COMP30024 Artificial Intelligence
Christopher Leckie and Sarah M. Erfani

Project Part B: Interface Notes

Updated By: Lida Rashidi
Last updated: 12th April 2016

Referee test framework

In order to test your game against an opponent, we have provided a basic testing framework, which will be available as a set of Java source files in `/home/subjects/comp30024/partB/Referee-1-0` on the MSE Student Unix servers (`dimefox.eng.unimelb.edu.au` or `nutmeg.eng.unimelb.edu.au`). This testing framework provides a "referee", which can be compiled with two player implementations to construct an executable game. The referee calls each player using a standard interface (Player.java). We also provide a small number of support classes that are required by the referee.

The referee initialises the game, calls each player to make a move, reports each move to the opponent, and reports when one of the players detects a winning situation or an illegal move.

In order to use the provided interface you need to add a package to your Java application named `aiproj.hexifence` and copy the interface files into this package. After this step you need to import the contents of this package into your player class using following statement:

```
import aiproj.hexifence.*
```

Player interface

You will need to build a class that implements the Player interface, which will be used by the referee to interact with your player agent. The following line shows that the class `Myplayer` implements the interfaces `Player` and `Piece` that have been provided as part of the referee framework.

```
public class Myplayer implements Player, Piece {
}
```

Note: You will need to give a unique class name to your class that implements the `Player` interface (don't use the name Myplayer!). We request that you select one of the usernames of your group members as the name of your class that implements the `Player` interface, e.g., Chris would use:

```
public class Caleckie implements Player, Piece {
}
```

The `Player` interface includes 5 methods that you need to implement:

> `public int init(int n, int p);`
> This method is called by the referee to initialise the player. The input parameters are as follows: `n` specifies the board dimension, and `p` specifies the piece that the player will use (according to the `Piece` interface format) as assigned to your class by the referee. Your implementation of this function should return a negative value if it does not initialise

successfully.

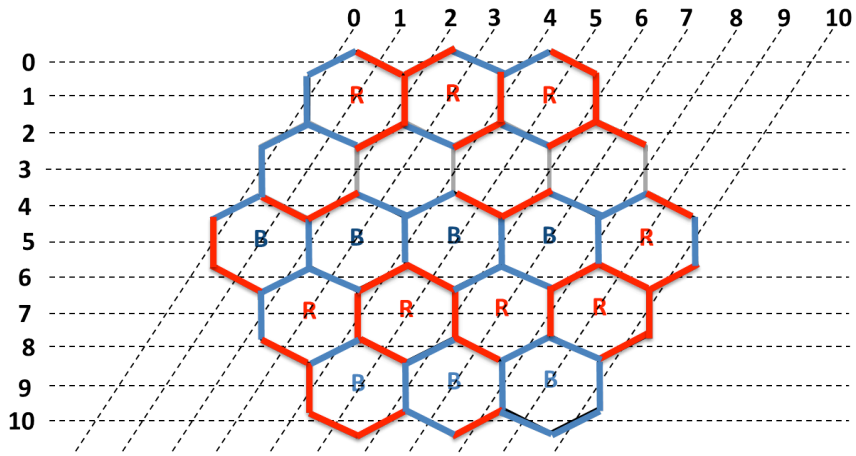
`public Move makeMove();`
This method is called by the referee to request a move by your player. Based on the current board configuration, your player should select its next move and return it as an object of the `aiproj.hexifence.Move` class. Note that each player needs to maintain its own internal state representation of the current board configuration.

`public int opponentMove(Move m);`
This method is called by the referee to inform your player about the opponent's most recent move, so that you can maintain your board configuration. The input parameter is an object from the class `aiproj.hexifence.Move`, which includes the information about the last move made by the opponent. Based on your board configuration, if your player thinks this move is illegal you need to return -1 otherwise this function should return 0 if no new cell has been captured or 1 if a new cell (or two) has been captured by the opponent.

This move is illegal: Return -1
This move is legal and no new cell has been captured by the opponent: Return 0
This move is legal and a new cell (or two) has been captured by the opponent: Return 1

`public int getWinner();`
This method should check the board configuration for a possible winner and return the winner as an integer value according to the `Piece` interface (`-1=INVALID, 0=EMPTY, 1=BLUE, 2=RED, 3=DEAD`). Note that `EMPTY` corresponds to a non-terminal state of the game, `DEAD` corresponds to a draw, and `INVALID` corresponds to the case that the game has ended because the opponent move was illegal (e.g., placing a piece on an already occupied or captured cell).

`public void printBoard(PrintStream output);`
This method is called by the referee to ask your player to print its board configuration to a `PrintStream` object `output`. Note that this output will be different from the input format specified in Project Part A, i.e., we would suggest using B, R, or + corresponding to a Blue edge in play, a Red edge in play, or an empty edge, respectively. The notations b, r, and - correspond to a captured Blue cell, a captured Red cell and a non-existing edge respectively. Please note that we place the captured cells on "non-existing" edges in the printout, for instance the first "r" in the second row is sitting in the position of a non-existing edge.



```
B  R  R  B  B  R  -  -  -  -  -
B  r  R  r  R  r  R  -  -  -  -
B  B  R  B  R  B  R  R  -  -  -
B  -  +  -  +  -  +  -  +  -  -
B  R  R  B  B  R  R  B  B  R  -
R  b  B  b  B  B  B  B  B  r  B
-  R  B  B  R  R  B  B  R  R  R
-  -  B  r  R  r  R  r  R  r  R
-  -  -  R  B  R  B  R  B  B  R
-  -  -  -  R  b  B  b  B  b  B
-  -  -  -  -  R  R  B  R  B  B
```

Piece interface

This is a final interface that defines types of pieces that can appear on a Hexifence board. It does not include any methods.

Move class

This class encapsulates a possible Hexifence move, and includes the following public fields.

```
public int P;
public int Row;
public int Col;
```

`P` shows the player who has made this move at the Row and Col edge given, using an integer value in the format of the `Piece` interface (note that only `BLUE` and `RED` are valid values).

Referee class

The `Referee` class is the top-level class of the game-playing program. The `main` method of the `Referee` accepts 3 input command line arguments in the following order:

1. The board dimension.
2. The full class name of the implementation of the `Player` class of the first player, i.e., including package name. This instance would play as BLUE.
3. The full class name of the implementation of the `Player` class of the second player, i.e., including package name. This instance would play as RED.

You can import the `Referee` source files in your Java editor and run it, or you can use the command prompt to run the `Referee`.

The following describes a scenario to use the command line to run the `Referee`.

Assume that (1) you are in a Windows or Unix command prompt where the current directory is the root of the `Referee-1-0/bin` directory; (2) binaries of an implementation of a player class called `SimplePlayer` is in "`../../SimpleDirectory/bin`"; (3) the full class name of `SimplePlayer` class is "`aiproj.hexifence.SimplePlayer`"; (4) this implementation uses the aima-core library which is in the directory "`../../aima-core.jar`". The following

command can be used to run the referee:

```
> java -cp ../../aima-core.jar;../../SimplePlayer/bin;.
aiproj.hexifence.Referee 6 aiproj.hexifence.SimplePlayer
aiproj.hexifence.SimplePlayer
```

Any questions about the referee and its interface should be directed to **Lida Rashidi** (lrashidi@student.unimelb.edu.au) via email (use the subject header `AI-projB'). A separate document will describe the submission instructions and expectations for Part B.