Octav Dragoi
Winston Huang
Simon Shuham
Alan Xie

Ocaml Connect Four Solver

We have fully decided to implement a Connect Four Solver, which is integrated into a system that allows players to play each other, or for players to play computers, or for computers to play other computers. Like it is described from the previous specification, the solver itself will use minimax with alpha-beta pruning to calculate what the best move is in a given position. The heuristic functions that we use for minimax can also be tinkered with, and we might end with tweaking our heuristic function and comparing various of these functions by playing computers against one another.

We envision several different signatures and modules that would be required to support our Connect Four game. To implement the players, we would have to have a PLAYER signature, with a function "move" that returns a move (which is basically an int for the column to place a piece into) given a certain board position. We will have a "Human" module that implements PLAYER, where the "move" function takes an input from the user. There will also be a "Computer" module that implements PLAYER, where the "make_move" function assesses the position and returns what it considers to be the best move based on the minimax algorithm.

We would also need an "Initializer" module, which sets up the game, getting choices for whether each player is a human or a computer, and containing a recursive "play" function that checks for whether one of the players has won, and if it is not the case, plays the next move in the game.

Finally, we might need another module, titled "Helpers" that includes all the related helper functions to make our program work. It could provide functions like "game_over" which can check if a game is completed, or "legal_move", to check if a move made is a legal one.

Timeline:

April 20-26

- Implement PLAYER signature
- Implement Human module
- Implement Computer module
- Implement Initializer module
- Implement Helpers module
- Implement minimax algorithm

April 27-May 2

- Implement alpha-beta pruning
- Try other heuristic functions
- Try threading
- Presentation video
- Final writeup

Code:

```
module type PLAYER
sig
  val make_move : position -> move
end

module Human : PLAYER =
struct
  let make_move = ...
end

module Computer : PLAYER =
struct
  let make_move = …
  let minimax (p : position) (d : int) (x : PLAYER) = …
     (* as per the minimax spec *)
end

module Initializer =
struct
  module Player1 = Human
  module Player2 = Computer

  let rec play = (* if game_over is true then print "Game
over"
    else make next move, then call play again *)
end

module Game =
struct
  type position = int list list
  type move = int
  let is_won (p : position) : PLAYER option
     (* Some 0/1 if won, None if not won *)
end
```