REPORT

Implementation of Virtual 1v1 Checkers Fight in C++

1. Project Overview:

Our endeavor revolves around the creation of a virtual 1v1 checkers fight using the C++ programming language. This game is designed to offer an engaging platform for two players, enabling them to engage in strategic maneuvers, execute captures, and ascend their pieces to the esteemed status of kings. The key components encompass game logic, adept utilization of data structures, a user-friendly console interface, robust error handling, and an effective mechanism for determining the conclusion of the game.

2. Utilized Data Structures:

2.1 Integration of a 2D Array:

To encapsulate the game board, we implemented a 2D array. Each cell within this array corresponds to a specific position on the checkers board. This data structure significantly streamlines the representation of the board, facilitating seamless updates and displays of the game state.

2.2 Incorporation of a Linked List:

A linked list found its purpose in storing essential player information. The Player class, encompassing player names and piece types, emerged as a pivotal tool for managing player related data through the course of the game.

2.3 Effective Use of a Stack:

The winners Stack stack was instrumental in tracking the progression of winners throughout the game. This stack proves invaluable in efficiently displaying the victorious player at the conclusion of each round.

3. Addressing Challenges and Resolutions:

3.1 Overcoming Implicit Backtracking:

A significant hurdle arose in managing the promotion of regular pieces to kings upon reaching The opposite end of the board. Skillful deployment of implicit backtracking was employed to Scrutinize this condition and seamlessly promote the pieces. Though faced with initial Challenges, meticulous design of the logic for king promotion ultimately led to a successful Resolution.

3.2 Tackling the Leap-Over Mechanism:

Implementation of the leap-over mechanism, especially concerning the capture of opponent Pieces, presented a formidable challenge. The solution involved a meticulous examination of

Conditions for a legally executed leap, followed by judicious updates to the board after each Successful capture.

4. Core Logic/Algorithm, Complexity, and Potential Enhancements:

4.1 Rigorous Move Validation:

The move validation logic, a cornerstone in ensuring adherence to game rules, boasts a Commendable O(1) complexity. This efficiency arises from the swift verification of a handful of Conditions based on the current and target positions. Further optimization avenues exist, Particularly in refining conditions related to leap-over captures.

4.2 Strategically Efficient Game over Check:

The game over check, responsible for evaluating remaining pieces on the board and declaring a Winner in the presence of a solitary piece, operates at an O(n^2) complexity, where 'n' signifies The board size. While the current algorithm is robust, ongoing exploration into advanced Algorithms could unlock additional optimization potential.

5. Project Conclusion:

In summary, our project aspires to deliver a dynamic and interactive virtual 1v1 checkers fight. Challenges confronted in implicit backtracking and the leap-over mechanism were met with Meticulous design and problem-solving, culminating in a successful implementation. The Thoughtful integration of data structures such as the 2D array, linked list, and stack contributes to A well-organized and easily navigable codebase.

Although the implemented logic and algorithms exhibit efficacy, continuous efforts in Optimization stand to further elevate the game's performance. This project serves as a rich Learning experience in the realms of game development and algorithmic refinement, aiming to Provide both functionality and an educational journey for the participants.

6. Part 1- Game initials:

contributions encompassed the design and implementation of essential components, with a particular focus on player and board initialization. Below is a breakdown of my key contributions:

Variables:

- Players: A critical list storing instances of the Player class, essential for representing the game

Participants.

- winnersStack: A stack implementation that holds the names of victorious players, a fundamental aspect of tracking winners throughout the game.

- board: The 2D array representation of the checkers game board, a central data structure used for managing the game state.

Functions:

- printPlayers(): A function designed to output the names of the players, providing essential visibility into the participants.

- printWinners(): This function displays the names of winners retrieved from the stack, contributing to the game's conclusion and winner announcement.

- sett_board(int pc1, int pc2): I took charge of the initialization process by implementing this function, responsible for setting up the game board with pieces for players 1 and 2.

Game Logic:

- The primary game loop, present until the variable game_running becomes false, reflects my commitment to structuring the core game mechanics.

- Players provide their names and lowercase initial letters to represent their pieces, showcasing attention to user interaction and input handling.

- The sett_board function, a crucial part of the game's initiation, was my responsibility, ensuring the proper arrangement of the initial game board.

- Players alternate turns, make moves, and the game state is updated and displayed after each move, reflecting my contribution to the game's flow and progression.

- The game loop persists until a winner is determined, highlighting my involvement in defining the game's termination conditions.

Display Board Function:

- The display_board() function, vital for presenting the current status of the game board, falls under my contributions, emphasizing the commitment to providing a clear and user-friendly interface.

Additional Functions:

- Functions such as input(), check_move(), move(), do_leap(), king(), and game_over() that handle core game logic bear my imprint, showcasing expertise in algorithmic design and implementation.

Main Function (a work done by all of us):

- The main () function, serving as the program's starting point, was shaped by all three of us. Players input their names, initiate the game loop, and I contributed to the section where, after the game concludes, the winners are exhibited, and the option to initiate another round is presented.

Note:

- A notable mention is the inclusion of a cautionary note regarding the use of a goto statement

(play_again) for restarting the game. My awareness and advice on employing more structured constructs, such as loops and conditional statements, demonstrate a commitment to modern and best coding practices in C++.

7.Part 2: Game Loop and Input Handling

The given code consists of variable declarations, function implementations, and a main function responsible for executing a checkers game. The declared variables include strings for player names (p1, p2), characters representing player pieces and turns (pc1, pc2, turn), boolean flags indicating leap status and game continuation (leap, game_running), and integer coordinates for move positions (row1, row2, column1, column2).

The functions in the code are designed to handle different aspects of the game:

display_board(): Outputs the current state of the checkers board.

input(): Collects user input for move coordinates.

check_move(): Validates the legality of a move considering checkers rules.

move(): Executes legal moves, updating the board accordingly.

do_leap(): Manages the logic for multiple leaps if the player chooses to jump over multiple opponents in one turn.

king(): Identifies and converts regular checkers pieces to king pieces when they reach the opposite end of the board.

game_over(): Determines if the game is over by counting remaining pieces on the board.

The main function handles the initialization of the game, including player names and initial board setup. It then enters a game loop that continues as long as the game_running flag is true. Within this loop, it displays the current board state, the current player's turn, and takes input for a move. It calls functions to validate and execute the move, checks for kings, handles multiple leaps, and updates the turn. The game-over condition is evaluated at the end of each iteration.

Additional functions, printPlayers() and printWinners(), are present to display the list of players and winners at the end of the game. The code also includes comments explaining various sections and user interaction points, such as prompting for names, moves, and asking if players want to play again.

8.Part 3: Board Display and Movement Logic:

1. Display board Function:

The display_board function is responsible for rendering the current state of the checkers board. It utilizes a nested loop to iterate through each cell of the 8x8 board and prints the corresponding checkers piece. The display includes row and column labels, creating a visually appealing representation of the game board. Each player's pieces are represented by lowercase letters, and kings are represented by uppercase letters.

2. Input Function:

The input function is responsible for taking user input for moving pieces. It prompts the user to input the source row and column (from where the piece will move) and the destination row and column (where the piece will be placed). The function also includes input validation to ensure that the entered values are within the valid range of 0 to 7.

3. Check move Function:

The check move function determines if a move is legal or not. It checks various conditions such as:

Whether a non-king piece is moving backward.

If the destination location is not empty.

If the source location contains a piece to be moved.

If the piece is moving diagonally.

If the piece is moving by more than one column and only one row.

If the piece is leaping over another piece from the opposing player.

The function returns a Boolean value indicating the legality of the move.

4. Move Function:

The move function updates the game board after a legal move. It first checks if the moved piece is a king or not and updates the destination cell accordingly. The turn is then switched to the other player.

5. do leap Function:

The do leap function is responsible for handling the leap over a checkers piece. After a successful leap, it prompts the user if they want to try a second leap. If the user agrees, the function updates the source and destination positions and checks for the legality of the leap. If the leap is invalid, the function prints an error message.

6. King Function:

The king function converts regular checkers pieces to kings when they reach the opposite side of the board. It iterates through the first and last rows of the board and converts the corresponding pieces to kings.

7. Game over Function:

The game over function checks if the game is over based on the number of remaining pieces. It counts the non-empty cells on the board, and if there is only one remaining piece, the game is considered over.

This part of the code is crucial for the gameplay and ensures that the checkers game is displayed correctly, user input is handled appropriately, and moves are validated before updating the game state. The functions work together to provide a seamless and interactive checkers gaming experience.