Design Submission - Checkers

Jacob Dunn

Part I – Requirements

1. Shall initialize the screen with the pieces in the starting position **10%**
2. May have an option to select piece identifier and who goes first **5%**
3. Shall declare a legal move as diagonally forward one space that is not already occupied **10%**
4. Shall declare a jump as a diagonal move over the top of opponent’s piece **10%**
5. May allow for multiple jumps if decided as a legal move **5%**
6. Shall take an input from Player and check if that is a legal move **15%**
7. Shall reprint the board after each move made by either player **5%**
8. May show the move history in a box beside the game board **10%**
9. Shall remove opponent’s piece if Player’s piece diagonally jumps over it **5%**
10. May allow a piece to begin moving backward and forward once it reaches the opposite end **10%**
11. Shall declare the winner to be whoever removes all the other player’s pieces first **5%**
12. May add an AI opponent to play against **10%**

Part II - Functional Decomposition

* Main
* Initialize struct (**Structs** – Basic / **2D Array** - Advanced)
* setPositions (**Processing of array data** – Basic)
* While game is not over

1. playerTurn (**Pointers / string manipulation** – Basic)

* displayEndMessage (**FILE I/O** – Basic)
* setPositions()
* Initialize struct values
* Initialize 2d array values
* Call drawBoard()
* drawBoard()
* Print the board with the current 2d array values
* playerTurn()
* Get input from user
* Call isLegalMove()
* Call drawBoard()
* isLegalMove()
* Check if move is legal
* Remove opponent’s piece if necessary
* Change player\_turn
* gameOver()
* Check if someone wins, and if so, exits the main loop
* Continue the game if no one has won
* displayEndMessage()
* Print message to screen containing winner and end the game

Part III – Structure Chart

* 1. **Your structure chart reflects your functional decomposition. Arrows correctly indicate caller/callee relationship.**



Documentation Statement: No help received