

## Vitro Project : Reversi Model AI Assignment

The board game Reversi (also called Othello™) is a classic two-player game<sup>1</sup>. This game will be the domain for learning about several topics from adversarial search. For this project, you will implement classic adversarial search algorithms (minimax, alpha-beta pruning) and an evaluation function in order to create agents for playing Reversi.

### Code Base

The following files are provided for this project. The main file `ReversiAssignment.java` is used to setup the game and assign agents to play the game. This is the only file you will need to modify as you complete each part of this project.

<code>ReversiAssignment.java</code>	The <b>main file</b> used to setup the domain, assign players to an agent implementation.
<code>Reversi.java</code>	The source file defining the game rules.
<code>ReversiView.java</code>	The source file defining the view of the game.
<code>ReversiGreedyBrain.java</code>	Code to implement a default player; follows a greedy policy of selecting next moves.
<code>vitro.jar</code>	Base Vitro project jar file.

In addition to files above, you will be creating additional “ReversiBrains” implementing minimax and alpha-beta pruning algorithms.

### Project Setup

For the assignment, un-zip the file (reversi.zip). Setup instructions will involve using the Eclipse IDE, but other IDEs may be used.

- Create a new Java project. On the second page of setup options, use the “Libraries” tab -> “Add External Jars” to import the Vitro codebase (vitro.jar)
- Link (or copy) the source files in the 'src/' folder within the Eclipse project workspace.
- Set `ReversiAssignment.java` as the main file.

Test your project; run the assignment using the default greedy agent as both players. The simulation using the default agents and assignment code should look like Figure 2.

### Othello Game

The rules for Reversi are summarized here. Full rules and suggested strategies are described in the links at the end of this document.

Reversi is typically played on a game board with 8x8 (64 squares). Opposing pieces (disks) are alternatively placed on the board until there are no spaces remaining or no legal moves left. The player with the most pieces on the board at the end, wins. A demo of this domain can be run:

```
java -classpath lib/vitro.jar demos.reversi.ReversiEye
```

The opening position starts with two pieces for each player on the board as shown in Figure 1(a). The two players will be called Player1 and Player2, green and grey pieces respectively in Fig. 1; the players may also be referred to as “dark” vs. “light” or “black” vs. “white”.

---

<sup>1</sup> [http://en.wikipedia.org/wiki/Othello\\_%28game%29](http://en.wikipedia.org/wiki/Othello_%28game%29)

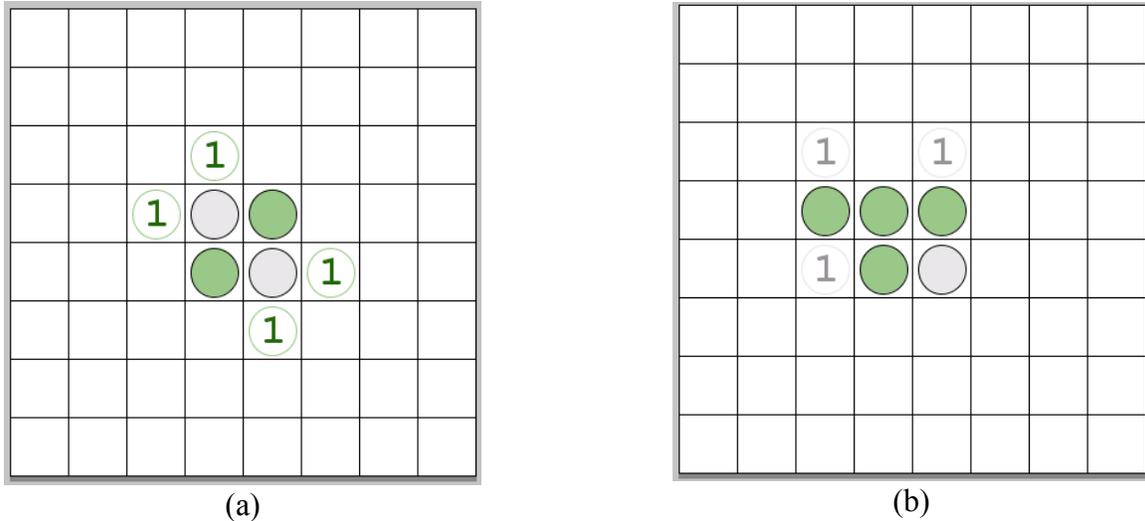


Figure 1. (a) Opening position for Reversi (Player1 - green, Player2 - grey), Player1 moves first.  
 (b) One of the possible first moves by Player1.

Player1 initiates the game with the first move; placing a new piece in one of the four positions shown in Figure 1 indicated by the circle outline. The number for each of the possible positions states the number of opposing pieces captured by taking the given move.

For each players turn, the possible legal positions to place a disk are such that at least one opponents pieces are “trapped”. Consider placing a dark piece, it may be set in any position such that there is one straight line (vertical, horizontal, and 45° diagonals) with one or more consecutive light pieces in between the two dark end points of the line. When the new piece is placed, the trapped pieces are switched to the opposite color. Figure 1(b) shows the results of one of the possible first moves of Player1.

### Project Interface

The default Reversi assignment will be initialized as a 4x4 board (note, the standard Reversi board involves an 8x8 grid). Note, concurrent runs of the simulation may have different colors for the two players. The starting position remains centered in the grid with Player1 in grey and Player2 in peach in Fig. 2. The top of the simulation displays the number of pieces each player has on the board. The interface buttons from left to right have the following functionality:

-  Reset the game to the starting position.
-  Step backwards; unrolling actions in their reverse order of play.
-  Start the simulation. Each players actions will be taken in turn.
-  Step forward; apply a single action at a time.
-  Enable/disable legend information.

Become familiar with the Reversi interface, to observe agent behavior. Finally, begin to familiarize yourself with the agent brains (method to select a player's next action) and Reversi domain (how board information is stored). Additionally, look through the appropriate Java documentation for the project.

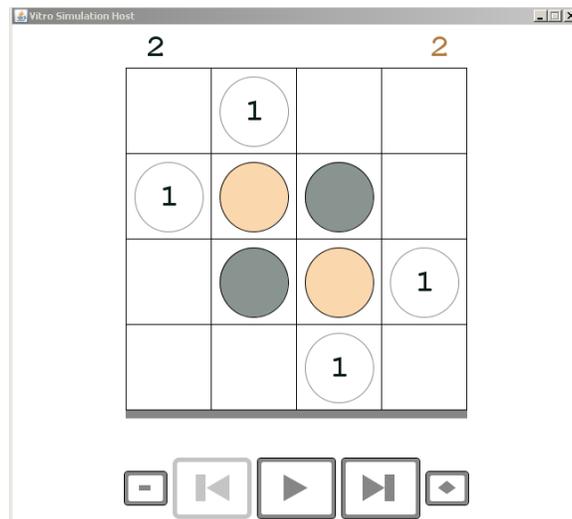


Figure 2. Opening display for Reversi Assignment.

*Part I – Minimax (5 points)*

For this part of the project, you will implement the minimax algorithm for use as an improved reversi player. For a board of this size, the complete game tree can be explored. Create a new Java source file `ReversiMinimaxBrain[LastName][FirstName].java` replacing `[LastName]`, `[FirstName]` with your information. Your implementation should work no matter if it is used for Player1 or Player2, the player under consideration is determined in the variable 'actor'. Note, it will be necessary to create an evaluation function; only apply this function to terminal states.

*Question I (2 points)*

Which player wins with Player1 using minimax and Player2 using the greedy method? What margin of victory occurs (i.e., number of winner's pieces – loser's pieces)? Report the average number of states explored in the minimax search space (this should be the cumulative sum throughout a single game). Which player wins with Player1 using a greedy and Player2 using a minimax method? What margin of victory occurs? Report the average number of states explored in the minimax search space (this should be the cumulative sum throughout a single game). Which player wins with both players using a minimax method? What margin of victory occurs? Report the average number of states explored in each player's minimax search space (this should be the cumulative sum throughout a single game).

*Part II – Alpha-beta pruning (5 points)*

Now you will create a new version of the Reversi player that uses alpha-beta pruning to select the best action. Create a new Java source file `ReversiAlphaBetaBrain[LastName][FirstName].java` replacing `[LastName]`, `[FirstName]` with your information. The alpha-beta pruning player should have the same behavior as a minimax player (barring tie-breakers).

*Question II (1 point)*

Determine the average number of states explored by each player when using alpha-beta pruning? Compare this number to the previous question.

*Part III – Evaluation Functions (2 points)*

Restrict the alpha-beta pruning search to a limited depth; create a new Java source file `ReversiHeuristicBrain[LastName][FirstName].java`. In addition, you will develop a heuristic evaluation function that evaluates the game at any point in the search state. Please note that while the number of

pieces a player has at the end of the game determines who wins; the number of pieces may fluctuate up and down throughout game plays as pieces are trapped and exchanged. Look to the Reversi resources at the end of this project description for information on strategies; these may help in designing the heuristic function (patterns, mobility, corners, and parity are some concepts to consider).

Run your new agent on a larger game board. Modify the ReversiAssignment.java line:

```
Reversi model = new Reversi(4, 4);  
to use a 6x6 or 8x8 grid:  
Reversi model = new Reversi(6, 6);  
or  
Reversi model = new Reversi(8, 8);
```

### *Question III – (2 points)*

Describe how your heuristic evaluation function works? What features or pieces of information are included in its calculation.

Which player wins with Player1 allowed to search to depth 2 and Player2 to depth 3? and vice-versa?

### *Extra Credit (1 point)*

Your heuristic search agent will be run against your fellow students and instructors. The top 10% of the class will receive an extra credit point; those in the top 25% will receive a half extra credit point.

### *Extra Credit (1 point)*

Allow the evaluation functions to be tied to a given domain, but separate the search solutions to be used in multiple domains. Text the minimax method on the Tic Tac Toe domain of the Vitro project.

### **Deliverables:**

Submit the all your code via the normal class submit procedures. In addition, include a text file, Questions[LastName][FirstName].txt, with your answers to the questions I-III.

### *Project Resources*

- Eclipse IDE, <http://www.eclipse.org/>

### *Reversi / Othello Resources*

#### Rules

- Othello rules, Pressman Toy, [http://www.pressmantoy.com/instructions/instruct\\_othello.html](http://www.pressmantoy.com/instructions/instruct_othello.html)
- Othello rules, British Othello Federation (BOF), <http://www.britishothello.org.uk/rules.html>

#### General Information

- Reversi, Wikipedia, <http://en.wikipedia.org/wiki/Reversi>
- Othello, Wikipedia, [http://en.wikipedia.org/wiki/Computer\\_Othello](http://en.wikipedia.org/wiki/Computer_Othello)
- There are several books written on Othello strategy

### *Reversi / Othello Programs*

- Cassio, Stephane Nicolet, <http://cassio.free.fr/>
- Edax, Richard Delorme, <http://abulmo.perso.neuf.fr/edax/4.2/index.htm>
- Herakles, Konstantinos Tournavitis, <http://www.herakles.tournavitis.de/>
- Logistello, Michan Buro, <http://skatgame.net/mburo/log.html>
- Zebra, Gunnar Andersson, <http://www.radagast.se/othello/>