The University of Melbourne
COMP300024 Artificial Intelligence

Project Part B

Playing the Game

James Taranto: 640092
Flying Solo

May 21, 2019

1 Program Structure

The program is divided into three major components;

Board Defines internal game representation and methods to manipulate and evaluate game state.

Player An abstract class containing all code necessary for a player to function within specification bar move selection. Including keeping track of and updating the player's representation of the game.

Player Subclasses `(brs, greedy, random)` Specific implementations of different player types, implements abstract method `Player.action` i.e. move selection. Best reply search, naive greedy and random, respectively.

2 Approach

2.1 Search Strategy

My program uses best reply search, [1] an adaptation of negamax with alpha-beta pruning to n-players. Best reply search modifies the search tree from n-players to two, by collapsing the moves of the opposing players into a singular super-player. The search alternates between the moves for the root, maximising player and the moves for both opposing players. This results in a higher average branching factor and illegal board configurations due to effectively submitting a pass for a minimising player, but effectively increases the search depth compared to paranoid and $\max^n$ algorithms.

2.2 Evaluation Function

The evaluation function consists of four parts weighted by decreasing factors of 10;

Count The amount of pieces on and off board, controlled by the root player, subtracted by the other players combined. Motivates the player to stay alive.

Score The current score of the root player, subtracted by the combined score of the other players. Motivates the player exit.

Adjacent Bonus A bonus for each piece of the root player next to another of it's own pieces. Motivates the player to keep pieces closer together, to inhibit losing control of pieces.

Distance The sum of inverse square roots of the distance to the nearest exit for the root player, subtracted by the other players combined. The inverse is taken so that closer to the exit means a higher score, and the square root so that moving more pieces towards the exit is worth more than moving a single piece towards the exit. Motivates the player to move closer to the exit.

Note, the factor of **Score** is increased by a factor of 100, to take precedence over **Count**, when the root player controls a majority of pieces.

3 Effectiveness

Compared to $\max^n$ and paranoid, best reply search performed better in the majority of cases. Searching to a greater depth was also typically much faster. Winning performance however was mostly bound by the strength of the evaluation function, as expected. Features and feature weights were determined by hand, manipulated after observation of performance against other player modules `(random, greedy)` and other teams on the battleground.

4 Discussion

4.1 Optimisation

4.1.1 Algorithm

To reduce the number of nodes expanded, best reply search utilises alpha-beta pruning. To further reduce nodes expanded, rudimentary move ordering was used, sorting the moves to explore by their type, in the order `(EXIT, JUMP, MOVE, PASS)`, reasoning that exits prevail over jumps, jumps over moves and moves over passes.
Also included is an early exit clause on a game-winning move found, telling the program to return that move instead of searching through all possible ways to win.

4.1.2 Memory

To reduce the memory consumption of the program, the search algorithm makes and un-makes moves to the player's state representation as it recurses through the tree, resulting in the program typically using only around `1.5 MiB`.
However, this limits the ability to perform iterative deepening, due to potentially leading to invalid boards without making copies of the board representation. Although implementing iterative deepening with a time limit was deemed inappropriate, as the time limit specified is not per turn. Subsequently, memoisation using a transposition table was also deemed unsuitable.

4.1.3 Time

Making and un-making moves before and after each recursive call also served to reduce the time consumption of the program. Another consideration to reduce the time consumption of the program was the control flow and data structures used in the evaluation function. The function was designed to only loop through the board once, gathering most of the data it needs to evaluate in one loop. The pieces of interest for calculating the **Adjacent Bonus** were added to a set to reduce the time needed for lookups in the following nested loops.

References

[1] Maarten PD Schadd and Mark HM Winands. Best reply search for multiplayer games. *IEEE Transactions on Computational Intelligence and AI in Games*, 3(1):57–66, 2011.