Applied Artificial Intelligence
Assignment 1: The Reversi Game

Philipp August and Franz Pucher

February 2016

Contents

0.1 Report

This report describes the first assignment: Reversi/Othello in AI which was written in C++ using Qt Creator. Section 0.2 gives a brief overview of the class interaction. A detailed description can be found in the automatically generated doxygen [1] documentation. Section 0.3 describes how we represented the board followed by the possible settings and the game flow in section 0.4. Finally the used alpha-beta algorithm and the used heuristics are discussed in section 0.5.

0.2 Class design

The game consists of the following classes:

- Board The Board class holds the board matrix (see also section 0.3 and stores every board on a stack with the current and opponent player. It is used to find all allowed moves for the current player and actually execute them.
- GameEngine The GameEngine class controls the course of the game by processing user input, executing moves from humans and the computer and informing the user about events using lists from the GUI. Furthermore it has an interface to the ui class UIGameScene to evoke redraws of the board.
- MainWindow constrains the window elements by fixing their size. Improves rendering of the board UIGameScene class inside the GraphicsView. Sets a validator to the user input for the search depth to allow only integer numbers (0 to 10). Sets the connects for the signal and slot mechanism for the newGame push button.
- Player The Player class defines the current and the opponent player. Each player has its own color, BLACK or WHITE, and is of type HUMAN or COMPUTER.
- UIGameScene The UIGameScene class stores the graphical representation of the board using UISquare class. The GameEngine uses this class to redraw the board matrix from the Board class. This class also forwards mouse release events to the GameEngine made by the user.
- UISquare The UISquare class represents a single square inside the QGraphicsScene (UIGameScene). It is used to show the different states of a square (BOARD, BLACK, WHITE, ALLOWED) to the user. The board matrix 2x2 QVector of the UIGameScene class consists of these UISquares.
- AI The AI class implements the alpha-beta pruning algorithm to find the next move for the computer player. Thereby three different heuristics (disc count, legal moves count, square rating), implemented in the evaluation funciton, are used.

0.3 Board representation

The board is represented by a 2x2 QVector of type enum State NONE, BOARD, BLACK, WHITE, ALLOWED, SUGGESTED inside the Board class 0.2. To display the board matrix to the user, the UIGameScene class which inherits from QGraphicsScene is used. This QGraphicsScene is embedded inside the QGraphicsView which is one of the main elements of the main window UI. The main window is created using Qt's GUI-Designer.

A square usally takes on the states: BOARD, BLACK, WHITE, ALLOWED. Where ALLOWED suggests the currently legal moves for the current player. All square states are stored in the compact 2x2 QVector of the Board class, which is used by the GameEngine to control the game flow. The whole board is displayed to the user by utilizing UIGameScene and UISquare.

0.4 Game settings and operation

The game is executed with the provided othello mac binary. In figure 0.1 the user is able to choose the number of human players where two is the maximum. If one is chosen, the computer is set as the black player who begins the game always.

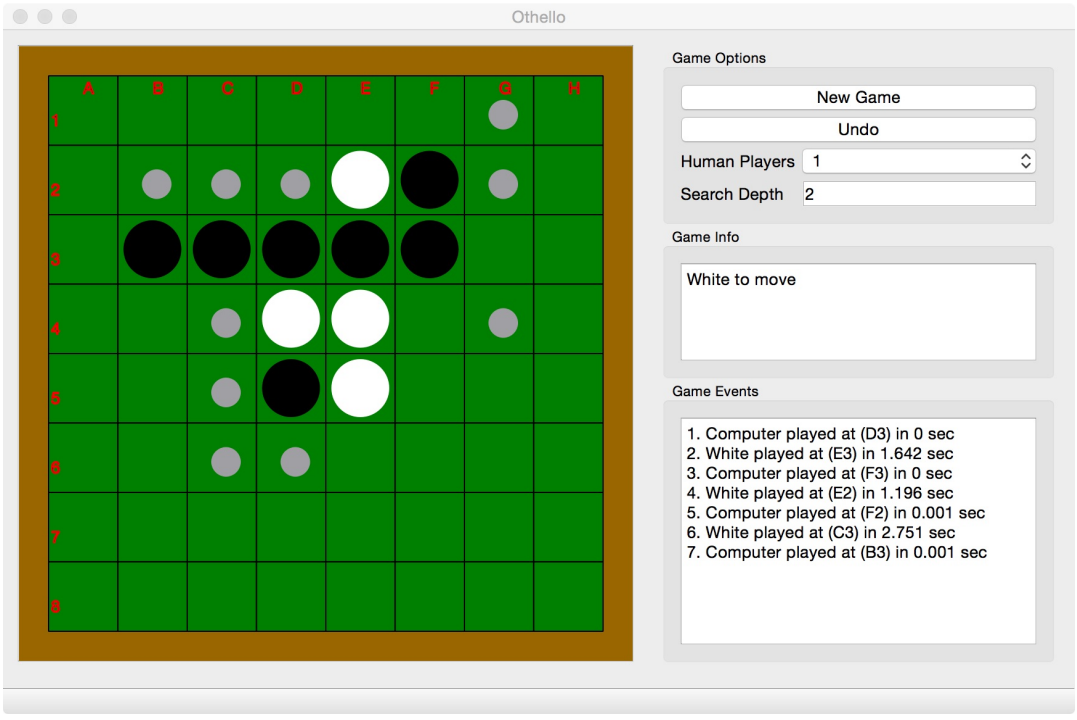

Figure 0.1: Main window of the othello game.

The alpha-beta pruning algorithm which is describedin section 0.5 can be modified by setting a integer search depth of between 0 and 10. However, values above 6 should be

avoided, due to long computing time.

After these settings a new game can be started by pressing the New Game button. The board is displayed in the graphicsView and the user is informed about move events, whos turn it is and the game results. These informations are provided by two textEdits. Furthermore the move-event-textEdit shows the time used for reasoning.

The user is able to press only the allowed squares indicated by the gray circles. Undoing a move is mainly useful for two human players. It should be used twice if a human plays against a computer.

When a game ends, the results are displayed inside the info-textEdit.

0.5 Algorithm design

Our othello game uses the alpha-beta pruning algorithm, which is implemented inside the AI class.

The *evaluateBoard*() function inside the AI class is used to evaluate the current board using three different heuristics. Those are disc count, number of legal moves, and rating of board position by using the m_heuristic matrix from the class.

The disc count is weighted with 1/100, legal moves are weighted with 1 and board positions according to the m_heuristic matrix. Here corners are rated very high, followed by edge corners. The computer should avoid inner squares. Those values were partly taken from [2].

Bibliography

[1] Doxygen August Philipp and Pucher Franz. Doxygen othello documentation, February 2016.

[2] Heuristics. http://www.cs.cornell.edu/~yuli/othello/othello.html. Accessed: 2016-02-15.

[3] Alpha-beta pruning algorithm. https://de.wikipedia.org/wiki/Alpha-Beta-Suche. Accessed: 2016-02-15.