Software Engineering Project 1

Assignment 2 - Focus

> *Evan Brierton - 19374423*
>
> https://github.com/evanbrierton/focus/

Design Decisions

The main program is structured around a loop that repeats as long as the game as not been won, this loop repeatedly performs 4 primary actions.

- Prints information about the players and the number of pieces they have reserved and captured.
- Prints the board in its updated state.
- Prompts the player for a given action, this action is then executed by the prompt function, and the `turn` counter is incremented if the action has used up a player's turn.
- Checks if either player has lost and sets the `won` and `loser` variables accordingly.

The Player

The `Player` struct contains the following information about the player.

- Colour
- Name
- Pieces captured.
- Pieces in reserve.

The `colour` property is an `enum` value that is used to determine whether the player can perform certain actions throughout the game. The `name` property is input by the player, but this is concatenated with colour formatting strings using the `getColourString` function to print the player's name in their colour.

The reserved and captured variables keep track of the number of the player's own pieces and the number of their opponent's pieces the player has popped off the bottom of stacks respectively.

The Board and Pieces

The board is a 2-dimensional array of `Square` structs with each `square` containing a double linked list of `Piece` structs. I used a double linked list as it made popping pieces off the bottom of each stack both easier and more efficient as I was able to avoid traversing through the linked list entirely.

The presence of a `tail` in the linked list also made merging stacks easier, as rather than traversing through the linked list from the `head` to link it's bottom element to the top of another list I could just point the `next` property of the `tail` to the `head` of the other stack.

Each `square` also contains a `bool valid` to determine whether a piece can be placed on it and a `size_t height` for more efficient access to the length of the linked list.

A number of methods acting upon `Squares` have also been implemented `clear`, `setEmpty` `setInvalid`, `setPiece`, `pop`, `push` and `merge` for initialisation, placing and moving of stacks.

H3 Utility Methods

There are several utility functions implemented including `getColourString` for pretty text formatting `distanceFromEdge` and `distanceFromCorner` used in generating the board layout. `taxiCabDistance`, `validOrigin` and `validTarget` used in determining whether a player can move a piece from and to given locations on the board. `canMove`, `canPlace` and `checkWin` for calculating the win condition, and `cleanup` for freeing memory at the end of program execution.

H3 Folder Structure

The folder structure implemented is very simple, the config files and two folders named `src` and `includes` are in the root, `CMakeLists.txt` uses `add_subdirectory` to add the `src` directory who's own `CMakeLists.txt` adds the executable files and uses `include_directories` to link the `.h` files contained in the `includes` directory.