Process Documentation

Purposes:

1. For tracing back errors arise after changing code.
2. To document the concepts for reference.
3. To explain about the problems encountered and the methods used to solve the problems.

Implementation:

1. Temporary scoreboard

* Implemented temporary scoreboard to observe the players’ movements.
* Players kept resetting back to beginning. Fixed: implemented accumulating formula to simulate players’ movements to further spaces.

1. Snakes and Ladders functions

* Implemented four arrays: snakes\_start[n], snakes\_end[n], ladders\_start[n], ladders\_end[n]. To get the same indices [i] for start and end, either snakes or ladders. Problems: More error prone as mix up between starts and ends likely happen; inefficient.
* Changed to two arrays: snakes[n], ladders[n]. index[i] is start, index[i+1] is end, either snakes or ladders. Problems: inefficient and error prone; hard to read data mixing in starts and ends in an array.
* Changed to one struct SNAKELADDER: int type as identifier it’s a snake or a ladder. Problems: inefficient to search through for a data as additional checking step required.
* Changed to two structs SNAKE, LADDER: int start and int end to store start and end spaces. Problems: had to search for matching space number (for start space in SNAKE/LADDER) to get the end space each time encountering a snake/ladder.
* Changed to include snakes and ladders locations in struct BOARD: with just int move for storing the end space of that snake/ladder, as reaching the current space with snake/ladder in it indicated that the start space.

1. Randomize snakes and ladders placements

* Implemented generate\_snakes\_ladders() function. Declared exist\_snakes\_ladders array for storing return values from generate\_snakes\_ladders(), for checking purpose to ensure no multiple snakes and/or ladders exist in the same space. Problems: inefficient, as each time storing the return value, linear search through exist\_snakes\_ladders had to be done to ensure no conflict values.
* Changed to storing return values from generate\_snakes\_ladders() directly into struct BOARD, and check for existing snakes/ladders placements by checking if current space in board has any snake/ladder value (placement).

1. Board rendering

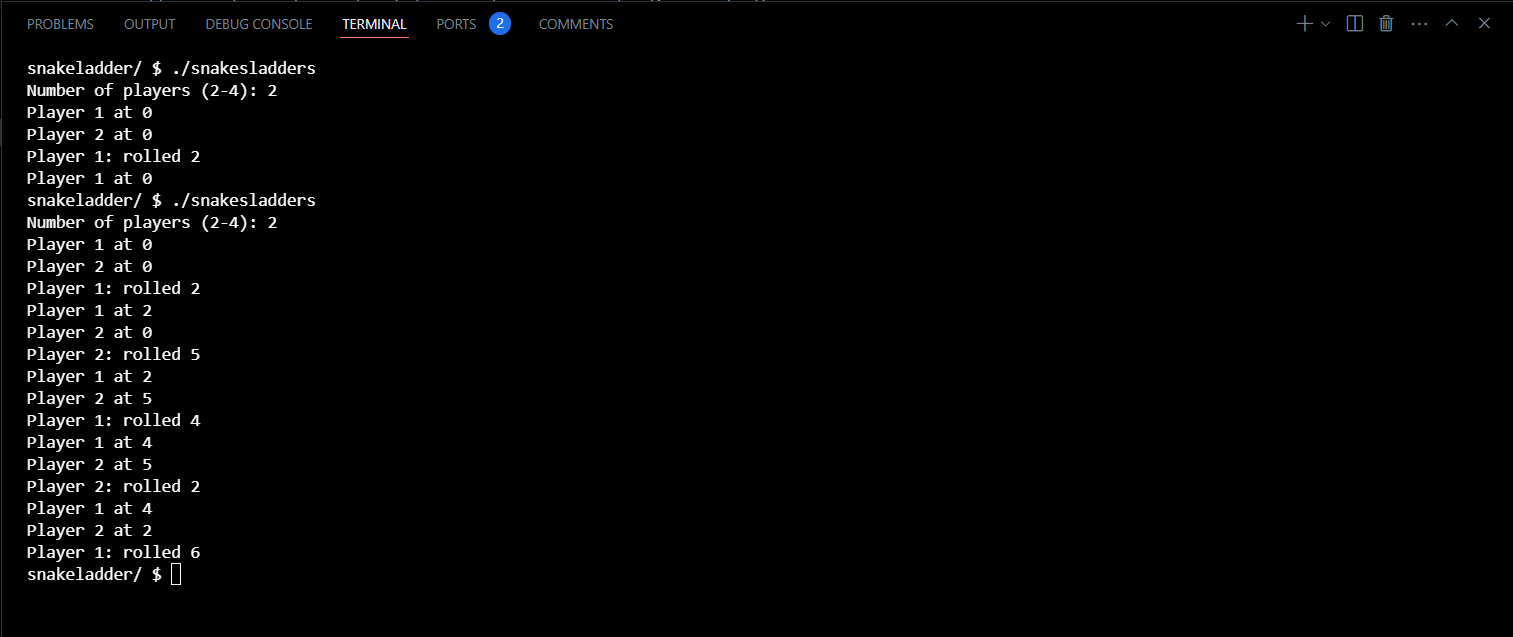
* Code design ideas:

1. Render board on the go (board rows \* cols). Problems: have to constantly check with additional functions isPlayerThere(), isSnake(), isLadder(), etc. for every move.
2. Using struct[rows][cols]. Can directly check struct[i][j] for players array, isSnakeLadder, etc., skipping unnecessary functions calls and searches.
3. Look and feel:

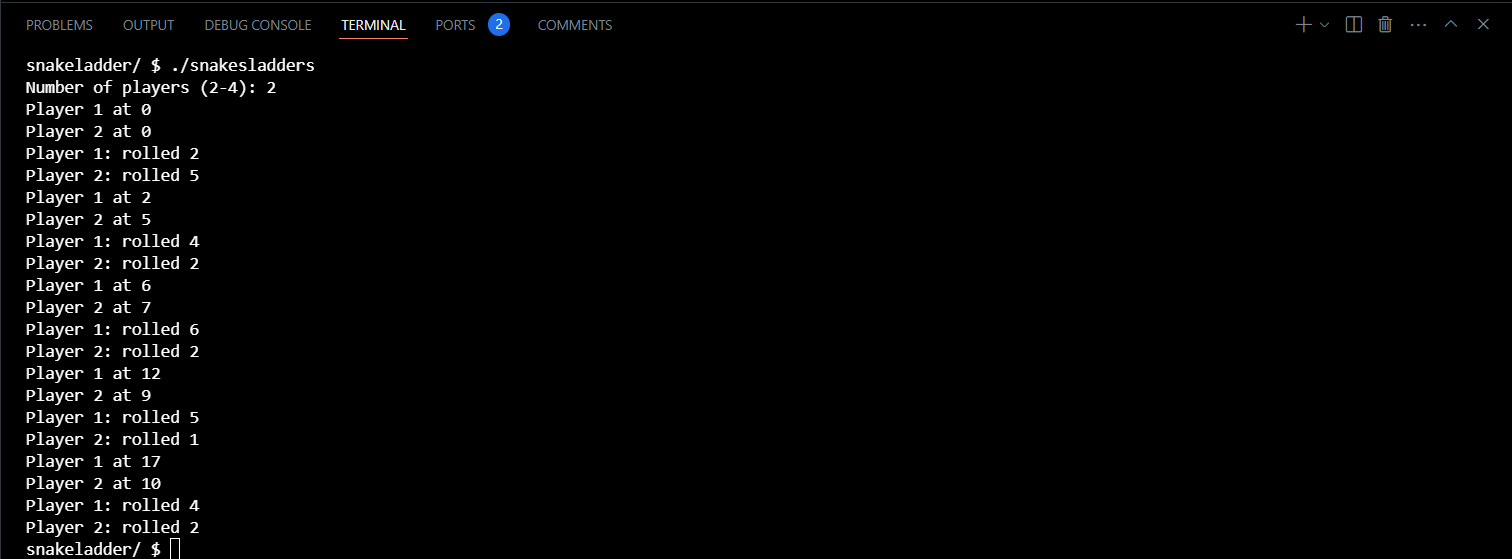
* Completely textual look.
* ASCII art look idea for version 2.

Testing Process:

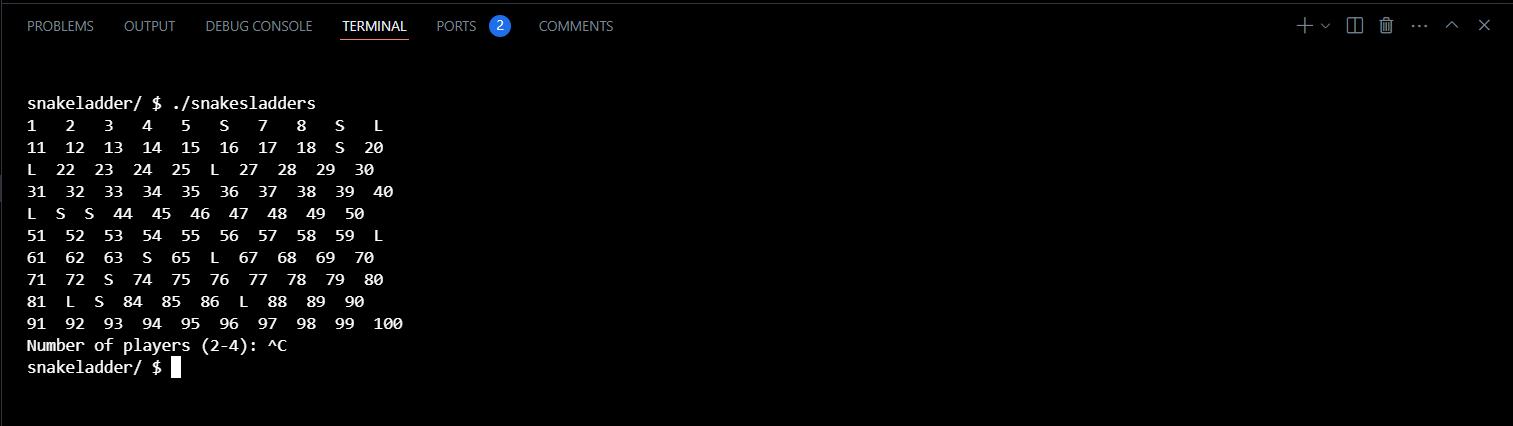
* Implemented get number of players function (2-4 players); temporary scoreboard to keep track of players’ current spaces; generate dice (randomize from 1-6).



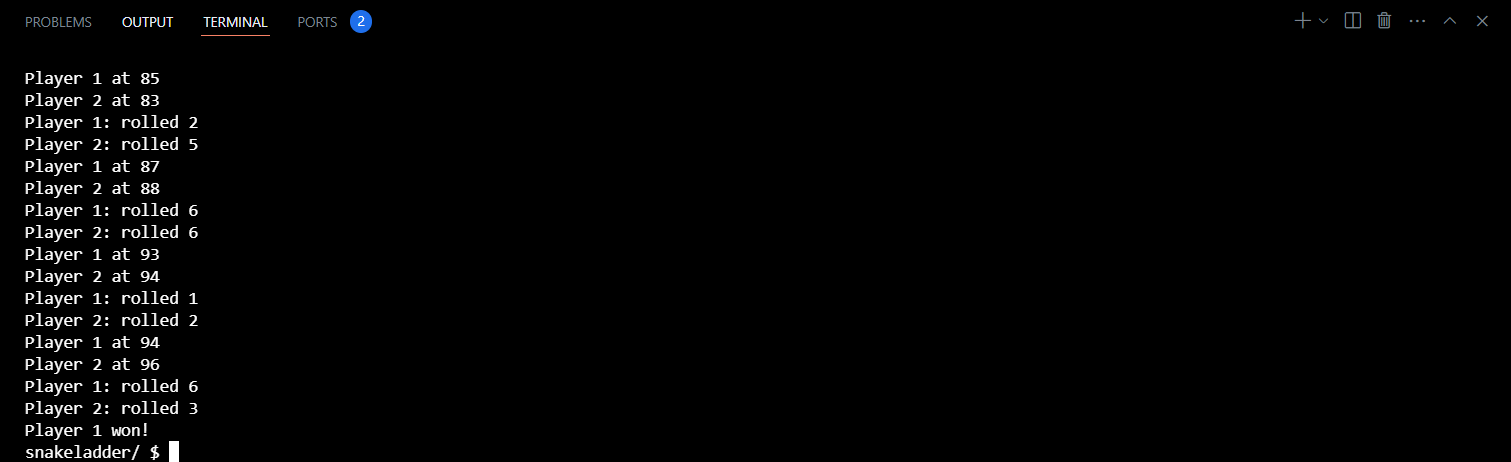
* Inefficient: only one player rolls the dice each round; players’ movements not recorded properly (not moving forward). Fixed: implemented an inner loop, all players roll the dice before moving next round; accumulating formula to record players’ rolls to simulate movements; implemented rounds function to abstract away rolling and accumulating functions.



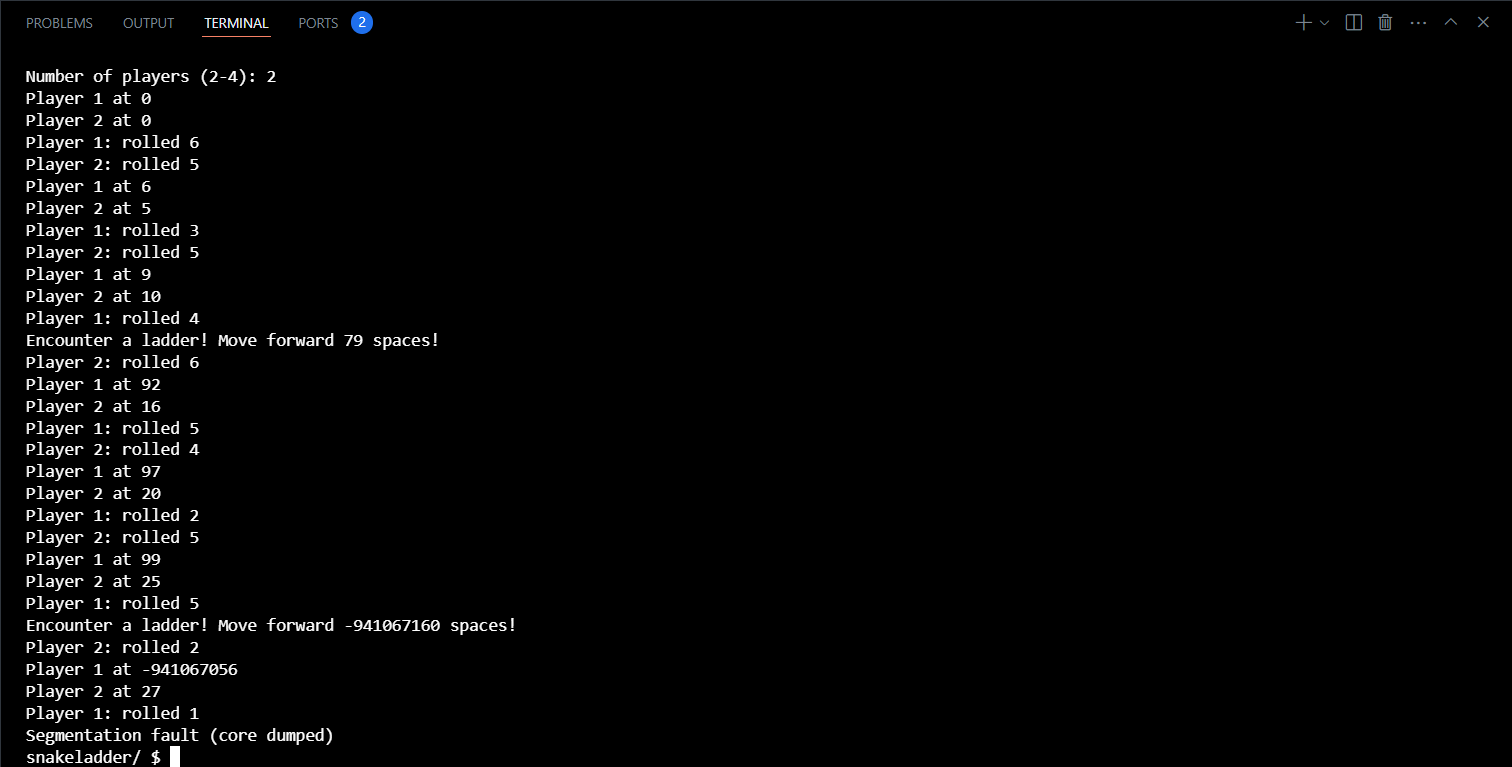
* No encounter snakes/ladders functions; no check for winner function.



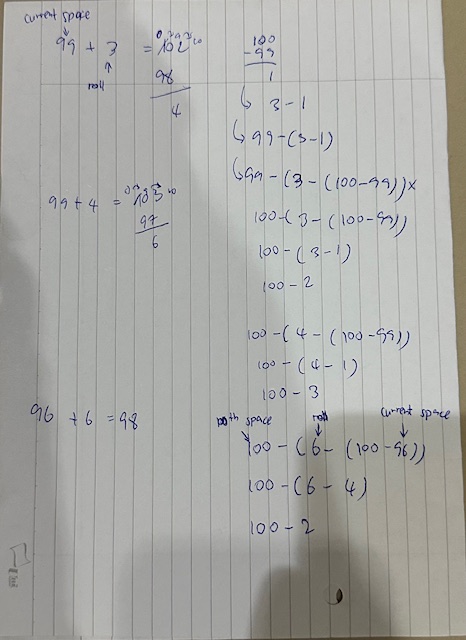
* Implemented and tested board generating function, snakes and ladders placements function with randomized locations each time re-running the program; improved retrieving snakes and ladders end points to be able to directly index into.



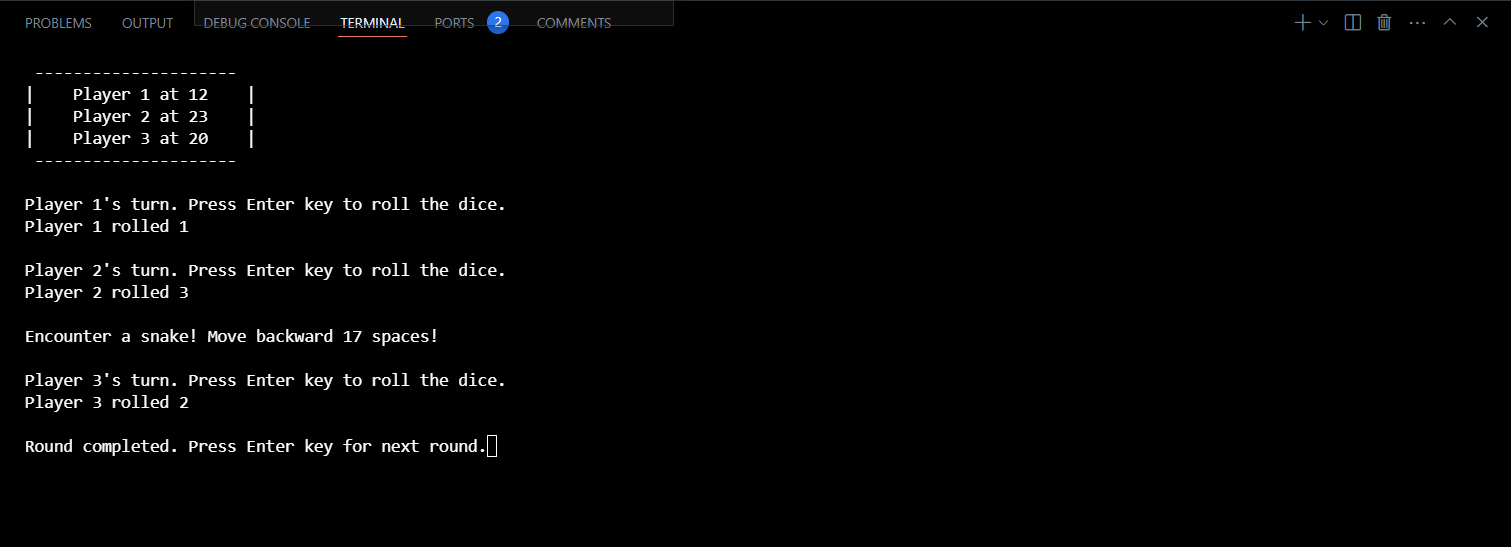
* Implemented function to check for winner and stop the game whenever a player reaches 100th space.



* Implemented encounter snake/ladder function. Segmentation fault occurred as validation when reaching over 100th space not implemented, and function calls indexed into invalid array indices (out of range).



* Fixed segmentation fault error by validating if current space + roll dice > 100, and apply the formula above for the value for current space.



* Version 1 completed. Slight improvement: cleaned up unnecessary variables; added manual intervention; cleaned up input stream that caused slight bug when added manual intervention (with just pressing Enter key to proceed with getchar()); added lines and spaces for aesthetic purposes.

Areas of Improvement (Urgent to least important):

1. Optimization - Potential board rendering.
2. Feature - Visualize board.
3. Feature - Visualize snakes and ladders.
4. Feature - Determine players’ first turns.
5. Feature - Game guides.
6. Feature - Challenge levels (normal, hard): next player reaching same space with existing players will kick existing players back to square one; more snakes.