COMP10050: Software Engineering Project 1

Assignment 2: Igel Ärgern

https://github.com/AmplifiedHuman/Igel-Argern-Game

~

Name	Student No.	GitHub
Tee Chee Guan	18202044	https://github.com/AmplifiedHuman
Rajit Banerjee	18202817	https://github.com/rajitbanerjee

~

Design specification key:

- Collaborative work
- Primarily done by Tee Chee Guan (Jason)
- Primarily done by Rajit Banerjee

Evaluation Criteria

1. The code is well commented and appropriately divided into modules (5%)
    - Added detailed comments and made changes to keep the style consistent.
2. Distribution of work among the teammates is balanced and code is often committed in the repository by both team members (10%)
    - https://github.com/AmplifiedHuman/Igel-Argern-Game/commits/master
    - We initially started with three separate branches on GitHub to try different approaches to the problem, then merged the branches to master after we were satisfied with a part of the solution.
3. The data structures adopted to represent game entities [num of players, players, board, squares (normal and obstacle), tokens per players] are appropriate (10%).
    - Added struct player to define a player of the game.
    - Modified struct token to add next token pointer used for linked list (self-referential struct).
    - Implemented push() and pop() functions for the stack (using linked list) to add or remove tokens from squares on the game board.
4. The game logic is correct (75%):
    a. Game entities: num of players, players, board, squares (normal and obstacle), tokens per player, are initialised correctly (10%)
        - Initial implementation of initialise_players() function in game_init.c file.
        - Modified initialise_players() to give players the option of choosing their token colour from available colours.
        - Initial implementation of place_tokens() function for placement of tokens in the first column of the board, before the main gameplay starts.
        - Modified place_tokens() to fix certain errors related to game rules, especially checking the token placement conditions, i.e., a player must place the token on the lowest stack, and cannot block their own token, unless all the lowest stacks have the player's colour at the top of the stack.

    - Implemented minWithDiffColour() function to help with the above check.

b. Dice rolling functionality (5%)
    - Implemented inside play_game() function.

c. Move sideways functionality (7%)
    - Fixed error in sideways move functionality in play_game() function.

d. The game does not allow a player to move sideways a token of another player (3%)
    - Implemented inside play_game() function.

e. Move forward functionality. The game allows moving forward any token that is placed on the row that is equal to the number indicated by the dice roll (5%)
    - Implemented inside play_game() function.

f. If a token lands on a square where there is already one or a stack of tokens, this token is placed on top of the stack. (10%)
    - Implemented using push() to the stack of tokens on a particular square on the board.

g. The game allows tokens that are on top of a stack to move. (5%)
    - Implemented inside play_game() function.
    - Tokens are removed from a square using pop() function on the stack of tokens. Then the token is pushed to a new selected square on the board depending on sideways/forward move.

h. The game does not allow movement of a token that is not placed on top of the stack. (5%)
    - The stack data structure only allows us to add or remove tokens from the top of the stack.

i. The game prevents moving a token placed in an obstacle square (5%)
    - Implemented blocked() function to check if the selected square is blocked and no token can be moved from that square.

j. An obstacle square becomes a normal square when there are no other tokens placed behind (on the left of) that column on any row of the board (5%)
    - Implemented inside play_game() function.

k. Game flow playability: players turns are managed in the correct order; the board is re-drawn at each turn correctly (10%)
    - Each of the above functionalities (steps b. to k.) is implemented in the game_logic.c file.
    - Implemented the play_game() function with functionalities for dice roll, possible sideways move and compulsory forward move.
    - Implemented the emptyRow() function to check if a row is empty or all tokens in row are in a deep pit (obstacle).
    - Removed certain unnecessary checks and loops in order to optimise the program.

l. The winner of the game is detected correctly. (5%)
    - Implemented the checkWin() function to check if a player has won the game, and ensured that the winner is displayed correctly after the game ends.