Interstellar Vehicle Driving Instructions

For MagicalVehicles™ "Challenge" CH024

2007

Contents

Chapter 1

Introduction

This is the instructions manual for the interstellar vehicle ("spaceship") CH024 (dubbed "Challenge" for its ease of control or lack thereof) manufactured by MagicalVehicles^{TM} (MaVe^{TM}).

The manual is organized as follows. This short introduction to the manual itself is followed by Chapter 2. Chapter 2 is a walk through of the ship and introduces the user to the various maintenance tasks to be performed and procedures of emergency. Chapter 3 is the chapter for the seasoned user dealing with advanced tasks, giving more detailed information about certain subsystems and containing various further reference material.

As this vehicle is designed to be controlled by crew members of different background, it gives analogies for difficult to understand sections based on the user's background. Please relax now and let the manual scan your mental structure to determine the correct analogies.

The manual has detected that you are a Kwarg hunter. The box on the right gives you an example of what kinds of analogies to expect for this background.

> This is like a Kwarg hunting season. First you will have to load your harpoons with singularities. Check your suit for any leakage before descending into the molten lava. Controlling the ship's generator could be looked at as the problem we face when we have to put the chopped up Kwargs into our backpack...

Please read the analogy carefully and once finished answer the following question by clearly thinking YES or NO. Is the detected mental structure correct?

Your answer was: NO.

The manual will now perform the analysis again. For maximizing its efficiency please hold the paper close to your head[1] and relax.

The manual has detected that you are a computer programmer. The box on the right gives you an example of what kinds of analogies to expect for this background.

> This is like a programming competition. For some tasks you will receive "points" and for some tasks "trophies". You will have to collect more points than any other ship in the known universe within the lifetime of your ship to win the competition. Look out for these analogy boxes for information related to the competition itself!

Please read the analogy carefully and once finished answer the following question by clearly thinking YES or NO. Is the detected mental structure correct?

Your answer was: YES.

Thank you for calibrating this manual. If you still find that you have questions or you feel that you have found an error in the manual please consult Section 3.8 for instructions on reporting errors.

Please observe safety instructions at all times!

Good luck and have fun with your "Challenge"!

[1] or other body part with the highest concentration of mentally responsible bits

Chapter 2

Walk through

2.1 Welcome aboard

> As you can see this year's competition transports you to a virtual universe in a virtual spaceship! Follow the manual patiently and try to work out what to do. Some tasks should be obvious while others somewhat vague. In some cases you will have to experiment with this virtual reality to find out the rules by which it operates.
> Have fun exploring your ship and the great beyond!

Welcome aboard the CH024 dear customer! Since users never bother to read the instructions manual until something goes wrong this chapter of the manual is based on the assumption that something has gone wrong and will introduce you to the methods of tracking down any problems and possibly fixing them (or blowing the ship up while trying).

Step 1 Connect to the onboard computer. The computer can either be controlled through direct manipulation of its magnetic field (the usual method for magnetic and antimatter beings) or via a web interface. To connect to the web interface open http://ship/ in your browser. For a reference on navigating the web interface see Section 3.1.

Step 2 You should instantly see that every module is online. If you find that any module is offline you should try bringing it online. Unless the ship has suffered severe damage the module should come back online. If this is not the case, please see the corresponding Section in this chapter of the manual for instructions on how to repair or replace the damaged component.

Step 3 Once you have made sure that every module is online it is safe to enter the hibernation pods. Interstellar travel can take as long as 30 minutes[1]. Passengers are advised to spend this time in hibernation so as to prevent aging during space travel.

2.2 General instructions on replacing components

The components are connected to the ship via True Connecting Power, the fantastic Power upon which the great Human Empire was built thousands of years ago. This single technology has uplifted our species from the Earth we once inhabited and has allowed us to spread to the galaxy!

[1]Editor's note: In the first edition of this manual 30 years were mentioned and hibernation pods still made sense. In the current days of the 284th edition space travel has been greatly sped up, but some passengers are so accustomed to hibernation that the pods were not removed from the CH024.

Our only enemies are the heretics still believing in the Untrue Disconnecting Power, but their numbers are dwindling by the day.

Every component is assigned a port. The components are of course part of the ship and as such part of an Infinite Power network. Whenever you want to bring a component online you have to open the web interface of the ship in your browser, enter the Infinite Power network address (IP address) of the component and press the TURN ON button. Upon pressing the button the ship will initiate a TCP connection to the specified IP address and the port associated with the component.

For example to replace the control system of the generator which is associated with port 2001 you would have to create a service that listens on port 2001 and enter the IP address of the host that the service runs on through the web interface.

A complete list of component-port associations can be found in Section 3.7.

Note that since every component has a dedicated port number associated with it, several components can be hosted on a single device (with a single IP address) in the case of emergency.

Protocol

Every component communicates with the ship through a special protocol. These protocols are described in the corresponding sections. There are however a few common traits characteristic of every component protocol.

The protocols use ASCII characters.

The protocols are line-based. Every message consists of a line terminated with a line ending character (`"\n"`, the Unix line ending or ASCII character 13 – not `"\r\n"`, the Windows line ending).

The messages can usually be partitioned to words delimited by space characters.

Communication is mostly case-insensitive (except for instances where meaningful strings are being submitted).

The ship sends "keepalive" messages from time to time. These messages consist of a single line with a single word ("KEEPALIVE") and should be ignored. They are meant to make sure that dead connections do not hang indefinitely.

The protocols are designed so that a component never has to poll the ship by repeatedly sending query requests or flood it with other kinds of messages. Please take the time to design your replacement units so that they do not abuse the ship either! If you find that – lacking some kind of feedback – the best solution for you were such a continuous flood of messages, we would rather you asked the onboard artificial intelligence for advice instead of trying the flooding approach.

The protocols avoid using complex types. They use short strings (words actually) or integer and real numbers (both encoded in decimal the usual way) most of the time. Some protocols include timestamps. Timestamps are integer numbers. The timestamp of the universe is increased once every 100ms which is called in quantum science jargon a "wing-beat"[2].

Most protocol violations result in terminated connections. Some errors, that are not violations of the protocol but results of the ship's operation, are handled by sending an error message to the component. Error messages start with "ERROR:" and contain a human-readable description of the problem. Always keep a human onboard to make sure someone can read these messages!

Services

Besides providing slots for components the ship also provides a number of services over TCP/IP. Similarly to components, services are listening on a given port. Unlike components, however,

[2]As you might have already known the world has a quantum nature. Time itself shares this nature and this means that nothing happens in a continuous way, but rather in small fixed timesteps. The length of a timestep (as determined by vigorous research over the last millennium) is 100ms. Precisely the length of the wing-beat of a swallow maintaining velocity! This uncanny coincidence has led to calling this 100ms interval a "wing-beat" or "wingbeat".

services are not connected to from the ship, but from components. A number of components can connect to the same service if there is a reason to. The list of port numbers on which to find services is given in Section 3.7. The IP address is that of the ship itself. The services themselves along with the associated protocols will be described later in this chapter.

Tests

Some mission critical and more delicate components have a number of automated tests that can be performed on them. Although the components can be powered up and used without passing all tests successfully this is not recommended and may result in the loss of warranty.

You can visit the testing page for a component by taking the component offline and following the TEST link. On this page you can start various tests by clicking their RUN buttons. Once the button is pressed the ship will initiate a connection to the component just like when turning it on. This time however the component is not given live input and its output does not actually control the ship. Instead it is given a pregenerated or random input set for which it has to submit the correct solution.

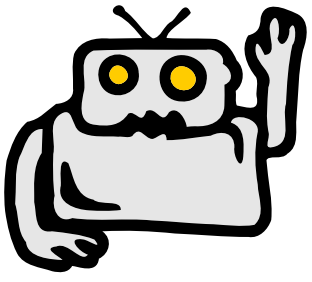
Test cases are usually similar to real-life scenarios but can be simpler or harder than they usually are. The actual goals for passing the tests are given in their description fields.

> A successfully completed test case is rewarded with 200000 points! If you manage to bring every component online and every test of the first kind successfully performed you are eligible for a "Completed Spaceship" trophy! The scoring of benchmarks is different. The results of the teams are compared, and only the top ten teams get points. Of course, the first place is worth the most points, and then the awarded points are decreasing linearly. On the testing interface the score awarded for the first place is displayed.
> Points of this nature are called *volatile* points because they may go away as soon as someone beats your score and your ranking is lowered. Volatile points are calculated separately from ordinary *permanent* points so that you can always know what is surely yours and what you may need to fight for some more.

Most tests can either fail or succeed, but there are tests which give you a numerical metric. These are benchmark tests for determining the performance of a single component in separation from other components and is a great tool for diagnosing performance problems.

2.3 Onboard Artificial Intelligence

The artificial intelligence located in a dark room by the end of corridor C can be identified from far away by its constant humming. Bad habit. You do not have to listen to it, however, when you access it through the central web interface of the ship. You can ask it questions using this interface.



The answer will not appear instantly. Sometimes it does not appear at all. When the thing works you will get the answer through the logging system with the prefix "[Artificial intelligence]". Make sure another sufficiently sophisticated AI or a sufficiently sophisticated member of the crew reads these messages! Such messages can come at any time without any warning even when you have not asked a question.

This component can not be replaced. If it does not appear to be working, you will have to travel to the nearest service planet and have a certified serviceperson examine it.

While the artificial intelligence has an enormous amount of knowledge and wisdom please be reasonable with your questions. Like all wise and intelligent beings the artificial intelligence has a short attention span and a matching temper. Be patient with it. Also note that this is the only piece of equipment onboard the CH024 that can surely not be fixed by a whack. Many have tried and failed miserably!

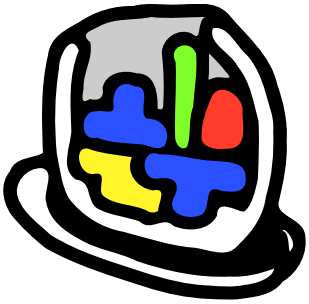
2.4 Logging

Logging on the CH024 is implemented as a service. For a general description of services see Section 2.2. The port associated with logging is 10001. Any component that connects to this port of the ship will receive new log messages and will be able to add new messages to the log.

Like all communication logging is line based – every message is one line and every line is one message. Any line that is sent to the logging service is a valid log message, they do not have to conform to any syntax. It is however suggested that they start with a string identifying its origin. For example messages from the artificial intelligence start with "[Artificial intelligence]". This way components have the possibility of processing log messages automatically and this might be a way of intercomponent communication (of course there may be much simpler solutions).

Once a line is sent to the logging service, it arrives in three places. First it is sent back to every component connected to the logging service with a timestamp placed before the message. Second it appears on the ship's web interface next time it is refreshed (along with a couple of the latest messages). And third it is stored in the log archive that is accessible from the web interface.

2.5 Generator



The generator is a central component of the ship. Users should not interfere with its workings as it is very reliable and fully automatic. This section is meant to give an impression on how wonderful it truly is.

The energetical system of the ship is made up of two components. The *generator* generates the vast amounts of energy and the *power manager* routes it to the consumer components of the ship. Because of the technology employed the generator periodically generates sizable amounts of energy instead of generating a constant stream. It is the responsibility of the power manager component to distribute this energy in a timely manner between the components that depend on it. The components using energy from the generator are the engines, the shields and the laser weapons. If any of these modules should come online but fail to perform as expected it is always a possibility that either the power manager does not route energy to the component or the generator fails to generate a sufficient amount of it.

The rest of this section is about the generator. For information about the power manager consult Section 2.6.

In the infinitely rare case that a generator would somehow stop working a replacement control system can be connected to the ship. The replacement has to accept connections on port 2001.

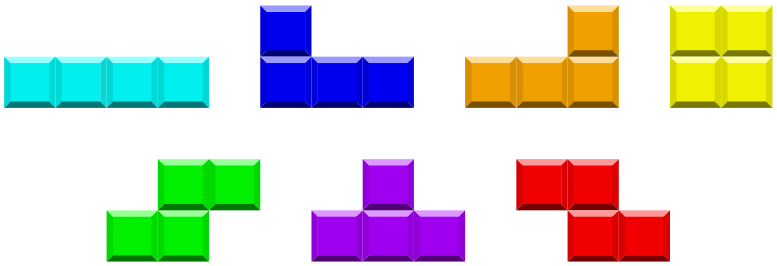
To understand the responsibilities of the generator control component a short introduction to the technology at work in the generator is necessary.

The generator is simply a hyperdimensional furnace at its core. The furnace used in CH024 has 10 dimensions numbered from 0 to 9. It has an energy layer capacity of 1000 meaning that

if more than 1000 layers of energy build up in it then the lowest layer of energy will collapse and disintegrate (without harmful effects to the ship, the crew or the environment of course).

It is useful to think about the furnace as a two-dimensional grid with 1000 rows and 10 columns. Describing its mechanics in this oversimplified way is practical for making 2-year-olds and humans understand its basics and so will be used in the rest of this section.

Exotic particles extracted from the fluctuations of the space-time can be described by their structure on the energy level-dimension grid. As the furnace is equipped with controllers that can perform various transformations on the particles they can be classified into 7 families. Members of these families can be created by applying the transformations in a certain order to other members of the family, while there is no series of transformations to transform a member of one family into a member of another family. The 7 families pictured below are denoted by the letters I, J, L and O in the top row and S, T and Z in the bottom row.



The controller of the furnace can set which transformations to apply to the particle before it is injected into the furnace. Using the two-dimensional model the transformations are a horizontal shift (left or right) and a clockwise rotation. Once injected into the furnace the particle will ascend to the highest possible energy level that it can reach while being supported by particles on lower energy levels.

The furnace generates energy when an energy level is completed (all 10 dimensions of it are filled by particles). When this happens this energy level dissolves and the energy levels above it descend one level. The amount of energy generated depends on how many levels were completed with the last injected particle. 1 level yields 40TWh, 2 levels yield 100TWh, 3 levels yield 300TWh and 4 levels (which can only be achieved by a strategically injected I family particle) yield an amazing 1200TWh!

Historical note

As an interesting sidenote it is worth mentioning that an alien race called Pajitnov has thought up a devious plan to breed the ultimate starship crew.

They came to the planet Earth and introduced the Earthlings to a game they called “TETRIS”. The inhabitants of the planet enjoyed the game because of its pretty colors and hypnotic music. Little did they know however that they were but being used as tools by the wicked Pajitnov!

The rules of the game were actually created to model the actual workings of the generator furnaces powering every spaceship in the galaxy. For millennia the Pajitnov have greatly profited from the sale of trained human slaves to spaceship manufacturers.

Please note that even though MagicalVehicles^{TM} (the manufacturer of CH024) had been a customer of the Pajitnov it is no longer the case. For the last 5 years no human slaves have been used as spaceship furnace operators (ever since the evolution of the rabbits has progressed to a point where they take up less space, require lower amounts of fodder and also play “TETRIS” better).

For those interested this is the list of differences “TETRIS” had from actual furnaces:

- The goal was to collect points as opposed to generating TWh.
- When the playing field filled up the game was over. In reality in this case a number of lines disappear from the bottom until the stack fits.

- The “playing field” is 1000 rows high in reality while the games often only had 20 lines. It is suggested that this owed to a limitation of the human mind.
- To demoralize the humans and make them work faster the Pajitnov have created the game so that the tetromino representing the incoming particle was continuously falling downward. This forced the humans to think fast (which they rarely would have done otherwise). In reality the particle is transformed before being injected into the furnace and there is all the time in the world to perform the transformations.
- Likewise the player was able to control the descent of the tetrominoes. As implied in the last point this would be impossible in reality. The Pajitnov had to be careful to only pick as slaves those humans that did all their moving of the tetrominoes once they appeared and then quickly dropped them.
- The tetrominoes representing the particles were colored and music played in the background. While those beings that are able to see exotic particles have observed them as having distinct flavors, none have described them as having distinct colors. Also there is no music in the furnace in reality.
- Touching the game had no effect on the player. Touching the furnace (or going within its 5 feet radius for that matter) instantly kills humans. Always wear protective gear when thinking about touching the furnace! And then think again! There is really no music and humans can not taste exotic particles anyway.

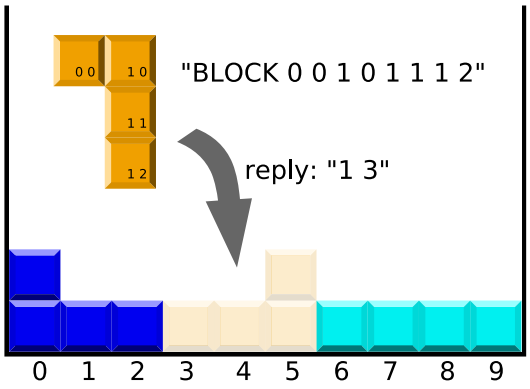
Protocol

The generator controller component should listen on port 2001. The line-based protocol used by the generator controller is based on what was described in Section 2.2. The ship sends two kinds of messages. The first kind of message is “KEEPALIVE” and should be ignored.

The format of the second kind of message is “BLOCK x_1 y_1 x_2 y_2 x_3 y_3 x_4 y_4”. This message describes an incoming tetromino. Every tetromino consists of four blocks. The tetromino is transformed so that its leftmost block has a 0 x coordinate and its topmost block has a 0 y coordinate.

As a reply to such a message the component has to send a message containing two integer numbers: “*rot pos*”. *rot* is the number of clockwise rotations to perform. *pos* is the index of the column the leftmost block of the tetromino should be in (after rotations).

Take a look at the following example:



In this case once the reply of “1 3” drops the tetromino a line is completed. This line will disappear bringing down the lines above it by one and generating 10TWh of energy.

Note that even though you can drop a tetromino instantly after receiving a “BLOCK” message a new “BLOCK” message will not arrive until 0.5 seconds later (exotic particles do not grow on

trees you know). If you send your reply within 0.5 seconds then you will have to wait for the next particle. If you take more than 0.5 seconds to ponder the placement of the tetromino you will not have to wait for the next particle. The ideal ~~slave~~ controller spends exactly 0.5 seconds to find the optimal placement. (The factory installed ~~slave~~ controller does!)

2.6 Power manager

Where does the power go once it is generated in the generator you might ask! Well it flows to the power manager. This component has not two ("ONLINE" and "OFFLINE") states but three ("ONLINE", "MECHANICAL" and "BROKEN"). It is by default set in the "MECHANICAL" state. This state is completely satisfying for the needs of the Sunday driver. For more control however you can build your own power manager component and connect it to port 2002. The third mode ("BROKEN") is not available through the web interface and it is not recommended to pursue activating this state.

In the mechanical mode the power distribution can be controlled through the web interface. In online mode the external component controls the distribution on an impulse-by-impulse scale.

Whenever an energy impulse arrives from the generator the ship sends a message to the power manager. This message is a single integer. This amount also gets added to the power manager's battery. This battery can hold at most 10000TWh of energy and any energy above this amount will get lost.

The component can send one kind of message to the ship. This message is of the format "*engines shields weapons*" and instructs the ship to send *engines*TWh of energy to the engines, *shields*TWh of energy to the shields and *weapons*TWh of energy to the weapons. If the sum of these three integer numbers is greater than the amount of energy in the power manager's battery then the command is illegal and ignored. Otherwise the given amount of energy is sent to the specified subsystems.

To enable the Sunday driver to control even a ship equipped with an advanced power manager system (such as the TURBULATOR300000000® from MagicalVehicles™) the web interface is still an option for controlling the power manager. Selecting an option on the web sends a message to the component. This message has the format "SET *engines shields weapons*". Three integer number that have the sum 100 and represent the percentages in which the energy should be distributed. This message is aimed to enhance the user experience and can be freely ignored by the component. (But power manager systems manufactured by MagicalVehicles™ all handle these messages as can be expected!)

The mechanical mode is equivalent to a component that respects the "SET" messages and immediately responds to incoming energy impulses by distributing them according to the set ratio. Thus it makes minimal use of the battery. (Note that the mechanical mode is not actually implemented by a TCP connected component and as such is a very reliable part of the ship.)

2.7 Panel manager

It is time to tell you the truth. Space. Is. Two. Dimensional.

Sorry about that. But so are spaceships too so it is not a problem at all. In fact, navigation has become a lot simpler since scientists have discovered this! So has spaceship construction! There is no need anymore to spend huge amounts of resources on building three-dimensional hulls. There are now a convenient three degrees of freedom instead of the dreaded six. You will not get motion sickness from the spaceship twisting and tumbling around three orthogonal axles. You do not have to be afraid of missing the orbit of a planet because of approaching in an angle to the ecliptic. Life is simple. Life is good.

The ship is made up of square-shaped panels. These panels are of the same size and arranged on a grid. This grid can be used to describe the panel structure of the ship in a coordinate system. Any which way the ship is rotated in the space, the coordinate system rotates with it. If there

was a panel "above" or "to the left" of some panel it will still be considered "above" or "to the left" of the panel after the ship has been rotated 180°.

Two panels are connected if they are touching with their edges (not just their corners). There is a special panel at coordinates $(0, 0)$ that contains the living quarters of the ship. While the purpose of the panel manager is to allow the restructuring of the ship by changing the types of panels, creating new panels and destroying panels, it does not have power over the special panel at $(0, 0)$. Rest assured, dear customer, you are in a safe place!

The type of a panel can be any one of the following:

- EMPTY
- BUILDER
- THRUSTER_UP
- THRUSTER_DOWN
- THRUSTER_LEFT
- THRUSTER_RIGHT
- LASER_UP
- LASER_DOWN
- LASER_LEFT
- LASER_RIGHT

There is an infinite number of EMPTY panels – they are not panels at all but places on the grid that are *not* occupied by panels. The special panel at $(0, 0)$ is of the type BUILDER and can not be changed.

It is panels of type BUILDER that the panel manager component can control. The panel manager component can output a panel restructuring program to the controller subsystem. This program consists of a series of instructions.

Panel designer

The panel designer is the friendly face of the panel management system that the user is normally going to encounter. This interface (available through the web) can be used to review the current architecture of the ship and to perform modifications. Designing a new panel structure is quite a straightforward process. You can use the buttons at the top of the interface to expand the area of the editor. Clicking on panels in the editor changes their types. Clicking cycles through the available panel types in one direction. If you need to go through the panel types in the other direction hold SHIFT while clicking.

Once you are ready with the design process click the "Build this configuration" button. This button forwards your design to the panel manager component (if online) so that it can generate a panel restructuring program for it. If you have not yet started designing a new panel configuration (that is the area has not been expanded and no panels have been changed) the interface will periodically update to reflect the actual structure of the ship. This means that you can sit back an relax watching the panel designer interface update as your grand design slowly takes form. Note that it takes one minute to change the type of a single panel, so you might as well want to hibernate for the duration of those more involving redesigns! The "Reset configuration" button cancels the current editing and returns to displaying the current configuration and automatically updating.

Panel reporter

The panel manipulating framework provides a service on port 10004. Connect to this service to keep updated about any change to the panel structure. This service is useful for quick repair of panels destroyed by the occasional near-lightspeed asteroids.

Whenever the panel structure changes (either through the actions of BUILDER panels or due to environmental effects) every subscriber of this service is sent a multiline message. The message starts with the line "CURRENTLY N PANELS" and continues with N lines of the format "PANEL $type_i$ AT x_i y_i" These lines describe the structure of the ship *after* the change.

Panel restructuring programs

It is panels of type BUILDER that the panel manager component can control. The panel manager component can output a panel restructuring program to the controller subsystem. This program consists of a series of instructions. These instructions are acted upon instantly. None of these instructions affect the actual structure of the ship. Instead they act on *command queues*.

Command queues are FIFO[3] storage units. Every node in the panel grid has its own command queue, even those nodes that do not have BUILDER panels and those nodes that are EMPTY. BUILDER nodes take the oldest command from the queue at their position and perform it. They can take and perform 1 command every minute and they act simultaneously. Two BUILDER nodes can even destroy each other! Other kinds of nodes leave the command queues alone. This means that you can assign commands to an EMPTY position, and later build a BUILDER at that position and this BUILDER will start processing the commands. Once another BUILDER destroys the BUILDER that still has commands in its queue the processing will stop. It will be resumed however when and if a new BUILDER is built at that position again.

The controller subsystem recognizes the following instructions:

LET x_b y_b **CHANGE** x_t y_t **TO** *type*: Adds a building command to the command queue at panel coordinates (x_b, y_b). When this command is executed it changes the panel at (x_t, y_t) to type *type*. The command is invalid (and terminates the connection) if the target is not next to the target (i.e. $|x_b - x_t| + |y_b - y_t| \neq 1$). *type* can be any one of the types listed at the start of the section. Use EMPTY as *type* to remove a panel.

LET x_b y_b **WAIT** *time*: Adds *time* wait commands to the command queue at panel coordinates (x_b, y_b). *time* has to be a non-negative integer or the connection will be terminated.

RESET x_b y_b: Clears the command queue at panel coordinates (x_b, y_b).

RESET: Clears every command queue. Useful at the start of a panel restructuring program.

SUSPEND: Suspends the execution of command queues. Useful at the start of a panel restructuring program.

RESUME: Resumes the execution of command queues. Place at the end of a panel restructuring program if SUSPEND was used at its beginning.

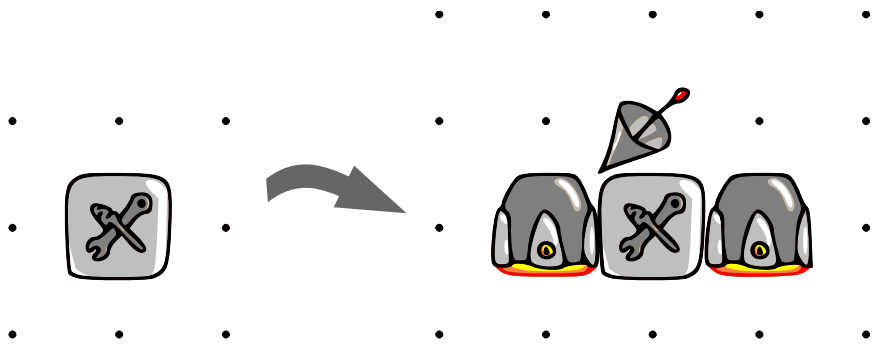
The panel manager component has to send programs made up of these instructions to the ship to restructure it.

While it is completely unnecessary to include this description in the manual (since factory installed panel manager components are very reliable and have almost never been known to fail), it is required by intergalactic regulations. To further comply with these regulations the following example is provided.

Let us assume that the ship was reduced to a single BUILDER unit (located at coordinates $(0, 0)$) during an exceptionally vehement meteor storm last night. Now let us add two thrusters and a laser cannon to the ship. We can achieve it with the following program:

[3]Ilwrathi abbreviation of "Fai Ingilgeneolam Felkathi Orx", which loosely translates to "the being which eats the head of the other being while its tail is still writhing in agony".

```
RESET
SUSPEND
LET 0 0 CHANGE -1 0 TO THRUSTER_DOWN
LET 0 0 CHANGE 1 0 TO THRUSTER_DOWN
LET 0 0 CHANGE 0 -1 TO LASER_UPRIGHT
RESUME
```



The RESET command is important to clear the command queues. The SUSPEND/RESUME commands are recommended in general but are mostly needed when there is a number of WAIT commands that are used to synchronize the operation of BUILDER panels.

Protocol

The protocol of the panel manager component is the following.

The component can send panel restructuring programs to the ship at any time. Each instruction must be on a separate line. These instructions are all the output that the panel manager component has.

The input for the panel manager component comes from the panel designer. Once a user has finished designing the ship of their dreams and they press the "Build this configuration" button a multiline message is sent to the panel manager component. The format of this message is similar to the panel reporter messages.

The first line of the message is "CURRENTLY N PANELS" and it continues with N lines of the format "PANEL $type_i$ AT x_i y_i" describing the current structure of the ship. The next line has the format "TARGET M PANELS" and is followed by M lines of the format "PANEL $type_i$ AT x_i y_i" describing the configuration designed by the user.

For example the above illustrated example would yield the following lines if inputted through the web interface:

```
CURRENTLY 1 PANELS
PANEL BUILDER AT 0 0
TARGET 4 PANELS
PANEL THRUSTER_DOWN AT -1 0
PANEL LASER_UPRIGHT AT 0 -1
PANEL BUILDER AT 0 0
PANEL THRUSTER_DOWN AT 1 0
```

Some third-party panel manager components ignore their input completely and provide their own user interfaces (often designed for members of exotic species) for designing the panel structure. The specification allows for this approach, but such components have to be clearly labeled as "not for the faint of heart" on their boxes and any marketing and promotional materials.

The panel manager component is also welcome to connect to the panel reporter service for getting feedback on the build process and to be able to respond to accidental destruction of panels.

2.8 Engines

Once the merry sounds of the generator fill the air of the ship and the vacuum of the space and you have checked the panel designer to make sure that there are thrusters on the ship, you are ready to go! The engine controller component can navigate the ship by setting the throttle on each thruster independently. The factory installed engine controller is a very user friendly piece of technology. You can navigate the ship to distant planets by showing photographs of the planet or direct it by simple voice commands, such as "Faster! Faster!".

Should the default controller not satisfy you, it can easily be replaced. The replacement controller has to accept connections on port 2003 and should send commands of the format "ENGINE *x y percentage time*". This command controls the thruster panel at coordinates (x, y), and sets its throttle to *percentage* (an integer number not less than 0 and not greater than 100) at *time*. *time* can be omitted in which case the throttle is immediately set to *percentage*.

The engine system has a volatile battery that can store an indefinite amount of energy, but if charged above a level the energy loss through interdimensional fatigue becomes a factor. The amount of energy lost per swallow wing-beat (100ms) is a factor of the amount of energy currently stored (E) and can be calculated as $w = \lfloor (E - 1) * 0.005 \rfloor$.

The consumption of one engine at full throttle is 1TWh/wing-beat. If the neccessary amount of energy is not available in the engine battery, then the available energy is shared among the engines in the ratio of their throttles. The pushing force of an engine is proportional to the energy it uses.

2.9 Radar

The radar is the long range surveillance device of the ship. The CH024 is fitted with a 10km-certified monoobjecticular radarcone. This means that the radar can detect objects within a radius of 10km, but can only yield detailed results if there is a single object within its cone. The width and direction of the cone are parameters controlled by the radar control component.

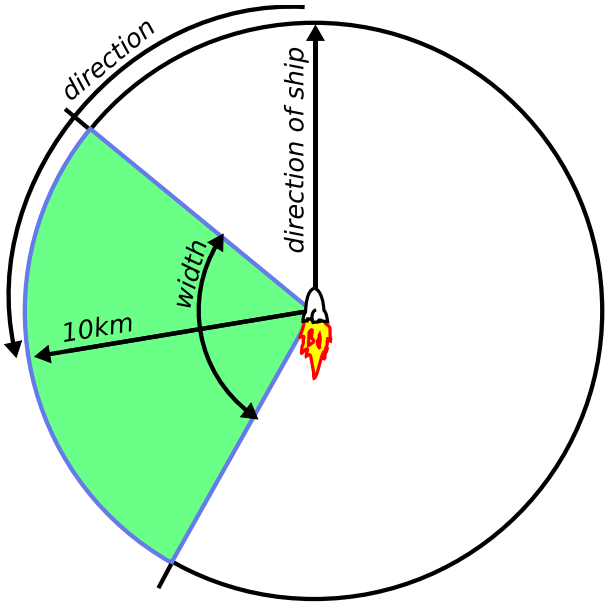
The radar is equipped with its own power source, but it needs a bit of time the get fully charged. To ensure a proper supply of power the control equipment only enables the radar to project a cone once every second.

Protocol

The control component has to accept connections on port 2006.

To project a radar cone the radar control component has to send a message of the following format to the ship: "SCAN *direction width* [AT *time*]". Both *direction* and *width* are angles measured in radians. $direction = 0$ is straight ahead (the "up" or $(0, -1)$ direction of the panel grid) and $direction = \pi/2$ is to the left (the "left" or $(-1, 0)$ direction of the panel grid).

The area scanned by the radar is the green area in the following illustration:



direction is considered as if modulo 2π – it can be negative or greater than 2π in absolute value without problems. *width* can be greater than 2π too. This would have the same effect as the value of 2π and would scan the entire area within 10km of the ship.

The *time* argument (along with the preceding "AT" preposition) is optional. If it is present, the cone will be projected at the specified time. There are two constraints for the values of *time*. First it must be in the future. Second it must be a multiple of 10 so as to ensure that the power source has enough time to charge between shots. The *time* argument is used to synchronize the radar operation with the weapons and navigation systems. If the *time* argument is not present, the radar will fire as soon as the ship clock reaches the next multiple of 10.

If two or more radar control instructions are given for the same moment in time (either explicitly through the *time* argument or implicitly by omitting it), the last instruction takes precedence.

Whenever a radar projection is performed, the radar control component is sent a message about the results of the scan. If there were no objects within the scanned area, the message is "NO OBJECTS DETECTED AT *time*" (*time* being the timestamp of the scan). If there were more than 1 objects the message is "MULTIPLE OBJECTS DETECTED AT *time*". If there was exactly 1 object detected, the message is "ONE OBJECT DETECTED AT *time* WITH PARAMETERS δ_x δ_y *direction* v_x v_y *spin type name*". (δ_x, δ_y) is the position of the detected object relative to the ship ($\delta = position_{object} - position_{ship}$). *direction* is the direction the object is facing in radians. (v_x, v_y) is the velocity vector of the object. *spin* is the derivative of the direction of the object. *type* is a string identifying the type of the object (there is no exhaustive list of object types). *name* is the name of the object if applicable (for example for spaceships and planets). *name* is not present for objects without a name (torpedoes and stars for example).

2.10 Lasers

The CH024 can be outfitted with laser weapons. They are mostly used for entertainment purposes, since the vivid flashing colors can turn even the most uneventful journey into an exciting party. Also galactic regulations forbid their use against sentient beings[4]. You should note though that there are some distance places in the universe where some malignant ships might not respect these regulations. But fortunately in the 1500 meter radius neighborhood of the bank planet this rule

[4]Shortly after this regulation went into effect all "sentience detectors" were recalled from civilian vehicles by the company. This action was cited as being in line with the "freedom of killing" amendment and helped the company achieve a 90% market share.

is strongly enforced. If a laser beam hits a ship in this area, the strong alien fighter fleet of the planet makes sure that the shooter will be severely punished.

For a higher sense of immersion the laser cannons are not remote controlled by the factory installed weaponry controller component. Instead the trigger happy user can jump into a pressure suit and make his or her way to the laser cannon to control it manually. The manual controls are easy to learn yet fun. The user has to press the red button when it lights up to shoot. If they do not press it to the "rhythm of the battle", they lose perfectness points.

The rest of the instructions in this section only apply to those who want to replace the default weaponry controller.

> If you manage to use your laser to destroy the ship of another team or an alien fighter for the first time, you are awarded 200000 point. For the second kill of the same ship you get only 100000, and for the third only 50000 points. After the third time, you will not get any point for the same ship. Also note that any treasure on board of the destroyed ship is automatically moved to your ship. See Section 2.14 for details.

To ensure the safety of your crew and those sharing the space with you make sure your replacement controller does not accidentally direct the laser weapons towards sentient beings.

The effective range of a laser beam is 5km. Laser beams travel with the speed of light: 500m/s. Due to the cannons having been developed by a team of Einstein-clones the speed of the ship is not added to the speed of the beam.

Protocol

The weaponry controller component has to listen on port 2007. It handles both the lasers and the torpedoes.

The weapons systems get their energy from the generator through the power manager. The weapons system has a volatile battery that can store an indefinite amount of energy, but if charged above a level the energy loss through interdimensional fatigue becomes a factor. The amount of energy lost per swallow wing-beat (100ms) is a factor of the amount of energy currently stored (E) and can be calculated as $w = \lfloor (E-1) * 0.01 \rfloor$.

Whenever an energy impulse arrives from the power manager, the weaponry controller is sent a message of the format "ENERGY: E", where E is the amount of energy in the battery after the impulse was added but before any amount of energy had been lost to fatigue.

The command to fire the lasers is "LASER p_x p_y *direction* [*program*] [AT *time*]". This command activates the laser panel with panel coordinates (p_x, p_y). The laser panels can only turn in a 90° range and so $-\pi/4 \leq direction \leq \pi/4$ must hold (*direction* is given in radians). *program* is the payload of the laser beam and will be executed in the shield-space of whatever gets hit by the laser. Laser microprogramming is discussed later in Section 2.13, because some knowledge of shield systems is inevitably necessary to understand them. If the *program* is omitted, an empty string is assumed as the microprogram. *time* is the timestamp indicating the time at which the laser should be fired. *time* has to be a time in the future. If not present the laser will be immediately fired.

When either a timed or an immediate fire order is processed, first the panel system is checked to ensure that there is actually a laser panel at (p_x, p_y). If there is then the volatile battery of the weapons system is checked. The laser beams contain high-energy photons. In fact these photons each bear 10TWh of energy. Because of the quantum nature of light there are no fractional photons, and the minimum amount of energy needed to fire the laser is 10TWh. This results in an energy beam containing a single photon. If there is more energy available, a higher number of photons will be generated. Due to safety reasons however the maximum number of photons in a single beam is 10. This means a maximal energy consumption of 100TWh per firing. If the panel at the specified coordinates is not a laser panel or there is not enough energy for firing, an error message is sent. Otherwise the laser is fired without further messages.

Besides the energy level messages and error messages there is one further kind of message sent to the weaponry controller. This message is of the format "LASER SHOT AT *time* HIT TARGET! SCATTER SPECTRUM IS *spectrum*". This message is sent if a laser has struck a target and is useful for fine-tuning its payload. More about scatter spectra can be found in Section 2.12.

2.11 Torpedoes

The CH024 is equipped with a Devastationalizer^{TM} ion torpedo generator and launcher. This device is so very dangerous that the CH024 does not even provide a way to fire it. If you feel that you definitely need the torpedo activation system found in the military grade CH026 and CH027 Mk II models, you can either upgrade to one of them or buy a torpedo controller separately.

The rest of this section describes the very dangerous and unworthwhile steps of building a torpedo controller on your own for the hobbyists who have grown to love the extensibility of the CH024 as a weapons platform.

Protocol

Torpedoes are launched by the weaponry controller that has control over both the lasers and the torpedoes. It has to listen on port 2007.

A torpedo takes 5 minutes to be generated and loaded into the torpedo bay. Once it is loaded the controller is sent the information message "TORPEDO LOADED". Loading is fully automatic, but only happens when the controller is online and for safety reasons torpedoes are unloaded when the controller goes offline.

Torpedoes can be fired with the "TORPEDO *target* X_0 A B C D" message. The *target* argument identifies the ship the torpedo should lock onto. It is a string and may contain space characters if it is bounded by quotation marks (`"`). The target can be anywhere in the universe thanks to the highly advanced locking system of the Devastationalizer^{TM}. The rest of the arguments describe the control system to be loaded onto the torpedo before launching. Error messages will be generated and the torpedo will not be fired if the torpedo has not yet been loaded or if *target* is not a valid name. Other violations of the protocol result in disconnections.

The torpedo is controlled by a MISO^{TM} linear control system. It has two inputs: u_1 is the distance of the target and u_2 is the difference between the direction of the target and the orientation of the torpedo. Together they form the vector U. The control system can have any number of state variables and they are denoted by the vector X of length N. The X_0 argument gives the initial state of the system and the rest of the arguments describe its evolution:

$$\begin{aligned} X_{n+1} &= AX_n + BU_n \\ y_n &= CX_n + DU_n \end{aligned}$$

y is the scalar output of the system. It controls both the turning and the acceleration of the torpedo. Its effects can be described with the following equations:

$$\begin{aligned} \omega_{n+1} &= y_n/5 \\ a_{n+1} &= 5|y_n| \end{aligned}$$

ω stands for the angular velocity of the torpedo and a for the acceleration. There is some electromagnetic friction occurring while the ion torpedo moves in an electromagnetic field (such as the space) and this causes a continuous slowing down of the projectile. Make sure that the acceleration is not zero or the torpedo will slowly come to a halt.

X_0 is a vector of length N, A is an $N \times N$ matrix, B is an $N \times 2$ matrix, C is a vector of length N and D is a vector of length 2. The matrices in the state space description of the controller

have to be given in the launching message in row-major order (row after row). A single space character comes between the numbers given in the usual decimal form (they are not constrained to integers). The following example should make all this very clear.

Example for $N = 3$:

$$\begin{pmatrix} x_{1,n+1} \\ x_{2,n+1} \\ x_{3,n+1} \end{pmatrix} = \begin{pmatrix} a_{1,1} & a_{1,2} & a_{1,3} \\ a_{2,1} & a_{2,2} & a_{2,3} \\ a_{3,1} & a_{3,2} & a_{3,3} \end{pmatrix} \cdot \begin{pmatrix} x_{1,n} \\ x_{2,n} \\ x_{3,n} \end{pmatrix} + \begin{pmatrix} b_{1,1} & b_{1,2} \\ b_{2,1} & b_{2,2} \\ b_{3,1} & b_{3,2} \end{pmatrix} \cdot \begin{pmatrix} u_{1,n} \\ u_{2,n} \end{pmatrix}$$

$$y_n = \begin{pmatrix} c_1 & c_2 & c_3 \end{pmatrix} \cdot \begin{pmatrix} x_{1,n} \\ x_{2,n} \\ x_{3,n} \end{pmatrix} + \begin{pmatrix} d_1 & d_2 \end{pmatrix} \cdot \begin{pmatrix} u_{1,n} \\ u_{2,n} \end{pmatrix}$$

Message sent: "TORPEDO *target* $x_{1,0}$ $x_{2,0}$ $x_{3,0}$ $a_{1,1}$ $a_{1,2}$ $a_{1,3}$ $a_{2,1}$ $a_{2,2}$ $a_{2,3}$ $a_{3,1}$ $a_{3,2}$ $a_{3,3}$ $b_{1,1}$ $b_{1,2}$ $b_{2,1}$ $b_{2,2}$ $b_{3,1}$ $b_{3,2}$ c_1 c_2 c_3 d_1 d_2"

Torpedoes are built to last for exactly 1 minute in space.

Effects of a torpedo hit

Torpedoes do not damage the panels of a ship and do not interact with its shield. Instead they turn off every component on the ship and prevent turning them on for 3 minutes.

Given that the Devastationalizer$^{\text{TM}}$ was built by the Ks-Zk, the same alien race that has built the torture chamber (see Section 3.4) there is a sinister connection between the two. To prevent the torpedoes from being used against them they have engineered them with a back-door. The lock-out effect of the torpedo hit that normally lasts for 2 minutes can be cancelled by solving an exercise similar to those in the torture chamber.

The actual exercise to be solved is the payload of the torpedo and it is read at launch from a file in the document store (see Section 3.5) called `torpedo_payload.txt`. This file actually contains the last program successfully run in torture chamber, so it is within your power to load whatever sensible code you want into the torpedo!

2.12 Shields

Since galactic regulations forbid (or at least limit) shooting at spaceships with a sentient crew, you would think shields are only installed for nostalgic reasons or because of their soothing resonance. Well, think again! The regulations are not easy to uphold if there is no one to report violations and you will not be able to report anything if you get blasted with your shields down. Also regulations are not too specific about the definition of sentience and some people just choose to shoot at anything that moves claiming that the ratio of inanimate and animate matter in the universe is within $(\infty - \epsilon, \infty + \epsilon)$ and thus it is infinitely unlikely that they would hit anything sentient.

The CH024 is equipped with a fully autonomous shield system. Just make sure it is online and that it is receiving a healthy dose of energy from the power manager. Otherwise... things can get nasty.

How nasty

Whenever a laser beam gets close to the ship, it has to go through the quantum gravitational field[5] of the ship. The dynamics of this field are very complex, but it is very thin (10 times the wavelength of a 10TWh photon) so we can take a simpler view of it when the object interacting with it is traveling at the speed of light. In this case the repeating patterns of the field can all be condensed into a single model of an infinite grid.

[5]Quantum gravitational field: a circular field centered around the center of mass of the ship that is of the minimal such radius that it is encompassing all of the panels that make up the ship.

Photons entering the field start at the origin of this grid-based model. If they reach the point at $(0, 10)$, they have crossed the field and will be able to destroy a random panel of the ship (the luck-based finish of the central panel with the living quarters ensures that the randomly picked panel is some uninhabited panel unless there are no uninhabited panels). If a laser beam is made up of more than one photons any number of them may be able to reach $(0, 10)$ destroying a matching number of panels. If the last panel of the ship is destroyed, it is all over. Everyone on board dies horribly in the furnace of the explosion or the absolute coldness of vacuum.

> You will however respawn after 15 minutes at a random point in space.
> No points are lost, but you can not connect to the server while being dead (you can work on your own computers however). Any treasure (see Section 2.14) on board is transferred to the ship that has destroyed yours. You can lose the points you have received for the treasure this way. Use the Bank planet to safely stash away treasures from pirates!

This is why it is recommended that the shield be online and energized at all times.

Protocol

If you are unsatisfied with the factory installed shield component (perhaps you find it too safe or too reliable), you can replace it by implementing your own component. It has to be open for connections on port 2008.

The shield works by resonating the outer hull of the ship in a way that due to the interference of the emerging gravitational waves a pattern of gravitational singularities emerges in the quantum gravitational field. When the field is modeled by the above described infinite grid, the shield appears as a number of singularities on this grid. The shield controller is responsible for building these singularities. 30TWh of energy are required to build a singularity and it can sustain itself for 10 seconds before dissolving due to Bekenstein-Hawking radiation. The shield component has a 30TWh battery, just enough to create a single singularity. When the battery gets full the controller component is sent the message "SHIELD BUFFER ENERGIZED". The component can send messages of the format "BUILD x y" to create new singularities at coordinates (x, y). x and y can be arbitrary integer numbers. If the battery is not energized, an error message is sent in response.

When a laser beam hits the quantum gravitational field its photons appear at $(0, 0)$ in the grid model. What happens next takes very little time (less than a wing-beat), but we will look at it at a granularity of 1 oscillation of the photons.

The photons start moving according to their microprograms. They can move 1 wavelength of distance during 1 oscillation and can only move in parallel to the axes of the grid coordinate system. They do not move every oscillation as they also need time to execute non-movement commands of their microprograms. They may move together as a group or separately from each other.

The singularities are fixed at the coordinates they were created. Every oscillation they are able to trap 1 randomly chosen photon from the photons in their area of effect. The area of effect of a singularity is the 3x3 area around it on the grid (the exact position of the singularity is not different from the other 8 grid nodes in the area of effect).

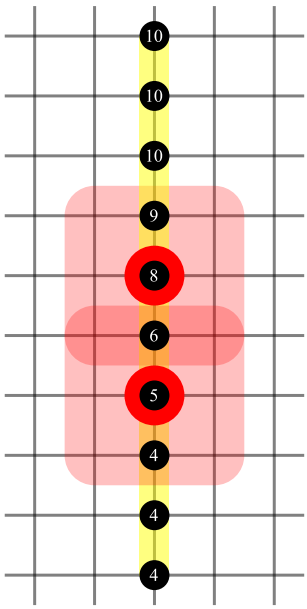
A photon can meet three kinds of fates: it can be trapped in the area of effect of a singularity, it can run out of time (it can only last for 100 oscillations before being scattered) or it can get through the quantum gravitational field by reaching $(0, 10)$. The scatter spectrum is a data series describing the positions of the photons when they have met their fates. It is of the format "(x_1,y_1),(x_2,y_2),(x_3,y_3),..." (no space characters, just parentheses and commas separating integer numbers).

Like the weapons component of the firing ship the shield component of the ship that was struck receives information about the scatter spectrum. The message containing this information is of

the format "LASER HIT DETECTED! SCATTER SPECTRUM IS *spectrum*", where *spectrum* is a scatter spectrum. This information can be useful for understanding the microprogram of the attacker and building singularities in a way that is most effective against it.

However unlikely this sounds, the following example might clarify the operation of shields even further.

In this example a laser beam of maximum strength (10 photons) has struck the ship. The ship has two singularities as shields at the time of the impact, at $(0, 4)$ and $(0, 6)$. The 10 photons appear at $(0, 0)$, the top of the illustration. They simply move downwards, towards the other side of the quantum gravitational field at $(0, 10)$, the bottom of the illustration. Note that this grid is infinite and the photons could very well go around the singularities.



The singularities are marked by red circles, their areas of effect by transparent red rounded squares and the trajectory of the photons by a faint yellow stripe.

The numbers in the small black circles are the numbers of photons that are not yet trapped by singularities at that position. We can see that 4 photons make it to the other side of the quantum gravitational field, which means that 4 panels are going to be destroyed.

Note that trapping a photon once it has reached the other end of the field is ineffective. If there were another singularity at $(0, 11)$, the laser beam would still destroy 4 panels. Also the photons entering the field are not yet sufficiently synchronized with it at the first oscillation and can not be trapped. This means that a singularity at $(0, -1)$ would not effect the results either. Physicists generally call this phenomenon the "photons step first" rule.

2.13 Laser microprograms

The photons of a laser beam move in the quantum gravitational field of the ship that they have hit according to their microprograms (see Section 2.10 and Section 2.12 for an introduction to these concepts in the case that you have missed them). Every photon starts at coordinates $(0, 0)$ and they execute the same microprogram, yet they can go on separate ways. The state of a photon is described by five variables: x, y, pc, reg and $scattered$. The initial values are $x = 0$, $y = 0$, $pc = 0$ and $scattered = false$. x and y are the coordinates describing the position of the photon on the grid model of the quantum gravitational field. pc is the "program counter" of the microprogram (indexed from 0). reg is the value of the integer-valued register of the photon. $scattered$ is a Boolean describing whether the photon has already been scattered or if it still executing its microprogram.

In addition to these five variables the whole of the beam has a coherence that can be used as a shared integer-valued variable.

The microprogram is a string in which every character is an instruction. At most 100 instructions can be evaluated before the photon is scattered by the fluctuations in the quantum gravitational field. The following list describes the instructions:

`l`, `r`, `u` and `d`: Moves the photon left, right, up or down. The coordinates are incremented by $(-1, 0)$, $(1, 0)$, $(0, -1)$ or $(0, 1)$ respectively.

`?` (question mark): Skips the next instruction with 50% probability.

`<` (less than), `>` (greater than), `^` (caret) and `v`: Skips the next instruction if there is a singularity to the left, right, above or below the photon. Only the central core of a singularity is detected

in this way and not their area of effect. The singularity can be at any distance, but must share a coordinate with the photon.

`1`, `2`, `3`, `4`, `5`, `6`, `7`, `8` and `9`: A jump of the given size. `1` is essentially a "nop", and `2` skips the next instruction. If the instruction preceding the jump instruction was `-` (dash) then the jump is a backward jump. In this case `2` continues to the instruction before the `-` for example.

`/` (slash), `\` (backslash): Increment or decrement the register by one. If the preceding instruction was `m`, increment or decrement the coherence (shared variable) instead.

`0` (zero): Skip the next instruction if the value in the register is zero. If the preceding instruction was `m`, perform the check on the coherence (shared variable) instead – skip the next instruction if the coherence is zero.

The microprogram can contain any other character (except newlines and spaces as dictated by the protocol used for their submission), but they do nothing (essentially "nop" instructions). The `-` and `m` characters while modifying the operation of other instructions are not different either.

pc is incremented by 1 after most instructions (except where otherwise implied). After every instruction *pc* is brought back within range by adding or subtracting the length of the program until necessary.

To give a viable default behaviour for those gunners who are not attracted to microprogramming the laser system adds a `d` instruction to the end of the submitted microprogram. This means that if the program is omitted when the laser is fired, the beam will still go through inactive shields without problems. This also means however that a more inclined microprogrammer has to take this extra `d` into consideration when writing their program.

2.14 Planetary exploration

Space is littered with planets. No one knows where they came from, but we have been trying to make some use of them since the dawn of ages. First they were used for evolution, but once we got tired of natural selection we have reused them as battlegrounds and as objects of colonization. Once they were hit by an alien invasion, we could do nothing but abandon them (those who did not wish to leave have practically chosen to use them as cemeteries instead). Today, however, humanity has long forgotten this silly history of the planets and uses them for the same thing as everything else: recreation.

A new fun form of planetary exploration has recently gained momentum: planetocaching. The rules of the game are simple: travel back in time, hide a treasure on one of the planets, travel back to the present and try to find it. You can even try to find it now, if you plan to travel back to the past in the future! What makes this game interesting is that if you travel to a distant enough past to hide the treasure, it will probably not only be moved together with the movement of continents, but it will also get disassembled.

The CH024 is equipped with a landing droid for use with planetocaching. You can simply deploy it to any planet, remote control it to find and reassemble the treasure and then take it onboard. The next time you come in to MagicalVehicles^{TM} for maintenance, we will download your planetocaching log and make sure the treasures are sent back! Guaranteed time-paradox-free entertainment for the family!

Protocol

The port for the controller of the landing droid is 2010.

You can control the droid with simple instructions such as “move north”, “pick up the apple” or “talk to the alien”. When the controller is first turned on, the droid is most likely in the droid bay. When you fly close enough to a planet, the droid can be deployed with the “land on *whatever the planet is called*” command. From there on the droid can explore the planet to its liking. If you decide you want to return to the ship, the easiest way is commanding the droid to “teleport back”. This command can even be used if the ship was not patient enough to wait for the droid and has left it behind. The downside is that the droid can not take any treasures with it using the teleport capability. To take the droid onboard together with the treasure, return to the planet and command the droid to “return”.

These are the basic commands, but the droid can apply a number of different tools littered on the surface. If a tool is appliable to an object then the operation results in an other object. Unfortunately, as your droid is a bit clumsy, it will break the tool while using it, so you can use one tool at most once. You should start with the object you find at your landing point.

A few guidelines:

- You are looking for the *treasure*. It is best to find someone who knows how to reassemble it, or you could wander around for ages. Or at least someone who knows some possible steps. Returning to the ship without the treasure is pointless. If you are unsure of what the treasure is, do not w orry. You will know once you find it!
- If you are feeling lost, just explore the planet for a while to get the hang of it. They may seem like an empty wasteland, but maybe you just have to walk a couple thousand steps until something interesting comes up. Planetocaching may take time, but at least the droid is playing outdoors!
- If you feel like you need a fresh start, just teleport back and land again. The planet will be reorganized to exactly the same way as it was before your visit thanks to the Universal Planetocachers’ Association. Likewise you should not worry about meeting other planetocachers on a planet – the Association has a manifesto that declares a strong spirit of non-interaction between planetocachers.
- Might want to draw a map.
- To ensure that you do not get stuck at a single planet collecting its treasure over and over, the droid is programmed to not land on a planet anymore once it has brought back the treasure from that planet.
- Only treasure is taken on board, the droid leaves everything else behind when returning.

Happy planetocaching!

Notes on handling treasure

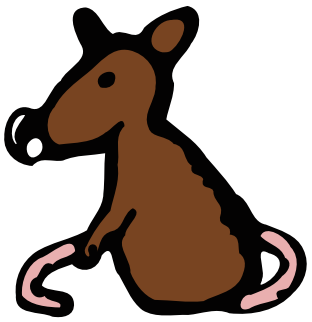
Once you have obtained a treasure from an exotic planet you will most likely want to carry it around on your ship and show it off to everyone. It is very ornamental in most cases and also a very nice gift idea. Pirates, however, are so fond of the idea of treasures as gifts, that they might want to shoot your ship to pieces and take the treasure. The prevent them from doing this, one planet was specially fitted with an advanced security system where you can safely store your treasures forever. The emphasis is on *forever*, because the planet is so safe that even you will be unable to get the treasures back from it. Not to worry however, the treasure still remains yours!

This fabulous planet is called Bank. To take treasures to it just land your droid on Bank. It will safely store all the treasure that you have on board.

Note that this planet is also guarded by a fleet of police vessels. If you fire lasers around Bank, prepare to be shot to pieces in return!

Treasures are worth 50000 points each! When you first bring them aboard, you receive half of this score permanently, but the other half will appear as volatile points, because it can go away if pirates destroy your ship. Take the treasure to the Bank to ensure their safety and to turn the volatile points into permanent points! Also note that the score chart on every ship enables the viewing of which teams have volatile points for some task, possibly attracting pirates to your ship!

2.15 Rat breeder



For thousands of years the greatest threat that humanity has known were the snakes on a plane. But today space-faring civilizations are chilled to the bone by a different horror. Rats on a spaceship! Even with clones of Samuel L. Jackson installed on every interstellar vehicle humanity's chances for survival are slim in the face of this evolutionary revolution. But now the tables are turned! You can show the world that you can be its master! You can perfect your own race of rats, a race of atomic superrats which will conquer the world!

The CH024 is the first consumer spaceship to feature a rat breeder interface, the Vermin Degenerator^{TM}. Through the Vermin Degenerator^{TM}you can plug a custom rat breeding program to the central computer system of the ship and shape the evolution of the critters to your liking. Want them cute and playful? Kill off the ugliest and most boring specimens of every generation! Want to turn them into hulking monstrosities to frighten your children in the morning? Kill off the harmless looking ones! Space rats are exceptionally fit for the purposes of the amateur breeder, because they mature very quickly (10 seconds is enough for a new generation to sprout) and – owing to the radioactive waste that they feed upon – they have a high percentage of serious mutations.

Once the competition is over, the favorite rats of every team will be pitted against each other in a tournament of tug of war! First, teams are randomly assigned to groups of four. All teams match with all other teams in the same group. Winners receive 2 tournament points, in case of a draw both players receive 1 tournament point. The team with the highest tournament score in each group receives 400000 points and the second receives 300000 points, and the first two teams proceeds to the playoffs. (Draws are resolved based on the sum of the position of the middle ball.) In the playoff phase you get points for every match won, 500000, 600000, 700000 and 800000 for the first, second, third and fourth respectively. So the winner of the tournament will get 2600000 points altogether in the playoff phase. If no favorite rat is selected then a random specimen will be picked for the occasion. If you have absolutely no rats on your spaceship a newly generated rat will be playing on your team.

Protocol

The rat breeder has to accept connections on port 2009.

The rats on board the CH024 can be split into several populations. The total number of rats can be at most 100. You can set the sizes of the populations with the "POPULATION_SIZE

population size" command. *population* is a string identifier for an existing or new population and *size* is an integer specifying the intended size of that population. This command is acknowledged with the "POPULATION SIZE SET TO *newsize*" message, where *newsize* is the actual size that was accepted. If the sum of the population sizes would climb above 100 (the maximum number of rats the system can handle) then *newsize* may be less than the *size* that was specified.

When the rats are bred, populations with more rats than specified will be decimated, and populations with less rats will be supplemented. Populations that contain no rats before breeding are supplemented with brand new and completely random rats, fresh out from the gene randomizer. Populations with at least 1 rat will be supplemented by creating new rats that are children of existing rats within the population[6]. These children get their genes from their parents (every gene is copied from either the mother or the father) and may get mutated before birth. Mutation destroys a gene and replaces it with a random new gene.

The probability of a mutation is closely related to the level of radiation the population is exposed to. At a radiation level of 0 no mutations occur and the newly bred rats have genes that are combinations of their parents genes. New populations start with a radiation level of 0. You can increase the amount of radiation with the command "SET_RADIATION *population radiation*". *population* is once again the string identifier of the population and *radiation* is the intended level of radiation – a number in the $[0, 1]$ interval (inclusive). When a new rat is generated as a child of two rats, its every gene has a probability of *radiation* to get destroyed and replaced by a random gene. At the maximal level of radiation the children will have absolutely no resemblance to their parents.

You can move rats from one population to another with the command "SPECIMEN_POPULATION *rat population*". *rat* is the the numerical id of the rat to move and *population* is the population to move to. Initially all 100 rats on the ship are in the population called "INITIAL", a population that has a size of 100 set, so you will need to decrease its size before creating your own populations. You can also use the "SPECIMEN_POPULATION" command to kill unwanted rats by moving them to the special population "DEAD". You can not call one of your own populations "DEAD" as this would be both a bad omen and a violation of the protocol.

A simple method of unnatural selection would work like this:

1. Increase radiation levels in "INITIAL" to something a bit higher than 0 (say 0.1).
2. Kill off maybe half of the rats that are the worst from your point of view.
3. Let the rats procreate.
4. Repeat steps 2 and 3 for a long time.
5. Once you see a rat that looks like something you were aiming for set it as your favorite rat.

Step 3 can be performed by the "STEP" instruction. Send a "STEP" message to initiate trimming and supplementing populations to the desired sizes. Step 5 can be done by sending a "FAVORITE *rat*" instruction. This saves the genetic material of the rat with id *rat* for future use. You can only have one favorite rat at any time.

When the component is turned on, it will receive the list of the current populations and the list of the live specimens. The population listing starts with the line "CURRENT POPULATIONS" and continues with lines of the format "POPULATION *population* SIZE *size* RADIATION *radiation*" where *population* is the string identifier of the population and *size* and *radiation* are its parameters set by the above described commands. The list ends with "END LIST". The list of the living rats follows starting with the line "CURRENT GENOTYPES" and continuing with lines of the format "*id population* $gene_1$ $gene_2$ $gene_3$ ...". *id* is the numerical identifier of the specimen, *population* is the string identifier of the population to which it belongs and the following genes are represented by integer numbers.

[6]It takes two rats to beget a child, but these two rats can be the same one. So yes, a single rat is enough to start a new population.

Evaluating specimens

So far you have learned of how to manage the breeding of your legion of rats. But how do you decide based on the genotype if you want to cultivate or extinguish a specimen? Well, you don't have to! MagicalVehicles™ provides you with the tools necessary to evaluate the rats based on their phenotypes – that is, their actual behavior and performance!

The first tool is the RatSimulator. It is a Java application that you can start with `java -jar RatSimulator <port>`. `<port>` is a mandatory argument that instructs the simulator to start on a specific port. You can connect to this port to introduce specimens into the simulator, set up their environment and analyze their behavior. The simulator also has a graphical user interface for your viewing pleasure that can be turned on and off.

Rats are represented in the simulator by 7 "balls" attached to each other by springs. 5 of these springs are actually muscles and their length can be controlled by the specimen. The genotype of the rat describes its actual structure and also the way it flexes its muscles. The nervous system of the rat has 15 inputs: 7 inputs tell it if the "balls" making up its body are touching the ground or not, 5 of its inputs give it information on how much their muscles are flexed, 2 are sine waves giving it a sense of rhythm and 1 tells it if the match has already started or not.

"Balls" can collide with each other and act according to the laws of physics. The environment in the simulator has a coordinate-system that has $x = 0$ in the middle of an infinite horizontal field, and positive x is to the right, while $y = 0$ is at the top of the simulator window, and positive y is downwards (the ground at the bottom of the screen is at $y = 300$).

Once connected to the simulator you can send the following commands:

step *steps*: Advances time by *steps* timesteps. If *steps* is omitted then it is considered to be 1. It is answered by "DONE".

newball p_x p_y v_x v_y *mass radius*: Creates a ball at the position (p_x, p_y) with the starting velocities (v_x, v_y) and the given mass and radius. A ball can be attached to other balls using springs or it can be nailed to a position. This command is answered by "ID *id*", *id* being the integer index of the newly created ball.

newspring $ball_1$ $ball_2$ *length strength*: Creates a new spring that is attached to balls $ball_1$ and $ball_2$ and has the given length and strength. This command is answered by "ID *id*", *id* being the integer index of the newly created spring.

followball *ball*: Sets the ball with index *ball* as the focus of the graphical presentation. The camera will move so that this ball is always visible. This command is answered by "OK".

nail *ball*: Adds a nail to the simulator that holds the ball with index *ball* still. This command is answered by "ID *id*", *id* being the integer index of the newly created nail.

delactor *actor*: Removes a spring or nail from the simulation. *actor* has to be the integer index of the spring or nail to remove. As a result of this command all springs and nails (but not balls) of indices greater than *actor* will have their indices decreased by one. This command is answered by "OK".

delball *ball*: Removes a ball from the simulation. *ball* has to be the integer index of the ball to remove. As a result of this command all balls (but not springs and nails) of indices greater than *ball* will have their indices decreased by one. This command is answered by "OK".

moveball *ball* p_x p_y: Moves the ball given by index *ball* to the position given by (p_x, p_y). This command is answered by "OK".

setspeed *ball* v_x v_y**:** Sets the velocity of the ball given by index *ball* to (v_x, v_y). This command is answered by "OK".

leftrat $gene_1$ $gene_2$ $gene_3$ ...**:** Adds the rat given by the genes listed into the simulation. Some rats are unfit for simulation and in their case this command fails with the answer "INVALID RAT". For valid rats the answer is "RAT ADDED AT p_x:p_y FACING RIGHT IDS id_1-id_7". It gives you the coordinates at which the rat was inserted and the the identifiers its balls have come to occupy (the ids in the $[id_1, id_7]$ interval inclusive). This is useful to know, because in tug of war games, the rope is always tied to the second ball of the rats!

rightrat $gene_1$ $gene_2$ $gene_3$ ...**:** This command is unavailable in the simulator but will be used in official tug of war tournaments. It inserts a mirrored rat into the simulator slightly to the right of the other rat. The rat added with "leftrat" has to pull towards the left and the rat added with "rightrat" has to pull towards the right. They will have a spring connecting their second in order balls. A small ball will hang in the middle of the rope and its position will determine the winner of a match once the match is over.

clear: Removes everything from the simulator, just like restarting it.

leftratpull *pull***:** Tug of war competitions start with a grace period, when rats have an opportunity to prepare for the match. The rats have an input that tells them if the match has already started or they are still in the grace period. You can set this input for the rat currently in the simulator with this command. Use $pull = 1$ to tell the rat that the competition has started. $pull = 0$ can be used to reset this input (but newly added rats come with the input set to 0 anyway).

getattr *ball***:** Retrieves the coordinates, velocity, mass and radius of the ball with index *ball*. The format of the answer is "AT p_x p_y SPEED v_x v_y MASS *mass* RADIUS *radius*".

The official tournament goes forth according to the following protocol:

1. A rat is added using "leftrat" and its opponent is added using "rightrat", which basically inserts a mirrored version of the rat 400 distance units to the right from the other rat.
2. For 150 timesteps the rats are allowed to fall to the ground and prepare for the action.
3. Using the "leftratpull 1" and "rightratpull 1" commands the rats are given notice of the race starting. At the same time a ball is inserted to the simulation between the two rats its center of mass 20 units above the ground, of radius 5 and mass 0.5. A spring is then created connecting the second ball of the left rat to the ball of strength 7 and a similar spring connecting the second ball of the right rat to the ball. Then time resumes.
4. The rats pull at the ball for 300 timesteps.
5. The winner is the rat in the direction of which the ball has been moved.

To see how well your rats would be faring another tool is provided. You can start this tool with `java -jar DuelSimulator <ratfile1> <ratfile2>`. The two arguments are the names of two files that should each contain a single line made up of the genes of a rat separated by space characters. These two rats will be loaded and a tug of war match simulated between the two. This is a visual tool to give you an impression of what to expect in a tournament, but it can not be controlled over a network connection so using it for specimen evaluation purposes would be most tedious.

2.16 Star map

The star map database of the ship is very important for safe navigation. In order for it to be easily replaceable when traveling to alternate universes, it is implemented as an external component.

Note that while the component is called a "star map", it is actually a map of *planets*. Also note that while this component supplies navigational data to the ship that is used by the factory installed navigational AI, third-party navigational modules do not have access to this database, and will have to access the star map using their own proprietary methods. MagicalVehicles$^{\text{TM}}$ is a strong believer in open standards and has started the standardization procedure of its star map interface. Once the standardization procedure is complete, the interface will be opened up the third-party manufacturers of navigational components. This should take no more than a few millennia, given the current pace of the procedure.

Protocol

The star map component should listen on port 2011.

Whenever the ship is in need of map data, it sends the following message to the component: "SHIP NEED DATA".

In response, the component should send a list of known planets in the following format. The first line should be "HERE IT GOES", followed by lines of the format "PLANET *planet* IS AT p_x p_y", where *planet* is the name of a planet and (p_x, p_y) are its coordinates. The coordinate system is defined by three important galactic objects. Planet Bank is the center of our universe, and thus has coordinates $(0, 0)$. The closest star is at coordinates $(1091, 649)$. Planet Azha, a popular hiking destination close to Bank is at coordinates $(-1595, -31)$. Every other planet should be given in the coordinate system defined by these three. Note that space is torus-shaped with a circumference of 30000 in both the x and y directions. This means, that for example the coordinates $(29969, 29969)$ are equivalent to $(-1595, -31)$ and both describe the location of planet Azha.

After the lines describing the known planets, the star map component has to tell the ship that there is no more with the message "NO MORE".

Chapter 3

Reference

3.1 Using the web interface

You should navigate the web interface using your *mouse*. No other animal should be used for navigation as anything larger than a mouse will get caught in the web and anything smaller will get eaten by the spider.

3.2 Visualizer

The visualizer is a tool that you can download from the document store. It is useful for visualizing a variety of data. You can run multiple visualizers on the same computer or on different computers. Your programs can connect to the visualizers to easily display data. Multiple programs can connect to the same visualizer, or a program could connect to multiple visualizers.

Common uses of the visualizer include:

- Displaying results from the radar component.
- Visualizing the progress of planetary exploration.
- Analyzing scatter spectra.
- Mapping the universe.
- Creating funny animations.

The visualizer is a Java application that runs on your computer and accepts connections on a freely chosen port. You can start the visualizer with the command `java -jar visualizer.jar <port> [ontop]`. `<port>` is a mandatory argument and is an integer specifying the port number on which to accept connections. `ontop` is an optional argument. If it is omitted, the visualizer window will behave just like any other window. If you add the `ontop` argument however, it will stay on top of other windows so you can better keep an eye on it (only works on platforms on which Java supports this feature).

Once the visualizer is started a black window appears on the screen. The default size is 500x500 pixels, but you can freely resize it [1]. The origin of the coordinate system of the visualizer will stay in the middle of the window. The coordinate system is pixel-based.

Now your own programs can connect to the visualizer on the given port. The protocol used is very similar to the protocols used on the ship. Every line is a message. A message is a series of words separated by space characters. Every message adds an object to the visualizer.

The syntax of the messages is "x y vx vy $fade$ *color type type-specific parameters*". x and y are the starting coordinates of the object. The origin of the coordinate system is in the middle

[1] Larger window sizes may use more system resources

of the window. The unit of the coordinate system is the size of a pixel. x grows towards the right of the screen and y grows towards the bottom. vx and vy describe the velocity of the object. The x and y coordinates are increased by vx and vy every 50ms. Every object starts with an opacity of 255 (completely opaque). Their opacity is decreased by $fade$ every 50ms until it reaches 0 (completely transparent). *color* is a string describing the color of the object. It is 6 characters long and uses the usual HTML color representation. The first two characters describe the red component as a hexadecimal integer in the range 0-255. The third and fourth characters describe the green component and the last two describe the blue component. E.g. "FFFFFF" is white, "FFFF00" is yellow. *type* is a string describing the type of the object to add along with the *type-specific parameters*.

The following types are supported:

CIRCLE *radius*: The circumference of a circle of the given radius.

SPOT *radius*: A filled circle of the given radius.

LINE x_2 y_2: A line from (x, y) to (x_2, y_2).

TEXT *message*: A piece of text. *message* may contain space characters. The bottom left corner of the message will be placed at (x, y).

IMAGE *filename scale*: Loads the image file *filename* (GIF, JPEG and PNG images are supported). The image added is scaled by *scale* – if $scale = 1$, the image is added in the original size. The *color* parameter is ignored for images.

POLYGON x_1 y_1 x_2 y_2 x_3 y_3 ...: An arbitrary polygon. The last point is connected to the first and the polygon is filled with *color*.

Every numerical argument can be a real number (not just integers). Use a $fade < 1$ for example to give your objects a life longer than 12.75 seconds. You can use $fade = 0$ to have an object stay forever. You can clear the visualizer of every object (even those with $fade = 0$) with the special message "CLEAR".

3.3 Telescope

The telescope is a tool that can be downloaded from the document store, similar to the visualizer. It is a Java application as well and can be started with the command `java -jar telescope.jar <ip or hostname> [ontop]`. `<ip or hostname>` is the address of the ship's central computer, "ship" in this case. `ontop` is an optional argument. If it is omitted, the telescope window will behave just like any other window. If you add the `ontop` argument however, it will stay on top of other windows so you can better keep an eye on it (only works on platforms on which Java supports this feature).

Once started you will instantly see that the program provides a detailed view of the short range surroundings of the ship. This image is constructed from the ship's optical sensors. The data stream that this is built from is available on port 10003 as a private service for the use of the telescope tool. Private, as in "not documented". Suffice it to say that the image built only contains the ship's immediate surroundings: everything within a 600m wide and tall square centered on the CH024.

The display of the telescope can be zoomed in and out with the mouse wheel, but it can not be rotated, translated, twisted, panned, shaken, skewered or stirred. The display window can be resized, but the viewable area will not extend beyond 300m in every direction. Expect to see objects vanishing and popping up in the middle of space if zoomed out too far in a large window. Also larger window sizes might require more system resources.

Using the optical input source service at 10003 either through the telescope tool or some other way may require significant resources on the part of the ship's central computer. Customers are advised to be reasonable about it and not run multiple instances of the tool. They just show the same thing anyway.

3.4 Torture chamber

The torture chamber is located in the eastern (soundproofed) wing of the CH024. During the Second Flame War it was used to interrogate captured Noobs by the wicked Leets. Its form of torture is primarily an exercise in programming languages that is not only considered painful by contemporary standards but also inappropriate or even perverted.

The torture chamber awards points to those who can solve a large number of its riddles within a time limit of 1 minute. To encourage the masochist in you the points are awarded at an increasing rate: solving the nth puzzle awards you $\lfloor 100000 \cdot 0.9^{111-n} \rfloor$ points. The 111th puzzle is the last one and completing it wins you a "Dedicated to Suffering" trophy.

The torture chamber was designed and manufactured by a very enthusiastic alien race, the Ks-Zk[2], a representative of whom has been a popular television personality in the early 30th century. The programming exercises are written in a translation of their native programming language. It has originally been designed by them as a means of destroying humanity. Humanity, however, has turned this threat into a useful computational device.

The language has two instructions for killing humans. The instruction "kill all humans" kills every human in the universe. However, the language also has a branching instruction "pick any X" which assigns a non-deterministically chosen value to the variable X. It is so non-deterministical that the universe actually branches into all the infinite possibilities for picking a value for X. The "kill all humans" instruction is then used to kill all humans in those instances of the universe where X does not satisfy some condition. After this instruction the human observer is sure to be in a universe where X did satisfy the condition. For some conditions it might be the only such universe.

Take good care of always having at least one universe in which humanity survives! (Because programmers are known to err from time to time, the interpreter for the language installed on the CH024 had the genocidal module replaced with a dummy.)

The following example demonstrates the computational power of this language:

```
What is a factor of N?
    Pick any K.
    Is N % K == 0?
        If answer == no then kill all humans.
    Answer K.

What is a factor of 221?
    Print answer.
```

This program would print either 13 or 17.

A further advantage of the language is that it is mostly useless in the hands of aliens. An alien could only find out if the result printed was right or wrong if he could reliably determine if humans have been all killed or not. A popular Geekian folk tale tells us the story of a young programmer who was abducted by a UFO. When he was interrogated about the secret password for a paysite he told them of the simple program that would give them the answer (very similar to the above example). The algorithm was so that if it returned the real password the boy would survive. The boy of course survived (at least in his – and our – universe) and the algorithm returned the real password. The boy however played dead and thus misled the aliens who never acquired access to the fabled site!

The other instruction for killing humans is "kill a human". Just like in any other programming language it kills a human and prints an empty line. Most languages disguise its nature by focusing on its side-effect when naming the instruction (such as `printf("\n")` or `println()`), but do not be fooled. There will be no less human deaths if you write the same program in C or Java!

(An understanding and appreciation of the exact syntax and semantics of the language will come with time spent in the torture chamber.)

[2]The Ks-Zk are the race also responsible for the torpedo launcher Devastationalizer^{TM}. For the sinister connection between the two technologies, please consult Section 2.11.

3.5 Document store

The document store is a boring and dusty part of the ship where nobody ever goes. Even though it does contain some files, their entertainment value is slim to none.

3.6 Black holes

Once believed to be fidgets of imaginations of some of the craziest scientists, black holes have proven to be something quite different. As they have assumed, the holes were leading to alternate universes. What the scientists had no way of hypothesizing is just how safe they would be. As it turns out, when you get blown to bits inside a black hole, the shock wave of the explosion creates a ripple inside the singularity, that gives a push to a copy of your ship that was frozen in time at the event horizon and dislodges it, so that it returns to the usual universe.

This property has led to people flying into black holes and doing all kinds of suicidal things for fun. After the wild early days however, companies have taken an interest in the movement, and a new sport emerged. Different crash courses were built, and today you find a fierce professional competition going on in every black hole. The competitions are open to everyone – just fly into a black hole, and listen to the radio broadcasts informing you of the particular goal of the competition in that hole. If you manage to complete the goals set by the creators of the crash course, your achievement will automatically be recorded, and you will be returned to the everyday universe. You will also be returned, if you fail to meet the goals or if you get the ship destroyed.

analogyLike in test cases, there are two kind of black holes. In the first case you get points for the completion of a task. In the second type of black holes your performance is measured and the top ten teams are awarded linearly decreasing volatile points. In both cases the score achievable are given in the welcome radio message, in the second case the score for ranking first is given.

3.7 List of port associations

Component	*Port*
generator	2001
power manager	2002
engines	2003
panel manager	2005
radar	2006
weapons	2007
shields	2008
rat breeder	2009
planetocaching	2010
star map	2011

Service	*Port*
logging	10001
tactical display	10002
optical sensors	10003
panel reports	10004

3.8 Reporting problems with the manual, the ship, the universe and everything

The manual, the ship and the universe are highly delicate instruments that were designed to perform reliably in a multitude of situations. They were thoroughly tested and have received the clearance of operation from the respective committees. They have been known to be very safe to

their users (with the possible exception of the universe) and no casualties have been reported as of the writing of this manual[3].

It is however possible that some rough edges are still present in the shipped products given how large they are (the universe being larger than the ship and the ship being larger than the manual). To report such rough edges and possibly errors please use the onboard artificial intelligence. It is directly coupled to the support center on the manufacturer's home planet and will be able to follow up on your error reports.

[3]Even though this predates the production and shipping of these products it is still good news. Some manuals have been known to cause casualties even during their writing!