Data Structures II - Project

Description:

The final project for this course is to build a tic-tac-toe game. The game can be played by two players, and each player could be either a human player or a computer player. You will provide three different computer players. Computer player 'R' will simply play randomly. Computer player 'L' will remember results from previous games and use this to learn how to play better. Computer player 'P' will play perfectly by searching the game space to determine the best move to make. Submissions with partial functionality must be made at 3 different due dates (given below). Note that the learning computer player implementation requires hashing and the game space search computer player will require some concepts from graphs. We will be discussing both of these topics in the coming weeks.

Provided Classes:

You have been provided a for displaying and interacting with a graphical tic-tac-toe board (TTTGUI). The getMove method returns a TTTPosition object that corresponds to the row and column in which a user clicked on the graphical board. Both of these classes are in the libraries that were provided to you when you first set up your eclipse environment at the beginning of the quarter. You have been provided with a very simple TicTacToe class that demonstrates how these classes can be used for your project.

Submission:

For each deliverable, you must submit **all** your source files (.java files) and you must include a link to your video in the comment box. If we cannot compile and run your code from the sources you provided, you will get a grade of 0, so please make sure to submit all your .java file.

Deliverable #1

The first deliverable is a working tic-tac-toe game that allows each player to be either human players or a random computer player. When the application starts, the user should be prompted to enter what kind of player (H for human or R for random computer) player 1 is and then what kind of player (H for human or R for random computer) player 2 is. Then a full game should be played (until either one of the players wins or all the spaces on the board are occupied). At the end of the game, a message should be displayed for every human player indicating whether they won or lost and the user should be prompted if they want to play another game. The application should continue playing games until the user answers 'N' to the prompt about playing another game. See the video for a demonstration of how your program should work.

Please submit a video where you answer the following questions:

- How do you prevent a human player from making an illegal move (from playing in an occupied square). Make sure to highlight the relevant code and to explain at a high level what you are doing. Do not just read your code to me.
- The Random computer player must make a random choice from all the possible moves. How do you determine the set of possible moves and then how do you choose one of them randomly? Make sure to highlight the code and explain your idea. Again, do not just read the code to me.
- How do you determine when a game is over? Highlight your code and explain what it is doing / how it is working. Do not just read your code to me. Talk in terms of the tic-tac-toe game (positions and pieces) and not in terms of variables in your program.

Deliberable #2:

The second deliverable is an extension of deliverable #1. The application should behave almost identically to deliverable #1 except that when the application starts, there should be a third type of player. Either player 1 or player 2 or both could be a learning computer player (chosen with 'L' rather than 'R' or 'H'). This learning player keeps track of a running score for each possible board position. The score of a board position is the number of times that board position has led to a win minus the number of times the board position has led to a loss. The idea is that board positions with high scores should be more likely to lead to a win. When making a move, the learning player must select a move that results I in the board position with the highest score. When more than one such move exists, the learning player is free to choose any of the maximal moves. See this clip from the move *War Games* for the inspiration for this project. https://www.youtube.com/watch?v=F7qOV8xonfY

Please submit a video where you answer the following questions:

- You will need to store scores for the various board positions in HashMap. Highlight the code for your hashing function and briefly explain your idea for the hash function and why you believe it is a good hash function. (Better hash functions will receive more points)
- As games are won and lost, you need to update the scores for the board positions reached in the game. How do you do this? Make sure to highlight the relevant code and to briefly explain at a high level how it works. Do not just read your code to me.
- The game needs to allow each of the two players to be either a human player, a random computer player, or a learning computer player. How did your structure your code so that it can handle any combination of human and computer players?

Deliverable #3:

The third deliverable is an extension of deliverable #2. The application should behave almost identically to deliverable #2 except that when the application starts, there should be a fourth type of player. Either player 1 or player 2 or both could be a perfect computer player (chosen with 'P' rather than 'R', 'H' or 'L'). This perfect computer player must look ahead to choose the best possible move. It must force a

win whenever possible and must never lose (by forcing a tie when a win is not possible). It does so by searching through the entire game space and assigning a score of 1 to any position that is a win or from which a win can be forced, a score of -1 from any position that is a loss or from which it is impossible to avoid a loss, and a score of 0 to any other position. This idea will make more sense after we discuss graphs and searches on graphs in class.

Please submit a video where you answer the following questions:

- As you traverse the state space for the game, you may arrive at the same board position with different sequences of moves. How does your code avoid recomputing the score for the same board position more than once? Highlight and explain the part of your code that does this.
- The value of a board position depends on what moves can be made from that board position. However, of all the possible next moves, the perfect AI would try to make the move that will lead a win for the AI player if it is the perfect AI player's turn while the opponent would presumably try to select the move that would lead to a loss for the AI player if it is the opponent's turn. When you search the set of all possible future moves, how do you take this into account (that your opponent will chose the best move possible from their perspective)?

Grading:

Each deliverable will be graded out of 100 points. Half the points will be determined by how well your application works. Does its behavior meet the requirements as specified in the writeup for the project? The other 50 points will be based on a video (maximum length is 3 minutes) where you answer specific questions about how your code works. As on the homework, part of the 50 points will be determined by how well you explain your code, but another part will be determined by the design choices you made. For example, a solution that is overly complex or that is not as efficient as it could be will be worth fewer points than a simpler solution that is efficient. This is assuming your code works. A general rule of thumb is that your video score cannot be higher than your code score. Your overall project grade is simply the average of the grades on the three deliverables.

Due dates:

Please make sure to see the submission folders on D2L for the submission deadlines for the three deliverables. There will be no late submission or resubmission possible on any of the deliverables.