Assignment 2: Snakes and Ladders

SOFT7019

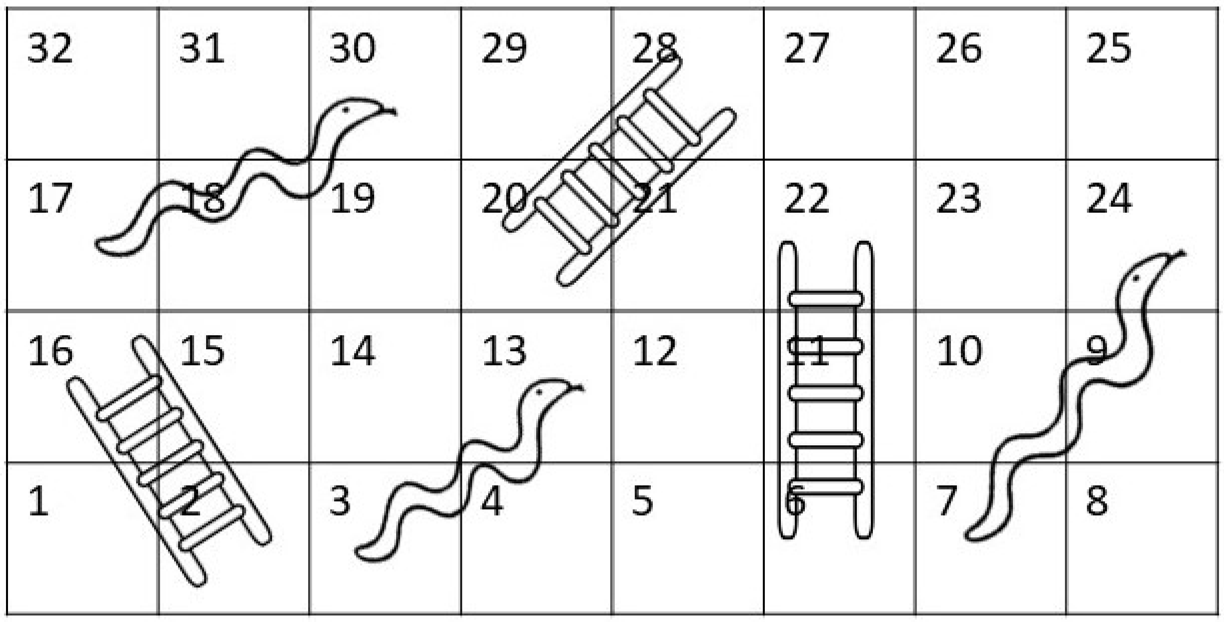
April 2022

1 Problem outline

The goal of this project is to implement a C application which can simulate a game of snakes and ladders. Snakes and ladders is a board game based on random chance. The player must navigate from the start of the board (square 1) to the final square (square n). The player can move forward a fixed number of squares based on the outcome of a single die roll; moves between 1 and 6 steps forward are possible with a 6-sided die.

Along the path there are a fixed number of snakes and ladders. If a player moves to a space with a ladder they can move directly forward to the space at the other side of the ladder. This brings them closer to the final square. If a player moves to a space with a snake they must move directly back to the space at the other side of the snake. This brings them further away from the final square.

Figure 1: Example of a typical snakes and ladders game.



The example in Figure 1 shows one possible configuration of a game board. In this example, the board consists of 25 squares which are ordered; each square has an index which represents its place on the board. Game play starts from square 1.

- If a player lands on a snake head they must move backwards e.g. if a player ends their turn on square 13 they move down the snake to square 3.

- If a player lands at the bottom of a ladder they must move upwards e.g. if a player ends their turn on square 6 they move forward to square 22.

When a player gets to the final square (square 25 for the example shown in Figure 1), they have successfully completed the game.

2 Solution requirements

In your project you will need to implement a data structure which represents the game board:

- The game board
  - Each square on the game board should be a user-defined structure which holds information about the current position on the game board. The program should check if it is the final square on the game board, or if it is the bottom of a ladder, or if it is the head of a snake.
  - The game board size should be randomly set each time the application is run, minimum size 32, maximum size 64.
  - The entire game board must be represented by a data structure of game square structures which you can traverse along, this can be implemented either using an array or a linked list.
- Snakes and ladders
  - On your game board you should randomly position a user-defined number of snakes and a user-defined number of ladders.
  - A ladder always transports the player to a higher index (cannot be outside the game board). A ladder should transport a player between 1 and 10 squares forward, this should be randomly set for each ladder upon board initialisation.
  - A snake always transports the player to a lower index (cannot be outside the game board). A snake should transport a player between 1 and 10 squares back, this should be randomly set for each snake upon board initialisation.
  - The last square on the board cannot be occupied by the head of a snake.
  - The first square on the board cannot be occupied by the foot of a ladder.
  - A square can be occupied by at most the foot of one ladder or one the head of one snake.
- Game play
  - The number of snakes and the number of ladders should be provided as two separate command line parameters when the function is called.
  - Your program should initalise a new board (of random length) and it should randomly place the number of snakes and ladders as designated by the user.
  - Your program should then simulate game play by rolling a die (random number generated between 1 and 6) and navigating through the board (following snakes and ladders where appropriate) until it gets to the final square.
  - Once the program terminates a report of the game-play should be printed to a file. This can be read by the user to determine the path their player followed through the game. This should include the start and end position of each move and whether a snake or a ladder was followed.

3 Grading (100)

- game board setup (25)
  - structure to represent squares on the board - 5
  - generate random board size and build the right sized board - 10
  - cursor to represent current position on the board - 5
  - check if you are at the final square - 5
- snakes and ladders setup (25)
  - read the number of snakes and ladders from the command line parameters - 4 (2/4 for error checking)
  - implementation of a snake - 8
    - length is a random number between 1 and 10 - 2
    - always moves the player backwards - 1
    - tail must be on the board - 1
    - the final square cannot be a snake head - 1
    - snake is a pointer to another square on the board - 3
  - implementation of a ladder - 8
    - length is a random number between 1 and 10 - 2
    - always moves the player forwards - 1
    - tail must be on the board - 1
    - the first square cannot be the foot of a ladder - 1
    - ladder is a pointer to another square on the board - 3
  - ensure that each square can have at most one ladder or one snake - 5
- game play (40)
  - function to simulate a die - 5
  - navigation through the board - 5
  - the player follows a snake if it lands on the snake head - 5
  - the player follows a ladder if it lands at the ladder foot - 5
  - printing game play report
    - connecting to the file - 5 (2/5 for error checking)
    - print the player progress - 10 (5 - basic report; 10 - messages about snakes and ladders included)
    - closing the file - 5
- code quality (10)
  - informative code comments - 5
  - informative variable names - 5

Note: implementations which use a linked list to represent the board will be graded out of 100%, implementations which use an array to represent the board will have their grades scaled to 60%.