Hex Game Engine Documentation

COMP34111 AI & Games

1. Submission checklist

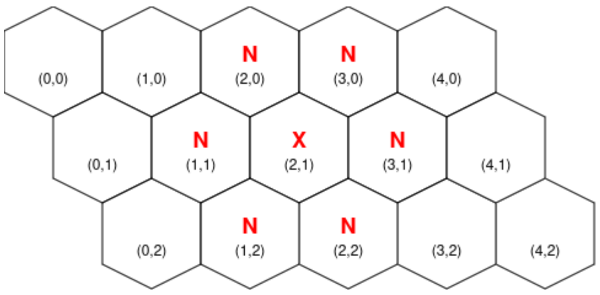
This is provided to help you check you have included everything in your submission. Make sure you test your submission and read this documentation carefully. Pay particular attention to the Docker instructions to ensure that your code will run in the test environment.

1. Correctly formatted cmd.txt file specifying your Python agent (see section Section 4.3)
2. A Python class implementing `AgentBase` - either your own implementation or the `ExternalAgent` (see Section 3)
3. Any other files required to run your agent in the Docker container (see section Section 4.1.2), such as extra Python files, your agent implementation in an alternative programming language, and compiled files if relevant (e.g. .class files for Java). **We will not compile your code, you should do this before submission in the specified Docker environment and include both the source code and the compiled files in your submission.**

2. General Information

- The rules for Hex can be found here.
- The board is 11x11 and it is represented by a 0-indexed two-dimensional matrix with rows that fall diagonally to the right. That is, the position (x, y) has neighbours left $(x - 1, y)$, right $(x + 1, y)$, above $(x, y - 1)$, below $(x, y + 1)$ and diagonally $(x - 1, y + 1)$ and $(x + 1, y - 1)$. Below you can see the neighbours of tile X marked as N:

	0	1	2	3	4
a(0)	0	0	N	N	0
b(1)	0	N	X	N	0
c(2)	0	N	N	0	0



- Red aims to connect top and bottom sides, while Blue connects horizontally. Red moves first.
- The pie rule is implemented
- Each agent has a maximum total of 5 minutes per match
- Your agent's performance will be evaluated by a combination of win rate (75%) and move speed (25%).
- Possible outcome of a match:
  - **Win** – one agent has connected their sides of the board
  - **Timeout** – one agent has used up all the time allocated to it (5 minutes). The opposing agent wins.
  - **Illegal move** – one agent tries to occupy an already occupied tile, play a move outside the board or does not return a `Move` object. The opposing agent wins.

3. Implementation

3.1. Overview

You have the option to implement your agent in Python, or in an alternative programming language of your choice. You can either implement the provided agent template class in Python, or use the provided `agents/DefaultAgents/ExternalAgent.py` class to launch your agent as an external process. More detailed instructions on how to do this are provided later, with an example implementation in Java.

3.2. Python agent

An agent template is provided at `agents/DefaultAgents/NaiveAgent.py`. The class has two methods, `__init__` and `make_move`.

When the game starts, the engine will call the `__init__` method in your Python agent, which will tell you which colour your agent will be playing as and allow you to perform any setup steps.

When it is your turn to move, the game engine will call the `make_move` method, which should always return a `Move` object. If the method is not a valid `Move` object, the game engine will classify the move as illegal, and your agent will lose the game immediately. When the `make_move` method is called, a `Board` object specifying the current game state will be provided, as well as a `Move` object specifying the opponent's move. The `Board` object contains a 2D list of `Tile` objects. `Tile` objects have `x` and `y` properties which match their position in the list returned by `Board.get_tiles()`.

The `Move` object can be used to make a swap move or a normal move. To make a swap move, the `Move` object is created with the parameters `(-1, -1)`. To make a normal move, the `Move` object is created with the parameters `(x, y)`, where x and y are the coordinates the tile will be placed at.

The agent's only implementation requirement is it must be a class that inherits the `AgentBase` class (located at `src/AgentBase.py`). `AgentBase` is an abstract base class that ensures you implement both the `__init__` and `make_move` methods yourself, and provides all the necessary methods for running a game.

3.3. Alternative languages

The provided `agents/DefaultAgents/ExternalAgent.py` class contains code to launch an external Java process and communicate with it via stdin and stdout to play the game. There is a corresponding `agents/DefaultAgents/NaiveAgent.java` file which demonstrates the communication for a Java agent implementation. You are free to use or adapt this example code however you like; your ability to write an interface between Python and another programming language is not part of the assessment for this module. You should be able to use this class to run an external agent in any programming language that can communicate with stdin and stdout by modifying lines 24-25 in the `__init__` method:

```
self.agent_process = Popen(["java", "-cp", "agents/DefaultAgents", "NaiveAgent",
colour.get_char(), "11"], stdout=PIPE, stdin=PIPE, text=True)
```

The provided example is equivalent to running the command `java -cp agents/DefaultAgents NaiveAgent B "11"` in the terminal, assuming the agent has been assigned the colour Blue. The `-cp agents/DefaultAgents` part of the command specifies the location of the compiled NaiveAgent.class file.

If using the provided ExternalAgent code, when the game is started your code will be launched using the command specified in the `__init__` method. On each of your agent's turns, a message will be sent via your process' stdin in the format described below, and the game will wait for a response via your process' stdout. Each message sent by the game will end in a newline (`\n`). Your response should consist of one line, also ending in a newline.

3.3.1. Message format

The basic message format that the game will send to your agent is "`COMMAND;MOVE;BOARD;TURN;\n`". Your agent should respond with "`x,y\n`", where x and y are the coordinates of the tile you are changing. To make a swap move, x and y should both be set to −1.

`COMMAND` can take the values `START`, `SWAP`, or `CHANGE`. `START` is only used on the first turn of the game - i.e. you will only receive a `START` message if you are Red. `SWAP` indicates that the opponent has chosen to swap colours. The `SWAP` move is only allowed by player 2 (red) on their first turn - i.e. only on turn 2 of the game. `CHANGE` specifies a tile that the opponent has changed to their colour. `MOVE` will be blank for `START` and `SWAP` messages, `BOARD` and `TURN` will always be specified.

`MOVE` takes the value of the (x,y) coordinates of the tile the opponent has changed when they make a `CHANGE` move.

`BOARD` specifies the state of the board at the beginning of your turn, with each tile taking the value 0, R or B, and each row separated by a comma. For example, a 2x2 board on turn 1 would be given as "00,00"

`TURN` is the turn number for the move you are about to make, starting from 1.

Example commands:

Command	Explanation
`START;;000,000,000;1;\n`	Request for the opening move of a game using a 3x3 board
`SWAP;;0R0,000,000;3;\n`	The opponent has swapped colours on turn 2 and the player agent is about to make move 3
`CHANGE;0,1;000,R00,000;2;\n`	The opponent changed tile (0,1) to red as their opening move

Example responses:

Response	Explanation
`1,2\n`	Change the tile with the coordinates (1,2) to your colour
`-1,-1\n`	Make a swap move

4. Running the game

4.1. Basics

4.1.1. From Terminal

The engine requires `Python >= 3.10`.

Example commands:

Command	Explanation
`python3 Hex.py`	Run the game engine with two default agents
`python3 Hex.py -p1 "agents.Group0.MyAgent MyGoodAgent"`	Run the game engine with player 1 as the class MyGoodAgent in the file agents/Group0/MyAgent, and player 2 as default agent

Command	Explanation
`python3 Hex.py -p2 "agents.Group0.MyAgent MyGoodAgent"`	Run the game engine with player 2 as the class MyGoodAgent in the file agents/Group0/MyAgent, and player 1 as default agent
`python3 Hex.py -p1 "agents.Group0.MyOtherAgent MyBadAgent" -p2 "agents.Group0.MyAgent MyGoodAgent"`	Run the game engine with player 1 as the class MyBadAgent from the file agents/Group0/MyOtherAgent and player 2 as the class MyGoodAgent in the file agents/Group0/MyAgent

This will play a game between two Naive agents, located at `agents/DefaultAgents/NaiveAgent.py` with the class name `NaiveAgent`. By default game log will also be printed to `stderr`. The details of the log formating will be explained in the next section.

To see all the available options and arguments of the engine use `python3 Hex.py --help` , or see the table below: (Note that quotes must be used when specifying arguments which contain spaces, i.e. the -p1 and -p2 arguments.)

Argument	Usage
`-p1` `--player1`	Specify the player 1 agent format: agents.GroupX.AgentFile AgentClassName E.g. "agents.Group0.NaiveAgent NaiveAgent"
`-p1Name` `--player1Name`	Specify the player 1 name
`-p2` `--player2`	Specify the player 2 agent format: agents.GroupX.AgentFile AgentClassName E.g. "agents.Group0.NaiveAgent NaiveAgent"
`-p2Name` `--player2Name`	Specify the player 2 name
`-v` `--verbose`	Enable verbose logging
`-b` `--board_size`	Specify the board size*. E.g. -b 11
`-l` `--log`	Save moves history to a log file, if the flag is present, the result will be saved to game.log. If a filename is provided, the result will be saved to the provided file. If the flag is not present, the result will be printed to the console, via stderr.

* Your agent will be tested on an 11x11 board, this option is provided for development.

4.1.2. Docker

We will run each game within a specific Dockerfile image provided in the repository and with this specific Docker environment. This means you can test your agent in the exact environment it will be tested in to ensure compatibility. If using a compiled programming language you should compile your code within the specified Docker environment when preparing your submission, this will ensure that your compiled code runs correctly when we test it.

To build the Dockerfile run:

```
docker build --build-arg UID=$UID -t hex .
```

The building process will take a while. It took 138s to build on an AMD 7950x CPU.

To run the container use:

```
docker run --cpus=8 --memory=8G -v `pwd`:/home/hex --name hex -it hex /bin/bash
```

After you are in the container, all the steps from Section 4.1.1 apply.

The current repo will be mapped to `/home/hex` within the container. If you `cd hex` you should be able to see all your local files. Any changes made to that directory will be reflected in your system directory. This will be the command we use to create the running environment for playing each game, so your agent can at most have 8 CPUs and 8 GB of memory.

To exit the docker container, you can do `exit`. This will stop the container.

To enter the container again you can use:

```
docker start -i hex
```

4.2. Log format

The game log is a CSV file. The following is an example of a log generated by running the engine with two default agents by using the command `python3 Hex.py -l`.

```csv
1   1,Alice,RED(x=7, y=4),4125
2   2,Bob,REDSWAP(),1584
3   3,Alice,BLUE(x=3, y=4),6542
4   4,Bob,RED(x=1, y=2),3293
5   5,Alice,BLUE(x=10, y=1),7917
6   6,Bob,RED(x=0, y=5),4793
7   7,Alice,BLUE(x=0, y=3),8917
8   8,Bob,RED(x=5, y=9),5710
9   9,Alice,BLUE(x=1, y=4),12126
10  Bob,11376
11  Alice,12126
12  winner,Alice,BAD_MOVE
```

Between line 1-9 is the move each agent played. Each column represents the turn number, player name, type of move, and playtime in nano-seconds, respectively. As a result, line 1 means, "At turn 1, player Alice holding red and played at $(x = 7, y = 4)$ taking $4125ns$ to make the move". Line 2 means, "Bob played the swap move and took 1584 ns".

The last lines show the total time spent by a player on the red side, then the blue side, and lastly, who the winner is and what the winning method is.

As mentioned, by default, the engine will output the result to `stderr`. Using `-l` will write the log to `game.log` in the directory where you start the engine. Each time, the file will be overwritten. You can save the log to another file by providing a file path, for example, `-l ./test/test.log`.

4.3. cmd.txt

You will specify the path to your agent file using a file named `cmd.txt`, which will be included within your group folder. For example, if your agent is located at `agents/Group999/NaiveAgent.py` and the class name is `VeryNaiveAgent`, then the content of the `cmd.txt` should be the following:

```
agents.Group999.NaiveAgent VeryNaiveAgent
```

Note that this is only the location and name of your agent class, and not any other parts of the command to launch the game engine. **It is very important that cmd.txt only includes the file path and the class name, and not any additional text.** No messages, no instructions - this text will be extracted from the file automatically during the marking process, so if you add any additional text to the file your code may fail to run.

The string will be the calling argument for performing the agent import, so the format needs to be precise. There should be only one white space between the agent path and the agent class name. If you wrote your agent in an alternative programming language, make sure to specify the **Python** agent in cmd.txt; you will launch your code from within this Python class as explained in section Section 3.3 Alternative languages.