Software Engineering Project 1

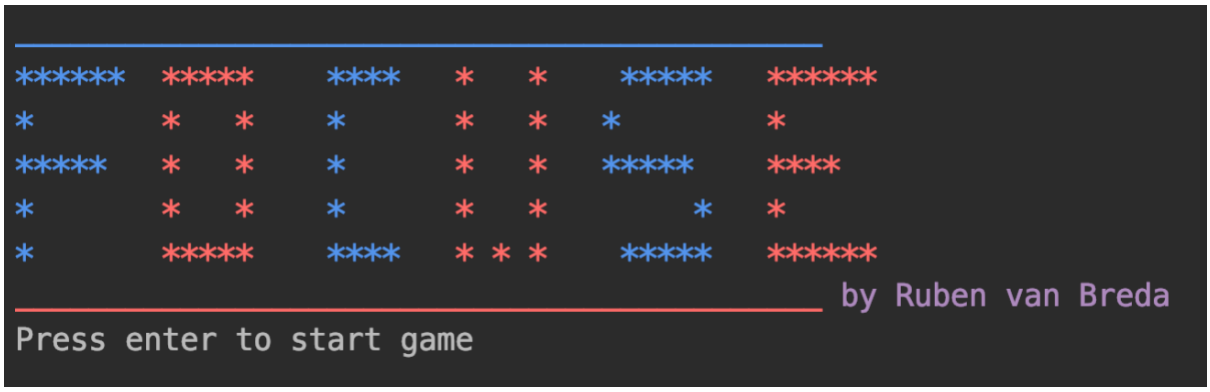
Comp10050

Assignment 2: Focus/Domination Game 27 April 2020

By Ruben van Breda

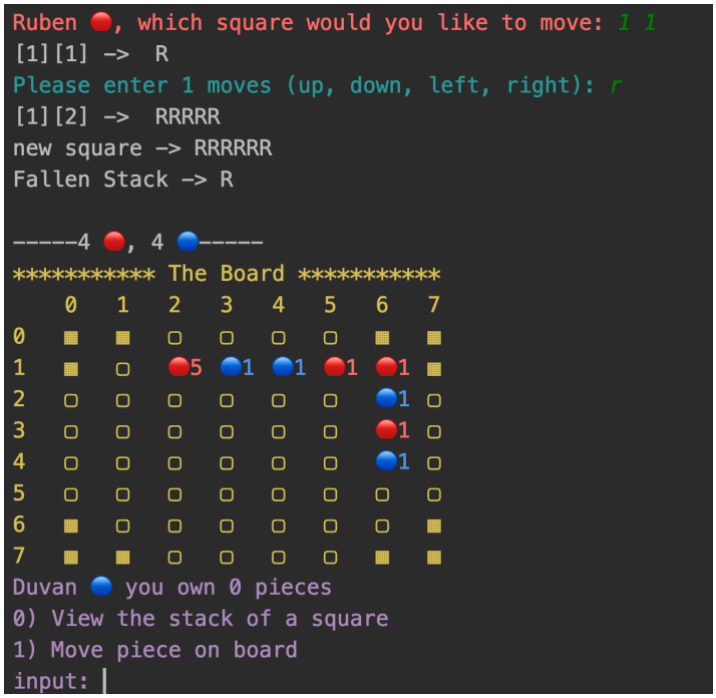
19200704

https://github.com/Ruben-van-Breda/FocusGame/tree/stable



GamePlay https://en.wikipedia.org/wiki/Focus_(board_game)

Two to four players move stacks of one to five pieces around a checkerboard with the three squares in each corner removed, thus forming a 6×6 board with 1×4 extensions on each side. Stacks may move as many spaces as there are pieces in the stack. Players may only move a stack if the topmost piece in the stack is one of their pieces. When a stack lands on another stack, the two stacks merge; if the new stack contains more than five pieces, then pieces are removed from the bottom to bring it down to five. If a player's own piece is removed, they are kept and may be placed on the board later in lieu of moving a stack. If an opponent's piece is removed, it is captured. The last player who is able to move a stack wins.



Here is a breakdown of a few methods used in the assignment:

Pseudo Program Flow

START:

Display game logo <- input_output
Initialize players <- game_init
Initialize board <- game_init

While(game is on going)
    Determine which players turn
    Display player options <- input_output
    Make a move <- GameLogic
    Check if there is a winner <- GameLogic
        If there is, game is done
        Else repeat while
display_gameover <- input_output

END:

Pseudo initialize_players (player players[PLAYERS_NUM])

DESC: *"This function handles the of the players"*

LOOP through the players; i
Create a string, p_name for the players name
Prompt the user to enter their name
Assign the player at index i member name to the value p_name
Assign the player at index i the color of i (0 – RED, 1- BLUE)
Display the players information to user

Pseudo GameLogic.MakeMove(…,bool fromReserves,.)

DESC: *"This function handles the logic to perform a move from a user per turn Prompt player to enter a square"*

IF player is using fromReserves

    Get valid coordinates <- **GetValidMove()**

    Preform Move <-**pushStack()**

    Handle stacks of greater than 5<-**fallenPieces()**

    RETURN

ELSE

Get valid coordinates <- **GetValidMove()**

Check stack for amount of actions needed to be taken

Get steps from player <- **GetSteps()**

Compute the destination from steps <- **GetValidDestination()**

Preform the move <-**pushStack()**

Handle stacks of greater than 5 <-**fallenPieces()**

Pseudo pushStack(s1,s2)

DESC: *"This function stacks top2 on top of stack top1"*

Check if |s1| = |s2| return s1

Create a pointer newPtr = the address of s1

Create a pointer cur to point to the starting address of newPtr

Loop till the end of newPtr ('s1')

Set the next member of newPtr = the address of the first node of s2

Set the address of s2 = the address of cur

Set the square 1 to empty

Return s2

Pseudo fallenPieces(square s1)

DESC: "*This function takes in a square and removes pieces from the bottom of the stack exceeding the STACK_LIMIT, then the new stack of fallen pieces are counted and allocated towards the players own_pieces and adversary members.*"

Check if |s1| <= STACK_LIMIT return s1
Create a pointer temp_Stack = to the address of s1
Create a pointer new_Stack = to the address of s1
Create a counter for number of pieces on s1
Loop thru tempStack and while count< STACK_LIMIT; increase counter
Loop till the end of new_stack
Set the new_stack next member to null, Thus completing s1
If temp_stack(the excess of pieces after count) is not null
Create a counter for my pieces and theirs
Loop through temp_stack and add up colors that are the same

Pseudo can_player_move(board)

DESC: "*This function will check if there is a player that has no pieces on the board and if so, return its player id else return -1 if all players have at least 1 piece on the table to play*"

Create an array for the player frequency
Loop through the board
    If the square is valid and the stack is not null
    If color is RED increase player 1 counter in array
    If color is BLUE increase player 2 counter in array
Create Boolean to check if the player can move
Loop through players; l
    playerCanMove = player_frequency is greater than 0
    if playerCanMove is False; return the index l
return -1 which means there is no player that cannot move