Checkpoint 2: Draft Specification

CS51: Final Project

Alexander Solis
Hana Kim
Stephen Xi
Kristina Hu

I. Brief Overview

In recent years, algorithmic game theory has become a popular topic of scholarly interest as Internet-based forms of interaction continue to dominate social vessels of communication. The Internet has not only fundamentally transformed such modes of communication but has also provided new computational tools for assessing virtual interaction. In such a framework, some of the major problems that must be addressed concern whether equilibria can be found and, if so, in what run time? Equilibria have been known to exist in many Internet applications such as virtual financial transactions and load balancing.

We will address these problems using simplified two-player games of perfect information. This will allow us to create specific parameters for which to analyze the construction of Internet-based interactions and mechanisms. Our main goal for this project is to create an unbeatable AI for tic tac toe, utilizing an optimized version of the Minimax algorithm using Alpha-Beta Pruning. We didn't end up implementing Alpha-Beta Pruning because including a separate text file with scores of game combinations achieved the same goal of decreasing run-time. Once we have completed this task, we hope to expand the scope of this algorithm to include games of greater complexity such as ConnectFour, Checkers, Three Men's Morris, or Othello. Although we didn't implement additional games, we included enough abstraction in our code to easily apply our signatures and modules to other games. Time permitting, we would like to implement additional features that include a visualization of the game board and increased interaction with the user (eg. hint provider). We didn't create a GUI for our board because we wanted to prioritize the implementation of our algorithm. We didn't feel a hint provider was necessary since the AI is unbeatable.

II. Feature List

Core Features: Minimal-complete final project

- We will implement the mini-max algorithm as described at http://web.stanford.edu/~msirota/soco/minimax.html
- User interface will be through the terminal

1 2 3
4 5 6
7 8 9

- Users will indicate which spot they would like to take by inputting the number of the space (1,2,3, etc.)

X 2 3
4 5 6
7 8 9 (Ex: Input = 1)

- That space will then be marked by an X

- After a user inputs their desired spot, our program will automatically decide the best next move based on the minimax algorithm and mark it on the board with an O
- This process will repeat until our program has won or drawn
- The program will then print out who won/drew

Cool Features As mentioned before, none of these were implemented in the final program

- Keeping score from multiple games of tic tac toe
- Implement Alpha-Beta Pruning to optimize the efficiency of our mini-max algorithm as described at http://cs.ucla.edu/~rosen/161/notes/alphabeta.html
- Augment code to work for more complex games like ConnectFour, Checkers, Three Men's Morris, or Othello.
- Create an interactive framework using Tk Python (cursor GUI) which displays the board graphically
- System that provides hints to human user

III. Technical Specification

Necessary Architecture

- **class Board**: a class that defines the game board for our Unbeatable Tic Tac Toe.
  - **def __init__(self)**: initializes the Board. The board is a 2D array, with positions specified by coordinates [x, y] (on the backend -- users will still input spaces on the frontend with digits 1 through 9). At first, all positions are empty.
    - **self.player = 'X':** the user
    - **self.ai = 'O':** our AI will always play O and go second. This can be modified later.
    - **self.empty = '.':** periods to denote empty spaces
    - **self.size = 3:** specifies the size of the Tic Tac Toe board. This can be modified later
    - **self.fields = {}**: contains the full set of 9 spaces on the board, [0,0] through [2,2]. At first, all 9 spaces are initialized to empty, or '.'
  - **def move(self, x, y)**: takes in an [x, y] point as arguments and marks the spot specified as 'X' or 'O' depending on who is playing. Returns the updated board.

    - **def __minimax(self, player)**: the algorithm function. Takes in a boolean value, player, which determines whose turn it is, and checks first to see if the game has been won or tied. Otherwise, it evaluates the possible game states and returns the optimal next move for the player. Our minimax algorithm ended up going in a separate helper functions class. We split it up into two separate methods, one private and one public.
    - **def best(self)**: returns the best possible position for a given player on its turn.
    - **def tied(self)**: returns false if any one position on the board is empty, else true (all spots are filled and no one has won → tie).
    - **def won(self)**: returns the winning configuration, if game has been won.
- **class GUI**: *to be implemented after the basic functionality of Unbeatable Tic Tac Toe works. This class contains our graphics, I/O and user interfacing functions.*

Ways to modularize this project include:

- Create the framework of the Board, Player, Minimax, etc. together. Split up details of Minimax algorithm as follows: split up possible formations on the Tic Tac Toe board by person, and assign points so that they can be properly sorted into a tree.
- Work on Unbeatable Tic Tac Toe together. Once it is fully functional, split up individually or in pairs to create another game such as Unbeatable Connect Four, Unbeatable Checkers, Three Mens Morris or other games which utilize the Minimax Algorithm framework.

IV. Next Steps

Language: Python

- Get environments entirely set up
    - TODO: set up a final project github repository
    - download the necessary python packages for Mac/Windows (TK and others should come installed, but we might have to run some things from the command line to make sure all is working)
- Begin playing with language and framework(s)
    - watch some Kahn academy videos to get a grasp of Python
- Need to figure out what environment to use.
- Look into what libraries we will want to use in order to execute our game