Dynamically Automated Snakes and Ladders

By: Shen Reddy

1. Abstract

The objective of this project was to program an automatic game of Snakes and Ladders in C++, such that the game required no user input in order to function. The program was designed using an iterative process of trial and error. The main results of this project demonstrated that a dynamically generated game of Snakes and Ladders could be successfully produced.

2. Introduction

Snakes and Ladders is a tabletop game in which players roll a die to determine the number of places they can move forward on a board. Players can progress further than their given dice roll value if they land at the base of a ladder, which will move them forward a set number of spaces. Players can also regress down the board if they land on the head of a snake, which will move them down a set number of spaces. The game ends once a player is the first to reach the last tile on the board.

This report details the development of a dynamically generated game of Snakes and Ladders. The outlined constraints of the project were to:

- Ensure the game had no user input during the gameplay.
- Dynamically generate a minimum number of snakes and ladders, as well as their starting positions and lengths.
- Allow gameplay for up to 6 players.
- Allow multiple games to be played consecutively.

The program was required to function and produce an output solely on a given set of input conditions that were to be read in from an input file. The input conditions that needed to be provided are as follows:

- For the first game played, the board size, number of players, and 7 digit number(given in both decimal and binary), were required.
- For any game following that, only the board size and number of players were required.

Report Contents:

3. Design

The following assumptions were made for this project:

- The user could not see any active gameplay, only the final results.
- A snake head could not be placed on the same space as the base of a ladder. If this were not the case and a player landed on this space, the player would have need to move to two different locations at the same time, which is not possible, and this could result in the program crashing.

- Each snake and ladder starting position had to be unique. For the same reason as above, the player needing to move to two different locations at once would result in the program crashing.
- The length of a snake or ladder could not exceed the size of the board. If this were not the case, the program could crash as the player would need to move to a space not defined by the board space.

In order to prevent these issues from arising, several functions were designed to ensure the program functioned without errors.

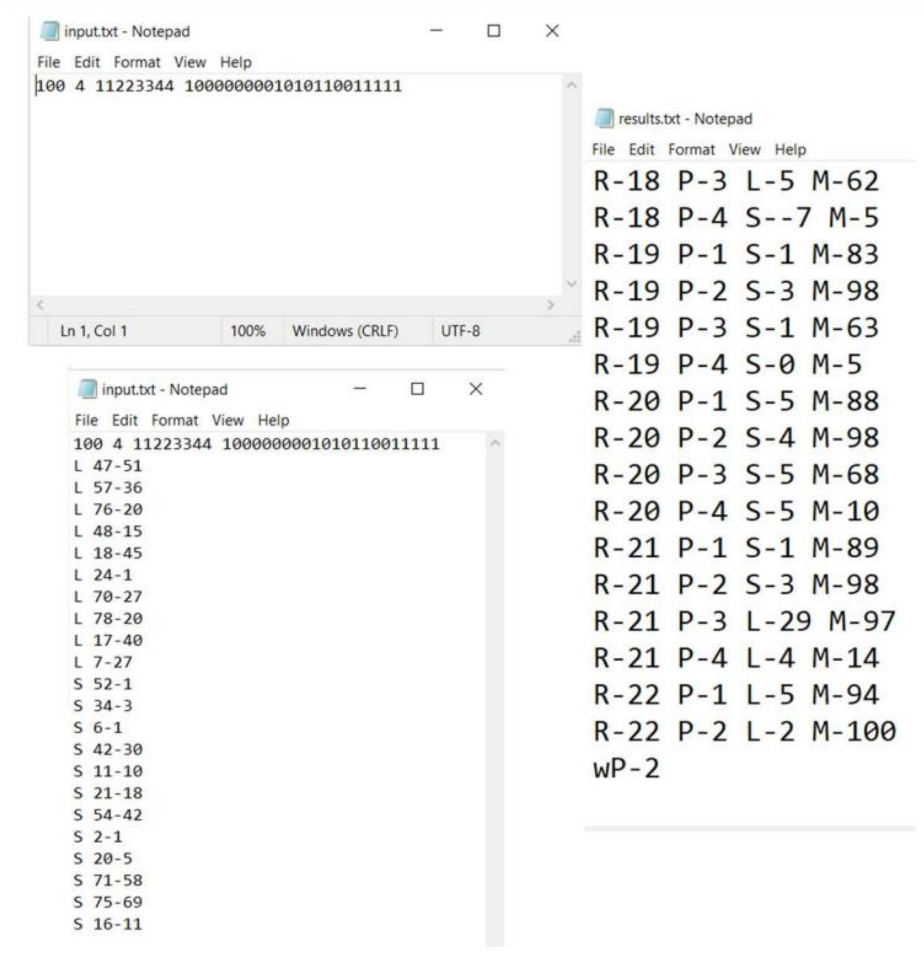
The number of snakes and ladders were generated randomly using a seeded function to ensure a different result would be produced each time the function was called. The starting position of each snake and ladder was also generated randomly. An algorithm was used to sort through each starting position of both the snakes and ladders and change and equivalent values between the number sets, such that each position generated would be unique.

A 2D vector was used to track the progress and results of the game. For each round of the game the vector would record the round number(R), the player turn(P), the die number(D), and the player moves(M). If a player landed on a snake or ladder, a string would be called to replace the D identifier with S or L respectively.

A while loop was used to track the progress of the entire game. The loop would only exit once a player was deemed to be a winner, and the game was set to game over. The process would then repeat if another game were played.

4. Results

Several tests were conducted, and all algorithms ran efficiently and produced the desired results.



```
100 4 11223344 1000000001010110011111
```

```
100 4 11223344 1000000001010110011111
L 47-51
L 57-36
L 76-20
L 48-15
L 18-45
L 24-1
L 70-27
L 78-20
L 17-40
L 7-27
S 52-1
S 34-3
S 6-1
S 42-30
S 11-10
S 21-18
S 54-42
S 2-1
S 20-5
S 71-58
S 75-69
S 16-11
```

```
R-18 P-3 L-5 M-62
R-18 P-4 S--7 M-5
R-19 P-1 S-1 M-83
R-19 P-2 S-3 M-98
R-19 P-3 S-1 M-63
R-19 P-4 S-0 M-5
R-20 P-1 S-5 M-88
R-20 P-2 S-4 M-98
R-20 P-3 S-5 M-68
R-20 P-4 S-5 M-10
R-21 P-1 S-1 M-89
R-21 P-2 S-3 M-98
R-21 P-3 L-29 M-97
R-21 P-4 L-4 M-14
R-22 P-1 L-5 M-94
R-22 P-2 L-2 M-100
wP-2
```

5. Discussion

The approach followed for the development of this project produced successful results. The initial assumptions made proved to be true and valid. By making use of several algorithms to sort and store unique values for each snake and ladder position, the program was prevented from crashing. The approach for this project could be improved by making use of better debugging practices while testing code. Initially little to no debugging was used to test individual elements of code, and instead the analysis of the functioning of the code was determined by the results from the input and output files alone. This led to increased time spent on problem solving simple issues, when using the console to output results for debugging would have significantly decreased this issue. The better approach for this project would be to use the console to test each segment of code.

6. Conclusions

The outlines of the project were successfully met. Making use of algorithms to sort and store data based on initial assumptions allowed the program to run effectively with no issues present. A dynamically generated game of Snakes and Ladders was successfully produced.

References

- Programiz.com. 2021. C++ break Statement (With Examples). [online] Available at: https://www.programiz.com/cpp-programming/break-statement [Accessed 2 June 2021]
- Programiz.com. 2021. C++ continue Statement (With Examples). [online] Available at: https://www.programiz.com/cpp-programming/continue-statement [Accessed 2 June 2021]
- Cplusplus.com. 2021. ofstream::open - C++ Reference. [online] Available at: http://www.cplusplus.com/reference/fstream/ofstream/open/ [Accessed 2 June 2021]
- Cplusplus.com. 2021. rand - C++ Reference. [online] Available at: https://www.cplusplus.com/reference/cstdlib/rand/ [Accessed 2 June 2021]
- Cplusplus.com. 2021. replace - C++ Reference. [online] Available at: https://www.cplusplus.com/reference/algorithm/replace/ [Accessed 2 June 2021]
- JournalDev. 2021. 2D Vectors in C++ - A Practical Guide 2D Vectors - JournalDev. [online] Available at: https://www.journaldev.com/42023/2d-vectors-in-c-plus-plus [Accessed 2 June 2021].

Instructions:

The game is run using a console in any IDE that can run and compile C++ code. The folder in which the .cpp file is run must also contain the provided input file in order for the program to run successfully.

To play the game, simply run the .cpp file as explained above, once the program runs and ends, an output file(results.txt) will be generated containing the appropriate results and descriptions of the game that was run.

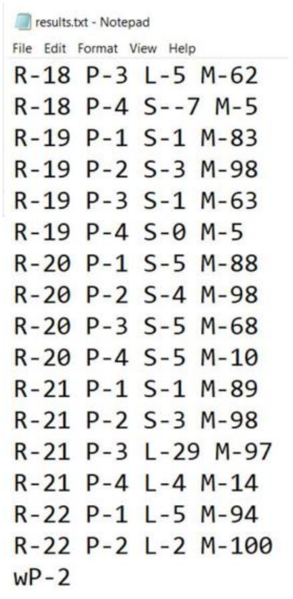
The ouput file contains the following information:

- The Round Number (R)
- The Player Number (P)
- The Die Roll Number (D)
- The Player’s New Position (M)

- The Ladder Position (L)
- The Snake Tail Position (S)
- The Winning Player (wP)
- End Condition (D) if players end in a draw

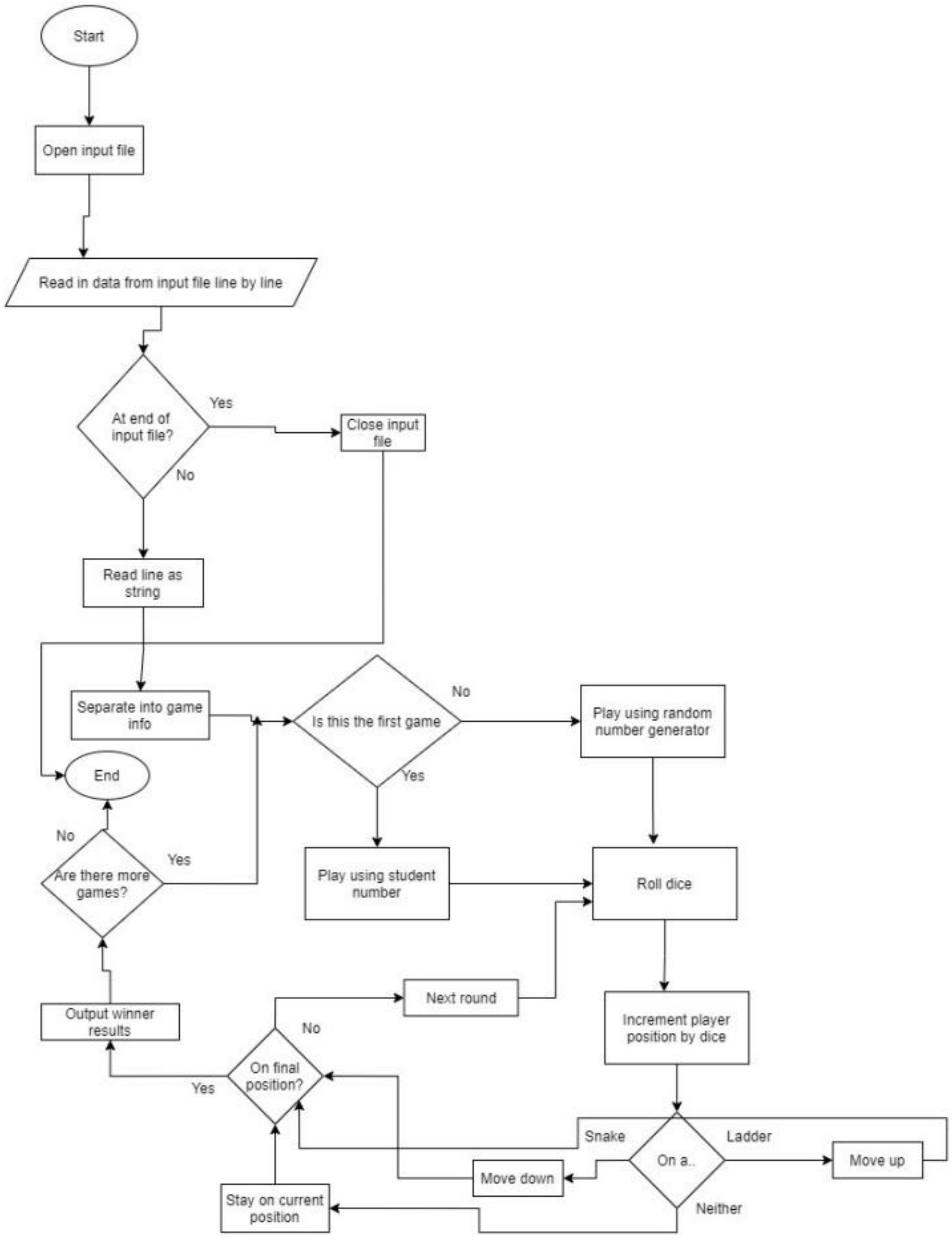
The output file will provide this information as a new line for each round until the game ends.

An Example of the output structure is given below:



```
R-18 P-3 L-5 M-62
R-18 P-4 S--7 M-5
R-19 P-1 S-1 M-83
R-19 P-2 S-3 M-98
R-19 P-3 S-1 M-63
R-19 P-4 S-0 M-5
R-20 P-1 S-5 M-88
R-20 P-2 S-4 M-98
R-20 P-3 S-5 M-68
R-20 P-4 S-5 M-10
R-21 P-1 S-1 M-89
R-21 P-2 S-3 M-98
R-21 P-3 L-29 M-97
R-21 P-4 L-4 M-14
R-22 P-1 L-5 M-94
R-22 P-2 L-2 M-100
wP-2
```

Appendix A


Start
Open input file
Read in data from input file line by line
At end of input file?
Yes
Close input file
No
Read line as string
Separate into game info
Is this the first game
No
Play using random number generator
Yes
End
No
Are there more games?
Yes
Play using student number
Roll dice
Increment player position by dice
Output winner results
Next round
No
On final position?
Yes
Snake
On a..
Ladder
Move up
Move down
Neither
Stay on current position