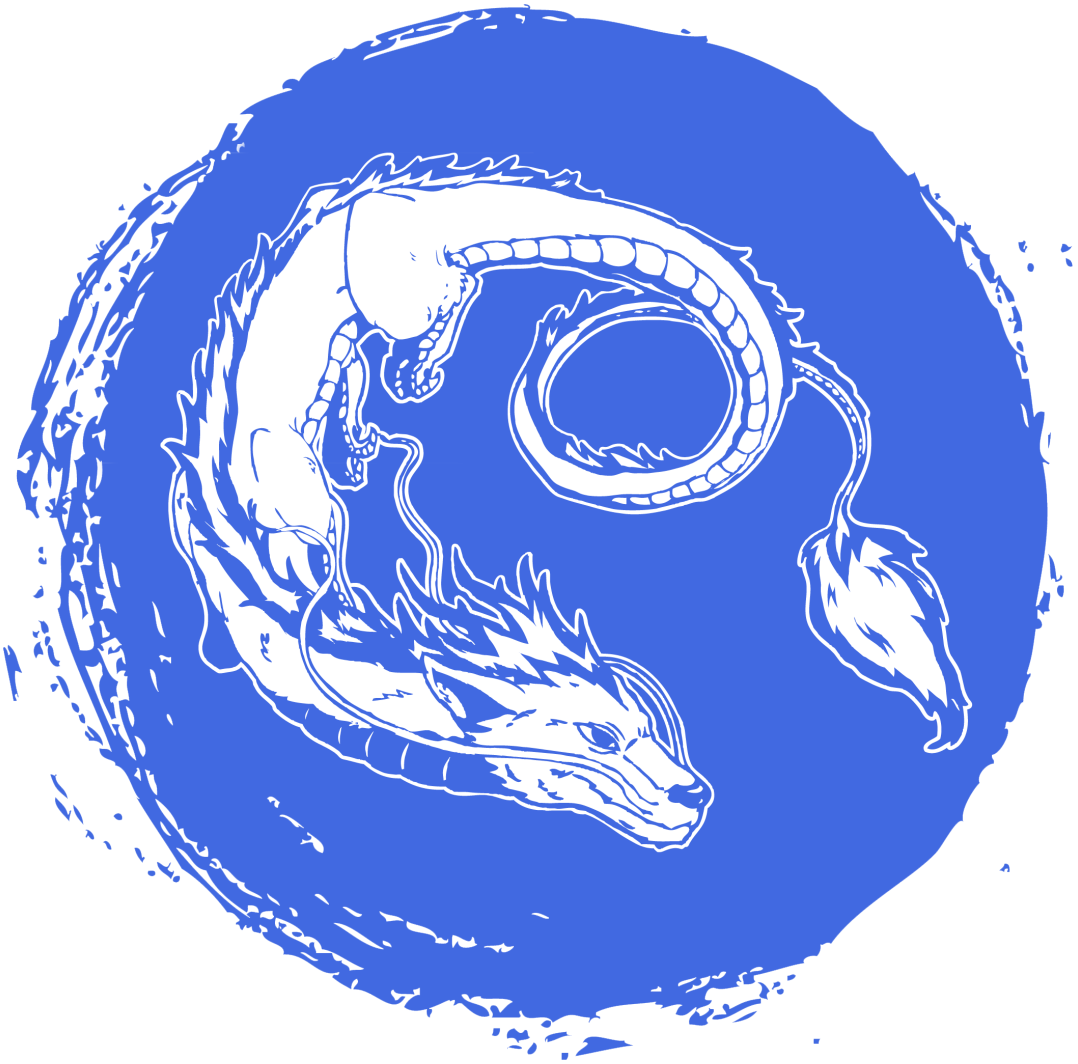
RUSH-JAVA — Subject

version **#v0.1.0**



YET ANOTHER KIND OF ASSISTANTS 2022 <yaka@tickets.assistants.epita.fr>

Copyright

This document is for internal use at EPITA (website) only.

The use of this document must abide by the following rules:

- ▷ You downloaded it from the assistants' intranet.*
- ▷ This document is strictly personal and must **not** be passed onto someone else.
- ▷ Non-compliance with these rules can lead to severe sanctions.

Contents

*https://intra.assistants.epita.fr

1 About given files

1.1 Files

Files to submit:

- ./src/*
- ./pom.xml

Provided files:

- ./src
- ./pom.xml
- ./README.md
- ./apidocs
- ./maps
- ./tests.http
- ./given-2.0-SNAPSHOT.jar
- ./given-2.0-SNAPSHOT-javadoc.jar
- ./creeps-voxel-2.0-SNAPSHOT.jar

1.2 Notes

The project's structure and the server are provided to you in the form of the architecture described above.

The build system used by this project is Apache Maven. The configuration file – `pom.xml` – is provided. Unless explicitly told by an assistant, do not modify this file, except to update your login.

The entirety of your source code must be placed under the `src/` folder. The entry point of your program is defined in `com.epita.creeps.Program::main`.

The following libraries are already configured in the provided `pom.xml`:

- Unirest: for REST calls.
- Jackson: for JSON parsing.
- Slf4j + logback: for logging.
- Validation-api: to add validation contracts to your objects.
- Junit: for unit testing.
- **Given: a library of code containing all DTOs (Data transfer object) and** some utility classes to play Creeps.

These are the only dependencies allowed for the project.

Before starting to code anything, make sure you understand all the provided files. We **strongly** advise you to read the README.md, the documentation given and the subject at least two times.

2 Creeps

2.1 Introduction

In creeps, players have one basic objective: collect resources and convert blocks, to become king of the hill. Two other advanced objectives are available and can be combined:

- survive against the horde
- avoid Hector

At the beginning of a game, you will be provided with two starting units – or `agents`, pseudo-randomly placed in a 3-dimensional world composed of blocks.

By using web services, you will make these units execute various commands, ultimately providing you with more resources, which in turn will allow you to spawn more agents.

Different kinds of agents and structures are available, each of them executing specific actions: they are detailled later in the subject.

Every block of the terrain can be converted to your color, adding one point to your expansion score and, in harder game modes, increasing the safety of your travels through space.

A player loses when he no longer owns at least a single block. In other words, when its expansion score drops down to zero.

Blocks can also be *mined* for resources. Mining a block has the additional effect of converting it to your color.

There are several ways to reach the end of a game. The game stops:

- When no user is left alive;
- When a timeout is reached (either in ticks or in seconds, depending on the server's configuration);
- When the last wave of the horde mode has been beaten by the players in cooperative mode.

2.2 Game modes

To help you reach cruising speed and to provide some more challenge to the most adventurous of you, the game can be tuned by enabling various features, presented hereafter.

2.2.1 Easy-mode

In this mode, all commands are performed synchronously and have a duration of one tick, allowing you to ease into the game more easily.

We recommend you try this first to get a feel of the game, but **don't forget to remove the training wheels** once you're confident enough.

Only a small number of achievements can be earned in this mode.

2.2.2 Horde

In this mode, players have to stay alive as long as possible against a horde of invaders.

After the server-specified warmup ticks, the first wave of enemies will spawn at random locations on any of the six sides of the playing field. Each tick, the defined number of horde agents will move as described below:

1. Is there a player's nexus in detection range? If so, move 1 block towards it.
2. Otherwise move 1 block towards the center of the map.
3. If the agent collides with any non-void block, the block explodes with a 5-blocks-wide blast.

After all agents have reached the center of the map or have been destroyed by players, a respite phase starts to allow players to reorganize.

And then, here we go again! The next phase begins with more horde agents.

This is considered the main dish of the subject, most end-game achievements require you to play with this option activated, and possibly with other players.

2.2.3 Hector

Hector is the "space-cleaner", hence the saying: "When you've got garbage, call Hector".

When Hector is enabled in the server options, every time a player reaches their unit cap, Hector will perform a GC pass. Additionally a GC pass will happen at fixed time intervals.

During a GC pass, Hector will try to determine reachable blocks. To do so, Hector will browse space starting at "GC roots", in our case, the player's alive nexuses and pylons.

From block to adjacent block, Hector will mark all reachable blocks.

Every block that is not reachable is considered "unreachable".

Once the mark phase is done, Hector will process to the collection phase, which in our case consists of collecting all agents not placed on a reachable block, and setting all unreachable blocks to the type void.

Hector is mean.

2.3 Viewer

To help you visualize what is happening during a game, you can connect to your server using any modern browser as long as *–enableWebClient* is set to true (which is the default). Whenever your server is running, you just have to connect to http://localhost:port, where the port is indicated by the value of *–httpPort* (default is 1337).

In order to move in the 3D world, you can use the following keys:

- `<space>`: Go UP
- `<shift>`: Go DOWN
- `<W>` / `<up arrow>`: Go FRONT
- `<S>` /`<down arrow>`: Go BACK
- `<A>` /`<left arrow>`: Go LEFT
- `<D>` / `<right arrow>`: Go RIGHT
- `<double space>`: Toggle fly mode

It is also possible to see around you using your mouse.

3 Units

This section presents all the Units and a general description of their role. All units are able to do mutliple actions, that are described in the summary section.

3.1 Agents

3.1.1 Probe

One of your two starting units, probes are versatile; they are capable of converting and mining blocks, scanning the environment, as well as building any structure unit.

3.1.2 Observer

Observers provide you with a quick and wide overview of the surrounding world with `scan4`. Note that they can neither mine nor build.

3.1.3 Dragoon

A combat unit that can fire a 1-blocks-wide blast within a range of 5 blocks. This blast will destroy all blocks contained within, as well as other players' agents, but will spare your own agents.

3.2 Structures

3.2.1 Nexus

One of your two starting units, nexuses allow you to spawn new units and get a detailed report over your current situation. They also serve as a safe point where the garbage-call Hector will never collect. But it also presents a yummy target for the Horde.

3.2.2 Pylon

Structures that can teleport agents between them.

3.2.3 Photon Cannon

Structures that can fire a 3-blocks-wide blast within a range of 7 blocks. This blast will destroy all blocks and agents contained within.

4 Commands

Each command has a cast time in ticks and its call might incur a cost in biomass and/or minerals. This cost information is provided to the player when he logs in.

When a player submits a command, the following process happens:

1. The game performs some safety checks on the command, that can lead it to being rejected. In this case, you will synchronously receive an error report as described in the command and report API section below.

   The safety checks cover everything that can be verified upfront:

   - Is the game running?
   - Is the opcode valid?
   - Is the player valid and alive?
   - Is the agent valid and alive?
   - Is the agent available (not already casting)?
   - Can the player pay for the command?
2. The necessary resources to pay for the command are removed from the player's resources.
3. The command is scheduled to execute after its cast time has passed.

4. Once the scheduled time is reached, the command executes.
5. The command can result in either success or failure for a variety of reasons (for example, the unit might be killed while casting), but since the command is performed in an asynchronous fashion, the game cannot notify you directly.
6. Instead, a report in Json format is produced, which the player can readily query and consult.
7. The agent is made available again.

All commands generate a report, and all reports will contain at least the following information:

```
{
    "opcode" : "status",          // Command opcode
    "reportId" : "1a2b3c4d5",     // Report ID
    "agentId" : "a1b2c3d4e",      // Agent ID
    "login" : "login_x",          // Player login
    "agentLocation" : {...},      // Location of the agent
    "status" : "SUCCESS"          // Execution status, SUCCESS OR ERROR
    "tick" : 42                   // The tick at which the command took place
}
```

An `error` field will appear in error reports, with a specific code to indicate what went wrong.

Some commands may add more information, which will be described on a per-command basis.

4.1 `move:<direction>`

Moves the agent according to the direction suffix. This suffix can either be `up` (y+), `down` (y-), `north` (z+), `south` (z-), `west` (x-) or `east` (x+).

Example: `move:south`.

4.2 `teleport`

A pylon command: teleports a unit standing on the same block as the pylon casting the command, from this block to the block of any pylon controlled by the same player.

This command takes a mandatory argument:

```
{
    "teleportedAgentId" : "theIdOfTheAgentToTeleport",
    "pylonId" : "theIdOfTheDestinationPylon"
}
```

4.3 convert

Converts the block to your block type and adds it to your territory, giving you one expansion point.

Beware that converting some nasty blocks may result in very bad side-effects.

May add a `causeOfDeath` to the report.

4.4 mine:minerals, mine:biomass, unload

Mines the block for the requested resource.

Probes have a defined minerals and biomass load rate (defaults to 8, sent by the server at login) which defines the maximum amount of resources one `mine` command can load upon the agent (obviously, if the block has less than that amount left, it becomes the amount loaded).

Probes can mine the same resources several times in a row to load more resources, up to a certain value defined by the server (defaults to 40). Further loading of this resource will have no effect beyond wasting precious ticks.

A probe can only hold resources of a single type at any given point, so mining one resource will make the probe lose all held content from the other resource.

Once a block has no more resource left, it is automatically converted to the probe's player block type, giving them one expansion point.

As with converting, make sure you are not mining anything either explosive or hot...

Adds the fields `payload` and `resourcesLeft`, indicating respectively the resources currently held by the agent and the resources left on the block. Both fields follow this format:

```
{
    "minerals" : 42,        // Minerals amount
    "biomass" : 42          // Biomass amount
}
```

To benefit from the mined resources, you will have to make the probe unload its payload at a nexus. To do so, simply move the probe to your nexus of choice and cast the `unload` command on the probe.

4.5 inspect

Get information about the block you are currently on.

Adds these fields to the report:

```
{
  "blockId": 17,
  "minerals": 0,
  "biomass": 0,
}
```

4.6 `scan:<power>`

Gives information on the cube centered on the agent. The size of the cube depends on the suffix. The suffix can either be `1` (cube of size 3), `2` (cube of size 5) or `4` (cube of size 9).

Adds two fields to the standard report:

- `blocks`: a string representing the scanned blocks, *run-length encoded*.
- `units`: the list of units discovered during the scan.

Each element in the `units` field follows this format:

```
{
    "opcode" : "dragoon",       // Agent's opcode
    "location" : {...},         // The agent location
    "player" : "login_x",       // Player login
}
```

4.6.1 About run-length encoding (RLE):

Imagine the output of a `scan:4` command: we have a 9-blocks-wide cube composed of $9^3 = 729$ blocks, each of which you want to get the position and block type for. Transmitting this information in standard JSON format would occur a *massive* waste of bandwidth and a performance hit on both the server and your client.

So instead we went with the *run-length encoding* way.

When we need to transmit the layout of a 3D area, we first convert it to a 2d array, iterating over the z axis, then over the y axis, then over the x axis (all iterations are from lowest to highest coordinate).

So a cube composed of:

- `[0, 0, 0] = A`
- `[1, 0, 0] = B`
- `[0, 1, 0] = C`
- `[1, 1, 0] = D`
- `[0, 0, 1] = E`
- `[1, 0, 1] = F`
- `[0, 1, 1] = G`
- `[1, 1, 1] = H`

Would be returned as `ACBDEFGH`.

Once the blocks are ordered, instead of simply passing them as-is, we group them by *runs*. A run is a sequence over which the same data value is repeated several times.

Finally, we replace these repetitive runs by a unique instance of the specific data value, prefixed by its occurrence count.

For example, the sequence `AAAAABAABBBBBBBBB` would be replaced with `5A1B2A9B`.

Since we are not dealing with single characters or digits, we need a little bit more ceremony to transmit data. As such, runs are separated with semicolons, and the data value and the number of occurrences within a run are separated with a colon.

Therefore, in the case of our previous example, we would actually get `5:A;1:B;2:A;9:B`.

Taking an example from start to finish:

The cube:

- `[0, 0, 0] = PLAYER_01`
- `[1, 0, 0] = PLAYER_01`
- `[0, 1, 0] = PLAYER_01`
- `[1, 1, 0] = VOID`
- `[0, 0, 1] = VOID`
- `[1, 0, 1] = VOID`
- `[0, 1, 1] = VOID`
- `[1, 1, 1] = MINERALS_HIGH`

The array:

```
["PLAYER_01", "PLAYER_01", "PLAYER_01", "VOID", "VOID", "VOID", "VOID", "MINERALS_HIGH"]
```

The RLE:

```
"3:PLAYER_01;4:VOID;1:MINERALS_HIGH"
```

Given with block id codes instead of block names:

```
"3:1;4:127;1:23"
```

You are provided with a RLE decoding method in the `given` jar.

4.7 `spawn:<unit>`

Spawns the given unit at the spot it is invoked. The unit can be a `dragoon`, `nexus`, `probe`, `observer`, `pylon`, or `photon-cannon`.

Also adds the fields `spawnedAgentId`, `spawnedAgentOpcode` and `spawnedAgentLocation` to the standard report.

4.8 release

Releases the agent.

4.9 noop

Does nothing. Useful mainly for testing and/or status purpose.

Adds the fields `block` and `agentStatus` to the standard report.

4.10 credit

Transfers resources to a fellow player in need.

4.11 message

Broadcasts a message of your choice. The location of the sending unit is added to the message.

4.12 fetch

Returns the list of all messages received since the last `fetch` invocation.

4.13 fire:<unit>

Fire one projectile, from the target agent to any specified destination block. The unit can either be `dragoon` or `photon-cannon`.

Dragoon fire has a range of four blocks in all directions, has no blast outside of the target blocks, and will not destroy the owner's units.

Photon-Cannon fire has a range of 6 blocks in all directions, plus a one block blast radius and will destroy both friendly and enemy units.

To successfully cast a fire command, you need to pass a parameter having this format:

```
{
    "destination" : {
        "x" : 42,
        "y" : 51,
        "z" : 21
    }
}
```

5 Agent Summary

Opcodes	Probe	Observer	Dragoon	Nexus	Pylons	Photon-Cannon	Duration
noop	yes	yes	yes	yes	yes	yes	1
message	yes	yes	yes	yes	yes	yes	1
release	yes	yes	yes	yes	yes	yes	1
inspect	yes	yes	yes	yes	yes	yes	1
unload	yes	no	no	no	no	no	3
convert	yes	yes	no	no	no	no	2
mine:*	yes	no	no	no	no	no	1
move:*	yes	yes	yes	no	no	no	1
scan:1	yes	yes	no	no	no	no	1
scan:2	no	yes	no	no	no	no	1
scan:4	no	yes	no	no	no	no	1
fire:dragoon	no	no	yes	no	no	no	1
fire:photon-cannon	no	no	no	no	no	yes	1
spawn:nexus	yes	no	no	no	no	no	10
spawn:photon-cannon	yes	no	no	no	no	no	5
spawn:pylon	yes	no	no	no	no	no	4
spawn:probe	no	no	no	yes	no	no	3
spawn:observer	no	no	no	yes	no	no	2
spawn:dragoon	no	no	no	yes	no	no	5
teleport	no	no	no	no	yes	no	1
fetch	no	no	no	yes	no	no	1
credit	no	no	no	yes	no	no	1

6 Server Interface

6.1 Creeps Server

In order to launch your program on the good IP and port, your program will be executed as follow:

```
java -jar target/<login_l>-creeps-1.0.0.jar [HOSTNAME] [PORT] [USERNAME]
```

6.2 Endpoints

A server exposes 5 HTTP endpoints through which you can interact:

- GET /status
- GET /statistics
- POST /init/`login`
- POST /command/`login`/`agentId`/`opcode`
- GET /report/`reportId`

In order to play the game, you will have to make HTTP requests to the server. You will have to use both GET and POST requests. As a reminder, a GET request is used for viewing some resource, while a POST request is used for changing a resource. In this last case, you may have to create a body, which will contain some information about what you want to modify. To be understood correctly by the server, all your bodies must be in a Json format.

For this rush, you're allowed to use Unirest with the Jackson library.

```
// send a GET request
Unirest.get(uri).asJson()
// send a POST method
Unirest.post(uri).body("{}").asJson()
// transform a json String to an object
ObjectMapper mapper = new ObjectMapper();
ExampleClass example = mapper.readValue(jsonString, ExampleClass.class);
```

You will need to read the documentation of those two libraries to have a better understanding of how to use them.

6.3 GET /status

Returns a JSON object with a single `running` field indicating whether or not the game has already started.

```
{
  "running": false
}
```

6.4 GET /statistics

Returns the current statistics of the server. You can also use this endpoint to check if the game has started.

```
{
  "freeSpace": 196607,
  "agentsPerType": {
    "nexus": 1,
    "probe": 1
```

(continues on next page)

(continued from previous page)


```
  },
  "agents": 2,
  "gameRunning": false,
  "tick": 23,
  "scheduledGameStartTick": 30,
  "scheduledGameEndTick": 1200,
  "hordeWave": 0,
  "hectorRuns": 0,
  "dimension": {
    "x": 64,
    "y": 48,
    "z": 64
  },
  "players": [
    {
      "agents": 2,
      "name": "psx",
      "status": "ALIVE",
      "agentsPerType": {
        "nexus": 1,
        "probe": 1
      },
      "blocks": 1,
      "minerals": 300,
      "biomass": 300,
      "achievements": []
    }
  ]
}
```


6.5 POST /init/`login`

In order to unlock achievements, you need to be connected to a public server with the following login: `yourlogin---commithash`, where `commithash` is the hash of the commit your are using.

i.e.: localhost:1664/init/`dumeige_a---f436998`

It will allow us to identify your AI.

You **must** use your own login and hash. Using another student's login or commithash will be considered cheating.

Init STEP by STEP:

- git add `<your files>`
- git commit # write your message
- git log # copy `<the last commit hash>`
- # make a string containing: localhost:1664/init/`yourlogin---<the last commit hash>`
- # post it to the server and receive the response
- # Then you're Done.

If you are not looking for achievements you are free to write whatever you want as login.

Once you connect to the game, you are provided with all the information you need to start playing. Namely:

- The dimensions of the world;
- Your position in said world;
- A summary of the cost of all commands available in the game;
- The block type that materializes your territory;
- Your starting resources;
- The server's configuration.

If the `error` field is not `null`, it contains one of the following error messages, describing what error occurred:

- `"login unavailable"`

  The `login` you provided is not available, try again with another one.

- `"too many players"`

  This server is full, try another one.

- `"new players not allowed to enter while a game is in progress"`

  A game is already in progress on this server, try again later.

For more information, take a look at the InitResponse class of the given library.

6.6 POST /command/`login`/`agentId`/`opcode`

Orders the agent with the given `agentId` to perform the command with the given `opcode`.

```
{
    "opcode" : "command",       // Information about command success
    "reportId" : "1a2b3c4d5",   // Report ID
    "errorCode" : null,         // Error description, might be omitted when null
    "error" : null,             // Full error, might be omitted when null
    "login" : "",               // Player login, might be empty
    "agentId" : "",             // Agent ID, might be empty
    "misses" : 0                // Number of misses
}
```

If the `errorCode` field is not `null`, one of the following errors occurred:

- `"notrunning"`

  The game is not running. Either it has yet to start or it has already ended.

- `"unrecognized"`

  The `opcode` you requested was not recognized.

- `"noplayer"`

  The `login` you provided did not match any player on the server. You might have been kicked for inactivity.

- `"unavailable"`

  Your agent is already doing something. Wait until it finishes before assigning it another job. Note that your missed calls counter has been incremented. If it goes over a certain value, the next missed call will lead to the death of the corresponding agent.

- `"nomoney"`

  You do not own enough resources at the moment. Try again later when you do.

- `"dead"`

  Your agent died because of too many missed calls. Note that that report is only sent once, after which you will receive a "noagent" response.

- `"noagent"`

  The `agentId` you provided did not match any of your units. Either you previously released it or it has died.

- `"initerror"`

  The body of the request caused an error.

6.7 GET /report/`reportId`

Retrieves the report with the given `reportId`.

You will find the structure of the response of each possible opcode in the *Commands* section.

The report is only available after the duration specified for each task has passed. If `reportId` does not exist, or if the report is not ready yet, you will get the following response:

```
{
    "opcode" : "noreport",
    "error" : "No such report",
    "reportId" : "173040eba"
}
```

7 Behaviour and Design Tips

7.1 Agents and threading model

Even though it would be possible to implement an IA over a single execution thread, said IA would be very limited in terms of its capabilities. We advise you to adopt a more advanced design wherein all your agents could be dispatched over not one but several threads. This would allow you to scale up to dozens or even thousands of agents on your computer depending on your implementation.

As you were told during the presentation of this project, we encourage you to gather information about the following subjects:

- Threading in Java
- BlockingQueue
- CompletableFuture

As most of these notions were covered during the courses, your first step should be to have a second – if not first – look at those.

7.2 Delays

As you probably know by now, a unit cannot just chain its commands. Each command takes a small amount of time to be executed (a delay), meaning that a unit can only execute a command once the delay of its previous command has passed. You are **not allowed** to use `Thread.sleep()` to make a unit wait between commands. It is up to you to find a smarter way to handle delays.

All durations are given in ticks. The number of ticks per second is communicated by the server in the init response.

7.3 Debugging Threads

Printing to standard output is a bad idea, you'll be somewhat flooded and you will not be warned of thread crashes. Using the debugger will work, provided that the good thread is caught.

You are strongly advised to use a `Logger` for each of your agents. The principle is simple, each agent contains a `Logger` instance that will write messages to a file with a specific name. This will allow you to follow the behaviour of your agents individually.

8 Achievements

You will be graded based on achievements, **not** on traces. This means it is very important that you submit code often (after testing it locally) in order to start securing them early. The more achievements you earned, the higher your grade will be. You are presented with a few achievements to reach.

To get the list of all achievements, add the `--printAchievements=true` parameter when starting the server.

Achievements need to be activated with `--trackAchievements=true`.

Note that some achievements can only be completed when a specific feature is enabled; in this case, requirements are explicitly indicated within the achievement's description.

Hereunder is a sample of unlockable achievements:

Name	Description
Hello, world	Connect to the game.
Oscar Mike	Move any unit.
Fire at will!	Fire with any capable unit.
It's Aliiiiivvvve!	Spawn any unit.
Running Free	Get to any boundary block of the map with any unit (just sixteen, a pickup truck, Out of money, out of luck...)
Show me the money	Mine 100 resources, cumulated.
Settlers	Build a base composed of a nexus, and a pylon and a photon cannon both within a 5x5x5 cube of the nexus.
Now you see me, now you don't	Teleport any unit using a pylon.
Command And Conquer	Expand your territory to 200 blocks in less than 1000 ticks.
CleanMemory	Survive one GC cycle.
Tidy	Survive 10 GC cycles without loosing any agent to Hector.
Survivor	Survive 5 turns in horde mode.
Fetch	Fetch Messages.
Communicate	Send a Message.
SOS	Send a unit at the location of the emitter of any received Message.
A Hole In My Pocket	Send resources to a fellow player.
Avengers Age Of Ultron	Successfully defeat at least 10 waves in horde mode with at least 5 players, all of them ending alive (Quicksilver doesn't count).
Avengers Endgame	Successfully defeat at least 10 waves in horde mode with at least 5 players, all of them ending alive, this time with Hector wrecking havoc.

Don't be afraid, I just wanna help you.