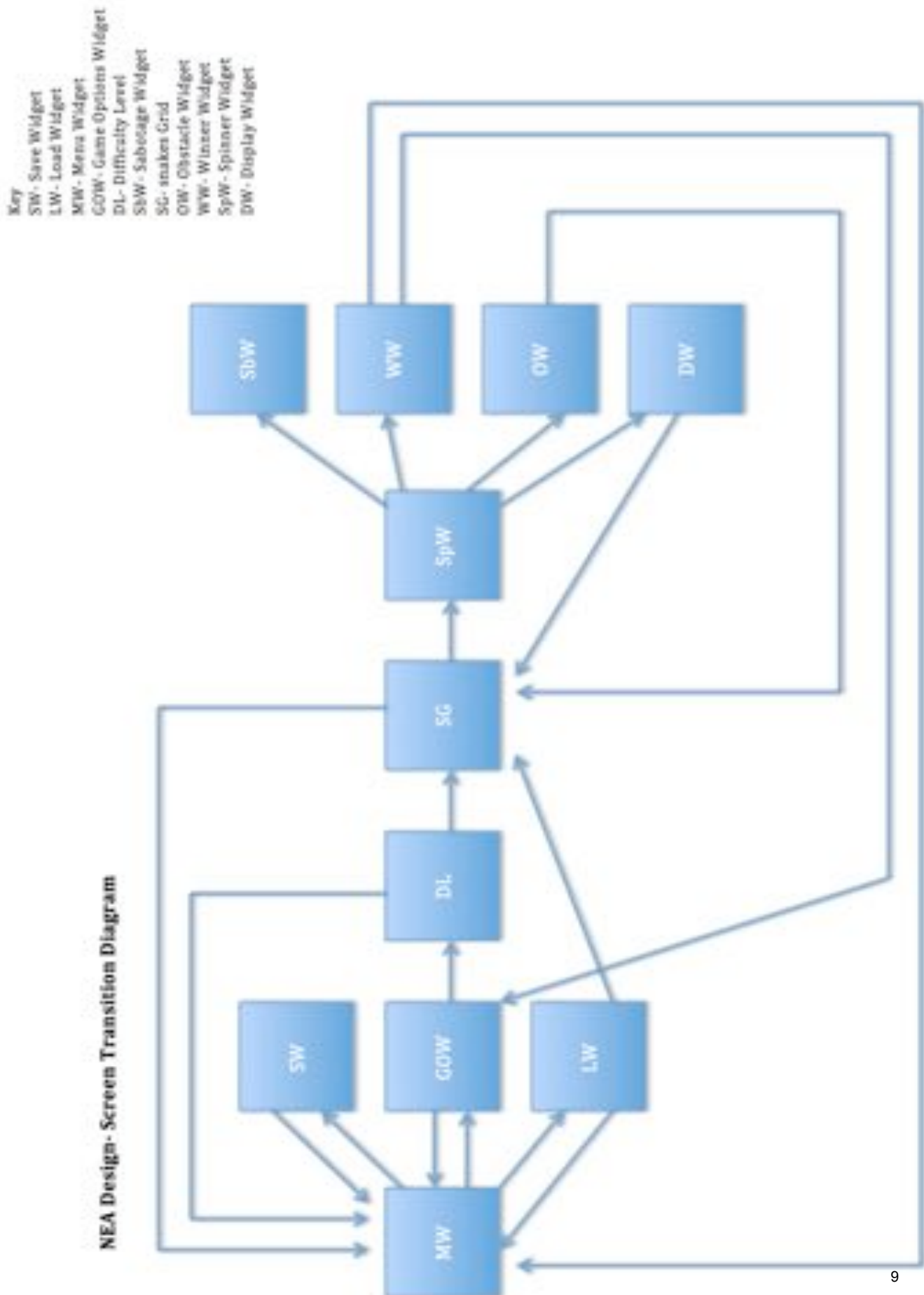


NEA Design- Screen Transition Diagram



NEA Design - Algorithms

```
CLASS snakes_and_ladders_grid:
    DEFINE create_grid_buttons:
        Make list_of_colours for button backgrounds
        IF difficulty level is easy:
            Set:
                rows = 7
                columns = 5
                sabotage = 3
                snakes = 1
                ladders = 3
        IF difficulty level is intermediate:
            Set:
                rows = 9
                columns = 7
                sabotage = 6
                snakes = 5
                ladders = 5
        IF difficulty level is hard:
            Set:
                rows = 11
                columns = 9
                sabotage = 8
                snakes = 7
                ladders = 5
        Number of buttons = Rows x Columns
        Maximum space = number of buttons - 2
        FOR number FROM 1 to maximum space:
            List of Possible Spaces + number

        IF game is being loaded:
            PASS
        ELSE:
            Call Create snakes positions subroutine
            Call Create ladders positions subroutine
            Call Create sabotage spaces positions subroutine
            Set player positions to 0
        FOR number FROM 1 to maximum space:
            Select colour from list of colours
            CREATE button with number for position
            IF position of button is in list of snakes positions:
                Change Background image of button to a snake
            ELSE IF position of button is in list of ladders positions:
                Change Background image of button to a ladder
            ELSE IF position of button is in list of sabotage spaces:
                Change Button text to SABOTAGE
            GRIDLAYOUT add current button
        CREATE button shortcut to main menu
```

```

Menu button when pressed go to menu screen
CREATE button shortcut to spinner screen
Spinner button when pressed go to spinner screen
GRIDLAYOUT add buttons: menu and spinner
IF game is being loaded:
    IF game is two player:
        Call function to position players 1 and 2
    ELSE:
        Call function to position player 1 and Computer

```

<<Example of creating obstacle: same can be used for snakes, ladders and sabotage spaces>>

(in snakes and ladders grid class)

```

DEFINE create obstacle (number of obstacles needed):
    FOR 1 to total number of obstacle:
        Space = randomly select from Possible Spaces
        WHILE space selected not in appropriate range:
            Space = randomly select from Possible Spaces
        Add space to list of existing Obstacles
        Remove space from list of Possible spaces

```

<<Positioning Player pieces>>

(in snakes and ladders grid class)

```

DEFINE player positions (current player):
    IF current player = 1: (do for players 1, 2 and the computer)
        Decide how many rows up the board the player piece is based on
        player position
        IF player position = 0: position player off board
        ELSE:
            Decide how many spaces across the player piece is based on
            player position
            Player position = width of window x (across/columns),
            height of window x (high/columns)
        set player piece position to calculated position

```

<<Moving a player piece>>

(in snakes and ladders grid class)

```

DEFINE move piece(current spin, current player):
    IF current player = 1: (repeat for 2 and computer)
        Player position + current spin
        IF player position >= maximum space:
            Current screen = Winner Screen (player = 1)
        IF game is in two player mode:
            IF both player positions > 0:
                IF player 1 position = player 2 position:
                    IF player 2 position >= maximum space:
                        Player 1 position = maximum space
                        Current = Winner screen (player 1)
                ELSE:

```

```

Player 1 position + 1
(ELSE IF game is in single player mode:
  IF both player and computer positions > 0:
    IF player position = computer position
      IF computer pos = maximum space:
        Player 1 position = maximum space
        Current = Winner screen (player 1)
      ELSE:
        Player 1 position + 1)<<only for player
1, do not repeat for player 2 and computer as game modes are a given>>
Current screen = display screen (current spin)
Check new position of player against obstacle positions
Position player piece: player positions (player 1)

```

<<checking for clashes between a player piece and an obstacle>>

(in snakes and ladders grid class)

DEFINE check for clashes (player):

```

if current player is player 1: (repeat for player 2 and computer)
  for snake in list of snake positions:
    if snake position is the same as player position:
      player position is moved down one row
      call display screen to tell user what they have
      landed on
    if player 2 position if the same as player 1 position:
      player 1 position increases by 1

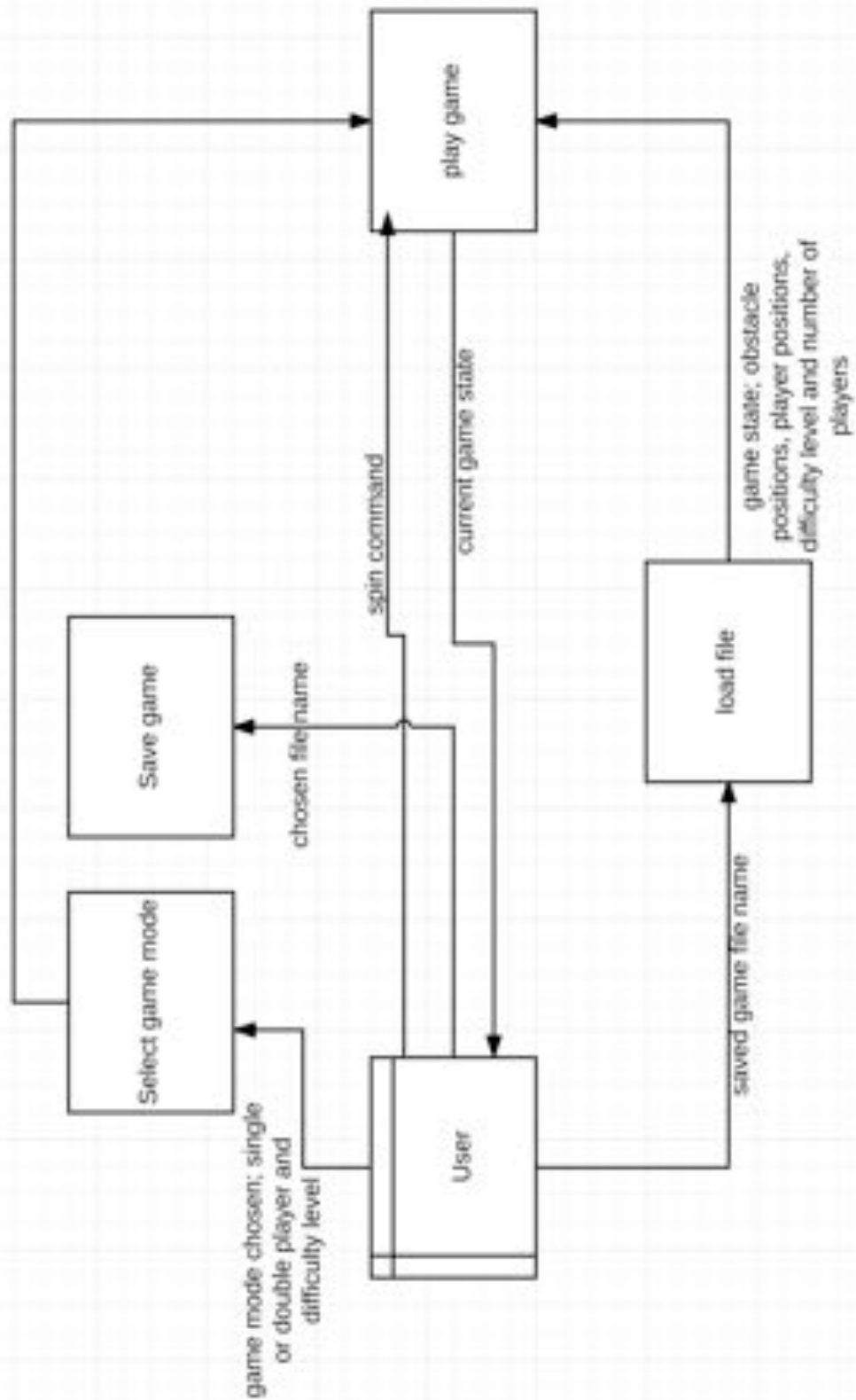
  for ladder in ladder positions:
    if ladder position is the same as player position:
      player position is moved up a row
      call display screen to tell user what they have
      landed on
    if player 2 position if the same as player 1 position:
      player 1 position increases by 1
    if player 1 position is the equal to or above the maximum
    number of spaces:
      call winner screen to tell player who has won

  for space in list of sabotage space positions:
    if player one position is the same as sabotage space
    position:
      display sabotage screen

  if player 1 position is the equal to or above the maximum number
  of spaces:
    call winner screen to tell player who has won

```

NEA Design – Data Flow Diagram



NEA Design- Data Structures

Index	Type of Data Structure	How it will be used	Where will it be used	Why I will use it
1	List (possible [])	To store all of the available spaces that an obstacle can be positioned on	Throughout the code wherever obstacles are created, but rooted in the snakesGrid class and referenced through that	It can help to ensure the obstacles don't clash in any way, and it is easier than using several if and while statements for each obstacle to ensure it doesn't clash. This way, I can just randomise the available spaces in whatever range is necessary for that obstacle, and once the obstacle position is set it can be removed from the list of available spaces ready for the next obstacle to be made
2	List (positions[], positions2 [], positions3 [], positions4 [])	To store the game data when a game is saved	In the MenuWidget class, in save_game()	This is the most efficient way I could find to store all of the game data. I could use several lists, for example, one for general data and then one for each category of obstacles. I could then use these lists to import the data into a text file for each list to be saved permanently
3	List (positions [])	To store loaded game data from a text file	In the MenuWidget class, in load_game()	In reverse of the SaveWidget lists, I could import the data from each text file to its own list, to then sort through and allocate to different variables to enable me to set up the game identically to how it was saved
4	List (snakes_pos [])	To store all the current snakes positions	In the snakesGrid class, referenced throughout the program, for example to compare player positions to snakes positions	When the snakes positions are created, I can simply import the positions straight into this list as they are set, then the list can be used whenever the snakes positions need to be referenced, such as when the board is created to see where the snake images need to be placed, when a game has to be saved including obstacle positions or when a player moves and the program has to check if they have clashed with an obstacle. It is easier to loop through a list to check rather than having several if statements for individual variables.
5	List	To store all the	In the snakesGrid	(same as snakes positions list)

	(ladders_pos [])	current ladders positions	class, referenced throughout the program	
6	List (sabotage[])	To store all the current sabotage space positions	In the snakesGrid class, referenced throughout the program	(same as snakes positions list)
7	List (colours [])	A list of lists of numbers to program the background colour of the buttons I create in a pattern.	In the SnakesGrid class in the function create_buttons.	

Hierarchy of objects

App Widget – this is the core class required and is the parent of all other class instances. The App has a run method that starts the program

Screen Manager – Contains all of the screens that can be called and handles the transitions between the screens.

Screens – There are 11 screens that can be displayed, each screen will have different widgets loaded onto them. The screen manager controls the currently displayed screen.

The 11 screens will be:

- Save Screen- This screen has the box the user enters a filename in to save a game that comes up when a player chooses to save a game
 - SW- SaveWidget
 - Grid Layout
 - Button to tell user to enter filename
 - Text Input box for the user to enter filename in
 - Clickable button to submit name
 - Clickable button to go back to menu screen (if filename entered already exists):
 - Button to tell user name is invalid
 - Button to tell user to retype filename
 - Clickable button to save over existing game
- Load Screen- This screen has the box the user enters a filename in to load an existing game
 - LW- LoadWidget
 - Grid Layout
 - Button to tell user to enter filename
 - Text Input box for the user to enter filename in
 - Clickable button to submit filename
 - Clickable button to go back to menu screen

- (if name entered doesn't exist):
 - Button to tell user the name entered is invalid
 - Button to tell user to retype name
- Menu Screen- This screen comes up first where the user can choose to play a new game, save a game or load a game. All screens should link back to this screen
 - MW- MenuWidget
 - Grid Layout
 - Clickable button to play new game
 - Clickable button to save game
 - Clickable button to load game
 - Clickable button to quit window
- Game Options Screen i.e. Single Player, Two player
 - GOW- GameOptionsWidget
 - Grid Layout
 - Clickable button for Single Player
 - Clickable button for Two Player
 - Clickable button to go back to Menu Screen
 - Button to show which game piece belongs to who (x3)
- Difficulty Level Screen i.e. Easy, Intermediate, Hard
 - DL- DifficultyLevel
 - Grid Layout
 - Clickable button for easy
 - Clickable button for intermediate
 - Clickable button for hard
 - Clickable button to go back to menu screen
- Sabotage Screen- when a player lands on a sabotage space the screen displays their options and what the rolled
 - SbW- SabotageWidget
 - Grid Layout
 - Button to tell player what they have rolled
 - Button to tell player they landed on a sabotage space
 - Clickable button to move 5 spaces ahead
 - Clickable button to move opponent 5 spaces behind
- Snakes Grid Screen- the main game screen containing the game board
 - SG- snakesGrid
 - Grid Layout
 - Button displaying either a snake, ladder, sabotage or the number space (x the number of spaces based on difficulty level)
 - Clickable button to make spin by going to spinner screen
 - Clickable button to go to menu screen
- Obstacle Screen- when a player lands on a snake or ladder this screen comes up to tell the player what they landed on
 - OW- ObstacleWidget
 - Grid Layout

- Button to tell player what they have landed on
 - Clickable button to go back to main screen to continue playing
- Winner Screen- when a player wins the game this screen appears to tell the players who won and gives them their next options
 - WW- WinnerWidget
 - Grid Layout
 - Button to tell players who has won
 - Clickable button to play another game
 - Clickable button to go back to main menu
 - Clickable button to quit window
- Spinner Screen- This screen contains the clickable image of a spinner for the players to make their spin
 - SpW- SpinnerWidget
 - Grid Layout
 - Button to tell the current player to click on the spinner
 - Clickable button to make a spin
- Display Screen- If no obstacle is landed on, this screen appears to tell the player what they rolled
 - DW- DisplayWidget
 - Grid Layout
 - Button to tell user which player has spun what
 - Clickable button to go back to main screen and continue playing game

○ Widgets

Each screen contains a widget. This is a container that has a layout child object, which contains the items displayed upon it. The widget contains functions, which are used by the children

▪ Layouts

Each widget has a layout to store all of the buttons needed for that screen. The pre-set layout organises the positions of the buttons, which can be edited by me to fit them into a certain order.

• Buttons

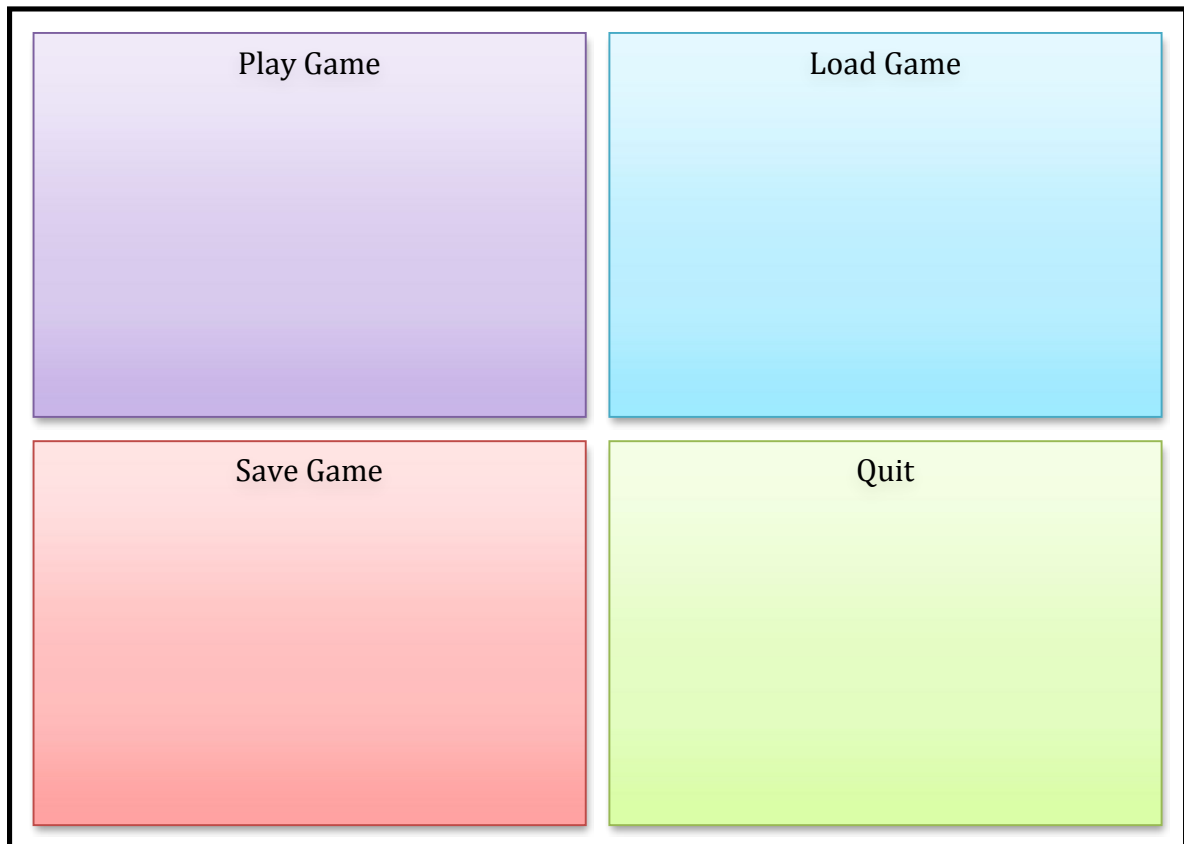
The screens have either clickable buttons, which will trigger a transition to a different screen, or non-clickable buttons (labels) that are used to give information to the user but will not do anything when clicked.

NEA Design- File Structures and Organisation

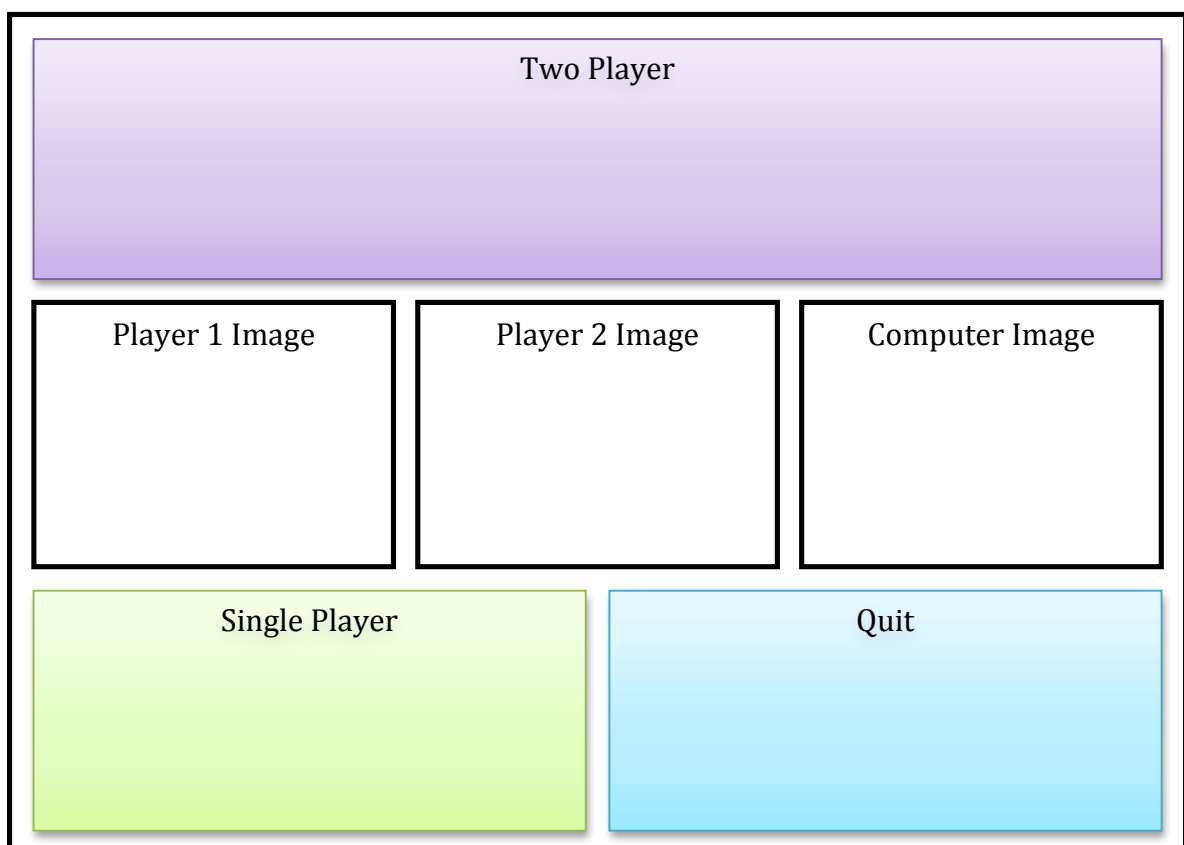
Index	File type	How it will be used	Where it will be used	Why I will use it
1	Text file (SavedGame"".txt)	To store some of the game data when a game is saved.	In the MenuWidget class, in save_game ()	<p>When a game is saved, the text file will be created using the file name entered by the user to contain all of the game data such as difficulty level, number of players, player positions and current player. The program when loading the game can then access this data to recreate the board accurately. Each piece of data is stored on a separate line by adding "\n" to the end of each line. This keeps the information separate, and the "\n" is removed when the data is reloaded to find the original data items.</p> <p>E.g. "10" = player 1 position 0 = player 2 position 2 = computer position 1 = number of players easy = difficulty level 1" = current player</p>
2	Text file (SavedS"".txt)	To store the positions of the snakes when a game is saved.	In the MenuWidget class, in save_game ()	<p>It will be easier to save the list of obstacles all separately to the rest of the data, as the number of each fluctuates depending on the difficulty level, which could prove complicated to predict when reloading game data. Each snake's location is saved on a separate line as before, using "/n", then the data can be extracted and looped through without concern about the amount of the obstacle.</p> <p>E.g. "23</p>

				12" Each line is a snake position.
3	Text file (SavedL"".txt)	To store the positions of the ladders when a game is saved.	In the MenuWidget class, in save_game ()	Same as snakes save file. E.g. "21 8 6" each line is a ladder position.
4	Text file (SavedB"".txt)	To store the positions of the sabotage spaces when a game is saved.	In the MenuWidget class, in save_game ()	Same as snakes save file. E.g. "4 15 5" Each line is a sabotage space position.

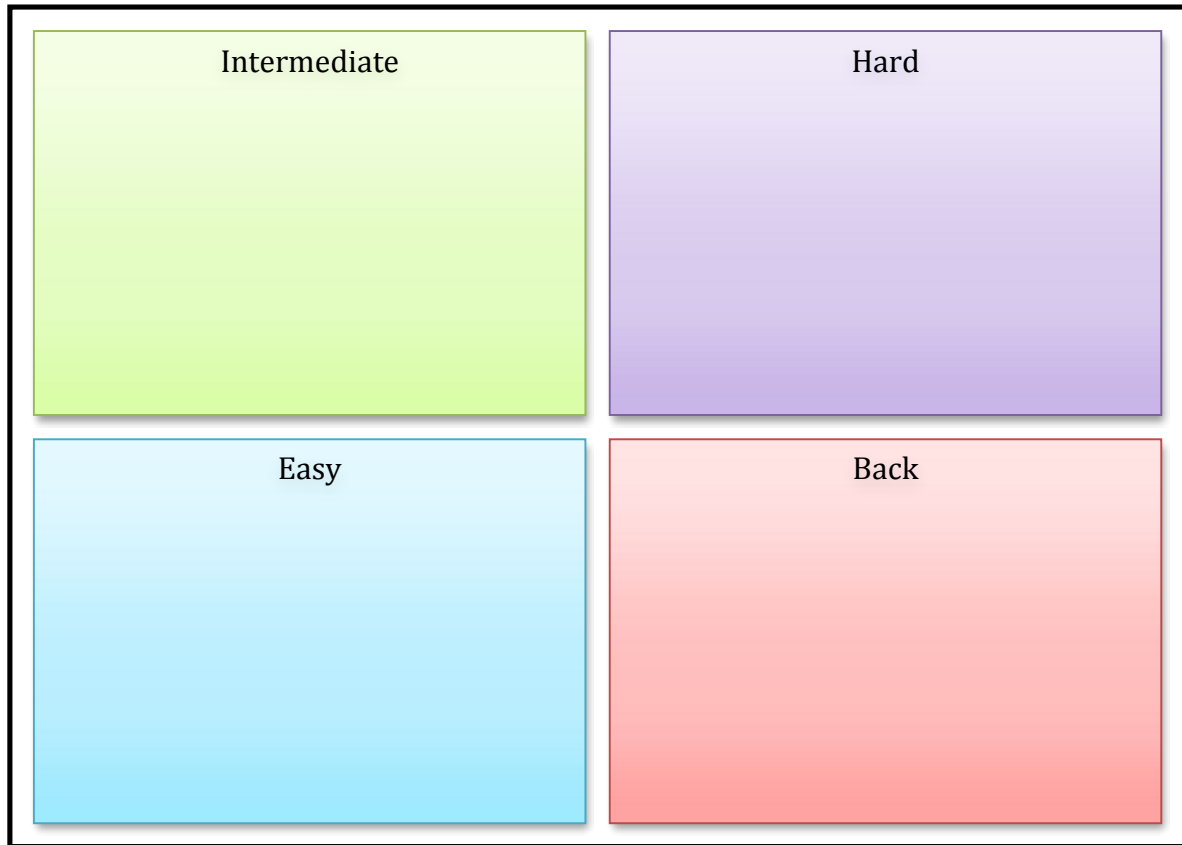
Menu Screen



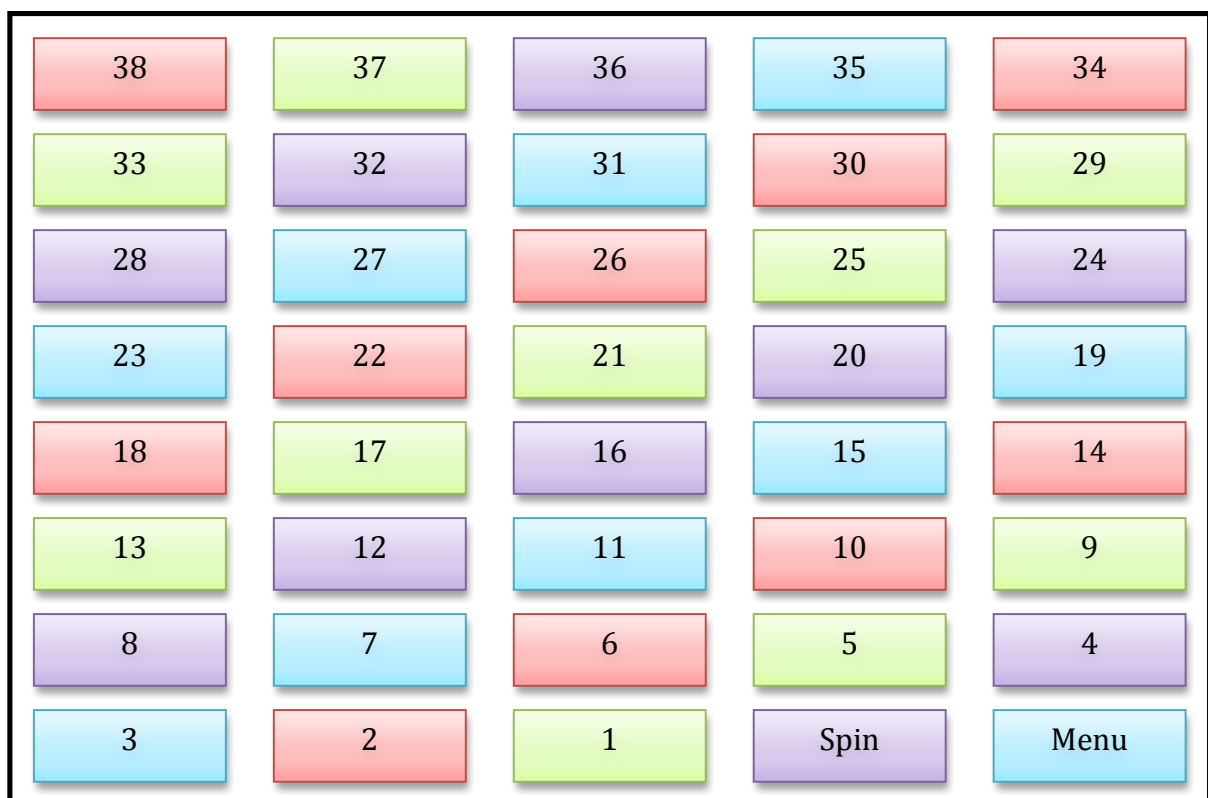
Game Options Screen



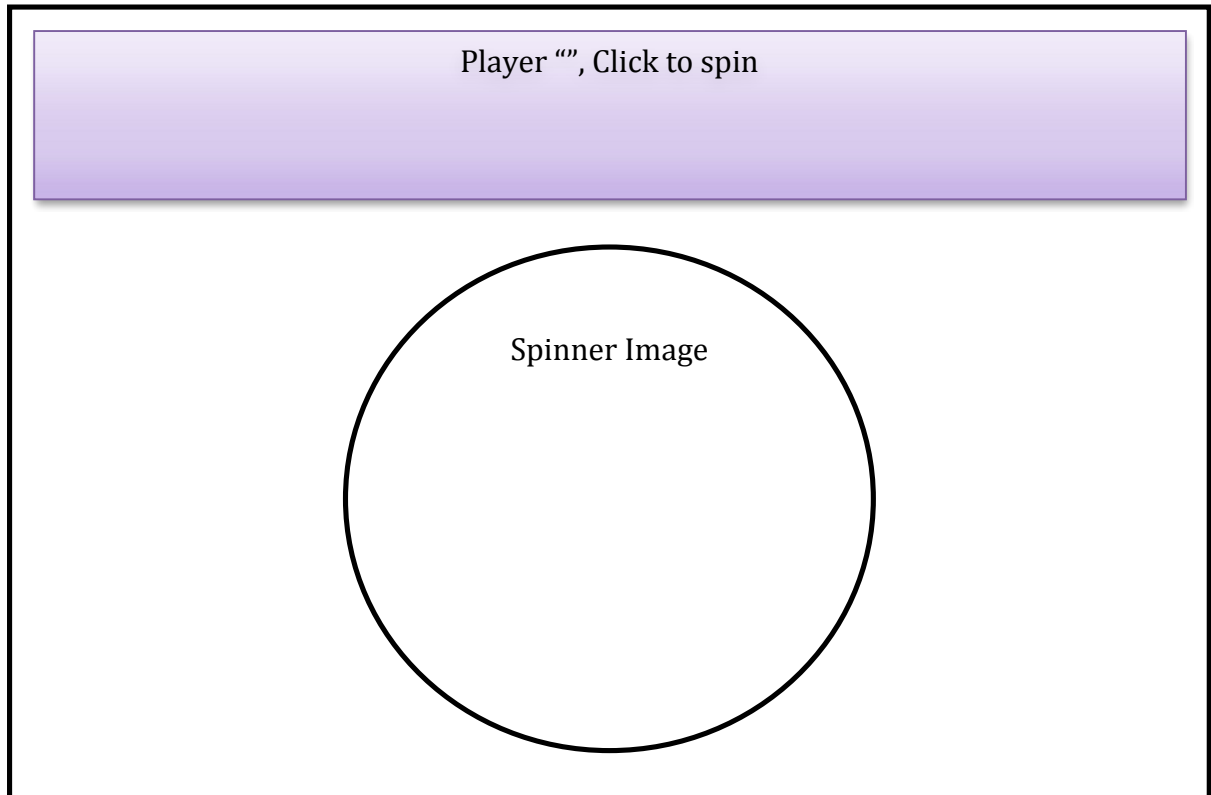
Difficulty Level Screen



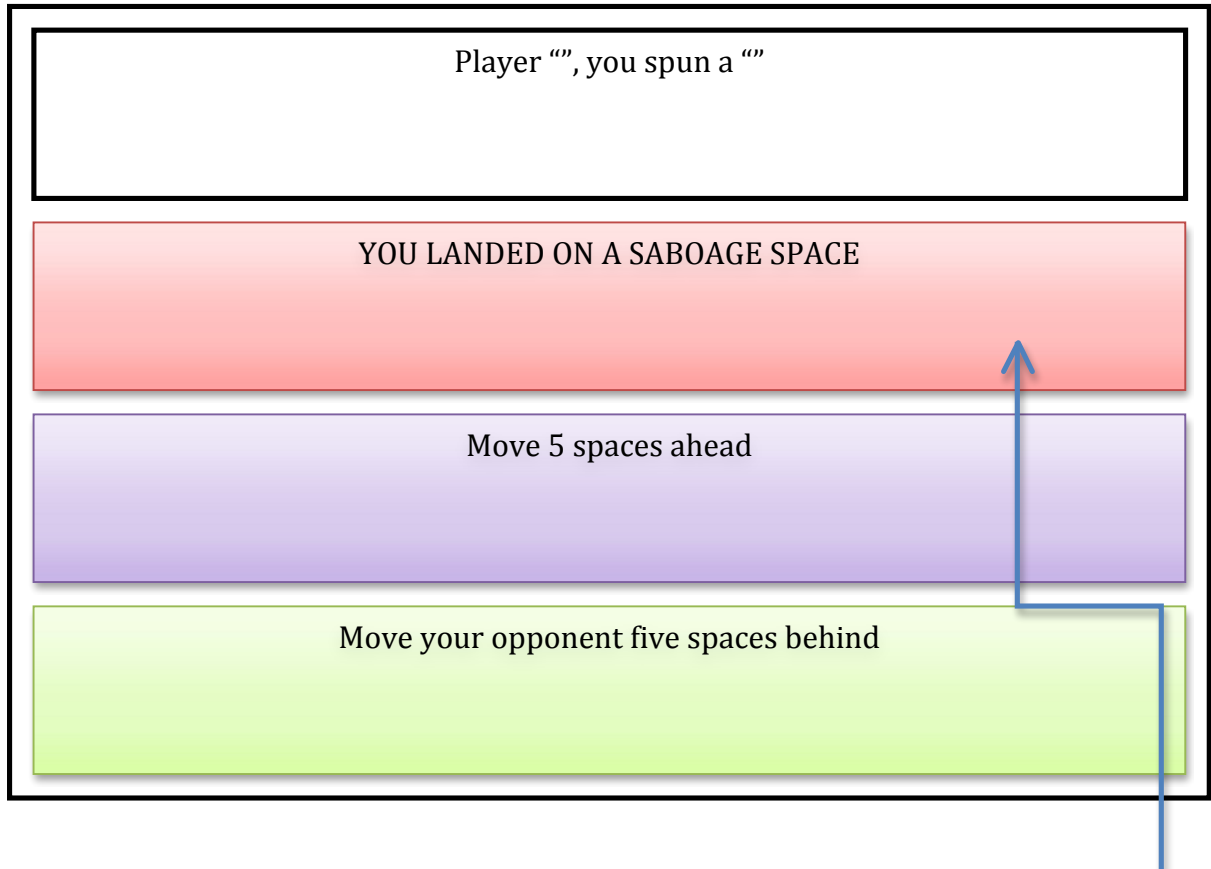
Main Game Screen example- would have varied number of spaces for different difficulties



Spinner Screen



Sabotage Screen



Gives player information before giving them options for gameplay

Display Screen

Player "", you spun a ""

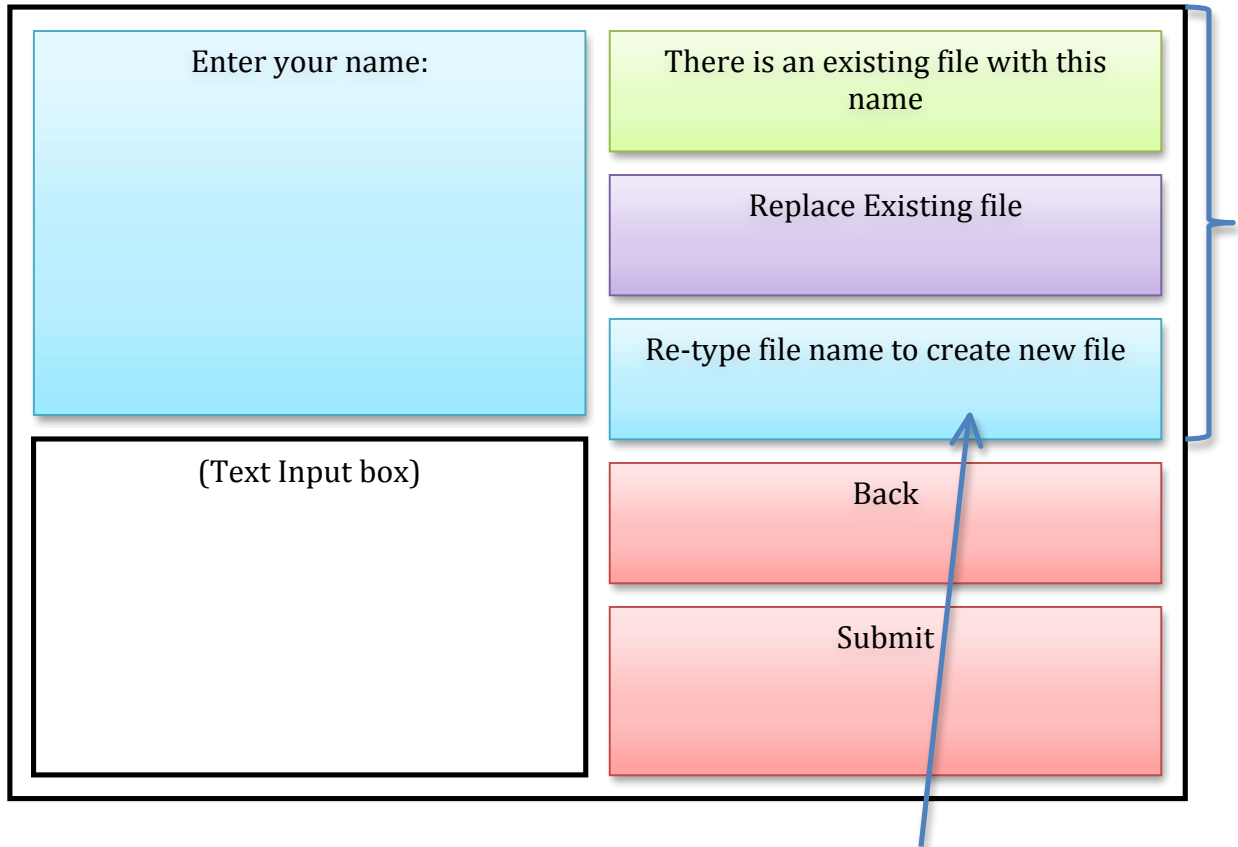
Back

Obstacle Display Screen

Player "", has landed on a ""

Back

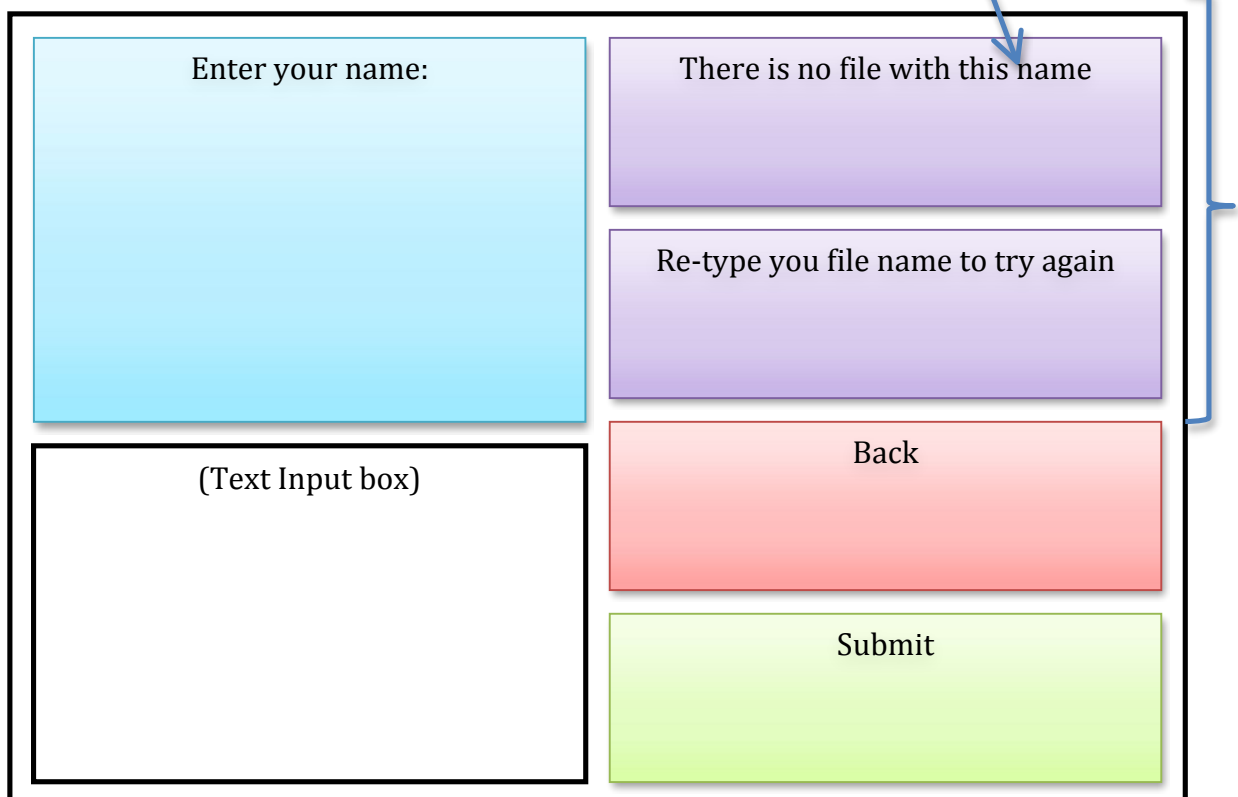
Save Screen



The Save Screen UI consists of a light blue box on the left with the text "Enter your name:" and a white box below it labeled "(Text Input box)". On the right, there is a vertical stack of four boxes: a light green box with "There is an existing file with this name", a light purple box with "Replace Existing file", a light blue box with "Re-type file name to create new file", and a light red box with "Back". Below the "Back" box is another light red box with "Submit". A blue arrow points from the "Submit" box to the "Re-type file name to create new file" box. A blue bracket on the right side groups the three boxes from "There is an existing file" down to "Re-type file name".

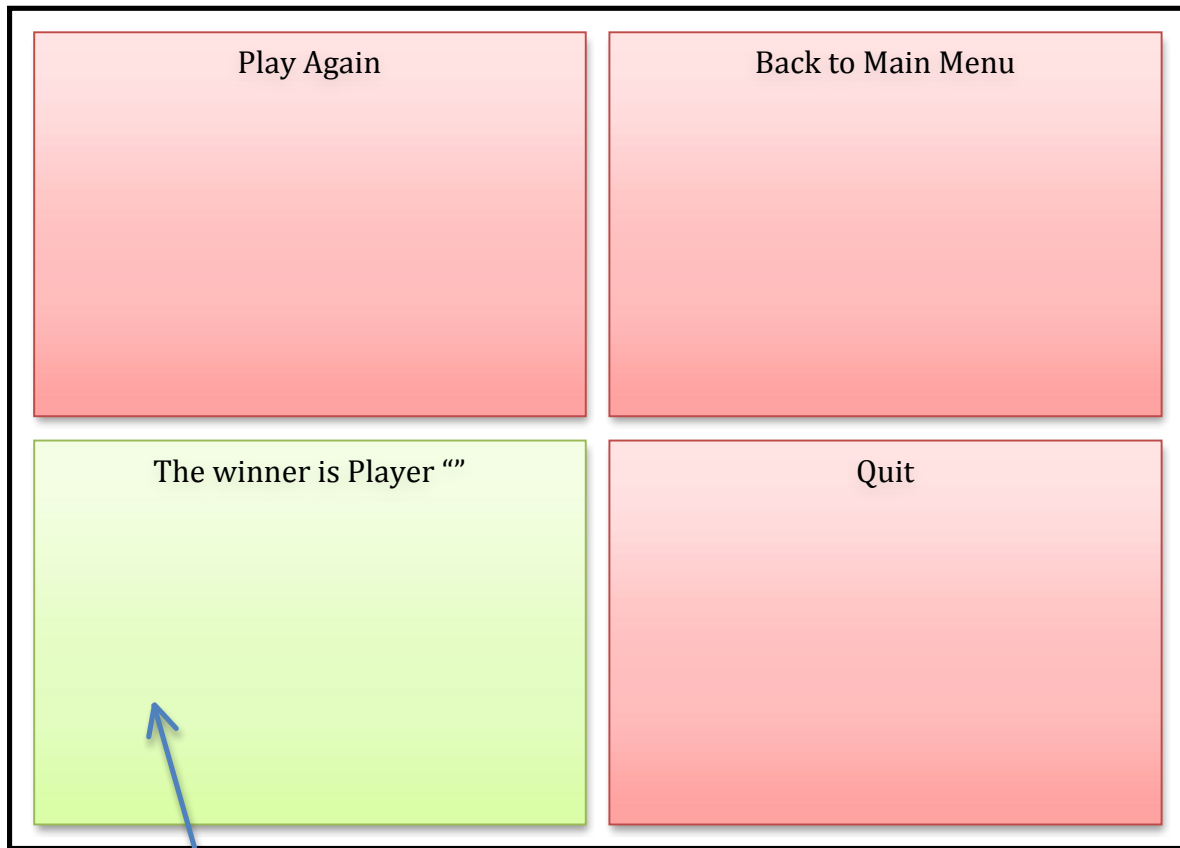
Error messages only appear when an invalid file name has been submitted.

Load Screen



The Load Screen UI consists of a light blue box on the left with the text "Enter your name:" and a white box below it labeled "(Text Input box)". On the right, there is a vertical stack of four boxes: a light purple box with "There is no file with this name", a light purple box with "Re-type you file name to try again", a light red box with "Back", and a light green box with "Submit". A blue arrow points from the "Submit" box of the Save Screen to the "There is no file with this name" box. A blue bracket on the right side groups the two purple boxes.

Winner Screen



This box is a different colour to highlight the message- the player has to read it because it will not be accessible once they click a button and leave this page.

I will use pastel theme colours, such as purple, blue, red and green/yellow as these suit both boys and girls and will appeal to younger audiences, which are my focus audience. I have specifically designed the buttons to be large, and should be clickable all over to make the game as simple as possible to use for young users. The instructions on buttons are clear and simple, for example when selecting a file name, the user is instructed to 'type all player's names', as logically the same two people, or one person playing single player, will not need more than one save each because they will complete their last game before starting a new one together. This has the advantage of making saving a game a lot easier for younger users, as they may not know what a 'file name' is, plus the first thing a child tends to be able to spell is their name. I will also use that standard font type for kivy (see example below of basic button I created as simulation) as it is rounded and simple for younger users to read. I also want to use a large font type to make the words as clear and visible as I can for the young user, around font size 40 or 50 within the application, depending on the scale of the button it is on.

Play Game

On the basic menu screens and in the game board screens, I will use a pattern of these theme colours to make it visually attractive. This mixture of colours will continue throughout all the screens, but some screens need to use the colour to bring attention to certain buttons. For example, on the winner screen, I will make the button with the information on about who has won a different colour to bring the user's attention to it, because once a button is clicked and this screen is gone the information is lost. Also on the load and save screens, I will use different combinations of the colours to make them identifiable because they look so similar. The mixed use of colour, I think, gives the game a wacky style, which again will appeal to my younger users.