Hex Game Engine Documentation

COMP34121 AI & Games

1. General information

- The rules for Hex can be found here. You can also play the game online here (with an AI visualiser), and for a downloadable version with resizable board check 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 above *(x-1,y)*, below *(x+1,y)*, left *(x,y-1)*, right *(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
0 0 0 N N 0        0 0 N N 0
1 0 N X N 0         0 N X N 0
2 0 N N 0 0          0 N N 0 0
```

- Red aims to connect top and bottoms 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 outcomes of a match:

- *Win* – one agent has connected their sides of the board
- *Timeout* – one agent has used up all the time allotted to it (5 minutes), failed to connect, or it has disconnected. The opposing agent wins.
- *Illegal move* – one agent has sent a message with disallowed text, range outside the available board, or on a tile that is already occupied. The opposing agent wins.

2. Game protocol

2.1 Informal description

At the start of the game, the engine will signal to the agent the size of the board and which colour it should assume by the message *START;n;<R|B>\n*.

The agent will specify its moves as *SWAP\n* in case of a swap, or *MOVE;<X>,<Y>\n* for a normal move, where *<X>* and *<Y>* are the 0-indexed row and column, respectively. After each move, the engine will notify **both** agents of the new state of the board as *CHANGE;<X,Y|SWAP>;<B>;<N>\n*, where *<B>* is the board and *<N>* is the colour to move next. In case of a swap, it is the agents' responsibility to assume their new roles, i.e. for the second agent to respond to Red calls and the first agent to behave as Blue.

The engine can, at any time, send the message *END;<R|B>*, marking the end of the game and the winner. If the game proceeds normally, this message will appear after a *CHANGE;…;END* message. Otherwise, *END* will be prompted by an illegal move or a timeout. In both cases, the opposing agent will be declared winner. After this, the engine will not send any other message and the agents should terminate.

2.2 Extended description

The following description of the protocol is in Backus-Naur form, with a few additions:

- Parentheses *()* denote a group of elements.
- Asterisk *** represents a repetition of the preceding group an arbitrary number of times (including 0).
- Asterisk followed by a number **n* indicates n-fold repetition of the preceding element.

All messages are of the form *<TYPE> (";" <ARGUMENT>)* "\n"*, where:

Argument	Can become
TYPE	"START"|"CHANGE"|"END"|"MOVE"|"SWAP"
ARGUMENT	<n>|<COLOUR>|<TURN>|<MOVE>|<ACTION>|<TILE>|<BOARD>
n	positive integer
COLOUR	"R"|"B"
TURN	<COLOUR>|"END"
MOVE	<X1> "," <X2>
X	nonnegative integer < n
ACTION	<MOVE>|"SWAP"
TILE	<COLOUR>|"0"
BOARD	(<TILE>*n ",")*(n-1) <TILE>*n

The engine sends the following messages: (?) and receives

- *"START" ";" n ";" <COLOUR> "\n"*

  This message notifies the agent of the start of the game, the size of the board, and the colour that it should assume.

  Note: R denotes Red and B denotes Blue.

- *"CHANGE" ";" <ACTION> ";" <BOARD> ";" <TURN> "\n"*

  Informs the agent about the new state of the board. The new action and the new state are both given so that the agent can be implemented to work either with boards or with matches.

  Note: *<BOARD>* is a comma-separated list of n n-length strings that represents the board. For each tile, R is Red, B is Blue and 0 is Empty. e.g. for b=2, the board will initially be *00,00* and its end state could be *BR,RR*.

  Note: *<TURN>* denotes the next player, or *END* if the last move was a winning move.

- *"END" ";" <COLOUR> "\n"*

  Notifies the agent about the end of the game and the winner.

  Note: There are three possible end conditions:

  - An agent has connected their sides of the board. In this case, *END* follows a *CHANGE;…;END* message.
  - An agent has sent an illegal message. The opposing agent wins.
  - An agent has timed out or disconnected. The opposing agent wins.

3 The engine

3.1 How do I run the engine?

The engine is run using the command *python3 Hex.py <ARGUMENTS>*. If fewer than two agents are specified, the missing agents will be replaced by the default agent. The first agent in order of writing will be Red and the second will be Blue (this can be overridden with the argument *-s*). Use this command with no agents and the arguments *-p b=2* for a better understanding of the protocol and the progress of a game.

Regardless of the arguments, some results will be printed to *stderr*. These are interpreted as follows:

- **Line 1:** Win | Illegal move | Timeout
- **Line 2:** Has player 1 won? | Total time (ns) | Total moves
- **Line 3:** Has player 2 won? | Total time (ns) | Total moves

Argument	Usage
"agent=name;cmd" *"a=name;cmd"*	Specifies one agent with the given name that runs by cmd. The name can be anything; in the tournament, it will be the unique ID for your group. If two agent strings are specified, their names must be different. e.g. *"a=DefaultAgent;python3 agents/DefaultAgents/NaiveAgent.py"*
-verbose *-v*	Prints the game progress in real time
-print_protocol *-p*	Prints all messages received from both agents and all messages sent to Red. Messages sent to Blue are not sent because they are identical (exception being *START*).
board_size=n *b=n*	Creates a custom board of size *nxn*. *n=11* by default.
-log *-l*	Records all moves and start/end statistics to a CSV file under *Hex/logs/*. See 3.2.
-switch *-s*	Switches the order of agents. Use with a single agent to make it Blue and the default agent Red.
-java *-j*	Uses the Java default agent instead of the Python default agent.
-double *-d*	If exactly one agent is specified, it will be instantiated twice and it will play both sides.

More examples below. Run through these to understand the arguments and game progress.

Command	Result
python Hex.py	Runs a normal game between two reference agents. This will only print the results.
python Hex.py "a=PNA;python agents\DefaultAgents\NaiveAgent.py" "a=JNA;java -classpath agents\DefaultAgents NaiveAgent" -v	Runs a normal game between two specified agents. Red will be PNA, the python reference agent, and Blue will be JNA, the java reference agent. The progress of the game will be printed to the screen in real time, in a human-readable format.

python Hex.py "a=good_agent;python agents\Group888\BestAgent.py " b=2 -p	Runs a game with board size 2x2 between the specified good_agent as Red and the Python reference agent as Blue. The protocol exchange will be printed in real time, i.e. all messages sent and received by the engine.
python Hex.py "a=888;python agents\Group888\BestAgent.py " "a=BadAgent;python agents\DefaultAgents\IllegalMessageAgent.py" -s -v -l	Runs a normal game between the specified "888" agent as Blue and BadAgent as Red (switched due to -s). BadAgent will send an illegal message. -v will print the progress of the game in real time, including the reception of an illegal message, and -l will log the wrong move and you will be able to see exactly what illegal message was sent in the created log.

3.2 Logs

Logs are saved in CSV format in the directory Hex/logs/. They can be interpreted as follows:

- **Line 1:** Date and time
- **Line 2:** Board size
- **Line 3:** Header of log table
- **Log table:**
  - **Column No:** Number of current move
  - **Column Player:** Which agent made the move
  - **Columns X,Y:** Coordinates of the move
    - -1,-1 is a swap
    - -1,s is an illegal move that sent the message s. Beware of OS-dependent line endings!
  - **Column Time:** time taken for the move (in ns).
- **Final rows:** 4 rows that detail the end results and have No=0
  - **Row 1:** Game end row – Winner | "End" | Cause of end | Whether a swap took place
  - **Row 2:** Total stats – "Total" | Number of moves | Total ns elapsed | Mean ns per move
  - **Rows 3 & 4:** Stats per colour – Colour | Number of moves | Total ns | Mean ns

3.3 Submitting to Blackboard

Your submission should be an archive *GroupXXX.tar*, where *XXX* is your group's unique ID. This archive will contain a directory *GroupXXX*, in which you will include your agent's files **and a text file *cmd.txt*** (see next section). Your agent can be written in any language, provided that no additional libraries should be installed.

For example, if your group is *Group024* and your agent is a python script called *Agent24.py*, then:

- Your agent should run correctly with the command *python3 agents/Group024/Agent24.py*
- This command should be the contents of the file *agents/Group024/cmd.txt*
- The engine should be able to match your agent against the default agent using the command *python3 Hex.py "a=Group024;python3 agents/Group024/Agent24.py"*
- You can archive your agent using the command *tar -cvf Group024.tar Group024.* This command works on Linux, MacOS, as well as on Windows.

An example directory *Hex/agents/Group888* has been provided to you as a guideline. Make sure that your archive extracts into a directory with structure similar to *Group888* when using the command *tar xzvf GroupXXX.tar*

3.4 cmd.txt

Your agent will be run from within a superdirectory *agents/*. You should plan that your files will be extracted under *./agents/GroupXXX/*, as is the case with the example directory *Group888.* Your file *cmd.txt* will be used to automatically extract your agent's run command. Therefore, you should keep in mind that:

1. Your agent's run string must account for the directory structure. e.g. if you would normally use the command *python Hex.py "a=my_agent;python MyAgent.py"*, then your cmd.txt should rather be *python agents/GroupXXX/MyAgent.py*
2. Your *cmd.txt* file cannot contain any other text. Otherwise, the command will fail and your agent will lose its games in the tournament.

3.5 How do I connect to the engine?

The engine uses a TCP socket to communicate with the agents. Host is *localhost/127.0.0.1* and port is *1234*. The engine starts the agents using their commands in the shell at the start of the game, then it waits for them to connect – so connecting is the first thing that your agent should do.

3.6 Technical considerations for your agent

- The engine runs using *Python 3.7* or later.
- Make sure that your agent works on the Virtual Machine and on the given test machine (aigamestest). You may need to be connected to the University VPN and to kilburn.cs.man.ac.uk to access the test machine.
- Communication between agent and engine is exclusively uppercase and no whitespace is permitted.
- Do not use *stdout* or *stderr* for your bot's communication with itself; use pipes or sockets instead. During the tournament, all output to these channels will be restricted and your agent will not work properly if you use them.
- The uni machine has no X-server, so you should avoid GUI.
- Your socket is only read during your turn. If you send a message out of order, that will be read and interpreted as your next move.

3.7 For a better understanding of the game

The source code for the engine is also provided in case you will use Python for your bot. It may be useful to read *Hex/src/Protocol.py* to understand how to connect to the socket. Some further debugging tips:

- If your agent appears to send illegal messages, run it against the reference agent (only specify one agent), enable protocol printing (*-p*), and enable logs (*-l*). You will see the message that your agent has sent through both channels (though logs may be easier to follow; the illegal message will be the last logged move). e.g. *python Hex.py "a=Group12;java -classpath agents/Group12 Agent.jar" -p -l*
- To easily test your agent as Blue against the reference agent, use *-s*.
- To easily test your agent against itself, use *-d*.
- You can change the maximum time allotted per player by editing Hex/Game.py line 20. Input the time in nanoseconds. *10**9ns* or *1s* is recommended when testing basic functionality.

Below is detailed a sequence of events in an example 2x2 game. Note that no agents are specified, so the Python default agent is used on both sides.

```
>>> python Hex.py -p b=2
```

```
NOTICE: Fewer than two agents specified. Missing players will be replaced with the default agent.
Connected DefaultAgent1 at ('127.0.0.1', 64636) // once the first agent has connected to the socket
Connected DefaultAgent2 at ('127.0.0.1', 64637) // once the second agent has connected to the socket
Sent START;2;R // the first agent is notified that it is Red, the second that it is Blue
Received 0,1 from DefaultAgent1 in ~0.0s. // Red sends its first move. For time < 10^-4s, the clock may not be accurate.
Sent CHANGE;0,1;0R,00;B // notified both agents of the new state
Received SWAP from DefaultAgent2 in ~0.0s. // Blue decides to swap sides
Sent CHANGE;SWAP;0R,00;B // notified both agents of the new state
Received 1,1 from DefaultAgent1 in ~0.0s. // Blue (now the first agent) makes its move
Sent CHANGE;1,1;0R,0B;R // notified both agents of the new state
Received 0,0 from DefaultAgent2 in ~0.0s. // Red (the second agent) now moves again
Sent CHANGE;0,0;RR,0B;B // notified both agents of the new state
Received 1,0 from DefaultAgent1 in ~0.0s. // Blue makes its winning move
Sent CHANGE;1,0;RR,BB;END // notified both agents of the new state; the last move was a winning move
Sent END;B // notified both agents that the game is over
Closed DefaultAgent2 at ('127.0.0.1', 64637) // closed Red socket
Closed DefaultAgent1 at ('127.0.0.1', 64636) // closed Blue socket
Win // this line is sent to stderr. It mentions whether the game has ended normally, in a timeout, or with an illegal move.
True 992700 3 // stderr; whether the first player has won | time (ns) | number of moves
False 993800 2 // stderr; whether the second player has won | time (ns) | number of moves
```

3.7.1 Default agents

A number of default agents are provided to you. You can use these to more easily understand the game progress, and as blueprints for your agent.

DefaultAgent.py: A Python implementation of the default agent. This agent works as follows:

1. Connect to the TCP socket. Go to state 2.
2. Wait for the START message.
    - **2.1.** If you are assigned Red, go to state 3.
    - **2.2.** Otherwise, go to state 4.
3. Make a move.
    - **3.1.** If current turn is 2, make a SWAP with a 50% chance.
    - **3.2.** Otherwise, choose a random move from the available pool.
    - **3.3.** Go to state 4.
4. Wait for a message.
    - **4.1.** If it is END or its last argument is END, go to state 5.
    - **4.2.** Otherwise, if it is a SWAP, change colours.
    - **4.3.** If the last argument matches your colour, go to state 3.
    - **4.4.** Otherwise, go to state 4.
5. Close the connection.

DefaultAgent.java: A Java implementation of the default agent. This agent works as follows:

```
1.  Connect to the TCP socket. Go to 2.
2.  Read a message.
    2.1.  If it is START, assume your colour.
          2.1.1. If you are Red, go to 3.
          2.1.2. Otherwise, go to 2.
    2.2.  If it is CHANGE
          2.2.1. If its last argument is END, go to 4.
          2.2.2. If it is SWAP, change colours.
          2.2.3. If the last argument matches your colour, go to 3.
          2.2.4. Otherwise, go to 2.
    2.3.  Otherwise, go to 4.
3.  Make a move.
    3.1.  If current turn is 2, make a swap with a 50% chance.
    3.2.  Otherwise, choose a random move from the available pool.
    3.3.  Go to 2.
4.  Close the connection.
```