Fabian Hough

Fh914

CS-GY 6613 – Artificial Intelligence

Project: Mini-Checkers Game

**Program Structure and Compilation**

The necessary files to compile the program are included in the “Mini-Checkers” folder, and the “SDL2” folder. Any Visual Studio files can theoretically be ignored, however it is preferred that other users compile this program in Visual Studio Community 2017.

Users of VS Community 2017 can generally just copy the entire project folder, and run it rom the solution. They will need to take a number of steps in order to ensure compilation success:

1. Retarget the solution
2. Open properties of the solution
   1. Ensure that the Active platform is Win32. This can be changed in the configuration manager located right next the “Platform” area in the properties page
   2. In the C/C++ properties, the “Additional Include Directories” need to retarget the current target to the SDL2/include folder.
   3. In the Linker properties, the “Additional Library Directories” need to retarget the current target to the SDL2/lib folder.
3. Build, and Run

For general users, it should be sufficient to include the primary “Mini-Checkers” folder and the “SDL2” folder. In compilation, it must be ensured that the program links to the additional “include” and “lib” folders in the “SDL2” folder.

The files that have been purely designed by myself include:

*main.cpp*

*cBoard.cpp*

*cBoard.h*

*Game.cpp*

*Game.h*

*TextureManager.cpp*

*TextureManager.h*

These can all be found within the “Mini-Checkers” folder.

**High-Level Description**

This Mini-Checkers program utilizes the standard Alpha-Beta Pruning algorithm. It starts off by asking users what difficulty level they would like to use (1,2,3), and what order they would like to take turns. A GUI representing the board is then shown, and users can interact directly with the board. The GUI is built using the SDL2 library, and constantly updates the game window.

Whenever it is the AI’s turn, it calls upon the Alpha-Beta algorithm function. This function, in turns, recursively calls the Max and Min functions up until a specific cutoff-level. This cutoff-level varies according to the difficulty level, however the maximum cutoff-level acceptable under 15 seconds is 16. (For higher-end processors, such as the i5-6600k used in testing, a cutoff-level of 17 is acceptable)

If the cutoff level does not result in a terminal state, then an evaluation function is called in order to determine an approximate value. This eval function calls on a modified Max-Min heuristic.

The “next best move” state is passed up to the current board state, and replaces it as the current state. This counts as the AI’s move. All AI statistics (max depth of tree, total nodes generated, and pruning in both Max and Min functions) are then outputted to the console, and the human player can make their move. This process repeats itself, until a terminal state is reached.

**States**

The terminal states are as follows:

* AI (Max) Win: 12
* Human (Min) Win: -12
* Draw: 0

**Eval Function and Heuristics**

The eval function is called whenever a utility value must be calculated for a non-terminal state; this is caused by reaching the set cutoff-level. Within the eval function, the heuristic used is a modified Max-Min heuristic. What this means is that instead of just doing MaxPieces – MinPieces in order to determine a utility value, this heuristic puts weights on certain pieces.

The utility values, thus, range from -12 to 12, similar to the range of the terminal states.

The heuristic operates as follows:

* Pieces past the threshold of the ‘enemies’ pieces are considered ‘safe’, and are given a value of 2
* Pieces still behind the threshold of the ‘enemies’ pieces are considered ‘unsafe’, and are given a value of 1
* The utility value is calculated by adding up all of the piece values for each player, and executing Max – Min.

To clarify, the threshold of the ‘enemies’ pieces refers to the area in between the last enemy piece, and the goal end of the board for the player.

The motivation for using this heuristic is that it simultaneously gives the AI motivation to reach the other side of the board as quickly as possible, and to still eliminate as many human player pieces as possible. This is in contrast to the normal Max – Min heuristic, which does not ‘reward’ the AI for reaching the goal end of the board.

**Difficulty Levels**

There are 3 difficulty levels for this game, which are dependent on predetermined cutoff levels. In essence, the thought behind the implementation of this difficulty level system is to emulate players who think a specific number of moves ahead.

For the easy difficulty, the AI player only ‘looks’ 1 move ahead; it sees the consequences and value of its immediate actions.

For the medium difficulty, the AI player looks 5 moves ahead. While still difficult, it is at least manageable to get a ‘draw’ condition.

For the hard difficulty, the AI player looks the full 16 moves ahead. As of yet, it is impossible to defeat.

When the human player chooses the difficulty level, the current cutoff-level is set to the corresponding predetermined cutoff-level.