School of Electrical and Information Engineering
UNIVERSITY OF THE WITWATERSRAND, JOHANNESBURG

ELEN2004/ELEN2020 Software Development I – Report
Snakes And Ladders

Madiga Sehlapelo , 2321753

May 30, 2021

Abstract

In this report I will discuss the design of a board game, snake and ladders. Snakes and Ladders is a very old board game invented in the 13th Century AD by Saint Gyandev or so is believed by some historians. The purpose of this report is to illustrate one of many ways in which one can design the game's board. In the report I will go in depth in explaining how the game is played, what are the rules for winning and what are the rules for the game ending if there isn't a winner.

Contents

This page is intentionally left blank

Introduction

Snakes And Ladders

Snakes and ladders is a multiplayer board game on a grid with dimensions (*n*,*n*). It is played using a 6 sided die, the rolled number on the die determines the number of moves one can should take. The goal is to reach get to the last block of the board. To continue with the explanation of the game, I will use an example.

Take for example a 10 *by* 10 grid numbered from 1 to 100 in order. To make things a little bit more interesting, let us look at the game as a stage of life where one is working towards attaining financial success or any other type of success for that matter. One will need to roll a die to move, the rolling of the die represents chance that one gets and is independent of work put in. The board consist of snakes and ladders at certain blocks of the grid. When one lends on the base of the ladder, one moves up to the top of the ladder, and when one lend on the head of the snake, one moves to the tail of the snake. The snake can signify all the things that can take you back in terms of not attaining the success you want,god forbid, like losing a family member and the ladder on the other hand can signify entities that help you escalate quicker than luck can, things like me getting good grades for this assignment and increasing my chances of passing the course. No player can go down a ladder if the land at the top of the ladder. Each player plays in rounds, a round for a player will only change given that it is another players turn. This means that rolling again before another player plays or going up the ladder or being eaten by a snake doesn't increase nor decrease the rounds you've played. Each player can only roll again within the same round provided that the previous roll landed on a 6. In the context of the problem we are trying to solve, each player can only have a number of rounds equal to the number of blocks in the grid, in the case of our example, each player can only have a 100 rounds to play. If every player in the game has played all their 100 rounds and non has reached the end, if the is one leading player, he is considered the winner otherwise the game is terminated with a draw.
That is how the game works in a nutshell.

Board and Game Design

The number of snakes and ladders are dependent on the binary encoding of my student number with the least significant digit on the right. The number of **0**'s represent the number of snakes and the number of **1**'s represent the number of ladders. **10001101101101010011001000** is the encoding of my student number, I added 3 extra **0**'s at the end for reasons specified in the *Considerations of design* subsection.

Possible game termination outcomes

The game has one of three outcomes, either it is terminated by a single player reaching the last block on the grid and therefore winning the game, or the game is terminated by a player winning due to being on the leading position when each and every player's round reaches max, or the game terminates as a draw when more than one player are on the leading position. Testing for the final case is difficult as this depends purely on luck.

Considerations of design

Ideally we want the game to go as long as possible to afford us the chance of getting each of the outcomes above on different trials in the testing phase. As mentioned in the specifications, we wish to have more snakes than ladders, I therefore added 3 extra **0**'s in the encoding of my student number to achieve this. In addition to the number of snakes and ladders, I believe it is ideal to have most snakes be long while most ladders are short. One other non-trivial consideration is where should the snakes and ladders be placed on the board. We do not want to have most a lot of snake heads in one area which might make it impossible to move past that point, neither do we want to have to snake heads or two ladder bases on one block for obvious reasons.

The Solution

The idea here is to use the object oriented design paradigm. In this paradigm, entities that are interacting are just objects.Each object has it's own attributes and functions. In this game we will have the following objects:

- **Player:** A player who's attributes might include but not limited to the player's position, player's recent/current round and whether they moved by snake, ladder or die roll. It's functions might be to roll or dice, to set the current state of the player etc. Each player will have their own die to roll, the idea is that each player should have a different seed to the random number generator.
- **Snake and Ladder Objects:** I put this in the same bullet point as they serve the same purpose just in opposite directions. Like all other objects, these will have their own attributes and functions as well. These will have the start positions and lengths as attributes and it's functions will be nothing but getters and setters.
- **Tile:** The tile object is probably my favorite designed object, a tile represents each block on the grid or board. The interesting part of the tile object is that each and every tile on the board will have a head of a snake as well as a base of a ladder. But recall that a snake or a ladder are just objects. the snakes and ladders will have a *status* attribute, which just says whether that snake or ladder is active or not. An inactive snake or ladder will not affect any player and has length zero.
- **Board:** This is the biggest piece of the puzzle. The reason the board is an object of its own is because we can read in multiple boards from one file. Each board instance can be looked as the whole game. Each board will have a configuration of snakes and ladders on the board. As attributes, the board will have a list of players, a list of tiles ordered from to the size of the board, since the number of players and tiles is unknown before hand, a vector is the storage mechanism we are going to use to represent our list. number of players, number of snakes, number of ladders and size will also be attributes of the board object. The functionality of this object will include but not limited to, adding a snake and ladder of to the board, checking the current board status after every move to look for a winner and setting the winner of the game played on the current board object.

Each board can be seen as a game it self. The main section in the implementation will consist of three parts only and they will be as follow:

- **Reading input:** I imagine it might be difficult to in interact directly with the input file. In this section, I will read in the file and store its contents into a list, where the list is a vector since we do not know how many lines are on the page. Each line will be an entry into the list. An blank line will not be read in to our list, therefore our detection mechanism for another board will be to detect if the next line is a starts with a number or not.

- **Boards initialization:** This is the next middle piece of the whole puzzle and very crucial, no mistakes can be afforded here. The idea here is that all boards that appear in the input file should be initialized way before the first game on the first board begins. Since every board is equivalent to a game as mentioned before, every board is stored into another list of boards or games and yes, a vector will be used as the storage mechanism or list. Every board will have its set amount of players and board configuration at the end of this stage. All that's left is for the games to begin.
- **Playing the game**: This is adding meat to the bones. At this point we've read in the data and initialized all the boards. The game will probably be run in a for loop for each board. We do do not know how many rounds will be played for each player or whether or not there will be a player. The idea then is to run the game as long as there is no winner or there's no draw, this functionality will be checked by the board object. Upon every single move a player makes, the move will be recorded in the output file as well as the relevant information about the current move made. Upon every move except being bitten by a snake, we should check the board or the game for a win so that we terminate as soon as a player wins the game.

Upon playing the game, we proceed to the very next board if any exists, otherwise we terminate.

Results

In this section I discuss the results that came from our codes in the form of tables. The results here are a result from boards that are designed by the student number as well as boards that are designed to check whether the code is able it give all possible outputs, namely, winning by finishing, winning by end of rounds and draw.

Results on boards designed from the student number

Below are tables for 3 separate trials. Here we expected a winner by finishing as there is a greater chance of that happening given any board. At first there are only 3 players to test if the code can handle multiple players. and the results are self-explanatory.

Player	Rounds played	won/lost
P-1	52	lost
P-2	52	won
P-3	52	lost

10x10 original board of three player run trial one

Player	Rounds played	won/lost
P-1	39	lost
P-2	39	lost
P-3	39	won

10x10 original board of three player run trial two

Player	Rounds played	won/lost
P-1	44	lost
P-2	44	won
P-3	43	lost

10x10 original board of three player run trial three

Three Trials of 3 players on a 10 by 10 or 100 blocks on the Original Designed board

If you look at the input file, I have designed 2 boards of different sizes and different number of players. This is approximately double the size and double the amount of snakes and ladders but different lengths of course. In every trial we get different results which shows that the code to produce results purely based on luck or chance.

Player	Rounds played	won/lost
P-1	110	lost
P-2	110	lost
P-3	110	lost
P-4	110	lost
P-5	110	lost
P-6	110	won

15x15 original board of 6 players run trial one

Player	Rounds played	won/lost
P-1	72	won
P-2	71	lost
P-3	71	lost
P-4	71	lost
P-5	71	lost
P-6	71	lost

15x15 original board of 6 players run trial one

Player	Rounds played	won/lost
P-1	38	lost
P-2	38	won
P-3	37	lost
P-4	37	lost
P-5	37	lost
P-6	37	lost

15x15 original board of 6 players run trial one

Three Trials of 3 players on a 10 by 10 or 100 blocks on the Original Designed board

Results on boards designed to test conditions of winning

In this section we doctored the design of the boards and the snakes were not purely random. there are two tables as well like we had in the above subsection. The aim was to show find out whether we can achieve a draw. The game board therefore reaches a point where there are 6 consecutive snakes that have the same tail point. This then means at some point all players in the game will get stuck at a single point until they run out of rounds. As a result, both the left most tables have draws. In the middle and right most table, we made it impossible pass a certain point and by adding 6 consecutive snakes but different tail points. That way all players get stuck in the game and the winner is decided by having lead in the game.

Player	Rounds played	won/lost
P-1	100	Draw
P-2	100	Draw
P-3	100	Draw

10x10 board of three player run trial one

Player	Rounds played	won/lost
P-1	100	lost
P-2	100	won
P-3	100	lost

10x10 board of three player run trial two

Player	Rounds played	won/lost
P-1	100	won
P-2	100	lost
P-3	100	lost

10x10 board of three player run trial three

Three Trials of 3 players on a 10 by 10 or 100 blocks on a boards designed for testing

Player	Rounds played	won/lost
P-1	225	Draw
P-2	225	Draw
P-3	225	Draw
P-4	225	Draw
P-5	225	Draw
P-6	225	Draw

15x15 board of 6 players run trial one

Player	Rounds played	won/lost
P-1	225	lost
P-2	225	lost
P-3	225	lost
P-4	225	lost
P-5	225	win
P-6	225	lost

15x15 board of 6 players run trial one

Player	Rounds played	won/lost
P-1	225	lost
P-2	225	lost
P-3	225	win
P-4	225	lost
P-5	225	lost
P-6	225	lost

15x15 board of 6 players run trial one

Three Trials of 6 players on a 15 by 15 or 225 blocks on a board designed for testing

Here we can then deduce that upon many trials and most of which are not included in the report, we can confidently say that our implementation and design is a perfect solution in terms of functionality. In terms of performance and maintainability, this could be arguable but I am quite confident that this could be way far from the worst solution to the problem.

Discussion

The Approach

The aim is to complete a given task by minimizing the amount of code one will write. Having this in mind and upon reading the requirements/specifications document, the objected oriented approach made the most sense to achieve the goal. This reason I went with this approach was to ensure consistency within similar entities of the i.e. I do don't have to worry about changing players characteristics individually. This approach also saves us a huge amount of redundant code.

The Solution

Like every game, the aim is to have while playing it, but there's is no fun in a game which is easy or very difficult to win. The element of chance makes it unpredictable to an extent, but the lengths of the snakes and ladders are what we can control to measure the fun players can have. The length of snakes was informed the idea that sometimes when life knocks you, it knocks you so hard that it takes you back to where you started from. The ladders on the other hand, their lengths depict how slow one moves to attain the success they need. The whole idea however is that psychologically, people tend to be interested in things they relate to. Therefore, making our solution depict the real world made sense even though the players is just a processors adding 1's and 0's in the background playing against itself.

Improvements and Validations

I can not think of any improvements I would have made in terms of the overall performance of the implemented solution. One improvement I could have made is in terms of reducing the amount of code I wrote by making the snake and ladder object as one object. This makes it easy to keep the code maintained. The reason why this improvement could have been made is because of the fact that the snake and the ladder does the exact same job just in opposite direction. The problem with the current implementation is that if I add a function to the snake class, I need to add the exact same function to the ladder class.
One more improvement I could have added is dynamically creating the input of snakes and ladders and the writing it to a file. This would cut off the dependency on input file which I believe could be outside the scope of this assignment, or maybe not.

This assignment wasn't at all easy, the only assumptions that were made were ones in relation to the time taken to complete the project. Having played the game before many times and understanding how it works, I assumed I would have completed it sooner that I did but I didn't, and that's okay because life is like rolling a 1 and landing on the head of snake and not get what you want sometimes.

Conclusion

All boxes have been checked, code works exactly as intended, or so I believe. There are many design patterns one could have employed, the object oriented design made it significantly easier to implement the solution than just sequentially writing code that is prone to having intractable bugs. Relating life events to the game of snake and ladders also helped in keeping me motivated to get to the finish line.

References

[1] The snakes and ladders game, play online

[2] Geeks For Geeks, Programming with C++

[3] Stack Overflow Error finding and debugging

[4] Programming with C++ site:Programiz

[5] C++ Reference

Appendix

A: Project Time Management

Time management is skill, is either you have it or you suffer from doing things on time and end up not submitting. Fortunately for me, I can be skillful sometimes and therefore am able to submit a fully fledged report with all checkboxes checked. Remember I said I have a skill, and listening is one of them, I listened when the I was told to use the time breakdown below.

Activity	Time Breakdown
Background (problem understanding, requirements)	15%
Analysis & Design	20%
Implementation (coding)	20%
Testing	20%
Documentation	25%

Well maybe there was an element of being lazy for not constructing another project breakdown but this one here looks great.

Predicted Time

Through out the predicted time and the actual time. I did my best to stick to the breakdown above. Predicted time was 24hrs when split across different days. Following the above project time align, 24hrs would distributed as follow:

- **Background:** 15% => 3.6 hours. Though I have played the game several times. I have to research the problem and see if there are any different rules that were applied that weren't applied here or vice versa.
- **Analysis & Design:** 20% => 4.8 hours. Coming up with the best solution isn't easy. Time spent here should be worth it.
- **Implementation**: 20% => 4.8 hours. Coding before design and analysis takes way much time than when has planned ahead. once designed and analyzed, it should be easy to code it.
- **Testing:** 20% => 4.8 hours. Here what should take more time is constructing boards that are useful to cover all our test cases. The rest is just running the code on multiple datasets and comparing results.
- **Documentation:** 25% => 6.0 hours. Documentation is the hardest part after looking at the requirements. There are a lot of things that needs to be accounted for. This includes making sure the grammar is correct, everything is included as well as well presented.

Actual Time

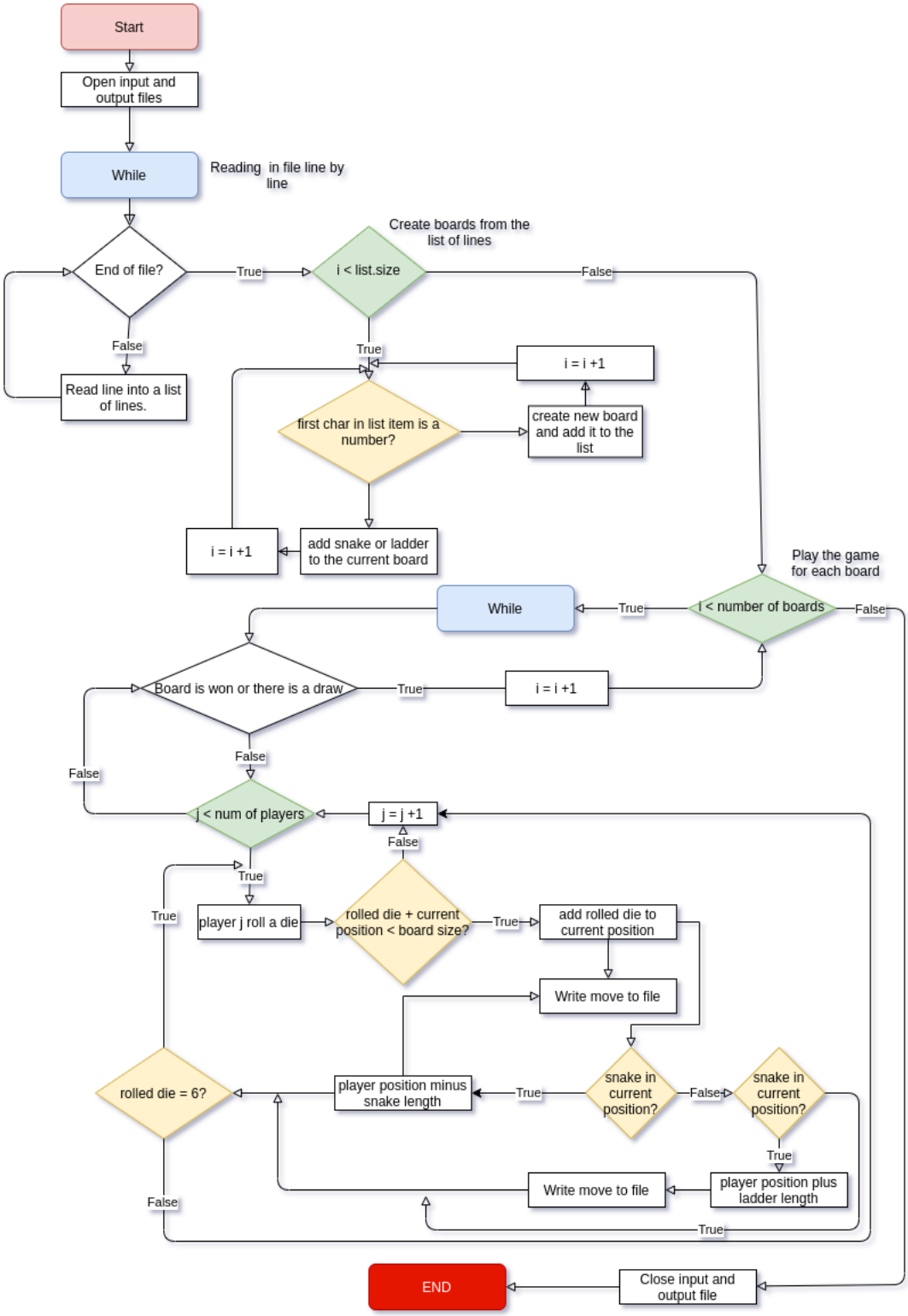
When I did the predicted time, I accounted for the margin of error. What is the worst case scenario in terms of time I will spend and what is the best case scenario. At best I thought I could finish in 20 hours and at worst 30 hours. Then

took an average of the 2 and subtracted an hour because I was confident in my abilities. The actual time came our better than the average. The actual times spent on the project was 22 hours broken down as follows.

- **Background:** 14% => 3 hours. This was because I am familiar with the game itself. I could have spent less time but I did not want to miss a detail. Time spent here includes time spent playing the online game in the docs.
- **Analysis & Design:** 18% => 4 hours. I was already familiar with object oriented design and therefore it didn't take time to get to the point of deciding what design to use. Bulk of the time was constructing diagrams on a whiteboard to see how the game would work.
- **Implementation**: 30% => 6.6 hours. Implementation could saved me way way less time. Debugging is what took time. I was stuck on the error of the random seed for rolling the die and the use of pointers in C++. Could have save an hour or 2 otherwise
- **Testing:** 13% => 2.9 hours. Testing was pretty straight forward. Bulk of the time involved looking at the output line buy line to see if it makes sense. I thought I would have spent time creating boards for testing but they were easy to construct.
- **Documentation:** 25% => 5.5 hours. Well As long as this document might seem to have taken more time than described above. In the whole process of designing and implementing the solution, I was jotting down points that need to be documented, that way documenting would take less time.

Time is the most important asset we all have. Spend time planning how you'll spend time that you don't spend while planning how to spend time, and what seems to take more time will take way less time and you'll have time to do other things, including planning for the extra time.

B: Flow Chart


Start
Open input and output files
While
Reading in file line by line
End of file?
True
False
Read line into a list of lines.
Create boards from the list of lines
i < list.size
False
True
i = i +1
first char in list item is a number?
create new board and add it to the list
i = i +1
add snake or ladder to the current board
Play the game for each board
While
True
i < number of boards
False
Board is won or there is a draw
True
i = i +1
False
False
j < num of players
j = j +1
False
True
True
player j roll a die
rolled die + current position < board size?
True
add rolled die to current position
Write move to file
rolled die = 6?
player position minus snake length
True
snake in current position?
False
snake in current position?
True
player position plus ladder length
Write move to file
False
True
END
Close input and output file