Documentation of Ludo Game Code

User

September 1, 2024

1 Introduction

This document provides an overview of the C code written for a Ludo game. The code is divided into several files, each responsible for different aspects of the game. The key functions and logic are described below.

2 Function.c

The `Function.c` file contains the core functions responsible for moving pieces, checking game conditions, and handling special scenarios such as energized or sick pieces.

2.1 Key Functions

2.1.1 Movement and Positioning

The code includes logic for moving pieces based on their state (e.g., energized or sick). If a piece is energized, it moves twice as far; if it is sick, its movement is halved.

Listing 1: Handling Movement in Function.c

```
if (previous_rolls[i] == 3 && dice[i] == 3 && mark== 1) {
    printf("[%s] rolled two consecutive 3s and %s piece move back kotuwa to base
    piece[i][temp2]=0;
    position[i][temp2]=0;
}
previous_rolls[i] = dice[i];
```

2.1.2 Special States Handling

The file also includes logic for managing special states such as energized or sick pieces, influencing their movement over several rounds.

Listing 2: Handling Energized and Sick Pieces

```
if (energized[i][j] == 1 && round_roll[i][j] <= 4) {
    dice[i] = dice[i] * 2;
    ...
}
if (sick[i][j] == 1 && round_roll[i][j] <= 4) {
    dice[i] = dice[i] / 2;
    ...
}
```

3 main.c

The `main.c` file is the entry point of the program. It initializes the game, handles user input, and coordinates the gameplay.

3.1 Game Initialization

This section initializes the game board and sets up the players.

Listing 3: Game Initialization in main.c

```
int main() {
    initialize_board();
    ...
}
```

4 behavior.c

The `behavior.c` file contains the behavior logic for different players, potentially differentiated by their color (e.g., red, blue, yellow). Each player's strategy is defined in a specific function.

4.1 Red Player Behavior

The red player's behavior prioritizes moving pieces that are close to home.

Listing 4: Red Player Behavior

```
int red_player_behavior(...) {
    int closest_to_home = -1;
    int closest_distance = 52;
    for (int i = 0; i < 4; i++) {
        if (piece[0][i] == 1) {
            int distance = (52 + position[0][i] - 0) % 52;
// 0 is red's home position
            if (distance < closest_distance) {
```

```
                closest_distance = distance;
                closest_to_home = i;
            }
        }
    }
    return closest_to_home != -1 ? closest_to_home : 0;
}
```

4.2 Blue Player Behavior

The blue player moves in a cyclic manner, ensuring all pieces are evenly moved.

Listing 5: Blue Player Behavior

```
int blue_player_behavior(...) {
    static int last_moved = -1;
    for (int i = 1; i <= 4; i++) {
        int current = (last_moved + i) % 4;
        if (piece[1][current] == 1) {
            last_moved = current;
            return current;
        }
    }
    return 0;
}
```