); ++i)
    {
      string r = request[i];
      branch tok = r.Split(",");
      if (tok.Size() != 2)
        throw ("Invalid request format: '" + r + "'");
      requestedShipsNumber += tok[1];
    } // for

    // Find all available defence ships
    branch fastFilter;
    fastFilter.Add("", "Assignment"); // no assignment
    fastFilter.Add("", "Fleet"); // does not belong to fleet
    branch availableShips = _app.GetShips(fastFilter, "IsDefenceShip");
    if (availableShips.Size() < requestedShipsNumber)
    {
      long required = requestedShipsNumber - availableShips.Size();
      trace << "To build star wall for " << request.Size()
            << " stars required " << required << " ships";
      long alreadyInProd = _app.CalcShipTypeInProd(0, defenceShipType);
      if (alreadyInProd)
        trace << "  already in production: " << alreadyInProd;
      long toBuild = required - alreadyInProd;
      if (toBuild > 0)
      {
...
      } // if
    } // if availableShips.Size() < stars.Size()

    // Match available ships to the stars
    branch matched =
      _app.FindBestMatchSendingShipsToStarsExt(availableShips, request);
    trace << "Found " << matched.Size() << " matches";

    // Assign ships to guard
```

```
    cmd login my_email@gmail.com/my_password;
    for (long i = 0; i < matched.Size(); i++)
    {
      branch info = matched[i];
      trace << "Assigning ship #" << info.ShipId
            << " to guard star #" << info.StarId << "...";
      cmd guard --star &info.StarId --ship &info.ShipId;
      trace << "...Ok";
    } // for
  } // if (messages.Size())

#}
;
```

bool CheckStarBelongsToManager(long starId, string managerName);

This function returns true if star system with id ‘starId’ belongs to automated manager ‘managerName’. It’s designed to check a particular star system belongs it to, say’ ColManager’ or not, and add it or remove it from the script.

Example:

```
if (!_app.CheckStarBelongsToManager(123, “ColManager”))
  cmd add star for manager --name ColManager --id 123;
```

double GetDistanceToNotColony(long starId);

This function returns a distance in light years to the closest star system from the star starId which is not a colony. It’s convenient when you need to estimate how far away are potentially not friendly star systems from your colony. This may have effect on the defence you might want to build on your colony.

Example:

```
  double distanceToNotColony = _app.GetDistanceToNotColony(_star.Id);
  trace << "Distance to closest non colony is " << distanceToNotColony;
  if (distanceToNotColony >= 200 || _star.Size < 1000) // distance >= 200
  {
    requiredNumberOfSmallShips = 1;
  }
  else
  {
    if (distanceToNotColony >= 100) // 100 <= distance < 200
    {
      requiredNumberOfBigShips = CorrectRequiredNumberOfBigShips(_star.Size() / 2000);
    }
    else // distance < 100
    {
      requiredNumberOfBigShips = CorrectRequiredNumberOfBigShips(_star.Size() / 1000);
    }
  }
```

bool CheckAllColoniesInDistance(long starId, long distance);

The function returns true if all star systems inside the radius from the star starId are colonies, otherwise it returns false. It operates similar to the function above (**GetDistanceToNotColony**).

Example:

```
if (_app.CheckAllColonisInDistance(_star.Id, 200)) // distance >= 200
{
  // safe in 200 light years around – needs one ship
  requiredNumberOfSmallShips = 1;
}
else
{
  // not safe: needs 10 ships
  requiredNumberOfSmallShips = 10;
}
```

Limitations

Ability to program triggers makes the server vulnerable during turn processing because of intentionally or not intentionally badly written code.

DO NOT EVER TRY THE FOLLOWING THINGS:

1. Unlimited loops

Example:

```
for (;;);
while (true);
```

2. Unlimited self-recursive function calls

Example:

```
void f()
{
  f();
}
```

3. Any other server-breaking attempts…

The server code has a protection against such. One of the method used is a time limitation for trigger executing. Triggers of type ‘OnStarBeforeTurn’ are supposed to be very short and quick. The server allows 1 second for the trigger to complete. Triggers of type ‘BeforeTurn’ have more relaxed timing: up to 30 seconds. All the ‘live’ examples above work inside these time limits for a nation with more than 20000 ships and 1000 colonies.

If a trigger breaches the time limits, or causes any fatal error (like stack overflow (mind recursive calls) or division to zero or whatever), the execution stops, the trigger code is automatically commented out what effectively disables it. The player should be notified about this fact by a message. If a player repeatedly tries to harm the server by bad written code, the GM may exclude the player from the game.

Turn Order

The game turns are calculated in the following order:

- Executed all scheduled orders collected from the players before the turn (orders can be scheduled for future turns)
- Processed all bonus requests and Galaxy Credits are added to players Treasury
- For each nation executed triggers ‘OnStarBeforeTurn’
- For each nation executed triggers ‘BeforeTurn’
- For each nation is calculated amount of 'Fleet Payments' and added to the “National Debt”
- “National Debt” is offset from the Treasury
- Processing Ships On Planet Orbits for each nation (for all the ships which are not in space):
    - Updated information about the star systems (if nation's ship on a planet orbit and it's not nation's colony, the up-to-date records are updated for this planet; if its someone else's colony, up-to-date information is recorded about colony's owner)
    - Performed load/unload operations for transport ships (this may result in a Ground Battle, if an invasion force has landed on enemy's planet)
- Calculated movements of all the ships currently in the space or launched.
- Ships which lost their fleets (for example due to severe drive damages they were automatically temporary excluded from the fleet) rejoin the fleets
- Running all space battles
- For each nation updated visible and own stars
- Processed Buy Stars requests
- For each nation run Exploration Manager if activated
- For each nation run Colonization Manager if activated
- For each nation run Upgrade Manager if activated
- Run 'In-Place' ships repair by Repair Units
- For each nation run Repair Manager if activated
- For each nation run Defence Manager if activated
- For each nation run Attack Manager if activated
- For each nation run Development Manager if activated
- For each nation run Transport Manager if there are transport orders
- For each nation run production
- Run 'Galaxy Promotion' calculation (identifying Galaxy Masters and Emperors)
- Again for each nation the Nation 'Debt' is offset by money from the Treasury (hoping that some money have been generated during production)

- Increasing population of each inhabited star system
- Updating information about other nations ships
- Checking players inactivity counters and removing inactive players (see Quitting the Game).
- Aristocracy promotion of players
- Generating and sending reports for each player

How to join the game

There are two ways to join this game:

- You may become a ruler of an existing nation if such is currently available. The available to join nations have no owner either because they have been originally created as AI nations, or the previous owner has left the game. Joining an existing nation you may benefit from already established economy, developed technologies, built fleets, number of colonies, and so on. From another hand you may find yourself surrounded by many neighbours not necessary friendly, inherit a number of existing enemies and running wars, problems, bad controlled finances. You'll be immediately involved in lots of activities without any hand-over from the previous player – imagine – starting to rule an existing state!

  Look at the rating table of “Available Nations” http://galaxy.magix.net/public/AvailableNations.html to make a guess which nation to select. Normally, the higher rating, the better chances to hit the pot. It also wise to look at technologies levels – even with not the best rating, the higher technologies have a good potential.

- You may choose to start a brand new nation. In this case by default you'll be placed in a random location of the Universe where the server can find for you a suitable Home World. In most cases it will be a new location, somewhere in the frontier to the not discovered by others sectors, but this is not guaranteed. You may be occasionally placed in the location, abandoned by other nations for some reasons, and this place can be in the middle of well developed and populated galaxy. Pray to not be discovered soon…

  But normally, as a new joiner, you'll have quite a number of turns for development before the first contact to aliens. Depending on your location, this period may differ from a few turns to a few tens of turns. Sometimes you may start to think you are 'alone in the Universe', but it's not true. Keep calm and carry on – use this time for development and build your military.

To join the game you need to send an order to the galaxy server (galaxyservergm@gmail.com) with a command '`join game`' where to specify the following details:

- Your player name, for example 'Alex the Great'
- Your gender: 'M' or 'F'
- Password
- Email where to send reports and messages
- Name of your nation. If you join with existing nation, the name must be an exact name of one of the available nations (http://galaxy.magix.net/public/AvailableNations.html) – just copy&paste the one you wish to join. If you join as a new nation, it must be a valid name different from any existing nation (check here: http://galaxy.magix.net/public/AllNations.html).

The server will reply with the order translation. If there are no errors, the translation email will contain a confirmation code. You need to repeat the command ‘join game’ with this code to

complete registration during 10 minutes (see details in Commands). If you fail to repeat command with a right confirmation code or the code has expired, the server will delete the pending request and you'll have to repeat the process.

Recently a new feature has been added allowing a new joiner to select a suitable home world (see http://galaxy.magix.net/public/GameLoader.html – joining the galaxy with the Game Loader). The Game Loader make joining process a way easier.

Quitting the game

Each player has the 'Inactivity Counter' set to 21 which is decremented each turn. As soon as it reaches 0 the player is automatically removed and it's nation is converted to AI nation. The counter is reset to its maximum value (21) every time the player sends successful '`log in`' command (see Commands). This gives to each player an inactivity time frame of 20 turns. 'Inactivity' basically means no control of the game – if you don't issue any orders or messages (It is wise to 'ping' the server periodically by '`log in`' commands even if you activated all the Automated Managers, just to avoid to be removed from the game). Note: commands “log in” issued by the triggers do not reset the counter.

When inactivity counter reaches 4, the server starts sending you warnings every turn in the report emails, something like: 'You will be quitting the game in X turns'.

If you quitted, your former nation will continue to exist, controlled by the server. If you decide to return to the game after a while, your former nation might be not available any more (either exterminated or picked up by some other player). So, you would need to start a new nation from scratch or pick up something available.

If you decide to quit the game quickly, you may send to the server a command '`quit the game`'. This command will set your Inactivity counter straight to 4.

Game Etiquette

Please always keep in mind that this is just a game. If your opponents behave treacherously, badly to your nation – don't worry – you can do the same against them. Everything is like in real diplomatic affairs...

The GM does not censor any messages, but avoid by all means any sort of abuse. The GM has rights (and ability!) to remove any bad behaving player from the game at once without warning.

It the Galaxy Kings it is not forbidden to play for several nations simultaneously, if you have time and stamina to do that. For each nation you are ruling you need a different email address.

Mini Games

This is a feature for those who likes “equal start conditions for all players”.

General game flow is virtually endless. When you join the game at some point there will be lots of nations/players who are well developed and may easily defeat you in a case of confrontation. Sure, you can find a Home World in a quiet sector of the Universe, but there is a possibility, that you’ll stay there alone for quite a while, meeting no one or just some aggressive AI races.

The idea of mini games is to collect a number of players together in a separate remote galaxy and join them the game together at the same turn. This will guarantee approximate equality of start conditions. A mini game will naturally end sooner or later by one of the following reasons:

- just one player has survived (this player is declared as a “Victor”)
- one of the players becomes a “Galaxy Master” (this player is declared as “Galaxy Master”)
- one of the players had the most votes (Not implemented yet: Voting system will be added soon)

The mini games are created by GM. Before the game has started (either automatically because it has necessary number of players or by GM command) it has a “Pending” state. New players can join this game. Players joining the pending game also have “Pending” status. When the game has started, a galaxy for the game is created and all the players are added to the game in this galaxy. The game is now in “Running” state and all the players are “Active”.

Each running mini game has its own rating table where the players can check how well they are progressing. When the running mini game ends, all players are automatically removed, all their nations are erased (actually, they are renamed to random names and erased only if there are no “real” players in the galaxy (see below)), the game galaxy is removed (if there are no “real” players), the victor is nominated, and the game moves to “Archived mini games”. Each archived mini game has a copy of the final rating table.

There is a possibility to join already running mini game if its galaxy has enough available home worlds. You can join a running mini game only as a new nation.

It may happen, that some players have left the mini game using command “quit the game”. In this case the abandoned nation automatically turns to AI (common procedure in the Galaxy Kings). But, the further fate of this nation is not monitored my the mini game any more. It keeps existing only to annoy the players who are still playing. It is also possible, that someone decides to join Galaxy Kings with this abandoned nation. It is permitted, but this player will not be a part of the mini game. This player can do its own “game” annoying existing players. Note, the mini games exist inside the global Universe and there is no rule to stop interfering from “outside”. There is possibility that some existing nation will invade the galaxy running under mini game, but this is hard to achieve, because such a nation would need to a) discover this galaxy what is hard b) overcome many hundreds or even thousands light years to reach it.

All players of a mini game are created as “temporary”. It means that they are automatically removed from the Galaxy Kings as soon as the game finishes. However, there is a possibility to continue the “global game” after the mini game ends. A player can change its status from “temporary” to “permanent” using command “set player mode permanent”. If a mini game when it ends has at least one real player (either converted from temporary or invaded from outside or joined later an abandoned player nation), the game galaxy is not removed, so, the “permanent” players can

carry on playing as long as they want.

The turn numeration in the mini games is “transparent”, i.e. Galaxy Kings numeration is used. For example, if a mini game can have start turn 717 and end turn 826.

To join existing mini game use command “join minigame”. To remove registration from a pending mini game use command “quit minigame”. You can’t quit this way, if the mini game is already running. In this case use command “quit the game”.

Orders

The orders is a way you control the game flow in your favour. The server keeps going according to the turns schedule independently from have you sent orders or not.

An order is text you can email to 'galaxyservergm@gmail.com' (Galaxy Viewer can send orders directly to the server via internet without email). If you send an email, the subject should contain words 'Galaxy Kings', for example:

“Re: ***Galaxy Kings***: MESSAGE TRANSLATION”

The server ignores (and doesn't reply!) any email if these words have not been found in the subject.

Orders may consists from a number of commands. Each command has following format:

```
<command name> [parameters];
```

where parameters can be anything, depending on the specific command syntax. Usually, parameters string is a number of pairs “parameter name/parameter value”. Normally, if a parameter name has more than one letter, it is prefixed by '`--`' (double dash); one letter parameters usually prefixed by single dash '`-`'. Prefixed parameters can appear in any order, for example:

```
change user --name Victor --email vct@o2.co.uk;
```

is the same as:

```
change user --email vct@o2.co.uk --name Victor;
```

Note: every command should be terminated by semicolon '`;`'.

There are few 'pre-processor' directives with syntax:

```
#<name>
```

They are not commands, but server needs them to translate the orders or messages inside an email text.

The list of commands should start with directive line '`#begin`' and end with directive '`#end`' – both should be on the separate lines. Example:

```
#begin
login myemail@virginmedia.com/123456;
set nation flag cross:000000:.=FF0000;

#end
```

The server ignores all the lines in the email before the first found '`#begin`' and all the lines after the first found '`#end`'. Don't try to write several 'begins' – the second one will be treated as error, because there is no such a command:

```
#begin
#begin ← bad, an error
```

```
login myemail@virginmedia.com/123456;
set nation flag cross:000000:.=FF0000;

#end
```

You can use comments in the order text:

`//` - single line comment

`/* … */` - multi-line comment.

Examples:

```
#begin
// this is a single line comment

/*
This is a multi-line
comment
*/

#end
```

The server ignores the commented text during parsing the order.

Don't create too long single line comments. Your email program may automatically split long lines on several shorter lines. As result, a single line comment can be split and part of it can become not a comment, but an invalid command:

Example:

You try to send this order:

```
#begin
// this is a very very very very long single line comment
...
#end
```

But email program may do this:

```
#begin
// this is a very very very very long single
 line comment ← OOPS! 'line comment' there is no such command!
...
#end
```

You may split long commands on several lines like here:

```
#begin
join game
     // change to your name here:
     --name Victoria
     // should be M or F:
```

```
    --gender F
    // change to your password:
    --password qwertty
    // change to your email address:
    --email adolf@gmx.de
    // change to available nation name:
    --nation England
    -e;
#end
```

what is equivalent to:

```
#begin
join game --name Victoria --gender F --password qwertty --email
adolf@gmx.de --nation England -e;
#end
```

You also can place several commands on the same line (command should terminate with semicolon, remember!):

```
#begin
login myemail@virginmedia.com/123456; get report --turn 0;
#end
```

The server executes commands immediately. There is no way to roll them back. However, each command can be cancelled by another command. For example, if you sent ship or fleet by mistake, you can cancel assignment for ship or fleet. If you placed a production order by mistake – you can cancel it by ‘cancel prod order’ command. And so on.

It is possible to schedule command for future turns (up to 20). To do this add a pre-processor directive #turn+<turn delta> in any place between #begin and #end. Better to do it right after #begin. It is permitted to use only one directive #turn in the order text. Example:

```
#begin
#turn+17
login myemail@virginmedia.com/123456;
send --ship 562 --to Moon;
#end
```

These commands will not be executed right now, but scheduled for turn current + 17.
If you have scheduled something by mistake, it’s easy to ‘neutralize’ it by scheduling another doing nothing:

```
#begin
#turn+17
login myemail@virginmedia.com/123456;
nop; // ←- no action, empty command
#end
```

You can use #turn+1 till #turn+20. Any deltas less than 1 or bigger than 20 are not processed and treated as an error.

Most of the commands need logging in to be properly executed. This means the '`login`' command should appear in the order before those commands. A good practise is to start your order with '`login`' command. “Logged in” state is kept till the end of the order (till #end).

Messages

To create a private or broadcast message send an email to 'galaxyservergm@gmail.com' (make sure the subject has words 'Galaxy Kings' anywhere).

A message should start with directive line '`#message`' and end with directive '`#end`' – both should be on separate lines. On the next line after the message directive should follow '`login:`' statement in format <email address/password> where you type your email (the one registered during joining the game) and password. On the next line after login should follow an address '`to:`' - who to send the message:

```
#message
login: <your email/password>
to: <who to send the massage>
<Message text line1>
<Message text line2>
...
<Message text lineN>
#end
```

Address 'to' can have following options:

- All – broadcast message to all nations in your visibility
- Nation Name (or several names, separated by comma) – a private message to one or more nations
- GM – message to the Game Master

Don't mix messages and orders in the same email! Send them separately. Server ignores everything in the text after directive '`#end`'. Obviously, if you add a message after an order or vice versa, only the first will be processed by the server.

Examples:

Broadcast message:

```
#message
login: myemail@gmail.com/123456
to: All
Hello All! Good day in the Galaxy.

Best wishes,
Alex The Great

#end
```

Message to Nation 'Gremlins':

```
#message
login: myemail@gmail.com/123456
to: Gremlins
```

```
Hello Ukrop, the Governor of Gremlins,
Please stop attacking my colony #123 or I'll destroy you!

Best Regards,
Alex The Great

#end
```

Message to 3 nations (Atlanta, Big Bunnies, French Republic):

```
#message
login: myemail@gmail.com/123456
to: Atlanta, Big Bunnies, French Republic


Dear All,

Let's unite our efforts to destroy the extremely annoying
Gremlins. I can send my fleet to destroy their colony #145 and
then move further to #156 and #172. Please let me know if your are
interested. In any case please keep this in secret from Gremlins.

Kind Regards,
Alex The Great

#end
```

Message to GM:

```
#message
login: myemail@gmail.com/123456
to: GM


Dear GM,

It would be great to improve the Ship Designer in the Report
Viewer to make it possible to freeze shield or speed when changing
other parameters.

Thank you,

Best Regards,

Alex The Great


#end
```

Commands in alphabetical order

activate/deactivate

Format:

`activate <automated manager name>`

`deactivate <automated manager name>`

Needs log in: yes

Description:

These two commands activate or deactivate an automated manager. The parameter '`automated manager name`' can be one of the followings (see Automated Managers for details):

- **ColManager**: colonization manager
- **ExplManager**: exploration manager
- **DevManager**: development manager
- **UpgradeManager**: upgrade manager
- **RepairManager**: repair manager
- **AttackManager**: attack manager
- **DefenceManager**: defence manager

Please use the exact names, case-sensitive.

Examples:

```
#begin
login myemail@gmail.com/myPassword;
activate ColManager; // activates colonization manager
activate RepairManager; // activates repair manager
deactivate AttackManager; // disables attack manager
activate upgradeManager;  ← bad, there is no such manager (mind the low case 'u')
activate Defence Manager;  ← bad, there is no such manager (mind the space)
activate ExplorationManager;  ← bad, there is no such manager (must be ExplManager)
#end
```

add science

Format:

```
add science
     --name <science name>
     --prop <drive tech>,<weapon tech>,<shield tech>,<cargo tech>
```

Needs log in: yes
Description:

Adds a new science with name '`science name`'. See Sciences for details. Parameter '`prop`' specifies the science 'parts' – proportions of different technologies the new science consists of.

Once added science can be removed by command '`delete science`'.

Examples:

```
#begin
login myemail@gmail.com/myPassword;

// Adding science 'TotalDrive' with proportions 40 for drive,
// 25 for weapon, 25 for shield and 10 for cargo technologies
add science
     --name TotalDrive
     --prop 40, 25, 25, 10;

add science
     --name Bad1
     --prop 40, 25, 25; ← bad, number of proportions must be 4

add science             ← bad, missed name
     --prop 40, 25, 25, 2;

add science
     --name TotalDrive ← bad, science 'TotalDrive' already exists! (added above)
     --prop 4, 2, 2, 1;

#end
```

add star for manager

Format:

```
add star for manager
     --name <auto manager name>
     --id <star id or name>
```

Needs log in: yes

Description:

Adds a star to be used by automated manager

Parameters:

- `name:` (mandatory) automated manager name. Can be one of 'ColManager', 'DefenceManager', 'AttackManager', 'DevManager', 'ExplManager', 'RepairManager'
- `id`: (mandatory) star id or name.

Examples:

```
#begin
login myemail@gmail.com/myPassword;

// Adds 'Solaris' for colonization by ColManager
add star for manager --name ColManager --id Solaris;
// Adds star 67 for exploration by ExplManager
add star for manager --name ExplManager --id #67;

#end
```

attach

Format:

```
attach
     --ship <ship id or name>
     --to <ship id or name>
```

Needs log in: yes

Description:

Attaches a ship to another ship with tow unit. The tow ship can drag the attached ship to any location and then detach the attached ship (using command “detach”). It is impossible to attach a ship, if a result speed after attachment is less than 1.

Parameters:

- `ship:` (mandatory) id or a name of the attaching ship
- `to:` (mandatory) id or a name of the ship with tow unit

Example:

```
#begin
login myemail@gmail.com/myPassword;

// attaches a ship with id 10 to another ship with name Carrier
// ship ‘Carrier’ must have a tow unit, capable to attach the ship
attach --ship 10 --to Carrier;
#end
```

add trigger

Format:

```
add trigger
     --type <trigger type name>
     --name <trigger name>
     --code
     #{
<zx script language text>
     #}
```

Needs log in: yes

Description:

Adds a trigger of specific type and name. Fails if a trigger ‘type’/’name’ already exists

Parameters:

- `type:` (mandatory) trigger type, can be one of the following types: ‘OnStarBeforeTurn’, ‘BeforeTurn’, ‘WhenRemovedFromObject’, ‘WhenCannotContinueRoute’
- `name:` (mandatory) trigger name. The name should obey the ‘object’ name rules (see naming stars, ships, fleets…).
- `code:` (mandatory) trigger code – should be valid compilable code written in zx script language

Example:

In this example a star system is checked for reaching MAX of development and population and adds an order to produce 10 ships of type ‘DefenceShip’.

(Note: don’t use this example for real – it just shows the syntax. Basically, this trigger will issue an order to produce 10 defence ships **every** turn once a star system, the trigger is attached to, has reached the MAX.)

```
#begin
login myemail@gmail.com/myPassword;

add trigger
  --type OnStarBeforeTurn
  --name ProduceDefenceShips
  --code
#{ ***star code macro***IND
    if (_star.IsReachedMax))
      cmd produce ships
        --type DefenceShip
        --where &_star.Id // using & is a way
                          // to pass variable into command
        --number 10;
#} ***end code macro***
```

```
; // semicolon must be on the new line after closing #}

#end
```

#} and **#{** are macros used by pre-processor on your client application to mark the start (#{) and the end (#}) of the trigger code. The problem is the trigger code may contain any free text in zx script language, including other commands. To make the server understand that all this block of text as a single command, the code has to be included inside the area limited by the macros #{...#}. These macros must be at the beginning of a new line (like #begin and #end). Also note, the entire command should be ended by semicolon – on the new line!
Sounds a bit complicated, but its the way its done to fit a free text into a single command. More advanced client applications may provide a convenient interface to write triggers avoiding this 'cryptology'.

A few examples of 'bad code':

```
#begin
login myemail@gmail.com/myPassword;

add trigger
  --type OnStarBeforeTurn
  --name ProduceDefenceShips
  --code #{ // bad: macro not on new line – won't work
    if (_star.IsReachedMax))
      cmd produce ships
        --type DefenceShip
        --where &_star.Id
        --number 10;
#}; // bad: semicolon on the same line as macro – won't work
#end
```

add trigger to object

Format:

```
add trigger to object
     --type <trigger type name>
     --name <trigger name>
     --id <object id>
```

Needs log in: yes

Description:

Attaches an existing trigger ‘type’/’name’ to an object with ‘id’.

Parameters:

- `type:` (mandatory) trigger type, can be one of the following types: ‘OnStarBeforeTurn’. Currently only this trigger can be attached to star systems
- `name:` (mandatory) trigger name. Should be a name of existing trigger
- `id:` (mandatory) object id (or name). Wild-card ‘*’ can be used to assign trigger to “all” objects of suitable type (for trigger ‘OnStarBeforeTurn’ it would mean ‘all star systems’)

Examples:

```
#begin
login myemail@gmail.com/myPassword;

// Adds trigger ‘BuildStaticDefence’ to all star systems
add trigger to object
  --type OnStarBeforeTurn
  --name BuildStaticDefence --id *;

// Adds trigger ‘BuildInd’ to star system 3443
add trigger to object
  --type OnStarBeforeTurn
  --name BuildInd --id 3443;

// Adds trigger ‘BuildInd’ to star system Moon
add trigger to object
  --type OnStarBeforeTurn
  --name BuildInd --id Moon;

#end
```

allow access to

Format:

```
allow access to --star <star id or name>
```

Needs log in: yes

Description:

Removes a specified star from the 'restricted area' (See Restricted Areas). To add star to the restricted area use command 'restrict access to'

Parameters:

- `star`: (mandatory) star id or name.

Examples:

```
#begin
login myemail@gmail.com/myPassword;

// Remove 'Solaris' from restricted area
allow access to --star Solaris;
// Remove star 67 from restricted area
allow access to --star 67;
#end
```

allow bombing

```
allow bombing --star <star id or name>
```

Needs log in: yes

Description:

Resumes ability of your ships to bomb the specified star system, prevented from bombing earlier by command 'prevent bombing'.

Parameters:

- `star`: (mandatory) star id or name. It must be one of the stars in your visibility areaю

Examples:

```
#begin
login myemail@gmail.com/myPassword;

// Resumes bombing the 'Solaris' star system
allow bombing --star Solaris;
// Resumes bombing #67 star system
allow bombing --star 67;
allow bombing --star #67; // the same as above
#end
```

buy clones

Format:

```
buy clones
     --amount <how much to buy>
     --where <location: star id or name>
     [--priority <order priority>]
```

Needs log in: yes

Description:

Places a 'buy-order' to buy clones or additional population (see Buying for details). Make sure you have enough Galaxy Credits to buy the requested amount in the specified location, otherwise the order will be rejected straight away.

Priority make sense only amongst the buy orders: those with higher priority will be executed first. Buy orders, which cannot be fulfilled (obviously because of lack of money) during the current turn will be cancelled.

Parameters:

- `amount`: (mandatory) number of units to buy
- `where`: (mandatory) location where to buy (can be star id or star name)
- `priority`: (optional) order priority

Examples:

```
#begin
login myemail@gmail.com/myPassword;

// Buying 50 clones units on Solaris
buy clones --amount 50 --where Solaris;

// Buying 150 clones units on the star system 123
// with priority 1
buy clones --amount 150 --where 123 --priority 1;

buy clones --where 123 --priority 1; ← bad, missed amount
buy clones --amount --where 123 --priority 1; ← bad, missed amount
buy clones --amount 10; ← bad, missed where
buy clones --amount 10 --where ; ← bad, missed where
#end
```

buy industry

Format:

```
buy industry
     --amount <how much to buy>
     --where <location: star id or name>
     [--priority <order priority>]
```

Needs log in: yes

Description:

Places a 'buy-order' to buy industry (see Buying for details). Syntax and meaning of parameters are the same as in '`buy clones`'.

Examples:

```
#begin
login myemail@gmail.com/myPassword;

// Buying 50 industry units on Solaris
buy industry --amount 50 --where Solaris;
#end
```

buy materials

Format:

```
buy materials
     --amount <how much to buy>
     --where <location: star id or name>
     [--priority <order priority>]
```

Needs log in: yes

Description:

Places a 'buy-order' to buy materials (see Buying for details). Syntax and meaning of parameters are the same as in '`buy clones`'.

Examples:

```
#begin
login myemail@gmail.com/myPassword;

// Buying 50 materials units on #123;
buy materials --amount 50 --where 123;
#end
```

buy ships

Format:

```
buy ships
     --type <existing ship type>
     --where <location: star id or name>
     [--number <how many ships to buy>]
     [--sendto <star: <star id> or fleet: <fleet name>>]
     [--priority <order priority>]
```

Needs log in: yes

Description:

Places a 'buy-order' to buy ships of existing ship type in the specified location (see Buying for details). It also gives a possibility to send the ready ships to specified star system or join them to a fleet.

Parameters:

- `type`: (mandatory) ship type name
- `number`: (optional) number of ships to buy, default 1
- `where`: (mandatory) location where to buy (can be star id or star name)
- `sendto`: (optional). If specified should have one of two 'directions':
    - `star <star id or Name>`: a location where the bought ships will be automatically sent
    - `fleet <Fleet Name>`: a fleet name the bought ships will be ordered to join
- `priority`: (optional) order priority

Examples:

```
#begin
login myemail@gmail.com/myPassword;

// Buying 10 'Scout' ship on star system #32
buy ships --where 32 --type Scout --number 10;

// Buying 1 'Transport Ship' ship on star system #32
buy ships --where 32 --type Transport Ship --number 1;
// Buying 1 'Transport Ship' ship on star system #32
// (note: '--number' by default is 1)
buy ships --where 32 --type Transport Ship;

// Buying 1 'AttackLineShip' on star system #32
// with priority 7 and sending it to star 'New Paris'
```

```
buy ships
     --where 32
     --type AttackLineShip
     --sendto star: New Paris
     --priority 7;

// Buying 1 'AttackLineShip' on star system #32
// and ordering it to join fleet 'Fleet 7'
buy ships
     --where 32
     --type AttackLineShip
     --sendto fleet: Fleet 7;

#end
```

buy star

Format:

```
buy star --name <star id or name>
```

Needs log in: yes

Description:

Places a 'buy-order' to buy a star system (see Buying Colonies for details).

Examples:

```
#begin
login myemail@gmail.com/myPassword;

// Buying star Zodiak
buy star --name Zodiak;

// Buying star system #123
buy star --name 123;
#end
```

buy tech

Format:

```
buy tech
     <--name <DRIVE,SHIELD,WEAPON,CARGO>
     [<--amount <number of pu>]
     [<--tolevel <tech level>]
     [--where <star name or id>]
     [--priority <order priority>]
```

Needs log in: yes

Description:

Places a 'buy-order' to buy specified technology (see Buying for details).

Parameters:

- `name` (mandatory): a technology name. Should be one of: `DRIVE,SHIELD,WEAPON` or `CARGO`
- `amount` (mandatory if `tolevel` is not specified): specifies amount of invested (actually 'bought') PUs
- `tolevel` (mandatory if `amount` is not specified): specifies a desirable level of technology (amount of necessary PUs will be calculated automatically)
- `where` (mandatory): a location to buy – a star id or name
- `priority` (optional): order priority

You should always specify either '`amount`' or '`tolevel`' parameters, but not both at the same time. If you specify '`tolevel`', the necessary amount of PUs to develop the technology from current level will be calculated.

Examples:

```
#begin
login myemail@gmail.com/myPassword;

// Buying investment of 1000 PUs for technology DRIVE on star #10
buy tech --name DRIVE --amount 1000 --where 10;

// Buying development tech WEAPON to level 3 on star Solaris
// with priority 1
buy tech --name DRIVE --tolevel 3 --where Solaris --priority 1;

buy tech --name Drive --amount 1000 --where 10; ← bad, unknown tech 'Drive': should be DRIVE

buy tech --name DRIVE,WEAPON --amount 1000 --where 10; ← bad, unknown tech 'DRIVE,WEAPON': command can handle only one tech at a time
```

```
buy tech --name DRIVE --amount 1000 --tolevel 3 --where 10; ← bad, parameters 'amount' and 'tolevel' are together in one command

buy tech --name DRIVE --where 10; ← bad, neither 'amount' nor 'tolevel' is specified

#end
```

cancel assignment for

Format:

```
cancel assignment for --ship/fleet <id or name>
```

Needs log in: yes

Description:

Cancels current assignment for ship or fleet.

This command is useful when you'd like to cancel an assignment for ship or fleet. It make sense for long running (taking more than one ‘jump’ or guarding) assignments like the ones created by command ‘route’ or ‘guard’.

Either --ship or --fleet option should be used, but not together

Parameters:

- `ship:` (mandatory) ship ids range or name (see command send for detailed description of ranges, the same is hear)
- `fleet`: (mandatory) fleet name

Examples:

```
#begin
login myemail@gmail.com/myPassword;

// Cancel assignment for ship ‘Death Star’
cancel assignment for --ship Death Star;

// Cancel assignment for ship 123
cancel assignment for --ship 123;

// Cancel assignment for fleet F VIII
cancel assignment for --fleet F VIII;

cancel assignment for --ship 1 --fleet V; ← bad, both --ship and --fleet used

#end
```

cancel prod order

Format:

```
cancel prod order <id>
```

Needs log in: yes

Description:

Cancels previously placed production order by its id.

This command is useful when you'd like to stop long running or not realistic or created by mistake production order. A list of existing production orders is available in the Galaxy Viewer where one can find a unique id for each order. Use this id to cancel the order.

Examples:

```
#begin
login myemail@gmail.com/myPassword;

// Cancel production order with id #123
cancel prod order 123;
#end
```

cancel transport order

Format:

```
cancel transport order <id>
```

Needs log in: yes

Description:

Cancels previously placed transport order by its id. A transport order can be created as a result of command 'ship'.

This command is useful when you'd like to stop long running or not realistic or created by mistake transport order. A list of existing transport orders is available in the Galaxy Viewer where one can find a unique id for each order. Use this id to cancel an order.

Examples:

```
#begin
login myemail@gmail.com/myPassword;

// Cancel transport order with id #123
cancel transport order 123;
#end
```

change user

Format:

```
change user
     [--name <new name>]
     [--gender <new gender M or F>]
     [--password <new password>]
     [--email <new report email>]
     [--code <confirmation code>]
```

Needs log in: yes

Description:

Changes player's attributes specified by '`join game`' when joined the game (see detailed parameters descriptions there).

Note, after changing an email all the messages and reports will start coming to the new email.

'gender' can be only one of two 'M' or 'F' – anything else is rejected as an error.

To change a nation name use another command '`rename nation`'.

Confirmation code is required only if you are changing the email. In this case like in ‘join game’ command a confirmation code will be sent to the new email and you have to repeat the command with the code to make the change be successful.

Examples:

Changes the name only to 'Napoleon The One':

```
#begin
login myemail@gmail.com/myPassword;
change user --name Napoleon The One;
#end
```

Changes email, password and gender:

```
#begin
login myemail@gmail.com/myPassword;
change user
     --email vicont@home.su
     --password sg388nnQwQ
     --gender M
     --code 34556; //← it is supposed the confirmation code is known
#end
```

Changes everything (without the confirmation code which will be sent by server to the new email):

```
#begin
login myemail@gmail.com/myPassword;
change user
     --email vicont@home.su
     --password sg388nnQwQ
     --name Victoria
     --gender F;
#end
```

claim bonus

Format:

```
claim bonus --id <transaction id>
```

Needs log in: yes

Description:

If you have made a donation, you can claim a bonus in Galaxy Credits sending this command with PayPal transaction id. If GM has already registered your donation, an appropriate sum of the Galaxy Credits will be added to your treasury. If GM has not registered your donation yet, the bonus request will be registered and converted to the Galaxy Credits as soon as the GM does the registration.

Make sure you use correct transaction id or your bonus request and GM registration will never match. The bonus request expires in 30 days after request has been done.

If you feel your bonus request is ignored, please contact the GM (see Messages to GM).

Note: the bonus is added during an upcoming turn, not immediately.

Example:

```
#begin
login myemail@gmail.com/myPassword;
claim bonus --id 939259037T1707500;
#end
```

clear stars for manager

Format:

```
clear stars for manager --name <auto manager name>
```

Parameters:

- `name:` (mandatory) automated manager name. Can be one of 'ColManager', 'DefenceManager', 'AttackManager', 'DevManager', 'ExplManager', 'RepairManager'

Needs log in: yes

Description:

Removes all stars added previously to automated manager by commands “add star for manager” or “keep current borders”

Examples:

```
#begin
login myemail@gmail.com/myPassword;

// Removes all stars assigned to ColManager
clear stars for manager --name ColManager;

#end
```

colonize

Format:

```
colonize
     --star <star id or name>
     --ship <colony ship id or name>
```

Needs log in: yes

Description:

Orders an existing colony ship to colonize a star system.

Parameters:

- `star` (mandatory): star id or name to colonize. The specified star system should be uninhabited
- `ship` (mandatory): id or name of an existing colony ship (any ship with component ‘colony unit’). The ship can be already on the orbit of the star or somewhere else. In the latter case the ship will move to the star system first.

Examples:

```
#begin
login myemail@gmail.com/myPassword;

// Order ship 12 to colonize system #234
colonize --star 234 --ship 12;
```

create ship type

Format:

```
create ship type
     --name <ship type name>
     --code <ship type code>
     [--service]
```

Needs log in: yes

Description:

Creates a new ship type.

Parameters:

- `name` (mandatory): unique ship type name up to 30 characters length (see Names).
- `code` (mandatory): specifies ship components in a format <component1>[,<component2>]...[,<componentN>]. (see Ship Design for details)
- `service` (optional): if specified, the ship is considered as a 'Service' ship. By default, all ships with weapons are 'Combat' ships.

Examples:

```
#begin
login myemail@gmail.com/myPassword;

// Create a combat ship type Fighter with drive=10, shield=5
// and one laser with power 3 (L3 = L3x1);
create ship type --name Fighter --code D10,S5,L3;

// Adding 'TransportShip' ship type with drive=10, cargo=5 and
// one laser cannon with power 1
// Note parameter 'service' explicitly states that the ship
// should be 'Service', not 'Combat'. Without this parameter,
// the ship would be created as a 'Combat' ship, because it
// has weapon (L1x1)
create ship type
     --name TransportShip
     --code D10,C5,L1x1
     --service;

// Adding another transport ship type like above
// But beware! Missed parameter 'service' – the ship will be
// treated as a 'Combat' ship.
create ship type
     --name TransportShip2
```

```
     --code D10,C5,L1x1;

// Adding another ship type 'Transport Ship 3'
// Same components as above, but without laser
// Such a type is automatically created as 'Service' ship type
// because it has no weapons
create ship type
     --name Transport Ship 3
     --code D10,C5;

// This command is equivalent to above
create ship type
     --name Transport Ship 3
     --code D10,C5
     --service;
#end
```

declare peace

Format:

```
declare peace --to <nation name>
```

Needs log in: yes

Description:

This command declares a state of Peace (see War and Peace) with a nation specified in parameter '`to`'. The 'nation name' should be a name of existing nation. If your nation is already in peace with specified nation, the command will return an error.

Examples:

```
#begin
login myemail@gmail.com/myPassword;

// Declaring a peace to nation 'Dodgy Runners'
declare peace --to Dodgy Runners;
#end
```

declare war

Format:

`declare war --to <nation name>`

Needs log in: yes

Description:

This command declares a state of War (see War and Peace) to a nation specified in parameter '`to`'. The 'nation name' should be a name of existing nation. If your nation is already in war with specified nation, the command will return an error.

Examples:

```
#begin
login myemail@gmail.com/myPassword;

// Declaring a war to nation 'Dodgy Runners'
declare war --to Dodgy Runners;
#end
```

delete bonus

Format:

```
delete bonus --id <transaction id>
```

Needs log in: yes

Description:

If you have added a bonus request and decided to remove it by whatever reason – use this command – the bonus request will be removed from the server if it has been found. One obvious reason – you have added a bonus request with wrong id. It's definitely better to remove it rather than wait till its expiration in 30 days.

Example:

```
#begin
login myemail@gmail.com/myPassword;
delete bonus --id 939259037T1707500;
#end
```

delete science

Format:

```
delete science --name <science name>
```

Needs log in: yes
Description: Removes existing science specified by parameter '`name`'.

Examples:

```
#begin
login myemail@gmail.com/myPassword;

// Removing science 'TotalDrive'
delete science --name Total Drive; ← bad, science 'Total Drive' doesn't exists
delete science --name TotalDrive; ← fine
#end
```

delete ship type

Format:

`delete ship type --name <ship type name>`

Needs log in: yes

Description: Removes existing ship type specified by parameter '`name`'. Note: you cannot delete a ship type if there is at least one existing ship of the type, or this ship type is used by an Automated Manager, or there is a production order to produce a ship with this type.

Examples:

```
#begin
login myemail@gmail.com/myPassword;

// Removing ship type 'TransportShip1'
delete ship type --name TransportShip1;
#end
```

delete trigger

Format:

```
delete trigger
     --type <trigger type name>
     --name <trigger name>
```

Needs log in: yes

Description:

Deletes a trigger of specific type and name. Fails if a trigger ‘type’/’name’ does not exist. Also fails if the trigger is attached to any object by command ‘add trigger to object’.

Parameters:

- `type:` (mandatory) trigger type
- `name:` (mandatory) trigger name

Example:

```
#begin

login myemail@gmail.com/myPassword;

delete trigger
  --type OnStarBeforeTurn
  --name ProduceDefenceShips;

#end
```

detach

Format:

```
detach
     --from <ship id or name>
```

Needs log in: yes

Description:

Detaches a ship attached before to a ship with tow unit.

Parameters:

- `from:` (mandatory) id or a name of the ship with tow unit

Example:

```
#begin
login myemail@gmail.com/myPassword;

// detaches a ship attached to Carrier
detach --from Carrier;
#end
```

dev science

Format:

```
dev science
     [--name <science name>]
     [--prop <drive tech>,<weapon tech>,<shield tech>,<cargo
tech>]
      --amount <number of PUs>]
     [--priority <order priority>]
     [--where <star name or ID>]
     [--hedge [stars:<list>],[sizes:<min..[max]>],[res:<min..
[max]>]
```

Needs log in: yes

Description:

Places a production order do develop a science specified either by parameter '`name`' or parameter '`prop`'. Only one or another parameter should be specified, not both. If parameter name is defined, it should be a name of existing science, otherwise the command will return error. Specifying parameter '`prop`' instead of '`name`' creates a production order to develop an 'anonymous' science existing only within this order. This parameter has exactly the same syntax and meaning as in command '`add science`'.

Parameter '`amount`' specifies a number of PUs you would like to be spent on developing of the science. This parameter is mandatory, must be >= 1.

Parameter '`priority`' specifies a priority of your order, '0' is default.

Parameter '`where`' defines where to place the order. The value should be either a star name or id.

If parameter '`where`' has not been specified, the order is considered as a 'common' order (see Production). The server will automatically find the most appropriate location or locations where and how to execute it. You can 'help' the server to select appropriate star systems to be selected for the science development specifying parameter '`hedge`'.

Parameter '`hedge`' can only be specified if parameter '`where`' is absent. It has the following syntax:

`[--hedge [stars:<list>],[sizes:<min..[max]>],[res:<min..[max]>]`
Hedge consists of up to 3 non mandatory parts separated by comma (you have to specify at least one):

- **stars**: specified explicit list of stars where the order has to be executed. All listed stars must be your colonies. The elements in the list should be separated by '|'. Each element can be a star id or a star name.
- **sizes**: specifies star sizes from min to max where the order has to be executed.
- **res**: specifies star resource levels from min to max (from 0 to 1) where the order has to be executed.

Why to bother with hedge? Ok, you may have many tens or hundreds of colonies and you want to develop a science placing a common order (you don't want to manually look for a suitable colony).

Why to disturb big colonies with high level of resources for something what doesn't need any resourced for development? The big colonies should produce big ships. Here, we can set a hedge like: `--hedge res:0..0.3,sizes:0..800`. Such a hedge will select only those of your colonies which have resource level <= 0.3 and sizes not bigger than 800.

Hedge is a sort of automation you may benefit when producing sciences, technologies and ships (it works for common orders only).

Obviously, the hedge parameters you define may conflict and be not realistic. You can specify a list of stars and level of resources or sizes, but there are no stars in the list with such resource levels or sizes. In this case, if, no colonies can be found satisfying the criteria, the server will place an order where it's possible. It's your responsibility to define reasonable criteria.

Examples:

```
#begin
login myemail@gmail.com/myPassword;

// Places a common order to develop an anonymous science
// investing 30000 PUs with priority 0 (default)
// (with proportions drive:10,weapon:15,shield:25,cargo:50)
dev science --prop 10,15,25,50 --amount 30000;

// Another anonymous order with priority 3 and
// with hedge specification:
// resource levels from 0 till 0.3
// sizes: up to 700
dev science
     --prop 50,15,25,5
     --amount 60000
     --priority 3
     --hedge res:..0.3,sizes:..700;

// Another anonymous order with priority 0 (default)
// assigned to colony Moon
dev science
     --prop 10,12,21,4
     --amount 10000
     --where Moon;
```

```
// Order to develop science TotalDrive with priority 5
// investing 48000PUs
// with hedge specification:
// resource levels from 0 till 0.3
// and list of stars
dev science
     --name TotalDrive
     --amount 48000
     --priority 5
     --hedge res:..0.3,stars:1|12|#45|Orion|Stella|78;

// Order to develop science Transport Hit with priority 2
// investing 10000PUs
// assigned to colony 342
dev science
     --name Transport Hit
     --amount 10000
     --priority 2
     --where 342;

// Some invalid orders:
dev science     ← bad, cannot have options 'name' and 'prop' in the same command
     --name Transport Hit
     --prop 50,15,25,5
     --amount 10000
     --priority 2
     --where 342;

dev science     ← bad, 'amount' is missing
     --prop 50,15,25,5
     --priority 2
     --where 342;

dev science     ← bad, 'where' and 'hedge' together in one command
     --prop 50,15,25,5
     --amount 10000
     --priority 2
     --hedge res:..0.5,stars:1|12|45|101|56|78
     --where 342;
#end
```

dev tech

Format:

```
dev tech
     <--name <DRIVE,SHIELD,WEAPON,CARGO>
     [<--amount <number of pu>]
     [<--tolevel <tech level>]
     [--where <star name or ID>]
     [--hedge stars:<list>, sizes:<min..[max]>, res:<min..[max]>]
     [--priority <order priority>]
```

Needs log in: yes

Description:

Places a production order to develop a specified in 'name' technology.

Parameters:

- `name` (mandatory): a technology name. Should be one of: `DRIVE,SHIELD,WEAPON` or `CARGO`
- `amount` (mandatory if `tolevel` is not specified): specifies amount of invested PUs
- `tolevel` (mandatory if `amount` is not specified): specifies a desirable level of technology (amount of necessary PUs will be calculated automatically)
- `where` (optional): defines where to place the order. The value should be either a star name or id. If parameter '`where`' has not been specified, the order is considered as a 'common' order (see Production). The server will automatically find the most appropriate location or locations where and how to execute it. You can 'help' the server to select appropriate star systems to be selected for the science development specifying parameter '`hedge`'.
- `hedge` (optional): specifies colony(s) where to develop the technology. This parameter has absolutely the same meaning as in command '`dev science`' (see there for details)
- `priority` (optional): order priority

You should always specify either '`amount`' or '`tolevel`' parameters, but not both at the same time.

Examples:

```
#begin
login myemail@gmail.com/myPassword;

// Placing an order to invest 1000 PUs for technology DRIVE
// on star #10
dev tech --name DRIVE --amount 1000 --where 10;
```

```
// Order to develop tech WEAPON to level 3 on star Solaris
// with priority 1
dev tech --name DRIVE --tolevel 3 --where Solaris --priority 1;

// 'Common' order to develop tech CARGO to level 3
dev tech --name CARGO --tolevel 3;

// Order to develop tech SHIELD with priority 5
// investing 48000PUs
// with hedge specification:
// resource levels from 0 till 0.3
// stars can be selected from the list of ids 1,12,45,101,56,78
dev tech
     --name SHIELD
     --amount 48000
     --priority 5
     --hedge res:..0.3,stars:1|12|45|101|56|78;

dev tech --name Drive --amount 1000 --where 10; ← bad, unknown tech 'Drive': should be DRIVE

dev tech --name DRIVE,WEAPON --amount 1000 --where 10; ← bad, unknown tech 'DRIVE,WEAPON': command can handle only one tech at a time

dev tech --name DRIVE --amount 1000 --tolevel 3 --where 10; ← bad, parameters 'amount' and 'tolevel are together in one command

dev tech --name DRIVE --where 10; ← bad, neither 'amount' nor 'tolevel' is specified

dev tech
     --name DRIVE
     --amount 1000
     --where 10
     --hedge res:..0.5; ← bad, both 'where' and 'hedge' options specified
#end
```

disband fleet

Format:

```
disband fleet
     --name <fleet name>
```

Needs log in: yes

Description:

Disbands specified fleet

Parameters:

- `name` (mandatory): a name of a fleet to be disbanded

All ships belonging to the specified fleet are released. The fleet itself is removed. Assignments for all ships joining the fleet are cancelled. Assignments in production orders to build ships for this fleet are removed.

This command returns error if you try to disband a fleet which is in space. It is also impossible to disband a system fleet (like SCOUT_FLEET).

Examples:

```
#begin
login myemail@gmail.com/myPassword;

disband fleet --name Fleet 7;
#end
```

execute script

Format:

```
execute script
     --code
     #{
<zx script language text>
     #}
```

Needs log in: yes

Description:

Executes a zx script as a command. Sometimes it's very handy if you need to process tens or hundreds of objects (ships or stars). Otherwise you would need to write these tens or hundreds commands manually.

Parameters:

- `code:` (mandatory) script code (see 'add trigger' for more details)

Example:

```
#begin
login myemail@gmail.com/myPassword;

execute script --code
#{
  void f(long start, long end)
  {
     for (long i = start; i <= end; ++i)
     {
        cmd cancel assignment for --fleet STAR WALL &i;
     }
  };

  f(1506, 1516);
  f(1548, 1552);
//… many more calls of f()
#}
;

#end
```

This simple script has a function f() which cancels assignments for fleets named 'STAR WALL x' (STAR WALL 1, STAR WALL 2…).
This is done to avoid copy&paste tens (may be hundreds) commands.
Output:

```
...
--Executing 'execute script --code
766F69642066286C6F6E672073746172742C206C6F6E6720656E64290D0A7B0D0A666F7220286C6F
```

```
6E672069203D2073746172743B2069203C3D20656E643B202B2B69290D0A7B0D0A636D642063616E
63656C2061737369676E6D656E7420666F72202D2D666C6565742053544152205741 4C4C2026693B
0D0A7D0D0A7D3B0D0A0D0A6628313530362C2031353136293B0D0A6628313534382C203135353229
3B0D0A6628313632352C2031363238293B0D0A6628313633302C2031363330293B0D0A6628313733
322C2031373333293B0D0A'...
Finished execution of script 'Custom Script'
Execution trace:
       : Assignment for  fleet 'STAR WALL 1506' has been cancelled successfully
       : Assignment for  fleet 'STAR WALL 1507' has been cancelled successfully
       : Assignment for  fleet 'STAR WALL 1508' has been cancelled successfully
       : Assignment for  fleet 'STAR WALL 1509' has been cancelled successfully
       : Assignment for  fleet 'STAR WALL 1510' has been cancelled successfully
       : Assignment for  fleet 'STAR WALL 1511' has been cancelled successfully
       : Assignment for  fleet 'STAR WALL 1512' has been cancelled successfully
       : Assignment for  fleet 'STAR WALL 1513' has been cancelled successfully
       : Assignment for  fleet 'STAR WALL 1514' has been cancelled successfully
       : Assignment for  fleet 'STAR WALL 1515' has been cancelled successfully
       : Assignment for  fleet 'STAR WALL 1516' has been cancelled successfully
       : Assignment for  fleet 'STAR WALL 1548' has been cancelled successfully
       : Assignment for  fleet 'STAR WALL 1549' has been cancelled successfully
       : Assignment for  fleet 'STAR WALL 1550' has been cancelled successfully
       : Assignment for  fleet 'STAR WALL 1551' has been cancelled successfully
       : Assignment for  fleet 'STAR WALL 1552' has been cancelled successfully
...
```

Actually, it’s a real example of removing a few hundreds fleets attached to guard star systems. One of the attempts to create a ‘star wall’ before the triggers were invented. (See building ‘Star Wall’ live trigger example)

force execute trigger

Format:

```
force execute trigger
     --type <trigger type name>
     --name <trigger name>
     [--id <object id>]
```

Needs log in: yes

Description:

Executes existing trigger as a command. Normally, the trigger are ‘fired’ by the server during turn processing, but it’s possible to force trigger execution using this command. It’s useful for debugging – you can run the trigger on specific object (or all objects) straight from the commands editor and see results.

Id should be used only for trigger types accepting ‘id’ like ‘OnStarBeforeTurn’. Trigger ‘BeforeTurn’, for instance, does not need it and should be ‘forced’ without id parameter.

Parameters:

- `type:` (mandatory) trigger type
- `name:` (mandatory) trigger name
- `id:` (optional) should be appropriate object id or wild card ‘*’ (meaning ‘all objects’).

Example:

```
#begin
login myemail@gmail.com/myPassword;

// Add a trigger (although it could be added before)
add trigger
  --type OnStarBeforeTurn
  --name BuildStaticDefence --code
#{
if (_star.IsOwn)
{
  branch def = _app.CalcDefenceForStar(_star.Id);
  if (def.DefenceMassToProduce > 0)
  {
    if (_star.ResourceFactor >= 0.1)
    {
      double dnum = def.DefenceMassToProduce/typeInfo.Mass;
      trace << “number of ships=” << dnum;
    } // if (_star.ResourceFactor >= 0.1)
    else
    {
      trace << "Static defence rejected: too low resources";
```

```
    } // else
  } // if (def.RequiredDefenceMass > 0)
} // if (_star.IsOwn)

#}
;

// Testing:

// TEST EXECUTE FOR ALL STARS
force execute trigger
  --type OnStarBeforeTurn
  --name BuildStaticDefence --id *;

// TEST EXECUTE FOR Solaris
force execute trigger
  --type OnStarBeforeTurn
  --name BuildStaticDefence --id Solaris;

// TEST EXECUTE BeforeTurn trigger (added before)
// (this trigger does not accept any ids)
force execute trigger
  --type BeforeTurn
  --name BuildStaticDefence;

#end
```

get report

Format:

`get report [--turn <turn number>}`

Needs log in: yes

Description:

The command requests the server to send you a report for specified turn. This makes sense if you have lost one. The '`turn number`' has to be >= the turn you have joined the game and <= 'current game turn'. Any other number is rejected with an error.

If parameter ‘turn’ is omitted, a report for the current turn is returned.

Examples:

Let's assume that the current turn is 34 and you have joined on turn 5:

```
#begin
login myemail@gmail.com/myPassword;
get report --turn 5; ← fine
get report --turn 30; ← fine
get report; ← fine returns report for the current turn
get report --turn 4; ← bad, only reports since turn 5 are available
get report --turn 35; ← bad, only reports up to turn 34 are available
#end
```

get turns schedule

Format:

```
get turns schedule
```

Needs log in: no

Description:

The command requests the server to send you a schedule for next 10 turns. This information is also available on the web page (http://galaxy.magix.net/public/TurnsSchedule.html).

Example:

```
#begin
get turns schedule;
#end
```

Server response will be something like this:

```
*** Galaxy Server v. 1.286.83.429 ***
EXECUTING ORDERS:
---Executing 'get turns schedule'...

Schedule for next 10 turns:
Turn 654: 28.07.2020 01:00:00
Turn 655: 31.07.2020 01:00:00
Turn 656: 04.08.2020 01:00:00
Turn 657: 07.08.2020 01:00:00
Turn 658: 11.08.2020 01:00:00
Turn 659: 14.08.2020 01:00:00
Turn 660: 18.08.2020 01:00:00
Turn 661: 21.08.2020 01:00:00
Turn 662: 25.08.2020 01:00:00
Turn 663: 28.08.2020 01:00:00
    ...Ok


*** TOTALLY EXECUTED 1 COMMANDS: 0 ERRORS, 1 SUCCEEDED ***
*** Orders have been processed successfully ***
```

guard

Format:

```
guard
     --ship <ship id or name>
     --fleet <fleet name>
     --star <star id or name>
```

Needs log in: yes

Description:

Assigns ship or fleet to guard a star system (“to guard” means “be on the orbit till the order is cancelled”).

Parameters:

- `ship` (mandatory, if `fleet` is not specified): id or name of a ship assigned to guard a star
- `fleet` (mandatory if `ship` is not specified): a name of a fleet assigned to guard a star
- `star` (mandatory): id or name of a star system to be guarded by ship or fleet.

You should always specify either '`ship`' or '`fleet`' parameters, but not both at the same time.

A ship or a fleet will move to the star system if it is not on the orbit already.

This assignment does not end until explicitly cancelled. It’s been designed to be used from triggers to check if a ship or fleet assigned to guard a specific star system or not.

Examples:

```
#begin
login myemail@gmail.com/myPassword;

guard --star 345 --ship 17;
guard --star Alpha --ship Victory;
guard --star 345 --fleet F VII;
...
```

join

Format:

```
join
     --ship <ship ids or name>
     --fleet <fleet name>
     --to <fleet name>
```

Needs log in: yes

Description:

Creates a new fleet or joins ships or another fleet to existing fleet.

Parameters:

- `ship` (mandatory, if `fleet` is not specified): range ids (see command “send” for detailed description of ranges, the same is hear) or name of a ship joining the target fleet (parameter '`to`')
- `fleet` (mandatory if `ship` is not specified): a name of a fleet joining the target fleet (parameter '`to`')
- `to` (mandatory for fleets, optional for ships): a name of the target fleet (can be existing fleet or or a new one). If not specified (for ships only), the ship defined in parameter '`ship`' is removed from a fleet it currently belongs to.

You should always specify either '`ship`' or '`fleet`' parameters, but not both at the same time.

If one fleet joins another, the joining fleet disappears, i.e. if existing fleet 'A' joins fleet 'B', all ships from the fleet 'A' are moved to fleet 'B'. After that, fleet 'A' doesn't exist any more. If fleet 'B' has not existed before this operation, the result of the command is effectively 'renaming' fleet 'A' to fleet 'B'.

It is not possible to join a ship already belonging to a fleet. You have to first remove ship from the its current fleet, then, join to a new fleet:

For example, let's assume the ship #10 belongs to some fleet. We want to join it to 'Fleet1'. It can be done by following commands:

```
join --ship 10; // remove ship from current fleet
join --ship 10 --to Fleet1; // join the ship to 'Fleet1'
```

Examples:

```
#begin
login myemail@gmail.com/myPassword;

// Joining 2 ships to 'Fleet1'
join --ship 10 --to Fleet1;
join --ship 11 --to Fleet1;
// Now, ships #10 and #11 belong to Fleet1
```

```
// Joining Fleet1 to Fleet2
join --fleet Fleet1 --to Fleet2;
// Now, ships #10 and #11 belong to Fleet2 and
// Fleet1 doesn't exist

// Removing ship #10 from fleet
join --ship 10;
// Now, ship #11 belongs to Fleet2;
// ship #10 doesn't belong to any fleet

join --ship 1 --fleet F1
     --to Fleet2; ← bad, parameters 'fleet' and 'ship' in one command

join --fleet F1; ← bad, parameter 'to' missed for fleet
#end
```

join game

Format:

```
join game
     --name <user name>
     --gender <M or F>
     --password <password to log in>
     --email <email where to send reports>
     --nation <nation name>
     [--hw <home world id>]
     [--id <bonus transaction id>]
     [--code <confirmation code>]
     [-e (if nation is an existing one)]
```

Needs log in: no

Description:

Send this command to the server to join the game.

Parameters:

- `name` (mandatory): your game user name not longer than 30 characters (see Names). You can use any names, event those used by others – there is no check for duplication
- `gender` (mandatory): 'M' or 'F'
- `password` (mandatory): must have length from 6 to 20 characters, only Latin letters or digits
- `email` (mandatory): your actual existing email address for receiving reports and messages sent by the game server. An email should be unique. If you repeat someone's email, the command will be rejected
- `nation` (mandatory): nation name: depending on option '-e' it should be either a valid new nation name (up to 30 characters (see Names) or an exact name of the existing nation
- `hw` (optional): id of a selected home world star system. This id can be picked by the Game Loader tool. Supplying an id of a star system which does not exist or cannot be a home world returns an error. This option is designed to be used by an automated tool like the Game Loader. Note: this option cannot be used together with ‘-e’ - when joining an existing nation.
- `id` (optional): bonus transaction id. If you have made a donation before joining the game, you may benefit for a double bonus rate in Galaxy Credits (see Donations). Specify the transaction id here to claim the bonus (see examples below). If server is not able to find the transaction, the command is rejected with an error.
- `code` (optional): confirmation code send by server to the specified email in option ‘email’ to confirm your email address. When joining the game you have to repeat the same command twice – once without the code (you don’t know it) and then again with the code sent by the server. This scheme is designed to be used by a tool like the Game Loader which does all the work for you in the background.

- '-e' (optional): if specified, a name in 'nation' should be a name of the existing nation, one of http://galaxy.magix.net/public/AvailableNations.html. If not specified, a name in 'nation' should be a valid name of not existing nation (should be absent in http://galaxy.magix.net/public/AllNations.html). Note: this option cannot be used with specified ‘hw’ (see above).

Examples:

Request to join as a brand new nation 'England':

```
#begin

join game
     --name Victoria The First
     --gender F
     --password qwertty
     --email victoria@gmx.de
     --nation England;
#end
```

If everything is Ok the server will respond the following way:

```
*** Galaxy Server v. 1.260.79.393 ***
EXECUTING ORDERS:
---Executing 'join game --name Victoria The First --gender F --password qwertty --email
victoria@gmx.de --nation England'...
Thank you for request to join the game.
Your validation code has been sent to victoria@gmx.de and valid for 10 minutes.

Please repeat a command with validation code
    ...Ok


*** TOTALLY EXECUTED 1 COMMANDS: 0 ERRORS, 1 SUCCEEDED ***
*** Orders have been processed successfully ***
```

Shortly you should receive an email (in this example dent to ‘victoria@gmx.de’) with subject “Galaxy Kings: VALIDATION CODE” and text:

```
Your Galaxy Kings validation code: 33020
```

To complete joining request – repeat the same command with the validation code:

```
#begin
join game
 --name Victoria The First
 --gender F
 --password qwertty
 --email victoria@gmx.de
 --nation England
 --code 33020;
#end
```

Request to join with existing nation 'Grand Germania':

```
#begin
join the game as
      --name Fon Zauerbach
      --gender M
      --password 12jjj3332
      --email zauerbach@deutschetelecom.de
      --nation Grand Germania
      -e;
#end
```

Example with bonus transaction id and home world id:

```
#begin
join the game as
      --name Fon Zauerbach
      --gender M
      --password 12jjj3332
      --email zauerbach@deutschetelecom.de
      --nation Grand Germania
      --id 939259037T1707500
      --hw 2112;
#end
```

join minigame

Format:

```
join minigame
     --id <mini game id>
     --name <user name>
     --gender <M or F>
     --password <password to log in>
     --email <email where to send reports>
     --nation <nation name>
     [--code <confirmation code>]
```

Needs log in: no

Description:

Send this command to the server to join existing mini game. The command is pretty similar to “join game”, but designed for mini games.

Parameters:

- `id` (mandatory): a mini game id in format “YYYYMMDD”, like “20210203”. Normally it reflects a date when the mini game was created
- `name` (mandatory): your game user name not longer than 30 characters (see Names). You can use any names, event those used by others – there is no check for duplication
- `gender` (mandatory): 'M' or 'F'
- `password` (mandatory): must have length from 6 to 20 characters, only Latin letters or digits
- `email` (mandatory): your actual existing email address for receiving reports and messages sent by the game server. An email should be unique. If you repeat someone's email, the command will be rejected
- `nation` (mandatory): nation name: it should be a valid new nation name (up to 30 characters (see Names)
- `code` (optional): confirmation code send by server to the specified email in option ‘email’ to confirm your email address. When joining the game you have to repeat the same command twice – once without the code (you don’t know it) and then again with the code sent by the server. This scheme is designed to be used by a tool like the Game Loader which does all the work for you in the background.

Examples:

Request to join a mini game 20210203:

```
#begin

join minigame
     --id 20210203
     --name Victoria The First
     --gender F
     --password qwertty
     --email victoria@gmx.de
     --nation England;
#end
```

keep current borders

Format:

```
keep current borders
```

Needs log in: yes

Description:

This command says the ColManager to set its stars (see **add star for manager** command) to all currently existing colonies and to the star systems assigned for colonization by existing colony ships.

Effectively, the command tells the ColManager “keep current colonies and don’t try to colonize anything new”. This is useful when you’d like to stop spreading for a while and maintain the order in your “house”.

The command works “incrementally”: it adds all suitable new star systems for colonization for Col Manager, not touching existing ones.

Example:

```
keep current borders; // that it!
```

load/unload

Format:

```
load/unload
     --ship <ship id or name>
     --fleet <fleet name>
     --cargo <cargo type>
     [--amount <amount>]
```

Needs log in: yes

Description:

Command '`load`' loads and command '`unload`' correspondingly unloads the cargo of specified type from/to a planet where a ship or fleet is on the orbit.

Parameters:

- `ship` (mandatory): an id or name of a ship (cannot be used together with --fleet)
- `fleet` (mandatory): a name of a fleet (cannot be used together with --ship)
- `cargo` (mandatory for command `load`, optional for `unload`): a type of cargo to load/unload. The cargo has to be one of the following:
    - **COL**: colonists
    - **POP**: population (see below)
    - **MAT**: materials
    - **IND**: industry
- `amount` (optional): specifies amount of cargo to load/unload.

The difference between COL (colonists) and POP (population) is important for load orders. If you specify 'COL' – only colonists (i.e. population exceeding the colony size) can be loaded (see Population, Colonists and Industry). If you specify 'POP', the loaded amount is limited by all available population, what is a big difference.

Example: you have a colony with size 1000 and population 1100. This makes 10 units of COL (1100 – 1000)/10 = 10. So, specifying COL you are able to load only 10 units, not more. If you specify 'POP', whole population is in the 'game' – in this case 1100/10=110 units of COL are available for loading.

This difference is very important when you use a transport command '`move cargo`' where you can order something like this:

```
move cargo --type POP --amount 200 --from 10 --to 35;
```

This command adds an order to the transport system to **move** 200 units of POP (in commands we always specify the cargo units: for COL/POP one unit is 10 of population unit), what makes 2000 units of population, from colony #10 to planet #35. If amount of COL on #10 is less than 200 units, the command will continue to move active population, trying to fulfil the ordered amount. Such command can wipe out nearly full population from colony #10 – may be it's not exactly what you

want, unless considering a kind of evacuation. Unintended moving of population would decrease the colony production or even remove the star system from the list of your colonies.

If you need to move only COL, correct the command to:

```
move cargo --type COL --amount 200 --from 10 --to 35;
```

This command will send only available COL, not exceeding 200 units, but not touching the active population.

You can't leave on a planet less than 10 units of population when you try to load POP from the planet. You can either load all available population, in this case the planet becomes uninhabited and removed from list of your colonies (note: if you did so, it would not be possible to return the colony back by just unloading the COL – you would need to send a colony ship there!), or load such amount of POP, to leave on the planet at least 10 units of population.

If parameter '`amount`' is not specified, the command will:

- when 'unload' – unload whole cargo of specified type
- when 'load' – load maximum available amount of cargo (either to the cargo bay size or how much is available on the planet)

Obviously, only the ships having cargo component can be loaded/unloaded.

Specifying --fleet instead of --ship does the same for a fleet. Command for fleet can be executed only if the fleet has at least one ship with cargo. When loading/unloading a fleet, the command tries to load/unload the different ships in a way to keep the fleet speed at the max possible level.

Examples:

```
#begin
login myemail@gmail.com/myPassword;

// loading the ship #1 by cargo MAT, amount 3.23 units
load --ship 1 --cargo MAT --amount 3.23;
// Loading 2 units of IND into the same ship
load --ship 1 --cargo IND --amount 2;
// Loading the rest of cargo bay capacity by POP
load --ship 1 --cargo POP;
// Now, ship has loaded 3.23MAT, 2IND and some POP

// Unloading cargo one unit of IND
unload --ship 1 --cargo IND --amount 1;
// Now, ship has loaded 3.23MAT, 1IND and some POP

// Unload whole MAT
unload --ship 1 --cargo MAT;
// Now, ship has loaded 1IND and some POP

// Unload the rest of the cargo
unload --ship 1;
// Now, ship is empty
```

```
// Loading fleet by 10 units of COL
load --fleet F III --cargo COL --amount 10;

// Loading fleet by all available COL
load --fleet F III --cargo COL;

// Unloading one unit of cargo (expected all ships in the fleet
// are loaded by the same type of cargo)
unload --fleet F III --amount 1;

// Unloading 3 units of MAT
unload --fleet F III --cargo MAT --amount 1;

// Unloads all cargo from all ships of a fleet
unload --fleet F III;

load --ship 1; ← bad, parameter 'cargo" missed. What to load!?
load --ship 1 --cargo Col; ← bad, unknown cargo 'Col', must be 'COL'
unload --cargo COL; ← bad, parameter 'ship' or 'fleet' missed.

unload --ship 1 --fleet F II --cargo COL; ← bad, cannot use --ship and --fleet
load --fleet F II; ← bad, cargo not specified
#end
```

login

Format:

`login <email address>/<password>`

Description:

authorises your access to the server for the 'command session'. 'Email address' and 'Password' must be the same you have specified in command '`join game`' when joined the game.

Example:

```
login myemail@gmail.com/123456;
```

move cargo

```
move cargo
     --type <cargo type>
     --amount <amount of cargo>
     --from <star>
     --to <star>
```

Needs log in: yes

Description:

Places a transport order to move a specified amount of cargo from one location to another (see Transporting Cargo).

Parameters:

- `type` (mandatory): a cargo name to be shipped. Can be following cargo names:
  - **IND**: industry
  - **MAT**: materials
  - **COL**: colonists
  - **POP**: population

  (see commands 'load/unload' for details about POP and COL)
- `amount` (mandatory): a number of units to transport
- `from` (mandatory): a star id or name of the source location, i.e. the system where the goods should be transported from
- `to` (mandatory): a star id or name of the destination star system

The transport system will automatically execute the order using the TRANSPORT_FLEET (see Transport Manager for details).

Examples:

```
#begin
login myemail@gmail.com/myPassword;

// Moving 300 industry units from star #13 to #56
move cargo --type IND --amount 300 --from 13 --to 56;

// Moving 100 COL units from star Solaris to #113
move cargo --type COL --amount 100 --from Solaris --to 113;

move cargo
     --type COL
     --amount 100
     --from Solaris; ← bad, parameter 'to' is missed
```

```
move cargo
     --type CAP ← bad, there is no such cargo 'CAP'
     --amount 100
     --from 123;

#end
```

name

Format:

`name --star/ship <ID or name of the ship or star> --as <new name>`

Needs log in: yes

Description:

This command changes a name of a star or a ship. By default, stars and ships have no names. They are identified by unique IDs. But you can assign (or change) a name for a star, if it's your colony, and you can name a ship (sometimes it's convenient to give names to big ships).

Named ships are not included in ships groups during the battles and shown separately in the Galaxy Viewer. Also, any damages or destructions of the named ships are reported to the players in the turn reports.

The names should be not longer than 30 characters (see Names for details) and be unique.

You can specify either option '`star`' or '`ship`' for what you are going to set a new name, but not both at the same time.

Examples:

```
#begin
login myemail@gmail.com/myPassword;

// naming star #1235 as 'Luna'
name --star 1235 --as Luna;
// Renaming star 'Luna' to 'Moon'
name --star Luna --as Moon;


// Naming ship #156 as BSG
name --ship 156 --as BSG;
// Renaming BSG to Battle Star Galactica
name --ship BSG --as Battle Star Galactica;

name --star Luna –ship 12 --as Something; ← bad, 'ship' and 'star' together

#end
```

nop

Format:

```
nop
```

Needs log in: no

Description:

This is a dummy 'NoOPeration' command. It does nothing. The command is useful when you would like to 'ping' the server response, or to “delete” an order saved on the server to be executed during a future turn.

Example:

```
#begin
nop;
#end
```

print trigger

Format:

```
print trigger
     --type <trigger type name>
     --name <trigger name>
```

Needs log in: yes

Description:

Traces trigger code. This command may be useful for debugging or just checking the trigger content.

Parameters:

- `type:` (mandatory) trigger type
- `name:` (mandatory) trigger name

Example:

```
#begin

login myemail@gmail.com/myPassword;

print trigger
  --type OnStarBeforeTurn
  --name ProduceDefenceShips;

#end
```

produce clones

Format:

```
produce clones
     --amount <amount>
     --where <star name>
     [--priority <order priority>]
```

Needs log in: yes

Description:

This command places a production order to produce clones (see Production).

Parameters:

- `amount` (mandatory): a number of units to produce
- `where` (mandatory): a star id or name – location where to produce
- `priority` (optional): an order priority, integer value, default = 0

Examples:

```
#begin
login myemail@gmail.com/myPassword;

// Placing an order to produce 1788 clone units
// on system Solaris with priority 4
produce clones --amount 1788 --where Solaris --priority 4;

// Placing an order to produce 100 clone units on system #15
// with default priority 0
produce clones --amount 100 --where 15;

produce clones --amount 100; ← bad, parameter 'where' is missing
produce clones --where 16; ← bad, parameter 'amount' is missing
#end
```

produce industry

Format:

```
produce industry
     --amount <amount>
     --where <star name>
     [--priority <order priority>]
```

Needs log in: yes

Description:

This command places a production order to produce industry (see Production).

Parameters:

- `amount` (mandatory): a number of units to produce
- `where` (mandatory): a star id or name – location where to produce
- `priority` (optional): an order priority, integer value, default = 0

Examples:

```
#begin
login myemail@gmail.com/myPassword;

// Placing an order to produce 1788 industry units
// on system Solaris with priority 4
produce industry --amount 1788 --where Solaris --priority 4;

// Placing an order to produce 100 industry units on system #15
// with default priority 0
produce industry --amount 100 --where 15;

produce industry --amount 100; ← bad, parameter 'where' is missing
produce industry --where 16; ← bad, parameter 'amount' is missing
#end
```

produce materials

```
produce materials
     --amount <amount>
     --where <star name>
     [--priority <order priority>]
```

Needs log in: yes

Description:

This command places a production order to produce materials (see Production).

Parameters:

- `amount` (mandatory): a number of units to produce
- `where` (mandatory): a star id or name – location where to produce
- `priority` (optional): an order priority, integer value ,default = 0

Examples:

```
#begin
login myemail@gmail.com/myPassword;

// Placing an order to produce 1788 materials units
// on system Solaris with priority 4
produce materials --amount 1788 --where Solaris --priority 4;

// Placing an order to produce 100 materials units on system #15
// with default priority 0
produce materials --amount 100 --where 15;

produce materials --amount 100; ← bad, parameter 'where' is missing
produce materials --where 16; ← bad, parameter 'amount' is missing
#end
```

produce money

```
produce money
     --amount <amount>
     --where <star name>
     [--priority <order priority>]
```

Needs log in: yes

Description:

This command places a production order to “produce” money (see Production). The money is not really produced, its reserved from spending

Parameters:

- `amount` (mandatory): a number of money units to produce
- `where` (mandatory): a star id or name – location where to produce
- `priority` (optional): an order priority, integer value ,default = 0

Examples:

```
#begin
login myemail@gmail.com/myPassword;

// Placing an order to produce 100 money units
// on system Solaris with priority 4
produce money --amount 100 --where Solaris --priority 4;

// Placing an order to produce 100 money units on system #15
// with default priority 0
produce money --amount 100 --where 15;
#end
```

produce ships

Format:

```
produce ships
     --type <existing ship type>
     [--where <location: star id or name>]
     [--hedge stars:<list>, sizes:<min..[max]>, res:<min..[max]>]
     [--number <how many ships to produce>]
     [--sendto <star: <star id> or fleet: <fleet name>>]
     [--priority <order priority>]
```

Needs log in: yes

Description:

Places a production order to build ships of existing ship type (see Production for details).
Parameters:

- `type`: (mandatory) ship type name
- `number`: (optional) number of ships to build, default 1
- `where` (optional): defines where to place the order. The value should be either a star name or id. If parameter '`where`' has not been specified, the order is considered as a 'common' order (see Production). The server will automatically find the most appropriate location or locations where and how to execute it. You can 'help' the server to select appropriate star systems to be selected for the ships production specifying parameter '`hedge`'.
- `hedge` (optional): specifies colony(s) where to build the ships. This parameter has absolutely the same meaning as in command '`dev science`' (see there for details)
- `sendto`: (optional). If specified should have one of two 'directions':
  - `star <star id or Name>`: a location where the built ships will be automatically sent
  - `fleet <Fleet Name>`: a fleet name the built ships will be ordered to join
- `priority`: (optional) order priority

Examples:

```
#begin
login myemail@gmail.com/myPassword;

// Placing an order to build 10 'Scout' ship on star system #32;
produce ships --where 32 --type Scout --number 10;

// Producing 1 'Transport Ship' ship on star system #32;
produce ships --where 32 --type Transport Ship --number 1;
// Producing 1 'Transport Ship' ship on star system #32;
// (note: '--number' by default is 1)
```

```
produce ships --where 32 --type Transport Ship;

// Producing 1 'AttackLineShip' on star system #32
// with priority 7 and sending it to star 'New Paris'
// when it is ready
produce ships
     --where 32
     --type AttackLineShip
     --sendto star: New Paris
     --priority 7;

// Producing 1 'AttackLineShip' on star system #32
// and ordering it to join fleet 'Fleet 7' when it is ready
produce ships
     --where 32
     --type AttackLineShip
     --sendto fleet: Fleet 7;

// Placing a common order to build 10 'AttackLineShip'
// and ordering the built ships to join fleet 'Fleet 7'
// (Note: no parameter 'where')
produce ships
     --type AttackLineShip
     --sendto fleet: Fleet 7;

// Placing a common order to build 100 'Fregate'
// and ordering the built ships to move to Klandatu system
// (Note: parameter 'hedge' demands the colonies selected for
// production must have size >=800 and resource factor >=0.5)
produce ships
     --type Fregate
     --sendto star: Klandatu
     --hedge res:0.5..,sizes:800..;

produce ships
     --type Fregate
     --sendto star: Klandatu
     --hedge res:0.5..,sizes:800..
     --where Solaris; ← bad, 'hedge' and 'where' in the same command

produce ships
     --sendto star: Klandatu
     --where Solaris; ← bad, 'type' is missing

#end
```

prevent bombing

```
prevent bombing --star <star id or name>
```

Needs log in: yes

Description:

Prevents any bombing by your ships the specified star system. This command can be used if you plan to capture the planet with all available industry (which would be destroyed by bombing). Please note, the bombing for this star system will be prevented forever until you cancel it by command 'allow bombing', or this star system becomes your colony

Parameters:

- `star`: (mandatory) star id or name. It must be one of the stars in your visibility area, but not one of your colonies.

Examples:

```
#begin
login myemail@gmail.com/myPassword;

// Prevent bombing the 'Solaris' star system
prevent bombing --star Solaris;
// Prevent bombing #67 star system
prevent bombing --star 67;
prevent bombing --star #67; // the same as above
#end
```

quit minigame

Format:

```
quit minigame
     --id <mini game id>
     --email <email used in join mini game>
     --password <password used in join mini game>
```

Needs log in: no

Description:

Send this command if you have decided to quit a pending mini game. You have to provide your email and password you have used when you joined the mini game. If executed successfully, your record will be immediately removed form the mini game. To quit already running mini game use “quit the game” command.

Example:

```
#begin

quit minigame
     --id 20210203 --email adolf@gmx.de –password 1234567;

#end
```

quit the game

Format:

```
quit the game
```

Needs log in: yes

Description:

Send this command if you have decided to quit the game gracefully. After accepting this command the server will set your 'Inactivity Counter' to 4 and start warning you about 'quitting' every remaining turn when sending a report (See Quitting the game for details). You can resume your game status (before the actual quitting) by sending any message or command to the server which needs a log in.

Example:

```
#begin
login myemail@gmail.com/myPassword;
quit the game;
#end
```

remove star for manager

Format:

```
remove star for manager
     --name <auto manager name>
     --id <star id or name>
```

Needs log in: yes

Description:

Removes a star from automated manager, registered before by command “add star for manager”.

Parameters:

- `name:` (mandatory) automated manager name. Can be one of 'ColManager', 'DefenceManager', 'AttackManager', 'DevManager', 'ExplManager', 'RepairManager'
- `id`: (mandatory) star id or name.

Examples:

```
#begin
login myemail@gmail.com/myPassword;

// Removes 'Solaris' from ColManager
remove star for manager --name ColManager --id Solaris;
// Removed star 67 from ExplManager
remove star for manager --name ExplManager --id #67;
```

remove trigger from object

Format:

```
remove trigger from object
     --type <trigger type name>
     --name <trigger name>
     --id <object id>
```

Needs log in: yes

Description:

Removes trigger ‘type’/’name’ from an object with ‘id’.

Note: Wild card ‘*’ used as ‘id’ means ‘all objects’. It doesn’t mix with explicit ids/names. If, for example, you added trigger to objects 1,2,3 and then removed from ‘*’, nothing would happen. Wild card ‘*’ **means** ‘all object’, but treated as a ‘special’ id.

Explanation:

add trigger to object --type OnStarBeforeTurn --name MyTrigger --id 1;

add trigger to object --type OnStarBeforeTurn --name MyTrigger --id 2;

add trigger to object --type OnStarBeforeTurn --name MyTrigger --id 3;

(at this point trigger MyTrigger has been added to 3 objects 1,2 and 3)

add trigger to object --type OnStarBeforeTurn --name MyTrigger --id *;

(at this point trigger MyTrigger has been added to 3 objects 1,2,3 and ‘all objects’)

remove trigger to object --type OnStarBeforeTurn --name MyTrigger --id *;

(at this point trigger MyTrigger has been added to 3 objects 1,2 and 3)

remove trigger to object --type OnStarBeforeTurn --name MyTrigger --id 2;

(at this point trigger MyTrigger has been added to 2 objects 1 and 3)

Parameters:

- `type:` (mandatory) trigger type
- `name:` (mandatory) trigger name.
- `id:` (mandatory) object id (or name). Wild-card ‘*’ can be used to remove trigger from “all” objects of suitable type (for trigger ‘OnStarBeforeTurn’ it would mean ‘all star systems’)

Examples:

```
#begin
login myemail@gmail.com/myPassword;

// Removes trigger ‘BuildStaticDefence’ from all star systems
remove trigger from object
  --type OnStarBeforeTurn
  --name BuildStaticDefence --id *;
```

```
// Removes trigger ‘BuildInd’ from star system 3443
remove trigger from object
  --type OnStarBeforeTurn
  --name BuildInd --id 3443;

// Removes trigger ‘BuildInd’ from star system Moon
remove trigger from object
  --type OnStarBeforeTurn
  --name BuildInd --id Moon;

#end
```

rename nation

Format:

```
rename nation --newname <new nation name>
```

Needs log in: yes

Description:

Use this command to give your nation a new name.

The new name should be a valid nation name (see Names) and it has to be a new name, i.e. not the one from the list of all nations (http://galaxy.magix.net/public/AllNations.html).

Example:

```
#begin
login myemail@gmail.com/myPassword;
rename nation --newname Grand Britania;
#end
```

rename ship type

Format:

```
rename ship type
     --name <existing ship type name>
     --newname <new ship type name>
```

Needs log in: yes

Description:

Renames existing ship type. Old ship type name is replaced by a new one everywhere (for each ship, in production orders, in the auto managers, even for nations knowing this ship type...)

Parameters:

- `name:` (mandatory) a name of existing ship type
- `newname`: (mandatory) a valid new ship type name.

Example:

```
#begin
login myemail@gmail.com/myPassword;

rename ship type --name Scout Ship --newname spy;
...
```

repair ship

```
repair ship --id <ship id or name> --priority <order priority>
```

Needs log in: yes

Description:

Places a production order to repair a ship which is on orbit of one of your colonies. This is a one-off order – it's automatically cancelled by the end of a turn. If ship hasn't been repaired at all, or repaired partly, you need to repeat the command for a next turn if you want the repair to finish.

Usually you most likely do not need to use this command, because the Repair Manager does exactly that automatically (see Repair Manager). The only motivation to use the command would be if you needed to repair a ship 'manually' when the Repair Manager is not activated (what is not recommended).

Issuing the command when the Repair Manager is activate makes no harm, most probably it would be just a duplication: one order will be executed, another – cancelled at the end of the turn.

Parameters:

- `id`: (mandatory) ship id or name
- `priority`: (optional) order priority

Examples:

```
#begin
login myemail@gmail.com/myPassword;

// Repairing ship #174 with default priority (0)
repair ship --id 174;
// Repairing ship Aurora with priority 5
repair ship --id Aurora --priority 5;
#end
```

restrict access to

```
restrict access to --star <star id or name>
```

Needs log in: yes

Description:

Adds a specified star to the 'restricted area' (See Restricted Areas). To remove star from the restricted area use command 'allow access to'

Parameters:

- `star`: (mandatory) star id or name. It must be one of your colonies – any other stars are rejected.

Examples:

```
#begin
login myemail@gmail.com/myPassword;

// Add 'Solaris' to restricted area
restrict access to --star Solaris;
// Add star 67 to restricted area
restrict access to --star 67;
#end
```

route

```
route
     --ship <ship ids range or name>
     --fleet <fleet name>
     --to <star names or ids separated by comma>
     [--loop]
     [--MaxYearsInSpace <value from 1 to 10>]
     [--DontUseOnlyOwnStars]
     [--DontAvoidAlienStars]
     [--DontAvoidEnemyStars]
     [--DontAvoidUnkownStars]
```

Needs log in: yes

Description:

In difference to command ‘send’ (see below), this command creates a route with a list of stars a ship or fleet will visit in the order specified in parameter ‘to’.

Parameters:

- `ship`: ship ids range or ship name (see command send for detailed description of ranges); cannot be used together with --fleet
- `fleet`: fleet name; cannot be used together with --ship
- `to`: (mandatory) destination star ids or names. A ship (or fleet) will start moving from current location to the first star in the list, then to the 2nd and so on.
- `loop` (optional): If specified the root will be looped forever (till assignment cancelled). I.e, after reaching the last star system in the ‘to’ list, a ship or fleet will be sent to the first star in the list to repeat the route all over again. This option is convenient to create a sort of patrols controlling a number of star systems.
- `MaxYearsInSpace`: (optional, default = 10) specifies a number of years a ship can spend in space during one 'jump'. Value must be from 1 to 10. If a destination star is far away, so ship needs to spend in the space, let's say 20 years, the server automatically splits the whole journey on smaller 'jumps' or 'legs'. Each leg can last from 1 to 10 years (depending on what is specified in `MaxYearsInSpace`)
- `DontUseOnlyOwnStars`: (optional). If specified and the whole journey consists of more than one leg, the intermediate star systems will be selected not only from own stars
- `DontAvoidAlienStars`: (optional). If specified and the whole journey consists of more than one leg, the intermediate star systems can be selected from alien (but friendly nations) stars
- `DontAvoidEnemyStars`: (optional). If specified and the whole journey consists of more than one leg, the intermediate star systems can be selected from enemy stars
- `DontAvoidUnknownStars`: (optional). If specified and the whole journey consists of more than one leg, the intermediate star systems can be selected from unknown stars

Either 'ship' or 'fleet' parameter has to be specified, but not both together. You can send ship or fleet only if they are not in space.

By default all the 'leg' options are restricted, i.e. the intermediate stars are selected from own stars only, alien, enemy and unknown stars are avoided. The 'leg' options are automatically 'relaxed' by the server, if it is not possible to find a path to the destination strictly following them. The first option to relax is `MaxYearsInSpace`. After it has reached maximum (10), using only own stars is ignored and search is repeated. If no success, the server stops avoiding alien stars, then unknown and enemy stars. If after all the 'relaxations' the path to the destination still cannot be found, the command fails.

Examples:

```
#begin
login myemail@gmail.com/myPassword;

// Routing ship #2 to Solaris star system.
// This is equivalent to the same ‘send’ command.
// The route consists of one star only
route --ship 2 --to Solaris;

// Routing ship Belfast to star systems #123, Solaris and #2
route --ship Belfast --to 123, Solaris, 2;

// Patrolling star systems #123, #34, #89 by
// fleet 'Fleet One'
route --fleet Fleet One --to 123, 34, 89 --loop;

// Routing fleet 'Fleet One' to star system #123, #89 using 'leg'
// parameters: max jump = 5 years, not own and unknown star
// systems can be used to find the path
route
     --fleet Fleet One --to 123, 89
     --MaxYearsInSpace 5
     --DontUseOnlyOwnStars
     --DontAvoidUnknownStars;

#end
```

scrap ship

```
scrap ship --id <ship id>
```

Needs log in: yes

Description:

Scrapping ship by its id (see Scrapping Ships).

Examples:

```
#begin
login myemail@gmail.com/myPassword;

// Scrapping ship #174
scrap ship --id 174;
#end
```

send

```
send
     --ship <ship ids range or name>
     --fleet <fleet name>
     --to <star name or id>
     [--MaxYearsInSpace <value from 1 to 10>]
     [--DontUseOnlyOwnStars]
     [--DontAvoidAlienStars]
     [--DontAvoidEnemyStars]
     [--DontAvoidUnkownStars]
     [-u]
```

Needs log in: yes

Description:

Sends a ship or a fleet to a star.

Parameters:

- `ship`: a name of a ship or ship ids (ranges) separated by comma. When specifying a range each ship must exist, otherwise the command will break with error. Ranges can be specified by start and end like <start>..<end>. Example: **104..126**. You can use multiple ranges and ids separated by comma. Example: **104..106, 5, 8, 105..107**. Ranges can intersect inside the command – no worries, the server will figure out properly all the ids. In the example before it would be ids: 5,8,104,105,106,107. Ship name cannot be mixed with ranges: either name or a range

  Range generator:

  There is another option to specify a range of ships – using a range generator pattern. It's a bit of a hack, but very useful. A trouble with ranges is they have to be very precisely written, i.e. an unknown ship id would generate an error. The range generator string avoids that. The syntax:

  <star id or name>:<ship type name>[:<fleet name>:[<number of ships>]]

  Example 2*34:Scout Ship::23* – will generate a range for 23 ships of type "Scout Ship"located on star 234. If there are less than 23 ships – it'll give you the maximum available. If nothing found – error is displayed.

  Fleet name is optional (leave it empty if you don't need it). Number of ships is optional – if not specified – all ships are returned.

  Only ships without assignment are selected by range generator.
- `fleet`: fleet name
- `to`: (mandatory) destination star id or name
- `MaxYearsInSpace`: (optional, default = 10) specifies a number of years a ship can spend in space during one 'jump'. Value must be from 1 to 10. If a destination star is far far away,

so ship needs to spend in the space, let's say 20 years, the server automatically splits the whole journey on smaller 'jumps' or 'legs'. Each leg normally lasts from 1 to 10 years (depending on what is specified in `MaxYearsInSpace`), but can be extended to 100 if the ship is too slow.

- `DontUseOnlyOwnStars`: (optional). If specified and the whole journey consists of more than one leg, the intermediate star systems will be selected not only from own stars
- `DontAvoidAlienStars`: (optional). If specified and the whole journey consists of more than one leg, the intermediate star systems can be selected from alien (but friendly nations) stars
- `DontAvoidEnemyStars`: (optional). If specified and the whole journey consists of more than one leg, the intermediate star systems can be selected from enemy stars
- `DontAvoidUnknownStars`: (optional). If specified and the whole journey consists of more than one leg, the intermediate star systems can be selected from unknown stars
- `u`: this option works for ships only. If specified, a ship unloads whole cargo (if available) on the destination planet.

Either 'ship' or 'fleet' parameter has to be specified. The command will fail, if the smallest possible leg is bigger than 10LY. You can send ship or fleet only if they are not in space.

By default all the 'leg' options are restricted, i.e. the intermediate stars are selected from own stars only, alien, enemy and unknown stars are avoided. The 'leg' options are automatically 'relaxed' by the server, if it is not possible to find a path to the destination strictly following them. The first option to relax is `MaxYearsInSpace`. After it has reached maximum (10), using only own stars is ignored and search is repeated. If no success, the server stops avoiding alien stars, then unknown and enemy stars. If after all the 'relaxations' the path to the destination still cannot be found, the command fails.

Examples:

```
#begin
login myemail@gmail.com/myPassword;

// Sending ship #2 to Solaris star system
send --ship 2 --to Solaris;

// Sending ships 2,3,4,5,6,7,8 to Solaris star system
send --ship 2..8 --to Solaris;

// Sending ships 1,2,3 to Solaris star system
send --ship 1,2,3 --to Solaris;

// Sending ships 1,2,3, 20,21,30,31,32 to Solaris star system
send --ship 1,2,3, 30..32, 20..21  --to Solaris;

// Sending ship 1 to Solaris star system
// (this is stupid, but finally specifies just ship 1)
send --ship 1..1, 1,1,1  --to Solaris;
```

```
// Sending ship Belfast to star system #123
send --ship Belfast --to 123;

// Sending ship Esel to star system #12 and unload whole cargo
send --ship Esel --to 12 -u;

// Sending 17 ships of type Discovery from #234 to #189
send --ship #234:Discovery::17 --to #189;

// Sending all ships of type Discovery from #234 to #189
send --ship #234:Discovery --to #189;

// Sending fleet 'Fleet One' to star system #123
send --fleet Fleet One --to 123;

// Sending fleet 'Fleet One' to star system #123 using 'leg'
// parameters: max jump = 5 years, not own and unknown star
// systems can be used to find the path
send
     --fleet Fleet One --to 123
     --MaxYearsInSpace 5
     --DontUseOnlyOwnStars
     --DontAvoidUnknownStars;

send
     --ship 23
     --fleet Fleet One  ← bad, 'ship' and 'fleet' in the same command
     --to 123;

send --ship 23;  ← bad, parameter 'to' not specified
send --to Solaris;  ← bad, parameter 'ship' or 'fleet' not specified
send --fleet 7 --to 1 -u;  ← bad, option 'u' doesn't work for fleets

send --ship 7, F1 --to 1;  ← bad, mixing range and name
send --ship F1, 7..78 --to 1;  ← bad, mixing range and name
send --ship 78..56 --to 1;  ← bad, range start > range end

#end
```

set nation flag

Format:

```
set nation flag <code string>
```

Needs log in: yes

Description:

This command allows you to change an existing national flag (see Flags). The code string consists of three parts:

```
<template name>:<background colour>[:<template colours>]
```

Colours should be specified as 6 characters strings representing 16-based RGB colours in format RRGGBB, where RR, GG and BB can be anything from 00 to FF (16 based numbers representing values from 0 to 255). For example: 00 is 0, FF is 255:

FF0000 = (255,0,0) – red

00FF00 = (0,255,0) – green

0000FF = (0,0,255) – blue

000000 = (0,0,0) – black

FFFFFF = (255,255,255) – white

To have a better idea google for 'RGB-to-Hex Conversion' or just visit one of my findings: http://www.javascripter.net/faq/rgbtohex.htm .

The code string parts:

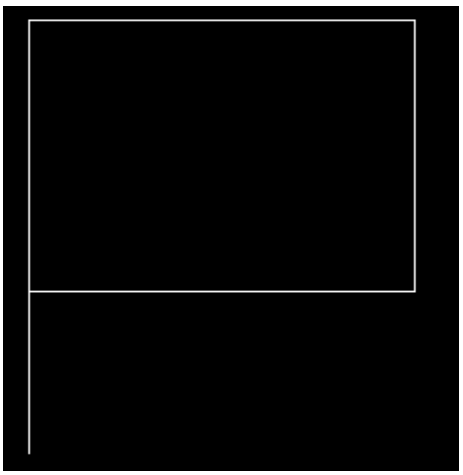
- **template name**: should be one of existing templates, supported by the server (see flags for details). It also can be empty – the latter gives you just a 'plain coloured' by the background colour flag
- **background colour**: a default colour used by the template. Any not specified template colour will be set in the background one.
- **template colours**: specify colours for template parts. The symbols of the template's parts depend on specific template.

Examples are better than any explanations.

Examples:

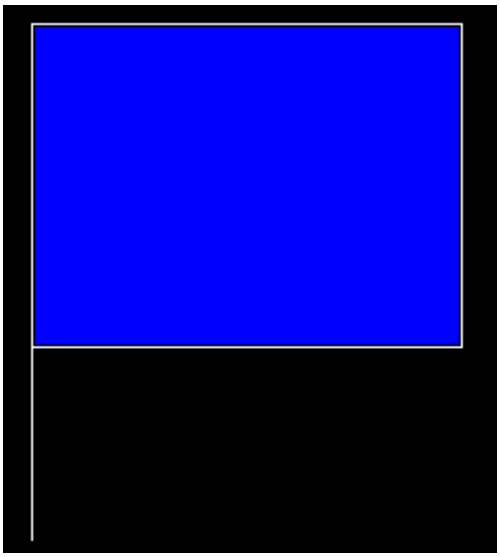
```
set nation flag :000000;
```

sets a black mono-colour flag (note – template name is missing):



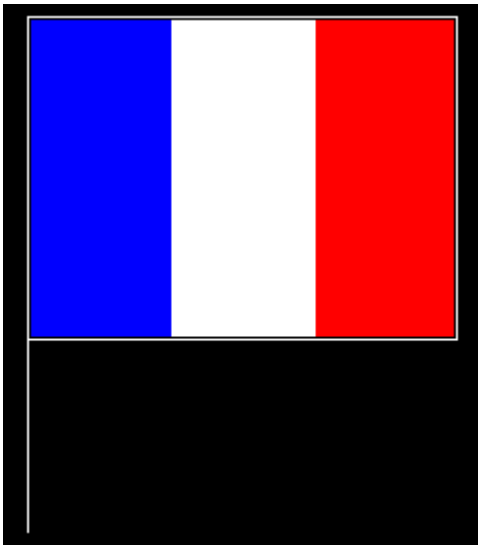
```
set nation flag :0000FF;
```

sets a blue mono-colour flag:



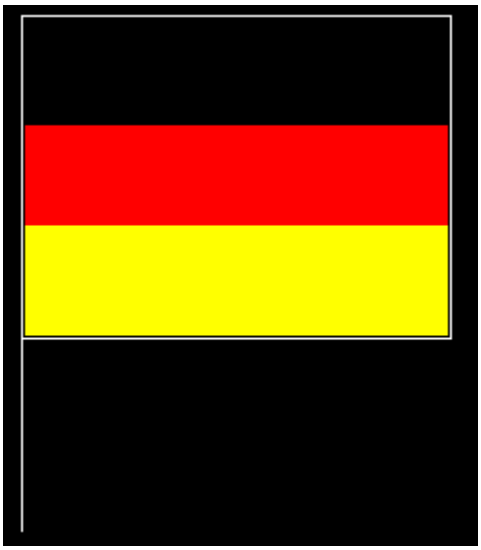
```
set nation flag 3x3:FFFFFF:1=0000FF|4=0000FF|7=0000FF|3=FF0000|
6=FF0000|9=FF0000;
```

sets a French tri-colour (note: template name is '3x3' with white background colour 'FFFFFF'; template sections '1','4' and '7' are set in blue (0000FF), sections '3','6' and '9' are set in red (FF0000). The 'middle' sections '2','5','8' are not mentioned – they simply take the background colour)



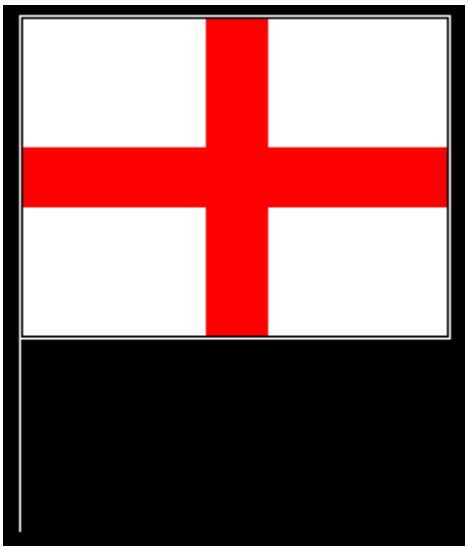
```
set nation flag 3x3:000000:4=FF0000|5=FF0000|6=FF0000|7=FFFF00|
8=FFFF00|9=FFFF00;
```

sets a German tri-colour in a similar way (note: template name is '3x3' with black background colour '000000'; template sections '4','5' and '6' are set in red (FF0000), sections '7','8' and '9' are set in yellow (FFFF00). The 'top' sections '1','2','3' are not mentioned – they simply take the background colour)



```
set nation flag cross:FFFFFF:.=FF0000;
```

creates the St.George's (English) flag (note: template name is 'cross' with white background colour 'FFFFFF'; the template section '.' (dot) which is the 'cross' is set in red 'FF0000' – fairly easy!)

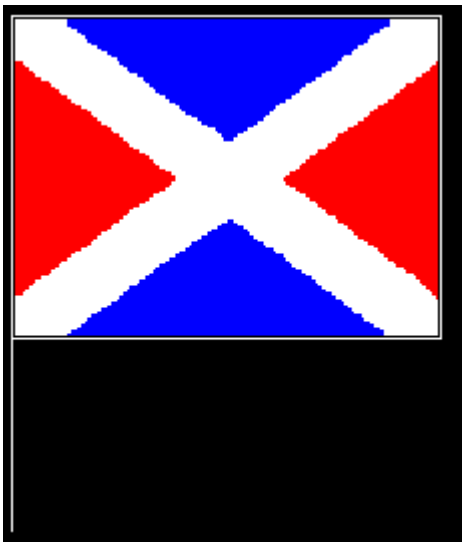


```
set nation flag scots_cross:0000FF:.=FFFFFF;
```

creates the Scottish flag (note: template name is 'scots_cross' with blue background colour '0000FF'; the template section '.' (dot) which is the 'cross' is set in white 'FFFFFF')

```
set nation flag scots_cross:FFFFFF:A=FF0000|C=FF0000|B=0000FF|
D=0000FF;
```

creates some 'custom' flag from the same template (note: template name is 'scots_cross' with white background colour 'FFFFFF'; the template sections 'A' and 'C' are red 'FF0000', sections 'B' and 'D' are blue '0000FF')



The same last command in the order:

```
#begin
login myemail@gmail.com/myPassword;
set nation flag
     scots_cross:FFFFFF:A=FF0000|C=FF0000|B=0000FF|D=0000FF;
#end
```

set player mode permanent

Needs log in: yes

Description:

Converts a temporary mini game player to the permanent one. This command is for those who wish to continue playing in Galaxy Kings after the mini game is over. Works only for players of running mini games. There is no command “set player mode temporary” - if you’d like to become a temporary player again… just use command “quit the game” after the mini game has finished!

Example:

```
#begin
login myemail@gmail.com/myPassword;

set player mode permanent;

#end
```

set prod order priority

`set prod order priority --id <order id> --priority <new priority>`

Needs log in: yes

Description:

Changes priority of an existing production order. Note: negative priorities must be put in quotes (because a dash is used as a parameter prefix)

Parameters:

- `id` (mandatory): order id
- `priority` (mandatory): new order priority

Examples:

```
#begin
login myemail@gmail.com/myPassword;

// Changing prod order priority with id 123 to -255
set prod order priority --id 123 --priority “-255”;

// Changing prod order priority with id 124 to 200
set prod order priority --id 123 --priority 200;

#end
```

set ship class

```
set ship class service for --type <ship type name>
set ship class combat for --type <ship type name>
```

Needs log in: yes

Description:

Change ship class from combat to service or from service to combat. The commands return error if a ship type already has a type to be set.

Parameters:

- `type` (mandatory): ship type name – a name of an existing ship type

Examples:

```
#begin
login myemail@gmail.com/myPassword;

// Changing ship class for type 'Esel' from 'combat' to 'service'
set ship class service for --type Esel;

// Changing ship class for type 'dk' from 'service' to 'combat'
set ship class combat for --type dk;
#end
```

show rating for

Format:

```
show rating for --all <1 or 0>
     [--rating]
     [--population]
     [--addedPopulation]
     [--lostPopulation]
     [--industry]
     [--addedIndustry]
     [--lostIndustry]
     [--shipsNumber]
     [--addedShipsNumber]
     [--lostShipsNumber]
     [--shipsMass]
     [--addedShipsMass]
     [--lostShipsMass]
     [--techDrive]
     [--techWeapon]
     [--techShield]
     [--techCargo]
     [--tech]
     [--coloniesNumber]
     [--addedColoniesNumber]
     [--lostColoniesNumber]
     [--coloniesSize]
     [--addedColoniesSize]
     [--lostColoniesSize]
     [--money]
     [--debt]
```

Needs log in: yes

Description:

This command enables or disables showing rating information about your nation for other players on the public web page (http://galaxy.magix.net/public/AllNations.html, http://galaxy.magix.net/public/AvailableNations.html). If you don't want to reveal your nation's technological levels or total population, industry, and so on to your potential enemies, you may just disable everything or selected columns. You can't disable your rating index and a year of foundation, everything else can be disabled.

Note, by default all information is published.

Options:

- **all**: 0 or 1 – specifies what to do: '0' – disable all options, or '1' – enable all options. Other options act depending on 'all' value. If all is '0', any other option enables the correspondent field, if all is '1' - any other option disables it

- **rating**: enables/disables information about rating
- **population**: enables/disables information about whole population
- **industry**: enables/disables information about whole industry
- **shipsNumber**: enables/disables information about total ships number
- **addedShipsNumber**: enables/disables information about added ships number
- **lostShipsNumber**: enables/disables information about lost ships number
- **shipsMass**: enables/disables information about total mass of all the ships
- **addedShipsMass**: enables/disables information about mass of added ships
- **lostShipsMass**: enables/disables information about mass of lost ships
- **techDrive**: enables/disables information about current level of TechDrive
- **techWeapon**: enables/disables information about current level of TechWeapon
- **techShield**: enables/disables information about current level of TechShield
- **techCargo**: enables/disables information about current level of TechCargo
- **tech**: enables/disables information about average technological level
- **coloniesNumber**: enables/disables information about total number of your colonies
- **addedColoniesNumber**: enables/disables information about added number of your colonies
- **lostColoniesNumber**: enables/disables information about lost number of your colonies
- **coloniesSize**: enables/disables information about total sizes of your colonies
- **addedColoniesSize**: enables/disables information about size of your added colonies
- **lostColoniesSize**: enables/disables information about size of your lost colonies
- **money**: enables/disables information about your available money (Treasury)
- **debt**: enables/disables information about your debt (not paid fleets payments)

Examples:

disables showing all information fields:

```
show rating for --all 0;
```

disables showing all information fields except average tech level and colonies number:

```
show rating for --all 0 --tech --coloniesNumber;
```

enables showing all information fields

```
show rating for --all 1;
```

enables showing all information fields except money, tech weapon and tech shield:

```
show rating for --all 1 --money --techWeapon --techShield;
```

Last command in the order:

```
#begin
login myemail@gmail.com/myPassword;
show rating for --all 1 --money --techWeapon --techShield;
#end
```

set ship type

```
set ship type --name <ship type name> --for <manager name>
```

Needs log in: yes

Description:

Using this command you can change the current ship type for an automated manager (see Automated Managers).

Parameters:

- `name` (mandatory): ship type name – a name of an existing ship type
- `for` (mandatory): Automated manager name. You can change the ship type only for these managers:
  - **ExplManager**: exploration manager
  - **TranspManager**: transport manager
  - **AttackManager**: attack manager
  - **DefenceManager**: defence manager
  - **ColManager**: colonization manager

  All other managers either have no ship types to set or changing the ship type is not supported.

Examples:

```
#begin
login myemail@gmail.com/myPassword;

// Setting new ship type for the Transport Manager
set ship type --name Horsa --for TranspManager;

// Setting new ship type for the Exploration Manager
set ship type --name Drone --for ExplManager;

set ship type
     --name Uragan
     --for TechManager; ← bad, not supported for Tech Manager
#end
```

tow

Format:

```
tow
     --ship <ship id or name>
     --by <ship id or name>
     --to <star id or name>
```

Needs log in: yes

Description:

Orders a ship with tow unit, specified in parameter “--by” to tow a ship, specified by parameter “--ship” to the star system, specified by parameter “--to”. If the tow ship is not in the same location as the towed ship, it will move first to the location where the towed ship is. Then it will attach the towed ship, move it to the desired location and automatically detach it after arriving.

Basically, this command does the following actions:

- *send --ship <TowShip> --to <location of TowedShip>* (only if locations are not the same)

- *attach --ship <TowedShip> --to <TowShip>*

- *send --ship <TowShip> --to <Star>*

- *detach --from <TowShip>*

Parameters:

- `ship:` (mandatory) id or a name of the ship to be towed to the destination star
- `by:` (mandatory) id or a name of the ship with tow unit
- `to:` (mandatory) id or a name of the destination star

Example:

```
#begin
login myemail@gmail.com/myPassword;

// orders ship Carrier to tow ship 23 to star system Solaris
tow --ship #23 --to Solaris --by Carrier;

// This command is equivalent of the following commands:
// if location of ship #23 and Carrier are not the same, then
// send --ship Carrier --to <location of ship #23>;
// ...wait till ship Carrier arrives...
// attach --ship #23 --to Carrier;
// send --ship Carrier --to Solaris;
// ...wait till ship Carrier arrives...
// detach --from Carrier;
```

transfer money

```
transfer money
     --amount <amount of money>
     --to <receiver nation name>
```

Needs log in: yes

Description:

Transfers a specified amount of Galaxy Credits to another nation (see Money Transfer)

Parameters:

- `amount` (mandatory): a number of Galaxy Credits to transfer (make sure the amount doesn't exceed your Treasury, otherwise the command will fail). The amount should be > 0.
- `to` (mandatory): a name of the known nation to transfer money.

Examples:

```
#begin
login myemail@gmail.com/myPassword;

// Transferring 100 CREG to Warlocks
transfer money --amount 100 --to Warlocks;

transfer money --amount 0 --to Warlocks; ← bad, amount should be > 0
transfer money --to Warlocks; ← bad, parameter 'amount' is not specified
transfer money --amount 1000; ← bad, parameter 'to' is not specified
#end
```

update trigger

Format:

```
update trigger
     --type <trigger type name>
     --name <trigger name>
     --code
     #{
<zx script language text>
     #}
```

Needs log in: yes

Description:

Updates a trigger of specific type and name. If trigger does not exist, adds it, otherwise – replaces it. This is the only difference to command ‘add trigger’ which fails, if a trigger if ‘type’ and ‘name’ already exists. See ‘add trigger’ for more detailed description about ‘code’ syntax.

Parameters:

- `type:` (mandatory) trigger type, can be one of the following types: ‘OnStarBeforeTurn’, ‘BeforeTurn’, ‘WhenRemovedFromObject’, ‘WhenCannotContinueRoute’
- `name:` (mandatory) trigger name
- `code:` (mandatory) trigger code – should be valid compilable code written in zx script language

Example:

```
#begin
login myemail@gmail.com/myPassword;

update trigger
  --type OnStarBeforeTurn
  --name ProduceDefenceShips
  --code
#{ // start code macro
    if (_star.IsReachedMax))
      cmd produce ships
        --type DefenceShip
        --where &_star.Id // using & is a way
                          // to pass variable into command
        --number 10;
#} // end code macro
; // semicolon must be on the new line after closing #}
#end
```

upgrade ship

```
upgrade ship
     --id <ship id>
     [--tech <DRIVE,SHIELD,WEAPON,CARGO>]
     [--priority <order priority>]
```

Needs log in: yes

Description:

Places a production order to upgrade a ship (see Ships Upgrade).

Parameters:

- `id` (mandatory): an id of a ship to be upgraded. The ship has to be on the orbit of one of your colonies
- `tech` (optional): a name of a technology to be upgraded. By one command, you can upgrade either one or all technologies (if tech is not specified).
- `priority` (optional): order priority

Examples:

```
#begin
login myemail@gmail.com/myPassword;

// Upgrade drive and weapons for ship #1
upgrade ship --id 1 --tech DRIVE --priority 5;
upgrade ship --id 1 --tech WEAPON --priority 4;

// Upgrade all techs for ship 2 with default priority
upgrade ship --id 2;

upgrade ship
     --id 3
     --tech SHIELD, WEAPON; ← bad, the is no tech 'SHIELD, WEAPON'
Do it this way:
upgrade ship --id 3 --tech SHIELD;
upgrade ship --id 3 --tech WEAPON;
#end
```

Quick start manual

The first thing you start with is joining the game. Send join command to the server:

```
#begin
join game
      --name Alex
      --gender M
      --password 123456
      --nation Federation
      --email my@gmail.com;
#end
```

The server responses with:

```
*** Galaxy Server v. 1.260.79.393 ***
EXECUTING ORDERS:
---Executing 'join game --name Alex --gender M --password 123456 --email my@gmail.com --nation
Federation'...
Thank you for request to join the game.
Your validation code has been sent to my@gmail.com and valid for 10 minutes.

Please repeat a command with validation code
   ...Ok
```

Shortly an email with subject “Galaxy Kings: VALIDATION CODE” will come to “my@gmail.com”

```
Your Galaxy Kings validation code: 46616
```

Send confirmation to the server with a validation code you received at address ‘my@gmail.com’:

```
#begin
join game
      --name Alex
      --gender M
      --password 123456
      --nation Federation
      --code 46616
      --email my@gmail.com;
#end
```

The server responses with:

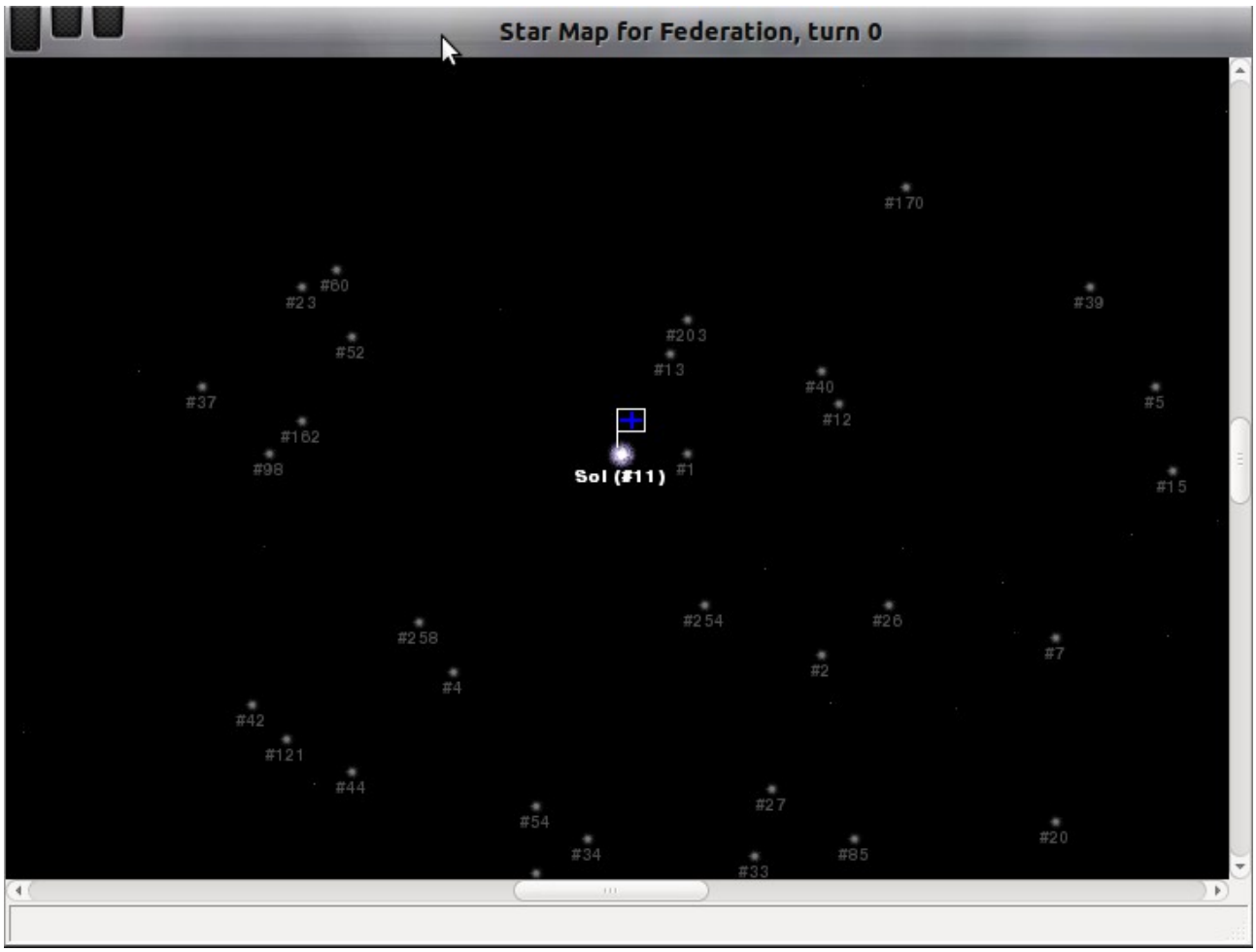
```
*** Galaxy Server v. 1.260.79.393 ***
EXECUTING ORDERS:
---Executing 'join game --name Alex --gender M --password 123456 --email
my@gmail.com --code 46616 --nation England'...
my@gmail.com has been validated successfully
* * *
Welcome to Galaxy Kings
Your first report will be for turn 1
Scheduled at 07.01.2020 01:00:00
```

```
Good luck!
Report 'Data/Reports/England__turn_00000.rep' has been generated. It'll be sent
to 'my@gmail.com'
    ...Ok
```

To see the report you need to download the Galaxy Viewer from
http://galaxy.magix.net/public/GalaxyViewer.html

(Note: here are still pictures taken from old Report Viewer, but they show relevant information)

After loading your report, you may see the star map like this:



(here on the picture the HW has already changed national flag and name what can be seen as a result of local order run in the Report Viewer).

At this point you possibly may want to do the following things:

1. Change your national flag
2. Give to your only colony (HW) some sensible name
3. Place an order to develop your HW to the maximum PP
4. Build some scout ships to research the nearby star systems

Let's change the flag:

```
#begin
login my@gmail.com/123456;
set nation flag cross:000000:.=0000FF;
#end
...

EXECUTING ADMIN COMMANDS:
---Executing 'login my@gmail.com/123456'...
Already logged in: 'Alex, the Governor of Federation'
    ...Ok

---Executing 'set nation flag cross:000000:.=0000FF'...
    ...Ok

*** TOTALLY EXECUTED 2 COMMANDS: 0 ERRORS, 2 SUCCEEDED ***
```

...and rename the colony and set up production:

```
#begin
login my@gmail.com/123456;
produce industry --where #11 --amount 703;
produce ships --type Scout Ship --where #11 --number 10 --priority 2;
name --star #11 --as Sol;
#end
```

Note, here we order to build remaining 703 industry units to reach the maximum PP (the colony size is 1703, but when you join, your HW has only 1000 units given you).

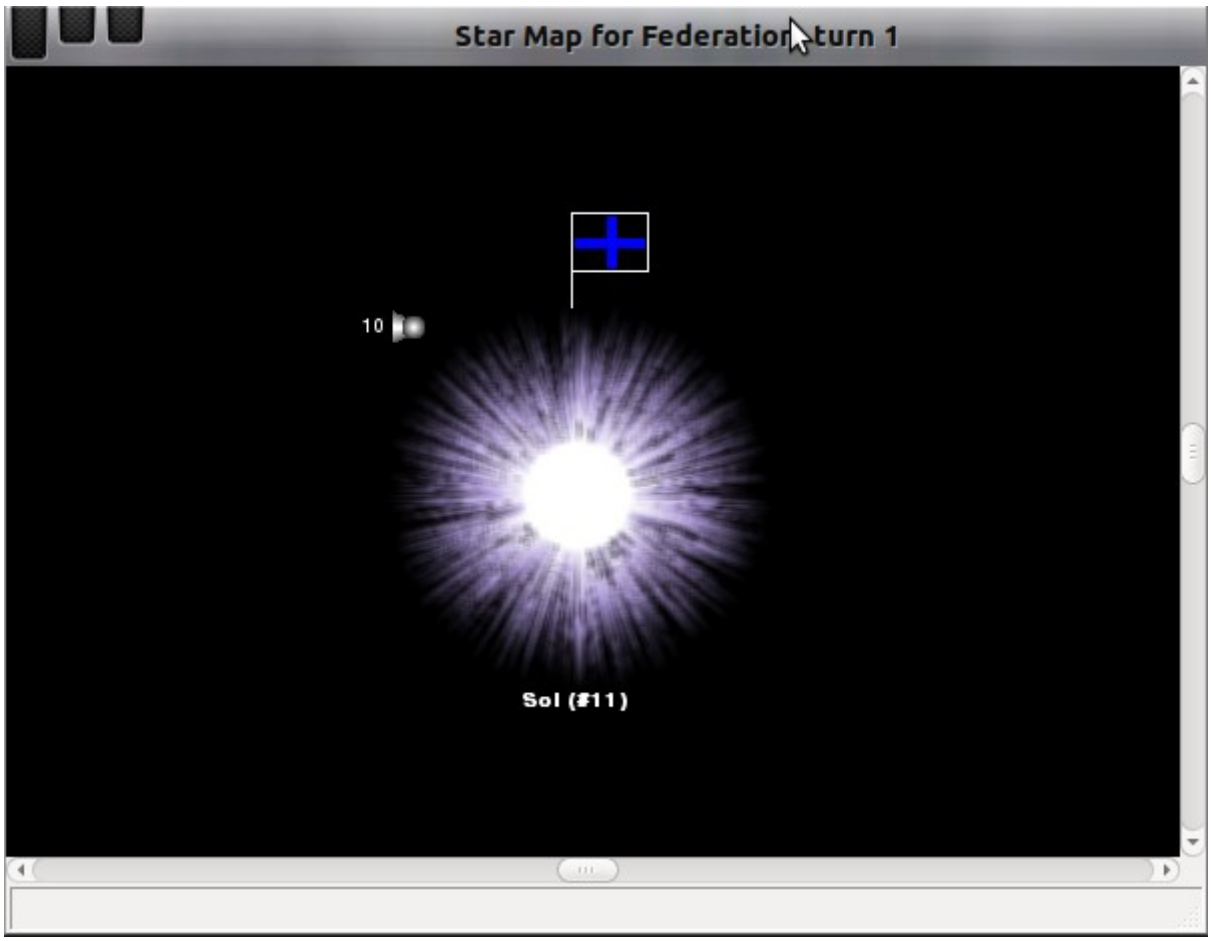
Then follows the order to produce Scout Ship's with priority 2 what is higher then default (0). This is done to make the server first to produce the ships, and then use the rest of PUs to produce industry. If you placed the ship building order with the same priority as the one for industry, your ships would be built AFTER whole industry order has finished – that's not what you want.

The last command just give your HW #11 a name 'Sol'. After the 'naming' you can refer this star system as '11' or '#11' or 'Sol'. Note: you can use '#' before ids, but this is optional.

The server response will be following:

```
...
* * *

CHECKING TURN COMMANDS:
---Executing 'login my@gmail.com/123456'...
Login has completed successfully: 'Alex, the Governor of Federation'
    ...Ok

---Executing 'produce industry --where #11 --amount 703'...
    ...Ok

---Executing 'produce ships --type Scout Ship --where #11 --number 10 --priority
2'...
    ...Ok

---Executing 'name --star #11 --as Sol'...
    ...Ok

*** TOTALLY EXECUTED 4 COMMANDS: 0 ERRORS, 4 SUCCEEDED ***
```

Let’s wait till the next turn happens and open the received report for turn 1. You would find that ordered to build the Scout Ships are ready and located on the orbit of 'Sol':



You next obvious order would be sending some (or all) the ships to nearby star system to explore them:

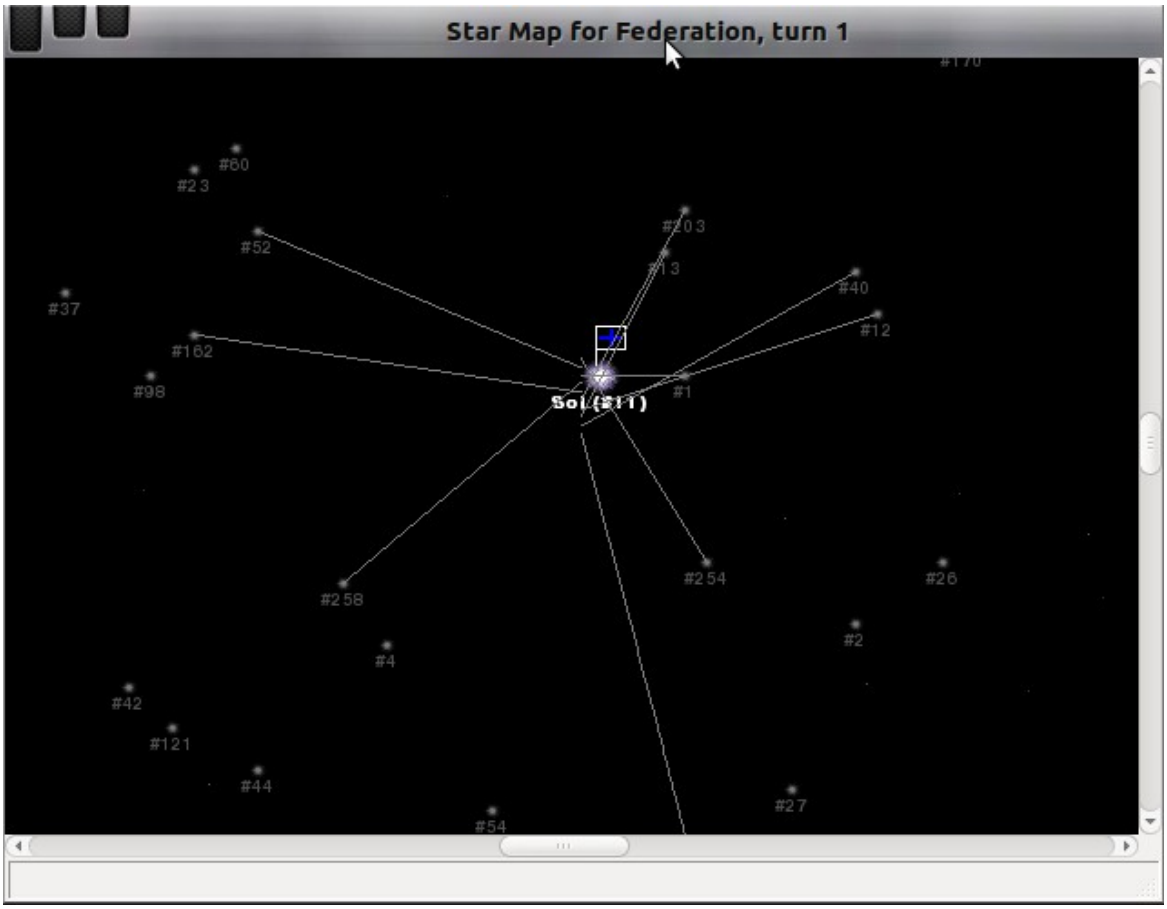
Order for turn 2:

```
#begin

login my@gmail.com/123456;

send --ship 1 --to #1;
send --ship 2 --to #13;
send --ship 3 --to #203;
send --ship 4 --to #254;
send --ship 5 --to #258;
send --ship 6 --to #52;
send --ship 7 --to #162;
send --ship 8 --to #40;
send --ship 9 --to #12;
send --ship 10 --to #14;

// New ship type
create ship type --name dk --code D1.0,L1x1@as,;
```

```
produce ships --type dk --where Sol --number 3;

#end
```

In this command I've also decided to create another 'armed' scout ship type 'dk' ('drone killer') and produce 3 of them.

After running 'dry' the order in the Report Viewer on can see the picture changes. The 'white' lines indicate directions the ships have been launched to.



Further down I'll just present the orders for the following turns with some comments without server responses. Please note – it's just one of possible scenarios how to start the game, only to give you some ideas.

Order for turn 3:

Produce a colony ship:

```
#begin

login my@gmail.com/123456;
produce ships --type ColonyShip --number 1 --where Sol --priority 1;

#end
```

Order for turn 4:

Colony ship is ready – colonize system #13. Creating new ship type 'Shuttle' and assign it to the

Transport Manager. I've decided to use bigger ship (then the default one) with some weaponry:

```
#begin
login my@gmail.com/123456;
colonize --star #13 --ship 11; // 11 is an id of the colony ship just produced
create ship type --name Shuttle --code D20,C15,L2x3@as,;
set ship type --name Shuttle --for TranspManager;
produce ships --type Shuttle --number 1 --where Sol --priority 5;
#end
```

Order for turn 5:

Not much to do here. The colony ship has already arrived to #13. The system #13 has become my colony – a very small one – just of size 70. Anyway – it's easy to make it fully developed and do some small production there in the future. The colony ship has been scrapped to create the colony. Meanwhile a transport ship is ready: load it by COL and send it to new colony. Also, in this command I order to produce industry to full colony size (70). Also, naming the new colony as 'Moon'. Ordering to 'invest' 400 PUs in TechDrive – the most valuable tech in the beginning.

```
#begin
login my@gmail.com/123456;

name --ship 11 --as Chelnok;
load --ship Chelnok --cargo COL --amount 2;
send --ship Chelnok --to #13;

produce industry --where #13 --amount 70;
name --star 13 --as Moon;
dev tech --name DRIVE --amount 400 --where Sol;
#end
```

Order for turn 6:

Just sending some ships around:

```
#begin
login my@gmail.com/123456;
send --ship 12 --to #203;
send --ship 13 --to #13;
send --ship 14 --to #12;
#end
```

Current star map looks so:

… skipping some turns...

Current turn is 10, order for turn 11:

Sending 'Chelnok' with some COL to system #13. Creating a defence orbital battle station 'Fortress' and ordering to produce it on 'Sol'. During last few turns I've discovered a number of hostile neighbours – need some defence ships to keep them away. Also, activated the Repair manager. This manager is the one you can activate from the very beginning, or at least after your first contact to other civilization.

```
#begin
login my@gmail.com/123456;
load --ship Chelnok --cargo COL;
send --ship Chelnok --to #13;
```

```
// New ship type
create ship type --name Fortress --code S30,M5x3@as/3/1/0.3,L10x2@as,L5x3@as,;
produce ships --type Fortress --where Sol --number 1;

activate RepairManager;
#end
```

When you have just a few colonies and ships it makes no sense to activate other automated managers. Later on, when you have about 10 or more colonies, you may think about activating the Exploration Manager and then the Colonization Manager, because sending all the scouts and moving goods around may become tedious. Also, it is wise to activate the Development Manager at about this stage to stop bothering about developing the new colonies.

Try to make peace with your neighbours. If some don't respond, they are possibly the AI nations – think about eliminating them. The AI nations are not very intelligent at present (but they'll become smarter and smarter, because server algorithms are being improved).

Good luck!

Technical Information

The Galaxy King server has been mostly developed on Linux environment (specifically Ubuntu/Mint). Whole code is written on C++, started with C++98, then massively used C++11/14/17 new features. The code can also be built on Visual Studio 2017.

The server is running on Linux.

The Galaxy Viewer has been done using Unity. At the moment there is a working version for Windows. Linux version should come soon

Glossary

AM Automated Managers
GM Game Master
HW Home World
LY Light-Year
PO Production Order
PP Production Potential
PU Production Unit
TM Transport Manager