Programming Languages

Project Report

Making a Ludo game in Go language

Amina Hukić, Eldar Gljiva, Ajdin Omeragić

January 2024

Introduction

The game we will be making in this project is Ludo.

Ludo is a classic board game that originated in India and has become popular worldwide. The game typically involves 2 to 4 players, and the goal is to move all your pieces (tokens or pawns) from the starting area to the centre of the board.

Here are the basic rules and components of the game:

Components:

1. Game Board:
   - The Ludo board is typically square and divided into a cross-shaped pattern.
   - Each arm of the cross has three columns of spaces, forming a total of 11 spaces per arm.
   - There are safe zones and home columns for each player.

2. Tokens or Pawns:
   - Each player has a set of tokens or pawns of a distinct colour.
   - The number of tokens for each player is usually four.

3. Dice:
   - A six-sided die is used to determine the number of spaces a player can move.

Rules:

1. Starting the Game:
   - Players take turns rolling the die to determine who goes first. The player with the highest roll starts.
   - Alternatively, the order of playing depends on the colour of the pieces and there is a set pattern.

2. Moving Tokens:
   - Players move their tokens based on the number rolled on the die.
   - Tokens start in the player's home column and must move around the board to reach the centre.

3. Entering the Board:
   - Players must roll a six to enter a token onto the board.
   - Once a token is on the board, the player can choose any of their tokens to move based on the die roll.

4. Safe Zones:
   - Each arm of the cross and the centre of the board have safe zones where tokens cannot be captured by opponents. Those are called safe houses.

5. Capturing Opponent's Tokens:
   - If a player lands on a space occupied by an opponent's token (excluding safe houses), the opponent's token is sent back to its home.

6. Reaching the Center:
   - Tokens move around the board and enter the centre column when the exact number is rolled.
   - The first player to get all their tokens into the safe houses wins.

7. Rolling a Six:
   - When a player rolls a six, they get an additional turn.

8. Special Rules for Doubles:
   - If a player rolls a double, they get an extra turn. (If the game uses two dice)

<u>Winning</u>:

The player who successfully moves all their tokens into the centre column first is declared the winner.

Ludo is a game that combines luck with some strategic decisions, especially in choosing which token to move and whether to capture opponents' tokens. The game is often played in a friendly and social setting, making it a popular choice for family gatherings and game nights.

<u>Fun facts</u>:

1. Ludo is believed to be derived from the ancient Indian game called "Pachisi." The game was played by Indian kings and queens and is considered one of the oldest board games still being played.

2. Ludo has different names and variations across the world. In Spain, it's known as "Parchís," in North America, it's called "Parcheesi," and in the UK, there's a similar game called "Uckers."

3. Ludo has inspired adaptations in popular culture. In some cultures, the game has been used as a metaphor or symbol in literature, movies, and art.

4. In some regions, Ludo tournaments are organized, bringing together players for competitive gameplay. These tournaments can be local or international, showcasing the game's enduring appeal.

5. Beyond entertainment, Ludo is also recognized for its educational benefits. The game involves counting, strategy, and decision-making, making it a valuable tool for learning, especially for younger players.

6. While Ludo is often considered a game of chance due to the dice, there are strategic elements involved. Players must decide which token to move and whether to focus on advancing or blocking opponents.

7. Ludo boards come in various designs and themes. While the traditional design is a cross-shaped pattern, some versions feature different layouts, colours, and illustrations.

Research

First, we tried to find various examples of people developing this game to see how they have implemented it and what the most effective way is according to their findings.

Then, we wanted to find out more about the language in which we will develop the game, Go.

For the first part, we went on the web and tried to find people who talked about their experience, and code snippets that were explained.

We looked for already finished projects of creating a ludo game. The best we have found is a completed code in JavaScript on github, by henriquegmendes. Here is a link:
https://github.com/henriquegmendes/ludo-game-by-henrique

We found that it had a class for dice and pawns, and a main class incorporating them and structuring the game. While it is a good starting point to see how the game could be constructed, Go does not use classes as other languages do, and this code could not help us much further.

We found another finished game with all the code, this time in Python. Link:
https://data-flair.training/blogs/python-ludo-game/
The code is well explained and helped us see a different way of creating it, this time without classes the way JavaScript and other languages have them.

Furthermore, we found the Golang website to be a great source of information. It gave us a lot of information, from the basics of how to install Go and use it to libraries we can use. Here is the website link:
https://www.golang.company

Implementation

We started by making a matrix that will function as the board. Each tile will have an index, a type (start, safehouse, path), and it will either have a pawn on it or be empty. Since this way we will have many unused cells, we will only pay attention to the ones that have a labelled type and ignore the default.

Here there is a number for number, house, safe house, and path. We made a matrix called "board" and labelled every relevant cell accordingly.

We made a simple throwDice function so we have a game die.

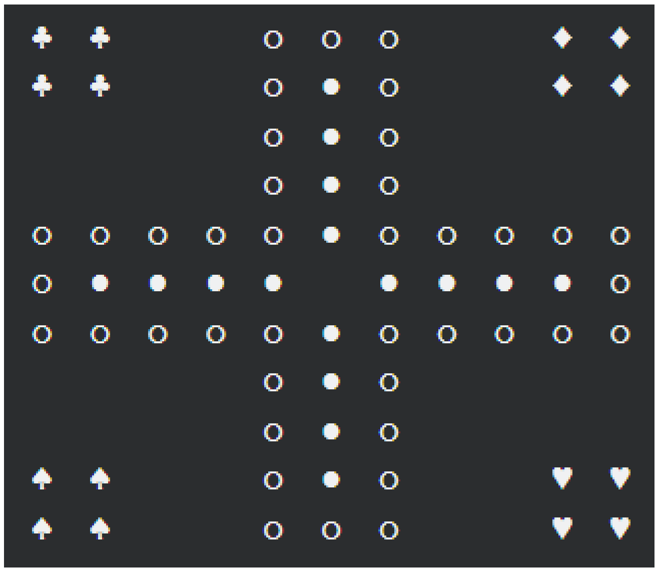
We then made a type for the player. It has colour, an InGame label which tells us if the player struct is currently in the game or not, and a smaller struct which tells us in which cell the player is.

Here we see the labels for colour and InGame. We chose card signs for showcasing different coloured players: ♣ for green, ♠ for blue, ♦ for yellow and ♥ for red. We made a function that will help us create a player and it automatically sets InGame to true since each player starts in the game.

Then we initialized all 16 players as well as a board where we set each relevant cell to its appropriate index. Afterwards, we made a print board function.

This function goes through the entire 11x11 matrix and prints each cell. If the cell has a player on it, we print the appropriate player. If the cell is a path or an empty house, we print a circle. If the cell is an empty safe house, we print a black circle. Otherwise, we print an empty space.

The cases are made so that when each player has their turn, the function prints the players' pieces as numbers so that the player can choose which piece to move by entering the appropriate number.
Here is a printed board in its initialisation:



```
♣ ♣     ○ ○ ○     ♦ ♦
♣ ♣     ○ ● ○     ♦ ♦
        ○ ● ○
        ○ ● ○
○ ○ ○ ○ ○ ● ○ ○ ○ ○ ○
○ ● ● ● ●   ● ● ● ● ○
○ ○ ○ ○ ○ ● ○ ○ ○ ○ ○
        ○ ● ○
        ○ ● ○
♠ ♠     ○ ● ○     ♥ ♥
♠ ♠     ○ ○ ○     ♥ ♥
```

Our next step is to make a function that correctly moves the player since the player moves along a path that is plus-shaped. The function looks at the row and column the player is at and moves it accordingly.

We take the player pointer, the dice number (we use the ThrowDice function), and the board so we can change it accordingly.

The function first looks at whether or not the player is in the house, we do that through board.Type, if it is a house type each colour has its designated starting point.
Then we look at the row, to simplify, if it is row 4 we move right along the columns, if it is row 6 we move left along the columns, if it is 5 we look at the column and move either up or down.
Then we look at the column, if it is column 4 we move up, if it is column 6 we move down, if it is column 5 we look at the row and move either right or left.

There are edge cases to consider, like when the player is at the end of a row/column and needs to move a specific way. All of those were taken into account, so now for each step the player takes we look at where the player is and move him to the next proper cell.

We then started modifying the functions to take into account cases of the player ending its turn on a different player and either send the player home or if the player is the same colour we move the player back one cell, we also take into account if a player is right in front of its safe house in which case the player will start walking into the safehouse.

With all of those taken into account, we modified the function and covered all the possible cases. Then, we started the process of debugging and finding other possible errors. This process consisted of playing the game and dealing with any issues found along the way. We had found some issues with printing the board and not properly keeping track of the board.HasPlayer variable, etc.

To ease our debugging process we ended up making a log file which would print important steps taken by the program, variable changes mostly.

We found that there was an inconvenience in the player does not have a piece on the board after the first three rolls, they would have to try all four pieces to move on. We fixed that by implementing a bool that checks if a piece was taken out.

We found some issues with cell types changing mid-game, but since that error happened rarely and we weren't able to replicate it we had to dismiss it.

We started getting an error mid-game where when a piece steps on a second piece, it would not compare their colour correctly. We have tried a few things, but we have not managed to find a solution. We tried comparing differently through if statements, if else, or two ifs. Since the comparison of colours seemed to be the issue, but it sometimes worked and sometimes not, we were unsure of what else we could do.

We also noticed that getting all four players in the safe house became an issue because if the cell was taken the player would continue on the board, so we had to try and fix that.

We were not able to implement requirement N.11, where in the safehouse pieces can not jump over other pieces. It would have been very complex to check each cell for each player and make sure the players do not jump over each other. Besides, it would not be performant. We would have to keep track of the individual cells and implement if statements that keep track of the order in which they have been filled.

We were also unable to implement requirement N.13, a save function. We looked at how it could be fun and found that a json file was needed, and what we would have to do is keep track of all the previous game data. Any change would have to be logged and kept track of, and since we have a lot of pieces and error-handling bools, it would have been very time-consuming to log everything down and we did not have the appropriate time and ability to implement such an option.

Conclusion

It was challenging for us to figure out a way to implement the game. When we find a way to solve one problem, often it clashes with something or creates a new problem. We were used to working with Object Oriented Programming languages, so Go presented some issues with the logic we were used to.

The main issues stemmed from the board and the way pieces interacted with it, we had to keep track of a lot of variables and often a small issue would go unnoticed until that specific case would happen in our test plays.

It was a very frustrating experience at times, and we would have to stop and look back at our previous code and see if it could be adjusted considering our new discoveries.

It showed us how important planning is, but also how adjusting your plan according to new discoveries is crucial for creating a well-made application.

References

Ludo game references:
- https://github.com/henriquegmendes/ludo-game-by-henrique
- https://data-flair.training/blogs/python-ludo-game/

Go language source:
- https://www.golang.company