California State Polytechnic University, Pomona

Solving m,n,k-game Using Alpha-Beta Pruning and Maximin Algorithm

Project 3

Joshua Gibbs
CS 420 - Artificial Intelligence
Dominick Atanasio
6 March 2018

Approach

The project contains eight class files, **`Main`**, **`BoardTile`**, **`BoardState,`** **`AbstractTileState, TileState, Pattern, ANSIColors`** and **`AI`**. The program entry is at **`Main`**, where we ask the user for the allotted time for each turn and which player is going first. Inside the **`Main.menu`** method, the board and list of moves are printed. We also initialize the game in this class by first creating an empty board via **`new BoardState()`**. After that if we have the first move, we will always move at **`E5`**. After this, we print the board and ask the opponent to enter their move. This loop continues until a terminal state is reached.

The previously mentioned BoardState class contains a 2D array of TileStates, a heuristic value, a record of the last move and its player, the current depth of the board, a boolean terminal state, and a PriorityQueue of children. A new board can be created by calling board_state_instance.copyBoardWithMove(), which accepts coordinates for the new tile to be laid down. After this board is generated, as with any BoardState, it's heuristic value and terminal state determination are calculated. We will discuss the heuristic function in this class in a separate section.

The TileState class is simply an enum of empty, X, and O states. This is very similar to the AbstractTileState class, which is also an enum of states. The ANSIColors class is a list of ANSI colors and font faces to be used in a unix terminal for colored output.

Heuristic

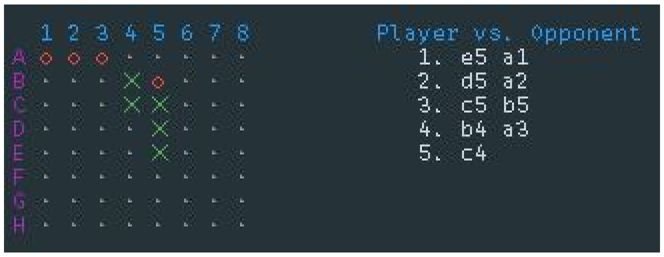
The heuristic used in the BoardState class works by identifying several pre-defined patterns, and gives them each a point value. For example, the pattern [ ][x][x][ ] is very valuable in this game. When a pattern is matched, that pattern's score is added to the heuristic value for that board.

Alpha-Beta Pruning

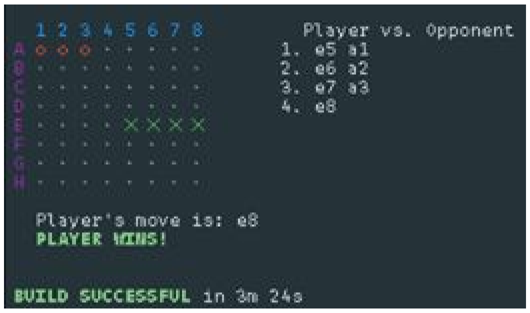
The alpha-beta pruning algorithm in the AI class was originally based off of the pseudo code provided in class. One of the unique parts of this implementation is that the only successors generated are ones that are touching another non-empty tile. From those successors, only the top 25% (by heuristic value) will be evaluated by maxValue and minValue. In addition to checking for a terminal state, maxValue and minValue use a cutoff method to determine if too much time has passed, and if we have gone deep enough. For my algorithm, I set the max depth to 10 layers. In about 50% of the cases, we will finish our algorithm just before time is up.

Analysis

While alpha beta pruning and my heuristic function do a decent job at picking a good move, there are many instances where a human would easily be able to defeat the program. With enough time, or a more efficient algorithm the AI could become much better, and given an infinite amount of time to run, be perfect.



```
  1 2 3 4 5 6 7 8          Player vs. Opponent
A o o o . . . . .              1. e5 a1
B . . . X o . . .              2. d5 a2
C . . . X X . . .              3. c5 b5
D . . . . X . . .              4. b4 a3
E . . . . X . . .              5. c4
F . . . . . . . .
G . . . . . . . .
H . . . . . . . .
```



```
  1 2 3 4 5 6 7 8          Player vs. Opponent
A o o o . . . . .        1. e5 a1
B . . . . . . . .        2. e6 a2
C . . . . . . . .        3. e7 a3
D . . . . . . . .        4. e8
E . . . . X X X X
F . . . . . . . .
G . . . . . . . .
H . . . . . . . .

  Player's move is: e8
  PLAYER WINS!


BUILD SUCCESSFUL in 3m 24s
```