Ludo Like UCSC

E.C Lakshan
Registration Number: 2023/CS/099
Index Number: 23000996

September 1, 2024

Chapter 1

Introduction

Ludo is a classic board game that has been enjoyed by players of all ages across the world. In this project, I have recreated the traditional Ludo game using the C programming language, with additional features to enhance gameplay. This documentation describes the implementation of the game, the special rules introduced, and the design decisions made during the development process.

Chapter 2

Game Rules and Mechanics

2.1 Traditional Ludo Rules

Ludo is a board game for two to four players. The objective is to move all four pieces from the base to the home row by rolling a six-sided dice. The first player to get all four pieces into the home wins the game.

- The face value of the dice is observed, and a piece is moved by the number of cells represented by the face value.
- To move a piece from the base to the starting square 'X', the player must obtain a six (6) by rolling the dice.
- At the beginning of the game, no piece belonging to any player will be on the standard cells unless a player has rolled a six.
- When a six is rolled, a player may move a piece from the base to X or move a piece already in the standard area by six cells. When a six is rolled, the player gets a second roll. If a six is rolled in the second roll, a third roll is also given. However, if a six is rolled for the third consecutive time, the roll is ignored, and the dice pass to the next player.
- When moving through the cells, a piece can jump over another piece. This other piece can be a piece of the same color or an opponent piece.
- When moving through the cells, if a piece lands on a cell occupied by a piece belonging to an opposing player, that piece is considered captured, and the opposing player's piece is returned to the base.
- After moving a piece from the base to X, the piece's movement is in a clockwise direction.
- If a piece is to land or pass the 'Approach' cell, the next cells after passing the 'Approach' cell will be the cells in the home straight.

- Once a piece is in the home straight, the player must roll the exact number to reach home.
- The first player to bring all their pieces home wins the game. The game may continue to find second, third, and fourth places.

2.2 Special Rules: Mystery Cells and Additional Gameplay

In addition to the traditional rules, the following special rules and features have been added to enhance the gameplay:

- **Capturing Bonus:** The traditional rules are extended by allowing the capturing player another roll as a bonus for capturing an opponent's piece.
- **Reset on Capture:** If any piece is captured and returned to the base, all information associated with that piece (such as position and status) will be reset.
- **Mystery Cell Introduction:** The game includes a mystery cell in the standard path. The mystery cell will appear on the board after two rounds have passed. It can occur randomly at any cell location in the standard path (52 cells) on a cell that, at the time of spawning, has no pieces on it. Once the mystery cell has appeared, it will remain in the same cell for four rounds before reappearing at another random location.
- **Mystery Cell Teleportation Options:** If a piece lands on the mystery cell, it will randomly select one of the following six options and teleport the piece to the specified location:
  - Bhawana (9th cell from the yellow approach cell)
  - Kotuwa (27th cell from the yellow approach cell)
  - Pita-Kotuwa (46th cell from the yellow approach cell)
  - Base
  - X of the piece's color
  - Approach of the piece's color

Chapter 3

Player Behaviors

3.1 Red Player

- Upon rolling the dice, if any opponent piece can be captured by moving the specified number of cells, the red player prioritizes capturing the opponent piece. If multiple opponent pieces can be captured by different red pieces, the player will prioritize capturing the opponent piece that is closest to its home.
- The red player consistently keeps one piece on the standard path and avoids moving another piece from the base to the path unless a six is rolled and no opponent pieces can be captured.

3.2 Green Player

- The green player prefers to maintain an empty base. Therefore, whenever a six is rolled, any pieces in the base will be moved to the X position, unless moving six cells allows the player to create a block.
- The green player's behavior is centered around emptying the base as a priority, reflecting a strategy focused on advancing pieces from the base to the playing field.

3.3 Yellow Player

- The yellow player consistently prioritizes winning above all else.
- Similar to the green player, the yellow player also prefers to maintain an empty base. Whenever a six is rolled, any pieces in the base will be moved to the X position.

- If no pieces remain in the base or the rolled value cannot transfer a piece from the base to the X position, the yellow player will prioritize capturing opponent pieces.

3.4 Blue Player

- The blue player adheres to a cyclic movement pattern. For example, if B1 is moved during the current round, B2 will be considered in the next, and so on.
- When determining the piece to be moved, the blue player prioritizes avoiding landing on a mystery cell.

Chapter 4

Data Structures and Justification

4.1 Structures Used to Represent the Board and Pieces

In the implementation of the Ludo game, several custom data structures have been defined to efficiently represent the game's components, including players, pieces, and game states. Below is an explanation and justification for the chosen structures.

4.1.1 Structure `piece`

- **index**: An integer that likely represents the piece's unique identifier.
- **location**: An integer representing the current location of the piece on the board (possibly -1 when not on the board).
- **distance**: An integer that might represent the distance the piece has traveled or some metric relevant to the game
- **captureCount**: An integer representing the number of pieces this piece has captured.
- **clockwise**: An integer that might represent movement direction or status.
- **pieceName[3]**: A character array to store a short name for the piece, e.g., ”Y1” for Yellow 1.

4.1.2 Structure `player`

- **PI[4]**: An array of ”piece” structs, representing the four pieces each player controls.

- **playerName[7]**: A character array for storing the player's name (e.g., "Yellow").
- **index**: An integer representing the player's index or unique identifier.
- **PiecesCount**$_B$*oard : Anintegercountingthenumberofpiecestheplayercurrentlyhasontheboard.***winPie**
  Anintegerforcountingthenumberofpiecesthathavesuccessfullyreachedthegoalorendcondition
- **directionSet**: A boolean to check if the movement direction has been set for the player.
- **direction**: An integer representing the direction of movement (e.g., clockwise or counter-clockwise).

4.2 Justification for the Used Structures

4.2.1 Modularity and Organization

Using structures like `piece` and `player` helps encapsulate related attributes together, making the code more modular and easier to manage. Each player has multiple pieces, so nesting the `piece` structure inside the `player` structure reflects the logical relationship.

4.2.2 Scalability

The use of arrays (`struct piece PI[4]` and `struct player players[]`) allows the game to easily handle multiple players and pieces without additional complex logic. It makes the codebase scalable, enabling you to expand or modify the number of players or pieces if the game rules change.

4.2.3 Clarity and Maintainability

Clearly defining structures and initializing them at the start of the game helps maintain clarity in the code. Any changes or debugging tasks are more straightforward because the relevant data is organized systematically.

4.2.4 Extensibility

By using specific fields (like distance, captureCount, clockwise, and directionSet), the code is designed with future enhancements in mind. These fields can accommodate additional game features, such as different movement strategies, game mechanics (e.g., capturing other pieces), and direction-based rules.

4.2.5 Game Logic Simplification

By organizing data this way, the game logic can reference player and piece attributes directly, reducing the complexity of function implementations for movement, capturing, winning conditions, etc

Chapter 5

Code Structure

5.1 File Organization

The project is organized into 3 folders:

- **Main Codes:** Core game logic and primary function calls.
  - `main.c`: Central control unit handling initialization, gameplay, and results display.
  - `logic.c`: Main game logic: piece movements, player decisions, and the game loop.
  - `types.h`: Data structure and variable definitions for the game.

5.2 Functions

5.2.1 `logic.c`

- `int diceRoll(char *playerName)`: This function simulates the rolling of a dice for a player whose name is passed as an argument.The function might use randomness to generate a dice roll result. The output value is used to determine how far a player's piece can move on the board.

- `int printFirstPlayer(struct player *players)`: This function identifies and prints the name of the player who gets to play first.It might use a random choice or some predefined rule to decide which player starts the game. Once the first player is identified, it prints the player's name.

- `void capture`$_{O}ther_{P}iece(structplayer * players, intindex, intpieceId)$:
  $This function handles the action of one player's piece capturing another player's piece. The function likelych$
  $This function moves a player's piece based on the direction chosen and the dice value rolled. It allows a player toc$

  - `void moveFromBaseToStart(struct player *player)`: This function moves a player's piece from the base area (starting point) to the starting position on the game board.It checks if a piece can be moved from the base to the starting position (possibly based on the dice roll or other rules) and updates the piece's position accordingly.

  - `void winThePlayer(struct player *players, int *winners, int index, int i)`: This function declares a player as a winner once their pieces meet the winning conditions.It updates the game state to reflect that the player has won, updates the winner list, and possibly handles any post-win actions (like printing a congratulatory message).

  - `void playerMovement(struct player *players, int diceVal, int index, int *winners)`:This function handles the movement of a player's piece based on the dice roll. It updates the position of the player's piece according to the dice roll and checks for any game events triggered by this move (like capturing another piece or winning the game).

  - `void repeatTheGame(struct player *players, int *winners)`: This function handles the logic for continuing the game, such as resetting states or allowing players to take turns again.It may handle looping through player turns, resetting variables for a new round, or restarting game play until a certain condition is met (like all but one player winning).

  - `void displayPlayerPieceStates(struct player *players)`: This function prints out the current state of each player's pieces.It outputs details such as each piece's location, distance traveled, and any other relevant states. This helps in visualizing the game state for debugging or player awareness.

  - `void printTheWinners(struct player *players, int *winners)`: This function prints the list of winners at the end of the game.It outputs the names and order of the players who have won the game, providing a summary of the results.

- `void game`$_{m}ain() : Thisfunctionisthemaindriverofthegamelogic.Itlikelyinitializesthegame, setsupplayers,$

Chapter 6

Conclusion

The Ludo game project has offered an excellent opportunity to apply C programming skills to create a functional and engaging game. Introducing new rules added a unique twist to the traditional gameplay, which required thoughtful planning and execution during both the design and implementation stages. This process not only enhanced the overall gameplay experience but also allowed for deeper exploration of programming concepts, logic structuring, and problem-solving techniques.

Chapter 7

References

- W3Schools.com. (n.d.). `https://www.w3schools.com/c/c_structs.php`
- Penguin, U. (2022, September 12). Adding color to your output from C. The Urban Penguin. `https://www.theurbanpenguin.com/4184-2/`
- GeeksforGeeks: `https://www.geeksforgeeks.org`
- GeeksforGeeks. (2022, November 11). How to clear console in C language? GeeksforGeeks. `https://www.geeksforgeeks.org/clear-console-c-language/`
- GeeksforGeeks. (2023, April 10). C typedef. GeeksforGeeks. `https://www.geeksforgeeks.org/typedef-in-c/`