CS B551

Elements of Artificial Intelligence

Homework 3. Adversarial Search

This assignment will give the opportunity to gain expertise in adversarial search and to address issues in real-time AI. You will implement minimax with alpha-beta pruning for Checkers. Your Checkers programs will compete in a class tournament under time constraints.

Due Dates

The assignment will be submitted in two parts. Part I submissions are due by **4pm on Sunday, October 11**. Part II submissions are due by **4pm on Tuesday, October 20.** Part II submissions cannot be submitted late (they need to be available for the AIs to run the first rounds of the competition). The final rounds will be run in class.

Part I: Minimax with Alpha-Beta Pruning

For this part you will write python code for Checkers, using minimax with alpha-beta pruning, and developing a reasonable evaluation function. Your submission will contain a procedure named nextMove which take 4 arguments, as follows:

- *Board state*. This is a nested list which represents the state of the board. Initially the Board state is as follows:

```
[
[' ', 'r', ' ', 'r', ' ', 'r', ' ', 'r'],
['r', ' ', 'r', ' ', 'r', ' ', 'r', ' '],
[' ', 'r', ' ', 'r', ' ', 'r', ' ', 'r'],
[' ', ' ', ' ', ' ', ' ', ' ', ' ', ' '],
[' ', ' ', ' ', ' ', ' ', ' ', ' ', ' '],
['w', ' ', 'w', ' ', 'w', ' ', 'w', '.'],
[' ', 'w', ' ', 'w', ' ', 'w', ' ', 'w'],
['w', ' ', 'w', ' ', 'w', ' ', 'w', ' ']
]
```

Cells are numbered 1 to 32 as shown in this image.
You will notice that half of the squares on board are never used. A move (in this case for for a red coin) can be defined by a pair such as [10,15]. In this case red coin moves from 10 to 15. A move having multiple jumps can be defined by an iinput such as [10, 19, 28], where a coin at 10 jumps over coin at 15 to land on 19, and then jumps over coin at 24 to land on 28, thus

removing coin at 15 and 24 from the board.

- *Color*: is either the string "r" or the string "w", representing the color to play. 'r' and 'w' are the Men while 'R' and 'W' are the King. A piece becomes a king if it reaches the far end of the board.
- *Remaining time*: Remaining time in seconds. **For part I, your code will ignore the third argument.**
- *Remaining moves*: Remaining moves in the game which is set to 150. **For part I, your code will ignore the fourth argument. However you can use this creatively for part 2. Again this is not mandatory.**

Computer Checkers Tournament

The tournament will give you the chance to develop creative game-playing strategies and to test your code against that of others in the class.

Tournament events

Outside of class, the AIs will match all competition version submissions against one another to determine a set of finalists. The matches will be timed, as done in the provided file named gamePlay.py.

We encourage a human tournament between students in the class (participation in the human tournament is optional).

We will hold a live tournament between the best programs to determine a champion, and then have a match between the best student player and the best program. Before the matches, we will invite the authors of top-rated programs to say a few words about the secrets of their programs (e.g., their evaluation functions or methods to increase efficiency), and then watch as their systems confront each other for the machine championship!

Rules for Checkers

- Checkers has a 8x8 board just like Chess. Each player has 12 coins at start of the game. The pieces have two states “man” (the basic state at which all pieces start), which can only move forward) and “king,” which can move backward.
- A piece becomes a king when you move it to the far end of the board. You can convert multiple men to Kings. While capturing an opponent’s coin if your coin becomes king and there is an opportunity to move back and capture another coin, then you are bound to make that move backward.
- Colors are Red and White. Red starts on the top side while white on the bottom. Red always starts the game.

- Pieces can move only diagonally. Horizontal and vertical moves are not allowed.
- In order to capture a coin of the opponent you have to make a jump over it. You can capture multiple coins together by making multiple jumps but all in ONE turn only. If there is a situation in which you can capture (jump over) your opponent's coin then you have to play that move. If there are multiple such opportunities then you have choice to play either of them.
- A player wins by capturing all of the opponent's coins or by leaving the opponent with no legal move. Game can be a draw, if limit to number of moves possible is exhausted, within the given time range. However in this case, we check for the color having more coins on board and that color is declared the winner in the interest of the tournament. Thus game can be a draw iff number of possible moves is exhausted and both sides have equal number of coins on board.
- You can read more rules here: Checkers Rules

Rules (basic tournament)

- All matches will use a standard Checkers board and starting positions.
- Each match will consist of two games, each with a different player as red (going first). In case of a tie (both games are tied, or each player wins one), the match will be repeated with the time reduced., with each program being allowed to go first once This will continue until the match is decided.
- Matches will be run on silo.
- In initial (non-tie-breaking) matches, each player gets 150 seconds for the entire game. This was selected to be a generous limit, making quality of play the most important factor. If a program runs out of time or makes an invalid move, it forfeits the game.
- At each move, the call to the competition program will provide the amount of time remaining for the program's moves in the rest of the game.
- The finalists will compete in a standard single-elimination tournament, with random initial match-ups.
- The tournament is no-holds-barred, **as long as the ideas are entirely yours (you may not consult any outside sources)**. You are not constrained to use the same method(s) you did in Part I. Be creative!
- For the tournament you must submit a new file with procedure nextMove taking the three arguments above. This version must adjust its behaviour based on the time argument.

Provided Files

The following files are provided to help you get started.

gamePlay.py: Plays two agents against each other. You will be using doMove(board, position) from this file to make a move. There are other utility functions in the file which you may want to use. From the command line, this function is invoked with:
% python gamePlay.py [-t<timelimit>] [-v] player1 player2

Where player1.py and player2.py are python files that contain a nextMove function. The flag -v stands for verbose output (display the board after every turn).

getAllPossibleMoves.py: This gets all possible moves possible for a color, on the board, as a list. You will be using function getAllPossibleMoves(board, color) to get all the possible moves from the given board state in your minimax algorithm.

randomPlay.py: Sample agent that makes a random legal move

humanPlay.py: A simple python which lets you play the game as human. You can simple give your input as comma separated values. For eg: 10,15 will move your coin from 10 to 15. Similarly 10,19,26

simpleGreedy.py: Sampe agent that uses a brain-dead evaluation function, with no search

For example, you could have two random players play against each other with:
% python gamePlay.py randomPlay randomPlay
If you wanted to play simpleGreedy against randomPlay (with simpleGreedy going first), seeing all the moves, with a clock of 64 seconds:

% python gameplay.py –t64 -v simpleGreedy randomPlay

What to submit

Your entry will be submitted via canvas.

Part I: (Basic version)
A single .py file (containing a nextMove function and other functions if you need), that *must* be named <username>.py. Your nextMove procedure should take three arguments as explained in the rules section,, but for this part you can ignore the remaining time in your implementation if you want. Your evaluation function should use reasonable criteria (performance should be much better than random play) and you should explain it in the comments.

Part II: (Tournament version)
For part II, you will submit a version to be run in the tournament. Note that you **must** submit for each part (the file name for this part should still be <username>.py)., even if your tournament procedure has only minimal changes For this part you must turn in a file containing a nextMove procedure which has *a good strategy for handling time* (be sure to explain it in your comments). You can reuse other functions from your submission for part I if you want, but you may also use other strategies. Be creative!

Grading

Be sure to comment your code and to include an explanation of your methods!

Part I
- Correct min/max and a good evaluation function: 20 points
- Correct alpha-beta pruning, game play: 20 points

Part II
- Improve nextMove procedure by adding a good strategy, which you devise yourself, for making your system's play adjust to the amount of time remaining: 10 points, and submit a version able to play in the tournament

- The author of the champion program will receive a bonus of 5 points extra credit on the assignment, and the authors of the next 5 will receive a bonus of 3 points each.

Please let us know if you have any questions, and good luck!