The Single-Player GVGAI Learning Framework Technical Manual

Jialin Liu[1], Diego Perez-Lebana[2], and Simon M. Lucas[1]

[1]School of Electrical Engineering and Computer Science, Queen Mary University of London, London E1 4NS, United Kingdom
[1]School of Electrical Engineering and Computer Science, Queen Mary University of London, London E1 4NS, United Kingdom
[2]School of Computer Science and Electrical Engineering, University of Essex, Colchester CO4 3SQ, United Kingdom

Last update: November 2017

Contents

Disclaimer and Acknowledgement

The Learning Track framework is based on the General Video Game AI framework. The implementation and testing of the Learning Track framework are joint work with Dr. Diego Perez-Liebana and Daniel-Valentin Ionita (University of Essex, UK). Prof. Simon Lucas (Queen Mary University of London, UK) and Prof. Julian Togelius (New York University, USA) provided valuable suggestions and technical support.

Thanks to Ercüment Íİlhan (Istanbul Technical University, Turkey) and Kamolwan Kunanusont (University of Essex, UK) for having provided sample agents written in Python and Java.

Thanks to Qi Zhang (National University of Defence Technology, China) for testing the framework under Windows system.

Thanks to Alexander Dockhorn (Otto-von-Guericke University, German) for providing valuable feedback.

1 Overview

This technical manual describes how to set up the Single-Player General Video Game AI (GVGAI) Learning framework, how to implement an agent, the competition process and submission guidelines. Section 2 briefly introduces the GVGAI framework and competitions. Section 3 gives step-by-step instructions for setting up the single-player GVGAI learning framework and running a sample agent. Section 4 details the framework and the technical requirements to implement a learning agent. Section 5 explains how to submit an agent to the competition.

Note that some of the guidelines or values for parameters are only for submissions to the competition. For research use, you do not need to respect them, and you can easily change some of the parameters (see more in Section 4.1).

2 General Video Game AI framework and competitions

Two main research questions in games to answer are can an AI program solve a particular *difficult* game and can an AI program solve a (large) set of *different* games? There has been research work and competitions around the former one, such as the Computer Game Olympia. We aim at answering the latter using the General Video Game AI (GVGAI) as benchmark problems.

The General Video Game AI (GVGAI) framework[1] was initially designed and developed by the University of Essex (UK) and New York University (USA), aiming at using as a research and competition framework for studying General Game Playing (GGP). Though it is called GVGAI, some board games are included in the game sets. When it was firstly announced, there was only the single-playing planning track [3]. In the following years, the framework has been improved and extended to hold two-player planning track, level generation track and rule generation track. The games were written in Video Game Description Language (VGDL) [1, 4], the framework was written in Java and only accept submissions written in Java until the framework has been extended again to hold the single-player learning track in the IEEE's 2017 Conference on Computational Intelligence in Games (CIG17) [2].

More about the competitions can be found on the GVGAI website. The following section mainly explains the procedure in the single-player learning competition.

3 Installing and run

3.1 Requirements and supports

System GVGAI can be built and run on **Unix**, **Mac OS X**, and **Windows**.

[1]`http://www.gvgai.net/`

Gson Gson is required but it is already included in the source code.

Language It supports agents written in **Java** or **Python**. The **Python** client has been tested with Python3.5.

Communication At every game tick, the agent can send a game action (`String`) or a command (`String`) to exit a game, and can choose either or both formats of the game station observation listed below

- a serialised JSON game observation (`String`)
- a screenshot of the game screen (`PNG`).

The choice can be changed at every game tick or at the beginning or end of a game playing. More can be found in Section 4.2.

Remark: Though there is a communication between the server and client, there is no need to run the server separately, it will be compiled and run automatically when you compile and run an agent using the provided scripts. For each of the scripts, two formats are provided for Windows (`bat`) and non-Windows users (`sh`).

3.2 Download

The compressed source code can be download from GitHub [2] (master), then unzipped manually to repository named `gvgai`. It can also be clone by: `git clone https://github.com/EssexUniversityMCTS/gvgai.git`

We will assume that the GVGAI framework is extracted to `gvgai` and we highly recommend not to change the name of the repository as it will require changes in the `bat`/`shell` scripts.

The single- / two-player planning and single-player learning competitions use the same framework, thus the single-player learning framework is included in `gvgai`. However, the setting is slightly different when importing the project using IDEs, as detailed in the Sections 3.3 and 3.4.

3.3 Java agent: build and run a sample agent

3.3.1 Step-by-step using IntelliJ

Import the project Launch the IDE, import the project `GVGAI-JavaClient`. *Remark: Please import the client* ***not*** *the whole project* `gvgai`.

Set your own Java agent (optional) If you want to use your own agent, you can change the name of agent in `TestLearningClient.java`. Otherwise, the sample random agent `sampleRandom/Agent.java` will be used. More advanced settings are explained in Section 4.1.

[2] `https://github.com/EssexUniversityMCTS/gvgai`

Run an agent Compile and run `TestLearningClient.java` in the package `gvgai/GVGAI-JavaClient/src`. It will compile the server, build the communication and launch the program. The output of the client will be shown on the console, while the output of the server will be logged in the `logs/` folder:

- `output_server_redirect.txt` for `stdout`,
- `output_server_redirect_err.txt` for `stderr`.

The agent will run on a given game (by default *Aliens*) with 5 minutes learning time.

3.3.2 Step-by-step using Eclipse

The basic steps are same as using IntelliJ, except that the build folder is not `out` but `bin`. You need to replace `out` by `bin` in `GVGAI-JavaClient/src/utils/runServer_nocompile.sh`

3.4 Python agent: build and run a sample agent

3.4.1 Step-by-step using IntelliJ

Import the project Launch the IDE, import the project `GVGAI-PythonClient`. *Remark: Please import the client **not** the whole project `gvgai`.*

Set your own Python agent (optional) If you want to use your own agent, you can pass the name of agent as a parameter to `TestLearningClient.py` in `gvgai/clients/GVGAI-PythonClient/src`. Otherwise, the sample random agent `sampleRandom/Agent.py` will be used.

Run an agent Compile and run `TestLearningClient.py` in the package `gvgai/clients/GVGAI-PythonClient/src`. It will compile the client, build the communication and launch the program. The output of the client will be shown on the console, while the output of the server will be logged in the `logs/` folder:

- `output_server_redirect.txt` for `stdout`,
- `output_server_redirect_err.txt` for `stderr`.

The agent will run on a given game (by default *Aliens*) with 5 minutes learning time.

3.4.2 Step-by-step using Eclipse

The basic steps are same as using IntelliJ, except that the build folder is not `out` but `bin`. You need to replace `out` by `bin` in `GVGAI-PythonClient/src/utils/runServer_nocompile_python.sh`.

3.5 Build and run an agent without IDE

3.5.1 Build and run a Java agent

Run `oneclickRunFromJavaClient.sh` or `oneclickRunFromJavaClient.bat`.

3.5.2 Build and run a Python agent

Run `oneclickRunFromPythonClient.sh` or `oneclickRunFromPythonClient.bat`.

3.6 Advanced parameter settings

A list of games have been given in folder `gvgai/examples/gridphysics`. The names of games are listed in `gvgai/src/tracks/singleLearning/TestSingleLearning.java`.

3.6.1 Legal arguments

You can pass arguments to your client. The legal ones are listed below:

- `-serverDir` [PathToServerCode]: Where the source folder of the server code locates. Default: `..\..\..` for Windows system or `../../..` for OS X and Unix.
- `-shDir [PathToTheScriptToBuildServer]`: Where the script to build and run the server locates. Default: `utils`.
- `-gameId [IdOfGameToPlay]`: The id of the game to play. Default: 0.
- `-agentName [NameOfAgentToRun]`: The name of your agent, should be the same to the package name and your GVGAI account username. Default: `sampleAgents.Agent` for Python and `agents.random.Agent` for Java.
- `-visuals` Visualisation of game playing on if this is passed. Otherwise, visualisation off.

3.6.2 Run with IDE

One can set the game to play and the agent to use by editing `TestLearningClient.java` or `TestLearningClient.py`, or by passing arguments to the executable file, otherwise the default values will be used. More can be found in the source files.

3.6.3 Run without IDE

One can set options by editing the `oneclickRunFromJavaClient.sh/.bat` or `oneclickRunFromJavaClient.sh/.bat`[3].

[3]The scripts only include basic usages and arguments for demonstration, please feel free to create your own.

Python client In `oneclickFromPythonClient.bat/.sh`:

```
python TestLearningClient.py -serverDir [PathToServerCode]
   -shDir [PathToTheScriptToBuildServer]
   -gameId [IdOfGameToPlay] -agentName [NameOfAgentToRun] [-visuals]
```

Java client In `oneclickFromJavaClient.sh`:

```
java -classpath $build_folder TestLearningClient -serverDir [PathToServerCode]
-shDir [PathToTheScriptToBuildServer]
   -gameId [IdOfGameToPlay] -agentName [NameOfAgentToRun] [-visuals]
```

Java client (Window user) In `oneclickFromJavaClient.bat`:

```
java -cp %build_folder% TestLearningClient -serverDir [PathToServerCode]
-shDir [PathToTheScriptToBuildServer]
   -gameId [IdOfGameToPlay] -agentName [NameOfAgentToRun] [-visuals]
```

3.7 Possible issues and solutions

java.net.BindException: Address already in use (Bind failed)? Maybe another process is already running at that port. In Unix, check: `lsof -i tcp:<port>`

If the connection hangs Please check if the default port 8080 is already being used. If yes, please try another one by editing both `SOCKET_PORT` in `CompetitionParameters.java` and `SOCKET_PORT` in `CompetitionParameters.py` or `CompetitionParameters.java`.

Path to client code If you move the `GVGAI-JavaClient` project to another folder, please change the path to the build and source folders in `runClient_nocompile.bat` or `runClient_nocompile.sh`.

Attention: This will affect the location of the output file `ClientLog.txt` (see `GVGAI-JavaClient/src/utils/IO.java`).

(Windows user) FileNotFound exception for `logs/ClientLog.txt` Please create manually a folder namely logs. Normally, the logs folder has been created and added to Git.

(Windows user) Path to SDK You may need to edit `runClient_nocompile.bat` to specify the path to the sdk.

(Windows user) If you run a Java client and it hangs If you fail with this, please check if the path to javac is set, please refer to *javac-isnt-working-in-windows-command-prompt*. If it still hangs, please try to run it using `oneclickRunFromPythonClient.sh` or `oneclickRunFromPythonClient.bat`. It has been observed that in Windows 7, the `oneclickRunFromPythonClient.bat` works but under Eclipse it does not work properly.

	Planning tracks		Single-Player Learning
	Single-Player	Two-Player	
Similarities	• Play unseen games, no game rules available • Access to game score, tick, if terminated • Access to legal actions • Access to observation of current game state		
Forward model?	Yes		No
History events?	Yes		No
State Observation?	Yes Java		Serialised SO Java &Python

Table 1: Comparison of the planning and learning tracks.

A **Q&A** page can be found in the wiki of GVGAI.

4 Single-player learning framework

The single-player learning framework is part of the GVGAI framework. Different from the planning tracks, no forward model is given to the controllers, thus, no simulation of games is accessible. Table 1 compares the single-player planning and learning tracks. An example of the serialised state observation is given in Appendix and a screenshot of the game screen are given in Figure 1.

4.1 Parameters

Before explaining the main procedure of the single-player learning competition, the notations and corresponding variables in the framework are listed below:

4.2 Implement an agent

A Java or Python agent should inherit from an abstract class `AbstractPlayer`, implement the constructor and three methods, `act`, `init` and `result`. The class must be named `Agent.java` or `Agent.py`.

`constructor` is called once per game and must finish in no more than `START_TIME` (by default $1s$) of CPU time, thus it is called once during the learning and validation phases of each game.

```
public class Agent extends utils.AbstractPlayer {

    /**
     * Set the format of serialised StateObservation to receive.
     * lastSsoType can be reset at anytime
     */
    public Types.LEARNING_SSO_TYPE lastSsoType
        = Types.LEARNING_SSO_TYPE.JSON;
```

Parameters for client (agent)		
Variable	Default value	Usage
`START_TIME`	1s	Time for agent's constructor
`INITIALIZATION_TIME`	1s	Time for `init()`
`ACTION_TIME`	40ms	Time for returning an action per tick
`ACTION_TIME_DISQ`	50ms	Threshold for disqualification per tick
`TOTAL_LEARNING_TIME`	5min	Time allowed for learning a game
`EXTRA_LEARNING_TIME`	1s	Extra learning time
`SOCKET_PORT`	8080	Socket port for communication

Parameters for server		
Variable	Default	Usage
`MAX_TIMESTEPS`	1000	Maximal game ticks a game can run
`VALIDATION_TIMES`	10	Game repeating time during validation

Table 2: The main parameters are defined and initialised in the class `CompetitionParameters.py` for the Python client or `CompetitionParameters.java` for the Java client, and `core.competition.CompetitionParameters.java` for the server.

```
    /**
     * Constructor
     * To be called at the start of the communication.
     * No game has been initialized yet.
     * Perform one-time setup here.
     */
    public Agent(){
        ......
    }
}
```

`init` is called at the beginning of every game playing. It should finish in no more than `INITIALIZATION_TIME` (by default 1s) of CPU time.

```
public class Agent extends utils.AbstractPlayer {
    ......

    /**
     * Public method to be called at the start of every level of a game.
     * Perform any level-entry initialization here.
     * @param sso Phase Observation of the current game.
     * @param elapsedTimer Timer (1s)
     */
    @Override
    public void init(SerializableStateObservation sso,
        ElapsedCpuTimer elapsedTimer) {
        ......

    }
    ......
}
```

`act` At each game tick of a game playing, is called and determines the next action of the agent within the prescribed CPU time `ACTION_TIME` (by default $40ms$). The possible actions are `ACTION_LEFT, ACTION_RIGHT, ACTION_UP, ACTION_DOWN, ACTION_USE` and `ACTION_NIL` (do nothing). The controller will be disqualified immediately if more than `ACTION_TIME_DISQ` (by default $50ms$) is taken. Otherwise, an `NIL` action (do nothing) is applied.

Remark: It's possible that in a game or at a game state, not all the actions listed above are available (legal) actions, but `ACTION_NIL` is always available.

```
public class Agent extends utils.AbstractPlayer {
    ......

    /**
     * Method used to determine the next move to be performed by the agent.
     * The agent can ABORT the current game at anytime.
     *
     * @param sso Observation of the current state of the game
     * @param elapsedTimer Timer (40ms)
     * @return The action to be performed by the agent.
     */
    @Override
    public Types.ACTIONS act(SerializableStateObservation sso,
        ElapsedCpuTimer elapsedTimer){
        ......

    }
    ......
}
```

`result` is called at the end of every game playing. It has no time limit, in other words, the controller doesn't get penalized for overspending other than the `TOTAL_LEARNING_TIME`. The controller can play with the time it spends on the `result` call to do more learning or to play more games.

```
public class Agent extends utils.AbstractPlayer {
    ......

    /**
     * Method used to perform actions in case of a game end.
     * This is the last thing called when a level is played.
     * Called at the end of every level.
     *
     * @param sso The current state observation of the game.
     * @param elapsedTimer Timer (5min + 1s)
     * @return The next level of the current game to be played.
     */
    @Override
    public int result(SerializableStateObservation sso,
        ElapsedCpuTimer elapsedTimer){
        ......
    }
    ......
}
```

Scenario	Return	Default

Return by `act`	`ACTION_LEFT`
	`ACTION_RIGHT`
	`ACTION_UP`
	`ACTION_DOWN`
	`ACTION_USE`
	`ACTION_NIL`
	`ABORT`
		`ACTION_NIL`

Return by `results` during learning phase 1	***Ignored***	1 and 2 after the first and second game playing respectively
Return by `results` during learning phase 2	0 1 2	Randomly chosen from $\{0, 1, 2\}$
Return by `results` during validation phase	***Ignored***	3 and 4 repeatly

Table 3: Legal messages for communication between server and client in different scenarios.

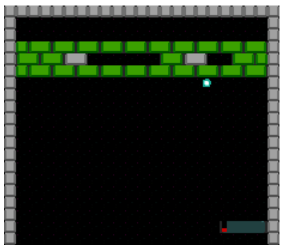

Figure 1: Example of a screenshot of the game screen.

Return At each call of `act` and `result`, an action or a level number should be returned. The legal returns for different scenarios are listed in Table 3.

State Observation At any scenario, the agent change select the format of game observation to receive at next game tick(s) using one of the follows

- `lastSsoType = Types.LEARNING_SSO_TYPE.JSON;` // for receiving JSON
- `lastSsoType = Types.LEARNING_SSO_TYPE.IMAGE;` // for receiving a screenshot of the game screen
- `lastSsoType = Types.LEARNING_SSO_TYPE.BOTH;` // for receiving both

The choice will be remembered until the agent makes another choice using the above commands. An example of the screenshot is given in Figure 1 and An example of the serialised state observation are given in Appendix A.

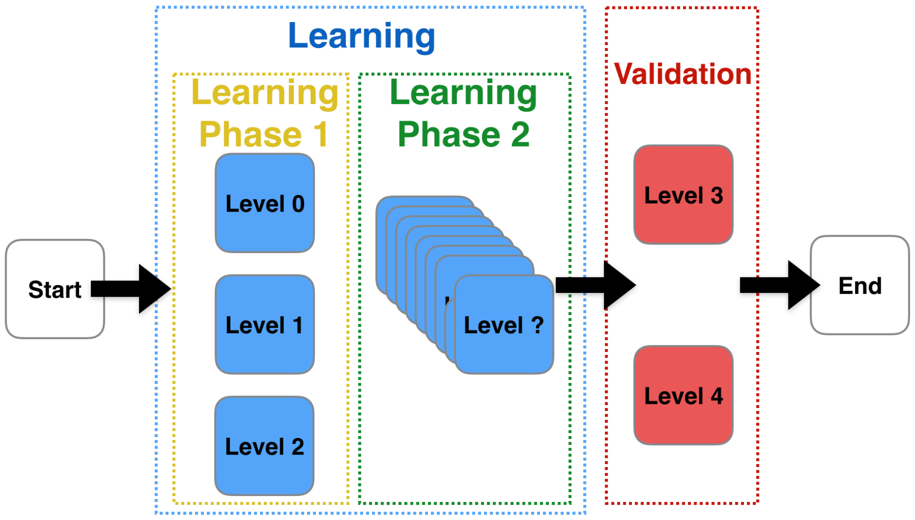

Figure 2: Learning and validation phase for one game in the single-player learning competition. In the competition, an agent will be executed on a set of (usually 10) unknown games, thus this process will repeat 10 times.

Termination A game playing terminates when the player wins / loses the game or the maximal game ticks (`MAX_TIMESTEPS`, 1,000 by default) is reached.

Time out If the agent returns an action after `ACTION_TIME` but no more than `ACTION_TIME_DISQ` (set to $50ms$ in the competition), then the action `ACTION_NIL` will be performed.

Disqualification If the agent returns an action after `ACTION_TIME_DISQ`, the agent is disqualified and loses the game.

4.3 Learning and validation phases

Each game has 5 levels, indexed by Level 0, Level 1, Level 2, Level 3 and Level 4 in Figure 2. Execution for one game in a set would have two phases: *learning phase* and *validation phase*.

The objective is to win a game as many time possible and get highest score possible during the *validation phase* (Levels 3 and 4).

4.3.1 Learning phase

An agent has a limited time `TOTAL_LEARNING_TIME` (by default $5min$) in total for both learning phases 1 and 2. The communication time is not included by `Timer`. In case that `TOTAL_LEARNING_TIME` has been used up, the results and observation of the game will still be sent to the agent and the agent will have no more than 1 *second* before the validation.

- *Learning phase 1*: The agent plays once Levels 0, 1 and 2 sequentially. At the end of each level, whether the game has terminated normally or the agent forces to terminate the game, the server will send the results of the (possibly unfinished) game to the agent.
- *Learning phase 2*: (Repeat until time up) After having finished *Learning phase 1*, the agent is free to select the next level to play (from Levels 0, 1 and 2) by calling the method `int result()` (detailed in Section 4.2). If the selected level id is not among 0, 1 and 2, then a random level id will be passed and a new game will start. This step is repeated until `TOTAL_LEARNING_TIME` has been used.

4.3.2 Validation phase

The agent repeats playing `VALIDATION_TIMES` times the Levels 3 and 4 sequentially. There is no more total time limit, but the agent respects the time limits for `init`, `act` and `result`, and can continue learning during the game playing.

4.4 Sample agents

Two Java sample agents are provided by Kamolwan Kunanusont (University of Essex, UK), named `kkunan` and `DontUnderestimateUchiha` in the directory `gvgai/clients/GVGAI-JavaClient/src/`. A Python sample agent is provided by Ercüment İlhan (Istanbul Technical University, Turkey), named `ercumentilhan` in the directory `gvgai/clients/GVGAI-PythonClient/src/`.

5 Single-player learning competition

5.1 Competition process

In each of the learning/validation/test game sets of the competition, 10 **unknown** games will be provided and each game has 5 levels, indexed by Level 0, Level 1, Level 2, Level 3 and Level 4 in Figure 2.

An agent has a limited time (`TOTAL_LEARNING_TIME`) in total for both learning phases 1 and 2 for each of the game. The communication time is not included by `Timer`. In case that `TOTAL_LEARNING_TIME` has been used up, the results and observation of the game will still be sent to the agent and the agent will have no more than 1 *second* before the validation.

The agent aims at maximising the average game score obtained during the validation phase over the 10 unknown games.

Figure 3: Training and test sets of games.

5.2 Evaluation of agents

Each agent will play on a set of unknown games (usually 10 games). The results on levels 3 and 4 during *validation phase* are used for evaluation, including

- the number of victories
- average score
- time spent (game ticks)

The submitted agents and sample agents are ranked and awarded with points: 25, 18, 15, 10, 8, 6, 4, 2, 1. The final ranking is determined by adding all points across all games.

All participants are invited to submit their agents to some given sets of games, observe the results, and re-submit some improved agents accordingly. When the submission is closed, no more update on submitted agents is possible. The current agents will play on another set of unknown games, called test set, following exactly the same *learning* and *validation phases* (cf. Figure 3).

5.3 Submission guidelines

The submission link can be found on the GVGAI website. Registration is required.

If submitting to the competition, the class of your agent should locate in a package with the same as the username you used to register to the website (this is in order to allow the server to run your controller when you submit), otherwise the package name does not matter.

Example If I have created an account named “jialin” for submitting a Java agent, then the agent should locate in `gvgai/clients/GVGAI-JavaClient/src/jialin` for local testing. You will submit a **zip** file named `jialin.zip`. After unzipping, a folder named `jialin` contained `Agent.java` should appear.

5.3.1 Java client

The package `gvgai/clients/GVGAI-JavaClient` contains all the necessary classes for client, including the communication via sockets and a sample random agent. You can create an agent by creating a class that inherits from `AbstractPlayer.java`. This class must be named `Agent.java`. During testing, the package can be located in `gvgai/clients/GVGAI-JavaClient/src`.

5.3.2 Python client

The package `gvgai/clients/GVGAI-PythonClient` contains the all the necessary classes for client, including the communication via sockets and a sample random agent. You can create an agent by creating a class that inherits from `AbstractPlayer.py`. This class must be named `Agent.py`. During testing, the package can be located in `gvgai/clients/GVGAI-PythonClient/src`.

6 Future work

The main work in the future is to provide a better learning agent which outperforms random playing.

If you see a bug or potential improvement in the framework, please email to jialin.liu@qmul.ac.uk with "[GVGAILearning]" in the subject of the email.

References

[1] Marc Ebner, John Levine, Simon M Lucas, Tom Schaul, Tommy Thompson, and Julian Togelius. Towards a video game description language. In *Dagstuhl Follow-Ups*, volume 6. Schloss Dagstuhl-Leibniz-Zentrum fuer Informatik, 2013.

[2] Jialin Liu. GVGAI Single-Player Learning Competition at IEEE CIG17. 2017.

[3] Diego Perez-Liebana, Spyridon Samothrakis, Julian Togelius, Tom Schaul, Simon M Lucas, Adrien Couëtoux, Jerry Lee, Chong-U Lim, and Tommy Thompson. The 2014 general video game playing competition. *IEEE Transactions on Computational Intelligence and AI in Games*, 8(3):229–243, 2016.

[4] Tom Schaul. A video game description language for model-based or interactive learning. In *Computational Intelligence in Games (CIG), 2013 IEEE Conference on*, pages 1–8. IEEE, 2013.

A Example of serialised game observation

```
SerializableStateObservation{
phase=ACT, isValidation=false, gameScore=8.0, gameTick=150, gameWinner=NO_WINNER, isGameOver=false,
worldDimension=[230.0, 200.0], blockSize=10, noOfPlayers=1, avatarSpeed=5.935588, avatarOrientation=[1.0, 0.0],
avatarPosition=[175.0, 180.0], avatarLastAction=ACTION_USE, avatarType=1, avatarHealthPoints=1,
avatarMaxHealthPoints=3, avatarLimitHealthPoints=3, isAvatarAlive=true,
availableActions=[ACTION_USE, ACTION_LEFT, ACTION_RIGHT], avatarResources={},
observationGrid={
Observation{category=6, itype=0, obsID=578, position=0.0 : 0.0, reference=-1.0 : -1.0, sqDist=2.0}
Observation{category=6, itype=0, obsID=624, position=0.0 : 10.0, reference=-1.0 : -1.0, sqDist=122.0}
Observation{category=6, itype=0, obsID=649, position=0.0 : 20.0, reference=-1.0 : -1.0, sqDist=442.0}
Observation{category=6, itype=0, obsID=674, position=0.0 : 30.0, reference=-1.0 : -1.0, sqDist=962.0}
Observation{category=6, itype=0, obsID=710, position=0.0 : 40.0, reference=-1.0 : -1.0, sqDist=1682.0}
Observation{category=6, itype=0, obsID=746, position=0.0 : 50.0, reference=-1.0 : -1.0, sqDist=2602.0}
Observation{category=6, itype=0, obsID=782, position=0.0 : 60.0, reference=-1.0 : -1.0, sqDist=3722.0}
Observation{category=6, itype=0, obsID=807, position=0.0 : 70.0, reference=-1.0 : -1.0, sqDist=5042.0}
Observation{category=6, itype=0, obsID=832, position=0.0 : 80.0, reference=-1.0 : -1.0, sqDist=6562.0}
Observation{category=6, itype=0, obsID=857, position=0.0 : 90.0, reference=-1.0 : -1.0, sqDist=8282.0}
Observation{category=6, itype=0, obsID=882, position=0.0 : 100.0, reference=-1.0 : -1.0, sqDist=10202.0}
Observation{category=6, itype=0, obsID=907, position=0.0 : 110.0, reference=-1.0 : -1.0, sqDist=12322.0}
Observation{category=6, itype=0, obsID=932, position=0.0 : 120.0, reference=-1.0 : -1.0, sqDist=14642.0}
Observation{category=6, itype=0, obsID=957, position=0.0 : 130.0, reference=-1.0 : -1.0, sqDist=17162.0}
Observation{category=6, itype=0, obsID=982, position=0.0 : 140.0, reference=-1.0 : -1.0, sqDist=19882.0}
Observation{category=6, itype=0, obsID=1007, position=0.0 : 150.0, reference=-1.0 : -1.0, sqDist=22802.0}
Observation{category=6, itype=0, obsID=1032, position=0.0 : 160.0, reference=-1.0 : -1.0, sqDist=25922.0}
Observation{category=6, itype=0, obsID=1057, position=0.0 : 170.0, reference=-1.0 : -1.0, sqDist=29242.0}
Observation{category=6, itype=0, obsID=1082, position=0.0 : 180.0, reference=-1.0 : -1.0, sqDist=32762.0}
Observation{category=6, itype=0, obsID=1108, position=0.0 : 190.0, reference=-1.0 : -1.0, sqDist=36482.0}
Observation{category=6, itype=0, obsID=580, position=10.0 : 0.0, reference=-1.0 : -1.0, sqDist=122.0}
Observation{category=6, itype=9, obsID=676, position=10.0 : 30.0, reference=-1.0 : -1.0, sqDist=1082.0}
Observation{category=6, itype=8, obsID=712, position=10.0 : 40.0, reference=-1.0 : -1.0, sqDist=1802.0}
Observation{category=6, itype=9, obsID=748, position=10.0 : 50.0, reference=-1.0 : -1.0, sqDist=2722.0}
Observation{category=6, itype=0, obsID=582, position=20.0 : 0.0, reference=-1.0 : -1.0, sqDist=442.0}
Observation{category=6, itype=8, obsID=678, position=20.0 : 30.0, reference=-1.0 : -1.0, sqDist=1402.0}
Observation{category=6, itype=8, obsID=750, position=20.0 : 50.0, reference=-1.0 : -1.0, sqDist=3042.0}
Observation{category=6, itype=0, obsID=584, position=30.0 : 0.0, reference=-1.0 : -1.0, sqDist=962.0}
Observation{category=6, itype=8, obsID=715, position=30.0 : 40.0, reference=-1.0 : -1.0, sqDist=2642.0}
Observation{category=6, itype=0, obsID=586, position=40.0 : 0.0, reference=-1.0 : -1.0, sqDist=1682.0}
Observation{category=6, itype=8, obsID=681, position=40.0 : 30.0, reference=-1.0 : -1.0, sqDist=2642.0}
Observation{category=6, itype=8, obsID=753, position=40.0 : 50.0, reference=-1.0 : -1.0, sqDist=4282.0}
Observation{category=6, itype=0, obsID=588, position=50.0 : 0.0, reference=-1.0 : -1.0, sqDist=2602.0}
Observation{category=6, itype=11, obsID=718, position=50.0 : 40.0, reference=-1.0 : -1.0, sqDist=4282.0}
Observation{category=6, itype=0, obsID=590, position=60.0 : 0.0, reference=-1.0 : -1.0, sqDist=3722.0}
Observation{category=6, itype=8, obsID=684, position=60.0 : 30.0, reference=-1.0 : -1.0, sqDist=4682.0}
Observation{category=6, itype=8, obsID=756, position=60.0 : 50.0, reference=-1.0 : -1.0, sqDist=6322.0}
Observation{category=6, itype=0, obsID=592, position=70.0 : 0.0, reference=-1.0 : -1.0, sqDist=5042.0}
Observation{category=6, itype=0, obsID=594, position=80.0 : 0.0, reference=-1.0 : -1.0, sqDist=6562.0}
Observation{category=6, itype=8, obsID=687, position=80.0 : 30.0, reference=-1.0 : -1.0, sqDist=7522.0}
Observation{category=6, itype=8, obsID=759, position=80.0 : 50.0, reference=-1.0 : -1.0, sqDist=9162.0}
Observation{category=6, itype=0, obsID=596, position=90.0 : 0.0, reference=-1.0 : -1.0, sqDist=8282.0}
Observation{category=6, itype=0, obsID=598, position=100.0 : 0.0, reference=-1.0 : -1.0, sqDist=10202.0}
Observation{category=6, itype=8, obsID=690, position=100.0 : 30.0, reference=-1.0 : -1.0, sqDist=11162.0}
Observation{category=6, itype=8, obsID=762, position=100.0 : 50.0, reference=-1.0 : -1.0, sqDist=12802.0}
Observation{category=6, itype=0, obsID=600, position=110.0 : 0.0, reference=-1.0 : -1.0, sqDist=12322.0}
Observation{category=6, itype=0, obsID=602, position=120.0 : 0.0, reference=-1.0 : -1.0, sqDist=14642.0}
Observation{category=6, itype=8, obsID=693, position=120.0 : 30.0, reference=-1.0 : -1.0, sqDist=15602.0}
Observation{category=6, itype=8, obsID=765, position=120.0 : 50.0, reference=-1.0 : -1.0, sqDist=17242.0}
Observation{category=6, itype=0, obsID=604, position=130.0 : 0.0, reference=-1.0 : -1.0, sqDist=17162.0}
Observation{category=6, itype=8, obsID=730, position=130.0 : 40.0, reference=-1.0 : -1.0, sqDist=18842.0}
Observation{category=6, itype=0, obsID=606, position=140.0 : 0.0, reference=-1.0 : -1.0, sqDist=19882.0}
Observation{category=6, itype=8, obsID=696, position=140.0 : 30.0, reference=-1.0 : -1.0, sqDist=20842.0}
Observation{category=6, itype=8, obsID=768, position=140.0 : 50.0, reference=-1.0 : -1.0, sqDist=22482.0}
Observation{category=6, itype=0, obsID=608, position=150.0 : 0.0, reference=-1.0 : -1.0, sqDist=22802.0}
Observation{category=6, itype=11, obsID=733, position=150.0 : 40.0, reference=-1.0 : -1.0, sqDist=24482.0}
Observation{category=6, itype=0, obsID=610, position=160.0 : 0.0, reference=-1.0 : -1.0, sqDist=25922.0}
Observation{category=6, itype=8, obsID=699, position=160.0 : 30.0, reference=-1.0 : -1.0, sqDist=26882.0}
Observation{category=6, itype=8, obsID=771, position=160.0 : 50.0, reference=-1.0 : -1.0, sqDist=28522.0}
Observation{category=6, itype=5, obsID=1139, position=165.0 : 60.0, reference=-1.0 : -1.0, sqDist=31277.0}
Observation{category=6, itype=0, obsID=612, position=170.0 : 0.0, reference=-1.0 : -1.0, sqDist=29242.0}
Observation{category=6, itype=5, obsID=1139, position=165.0 : 60.0, reference=-1.0 : -1.0, sqDist=31277.0}
Observation{category=0, itype=1, obsID=1093, position=175.0 : 180.0, reference=-1.0 : -1.0, sqDist=63737.0}
Observation{category=6, itype=0, obsID=614, position=180.0 : 0.0, reference=-1.0 : -1.0, sqDist=32762.0}
Observation{category=6, itype=8, obsID=702, position=180.0 : 30.0, reference=-1.0 : -1.0, sqDist=33722.0}
Observation{category=6, itype=8, obsID=774, position=180.0 : 50.0, reference=-1.0 : -1.0, sqDist=35362.0}
Observation{category=0, itype=1, obsID=1093, position=175.0 : 180.0, reference=-1.0 : -1.0, sqDist=63737.0}
Observation{category=6, itype=0, obsID=616, position=190.0 : 0.0, reference=-1.0 : -1.0, sqDist=36482.0}
Observation{category=6, itype=8, obsID=739, position=190.0 : 40.0, reference=-1.0 : -1.0, sqDist=38162.0}
Observation{category=6, itype=0, obsID=618, position=200.0 : 0.0, reference=-1.0 : -1.0, sqDist=40402.0}
Observation{category=6, itype=8, obsID=705, position=200.0 : 30.0, reference=-1.0 : -1.0, sqDist=41362.0}
Observation{category=6, itype=8, obsID=777, position=200.0 : 50.0, reference=-1.0 : -1.0, sqDist=43002.0}
Observation{category=6, itype=0, obsID=620, position=210.0 : 0.0, reference=-1.0 : -1.0, sqDist=44522.0}
Observation{category=6, itype=9, obsID=742, position=210.0 : 40.0, reference=-1.0 : -1.0, sqDist=46202.0}
Observation{category=6, itype=0, obsID=622, position=220.0 : 0.0, reference=-1.0 : -1.0, sqDist=48842.0}
Observation{category=6, itype=0, obsID=647, position=220.0 : 10.0, reference=-1.0 : -1.0, sqDist=48962.0}
Observation{category=6, itype=0, obsID=672, position=220.0 : 20.0, reference=-1.0 : -1.0, sqDist=49282.0}
Observation{category=6, itype=0, obsID=708, position=220.0 : 30.0, reference=-1.0 : -1.0, sqDist=49802.0}
Observation{category=6, itype=0, obsID=744, position=220.0 : 40.0, reference=-1.0 : -1.0, sqDist=50522.0}
```

```
Observation{category=6, itype=0, obsID=780, position=220.0 : 50.0, reference=-1.0 : -1.0, sqDist=51442.0}
Observation{category=6, itype=0, obsID=805, position=220.0 : 60.0, reference=-1.0 : -1.0, sqDist=52562.0}
Observation{category=6, itype=0, obsID=830, position=220.0 : 70.0, reference=-1.0 : -1.0, sqDist=53882.0}
Observation{category=6, itype=0, obsID=855, position=220.0 : 80.0, reference=-1.0 : -1.0, sqDist=55402.0}
Observation{category=6, itype=0, obsID=880, position=220.0 : 90.0, reference=-1.0 : -1.0, sqDist=57122.0}
Observation{category=6, itype=0, obsID=905, position=220.0 : 100.0, reference=-1.0 : -1.0, sqDist=59042.0}
Observation{category=6, itype=0, obsID=930, position=220.0 : 110.0, reference=-1.0 : -1.0, sqDist=61162.0}
Observation{category=6, itype=0, obsID=955, position=220.0 : 120.0, reference=-1.0 : -1.0, sqDist=63482.0}
Observation{category=6, itype=0, obsID=980, position=220.0 : 130.0, reference=-1.0 : -1.0, sqDist=66002.0}
Observation{category=6, itype=0, obsID=1005, position=220.0 : 140.0, reference=-1.0 : -1.0, sqDist=68722.0}
Observation{category=6, itype=0, obsID=1030, position=220.0 : 150.0, reference=-1.0 : -1.0, sqDist=71642.0}
Observation{category=6, itype=0, obsID=1055, position=220.0 : 160.0, reference=-1.0 : -1.0, sqDist=74762.0}
Observation{category=6, itype=0, obsID=1080, position=220.0 : 170.0, reference=-1.0 : -1.0, sqDist=78082.0}
Observation{category=6, itype=0, obsID=1106, position=220.0 : 180.0, reference=-1.0 : -1.0, sqDist=81602.0}
Observation{category=6, itype=0, obsID=1131, position=220.0 : 190.0, reference=-1.0 : -1.0, sqDist=85322.0}
}, NPCPositions=null, immovablePositions=null, movablePositions={
Observation{category=6, itype=0, obsID=578, position=0.0 : 0.0, reference=-1.0 : -1.0, sqDist=2.0}
Observation{category=6, itype=0, obsID=580, position=10.0 : 0.0, reference=-1.0 : -1.0, sqDist=122.0}
Observation{category=6, itype=0, obsID=582, position=20.0 : 0.0, reference=-1.0 : -1.0, sqDist=442.0}
Observation{category=6, itype=0, obsID=584, position=30.0 : 0.0, reference=-1.0 : -1.0, sqDist=962.0}
Observation{category=6, itype=0, obsID=586, position=40.0 : 0.0, reference=-1.0 : -1.0, sqDist=1682.0}
Observation{category=6, itype=0, obsID=588, position=50.0 : 0.0, reference=-1.0 : -1.0, sqDist=2602.0}
Observation{category=6, itype=0, obsID=590, position=60.0 : 0.0, reference=-1.0 : -1.0, sqDist=3722.0}
Observation{category=6, itype=0, obsID=592, position=70.0 : 0.0, reference=-1.0 : -1.0, sqDist=5042.0}
Observation{category=6, itype=0, obsID=594, position=80.0 : 0.0, reference=-1.0 : -1.0, sqDist=6562.0}
Observation{category=6, itype=0, obsID=596, position=90.0 : 0.0, reference=-1.0 : -1.0, sqDist=8282.0}
Observation{category=6, itype=0, obsID=598, position=100.0 : 0.0, reference=-1.0 : -1.0, sqDist=10202.0}
Observation{category=6, itype=0, obsID=600, position=110.0 : 0.0, reference=-1.0 : -1.0, sqDist=12322.0}
Observation{category=6, itype=0, obsID=602, position=120.0 : 0.0, reference=-1.0 : -1.0, sqDist=14642.0}
Observation{category=6, itype=0, obsID=604, position=130.0 : 0.0, reference=-1.0 : -1.0, sqDist=17162.0}
Observation{category=6, itype=0, obsID=606, position=140.0 : 0.0, reference=-1.0 : -1.0, sqDist=19882.0}
Observation{category=6, itype=0, obsID=608, position=150.0 : 0.0, reference=-1.0 : -1.0, sqDist=22802.0}
Observation{category=6, itype=0, obsID=610, position=160.0 : 0.0, reference=-1.0 : -1.0, sqDist=25922.0}
Observation{category=6, itype=0, obsID=612, position=170.0 : 0.0, reference=-1.0 : -1.0, sqDist=29242.0}
Observation{category=6, itype=0, obsID=614, position=180.0 : 0.0, reference=-1.0 : -1.0, sqDist=32762.0}
Observation{category=6, itype=0, obsID=616, position=190.0 : 0.0, reference=-1.0 : -1.0, sqDist=36482.0}
Observation{category=6, itype=0, obsID=618, position=200.0 : 0.0, reference=-1.0 : -1.0, sqDist=40402.0}
Observation{category=6, itype=0, obsID=620, position=210.0 : 0.0, reference=-1.0 : -1.0, sqDist=44522.0}
Observation{category=6, itype=0, obsID=622, position=220.0 : 0.0, reference=-1.0 : -1.0, sqDist=48842.0}
Observation{category=6, itype=0, obsID=624, position=0.0 : 10.0, reference=-1.0 : -1.0, sqDist=122.0}
Observation{category=6, itype=0, obsID=647, position=220.0 : 10.0, reference=-1.0 : -1.0, sqDist=48962.0}
Observation{category=6, itype=0, obsID=649, position=0.0 : 20.0, reference=-1.0 : -1.0, sqDist=442.0}
Observation{category=6, itype=0, obsID=672, position=220.0 : 20.0, reference=-1.0 : -1.0, sqDist=49282.0}
Observation{category=6, itype=0, obsID=674, position=0.0 : 30.0, reference=-1.0 : -1.0, sqDist=962.0}
Observation{category=6, itype=0, obsID=708, position=220.0 : 30.0, reference=-1.0 : -1.0, sqDist=49802.0}
Observation{category=6, itype=0, obsID=710, position=0.0 : 40.0, reference=-1.0 : -1.0, sqDist=1682.0}
Observation{category=6, itype=0, obsID=744, position=220.0 : 40.0, reference=-1.0 : -1.0, sqDist=50522.0}
Observation{category=6, itype=0, obsID=746, position=0.0 : 50.0, reference=-1.0 : -1.0, sqDist=2602.0}
Observation{category=6, itype=0, obsID=780, position=220.0 : 50.0, reference=-1.0 : -1.0, sqDist=51442.0}
Observation{category=6, itype=0, obsID=782, position=0.0 : 60.0, reference=-1.0 : -1.0, sqDist=3722.0}
Observation{category=6, itype=0, obsID=805, position=220.0 : 60.0, reference=-1.0 : -1.0, sqDist=52562.0}
Observation{category=6, itype=0, obsID=807, position=0.0 : 70.0, reference=-1.0 : -1.0, sqDist=5042.0}
Observation{category=6, itype=0, obsID=830, position=220.0 : 70.0, reference=-1.0 : -1.0, sqDist=53882.0}
Observation{category=6, itype=0, obsID=832, position=0.0 : 80.0, reference=-1.0 : -1.0, sqDist=6562.0}
Observation{category=6, itype=0, obsID=855, position=220.0 : 80.0, reference=-1.0 : -1.0, sqDist=55402.0}
Observation{category=6, itype=0, obsID=857, position=0.0 : 90.0, reference=-1.0 : -1.0, sqDist=8282.0}
Observation{category=6, itype=0, obsID=880, position=220.0 : 90.0, reference=-1.0 : -1.0, sqDist=57122.0}
Observation{category=6, itype=0, obsID=882, position=0.0 : 100.0, reference=-1.0 : -1.0, sqDist=10202.0}
Observation{category=6, itype=0, obsID=905, position=220.0 : 100.0, reference=-1.0 : -1.0, sqDist=59042.0}
Observation{category=6, itype=0, obsID=907, position=0.0 : 110.0, reference=-1.0 : -1.0, sqDist=12322.0}
Observation{category=6, itype=0, obsID=930, position=220.0 : 110.0, reference=-1.0 : -1.0, sqDist=61162.0}
Observation{category=6, itype=0, obsID=932, position=0.0 : 120.0, reference=-1.0 : -1.0, sqDist=14642.0}
Observation{category=6, itype=0, obsID=955, position=220.0 : 120.0, reference=-1.0 : -1.0, sqDist=63482.0}
Observation{category=6, itype=0, obsID=957, position=0.0 : 130.0, reference=-1.0 : -1.0, sqDist=17162.0}
Observation{category=6, itype=0, obsID=980, position=220.0 : 130.0, reference=-1.0 : -1.0, sqDist=66002.0}
Observation{category=6, itype=0, obsID=982, position=0.0 : 140.0, reference=-1.0 : -1.0, sqDist=19882.0}
Observation{category=6, itype=0, obsID=1005, position=220.0 : 140.0, reference=-1.0 : -1.0, sqDist=68722.0}
Observation{category=6, itype=0, obsID=1007, position=0.0 : 150.0, reference=-1.0 : -1.0, sqDist=22802.0}
Observation{category=6, itype=0, obsID=1030, position=220.0 : 150.0, reference=-1.0 : -1.0, sqDist=71642.0}
Observation{category=6, itype=0, obsID=1032, position=0.0 : 160.0, reference=-1.0 : -1.0, sqDist=25922.0}
Observation{category=6, itype=0, obsID=1055, position=220.0 : 160.0, reference=-1.0 : -1.0, sqDist=74762.0}
Observation{category=6, itype=0, obsID=1057, position=0.0 : 170.0, reference=-1.0 : -1.0, sqDist=29242.0}
Observation{category=6, itype=0, obsID=1080, position=220.0 : 170.0, reference=-1.0 : -1.0, sqDist=78082.0}
Observation{category=6, itype=0, obsID=1082, position=0.0 : 180.0, reference=-1.0 : -1.0, sqDist=32762.0}
Observation{category=6, itype=0, obsID=1106, position=220.0 : 180.0, reference=-1.0 : -1.0, sqDist=81602.0}
Observation{category=6, itype=0, obsID=1108, position=0.0 : 190.0, reference=-1.0 : -1.0, sqDist=36482.0}
Observation{category=6, itype=0, obsID=1131, position=220.0 : 190.0, reference=-1.0 : -1.0, sqDist=85322.0}
Observation{category=6, itype=5, obsID=1139, position=165.0 : 60.0, reference=-1.0 : -1.0, sqDist=31277.0}
Observation{category=6, itype=8, obsID=678, position=20.0 : 30.0, reference=-1.0 : -1.0, sqDist=1402.0}
Observation{category=6, itype=8, obsID=681, position=40.0 : 30.0, reference=-1.0 : -1.0, sqDist=2642.0}
Observation{category=6, itype=8, obsID=684, position=60.0 : 30.0, reference=-1.0 : -1.0, sqDist=4682.0}
Observation{category=6, itype=8, obsID=687, position=80.0 : 30.0, reference=-1.0 : -1.0, sqDist=7522.0}
Observation{category=6, itype=8, obsID=690, position=100.0 : 30.0, reference=-1.0 : -1.0, sqDist=11162.0}
Observation{category=6, itype=8, obsID=693, position=120.0 : 30.0, reference=-1.0 : -1.0, sqDist=15602.0}
Observation{category=6, itype=8, obsID=696, position=140.0 : 30.0, reference=-1.0 : -1.0, sqDist=20842.0}
Observation{category=6, itype=8, obsID=699, position=160.0 : 30.0, reference=-1.0 : -1.0, sqDist=26882.0}
Observation{category=6, itype=8, obsID=702, position=180.0 : 30.0, reference=-1.0 : -1.0, sqDist=33722.0}
Observation{category=6, itype=8, obsID=705, position=200.0 : 30.0, reference=-1.0 : -1.0, sqDist=41362.0}
Observation{category=6, itype=8, obsID=712, position=10.0 : 40.0, reference=-1.0 : -1.0, sqDist=1802.0}
Observation{category=6, itype=8, obsID=715, position=30.0 : 40.0, reference=-1.0 : -1.0, sqDist=2642.0}
```

```
Observation{category=6, itype=8, obsID=730, position=130.0 : 40.0, reference=-1.0 : -1.0, sqDist=18842.0}
Observation{category=6, itype=8, obsID=739, position=190.0 : 40.0, reference=-1.0 : -1.0, sqDist=38162.0}
Observation{category=6, itype=8, obsID=750, position=20.0 : 50.0, reference=-1.0 : -1.0, sqDist=3042.0}
Observation{category=6, itype=8, obsID=753, position=40.0 : 50.0, reference=-1.0 : -1.0, sqDist=4282.0}
Observation{category=6, itype=8, obsID=756, position=60.0 : 50.0, reference=-1.0 : -1.0, sqDist=6322.0}
Observation{category=6, itype=8, obsID=759, position=80.0 : 50.0, reference=-1.0 : -1.0, sqDist=9162.0}
Observation{category=6, itype=8, obsID=762, position=100.0 : 50.0, reference=-1.0 : -1.0, sqDist=12802.0}
Observation{category=6, itype=8, obsID=765, position=120.0 : 50.0, reference=-1.0 : -1.0, sqDist=17242.0}
Observation{category=6, itype=8, obsID=768, position=140.0 : 50.0, reference=-1.0 : -1.0, sqDist=22482.0}
Observation{category=6, itype=8, obsID=771, position=160.0 : 50.0, reference=-1.0 : -1.0, sqDist=28522.0}
Observation{category=6, itype=8, obsID=774, position=180.0 : 50.0, reference=-1.0 : -1.0, sqDist=35362.0}
Observation{category=6, itype=8, obsID=777, position=200.0 : 50.0, reference=-1.0 : -1.0, sqDist=43002.0}
Observation{category=6, itype=9, obsID=676, position=10.0 : 30.0, reference=-1.0 : -1.0, sqDist=1082.0}
Observation{category=6, itype=9, obsID=742, position=210.0 : 40.0, reference=-1.0 : -1.0, sqDist=46202.0}
Observation{category=6, itype=9, obsID=748, position=10.0 : 50.0, reference=-1.0 : -1.0, sqDist=2722.0}
Observation{category=6, itype=11, obsID=718, position=50.0 : 40.0, reference=-1.0 : -1.0, sqDist=4282.0}
Observation{category=6, itype=11, obsID=733, position=150.0 : 40.0, reference=-1.0 : -1.0, sqDist=24482.0}
}, resourcesPositions=null, portalsPositions=null, fromAvatarSpritesPositions=null}
```

B Example of game output

```
Listening for transport dt_socket at address: 8888
[GAME] Game idx:119
[PHASE] Starting First Phase of Training in 3 levels.
[TRAINING] Result (1->win; 0->lose): level:0, Player0:1, Player0-Score:1.0, timesteps:313
[TRAINING] Result (1->win; 0->lose): level:1, Player0:0, Player0-Score:0.0, timesteps:0
[TRAINING] Result (1->win; 0->lose): level:2, Player0:0, Player0-Score:0.0, timesteps:22
[PHASE] Starting Second Phase of Training in 3 levels.
[TRAINING] Result (1->win; 0->lose): level:0, Player0:1, Player0-Score:1.0, timesteps:509
[TRAINING] Result (1->win; 0->lose): level:2, Player0:0, Player0-Score:0.0, timesteps:31
[TRAINING] Result (1->win; 0->lose): level:0, Player0:1, Player0-Score:1.0, timesteps:462
......
[PHASE] Starting Validation in 2 levels.
[VALIDATION] Result (1->win; 0->lose): level:3, Player0:0, Player0-Score:0.0, timesteps:0
[VALIDATION] Result (1->win; 0->lose): level:4, Player0:0, Player0-Score:0.0, timesteps:232
[VALIDATION] Result (1->win; 0->lose): level:3, Player0:0, Player0-Score:0.0, timesteps:0
[VALIDATION] Result (1->win; 0->lose): level:4, Player0:0, Player0-Score:0.0, timesteps:229
[VALIDATION] Result (1->win; 0->lose): level:3, Player0:0, Player0-Score:0.0, timesteps:15
[VALIDATION] Result (1->win; 0->lose): level:4, Player0:0, Player0-Score:0.0, timesteps:47
[VALIDATION] Result (1->win; 0->lose): level:3, Player0:0, Player0-Score:0.0, timesteps:23
[VALIDATION] Result (1->win; 0->lose): level:4, Player0:0, Player0-Score:0.0, timesteps:199
[VALIDATION] Result (1->win; 0->lose): level:3, Player0:0, Player0-Score:0.0, timesteps:31
[VALIDATION] Result (1->win; 0->lose): level:4, Player0:0, Player0-Score:0.0, timesteps:0
[VALIDATION] Result (1->win; 0->lose): level:3, Player0:0, Player0-Score:0.0, timesteps:33
[VALIDATION] Result (1->win; 0->lose): level:4, Player0:0, Player0-Score:0.0, timesteps:95
[VALIDATION] Result (1->win; 0->lose): level:3, Player0:0, Player0-Score:0.0, timesteps:0
[VALIDATION] Result (1->win; 0->lose): level:4, Player0:0, Player0-Score:0.0, timesteps:0
[VALIDATION] Result (1->win; 0->lose): level:3, Player0:1, Player0-Score:1.0, timesteps:112
[VALIDATION] Result (1->win; 0->lose): level:4, Player0:0, Player0-Score:0.0, timesteps:148
[VALIDATION] Result (1->win; 0->lose): level:3, Player0:0, Player0-Score:0.0, timesteps:0
[VALIDATION] Result (1->win; 0->lose): level:4, Player0:0, Player0-Score:0.0, timesteps:0
[VALIDATION] Result (1->win; 0->lose): level:3, Player0:0, Player0-Score:0.0, timesteps:98
[VALIDATION] Result (1->win; 0->lose): level:4, Player0:0, Player0-Score:0.0, timesteps:939
[PHASE] End Validation in 2 levels.

         --> Real execution time: 1 minutes, 27 seconds of wall time.
```