Report on ELEN2020 Course Project:

Generating Games of Snakes and Ladders Dynamically using C++

ELEN2020 – Software Development
School of Electrical and Information Engineering (EIE)
Engineering Branch: Digital Arts
Jenna Dunford
2127324

30 May 2021

I, Jenna Dunford,____________, a student at the University of the Witwatersrand, with student number 2127324, declare that this is my own work and that all resources, aides, and other contributing references have been appropriately cited and acknowledged in this document.

Abstract

The purpose of the following report is to detail and discuss the planning, design and creation of dynamically created games of snakes and ladders using. A methodical approach was taken to design and program one functionality at a time. The solution created was successful in terms of the project requirements.

1. Introduction

Snakes and Ladders is a popular children's game involving an n x n sized board, numbered, with a starting and finishing square and a six-sided die. On the board there are snakes and ladders. If a player lands at the base of a ladder, they move up to the square at the top of the ladder. If a player lands on the head of a snake, the player moves down until they reach the square at the tail of the snake.

This report details the designing, planning and implementation of dynamically created games of snakes and ladders using the C++ programming language. The specifications of this project were for the program to read in a specified board size from a file, read in a board designed using the binary equivalent of a student number, read in a different board design with a similar format, if a player lands at the head of a snake, the player moves back to the tail of the snake, if a player lands at the base of a ladder the player must be moved forward to the top of the ladder, if the generated dice roll is equal to six the player must roll again, the player must be on the last position on the board in order to win, no global variables were allowed to be used and the board's design and all of the player's moves for each round must be stored to an output file in a specified format.

This report will discuss the design of this project, testing the final program and analyzing the results and critically discussing the findings in this report. A flowchart of the program code is provided in Appendix B to be referred to when reading section 2 of this report. In Appendix A the time management of the project is shown and discussed.

2. Design

The design planning began as analyzing the various requirements of the problem. The program would need to accept data in two formats:

1. Board size, number of players, a student number and the binary conversion of the student number.
2. Board size and the number of players only.

Therefore, the program should be able to read in and recognize these two formats. The program should be able to store the board size, number of players and appropriate values for the snakes and ladders. The program should be able to output the results of the game in the specified "R – P – D – M – " format and the winner of the game and then move on to the next game. The game should allow for up to six players.

It was assumed that the input file would need to be cleared and then re-appended at some point, that the lengths of the snakes and ladders should be controlled to prevent certain game-breaking events from happening, such as a snake or a ladder taking the player off of the game board. A number of vectors would be needed as many variables of the game do not have set lengths.

The final program solution involved a game data class which would hold important aspects of the game such as the board size, number of players, student number and binary number. Two methods of determining the number of snakes and ladders were used. The first method involved using the binary value and using the number of zeroes as the number of snakes and the amount of one's as the amount of ladders. The second method involved using random numbers to determine the amount of snakes and ladders. To generate the lengths and positions of the snakes and ladders, random numbers were used. For snakes, the length of the snake was a random number between one and fifty and the position of the snake minus the length of the snake could not be zero or less. For ladders, the length of the ladder was a random number between one and fifty and the ladder's length plus the position of the ladder could not exceed the size of the board. There could not be a ladder starting on the final position of the board and a snake head could not start on the final position of the board. The positions and lengths of the snakes and ladders were stored in 2D vectors. The positions of both the snakes and the ladders were stored in a single vector, this vector was checked every time a snake or ladder position was generated to ensure that there could not be multiple snakes and or ladders on a single position on the game board.

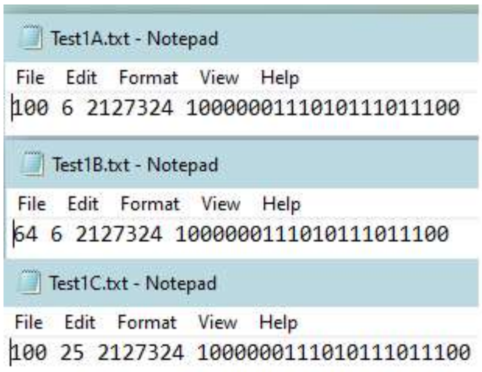
A while loop determined when the game was being played. The game itself involved a for loop that incremented for each player. For round 0, the players were initialized and then each player would be able to obtain a dice roll value between one and six for each turn. If the roll was equal to six, the player would roll again and the number of positions the player would move was incremented by the appropriate value. The program would then check if the player had landed on a snake, a ladder or a blank position. If the position was blank, the program would check if the player was on the final square, if true, the final round results would be printed to the output file and the winner would also be outputted. Then, if any more games were remaining, the next game would begin. If false, the players round results would be outputted and the for loop would move on to the next player. If the player landed on a snake, the player's position would decrement by the length of the snake, their round results would be outputted and the for loop would move on to the next player. If the player landed on a ladder, the player's position would be incremented by the length of the ladder, the program would check if the player had won and if true, the winner conditions would be outputted, if false, the round results would be outputted to the results file and the for loop would move on to the next player.

The flowchart (which can be viewed in Appendix B) shows the above process graphically. The board design (snakes, ladders and positions) were motivated by ease of implementation. The set lengths and positions of the snakes and ladders were generated in compliance with the requirements set out in the project brief. The solution that was created allows for up to and including six players and any board size. Snakes and ladders are always generated dynamically using either the binary number or random generation.

3. Results

To test the efficiency of the program two test situations were conducted. The first test used only the student number input format. The second test used the student number input format and one board size and player number format.

Test 1:


Test1A.txt - Notepad
File Edit Format View Help

```
100 6 2127324 1000000111010111011100
```

Test1B.txt - Notepad
File Edit Format View Help

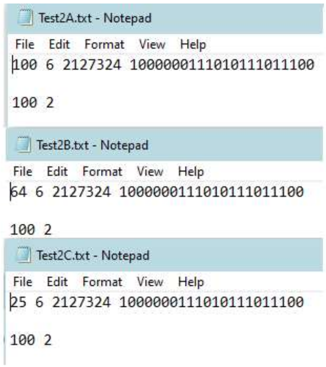
```
64 6 2127324 1000000111010111011100
```

Test1C.txt - Notepad
File Edit Format View Help

```
100 25 2127324 1000000111010111011100
```

Figure 2

Test 2:


Test2A.txt - Notepad
File Edit Format View Help

```
100 6 2127324 1000000111010111011100

100 2
```

Test2B.txt - Notepad
File Edit Format View Help

```
64 6 2127324 1000000111010111011100

100 2
```

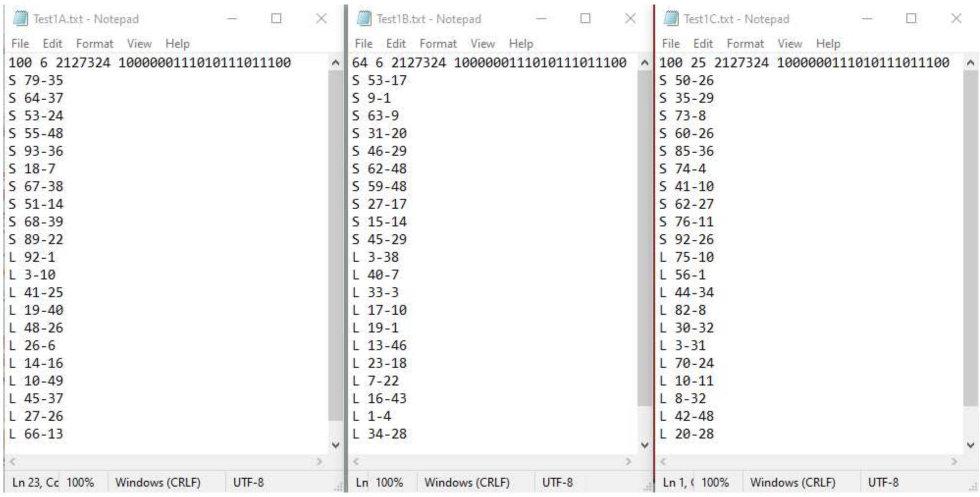
Test2C.txt - Notepad
File Edit Format View Help

```
25 6 2127324 1000000111010111011100

100 2
```

Figure 1

Test 1 appended:


Test1A.txt - Notepad
File Edit Format View Help
Ln 23, Cc 100% Windows (CRLF) UTF-8

```
100 6 2127324 10000001110101110111100
S 79-35
S 64-37
S 53-24
S 55-48
S 93-36
S 18-7
S 67-38
S 51-14
S 68-39
S 89-22
L 92-1
L 3-10
L 41-25
L 19-40
L 48-26
L 26-6
L 14-16
L 10-49
L 45-37
L 27-26
L 66-13
```

Test1B.txt - Notepad
File Edit Format View Help
Ln 100% Windows (CRLF) UTF-8

```
64 6 2127324 10000001110101110111100
S 53-17
S 9-1
S 63-9
S 31-20
S 46-29
S 62-48
S 59-48
S 27-17
S 15-14
S 45-29
L 3-38
L 40-7
L 33-3
L 17-10
L 19-1
L 13-46
L 23-18
L 7-22
L 16-43
L 1-4
L 34-28
```

Test1C.txt - Notepad
File Edit Format View Help
Ln 1, (100% Windows (CRLF) UTF-8

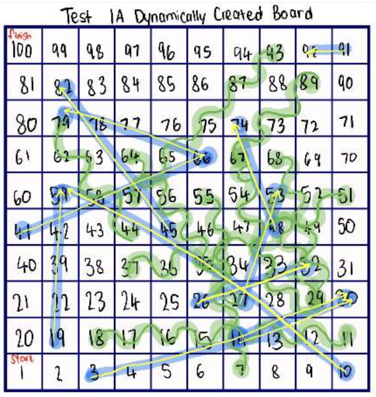
```
100 25 2127324 10000001110101110111100
S 50-26
S 35-29
S 73-8
S 60-26
S 85-36
S 74-4
S 41-10
S 62-27
S 76-11
S 92-26
L 75-10
L 56-1
L 44-34
L 82-8
L 30-32
L 3-31
L 70-24
L 10-11
L 8-32
L 42-48
L 20-28
```

Figure 3

Test 2 appended:

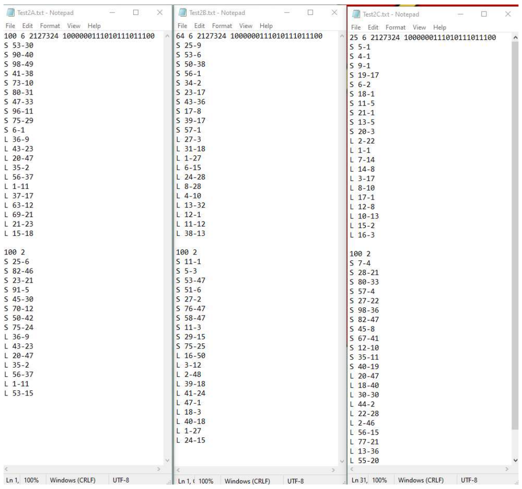
A summary of the results that were found from these tests are shown in tables 1 and 2.

Below shows the 10x10 board created by Test 1A.



100	99	98	97	96	95	94	93	92	91
81	82	83	84	85	86	87	88	89	90
80	79	78	77	76	75	74	73	72	71
61	62	63	64	65	66	67	68	69	70
60	59	58	57	56	55	54	53	52	51
41	42	43	44	45	46	47	48	49	50
40	39	38	37	36	35	34	33	32	31
21	22	23	24	25	26	27	28	29	30
20	19	18	17	16	15	14	13	12	11
1	2	3	4	5	6	7	8	9	10

Figure 4


Test2A.txt - Notepad
File Edit Format View Help
Ln 1, 100% Windows (CRLF) UTF-8

```
100 6 2127324 10000001110101110111100
S 53-30
S 90-40
S 98-49
S 41-38
S 73-10
S 80-31
S 47-33
S 96-11
S 75-29
S 6-1
L 36-9
L 43-23
L 20-47
L 35-2
L 56-37
L 1-11
L 37-17
L 63-12
L 69-21
L 21-23
L 15-18

100 2
S 25-6
S 82-46
S 23-21
S 91-5
S 45-30
S 70-12
S 50-42
S 75-24
L 36-9
L 43-23
L 20-47
L 35-2
L 56-37
L 1-11
L 53-15
```

Test2B.txt - Notepad
File Edit Format View Help
Ln 1, (100% Windows (CRLF) UTF-8

```
64 6 2127324 10000001110101110111100
S 25-9
S 53-6
S 50-38
S 56-1
S 34-2
S 23-17
S 43-36
S 17-8
S 39-17
S 57-1
L 27-3
L 31-18
L 1-27
L 6-15
L 24-28
L 8-28
L 4-10
L 13-32
L 12-1
L 11-12
L 38-13

100 2
S 11-1
S 5-3
S 53-47
S 51-6
S 27-2
S 76-47
S 58-47
S 11-3
S 29-15
S 75-25
L 16-50
L 3-12
L 2-48
L 39-18
L 41-24
L 47-1
L 18-3
L 40-18
L 1-27
L 24-15
```

```
25 6 2127324 10000001110101110111100
S 5-1
S 4-1
S 9-1
S 19-17
S 6-2
S 18-1
S 11-5
S 21-1
S 13-5
S 20-3
L 2-22
L 1-1
L 7-14
L 14-8
L 3-17
L 8-10
L 17-1
L 12-8
L 10-13
L 15-2
L 16-3

100 2
S 7-4
S 28-21
S 80-33
S 57-4
S 27-22
S 98-36
S 82-47
S 45-8
S 67-41
S 12-10
S 35-11
S 40-19
L 20-47
L 18-40
L 30-30
L 44-2
L 22-28
L 2-46
L 56-15
L 77-21
L 13-36
L 55-20
```

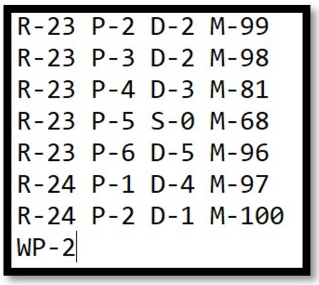
Figure 5

Table 1: Results from tests 1 A, B and C

	Rounds	Number of snakes	Number of ladders	Winner
A	24	10	11	P - 2
B	10	10	11	P - 6
C	25	10	11	P - 3

Table 2: Results from tests 2 A, B and C

Game 2 board size 100	G1 Board size	Game 1 Rounds	Game 2 Rounds	Game 1 snakes	Game 1 ladders	Game 2 snakes	Game 2 ladders	Game 1 winner	Game 2 winner
A	100	25	25	10	11	8	7	P - 6	P - 2
B	64	11	21	10	11	10	10	P - 1	P - 2
C	25	7	29	10	11	12	10	P - 4	P - 1



```
R-23 P-2 D-2 M-99
R-23 P-3 D-2 M-98
R-23 P-4 D-3 M-81
R-23 P-5 S-0 M-68
R-23 P-6 D-5 M-96
R-24 P-1 D-4 M-97
R-24 P-2 D-1 M-100
WP-2
```

Figure 6

The image to the left shows the last seven turns for the game generated from Test 1A.

The outputs from the tests done show that the program was successful in dynamically creating and playing snakes and ladders games. Each game had a winner and appropriate amounts of snakes and ladders. The winners were sufficiently randomly to show that there was no predictability in which player would win the game, in general with a smaller board size there are a fewer number of rounds, and the rounds never exceeded the size of the board.

4. Discussion

Although clearly functional, the final solution and source code is very convoluted. More effort could have been made to streamline the solution process and include more created functions for a more compact and easier to follow source code. Time management heavily effected this outcome.

The results also show that the program is functional and follows the input and result requirements that were laid out in the project brief. The attempt to make a dynamic game of snakes and ladders through a program coded in the C++ language was successful.

The initial assumptions made in the planning phase of the project were correct and many of the initial plans made did follow through successfully.

For future projects, functionally could be added to tailor the snakes and ladders positions and lengths to the size of the board. Additional functionality to include more than 6 players would also be

5. Conclusion

The principle findings of this project included discovering how to dynamically generate games of snakes and ladders using specified inputs using the C++ programming language. For future projects, more streamlined and concise programming techniques should be utilized in programming solutions of this nature.

6. References

1. Staff, C., 2021. <algorithm> - C++ Reference. [online] Cplusplus.com. Available at: <https://www.cplusplus.com/reference/algorithm/> [Accessed 1 June 2021].

2. Robinson, P., 2015. The Museum of Gaming: NewsLetter. [online] Museumofgaming.org.uk. Available at: <http://www.museumofgaming.org.uk/documents/Newsletter2.pdf> [Accessed 1 June 2021].

3. Staff, C., 2021. vector - C++ Reference. [online] Cplusplus.com. Available at: <https://www.cplusplus.com/reference/vector/vector/> [Accessed 1 June 2021].

4. Staff, T., 2021. C++ goto statement - Tutorialspoint. [online] Tutorialspoint.com. Available at: <https://www.tutorialspoint.com/cplusplus/cpp_goto_statement.htm> [Accessed 1 June 2021].

5. Verma, A., 2021. 2D Vector In C++ With User Defined Size - GeeksforGeeks. [online] GeeksforGeeks. Available at: <https://www.geeksforgeeks.org/2d-vector-in-cpp-with-user-defined-size/> [Accessed 1 June 2021].

6. Walls, C., Ledin, J., Newton, E., King-Smith, T. and Yoshida, J., 2021. Structures and classes in C++ - Embedded.com. [online] Embedded.com. Available at: <https://www.embedded.com/structures-and-classes-in-c/> [Accessed 1 June 2021].

End of Report.
Appendices are found from the next page.

Appendix A: Project Time Management

Predicted plan: Day 1 (29th May)
Write detailed plan for code (2 hours)
Start programming basic solution (3 hours)
Debug basic solution and add dynamic board creation (2 hours)

Predicted plan: Day 2 (30th May)
Ensure that program follows all guidelines of project (3 hours)
Debug any small issues (2 hours)
Begin writing report (2 hours)

Predicted plan: Day 3 (31st May)
Finish first draft of report (2 hours)
Draw flowchart (1 hour)
Add all references (2 hours)
Check report, add any necessary changes, convert to PDF (1 hour)
Check that all submittable files are properly named and zipped (30 minutes)
Submit

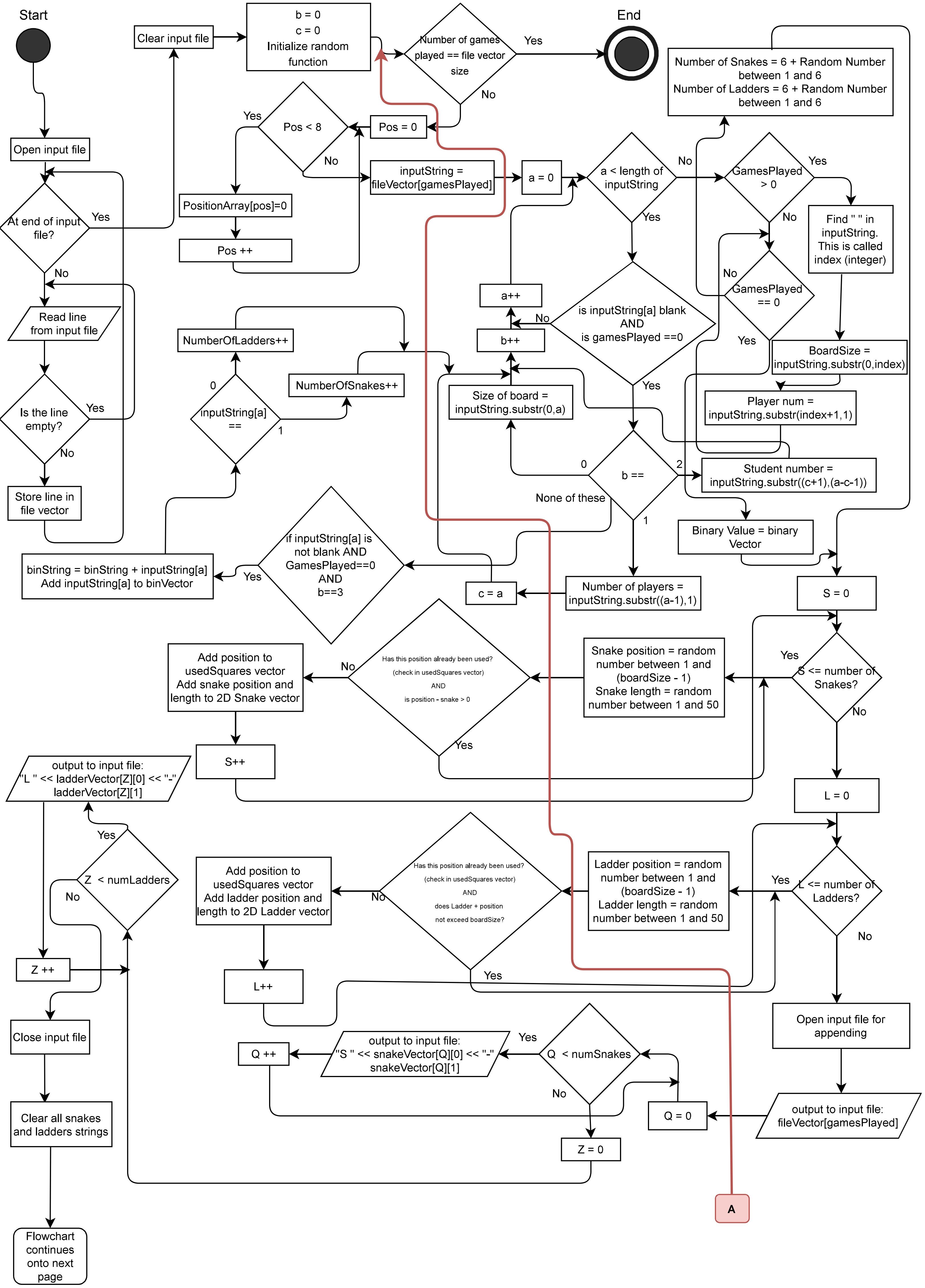
Actual time spent:
Day 1 (29th of May)
Detailing planning of code took 1 hour
Basic solution programming took 4 hours
Adding dynamic board creation took 1 and a half hours

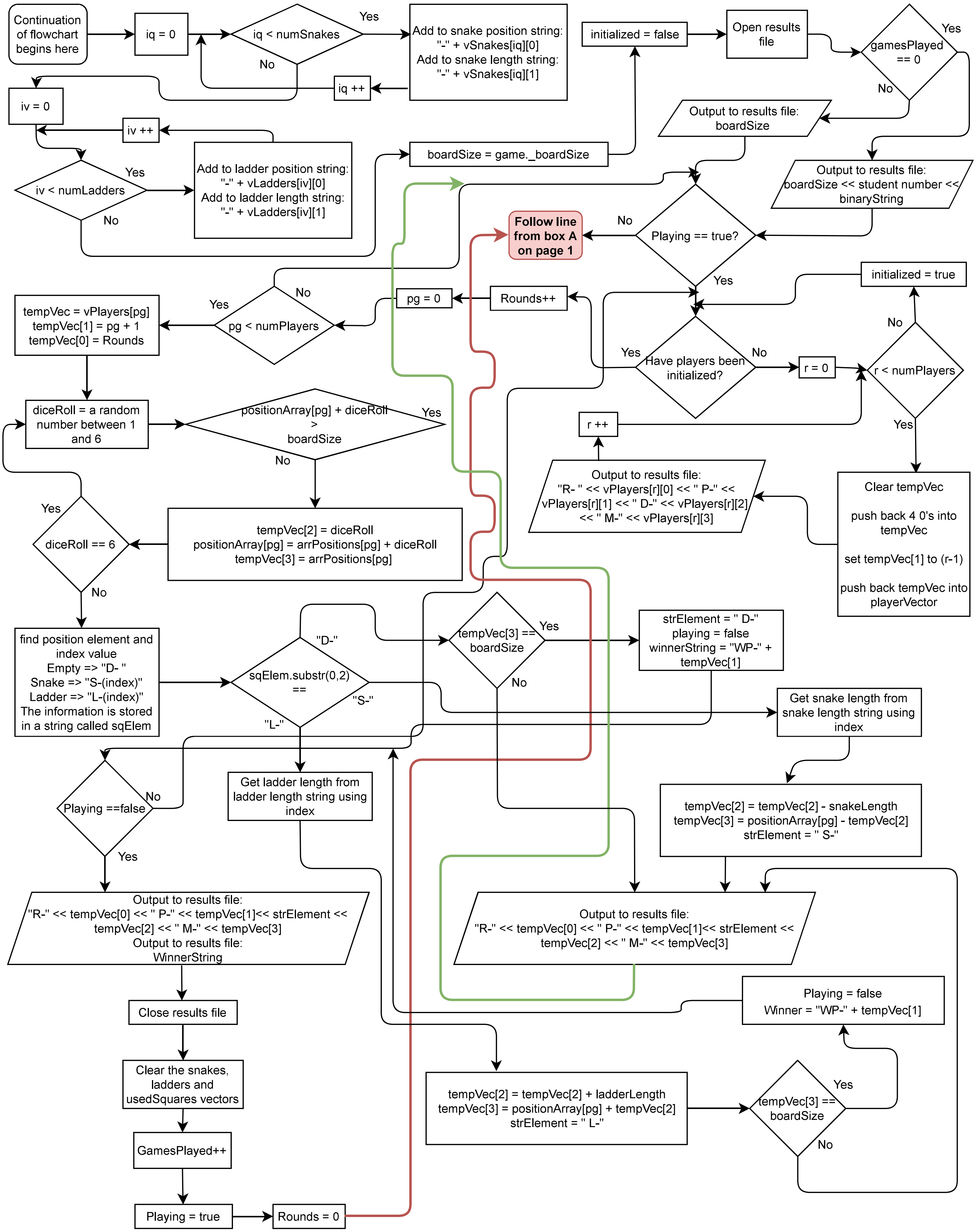
Day 2: (30th of May)
Checking that program followed all guidelines took 1 hour
Debugging of small remaining issues took 1 hour
Report planning took 1 hour

Day 3: (31st of May)
Finished first draft of report in 2 and a half hours
Drawing flowchart took 1 and a half hours
Day 4: (1st June)
Finished final report (3 hours)
Added reference list (1 hour)
Submission took place after completion of this report.

Time Management Discussion:
Initially, the intended starting date for this project was the 15th of May, which would have given two weeks of time to program and submit the project. However, on the 14th of May unprecedented obstacles occurred and health issues lead to me being hospitalized and going through surgery from which I recovered fully on the 29th of May, which is the date that this project was started.
I should have begun the project much earlier in order to prevent a situation like this from occurring. However, I do believe that my time management over these 3 days was sufficient and that I have produced an acceptable submission for this project. For future projects, I shall begin much earlier in case unforeseen circumstances arise again which would prevent me from having ample time to plan, design and submit an assignment.

Appendix B: Flowchart begins on following page


Start
Open input file
At end of input file?
Yes
No
Clear input file
b = 0
c = 0
Initialize random function
Number of games played == file vector size
Yes
End
No
Pos = 0
Pos < 8
Yes
No
PositionArray[pos]=0
Pos ++
inputString = fileVector[gamesPlayed]
a = 0
a < length of inputString
No
Yes
GamesPlayed > 0
Yes
No
Number of Snakes = 6 + Random Number between 1 and 6
Number of Ladders = 6 + Random Number between 1 and 6
Find " " in inputString. This is called index (integer)
GamesPlayed == 0
No
Yes
BoardSize = inputString.substr(0,index)
Player num = inputString.substr(index+1,1)
Read line from input file
Is the line empty?
Yes
No
Store line in file vector
is inputString[a] blank AND is gamesPlayed ==0
No
Yes
a++
b++
NumberOfLadders++
NumberOfSnakes++
inputString[a] ==
0
1
Size of board = inputString.substr(0,a)
b ==
0
2
1
None of these
Student number = inputString.substr((c+1),(a-c-1))
Binary Value = binary Vector
if inputString[a] is not blank AND GamesPlayed==0 AND b==3
Yes
binString = binString + inputString[a]
Add inputString[a] to binVector
c = a
Number of players = inputString.substr((a-1),1)
S = 0
S <= number of Snakes?
Yes
No
Snake position = random number between 1 and (boardSize - 1)
Snake length = random number between 1 and 50
Has this position already been used?
(check in usedSquares vector)
AND
is position - snake > 0
No
Yes
Add position to usedSquares vector
Add snake position and length to 2D Snake vector
S++
L = 0
L <= number of Ladders?
Yes
No
Ladder position = random number between 1 and (boardSize - 1)
Ladder length = random number between 1 and 50
Has this position already been used?
(check in usedSquares vector)
AND
does Ladder + position
not exceed boardSize?
No
Yes
Add position to usedSquares vector
Add ladder position and length to 2D Ladder vector
L++
output to input file:
"L " &lt;&lt; ladderVector[Z][0] &lt;&lt; "-"
ladderVector[Z][1]
Yes
Z &lt; numLadders
No
Z ++
Close input file
Clear all snakes and ladders strings
Flowchart continues onto next page
Open input file for appending
output to input file:
fileVector[gamesPlayed]
Q = 0
Q &lt; numSnakes
Yes
No
output to input file:
"S " &lt;&lt; snakeVector[Q][0] &lt;&lt; "-"
snakeVector[Q][1]
Q ++
Z = 0
A


Continuation of flowchart begins here
iq = 0
iq < numSnakes
Yes
No
Add to snake position string: "-" + vSnakes[iq][0]
Add to snake length string: "-" + vSnakes[iq][1]
iq ++
iv = 0
iv ++
iv < numLadders
Yes
No
Add to ladder position string: "-" + vLadders[iv][0]
Add to ladder length string: "-" + vLadders[iv][1]
boardSize = game._boardSize
initialized = false
Open results file
gamesPlayed == 0
Yes
No
Output to results file: boardSize
Output to results file: boardSize << student number << binaryString
Follow line from box A on page 1
No
Playing == true?
Yes
initialized = true
No
pg = 0
Rounds++
Yes
Have players been initialized?
No
r = 0
r < numPlayers
Yes
pg < numPlayers
No
Yes
tempVec = vPlayers[pg]
tempVec[1] = pg + 1
tempVec[0] = Rounds
r ++
Output to results file: "R- " << vPlayers[r][0] << " P-" << vPlayers[r][1] << " D-" << vPlayers[r][2] << " M-" << vPlayers[r][3]
Clear tempVec
push back 4 0's into tempVec
set tempVec[1] to (r-1)
push back tempVec into playerVector
diceRoll = a random number between 1 and 6
positionArray[pg] + diceRoll > boardSize
Yes
No
tempVec[2] = diceRoll
positionArray[pg] = arrPositions[pg] + diceRoll
tempVec[3] = arrPositions[pg]
Yes
diceRoll == 6
No
find position element and index value
Empty => "D- "
Snake => "S-(index)"
Ladder => "L-(index)"
The information is stored in a string called sqElem
"D-"
sqElem.substr(0,2) ==
"S-"
"L-"
tempVec[3] == boardSize
Yes
No
strElement = " D-"
playing = false
winnerString = "WP-" + tempVec[1]
Get snake length from snake length string using index
tempVec[2] = tempVec[2] - snakeLength
tempVec[3] = positionArray[pg] - tempVec[2]
strElement = " S-"
Playing ==false
No
Yes
Get ladder length from ladder length string using index
Output to results file:
"R-" << tempVec[0] << " P-" << tempVec[1]<< strElement << tempVec[2] << " M-" << tempVec[3]
Output to results file:
WinnerString
Output to results file:
"R-" << tempVec[0] << " P-" << tempVec[1]<< strElement << tempVec[2] << " M-" << tempVec[3]
Playing = false
Winner = "WP-" + tempVec[1]
Close results file
Clear the snakes, ladders and usedSquares vectors
GamesPlayed++
tempVec[2] = tempVec[2] + ladderLength
tempVec[3] = positionArray[pg] + tempVec[2]
strElement = " L-"
tempVec[3] == boardSize
Yes
No
Playing = true
Rounds = 0

Appendix C: List of Figures and Tables

All tables and figures were either screenshotted from relevant text files, drawn up or created by Jenna Dunford, student number: 2127324.