Teacher Standardisation

Spring 2018

A-level Computer Science (7517)

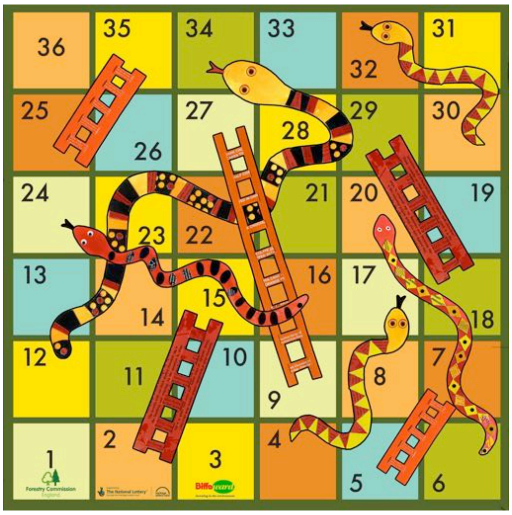
Booklet 1

Table of Contents

NEA Analysis

Introduction:

[1]



Snakes and Ladders is a traditional board game where two players take turns to roll dice and move their corresponding pieces up the board, aiming to reach the end space before their opponent. The snakes and ladders are positioned across the board and if the player lands on either the bottom of a ladder or the top of a snake, they are moved up or down the board respectively.

The game is popular and there are many variations of it already created, so it could be difficult to put an individual stamp on a new version of the game, or whether it would even be a good idea to make the game stray from its traditional format. However, these days, physical board games are played less and less, and people tend to look to their devices to play games, so I decided to create an application that could be played on computer or transferred to a device, so that people can play the board game wherever they are.

My target audience is children between about 5 and 10 years of age and families, and almost all children I know play on tablets and smart phones, so I think it is important that I create a game which is transferable to a hand-held device. This is an example of a traditional snakes and ladders board (see fig 1). It uses lots of bright colours and has numbered spaces to make it appropriate for children. These are some of the qualities that I want to incorporate in my adaptation, along with other features specifically aimed at young users.

For example, the game needs to be simple to control, so I will most likely try to have clickable buttons as controls rather than more complicated text commands. This will affect which application I choose to use when designing my app. The graphics will also need to be clear, bright and inviting, possibly incorporating cartoons to invite the attention of the users. There will also need to be clear labelling for each button because very young users may need lots of indicators as well as graphical suggestions to understand how to use the app.

How I researched:

While conducting my research, I downloaded several phone applications and looked at online versions of snakes and ladders and other, similar board games. I tried to look at the more popular game adaptations, as I want to look at the most successful methods and interpretations of traditional games. I wanted to analyse what was a better technique; keeping the game traditional to make it

[1] http://www.prateeknarang.com/Winning-a-Snakes-and-Ladders-Game/

recognisable and comfortable for the user, or utilise the fact that I am creating a digital version of the game by developing the game in a new way.

There was a range of sources that I looked at, but I decided on the current system to base my solution from because it was in a similar format to what I think I will make; with separate sections for each traditional component of the game, even where it may not be necessary on a digital device, to keep it realistic, and it was the one of the most popular hits when I searched for 'snakes and ladders game'.

Description of current system:

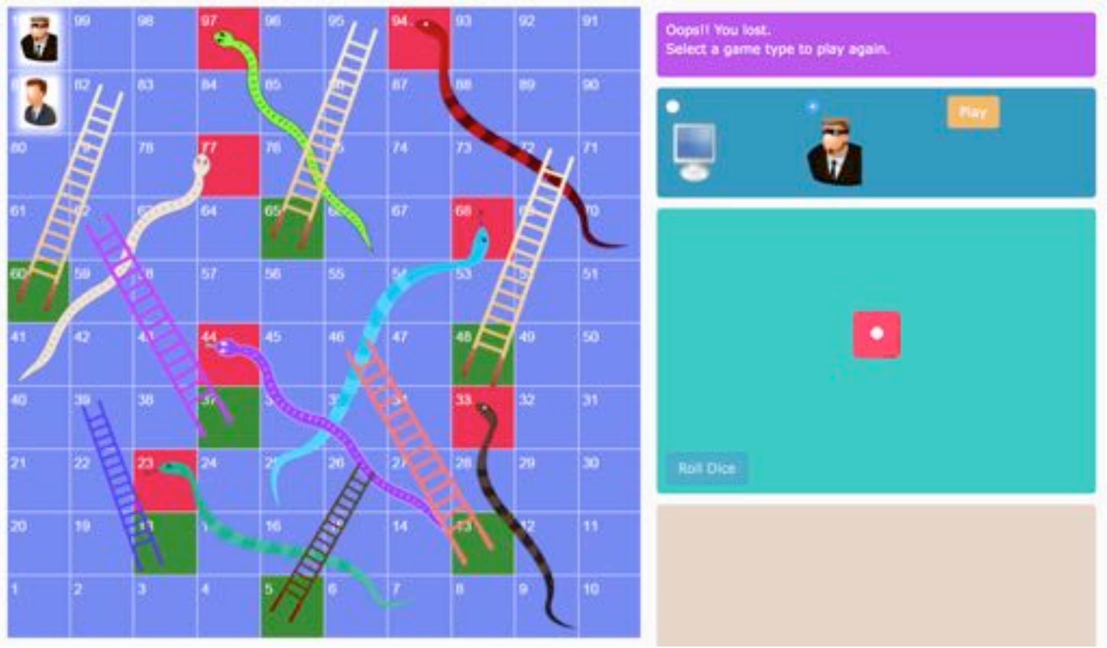
One of the most popular snakes and ladders games online was on a website called 'playonlinedicegames.com'. I decided to create my new system using this as a current system, while also considering true-life snakes and ladders to try and keep the game somewhat traditional.

The current system has a menu screen, dice screen and game screen all on the same page, so all are visible all of the time. It doesn't give the option to play with two players across separate devices, but you can choose to either let the computer play the other side automatically, or you click the dice for both sides both players. The game is somewhat ambiguous, however, because the two options of game play are either a computer symbol (where the other player's turns are completed automatically) or a person (where you, the user, has to click for both players). Even on the 'two player' mode, however, the game only references "you" and "the opponent" instead of player 1 and player 2, so it does not seem like an actual two player mode. This is a negative for gameplay for the user because the only option for two-player mode is confusing, especially for younger users who are likely trying to play the game with their friends.

There is a button in the dice section, and a message board above it to say whose roll it is, which helps to make following the game easier for the user to follow. Both players can land on the same space, but the icons block each other out which is confusing when playing. This is another negative from the user's perspective as it can be easy to lose track of which piece belongs to whom and where the individual pieces are. There is also no notification when a player is sent wither up a ladder or down a snake, so you have to watch the board very carefully to keep track of what is happening. Younger users especially are not likely to be paying this kind of close attention, so they could easily fail to trace the game properly, which could seriously impact gameplay quality. When a player wins, a banner appears in above the dice and menu sections to say whether you have won or lost, which is also a useful indicator for the user.

The game is very standard, and has no real options of game play like difficulty or any extra features, so I feel like they have not taken full advantage of the possibilities of creating the traditional game digitally. This means that the user could become bored with the game very quickly, especially modern, young users who generally require more amusement than a standard board game can provide. I want to improve on this dullness for my own version of the game.

[2]



Oops!! You lost.
Select a game type to play again.

Description of My System:

Based on the current system, I have decided to create a system which is traditional enough so that people can easily recognise the game, as this seems popular when conducting my research, but I want to advance the system to make it more interesting, as I feel like a lot more can be accomplished by creating a digital solution.

I want the user to be able to play single player, or have the option to play two player on the same device; I found this is many apps in my research, the game references the players as separate, but on the same device, so the phone/tablet can simply be passed between players as they have their turns. This feels more like the traditional game to me; several people playing on the same board. I want to add some extra form of gameplay, however, where the players can affect each other's positions, for example, to add some originality to the game where I can.

I want to have separate screens or windows for menus and dice so that the game is dynamic and doesn't always stay on the same screen, as this can get boring to look at. I will try to make improvements of things in the current system, which are user-unfriendly. For example, I won't allow the players to land on the same space, and I will make sure to put up a notification on the screen somewhere to tell the players if someone lands on a snake or ladder, to make the game easier to follow.

I have decided to use kivy and python to create my system, as I feel more comfortable with the parent-child relationships in kivy and how it produces python objects than I do with any other graphically capable language. I want to use something which can produce a graphical window so that the game can be

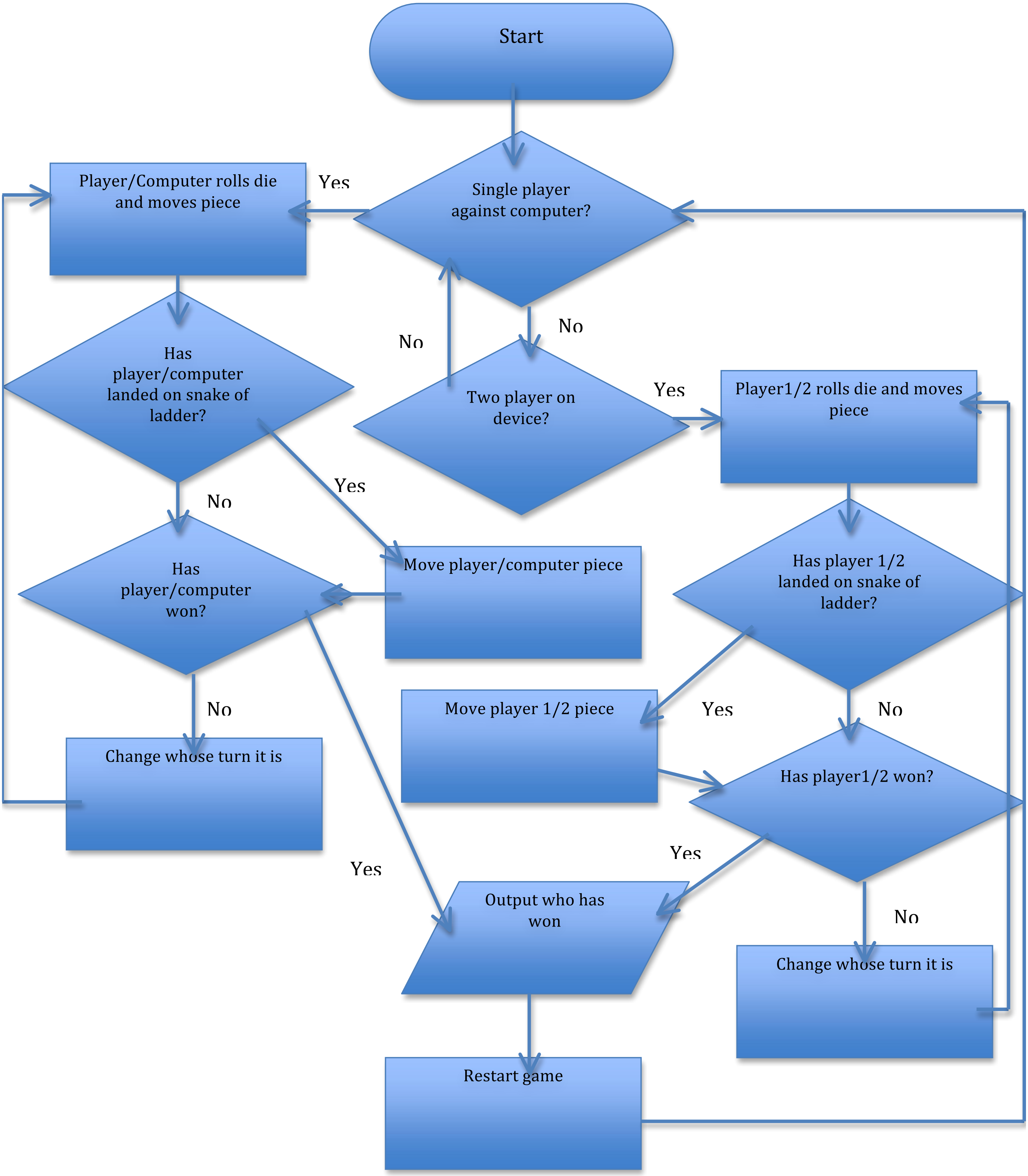
[2] http://www.playonlinedicegames.com/snakesandladders

played on a device, and has buttons which can be clicked and a board which is colourful, rather than a python-based text-based game, which can be hard to follow and look and feel messy when playing as the user has to type to signal any kind of action.

My Aims **and Objectives:**

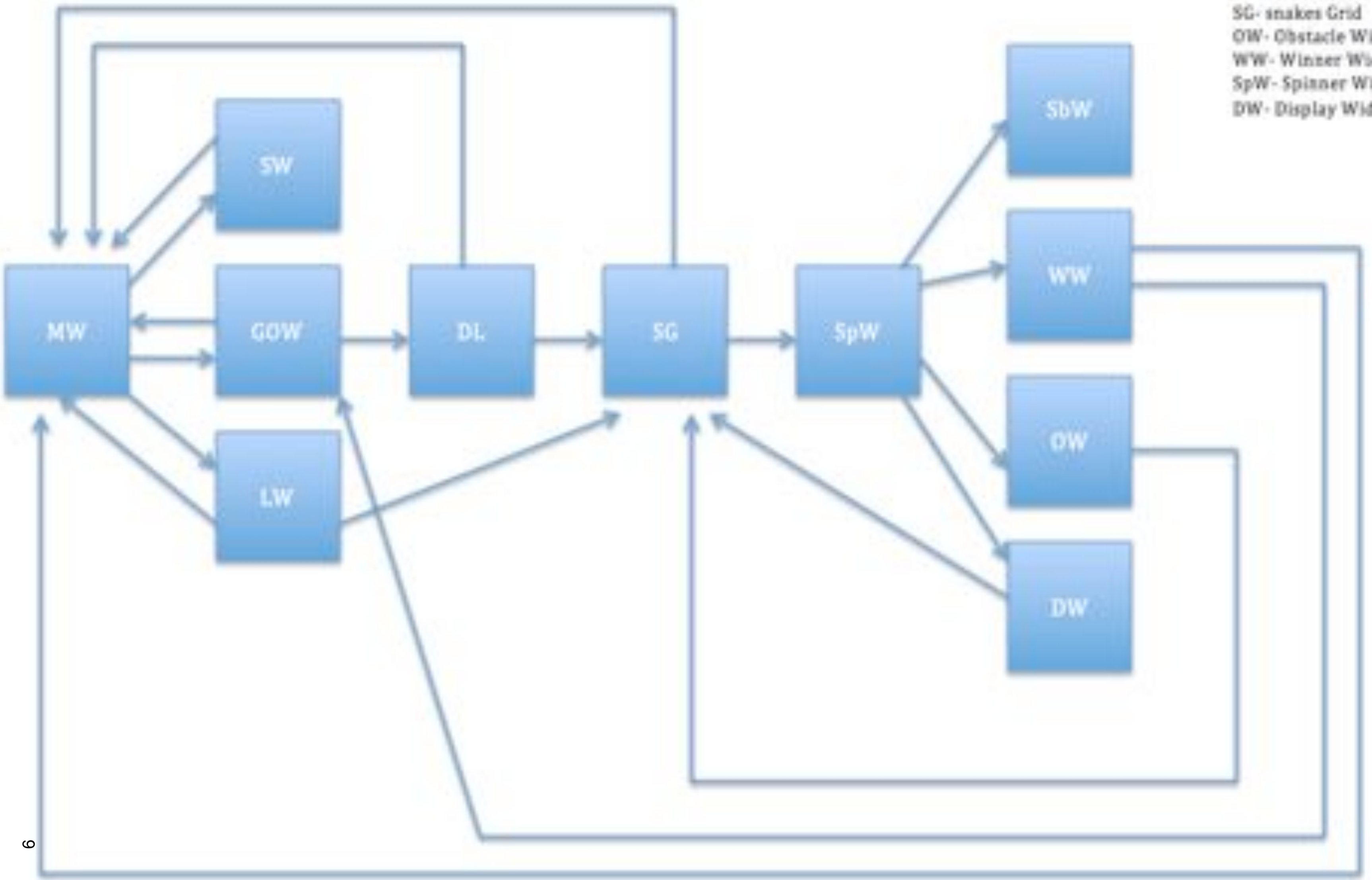
- **Make a menu screen**
  - Should be able to grant access to different areas of the game such as load game, save game or play game
  - Should be able to quit the game from here
- **Create a board with a grid of game spaces which a piece can move up**
  - The grid should be colourful and be numbered to make following the gameplay easier
  - Any obstacles should be randomly generated
  - Should have an end space which, if reached, determines a winner based on which player landed on it
- **Make Snakes**
  - Can alter a player's position backwards on the board
  - Has a graphical or clear representation on the board space
- **Make Ladders**
  - Can alter a player's position backwards on the board
  - Has a graphical or clear representation on the board space
- **Make dice/spinner to give the illusion of a traditional physical board game**
  - Should have a graphical screen where the user can click something to produce a dice roll/spin.
  - Should then call something in main program to randomly generate a number for a dice roll/spin.
- **Create Players**
  - Should have different, identifiable pieces for player one, player two and the computer
  - Should be able to move across the board and land in any specific space
- **Make single player mode so that the user can play against the computer**
  - The computer should roll automatically, but the player should be told what they have rolled so that they can easily trace the game
  - There should be a menu created so that the user can choose whether to play in single or two player mode.
- **Be able to save games**
  - Should be able to save several games with unique file names, which can be inputted by the user to load a specific game.
  - Player positions, game format (e.g. how many players there are), and obstacle positions should all be saved and be able to be reloaded in exactly the same way.
- **Be able to load games**

    - Player positions, game format (e.g. how many players there are), and obstacle positions should be able to be reloaded in exactly the same way as they were saved.
    - The game should continue as if it was never paused; for example, the game should remember who had the last turn and start from the next person (if in two player mode).
- **Create additional gameplay features**
    - To add my own twist to the game, I want to add new features such as being able to play the game on different difficulty levels. As the game gets harder:
        - The amount of obstacles on the board should increase; the proportion of snakes to ladders should favour snakes as the difficulty gets harder.
        - The size of the board should increase to make the game longer
    - I also want to add extra gameplay features such as spaces where the players can either improve their own position or sabotage their opponent if they land on it. I feel like this would increase the competitiveness of the game and make it more exciting.


Start
Single player against computer?
Yes
Player/Computer rolls die and moves piece
No
Two player on device?
No
Yes
Player1/2 rolls die and moves piece
Has player/computer landed on snake of ladder?
Yes
No
Move player/computer piece
Has player/computer won?
No
Change whose turn it is
Yes
Has player 1/2 landed on snake of ladder?
Yes
No
Move player 1/2 piece
Has player1/2 won?
Yes
No
Output who has won
Change whose turn it is
Restart game

NEA Design- Screen Transition Diagram

NEA Design- Screen Transition Diagram



Key
SW- Save Widget
LW- Load Widget
MW- Menu Widget
GOW- Game Options Widget
DL- Difficulty Level
SbW- Sabotage Widget
SG- snakes Grid
OW- Obstacle Widget
WW- Winner Widget
SpW- Spinner Widget
DW- Display Widget

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>>

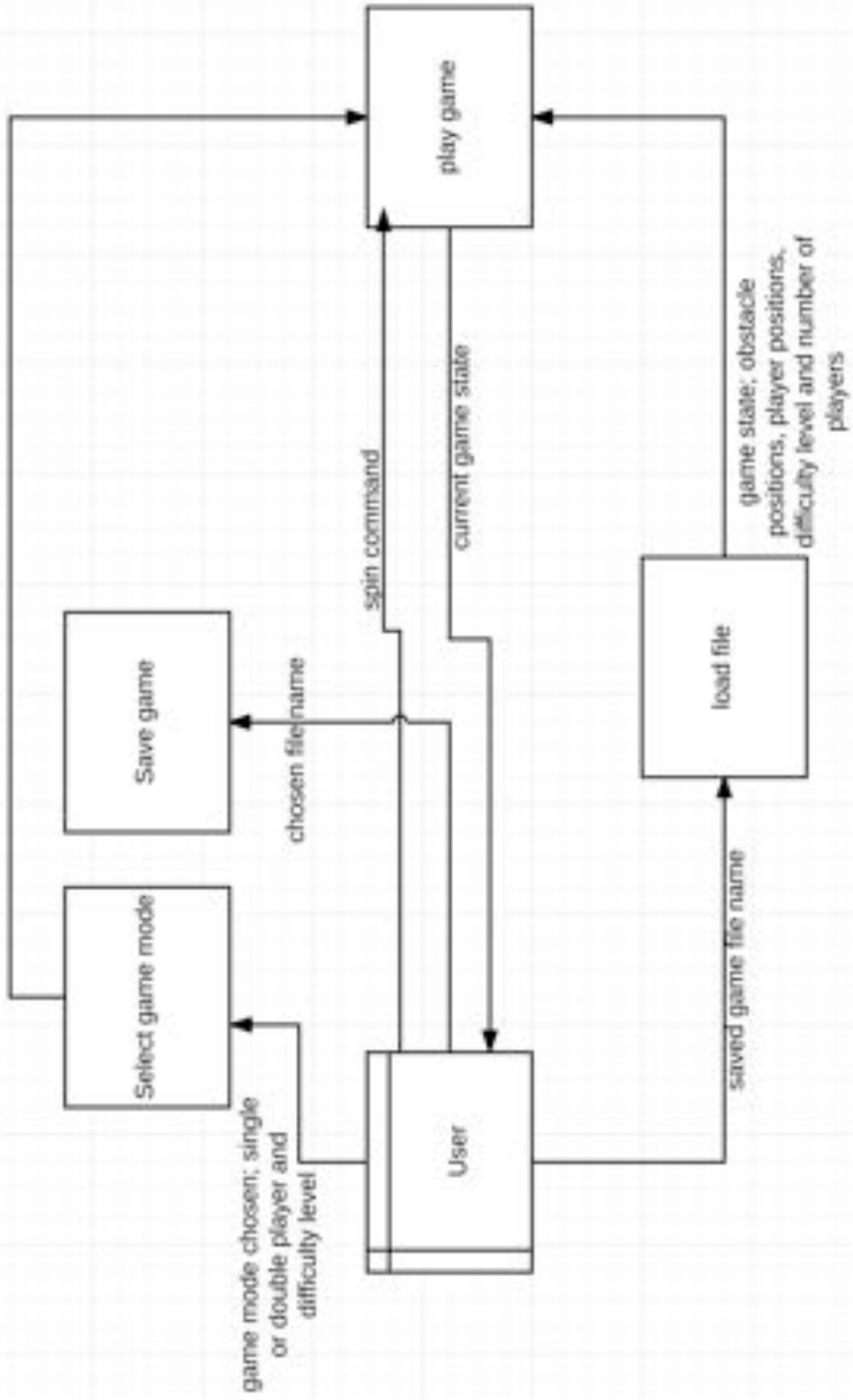
```
(in snakes and ladders grid class)
DEFINE move piece(current spin, current player):
        IF current player = 1: (repeat for 2 and computer)
                Player position + current spin
                IF player position >= maximum space:
                        Current screen = Winner Screen (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


play game
Save game
Select game mode
User
load file
spin command
current game state
chosen file name
game mode chosen; single or double player and difficulty level
saved game file name
game state; obstacle positions, player positions, difficulty level and number of players

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.

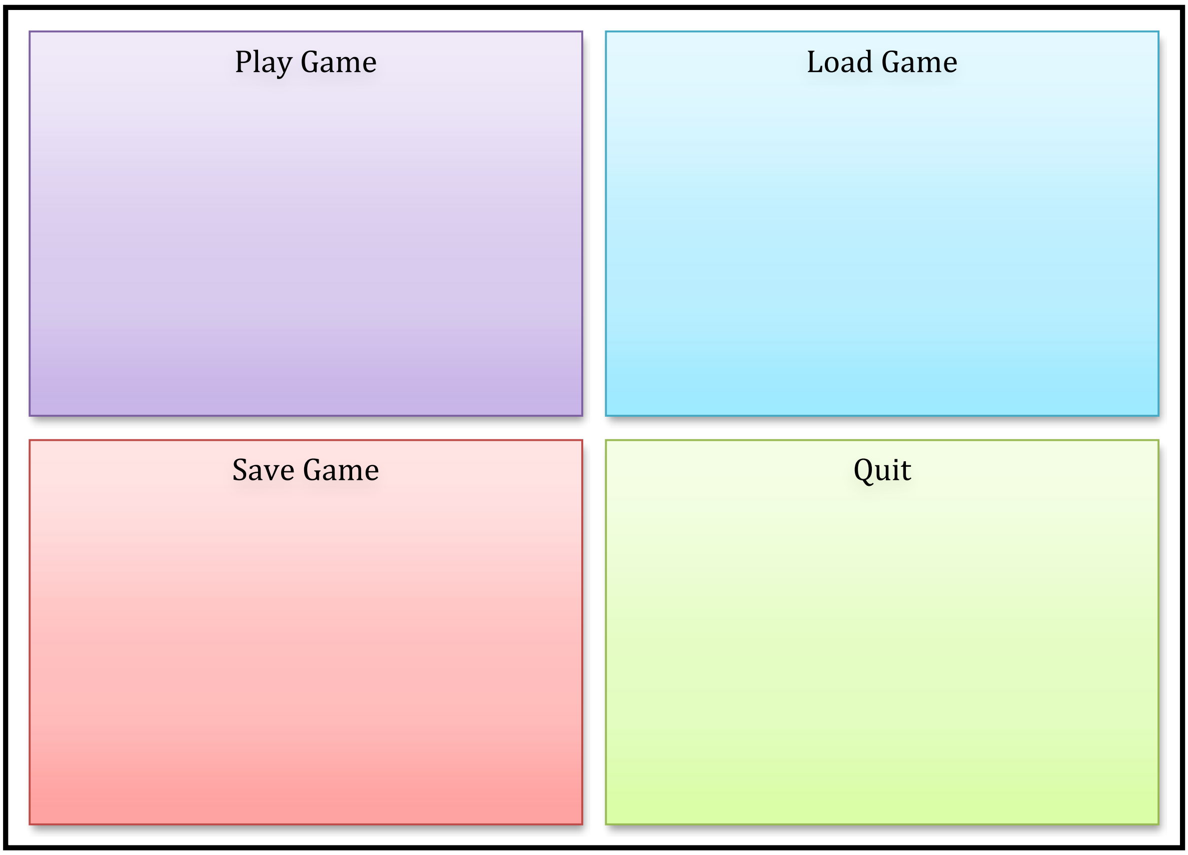
NEA Design- File Structures and Organisation

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 ()	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. E.g. "10 = player 1 position 0 = player 2 position 2 = computer position 1 = number of players easy = difficulty level 1" = current player
2	Text file (SavedS"".txt)	To store the positions of the snakes when a game is saved.	In the MenuWidget class, in save_game ()	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. E.g. "23

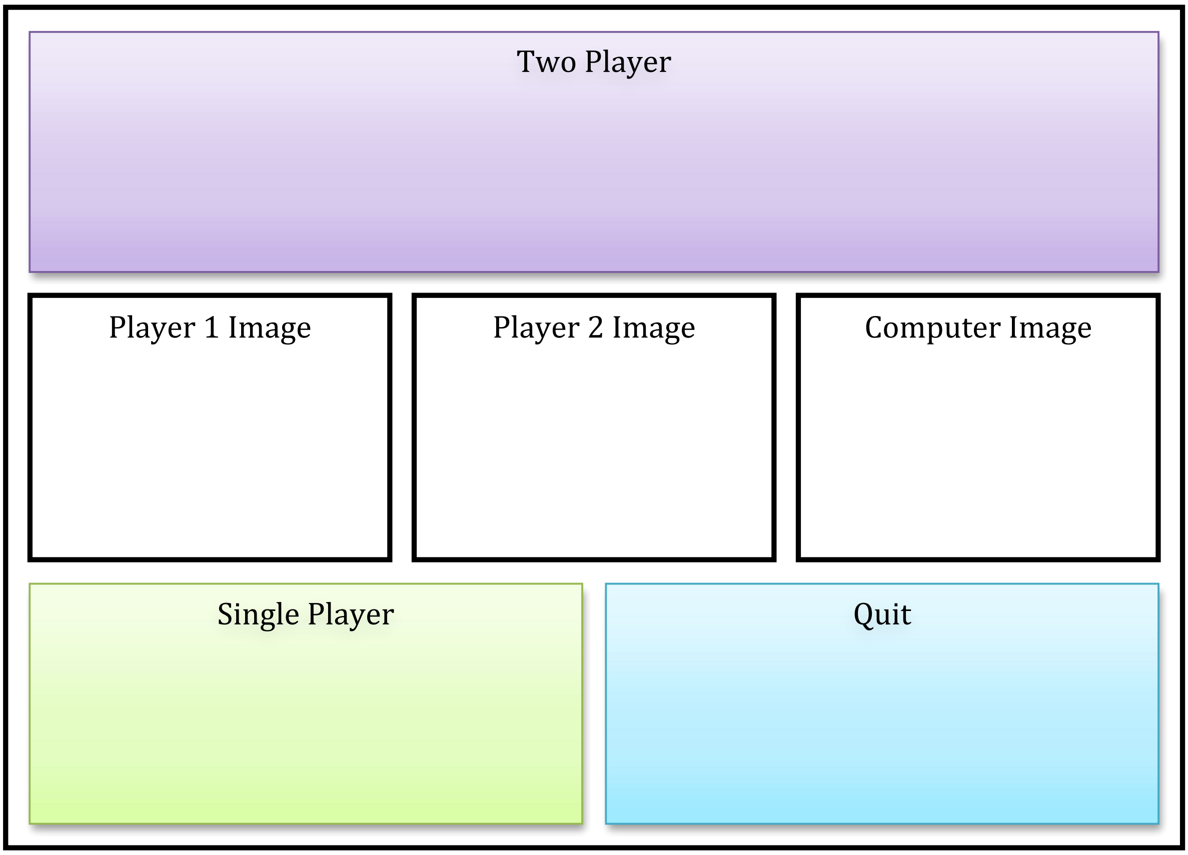
				12” Each line is a snake position.
3	Text file (SavedL””.txt)	To store the positions of the ladders when a game is saved.	In the MenuWidget class, in save_game ()	Same as snakes save file. E.g. “21 8 6” each line is a ladder position.
4	Text file (SavedB””.txt)	To store the positions of the sabotage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

NEA Design- HCI

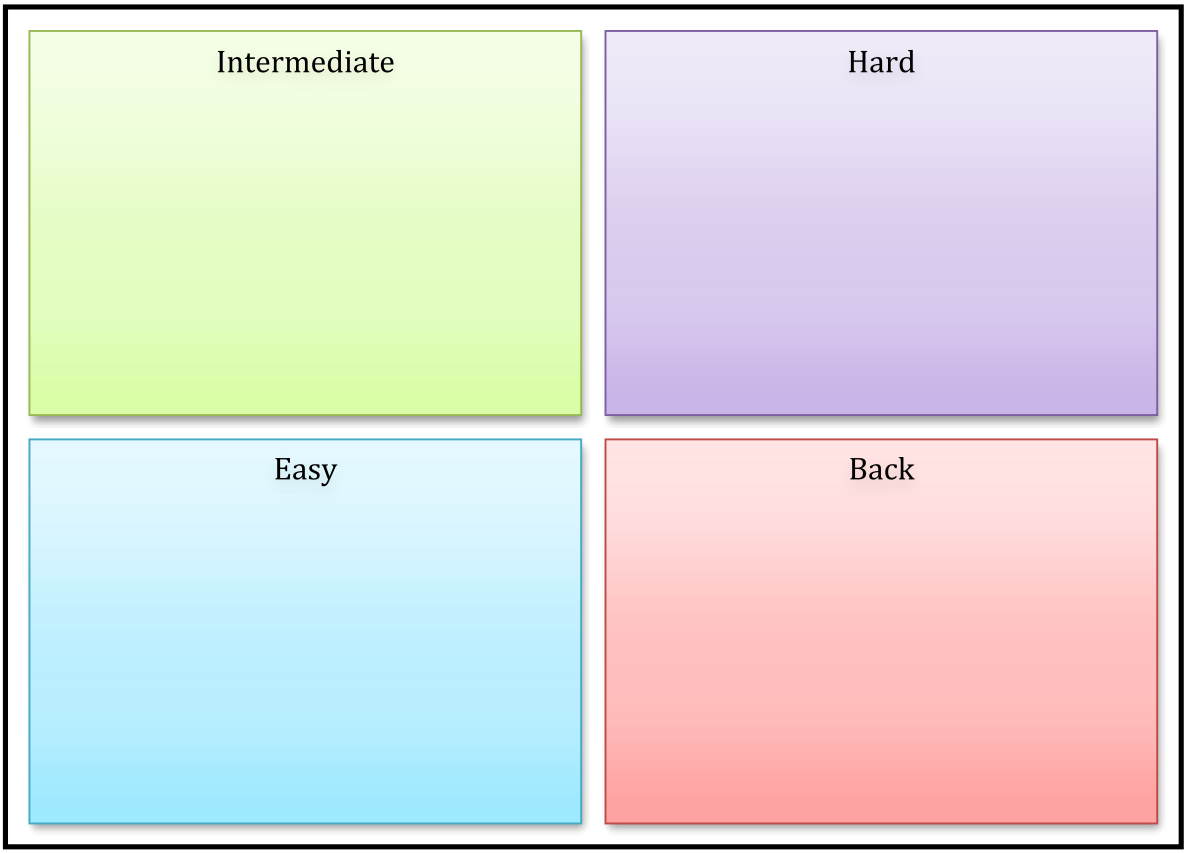
Menu Screen



Game Options Screen



Difficulty Level Screen



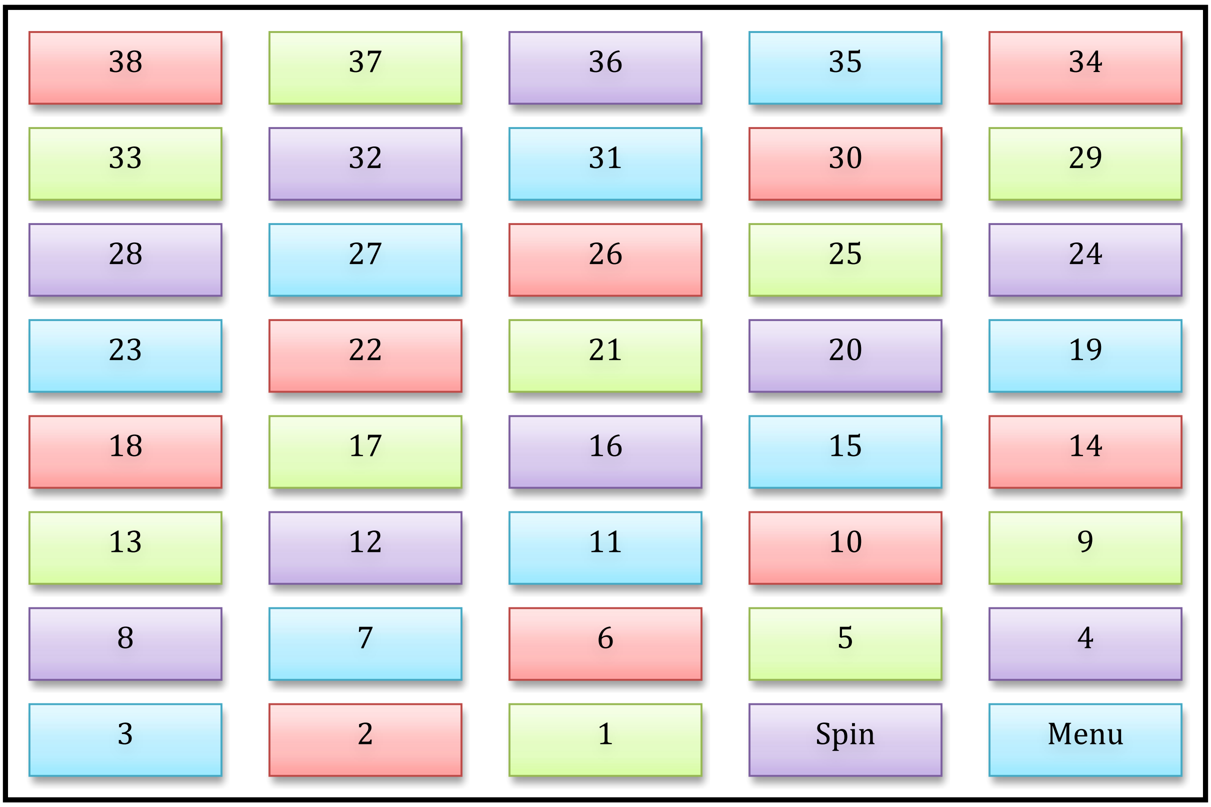
Intermediate

Hard

Easy

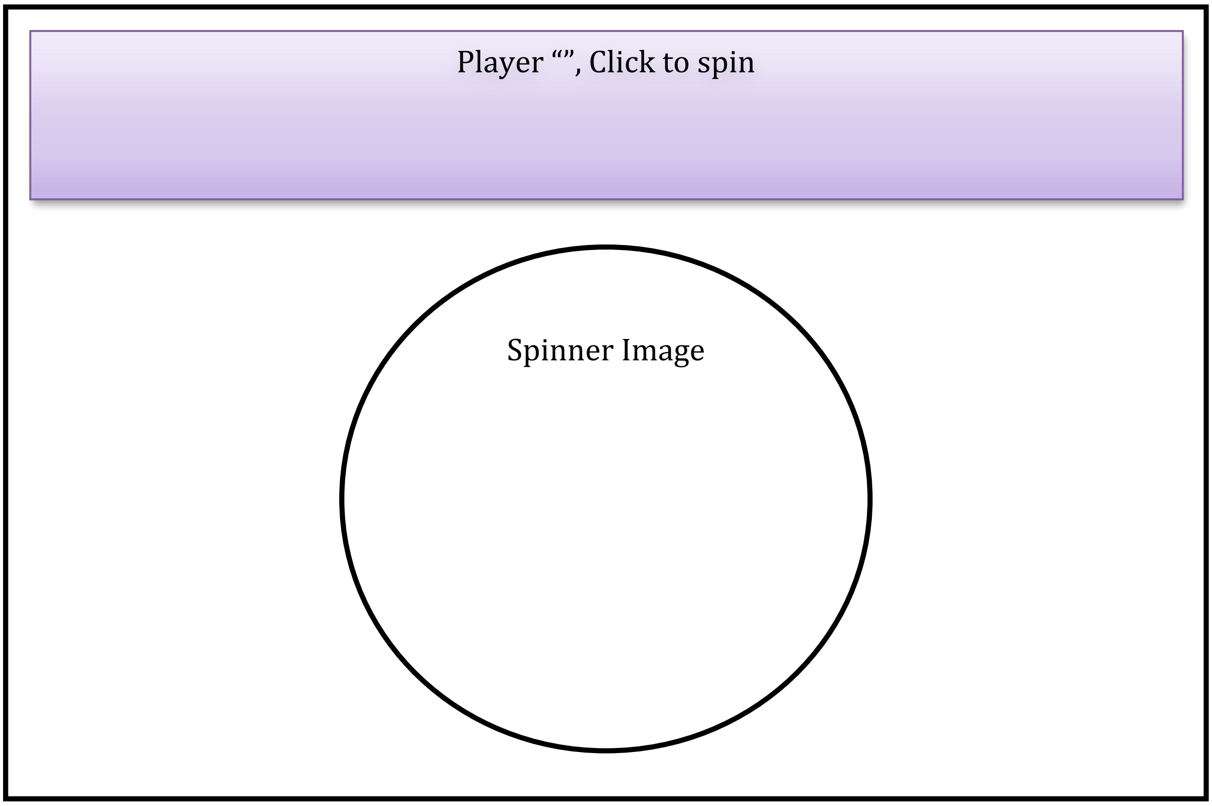
Back

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



38	37	36	35	34
33	32	31	30	29
28	27	26	25	24
23	22	21	20	19
18	17	16	15	14
13	12	11	10	9
8	7	6	5	4
3	2	1	Spin	Menu

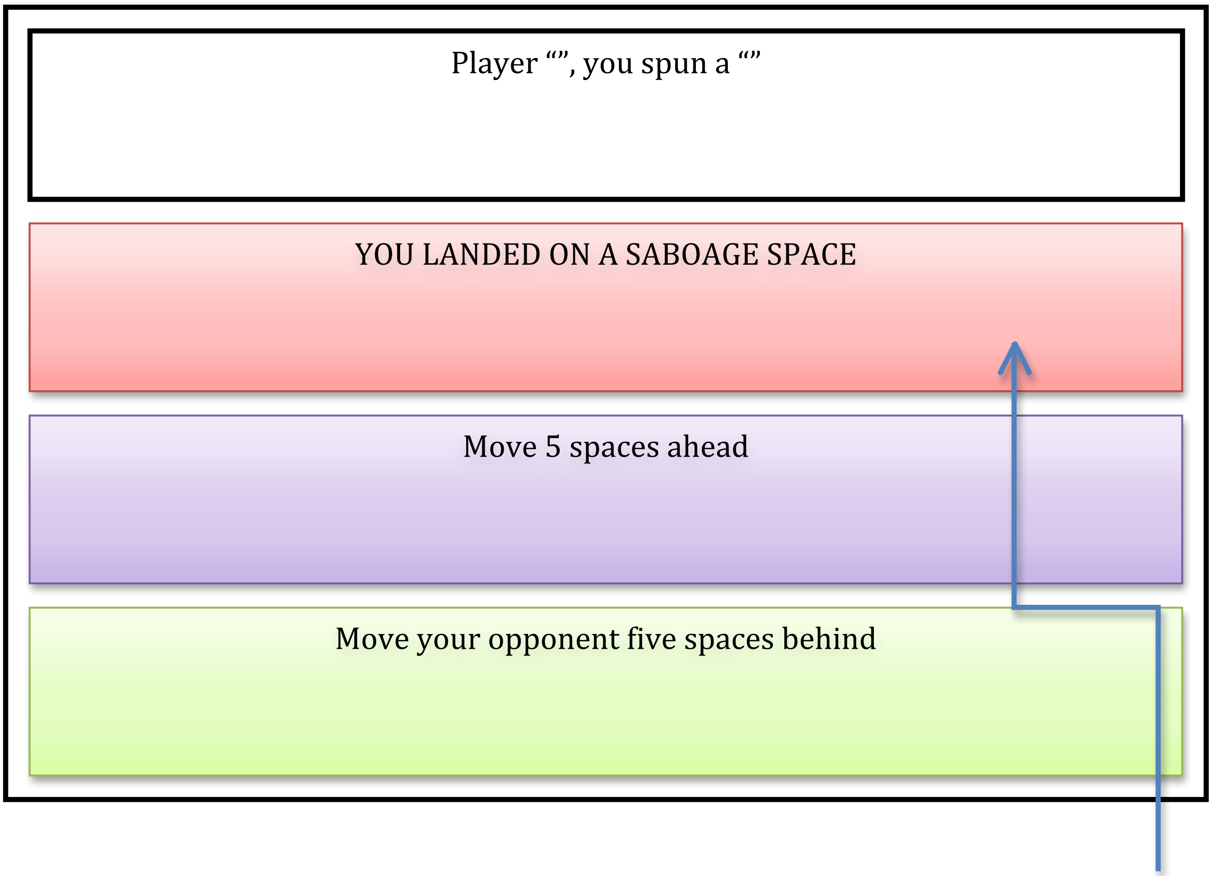
Spinner Screen



Player “”, Click to spin

Spinner Image

Sabotage Screen



Player “”, you spun a “”

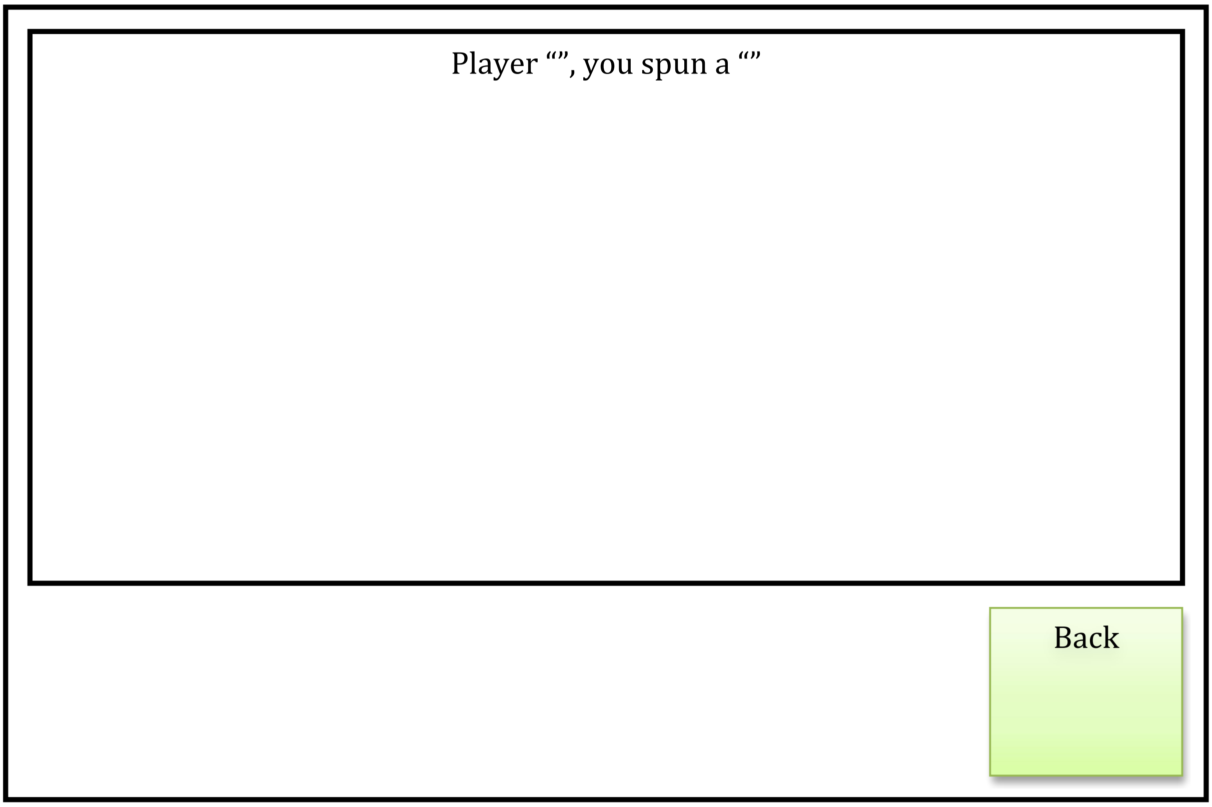
YOU LANDED ON A SABOAGE SPACE

Move 5 spaces ahead

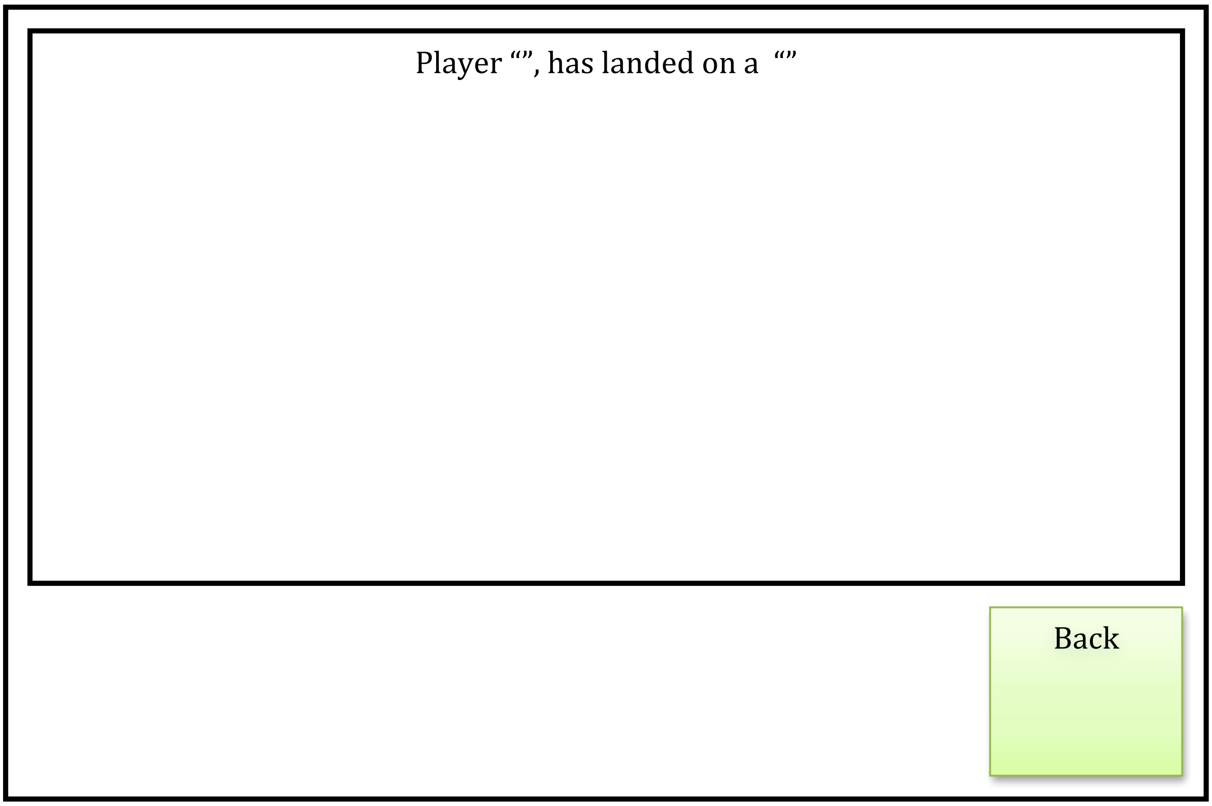
Move your opponent five spaces behind

Gives player information before giving them options for gameplay

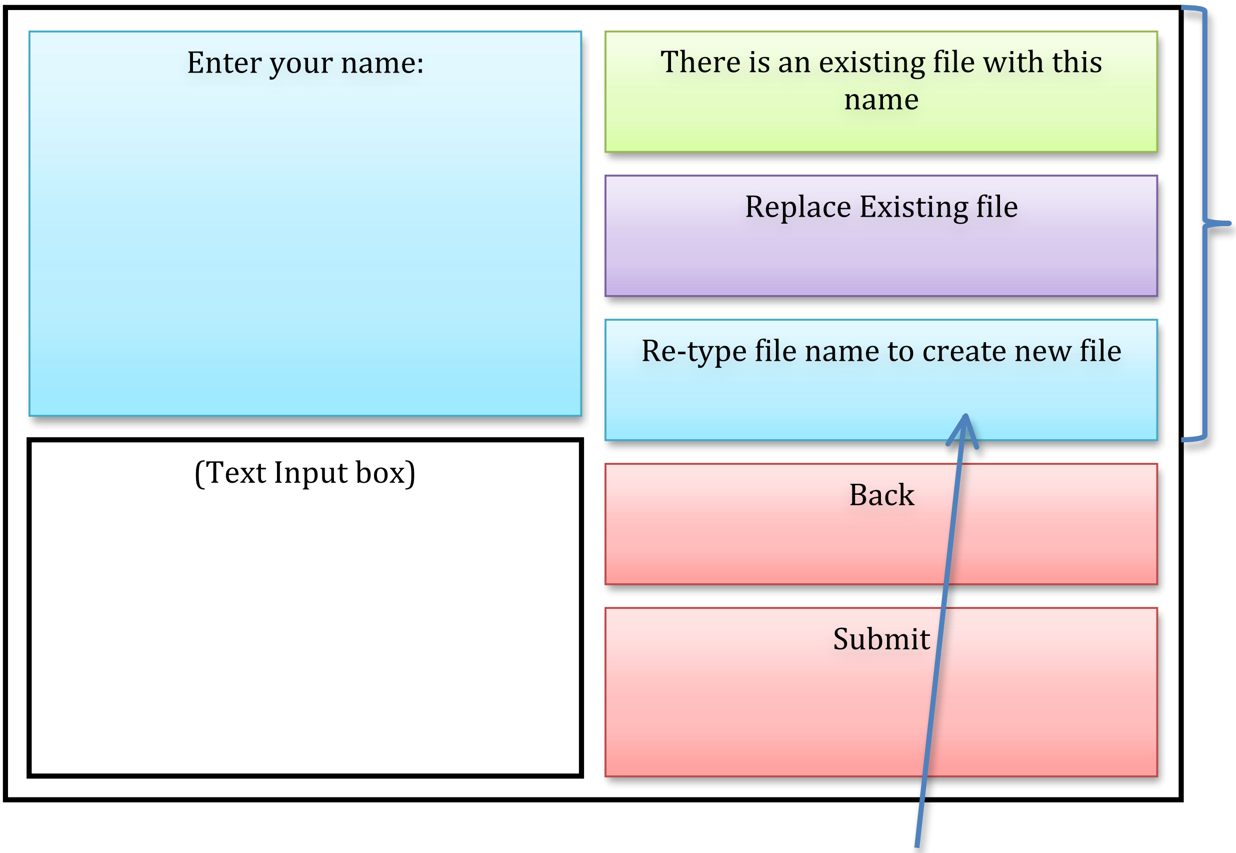
Display Screen


Player “”, you spun a “”

Obstacle Display Screen


Player “”, has landed on a “”

Save Screen



Enter your name:

There is an existing file with this name

Replace Existing file

Re-type file name to create new file

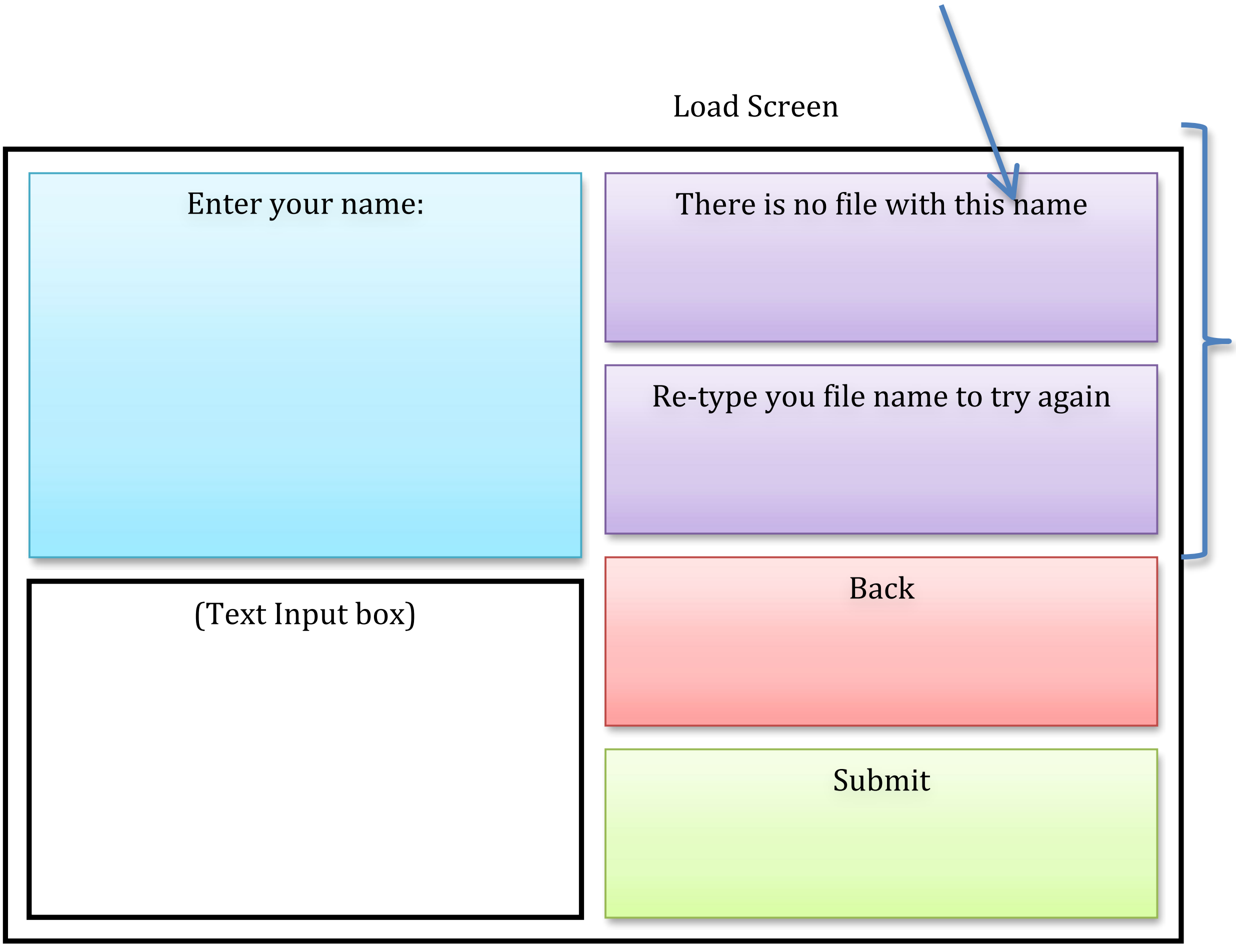
(Text Input box)

Back

Submit

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

Load Screen



Enter your name:

There is no file with this name

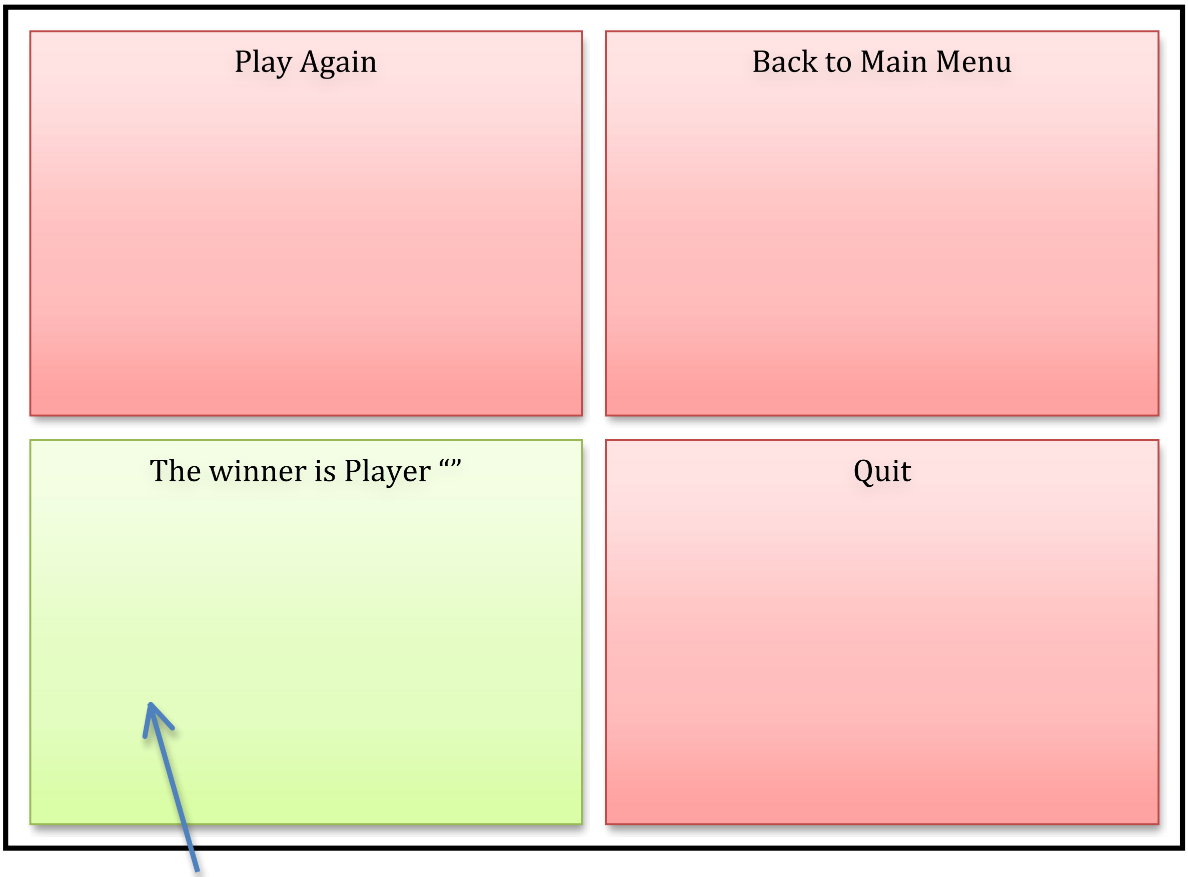
Re-type you file name to try again

(Text Input box)

Back

Submit

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.

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.

Annotated program code

```
from kivy.app import App
from kivy.uix.widget import Widget
from kivy.uix.screenmanager import ScreenManager, Screen, FadeTransition,
WipeTransition, SwapTransition, RiseInTransition, FallOutTransition
from kivy.uix.popup import Popup
from kivy.uix.label import Label
from kivy.uix.button import Button
from kivy.uix.boxlayout import BoxLayout
import random
import time
from kivy.clock import mainthread
from kivy.graphics import BorderImage
from kivy.uix.image import Image
from kivy.graphics import Color, Line, Rectangle
from kivy.uix.filechooser import FileChooserListView, FileChooserIconView
from kivy.core.window import Window; Window.clearcolor = (255,255,255,1)

class SaveWidget(Widget):
    error = False
    '''widget type to enable class object to be added to screenmanager'''
    btn = Button()
    btn2 = Button()
    btn3 = Button()
    '''set buttons as direct children of class so that they can be
    referenced using 'self' or referencing the class object to be
    able to remove and add them within different functions within
    and out of the class'''
    def reset(self):
        self.error = False
    def remove(self):
        gr = self.ids.SaveGrid
        gr.remove_widget(self.btn)
        gr.remove_widget(self.btn2)
        gr.remove_widget(self.btn3)
        '''function is called as entering screen from clicking button
        Save Game, buttons displaying error messages are removed
        so that player is allowed to enter a file name first'''
    def check_input(self):
        names = []
        myLine = ''
        try:
            files = open("filenames.txt", "r")
            myList = files.readlines()
            files.close()
            write_files = open("filenames.txt","a")
            for line in myList:
                for character in range(0,len(line)-1):
                    myLine += line[character]
                names.append(myLine)
                myLine = ''
                '''when text file was saved, '\n' was used to save on
                separate lines, this for loop removes the '\n' to find
the
                root file name'''
            if self.filename.text not in names:
                self.error = False
                name = str(self.filename.text)
                write_files.write(name+"\n")
                write_files.close()
                mw =
self.parent.manager.get_screen('Menu').children[0]
                mw.save_game(self.filename.text)
                self.parent.manager.current = 'Menu'
            else:

                gr = self.ids.SaveGrid
                if self.error == False:
                    self.btn = Button(text="There is an existing
file with this name", size=(Window.width/2, Window.height/5),
pos=(Window.width/2,Window.height*(4/5)),
background_color=(255,255,0,0.3), color=(0,0,0,1))
                    self.btn2 = Button(text="Replace existing
file", size=(Window.width/2, Window.height/5), pos=(Window.width/2,
Window.height*(3/5)),background_color=(0,0,255,0.5), color=(0,0,0,1))
                    self.btn3 = Button(text="Re-type your file
name to create new file",
size=(Window.width/2,Window.height/5),pos=(Window.width/2,Window.heigh
t*(2/5)),background_color=(0,128,128,0.5), color=( 0,0,0,1))
                    self.btn2.bind(on_press=self.save)
                    gr.add_widget(self.btn)
                    gr.add_widget(self.btn2)
                    gr.add_widget(self.btn3)
                    self.error = True
                    '''program opens the text file containing all
the filenames
                    entered so far in the game, if the filename
entered already exists
                    in the text file the error buttons are printed
allowing the
                    player to either save over the file with the
same name or
                    retype their file name. When the submit
button is repressed
```

```
                    this process is repeated so no loop is
required'''
        except FileNotFoundError:
            print("HERE")
            files = open("filenames.txt","w")
            files.write(str(self.filename.text)+"\n")
            files.close()
            mw = self.parent.manager.get_screen('Menu').children[0]
            mw.save_game(self.filename.text)
            self.parent.manager.current = 'Menu'
            '''if the file is not found this means no file has ever been
            saved. In this case, the text file is created and the name
saved,
            then the program skips straight to the saving the game
because if no
            file names exist there can be no error'''



    def save(self, instance):
        mw = self.parent.manager.get_screen('Menu').children[0]
        mw.save_game(self.filename.text)
        self.parent.manager.current = 'Menu'
        '''this function is called from within the same class from the
        button 'replace existing file', when it is clicked. It goes to
        the save function within snakedGrid class to save all of the
        obstacle and player positions'''
class LoadWidget(Widget):
    error = False
    btn = Button()
    btn2 = Button()
    '''set buttons as direct children of class so that they can be
    referenced using 'self' or referencing the class object to be
    able to remove and add them within different functions within
    and out of the class'''
    def reset(self):
        self.error=False
    def remove(self):
        gr = self.ids.LoadGrid
        gr.remove_widget(self.btn)
        gr.remove_widget(self.btn2)
        '''error message buttons removed when Open Saved Game button
        in Menu Widget is clicked, and they are only regenerated when
        player enters an unknown file name'''
    def load_game(self):
        myLine = ''
        names = []
        try:
            files = open("filenames.txt", "r")
            myList = files.readlines()
            files.close()
            for line in myList:
                for character in range(0,len(line)-1):
                    myLine += line[character]
                names.append(myLine)
                myLine = ''
                '''removes '\n' from text file names to find root
names'''
            if self.loadname.text not in names:
                self.not_found()
            else:
                self.error = False
                mn =
self.parent.manager.get_screen('Menu').children[0]
                mn.load_game(self.loadname.text)
        except FileNotFoundError:
            self.not_found()
            '''opens text file containing all saved filenames, if the name
            the player is trying to load is not in the text file, function
            not_found is called(see next comments). if the file
containing
            the names cannot be found that means there aren't any
saved files,
            so the same not_found function is called'''
    def not_found(self):
        if self.error == False:
            gr = self.ids.LoadGrid
            self.btn = Button(text="There is no existing file with this
name", size=(Window.width/2, Window.height/4),
pos=(Window.width/2,Window.height*(3/4)), background_color=(0,0,255,0.5),
color=(0,0,0,1))
            self.btn2 = Button(text="Re-type your file name to try
again",
size=(Window.width/2,Window.height/4),pos=(Window.width/2,Window.heigh
t*(2/4)), background_color=(0,0,255,0.5), color=(0,0,0,1))
            gr.add_widget(self.btn)
            gr.add_widget(self.btn2)
            self.error = True

            '''If the file name entered does not exist within the text file
of
            saved names, the error messages telling the player to re-
enter
            the name appear in the Gridlayout'''

class MenuWidget(Widget):
    file_no = 0
    def remove(self):
        load = self.parent.manager.get_screen('Load').children[0]
        load.remove()
        '''calls the error messages from the load screen to be removed
        so that the page is refreshed. Called when Open Saved Game
        button is pressed'''
    def check(self):
        gd = self.parent.manager.get_screen('Game').children[0]
        sv = self.parent.manager.get_screen('Save').children[0]
        if gd.been_in_game == True and gd.complete == False:
            sv.remove()
            self.parent.manager.current =  'Save'
        else:
            self.ids.Save.color = 0,0,0,0.4
            '''This function handles the error that the Player could
choose to
            save a game when the game screen had not been entered,
which means
            there were no values to save and the program errored. I
made a
            boolean value been_in_game which is set to true when the
game
            board is created and set back to False when the window
reloads,
            and while the value is false if the colour of the save value
text
            is greyed out and the user cannot select it.'''
    def save_game(self, name):
        self.ids.Save.color = 0,0,0,1
        '''save colour text returned back to normal'''
        gd = self.parent.manager.get_screen('Game').children[0]
        sp = self.parent.manager.get_screen('Spinner').children[0]

        self.file_no += 1
        positions = []
        positions2 = []
        positions3 = []
        positions4 =[]
        save = open("SavedGame"+str(name)+".txt","w")
        positions.append(str(gd.one_pos))
        print("P1", gd.one_pos)
        positions.append(str(gd.two_pos))
        print("P2", gd.two_pos)
        positions.append(str(gd.auto_pos))
        print("Computer", gd.auto_pos)
        positions.append(str(gd.player_number))
        print("Number of players", gd.player_number)
        positions.append(str(gd.level))
        print("Difficulty", gd.level)
        positions.append(sp.player)
        print("Current player", sp.player)
        print("player",gd.player_number)
        print("snakes positions", gd.snakes_pos)
        print("ladder positions", gd.ladders_pos)
        print("sabotage space positions", gd.sabotage)
        save.writelines( "%s\n" % item for item in positions )
        save.close()
        save2 = open("SavedS"+str(name)+".txt","w")
        for snake in gd.snakes_pos:
            positions2.append(snake)
        save2.writelines( "%s\n" % item for item in positions2 )
        save2.close()
        save3 = open("SavedL"+str(name)+".txt","w")
        for ladder in gd.ladders_pos:
            positions3.append(ladder)
        save3.writelines( "%s\n" % item for item in positions3 )
        save3.close()
        save4 = open("SavedB"+str(name)+".txt","w")
        for space in gd.sabotage:
            print("sabotage", gd.sabotage)
            positions4.append(space)
        save4.writelines("%s\n" % item for item in positions4)
        save4.close()
        '''makes separate text files to save snakes, ladders and sabotage
        positions and another for everything else. I created them
separately
        because the amount of each are not set, so it is simpler to
        loop through the lines to find out how many there are rather
        than have a set of if statements for each to find out how many.
        The file name selected by the user is saved into the name to make
        it identifiable'''


    def load_game(self,name):
        gd = self.parent.manager.get_screen('Game').children[0]
        ww = self.parent.manager.get_screen('Win').children[0]
        sp = self.parent.manager.get_screen('Spinner').children[0]
        sv = self.parent.manager.get_screen('Save').children[0]
        gd.complete = False
        gd.ladders_pos = []
        gd.snakes_pos = []
        gd.sabotage = []
        gd.loading = True
        ww.restart()
        print("FILE FROM LOAD", sv.filename.text)
        try:
            load = open("SavedGame"+str(name)+".txt","r")
            saved = load.readlines()
            count = 0
            positions = []
            myLine = ''
            for line in saved:
                for character in range(0,len(line)-1):
                    myLine += line[character]
                positions.append(myLine)
                myLine = ''
                '''removes the \n from the lines'''
            gd.one_pos = int(positions[0])
            gd.two_pos = int(positions[1])
            gd.auto_pos = int(positions[2])
            gd.player_number = int(positions[3])
            gd.level = positions[4]
            print("POS4", positions[4])
            if positions[4] == "easy":
                rows = 7
                cols = 5
            elif positions[4] == "inter":
                rows = 9
                cols = 7
            elif positions[4] == "hard":
                rows = 11
                cols = 9


            gd.max = (rows*cols)-2
            for space in range(1,gd.max):
                gd.possible.append(space)
                '''sets rows and columns of game board based on
difficulty
                level saved from previous game, then sets
snakesGrid
                attribute 'max' based on this to set attribute
'possible'
                which is used when placing obstacles on the board'''
            sp.player = positions[5]
            load.close()
            load2 = open("SavedS"+str(name)+".txt","r")
            saved2 = load2.readlines()
            count = 0
            myLine = ''
            for line in saved2:
                for character in range(0,len(line)-1):
                    myLine += line[character]
                gd.snakes_pos.append(int(myLine))
                myLine = ''
            for snake in gd.snakes_pos:
                gd.possible.remove(snake)
                gd.possible.remove(snake-cols)
                if snake <(gd.max-cols):
                    gd.possible.remove(snake+cols)
            load2.close()
            load3 = open("SavedL"+str(name)+".txt","r")
            saved3 = load3.readlines()
            count = 0
            myLine = ''
            for line in saved3:
                for character in range(0,len(line)-1):
                    myLine += line[character]
                gd.ladders_pos.append(int(myLine))
                myLine = ''
            for ladder in gd.ladders_pos:
                gd.possible.remove(ladder)
                gd.possible.remove(ladder+cols)
                if ladder >cols:
                    gd.possible.remove(ladder-cols)
            load3.close()
            load4 = open("SavedB"+str(name)+".txt","r")
            saved4 = load4.readlines()
            count = 0
            myLine = ''
            for line in saved4:
                for character in range(0,len(line)-1):
                    myLine += line[character]
                gd.sabotage.append(int(myLine))
                myLine = ''
            for space in gd.sabotage:
                print (space)
                print (gd.possible)
                gd.possible.remove(space)
            load4.close()

            gd.create_buttons(gd.level)
            gd.parent.manager.current = 'Game'
            '''snakesGrid attributes are set based on the text file
            saved for that game. each time a new obstacle is set,
            that value in the list attribute 'possible' is removed.'''
        except FileNotFoundError:
            print("There is no saved game")
            self.parent.manager.current = 'Menu'
            '''if the game is working correctly, this error should never
            occur because the error checking when a file name is
entered should
            prevent it, but it is another method of error handling in case
            something is not working in the original error checking'''


class GameOptionsWidget(Widget):
    def single_player(self):
        gd = self.parent.manager.get_screen('Game').children[0]
        gd.auto_pos = 0
        gd.one_pos = 0
        gd.player_number = 1
    def two_player(self):
        print(self.parent.manager.get_screen('Game'))
        gd = self.parent.manager.get_screen('Game').children[0]
        gd.one_pos = 0
        gd.two_pos = 0
        gd.player_number = 2
        '''when, on the options screen, either Single Player or Two player
        is selected, these functions are called to set the attributes for
        snakesGrid so the program knows whether to use players one and
        two or one and auto'''

class DifficultyLevel(Widget):
    def easy(self):
        self.parent.manager.current = 'Game'
        gd = self.parent.manager.get_screen('Game').children[0]
        gd.level = 'easy'
        self.set_board()
        gd.create_buttons('easy')

    def inter(self):
        gd = self.parent.manager.get_screen('Game').children[0]
        gd.level = 'inter'
        self.set_board()
        gd.create_buttons('inter')
        self.parent.manager.current = 'Game'

    def hard(self):
        self.parent.manager.current = 'Game'
        gd = self.parent.manager.get_screen('Game').children[0]
        gd.level = 'hard'
        self.set_board()
        gd.create_buttons('hard')
        '''when player selects difficulty from Difficulty Widget the
        program is brought to the corresponding function, so the correct
        amount of buttons for the board are created and the snakesGrid
        attribute 'level' is set to the difficulty. Then, for each,
        the function 'restart' is called from the winner widget which
        ensures all of the snakesGrid attributes are set to their original
        values ready to be altered in the rest of the game'''
    def set_board(self):
        ww = self.parent.manager.get_screen('Win').children[0]
        ww.restart()
```

```
class SabotageWidget(Widget):
    btn = Button()
    def ahead(self):
        self.bring_back_color()
        '''original text colour of 'Behind' button is brought back'''
        self.parent.manager.current = 'Game'
        gd = self.parent.manager.get_screen('Game').children[0]
        sp = self.parent.manager.get_screen('Spinner').children[0]
        if sp.player == 'O':
            gd.one_pos+= 5
            gd.check_for_clashes(1)
            gd.player_pos(1)
        elif sp.player == 'T':
            gd.two_pos += 5
            gd.check_for_clashes(2)
            gd.player_pos(2)
            '''from the Sabotage Widget screen the player can select
whether they
            want to go 5 spaces ahead or move their opponent 5 spaces
back. If they
            choose to move ahead their position is increased by 5, the
new position
            is checked in a grid in snakesGrid to make sure they havent
landed
            on a snake or ladder and their new position is displayed in
snakesGrid
            function 'player_pos'. The player whose turn it is is found in
            Spinner Widget'''
    def behind(self):
        self.bring_back_color()
        '''original text colour of 'Behind' button is returned'''
        gd = self.parent.manager.get_screen('Game').children[0]
        sp = self.parent.manager.get_screen('Spinner').children[0]
        if sp.player == 'O' and gd.player_number == 2:
            print("in behind first if")
            if gd.two_pos>=5:
                gd.two_pos -= 5
                gd.check_for_clashes(2)
                gd.player_pos(2)
                self.parent.manager.current = 'Game'
            else:
                self.grey_out()
        if sp.player == 'O' and gd.player_number == 1:
            if gd.auto_pos>=5:
                gd.auto_pos -= 5
                gd.check_for_clashes('A')
                gd.player_pos('A')
                self.parent.manager.current = 'Game'
            else:
                self.grey_out()
        if sp.player == 'T':
            if gd.one_pos>=5:
                gd.one_pos -= 5
                gd.check_for_clashes(1)
                gd.player_pos(1)
                self.parent.manager.current = 'Game'
            else:
                self.grey_out()
                '''if player selects behind, the program checks how
many players there
                are and which players turn it is to decide which
player position to
                change. The correct player is then taken back 5
spaces, the new position
                checked for clashes with snakes and ladders then the
piece is moved.
                If the opposing player has not yet reached 5 spaces
along the board
                the button text is greyed out and the player is not
allowed to select
                it. For both this button press and 'ahead' button, the
current screen
                is changed to the game screen to continue with the
game'''
    def show(self):
        sp = self.parent.manager.get_screen('Spinner').children[0]
        gd = self.parent.manager.get_screen('Game').children[0]
        gr = self.ids.Sab
        gr.remove_widget(self.btn)
        '''previous banner is removed from grid layout ready for new one'''
        if gd.player_number == 2:
            if sp.player == 'O':
                player = "Player 1"
            elif sp.player == 'T':
                player = "Player 2"
            message = str(player)+", you spun a "+str(sp.spin)
            btn = Button(text=message, font_size=(120),
pos=(0,Window.height*(3/4)), size=(Window.width,(Window.height)/4),
color=(0,0,255,0.5))
            btn.background_normal = 'images/blank.png'
            gr.add_widget(btn)
        elif gd.player_number == 1:
            message = "Player 1, you spun a "+str(sp.spin)
            message2 = "The Computer spun a "+str(sp.auto_spin)
            btn = Button(text=message, font_size=(120),
pos=(0,Window.height*(3/4)), size=(Window.width,(Window.height)/8),
color=(0,0,255,0.5))
```

```
            btn2 = Button(text=message2, font_size=(120),
pos=(0,(Window.height*(7/8))), size=(Window.width,(Window.height)/8),
color=(0,0,255,0.5))
            gr.add_widget(btn)
            gr.add_widget(btn2)
            btn.background_normal = 'images/blank.png'
            btn2.background_normal = 'images/blank.png'
            '''this function makes a banner at the top of the Sabotage
Widget to
            tell the player what they and, is applicable, the computer
has spun
            so hat they can more easily keep track of their positions.
The spin
            values are retrieved from the spinner widget'''

    def grey_out(self):
        self.ids['Behind'].color = 0,0,0,0.3

    def bring_back_color(self):
        self.ids['Behind'].color = 0,0,0,1
        '''called to change the text colour of Behind button depending on
        whether it is a valid option or not'''

class snakesGrid(Widget):
    one_pos = 0
    two_pos = 0
    complete = False
    player_number = 0
    auto_pos = 0
    level = ""
    snakes_pos = []
    ladders_pos = []
    max = 0
    possible = []
    sabotage = []
    loading = False
    been_in_game = False
    '''all attributes made here so they can be referenced using self
    throughout the whole class'''

    def create_buttons(self,dl):
        self.been_in_game = True
        self.complete = False
        mn = self.parent.manager.get_screen('Menu').children[0]
        mn.ids.Save.color = 0,0,0,1
        colours =
[(0,128,128,0.5),(255,0,0,0.4),(255,255,0,0.3),(0,0,255,0.5)]

        '''snakes and ladders lists only deleted in their
        own functions, so shouldn't have to add them in'''
        gl = self.ids.Grid
        if dl == "easy":
            gl.rows = 7
            gl.cols = 5
            ladders = 3
            snakes = 1
            sabotage = 3
            font = 40
        elif dl == "inter":
            gl.rows = 9
            gl.cols = 7
            ladders = 5
            snakes = 5
            sabotage = 6
            font = 35
        elif dl == "hard":
            gl.rows = 11
            gl.cols = 9
            ladders = 5
            snakes = 7
            sabotage = 8
            font = 30
            '''dl passed in from Difficult Widget class when player
selects
            button with difficulty on. Used to decide how many rows,
            columns and obstacles the board should have'''
        total_btns = gl.cols * gl.rows
        self.max = (gl.cols*gl.rows)-2
        for space in range(1,self.max):
            self.possible.append(space)
            '''possible attribute made for use when positioning
obstacles,
            based on size of board from st rows and columns'''

        if self.loading == False:
            self.add_snakes(snakes)
            self.add_ladders(ladders)
            self.add_sabotage_spaces(sabotage)
            self.one_pos = 0
            self.two_pos = 0

        else:
            pass
            '''self.loading is set to True when the program enters the
            Load Game function in the Menu Widget. If the game is
loading
            the obstacles do not need to be created because they are
            already set from the saved game. The player positions also
            do not need to be set back to zero because the player
positions
            were saved'''
        for i in range(1,total_btns-1):
            colour = colours[i%4]
            j = total_btns - 1 - i
            print("Creating ",j)
            btn = Button(id=str(j), text=str(j), color=(0,0,0,1),
background_color = colour, font_size=font)
            if j in self.snakes_pos:
                btn.background_normal = 'images/snake1.png'
                btn.text = ''

            elif j in self.ladders_pos:

                btn.background_color = (0,0,255,0.5)
                btn.background_normal = 'images/ladder1.png'
                btn.background_down = 'images/angry.png'
                btn.isLaddder = True
                btn.text = ''
            elif j in self.sabotage:
                btn.text = "SABOTAGE"
            else:
                btn.text = str(j)
            gl.add_widget(btn)
            '''The buttons are created based on the total buttons based
on
            the difficulty of the game. The background colours are
decided
            by looping through the list of available colours and the text
            for the number space is made from how far the loop the
button
            is made. If the button position is in either the list of snakes,
            sabotage or the list of ladder positions, then the background
image
            is changed to the corresponding image or text. If the image
is a ladder
            i manually set the background colour to blue as some of the
            other colours were not very visible. '''
        colour = colours[(i+1)%4]
        btn1 =
Button(id="Spinner",text="Spin",color=(0,0,0,1),background_color=colour,
font_size=font)
        btn1.bind(on_press = self.spinner)
        gl.add_widget(btn1)
        colour = colours[(i+2)%4]
        btn2 = Button(id="Menu",
text="Menu",color=(0,0,0,1),background_color=colour, font_size=font)
        btn2.bind(on_press = self.menu)
        gl.add_widget(btn2)
        '''the menu and spinner buttons are created at the end of
        the loop, binding their corresponding functions to them
        and deciding background colours from the next two colours
        from the list'''
        if self.loading == True:
            if self.player_number == 2:
                self.player_pos(1)
                self.player_pos(2)
            elif self.player_number == 1:
                self.player_pos(1)
                self.player_pos('A')
        self.loading = False
        '''if the game is being loaded then it means the program has
        not been in move_piece, which is called from spinner widget,
        because the position already exists without being spun.
        Because of this, the function to position the player pieces
        on the board is called separately. Immediately after,
        self.loading is set to False because the game can now run
        normally'''

    def player_pos(self,player):
        gl = self.ids.Grid
        max_high = gl.rows
        if player == 1:
            if self.one_pos == 1 or self.one_pos == 2:
                high = 0
            elif (self.one_pos+2)%gl.cols == 0:
                high = ((self.one_pos+2)/gl.cols)-1
            else:
                high = (round(((self.one_pos+2)/gl.cols)+0.5))-1
                '''variable 'high' is based on the value of the player
position
                divided by the width of the board'''

            if self.one_pos == 0:
                pos = 10000, 0
            else:
                from_right = (self.one_pos+2)%gl.cols #number of
spaces from right
                across = gl.cols-from_right #spaces from left
                if across == gl.cols:
                    across = 0
                print("HIGH", high)
                print("ACROSS", across)
                width = Window.width
                height = Window.height
                pos = width*(across/gl.cols),height*(high/max_high)
            self.ids['p1'].pos = pos
            '''if player position is zero, position player piece so it is
            not visible on board. If the player has a valid position,
            variable across is based on how many spaces are left over
when
            player position is divided by width of board. (player
position
            has 2 added to it because of menu and spinner buttons
taking up
            space on bottom row.) The position is then set by
multiplying
            the window width and height by the fraction of the
variables high
            and across divided by the maximum high or across the
board goes.'''
        elif player == 2:

            if self.two_pos == 1 or self.two_pos == 2:
                high = 0
            elif (self.two_pos+2)%gl.cols == 0:
                high = ((self.two_pos+2)/gl.cols)-1
            else:
                high = (round(((self.two_pos+2)/gl.cols)+0.5))-1


            if self.two_pos == 0:
                pos = 10000, 0
            else:
                from_right = (self.two_pos+2)%gl.cols #number of
spaces from right
                across = gl.cols-from_right #spaces from left
                if across == gl.cols:
                    across = 0
                print("HIGH", high)
                print("ACROSS", across)
                width = Window.width
                height = Window.height
                pos = width*(across/gl.cols),height*(high/max_high)


            self.ids['p2'].pos = pos
        elif player == 'A':
            if self.auto_pos == 1 or self.auto_pos == 2:
                high = 0
            elif (self.auto_pos+2)%gl.cols == 0:
                high = ((self.auto_pos+2)/gl.cols)-1
            else:
                high = (round(((self.auto_pos+2)/gl.cols)+0.5))-1
            if self.auto_pos == 0:
                pos = 10000, 0
            else:
                from_right = (self.auto_pos+2)%gl.cols #number of
spaces from right
                across = gl.cols-from_right #spaces from left
                if across == gl.cols:
                    across = 0
                print("HIGH", high)
                print("ACROSS", across)
                width = Window.width
                height = Window.height
                pos = width*(across/gl.cols),height*(high/max_high)
            self.ids['pA'].pos = pos
            '''The same process is repeated for player two and the auto
            player'''

    def move_piece(self,spin,player):
        print ("Player "+str(player)+":")
        print("Spin is "+str(spin))
        ww = self.parent.manager.get_screen('Win').children[0]
        sp = self.parent.manager.get_screen('Spinner').children[0]
        if(player == 1):
            self.one_pos += spin
            if self.one_pos >=self.max:
                self.one_pos = self.max
                ww.display_winner(1)
            if self.player_number == 2:
                if self.one_pos > 0 and self.two_pos > 0:
                    if self.one_pos == self.two_pos:
                        if self.two_pos == (self.max-1):
                            self.one_pos = self.max
                            ww.display_winner(1)
                        else:
                            self.one_pos += 1
            elif self.player_number == 1:
                if self.one_pos > 0 and self.auto_pos >0:
                    if self.one_pos == self.auto_pos:
                        if self.auto_pos == (self.max-1):
                            self.one_pos = self.max
                            ww.display_winner(1)
                        else:
                            self.one_pos+=1
            sp.display_spin()
            self.check_for_clashes(1)
            self.player_pos(1)
            print("Position is "+str(self.one_pos))
            '''the current spin passed in as a parameter from Spinner
Widget
            is added onto the player position. Which player's position it
            is added to is decided by the other parameter 'player' also
```

```
                        passed from Spinner Widget. If the new addition means the
                        player's position is now above or equal to the maximum
                        position of the board (found from snakesGrid attribute
'max')
                        then the display winner function from the winner widget is
                        called with that player as a parameter. If both of the player
                        positions are valid, and if they are the same as each other,
                        if the opposing player position is one less than the
maximum
                        then when the player skips over that space (because two
                        players cannot be on the same space) that means the
current
                        player has won, so the program also calls 'display winner'
                        The screen to tell the player what they spun is then
                        called, the new position is checked for clashes with board
                        obstacles in function 'check_for_clashes' and the player
                        piece is then moved to the correct position in function
                        "player pos"''
                elif(player == 2):
                        self.two_pos += spin
                        if self.two_pos >= self.max:
                                self.two_pos = self.max
                                ww.display_winner(2)
                        if self.one_pos > 0 and self.two_pos > 0:
                                if self.one_pos == self.two_pos:
                                        if self.one_pos == (self.max-1):
                                                self.two_pos = self.max
                                                ww.display_winner(2)
                                        else:
                                                self.two_pos += 1
                        sp.display_spin()
                        self.check_for_clashes(2)
                        self.player_pos(2)
                        print("Position is "+str(self.two_pos))
                elif player == 'A':
                        self.auto_pos += spin
                        if self.auto_pos >= self.max:
                                self.auto_pos = self.max
                                ww.display_winner('A')
                        if self.one_pos > 0 and self.auto_pos > 0:
                                if self.one_pos == self.auto_pos:
                                        if self.one_pos == (self.max-1):
                                                self.auto_pos = self.max
                                                ww.display_winner('A')
                                        else:
                                                self.auto_pos += 1
                        self.check_for_clashes('A')
                        self.player_pos('A')
                        print("Position is "+str(self.auto_pos))
```

```
        else:
            pass
            '''the same process is repeated for if the current player is
            either player 2 or the computer'''
    def display_obstacle(self,type,player):
        if player == 1:
            p = 'Player One '
        elif player == 2:
            p = 'Player Two '
        elif player == 'a':
            p = 'The Computer '
        if type == 'snake':
            t = 'Oops, '
        elif type == 'ladder':
            t = 'Lucky, '
        message = t+p+"has landed on a "+type
        ob = self.parent.manager.get_screen('Obstacle').children[0]
        sp = self.parent.manager.get_screen('Spinner').children[0]
        message2 = p+"has rolled a "+str(sp.spin)
        self.parent.manager.current = 'Obstacle'
        ob.display(message,message2)

    def check_for_clashes(self,player):
        ww = self.parent.manager.get_screen('Win').children[0]
        sb = self.parent.manager.get_screen('Sabotage').children[0]
        gl = self.ids.Grid
        if player == 1:

            for snake in self.snakes_pos:
                if snake == self.one_pos:
                    self.one_pos -= gl.cols
                    self.display_obstacle('snake',1)
                if self.two_pos == self.one_pos:
                    self.one_pos += 1

            for ladder in self.ladders_pos:
                if ladder == self.one_pos:
                    self.one_pos += gl.cols
                    self.display_obstacle('ladder',1)
                if self.one_pos == self.two_pos:
                    self.one_pos += 1
                if self.one_pos == self.max:
                    ww.display_winner(1)

            for space in self.sabotage:
                if space == self.one_pos:
                    self.parent.manager.current = 'Sabotage'
                    sb.show()
            if self.one_pos >= self.max:
                ww.display_winner(1)
                '''this function is called for when a new player
position has
                been created to check whether the player has landed
on a space
                containing an obstacle. The program loops through
each list
                of positions for each of the types of obstacle and
changed the
                player position accordingly, or, if it is a sabotage
space, calls
                the function to display the sabotage screen. If this
new position
                lands the player on the same space as the other
player, it is
                moved one more forward, or the final space on the
board, then
                the winner widget is called to display them as the
winner'''

        elif player == 2:

            for snake in self.snakes_pos:
                if snake == self.two_pos:
                    self.two_pos -= gl.cols
                    self.display_obstacle('snake',2)
                if self.two_pos == self.one_pos:
                    self.two_pos += 1

            for ladder in self.ladders_pos:
                if ladder == self.two_pos:
                    self.two_pos += gl.cols
                    self.display_obstacle('ladder',2)
                if self.two_pos == self.one_pos:
                    self.two_pos += 1
                if self.two_pos == self.max:
                    ww.display_winner(2)

            for space in self.sabotage:
                if space == self.two_pos:
                    self.parent.manager.current = 'Sabotage'
                    sb.show()

            if self.two_pos >= self.max:
                ww.display_winner(2)

        elif player == 'A':
            for snake in self.snakes_pos:
                if snake == self.auto_pos:
                    self.auto_pos -= gl.cols
                    self.display_obstacle('snake','a')
                if self.auto_pos == self.one_pos:
                    self.auto_pos += 1

            for ladder in self.ladders_pos:
                if ladder == self.auto_pos:
                    self.auto_pos += gl.cols
                    self.display_obstacle('ladder','a')
                if self.auto_pos == self.one_pos:
                    self.auto_pos += 1
                if self.auto_pos == self.max:
                    ww.display_winner('A')

            if self.auto_pos >= self.max:
                ww.display_winner('A')

        else:
            pass
            '''the same process is repeated for the other two players
            depending on whose turn it is, based on the 'player'
            passed from move_piece'''
    def spinner(self,instance):
        self.parent.manager.current = 'Spinner'
        sp = self.parent.manager.get_screen('Spinner').children[0]
        sp.banner()

    def menu(self,instance):
        self.parent.manager.current = 'Menu'
        '''these two functions are bound to the menu and spinner
        buttons created in create_buttons function. They have both
        self and the instance of the button passed in to be able to
        be attached to the buttons. They change the current screen
        the appropriate screen for the button, and the banner
        displaying which player needs to make the spin is made
        in a function in Spinner Widget'''
    def add_sabotage_spaces(self, number):
        for i in range(number):
            space = random.choice(self.possible)
            self.sabotage.append(space)
            self.possible.remove(space)
            '''function that creates sabotage spaces based on which
            spaces are still available, found from snakesGrid attribute
            'possible', then position is deleted from possible. "number"
            is passed in from create_buttons function based on
            difficulty level of game'''

    def add_snakes(self, number):
        self.snakes_pos = []
        gd = self.parent.manager.get_screen('Game').children[0]
        gl = self.ids.Grid
        max = gd.max-1
        for i in range(number):
            pos = 0
            while pos not in range (gl.cols,max) or (pos-gl.cols) not in
self.possible or (pos+gl.cols) not in self.possible:
                print("POS", self.possible)
                pos = random.choice(self.possible)

            self.possible.remove(pos)
            self.possible.remove(pos-gl.cols)
            if pos <(max-gl.cols):
                self.possible.remove(pos+gl.cols)
            self.snakes_pos.append(pos)
            '''snakes positions list is deleted as this function is only
            called at the beginning of each game. maximum position
            allowed is set to one less than snakesGrid max because if
            there was a snake on the end space nobody could win.
"number"
            passed in from create_buttons based on difficulty level. If
            position generated is not in the correct range(at least one
            row up and not on the final square) or the spaces above it
and
            below it are not in "possible", the position keeps being
            generated. Then all the necessary values are removed from
            self.possible ready for the next obstacle to be set. The new
            position is added to the list of snake positions'''

    def add_ladders(self, number):
        self.ladders_pos = []
        print("adding ladders")
        gd = self.parent.manager.get_screen('Game').children[0]
        gl = self.ids.Grid
        print (gd.level)
        for i in range(number):
            pos = 0
            print (self.possible)
            while pos not in range(1,(gd.max-gl.cols)) or (pos-gl.cols)
not in self.possible or (pos+gl.cols) not in self.possible:
                print("POS", self.possible)
                pos = random.choice(self.possible)
                print("pos current", pos)
            print (pos)
            self.possible.remove(pos)
            self.possible.remove(pos+gl.cols)
            if pos > gl.cols:
                self.possible.remove(pos-gl.cols)
            self.ladders_pos.append(pos)
            '''generating ladders is nearly identical to setting the
snakes,
            except the acceptable range it can be positioned changes
from
            having to be at least one row above the minimum, to one
row below
            the maximum'''
class ObstacleWidget(Widget):
    def display(self,message,message2):
        btn = Button(text=message, font_size=80,color = (0,0,255,0.4),
size=(Window.width,(Window.height-300)/2), pos=(0,300))
        btn.background_normal = 'images/blank.png'
        self.ids.ObGrid.add_widget(btn)
        btn2 = Button(text=message2, font_size=80, color = (0,0,255,0.4),
size =(Window.width, (Window.height-300)/2), pos=(0,((Window.height-
300)/2)+300))
        btn2.background_normal = 'images/blank.png'
        self.ids.ObGrid.add_widget(btn2)
        '''message is passed from display_obstacle function in snakesGrid
        based on the player and the obstacle it involved. Uses message in
        a separate screen with a button to tell the player when they have
        encountered an obstacle and what it was'''
class WinnerWidget(Widget):

    def display_winner(self,player):
        self.parent.manager.current = 'Win'
        gd = self.parent.manager.get_screen('Game').children[0]
        gd.complete = True
        if player == 'A':
            self.ids.one.text = "The computer has won"
        else:
            self.ids.one.text = "The Winner is Player "+str(player)
            '''edits an existing button in the winner widget scren to tell
the player
            who has won. player is passed from wherever the winner
has been
            found, getting the current player from the spinner widget'''

    def restart(self):
        grid = self.parent.manager.get_screen('Game').children[0]
        sab = self.parent.manager.get_screen('Sabotage').children[0]
        grid.possible = []
        grid.sabotage = []
        grid.one_pos = 0
        grid.two_pos = 0
        grid.auto_pos = 0
        grid.ids['p1'].pos = 10000,0
        grid.ids['p2'].pos = 10000,0
        grid.ids['pA'].pos = 10000,0
        grid.ids['Grid'].clear_widgets()
        self.parent.manager.current = 'Game'
        '''called whenever the board restarts- so from when the player
        selects the difficulty from the menu. sets all the player positions
        to be off screen and not visible, zeroes out the possible and
        sabotage space lists as well as zeroing the player positions.
        It also clears all the buttons of the game board so that the can
        be remade in the correct amount in create_buttons. The screen is
        then changed to the game screen because it is ready to play'''
class SpinnerWidget(Widget):
    player = 'O'
    spin = 0
    auto_spin = 0
    btn = Button()


    def banner(self):
        self.remove_widget(self.btn)#button removed before remade
        sg = self.ids.SGrid
        gd = self.parent.manager.get_screen("Game").children[0]
        if gd.player_number == 2:
            if self.player == 'T':#player hasn't been reset yet, done
on_press
                message = "Player 1, Click to Spin"
            elif self.player == 'O':
                message = "Player 2, Click to Spin"
        else:
            message = "Player 1, Click to Spin"
        self.btn = Button(text=message,size=((Window.width), 100),pos=
(0, (Window.height-100)),background_color= (0,255,0,0.7), font_size=100 )
        sg.add_widget(self.btn)
        '''creates the banner at the top of the screen to tell the
        players whose turn it is to spin. called when the spinner
        button from the snkesGrid board is pressed.'''

    def do_spin(self):
        self.parent.manager.current = 'Display'
        self.spin = random.randint(1,6)
        gd = self.parent.manager.get_screen('Game').children[0]
        if gd.player_number == 2:
            if self.player == 'O':
                self.player = 'T'
                gd.move_piece(self.spin,2)
            else:
                self.player = 'O'
                gd.move_piece(self.spin,1)
        elif gd.player_number == 1:
            self.auto_spin = random.randint(1,6)
            gd.move_piece(self.spin,1)
            gd.move_piece(self.auto_spin,'A')
            '''called when spinner image in spinner widget screen is
            pressed. determines how many players where are based on
            the snakesGrid attribute player_number. If it is two player
            the current player is changed and the function move_piece
            is called to check and position the player with the new
            randomly generated spin added to it. If the game is single
            player, the player is not changed as the computer's turn is
            automatic, and both player one and the computer are
            positioned with their random spins. The current screen
            is changed to display to tell the players what has been
spun'''

    def display_spin(self):
        ds = self.parent.manager.get_screen('Display').children[0]
        ds.show()
        '''called from move_piece after the spin has been decided,
        to show the player what has been spun'''
class DisplayWidget(Widget):
    def show(self):
        self.parent.manager.current = 'Display'
        sp = self.parent.manager.get_screen('Spinner').children[0]
        gd = self.parent.manager.get_screen('Game').children[0]
        gr = self.ids.Dis
        if gd.player_number == 2:
            if sp.player == 'O':
                player = "Player 1"
            elif sp.player == 'T':
                player = "Player 2"
            message = str(player)+", you spun a "+str(sp.spin)
            btn = Button(text=message, font_size=(120), pos=(0,300),
size=(Window.width,(Window.height-300)), color=(0,0,255,0.5))
            btn.background_normal = 'images/blank.png'
            gr.add_widget(btn)
        elif gd.player_number == 1:
            message = "Player 1, you spun a "+str(sp.spin)
            message2 = "The Computer spun a "+str(sp.auto_spin)
            btn = Button(text=message, font_size=(120), pos=(0,300),
size=(Window.width,(Window.height-300)/2), color=(0,0,255,0.5))
            btn2 = Button(text=message2, font_size=(120),
pos=(0,300+(Window.height-300)/2), size=(Window.width,(Window.height-
300)/2), color=(0,0,255,0.5))
            gr.add_widget(btn)
            gr.add_widget(btn2)
            btn.background_normal = 'images/blank.png'
            btn2.background_normal = 'images/blank.png'
            '''if the number of players from snakesGrid attribute
player_number
            is two, the button is made with either "player one" or
"player two"
            with a blank background from a png image and the current
spin found
            from the spinner widget. If it is single player, two buttons
are
            made to display what the computer and player 1 spun,
using "spin"
            and "auto_spin" attributes from the winner widget'''

class WorkingVersionCApp(App):

    def build(self):
        sm = ScreenManager(transition =  FadeTransition())

        mscreen = Screen(name='Menu')
        mscreen.add_widget(MenuWidget())
        wscreen = Screen(name='Win')
        wscreen.add_widget(WinnerWidget())
        sscreen = Screen(name='Spinner')
        sscreen.add_widget(SpinnerWidget())
        gscreen = Screen(name='Game')
        gscreen.add_widget(snakesGrid())
        oscreen = Screen(name='Options')
        oscreen.add_widget(GameOptionsWidget())
        dscreen = Screen(name='Difficulty')
        dscreen.add_widget(DifficultyLevel())
        bscreen = Screen(name = 'Sabotage')
        bscreen.add_widget(SabotageWidget())
        pscreen = Screen(name = 'Display')
        pscreen.add_widget(DisplayWidget())
        vscreen = Screen(name = 'Save')
        vscreen.add_widget(SaveWidget())
        lscreen = Screen(name = 'Load')
        lscreen.add_widget(LoadWidget())
        obscreen = Screen(name = 'Obstacle')
        obscreen.add_widget(ObstacleWidget())

        sm.add_widget(wscreen)
        sm.add_widget(mscreen)
        sm.add_widget(sscreen)
        sm.add_widget(gscreen)
        sm.add_widget(oscreen)
        sm.add_widget(dscreen)
        sm.add_widget(bscreen)
        sm.add_widget(pscreen)
        sm.add_widget(vscreen)
        sm.add_widget(lscreen)
        sm.add_widget(obscreen)
        sm.current='Menu'

        return sm
        '''the widgets made in the kv file are turned into screens and
        the classes of widgets in the py file are added to them so that
        when referencing a particular widget the program doesn't make
        separate instances. all of these screens are then added to the
        screen manager '''
WorkingVersionCApp().run()
```

<<Kivy File>>

```
<snakesGrid>:
  name:'Game'
    GridLayout:
        id: Grid
        size:root.width,root.height
    Button:
        id: p1
        background_normal: 'images/player1.png'
        size: 100,100

    Button:
        id: p2
        background_normal: 'images/player2.png'
        size: 100,100

    Button:
        id: pA
        background_normal: 'images/player3.png'
        size: 100,100

<DifficultyLevel>:
    name: 'Difficulty'
    Button:
        size: (root.width)/2,(root.height)/2
        text: 'Easy'
        on_press: root.easy()
        pos: 0,0
        background_color: 0,255,255,0.3
        color: 0,0,0,1
        font_size: 40
    Button:
        size: (root.width)/2,(root.height)/2
        text: 'Intermediate'
        on_press: root.inter()
        pos: 0, (root.height)/2
        background_color: 255,255,0,0.3
        color: 0,0,0,1
        font_size: 40
    Button:
        size: (root.width)/2,(root.height)/2
        text: 'Hard'
        on_press: root.hard()
        pos: (root.width)/2, (root.height)/2
        background_color: 0,0,255,0.5
        color: 0,0,0,1
        font_size: 40
    Button:
        text: "Back"
        on_press: root.parent.manager.current = 'Options'
        size: (root.width)/2,(root.height)/2
        pos: (root.width)/2,0
        background_color: 255,0,0,0.4
        color: 0,0,0,1
        font_size: 40


<MenuWidget>:
    name: 'Menu'
    GridLayout:
        Button:
            background_color: 255,0,0,0.4
            size:(root.width)/2,(root.height)/2
            pos: 0,(root.height)/2
            text:"Play Game"
            color: 0,0,0,1
            font_size: 40
            on_press:
                root.parent.manager.current = 'Options'

        Button:
            id: Save
            background_color: 0,0,255,0.5
            size:(root.width)/2,(root.height)/2
            pos: 0,0
            text:"Save Game"
            color: 0,0,0,1
            on_press:root.check()
            font_size: 40
        Button:
            background_color: 0,255,255,0.3
            size:(root.width)/2,(root.height)/2
            pos:(root.width)/2,(root.height)/2
            text:"Open Saved Game"
            color: 0,0,0,1
            on_press: root.parent.manager.current = 'Load'
            on_press: root.remove()
            font_size: 40
        Button:
            background_color: 255,255,0,0.3
            size:(root.width)/2,(root.height)/2
            pos:(root.width)/2,0
            text:"Quit"
            color: 0,0,0,1
            on_press: app.stop()
            font_size: 40

<GameOptionsWidget>:
    name: 'Options'
    GridLayout:
        Button:
            text: "Single Player"
            on_press: root.single_player()
            on_release: root.parent.manager.current = 'Difficulty'
            size: (root.width)/2,(root.height)/3
            pos: 0,0
            background_color: 255,255,0,0.3
            color: 0,0,0,1
            font_size: 40
        Button:
            text: "Two-Player"
            size: (root.width),(root.height)/3
            pos: 0,(root.height)*(2/3)
            on_press: root.two_player()
            on_press: root.parent.manager.current = 'Difficulty'
            background_color: 0,0,255,0.5
            color: 0,0,0,1
            font_size: 40
        Button:
            text: "Back to Main Menu"
            size: (root.width)/2,(root.height)/3
            pos: (root.width)/2, 0
            font_size: 40
            on_press:
                root.parent.manager.current = 'Menu'

            background_color: 0,255,255,0.3
            color: 0,0,0,1
        Button:
            text: "Player 1"
            color: 255,0,0,1
            background_normal: 'images/player1.png'
            size: (root.width)/3, (root.height)/3
```

```
                pos: 0,(root.height)*(1/3)
                font_size: 60
                bold: True
            Button:
                text: "Player 2"
                color: 255,0,0,1
                background_normal: 'images/player2.png'
                size: (root.width)/3, (root.height)/3
                pos: (root.width)*(1/3), (root.height)*(1/3)
                font_size: 60
                bold: True
            Button:
                text: "Computer"
                color: 255,0,0,1
                background_normal: 'images/player3.png'
                size: (root.width)/3, (root.height)/3
                pos: (root.width)*(2/3), (root.height)*(1/3)
                font_size: 60
                bold: True

<WinnerWidget>:
    name: 'Win'
    GridLayout:
        Button:
            pos: 0,0
            size: (root.width)/2,(root.height)/2
            background_color: 255,255,0,0.3
            id:one
            text:''
            color: 0,0,0,1
            font_size: 40
        Button:
            pos: 0, (root.height)/2
            size: (root.width)/2,(root.height)/2
            id:back
            text:'Play again'
            color: 0,0,0,1
            on_press: root.parent.manager.current = 'Options'
            background_color: 255,0,0,0.4
            font_size: 40
        Button:
            id: quit
            background_color: 255,0,0,0.4
            text: "Quit"
            color: 0,0,0,1
            on_press: app.stop()
            size: (root.width)/2,(root.height)/2
            pos: (root.width)/2,0
            font_size: 40
```

```
                Button:
                        text: "Back to Main Menu"
                        on_press: root.parent.manager.current = 'Menu'
                        background_color: 255,0,0,0.4
                        color: 0,0,0,1
                        size: (root.width)/2,(root.height)/2
                        pos: (root.width)/2,(root.height)/2
                        font_size: 40

<SpinnerWidget>:
  name:'Spinner'
  GridLayout:
                id: SGrid
     Button:
                        size:(root.height-200), (root.height-200)
       pos:(root.width-root.height)/2+100,100
       on_press:root.do_spin()

       background_normal: 'images/spinner1.png'


<DisplayWidget>:
        name: 'Display'
        GridLayout:
                id:Dis
        Button:
                size: 300,300
                pos: (root.width)-300, 0
                background_normal: 'images/next.png'
                on_press: root.parent.manager.current = 'Game'

<SabotageWidget>:
        name: 'Sabotage'
        GridLayout:
                id:Sab
                Button:
                        text: "SABOTAGE SPACE"
                        size: root.width, root.height/4
                        pos: 0,root.height*(2/4)
                        background_color: 255,0,0,0.4
                        color: 0,0,0,1
                        font_size: 50
                Button:
                        id: Ahead
                        text:
                                "Move yourself 5 spaces ahead"
                        on_press: root.ahead()
                        size: (root.width),(root.height)/4
                        pos: 0, (root.height)*(1/4)
```

```
            background_color: 0,0,255,0.5
            color: 0,0,0,1
            font_size: 40

        Button:
            id: Behind
            text: "Move your opponent 5 spaces behind"
            on_press: root.behind()
            size: (root.width),(root.height)/4
            pos: 0,0
            background_color: 255,255,0,0.3
            color: 0,0,0,1
            font_size: 40

<SaveWidget>:
    name: 'Save'
    filename: file
    GridLayout:
        id: SaveGrid
        Button:
            text: 'Enter file name:'
            size: root.width/2, root.height/2
            pos: 0, root.height/2
            background_color: 0,128,128,0.5
            color: 0,0,0,1
        TextInput:
            id: file
            size: root.width/2, root.height/2

        Button:
            text: 'Submit'
            size: root.width/2, root.height/5
            pos: root.width/2,0
            on_press:
                root.check_input()
            background_color: 255,0,0,0.4
            color: 0,0,0,1
        Button:
            text: 'Back'
            size: root.width/2, root.height/5
            on_press:
                root.parent.manager.current = 'Menu'
                root.reset()
            background_color: 255,0,0,0.4
            color: 0,0,0,1
            pos: root.width/2, root.height*(1/5)

<LoadWidget>:
    name: 'Load'
    loadname: name
    GridLayout:
        id: LoadGrid
        Button:
            text: "Enter file name"
            size: root.width/2, root.height/2
            pos: 0,root.height/2
            background_color: 255,0,0,0.4
            color: 0,0,0,1
        TextInput:
            id:name
            size: root.width/2, root.height/2
            pos:0,0
        Button:
            text: "Submit"
            size: root.width/2, root.height/4
            pos: root.width/2,0
            on_press: root.load_game()
            background_color: 255,255,0,0.3
            color: 0,0,0,1
        Button:
            text: "Back"
            size: root.width/2,root.height/4
            pos: root.width/2, root.height*(1/4)
            on_press:
                root.parent.manager.current = 'Menu'
                root.reset()
            background_color: 0,128,128,0.5
            color: 0,0,0,1

<ObstacleWidget>:
    name: 'Obstacle'
    GridLayout:
        id: ObGrid
        Button:
            size: 300,300
            background_normal: 'images/next.png'
            on_press: root.parent.manager.current = 'Game'
            pos: root.width-300,0
```

Testing - Test Strategy

My game will need to be tested thoroughly from a user's point of view, and from a programming point of view to make sure that everything works as it should, but also logically so that the user can play the game without any confusion. I will do most of the testing by playing the game in as many different modes and ways that I can to try and come across any errors that I can and correcting them once they are found. For some of the more intricate testing such as the mathematical algorithms that position the player pieces, I will print the background data in Terminal to see what is happening behind the UI, as errors will be hard to trace simply by observing the game.

For testing the UI, I will need to alpha test all of my games functionality including each screen, the screen transitions, the player piece and obstacle positions, the saved and loaded files accuracy, the responses to the user based on actions of the game, whether a player can win, the single player mode of the game, whether the obstacles move the pieces like they are supposed to and the game's quality of play, such as how difficult it is to win and how long it takes for each level of difficulty.

I will begin by testing the menu screens and how they link to one another, to make sure that each button has the correct function bound to it to create the correct screen transition. I will also inspect the logical structure of each screen, including whether the board screen is generated correctly for the current difficulty mode it is in, and their order to make the game as user friendly as possible. I will also need to inspect the clarity of the text on each button on each screen to ensure the user knows what it does and how to use it.

The next thing to test will be that the player and obstacle pieces are generated in the correct place. The obstacles are generated within the buttons so should be easier to test as I can simply loop through the button attributes so see where the obstacles have landed, where as the player positions are based off my own logic so errors could be more unpredictable. I will have to print what the program thinks the positions should be through the Terminal, and then compare that to where they appear to be on the UI screen.

The save and load file options should be able to be tested form the UI by simply checking whether all of the pieces, obstacles and number of board spaces are the same and nothing is generated too many times or anything like that. If there is an error, however, the testing could be complicated as I will need to look into the structure of the text file and the order things are loaded by hand tracing the code, which would be time consuming. When entering the file name to save or load a game, there should either be no existing file with that name when saving, and there should be a file with that name when loading. I can test this with normal, boundary and erroneous data to make sure that no possibilities are allowed which shouldn't be.

Things such as the outputs to the user after a roll and the quality of game play can be tested through the UI, and should be easily fixed if the wrong thing is being outputted, or something is not user-friendly by just changing the text output or checking the value of the variable being used in the message. If something is drastically wrong with the game play, however, it could take more work to solve.

The single player mode can be tested by playing the game and tracing where the piece should be and how it should react in comparison to how it does. In theory, it should not be overcomplicated to test as it is not dissimilar to the two-player mode, only with an automatically generated dice roll and the computer should not respond to the sabotage spaces. Therefore, if there is an error, it should be fixable by looking at the other player modes, assuming those are working. The only real issue would be if there was a logic error in how I implemented the single player mode, in which case I would have to re-evaluate the structure of the game.

To test the obstacles effect on the pieces if they land on them could be time consuming, as I would have to play the game for a long time to get a piece to randomly land on an obstacle in different positions on the board. If I do find any errors such as the players being moved ahead or behind the wrong amount of spaces, hopefully it should be relatively simple to correct by looking at the instructions inside the functions where the program checks for clashes between player and obstacle positions.

Testing if a player can win will also be time consuming because I will have to play through each difficulty level in both single and two player mode to ensure all of the different combinations of players in different modes can all win the game. If there is an error somewhere, it will likely have to do with what the maximum number of spaces is set to, or an error in my logic in when to check whether a player has won. I will have to debug by inspection in that case.

Testing - Test Plan

Test No.	Purpose of the test	Test Data	Expected Outcome	Actual Outcome	Changes Needed	Screen shot Proof
1.1	To see whether the Menu Screen “Save” button links to the correct screen	Click “Save” when game screen has been entered	The screen should transition to the save widget screen with an instruction button, a text input box and a clickable	The screen transitions to the save widget screen and has only the correct boxes.	None.	Fig 1.1

			submit button and nothing else.			
1.2	To see whether save screen is blocked when it should be	Click "Save" when game screen has not been entered	The save option text should grey out once it has been clicked and the screen should not change.	The button does grey out when selected without entering a game.	None.	Fig 1.2
1.3	To see whether the Menu Screen "Load" button links to the correct screen	Click "Load"	The screen should transition to the load widget screen with an instruction button, a text input box and a clickable submit button and nothing else.	The screen transitions to the load widget screen with only the correct boxes.	None.	Fig 1.3
1.4	To see whether the Menu Screen "Play Game" button links to the correct screen	Click "Play Game"	The screen should transition to the Game Options widget where the user can choose to either play single player, two player or go back to the main menu. The screen should also have images in the middle of the page of the player pieces and who they belong to.	Screen transitions to the game options screen with the correct options available, with the player piece images and labels.	None.	Fig 1.4
1.5	To see whether the Menu Screen "Quit"	Click "Quit"	The entire window should shut down.	The window shuts down.	None.	Fig 1.5

	button makes the game quit					
1.6	Check that the correct dimension board with the correct number of obstacles of each type are generated in mode Easy.	Load game in mode “easy”	The board should be 7x5, with 3 ladders, 1 snake and 3 sabotage spaces. Inspect to check this is logical for gameplay.	The board appears with the correct amount of spaces, in the correct format, with the correct obstacles, and seems logical for gameplay.	None.	Fig 1.6
1.7	Check that the correct dimension board with the correct number of obstacles of each type are generated in mode Intermediate.	Load game in mode “intermediate”	The board should be 9x7, with 5 snakes, 5 ladders and 6 sabotage spaces. Inspect to check this is logical for gameplay.	The board appears with the correct amount of spaces, in the correct format, with the correct obstacles, and seems logical for gameplay.	None.	Fig 1.7
1.8	Check that the correct dimension board with the correct number of obstacles of each type are generated in mode Hard.	Load game in mode “hard”	The board should be 11x9, with 5 ladders, 7 snakes and 8 sabotage spaces. Inspect to check this is logical for gameplay.	The board appears with the correct amount of spaces, in the correct format, with the correct obstacles, and seems logical for gameplay.	None.	Fig 1.8
2.1	Make sure the obstacles are generated in the correct places.	Generate board, print in Terminal values of obstacle positions.	The obstacles should appear on the board in the correct places compared to the intended positions printed in the	The obstacle positions match what they should be based on the variable values.	None.	Fig 2.1

			Terminal.			
2.2	Make sure the player pieces land correctly throughout the board.	Press spinner on spinner screen, print the previous player position, the current spin then the current player positions at the end of each turn in the Terminal.	The numbers should match up (test by inspection) and the correct player pieces should appear over the space on the board with the same value as the current player positions printed in the terminal.	They player pieces move in coordination with what the players roll.	None.	Fig 2.2
3.1	To make sure the error message buttons appear on the save widget screen when an existing file name is entered.	Type existing file name "1" into text input box on save widget screen. Erroneous data	Three error message buttons should appear on the right side of the screen telling the user what is wrong, then giving them the option to either write over the file or re-type their file name to try again.	The correct error messages appear in the correct places.	None.	Fig 3.1
3.2	Make sure the error message buttons only appear when data is erroneous.	Type "1." Into text input box on save widget screen and click submit. Boundary data.	The error message buttons should not appear, and the file should be saved.	The error messages do not appear, and when looking through the computer's files, the save files were created.	None.	Fig 3.2
3.3	See whether the file is saved under the	Type "New" into text input box on	The file names in documents should have "New" in the	The filenames are correct in my saved	None.	Fig 3.3

	correct name based on what is entered in the text input box.	save widget screen and click submit. Normal data.	middle of each filename.	files.		
3.4	To see whether the overwrite file button on the save widget screen works.	Type “1” into text input box and click submit, then click “overwrite file” when the error messages appear	The text file when opened in documents should now contain the values of the new game, and when opened should look like the most recently saved game.	The contents of the file changed when the game was overwritten and the new contents match the obstacle and player positions of the overwriting game.	None.	Fig 3.4
3.5	To see whether the error message buttons disappear when going back to the save widget screen.	Type “1” into text input box and submit. Go back to main menu screen and then return to save widget screen.	The error message buttons should no longer be there.	If there was only one set of buttons generated from an incorrect input, it worked fine. I found, however, that if an incorrect name was submitted more than once, more error message buttons were generated on top of the existing ones so not all of them were deleted	I will need to add some error handling so that once the error message buttons are generated, no more can be made until the screen is changed.	Fig 3.5

				when going back to the save page.		
3.5 corrected	To stop buttons from being generated on top of each other when more than one incorrect filename is submitted in the save screen.	Type “1” into the text input box of the save screen several times.	Inspect to see if more buttons are generated on top of each other. If the buttons are generated again, the colour will go darker. This should not happen.	The error buttons are not generated over each other, and disappear when I leave the screen and go back to it.	None.	Fig 3.6
4.1	To see whether the error messages appear when a non-existing file name is entered into the text input of the load screen.	Type “wrong” into load text input and click submit. Erroneous data.	The error message buttons should appear in the far right of the screen.	The error boxes appear correctly, in the right places.	None.	Fig 4.1
4.2	To make sure the error message buttons always appear if a file name doesn’t exist, even if the name is similar.	Save a file name called “test”, then try and load a game called “test.”. Boundary data.	The error messages should appear and the “test” game should not be loaded.	The error messaged do appear correctly, and no game is loaded.	None.	Fig 4.2
4.3	To make sure the error message buttons disappear when re-entering the load screen.	Type “incorrect” into text input box, return to menu screen, then go back into	The error messages should appear when “incorrect” is typed, but should disappear when the	If there was only one set of buttons generated from an incorrect input, it worked fine. I found,	I will need to add some error handling so that once the error message	Fig 4.3

		load screen.	screen is returned to.	however, that if an incorrect name was submitted more than once, more error message buttons were generated on top of the existing ones so not all of them were deleted when going back to the load page.	buttons are generated , no more can be made until the screen is changed.	
4.3 corrected	To stop buttons from being generated on top of each other when more than one incorrect filename is submitted in the load screen.	Type "Wrong" into the text input box of the load screen several times.	Inspect to see if more buttons are generated on top of each other. If the buttons are generated again, the colour will go darker. This should not happen.	The error buttons are not generated over each other, and disappear when I leave the screen and go back to it.	None.	Fig 4.4
4.4	To make sure the error message buttons don't appear when an existing file name is entered.	Type "1" into text input box, click submit. Normal data.	No error messages should appear, the screen should skip straight to the game screen with the existing game loaded onto it.	No error messages appear and the correct game is loaded.	None.	Fig 4.5
5.1	Test accuracy of loaded game.	Save game and make note of player and obstacle	There should be the same number of players, all the pieces should	All of the positions and game modes remain the	None.	Fig 5.1

		positions.	be in the correct place, and the game should be in the correct difficulty mode.	same.		
5.2	To make sure gameplay in continuous from loaded game.	Let game finish with player 1 having a turn, save game.	Player 2 should have first turn when game is loaded.	The turn does go to player 2 when the game is opened.	None.	Fig 5.2
5.3	“”	Repeat vice versa; finish with player 2, save.	Player 1 should have first turn when game is loaded.	The turn does go to player 1 when the game is opened.	None.	Fig 5.3
6.1	To test quality of outputs to the user in the display screen.	Play until player lands on an unoccupied space.	Display screen should appear with correct player and correct value of roll. Check value of roll by returning to game screen and tracking movement.	The display screen appears with the correct value of roll based on value in terminal and on game screen.	None.	Fig 6.1
6.2	To test quality of outputs to the user in the sabotage screen.	Play until player lands on a sabotage space.	Sabotage screen should appear, telling the user that they have landed on a sabotage space and what they rolled, and give them the option to either move ahead or move their opponent behind. The screen should also address the correct player.	Sabotage screen appears with the correct information, addressing the correct player.	None.	Fig 6.2

6.3	To test quality of outputs to the user in the obstacle screen.	Play until a player lands on a snake or ladder.	The user should be made aware of what they rolled, what they have landed on and the correct player should be addressed.	Everything went as planned except that the roll was not displayed to the user, as the display screen was skipped.	Add in message button to tell the player what the rolled on the obstacle screen.	Fig 6.3
6.3 corrected	Test implementation of a button in the obstacle display screen to tell the user what they rolled.	Play until a player lands on a snake or ladder.	The user should be told what they have rolled and what they have landed on.	The obstacle display screen appears with the correct information.	None.	Fig 6.4
7.1	To test the single player mode to make sure that the computer's roll is generated at the same time as the user rolls, and moves at the same time on the board.	Make spin in single player mode.	The Computer's piece should move the same amount of spaces that is displayed in the display screen, at the same time as the player's piece moves.	The computer's piece moves correctly and at the same time as the player piece.	None.	Fig 7.1
7.2	To make sure the computer landing does not have all the same functionality as the player.	Play in single player mode until the computer lands on a sabotage space.	The sabotage screen should not appear, and the game should continue on as normal.	The game continues as normal.	None.	Fig 7.2
7.3	To make sure the	Play in single	The computer's	The computer's	None.	Fig 7.3

	computer is still affected by the obstacles.	player mode until the computer lands on a snake.	piece should be moved a row down.	piece is moved correctly, a row down.		
7.4	“”	Play in single player mode until the computer lands on a ladder.	The computer’s piece should be moved up a row.	The computer’s piece is moved correctly, a row up.	None.	Fig 7.4
8.1	Test if the basic obstacles work effectively for user-controlled players.	Play until player 1 lands on a ladder.	Player 1’s piece should move one row up on the board.	Player 1’s piece moves correctly, one row up.	None.	Fig 8.1
8.2	“”	Play until player 1 lands on a snake	Player 1’s piece should move down one row.	Player 1’s piece moves correctly, one row down.	None.	Fig 8.2
8.3	“”	Play until player 2 lands on a ladder	Player 2’s piece should move one row up on the board.	Player 2’s piece moves correctly, one row up.	None.	Fig 8.3
8.4	“”	Play until player 2 lands on a snake	Player 2’s piece should move one row down on the board.	Player 2’s piece moves correctly, one row down.	None.	Fig 8.4
9.1	Test that, after already testing that the sabotage screen appears, that the sabotage functionality works	Play until player 1 lands on a sabotage space. Select “move yourself ahead”	Player 1’s pieces should be moved 5 spaces ahead on the board.	Player 1’s piece moves correctly, 5 spaces ahead.	None.	Fig 9.1

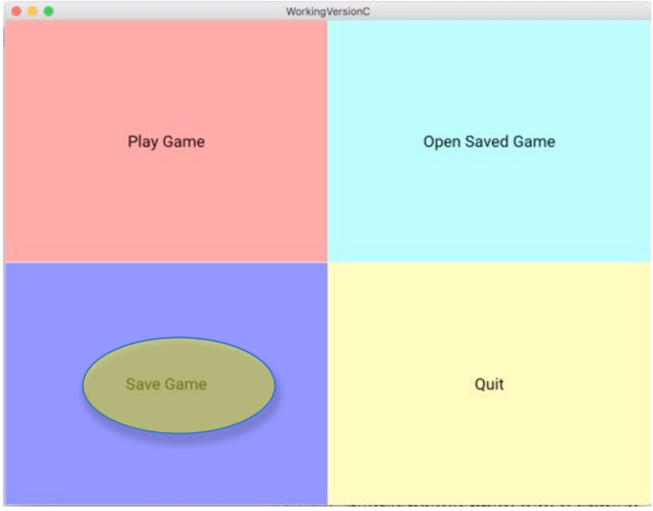
	correctly for player 1.					
9.2	“”	Play until player 1 lands on a sabotage space, while player 2 is more than 5 spaces from the start. Select “move your opponent 5 spaces behind”	Player 2’s piece should be moved 5 spaces behind on the board while player 1’s piece remained in the same space.	Player 1’s piece stays the same, and player 2’s piece is moved correctly, 5 spaces behind.	None.	Fig 9.2
9.3	“”	Play until player 1 lands on a sabotage space, and player 2 is less than 5 spaces from the start. Select “move your opponent 5 spaces behind”	The option’s text should be greyed out once it has been clicked and the screen should remain on the sabotage screen until the “move ahead” option has been selected, when player 1’s piece should be moved 5 spaces ahead.	The ‘move behind’ option is behind is greyed out and the screen remains the same. When the ‘move ahead’ button is selected, Player 1’s piece is moved correctly, 5 spaces ahead.	None.	Fig 9.3
9.4	Test that, after already testing that the sabotage screen appears, that the sabotage functionalit	Play until player 2 lands on a sabotage space. Select “move yourself ahead”.	Player 2’s piece should move 5 spaces ahead on the board.	Player 2’s piece moves correctly, 5 spaces ahead.	None.	Fig 9.4

	y works correctly for player 2.					
9.5	“”	Play until player 2 lands on a sabotage space, while player 1 is more than 5 spaces from the start. Select “move your opponent 5 spaces behind”	Player 1’s piece should be moved 5 spaces behind on the board while player 2’s piece remained in the same space.	Player 2’s piece stays the same, and player 1’s piece is moved correctly, 5 spaces behind.	None.	Fig 9.5
9.6	“”	Play until player 2 lands on a sabotage space, and player 1 is lss than 5 spaces from the start. Select “move your opponent 5 spaces behind”	The option’s text should be greyed out once it has been clicked and the screen should remain on the sabotage screen until the “move ahead” option has been selected, when player 2’s piece should be moved 5 spaces ahead.	The ‘move behind’ option is behind is greyed out and the screen remains the same. When the ‘move ahead’ button is selected, Player 2’s piece is moved correctly, 5 spaces ahead.	None.	Fig 9.6
10.1	See if players can win in all modes: single player in easy.	Load game in single player in easy.	Whichever piece reached the end of the board first, either spinning a value exactly to the end space or above it, should win and the winner	A player wins and the winner screen appears with the correct information.	None.	Fig 10.1

			screen should appear with that character or computer’s name in the message box telling the user who won.			
10.2	See if players can win in all modes: Single player in intermediate.	Load game in Single player in intermediate.	“”	A player wins and the winner screen appears with the correct information.	None.	Fig 10.2
10.3	See if players can win in all modes: Single player in hard.	Load game in Single player in hard.	“”	A player wins and the winner screen appears with the correct information.	None.	Fig 10.3
10.4	See if players can win in all modes: Two player in easy.	Load game in Two player in easy.	“”	A player wins and the winner screen appears with the correct information.	None.	Fig 10.4
10.5	See if players can win in all modes: Two player in intermediate.	Load game in Two player in intermediate.	“”	A player wins and the winner screen appears with the correct information.	None.	Fig 10.5
10.6	See if players can win in all modes: Two player in hard.	Load game in Two player in hard.	“”	A player wins and the winner screen appears with the correct information.	None.	Fig 10.6
10.7	Test buttons from winner screen to	Once a game is won, select “play again”	The screen should go to the game options screen so that the	The screen changed to the game options screen with	None.	Fig 10.7

	make sure that they connect to the correct screens.		user can select and start a new game.	the correct options.		
10.8	“”	Once a game is won, select “Main Menu”	The screen should go to the main menu so that the user has the full range of choices, except for save game as the game is completed.	The user was allowed to save the game, and when this was loaded, the screen game loaded but one of the player positions was on the last space or off of the screen based on their last position value from winning. This then creates a continuous loop where a player has a turn and wins.	The save game option should grey out when it is selected and the game should remain on the main menu screen.	Fig 10.8
10.8 (corrected)	Make sure the user cannot attempt to save a completed game.	Finish a game, select to go back to main menu then try to select ‘Save game’.	The save game option should grey out when it is selected and the screen should stay on the main menu.	The save option greyed out and the screen remained the same.	None.	Fig 10.9
10.9	“”	Once a game is won, select “Quit”	The window should shut down.	The window shuts down.	None.	Fig 10.10

Testing Screenshots



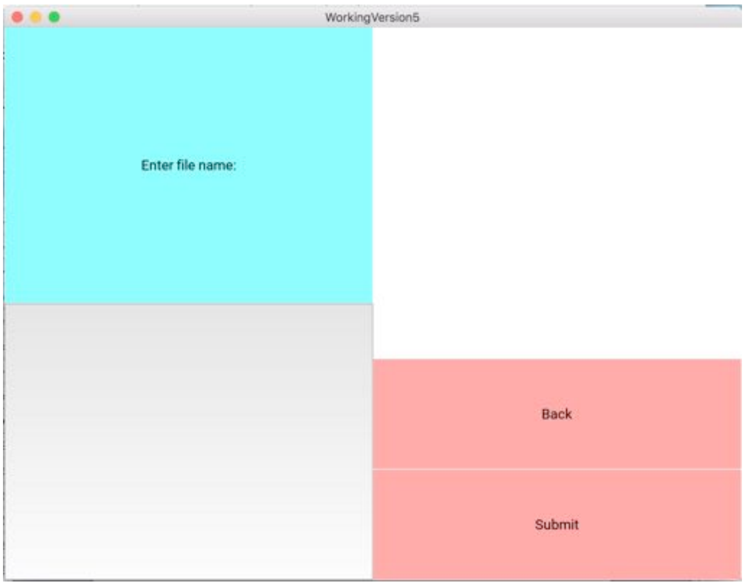
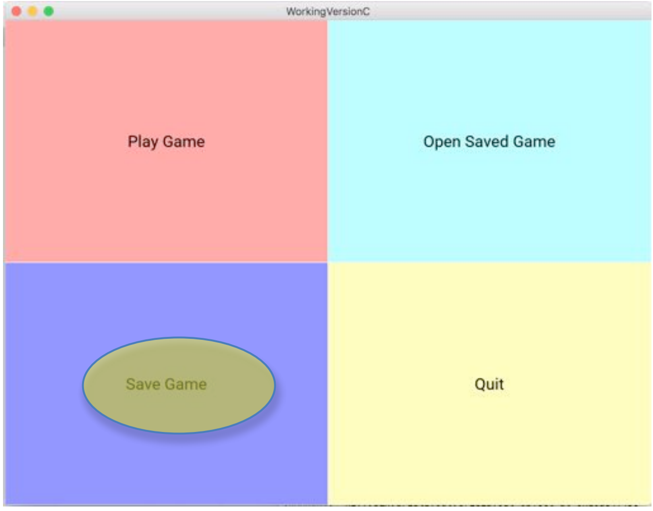


Fig 1.1

The menu screen transitioned to the save screen when ‘Save Game’ was selected. Success.



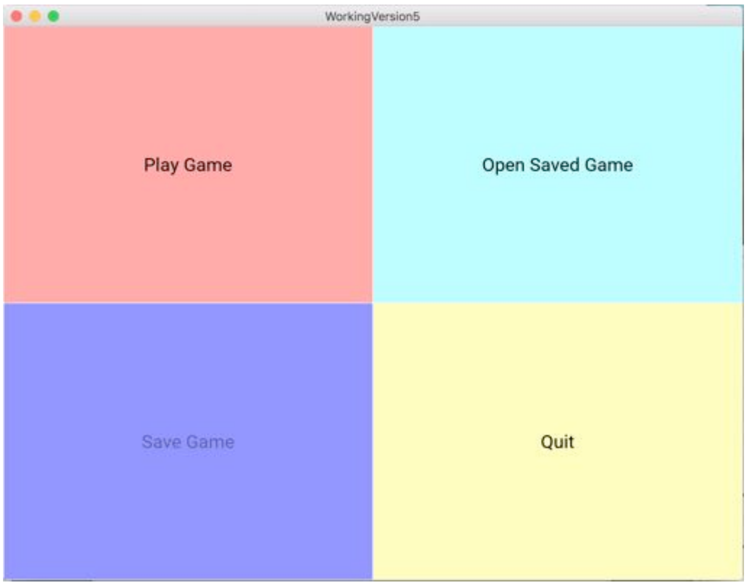


Fig 1.2

The save button was greyed out when it was selected without entering a game first. Success.

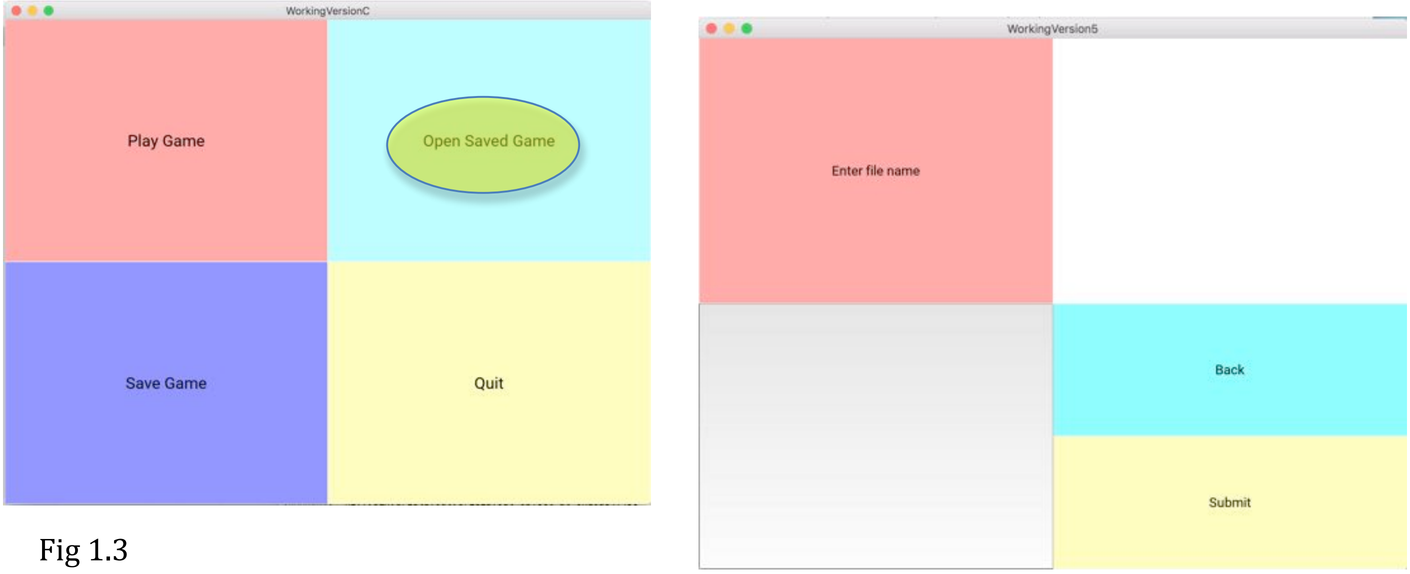


Fig 1.3

The menu screen transitioned to the load screen when ‘Open Saved Game’ was selected

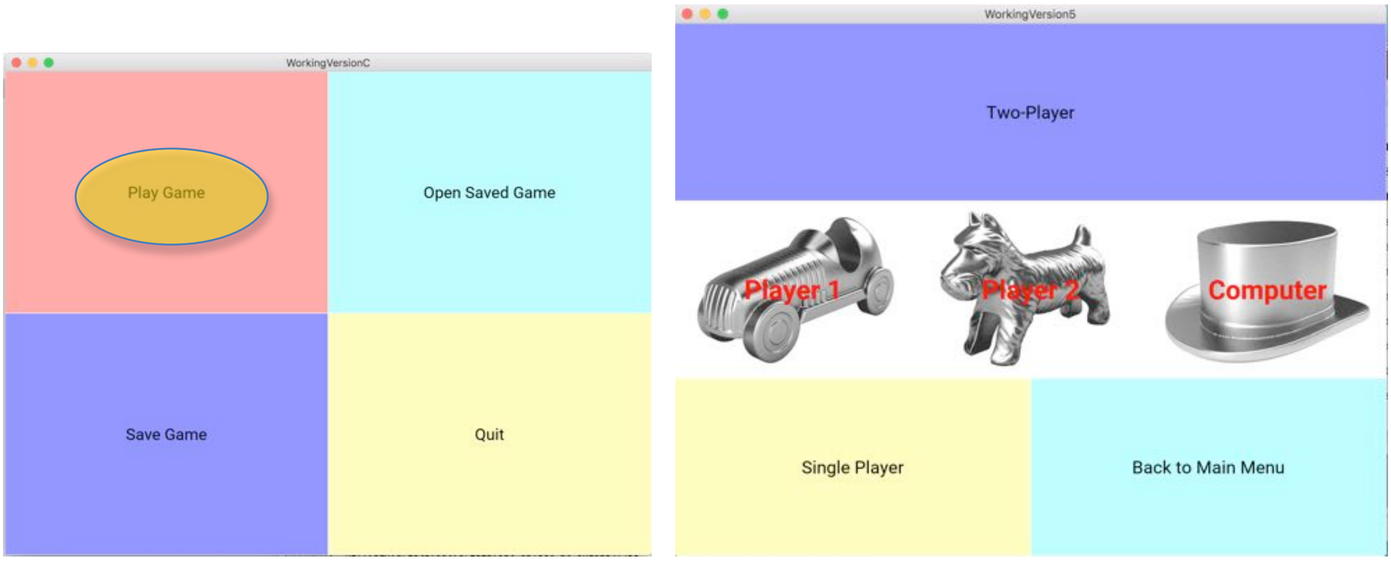
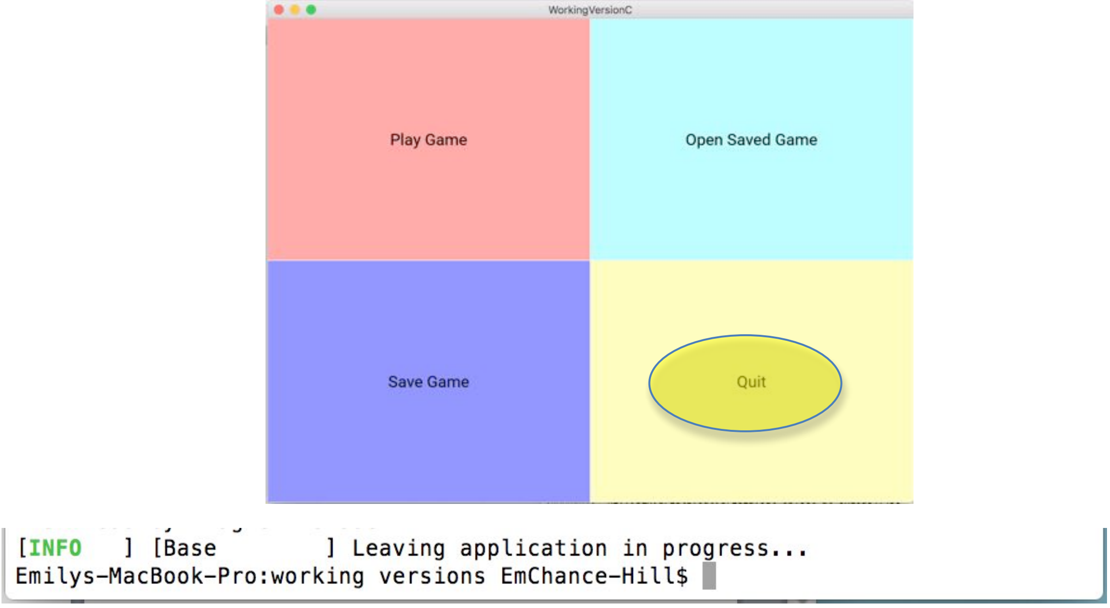


Fig 1.4

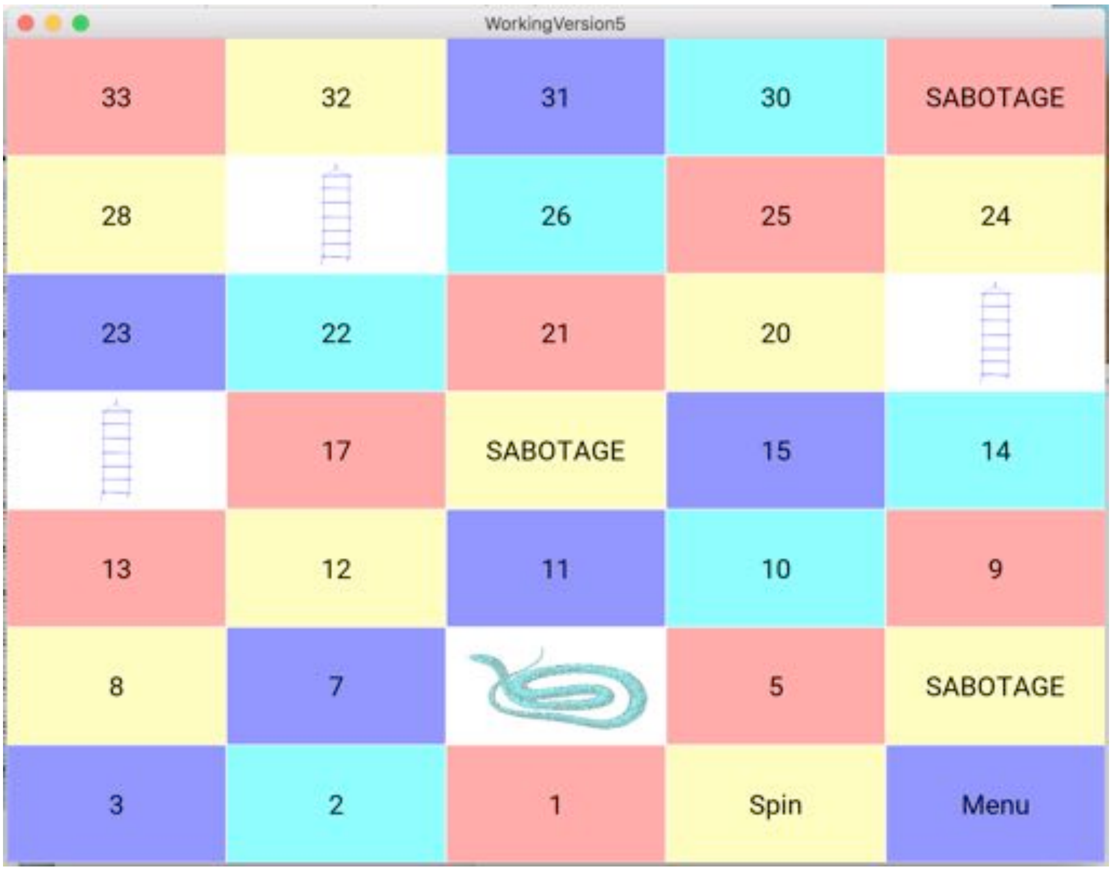
The menu screen transitioned to the game options screen when ‘Play Game’ was selected



```
[INFO   ] [Base        ] Leaving application in progress...
Emilys-MacBook-Pro:working versions EmChance-Hill$
```

Fig 1.5

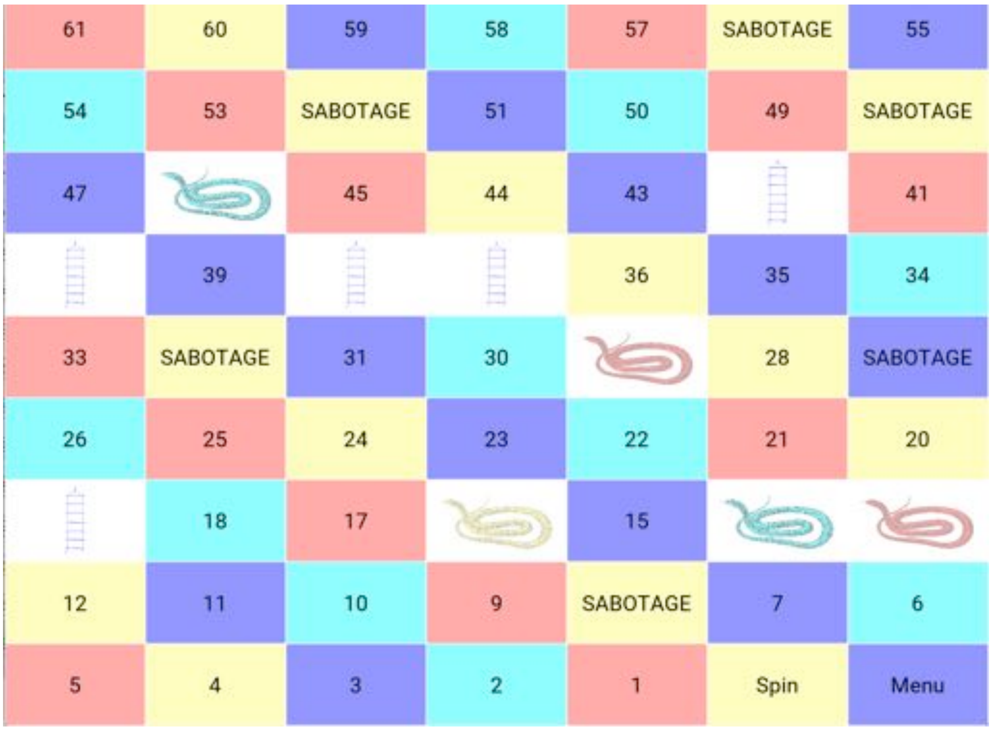
The entire window shut down when 'Quit' was selected. Success.



33	32	31	30	SABOTAGE
28		26	25	24
23	22	21	20	
	17	SABOTAGE	15	14
13	12	11	10	9
8	7		5	SABOTAGE
3	2	1	Spin	Menu

Fig 1.6

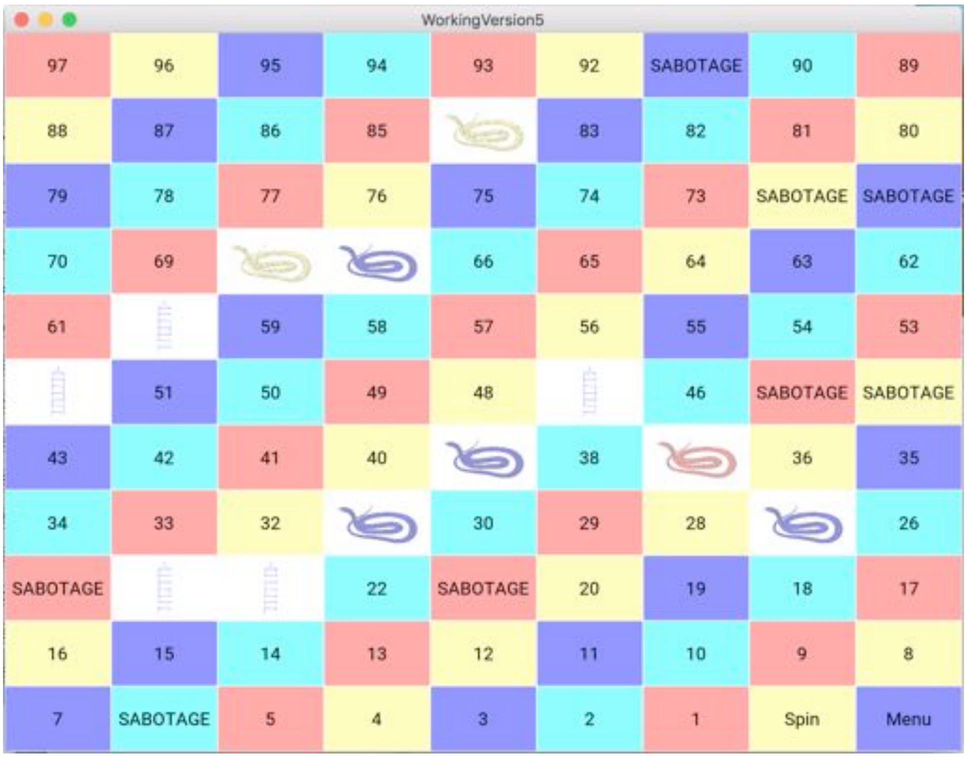
The board when 'easy' level is selected is 7x5, with 3 ladders, 1 snake and 3 sabotage spaces. Success.



61	60	59	58	57	SABOTAGE	55
54	53	SABOTAGE	51	50	49	SABOTAGE
47		45	44	43		41
	39			36	35	34
33	SABOTAGE	31	30		28	SABOTAGE
26	25	24	23	22	21	20
	18	17		15		
12	11	10	9	SABOTAGE	7	6
5	4	3	2	1	Spin	Menu

Fig 1.7

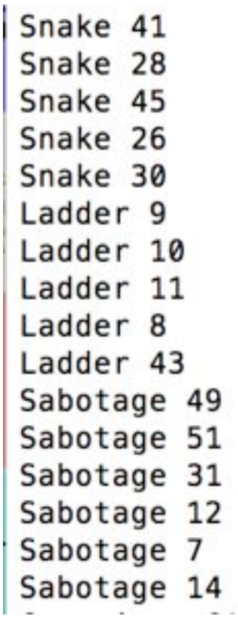
The board when 'intermediate' level is selected is 9x7, with 5 snakes, 5 ladders and 6 sabotage spaces. Success.



97	96	95	94	93	92	SABOTAGE	90	89
88	87	86	85		83	82	81	80
79	78	77	76	75	74	73	SABOTAGE	SABOTAGE
70	69			66	65	64	63	62
61		59	58	57	56	55	54	53
	51	50	49	48		46	SABOTAGE	SABOTAGE
43	42	41	40		38		36	35
34	33	32		30	29	28		26
SABOTAGE			22	SABOTAGE	20	19	18	17
16	15	14	13	12	11	10	9	8
7	SABOTAGE	5	4	3	2	1	Spin	Menu

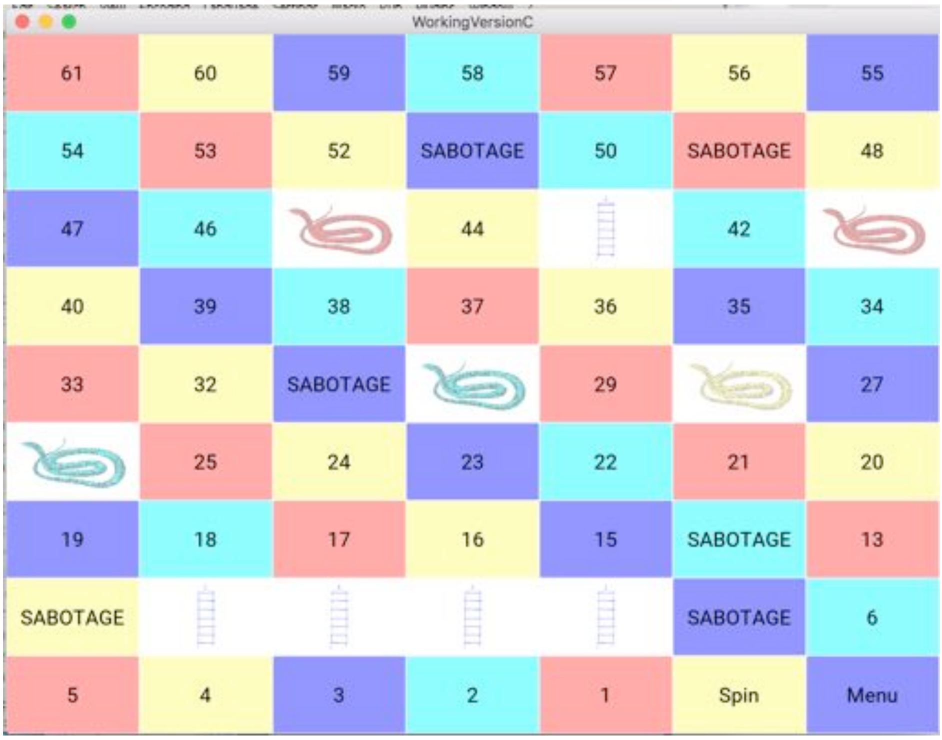
Fig 1.8

The board when 'hard' level is selected is 11x9, with 5 ladders, 7 snakes and 8 sabotage spaces. Success.



```
Snake 41
Snake 28
Snake 45
Snake 26
Snake 30
Ladder 9
Ladder 10
Ladder 11
Ladder 8
Ladder 43
Sabotage 49
Sabotage 51
Sabotage 31
Sabotage 12
Sabotage 7
Sabotage 14
```

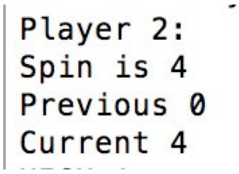
Fig 2.1



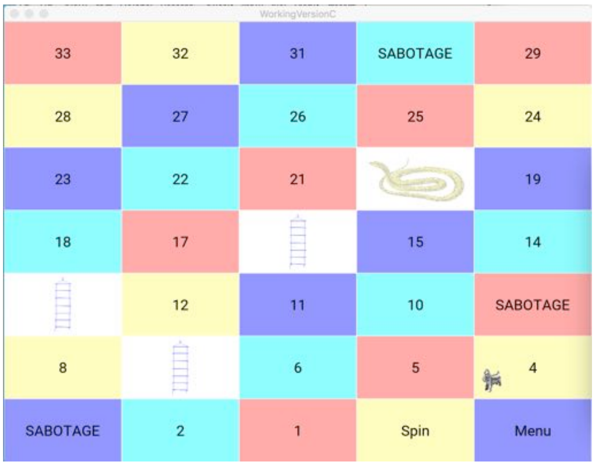
61	60	59	58	57	56	55
54	53	52	SABOTAGE	50	SABOTAGE	48
47	46		44		42	
40	39	38	37	36	35	34
33	32	SABOTAGE		29		27
	25	24	23	22	21	20
19	18	17	16	15	SABOTAGE	13
SABOTAGE					SABOTAGE	6
5	4	3	2	1	Spin	Menu

All of the generated values for the obstacles match up to what can be seen on the image positions on the board. Success.

The player pieces on the board move accurately compared to the randomly generated roll for each turn. Success.

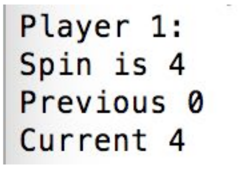


```
Player 2:
Spin is 4
Previous 0
Current 4
```

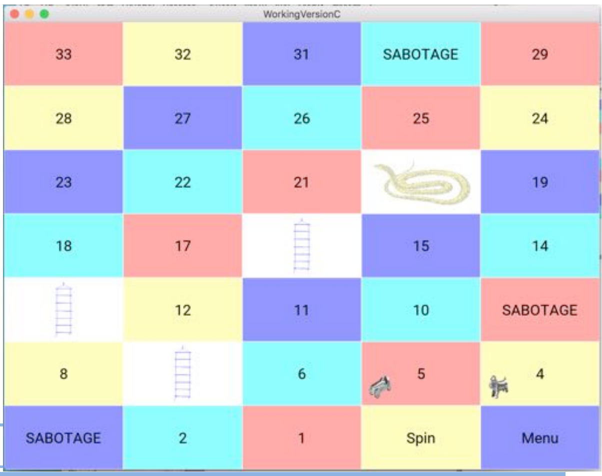


33	32	31	SABOTAGE	29
28	27	26	25	24
23	22	21		19
18	17		15	14
	12	11	10	SABOTAGE
8		6	5	4
SABOTAGE	2	1	Spin	Menu

Fig 2.2



```
Player 1:
Spin is 4
Previous 0
Current 4
```



33	32	31	SABOTAGE	29
28	27	26	25	24
23	22	21		19
18	17		15	14
	12	11	10	SABOTAGE
8		6	5	4
SABOTAGE	2	1	Spin	Menu

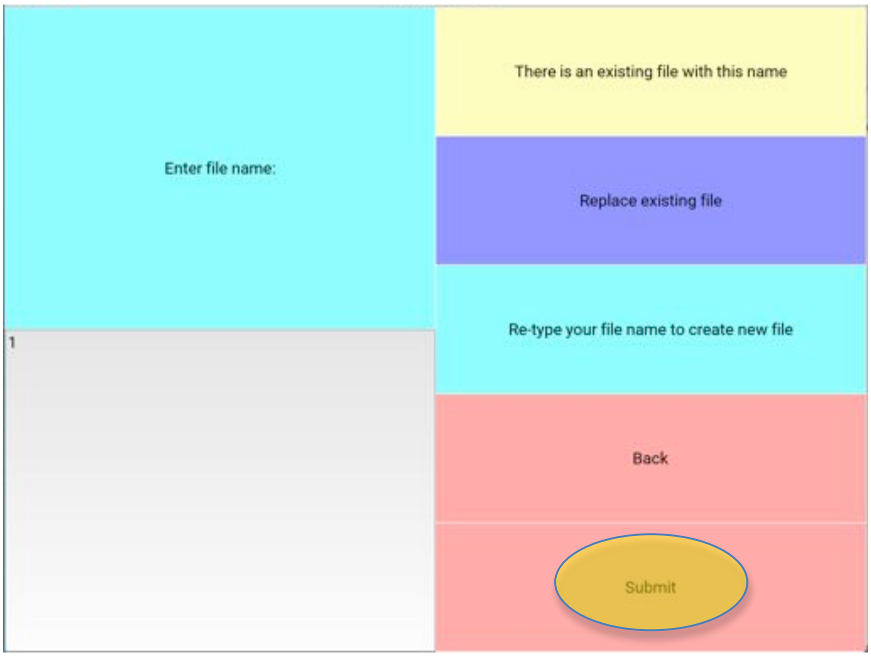

Enter file name:
1
There is an existing file with this name
Replace existing file
Re-type your file name to create new file
Back
Submit

Fig 3.1

When ‘1’ was entered into the text box and submitted, the error messages appeared. Successful, because that file name already exists.

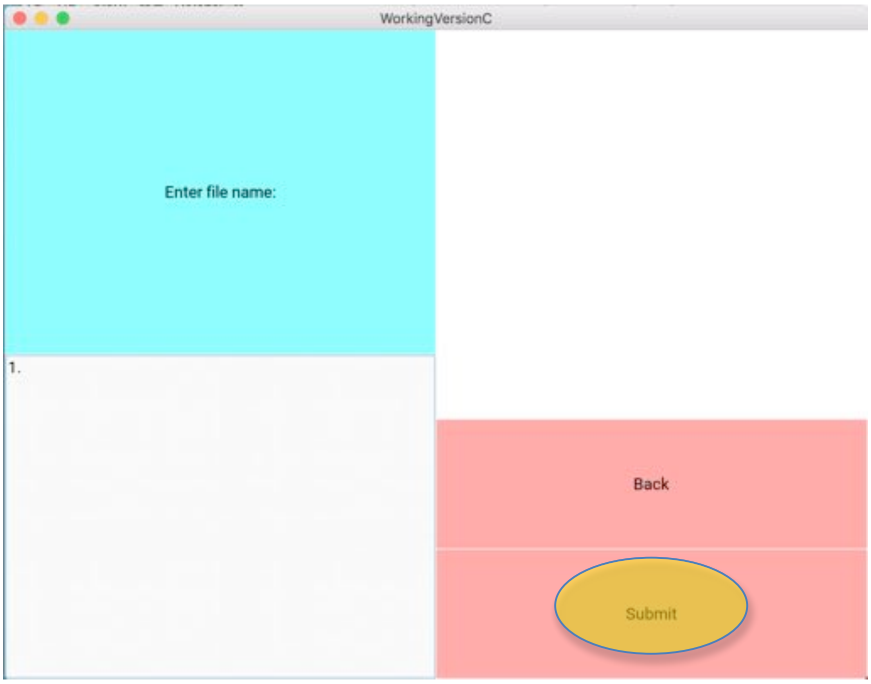
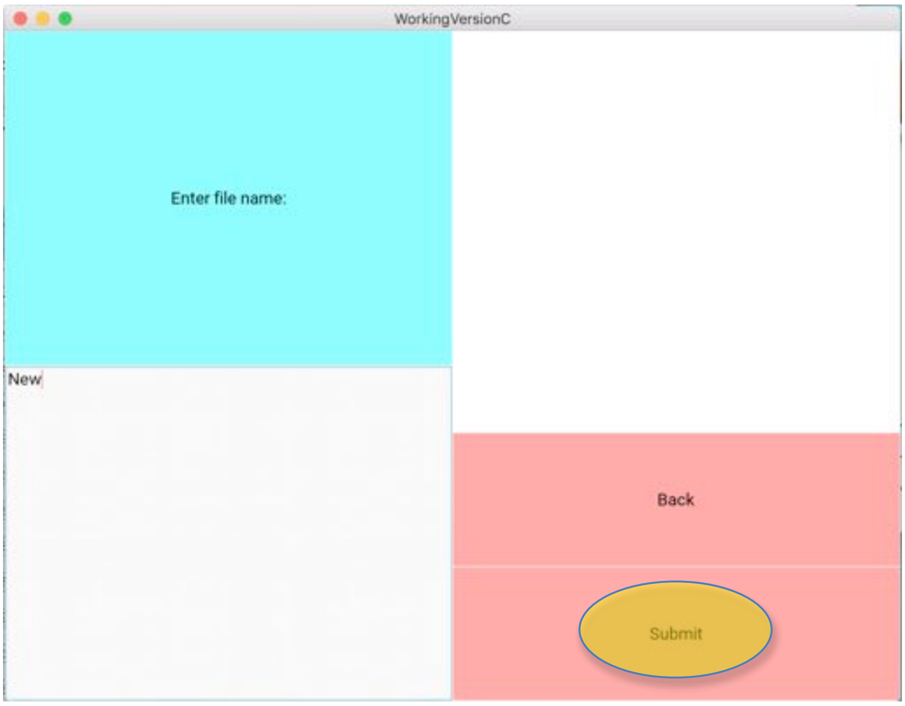


Fig 3.2

When ‘1.’ was entered into the text box and submitted, the error messages did not appear. Successful, because that file did not exist at that time.



Enter file name:

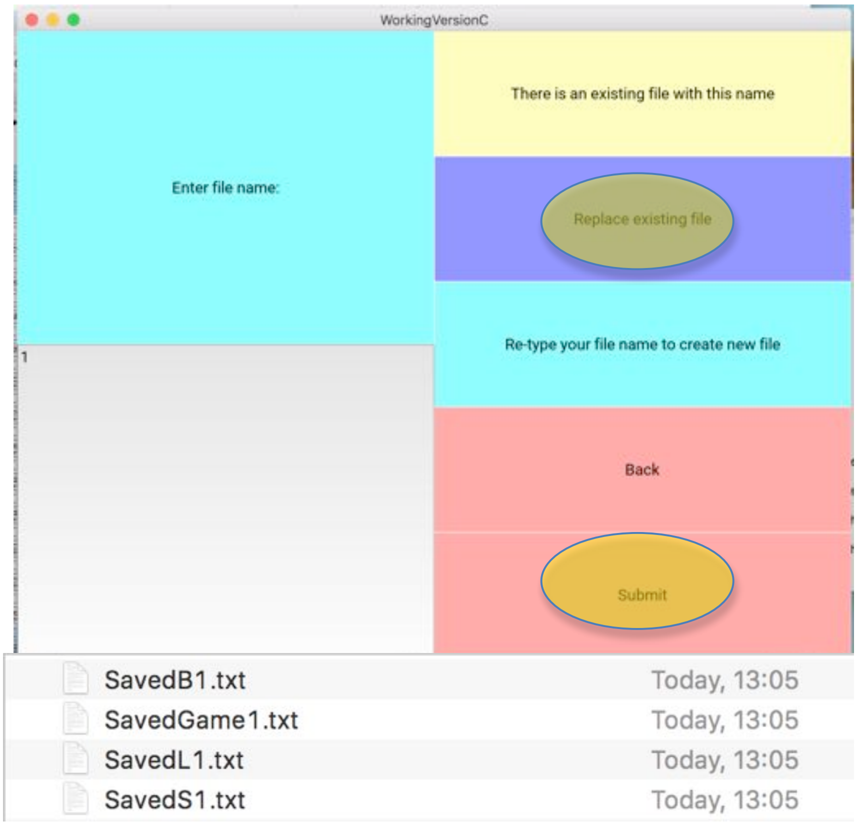
New

Fig 3.3

When ‘New’ was entered into the text box and submitted, the error messages did not appear. Successful, because that file name does not exist.

Fig 3.4

When the ‘replace existing file’ button is pressed, new files are created over the existing files with the unique file name appended correctly. These files contain the correct information for the new game that was saved. Successful.



There is an existing file with this name

Enter file name:

Re-type your file name to create new file

1

SavedB1.txt	Today, 13:05
SavedGame1.txt	Today, 13:05
SavedL1.txt	Today, 13:05
SavedS1.txt	Today, 13:05

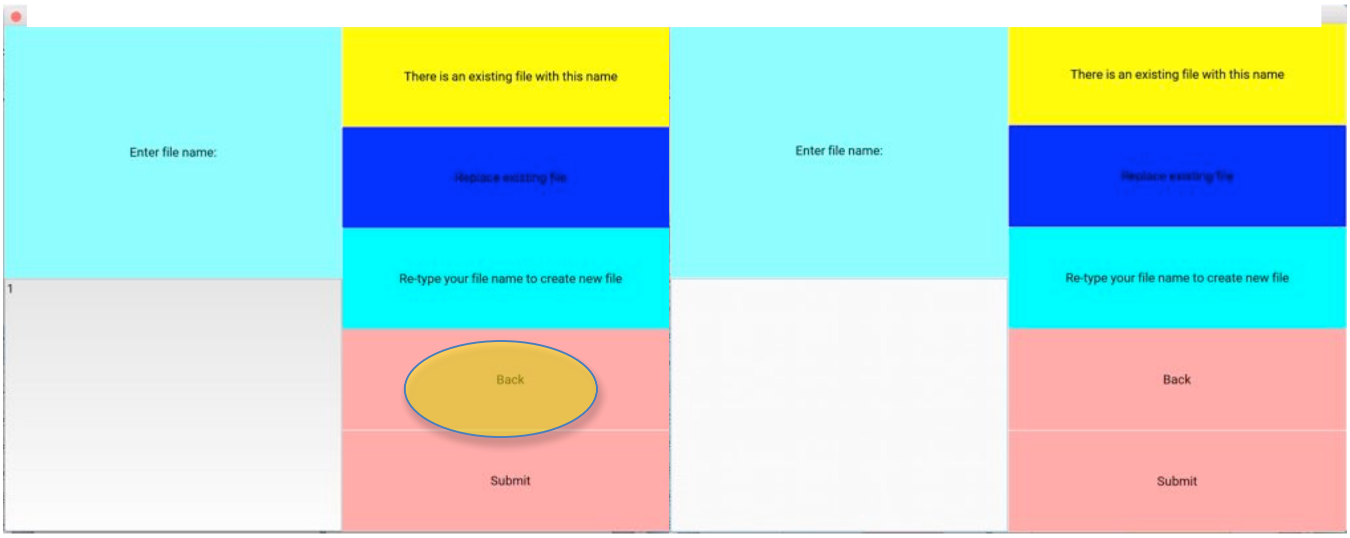

Enter file name:
1
There is an existing file with this name
Replace existing file
Re-type your file name to create new file
Back
Submit
Enter file name:
There is an existing file with this name
Replace existing file
Re-type your file name to create new file
Back
Submit

Fig 3.5

The error messages do not disappear when going back to the save screen after going back to the main menu when more than 1 invalid file name is submitted, because buttons are regenerated over existing ones and when going back to the main menu only the most recently generated buttons are cleared. Unsuccessful.

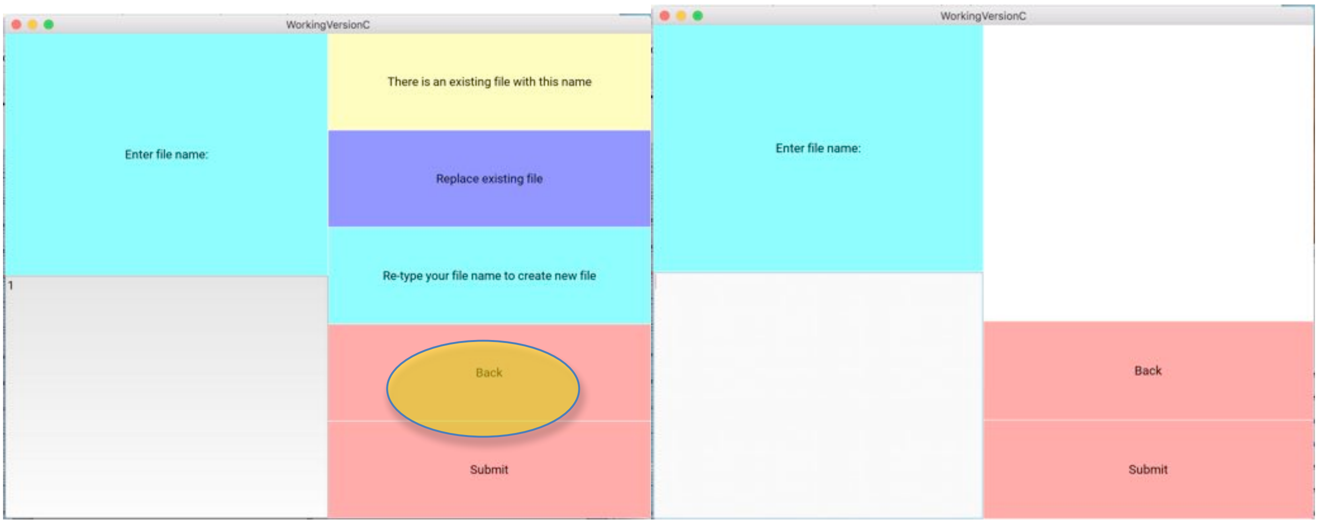
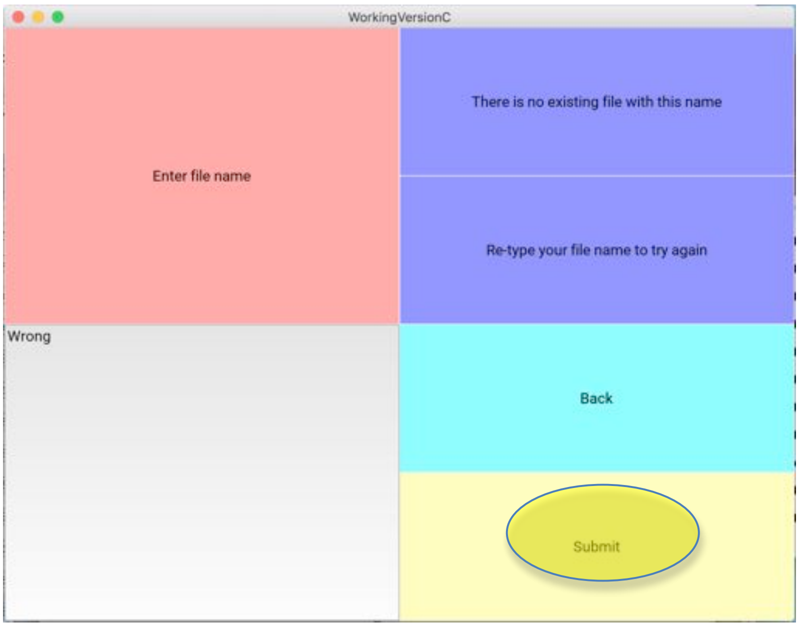


Fig 3.6

The error messages now clear when going back to the save screen from the main menu screen because I added in error handling to make sure no more than 1 set of error buttons can be generated. Successful.

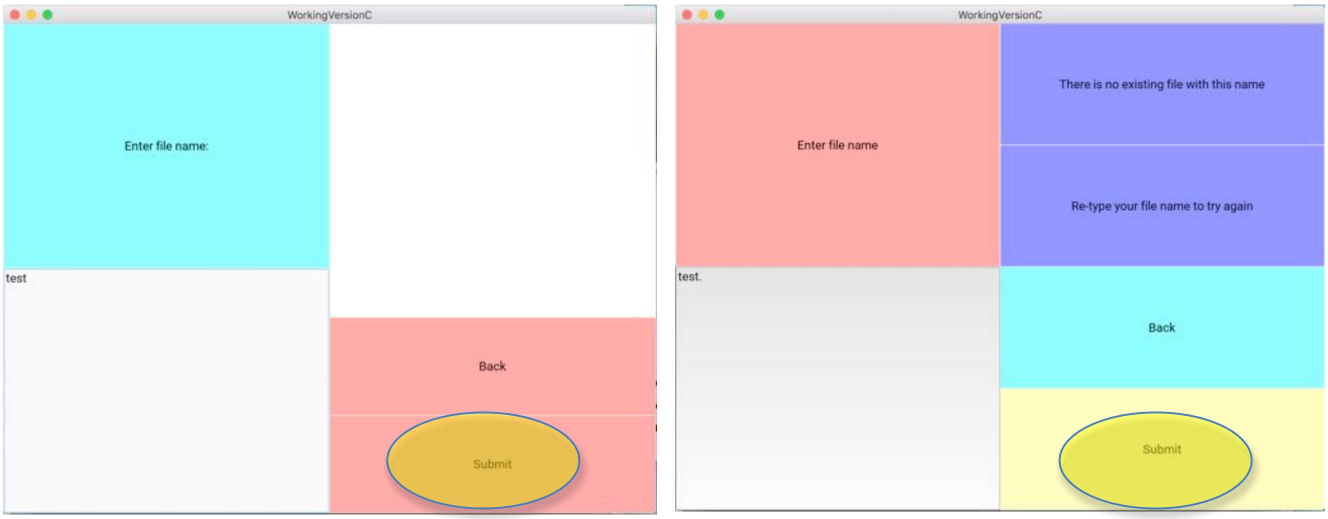


There is no existing file with this name

Re-type your file name to try again

Fig 4.1

When the file name ‘wrong’ is entered, the error messages appear. Successful, because there is no existing file with the name ‘wrong’.



There is no existing file with this name

Re-type your file name to try again

Fig 4.2

After saving a file with name ‘test’, and trying to load a file with name ‘test.’, the error messages appear. Successful, because name is not the same as ‘test’, and no file exists with name ‘test.’.

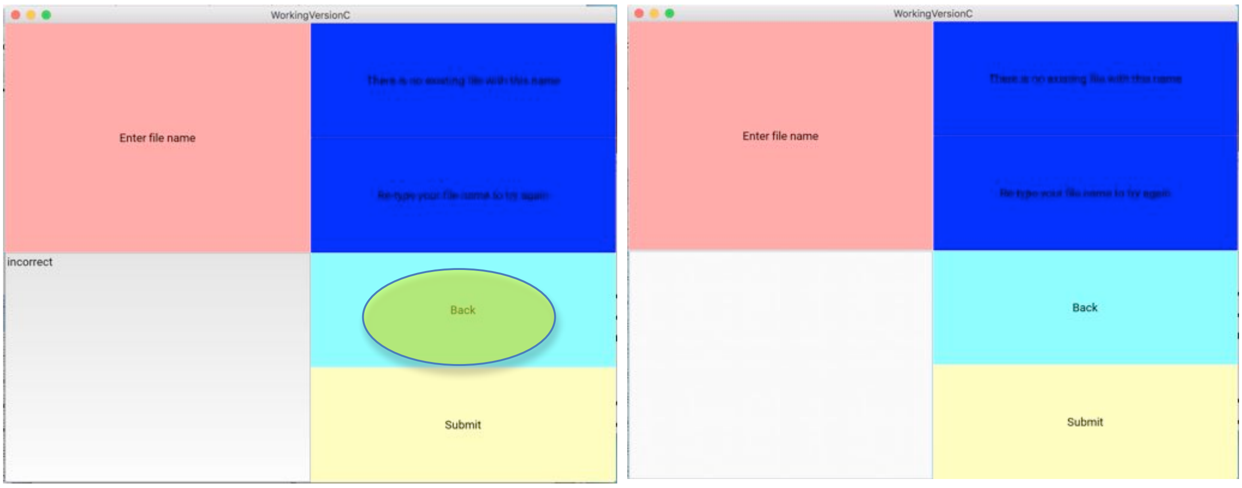

WorkingVersionC
Enter file name
There is no existing file with this name
Re-type your file name to try again
incorrect
Back
Submit
WorkingVersionC
Enter file name
There is no existing file with this name
Re-type your file name to try again
Back
Submit

Fig 4.3

The error messages do not disappear when going back to the load screen after going back to the main menu when more than 1 invalid file name is submitted, because buttons are regenerated over existing ones and when going back to the main menu only the most recently generated buttons are cleared. Unsuccessful.

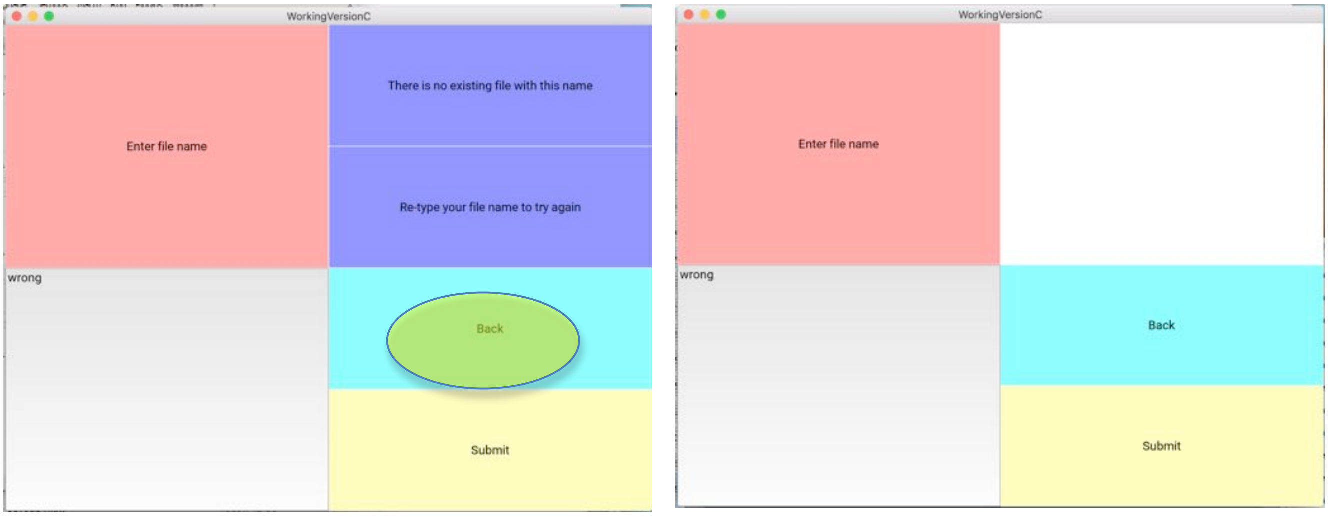


Fig 4.4

The error messages now clear when going back to the load screen from the main menu screen because I added in error handling to make sure no more than 1 set of error buttons can be generated. Successful.

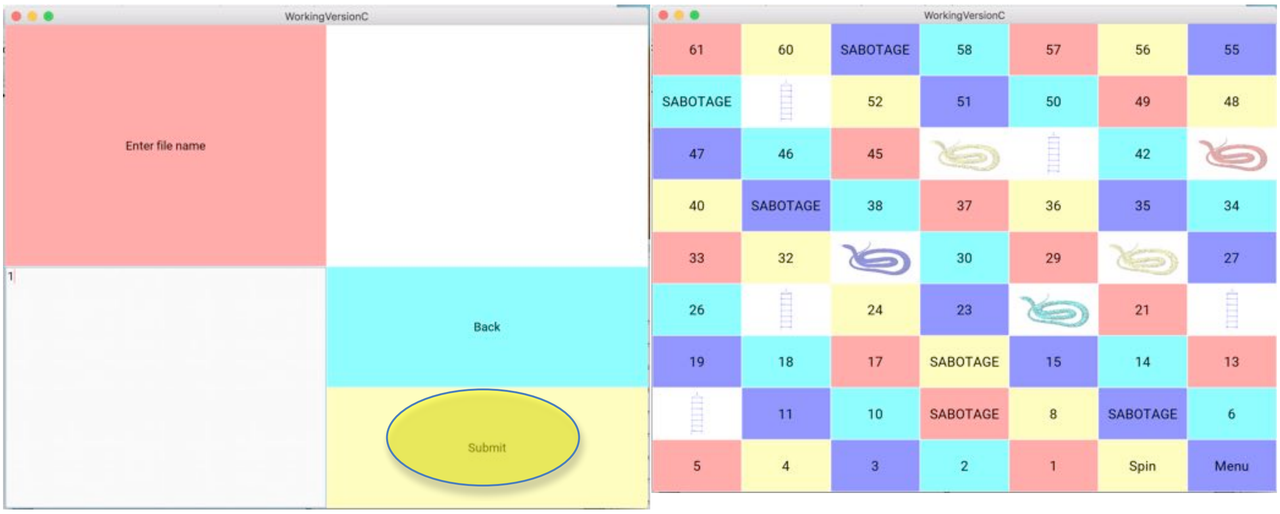
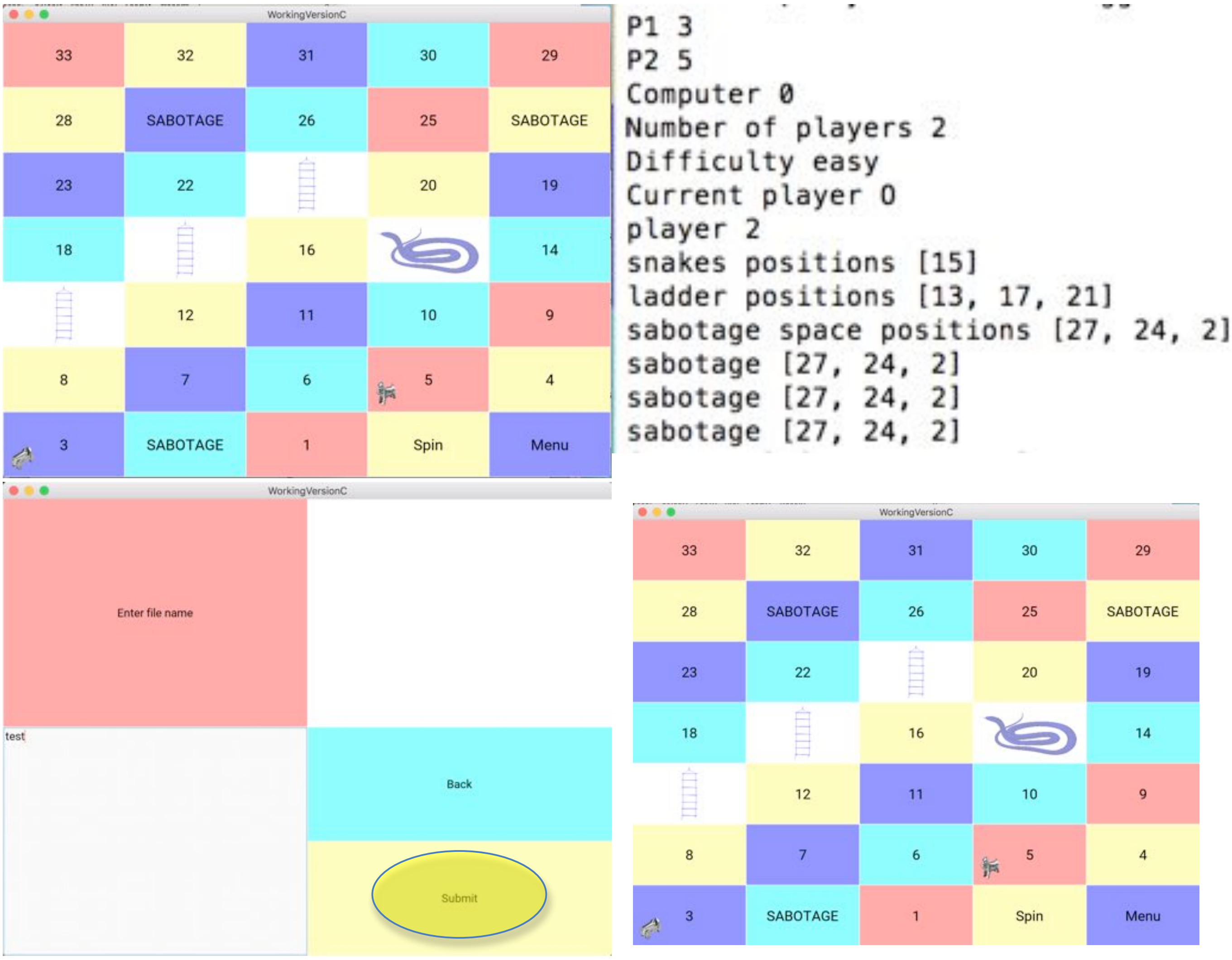


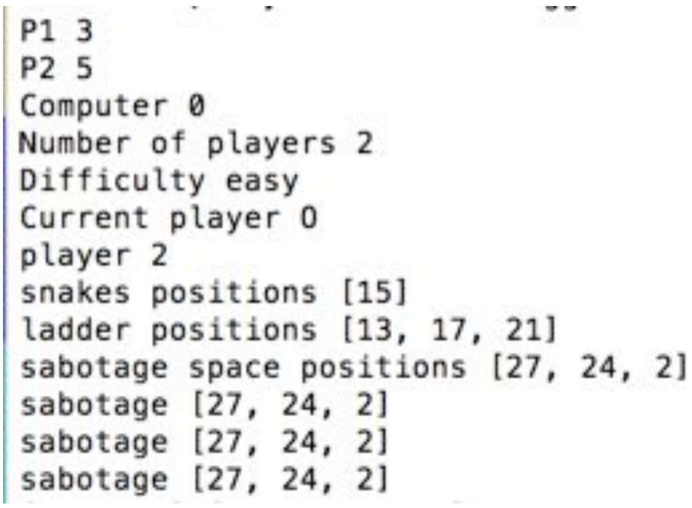
Fig 4.5

No error messages appear when loading file with name ‘1’ because there is an existing file with this name. The game then loads accurately compared to the file that was saved. Successful.



```
P1 3
P2 5
Computer 0
Number of players 2
Difficulty easy
Current player 0
player 2
snakes positions [15]
ladder positions [13, 17, 21]
sabotage space positions [27, 24, 2]
sabotage [27, 24, 2]
sabotage [27, 24, 2]
sabotage [27, 24, 2]
```

Fig 5.1 Played game, saved and printed values in terminal. Loaded same game and the reproduction was the same. Successful.



```
P1 3
P2 5
Computer 0
Number of players 2
Difficulty easy
Current player 0
player 2
snakes positions [15]
ladder positions [13, 17, 21]
sabotage space positions [27, 24, 2]
sabotage [27, 24, 2]
sabotage [27, 24, 2]
sabotage [27, 24, 2]
```

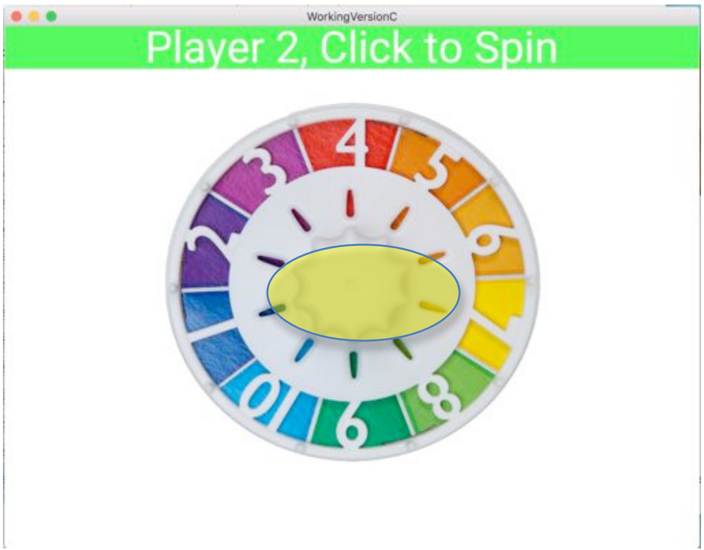
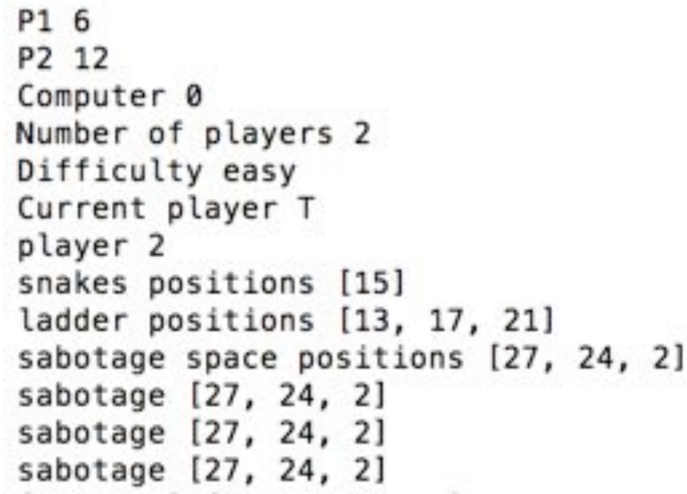


Fig 5.2 final player in game was player 1, so next game started with player 2



```
P1 6
P2 12
Computer 0
Number of players 2
Difficulty easy
Current player T
player 2
snakes positions [15]
ladder positions [13, 17, 21]
sabotage space positions [27, 24, 2]
sabotage [27, 24, 2]
sabotage [27, 24, 2]
sabotage [27, 24, 2]
```

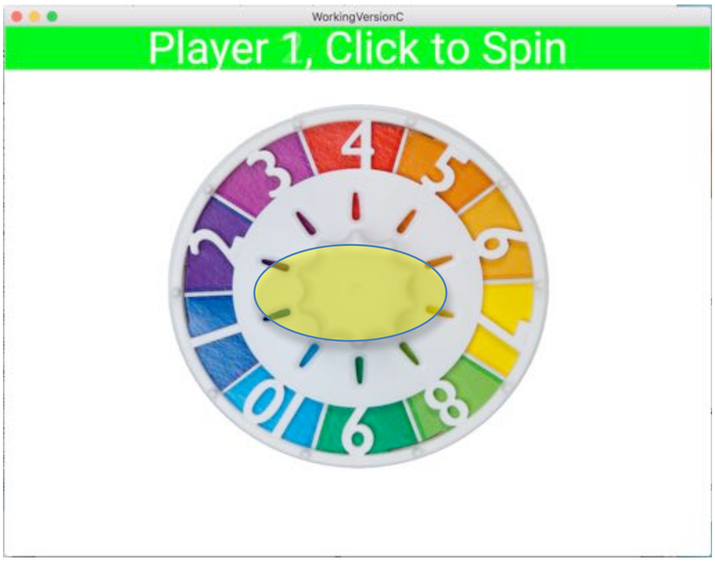
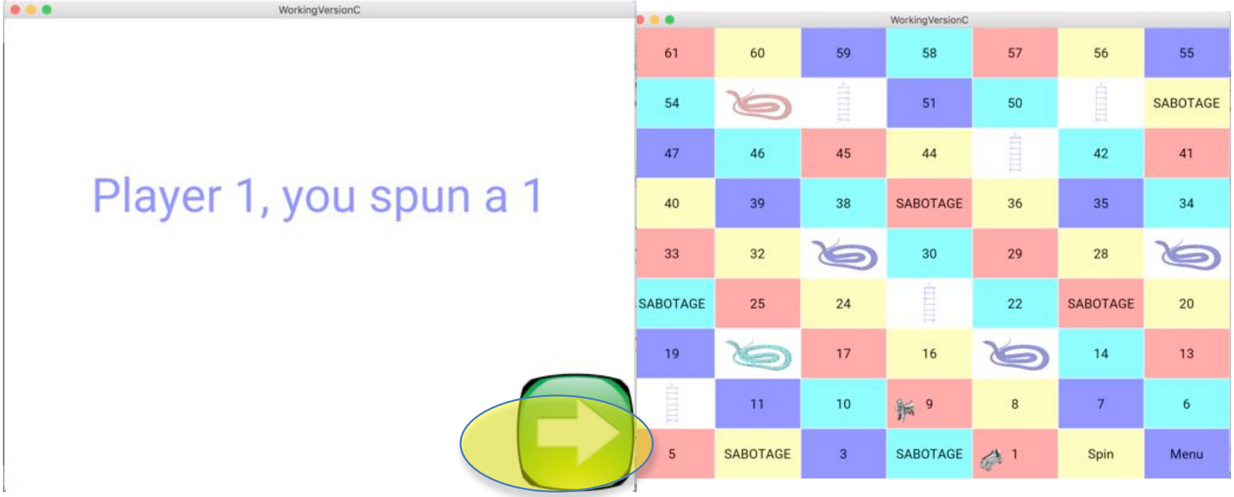


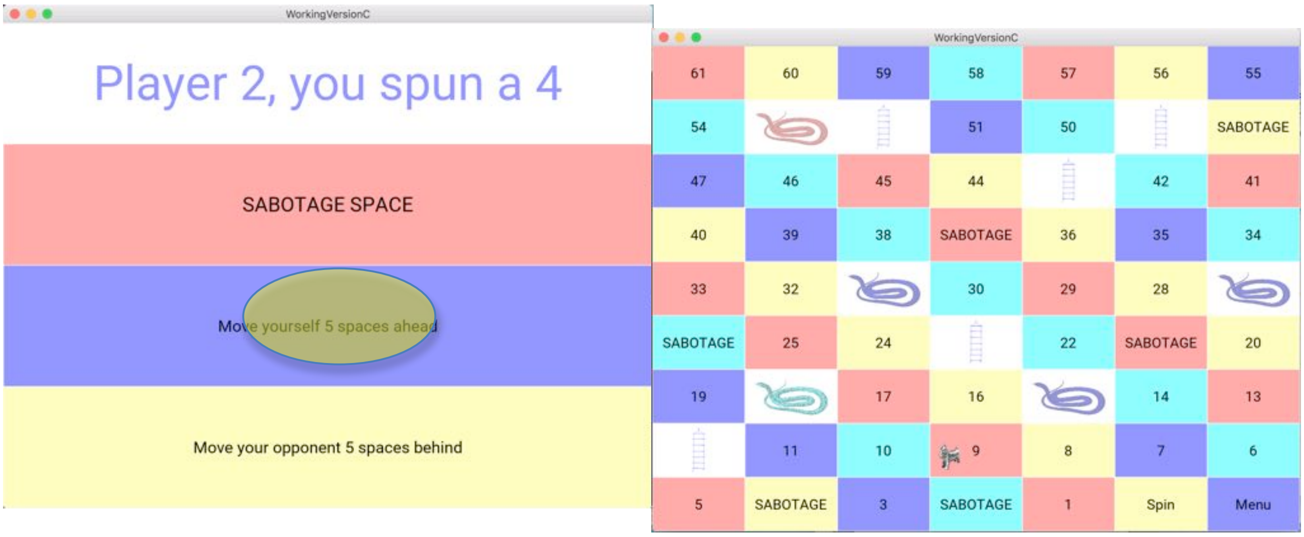
Fig 5.3 Final player in game was player 2, so next game started with player 1



Player 1, you spun a 1

61	60	59	58	57	56	55
54			51	50		SABOTAGE
47	46	45	44		42	41
40	39	38	SABOTAGE	36	35	34
33	32		30	29	28	
SABOTAGE	25	24		22	SABOTAGE	20
19		17	16		14	13
	11	10	9	8	7	6
5	SABOTAGE	3	SABOTAGE	1	Spin	Menu

Fig 6.1 First roll for player 1, display screen says they rolled a 1 and they moved 1 space on the board. Successful.



Player 2, you spun a 4

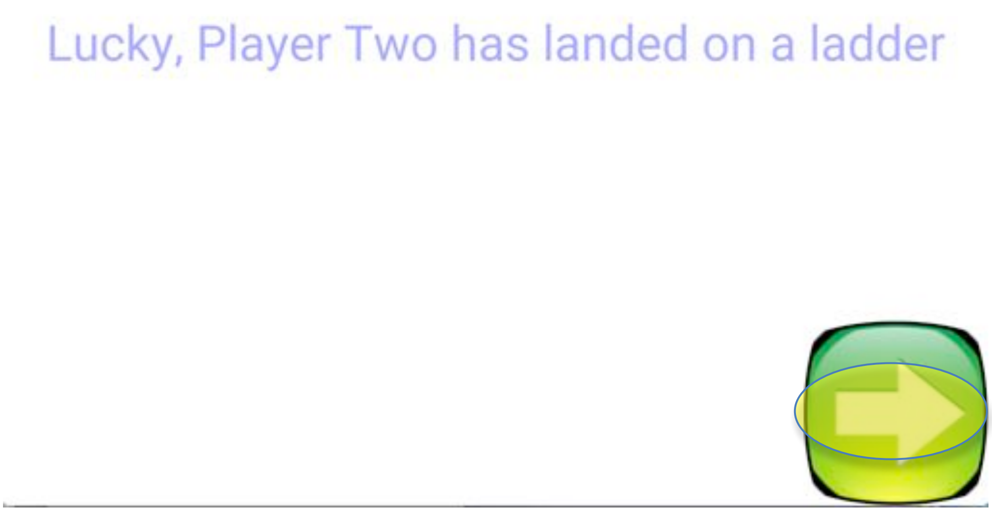
SABOTAGE SPACE

Move yourself 5 spaces ahead

Move your opponent 5 spaces behind

61	60	59	58	57	56	55
54			51	50		SABOTAGE
47	46	45	44		42	41
40	39	38	SABOTAGE	36	35	34
33	32		30	29	28	
SABOTAGE	25	24		22	SABOTAGE	20
19		17	16		14	13
	11	10	9	8	7	6
5	SABOTAGE	3	SABOTAGE	1	Spin	Menu

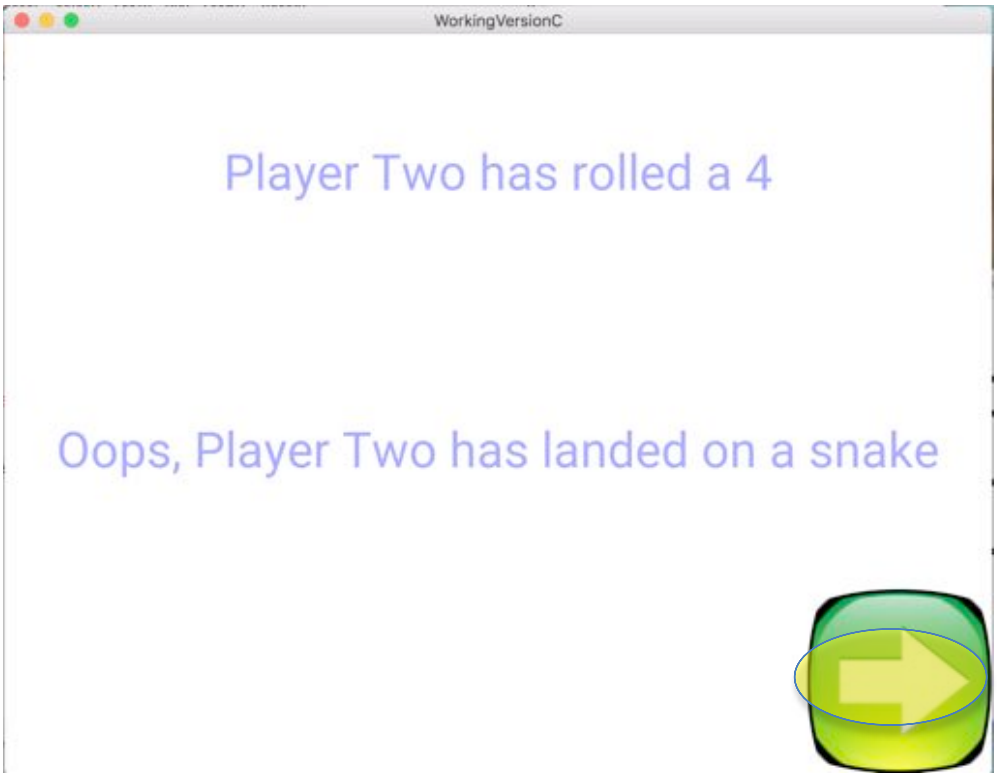
Fig 6.2 First turn for player two, landed on sabotage space and told they rolled a four. Chose to move 5 spaces ahead, player piece is now on space 9. Successful.



Lucky, Player Two has landed on a ladder

Fig 6.3

When player 2 landed on a ladder, the notification screen appeared to tell them what happened, but not what they rolled to land on that space. Unsuccessful

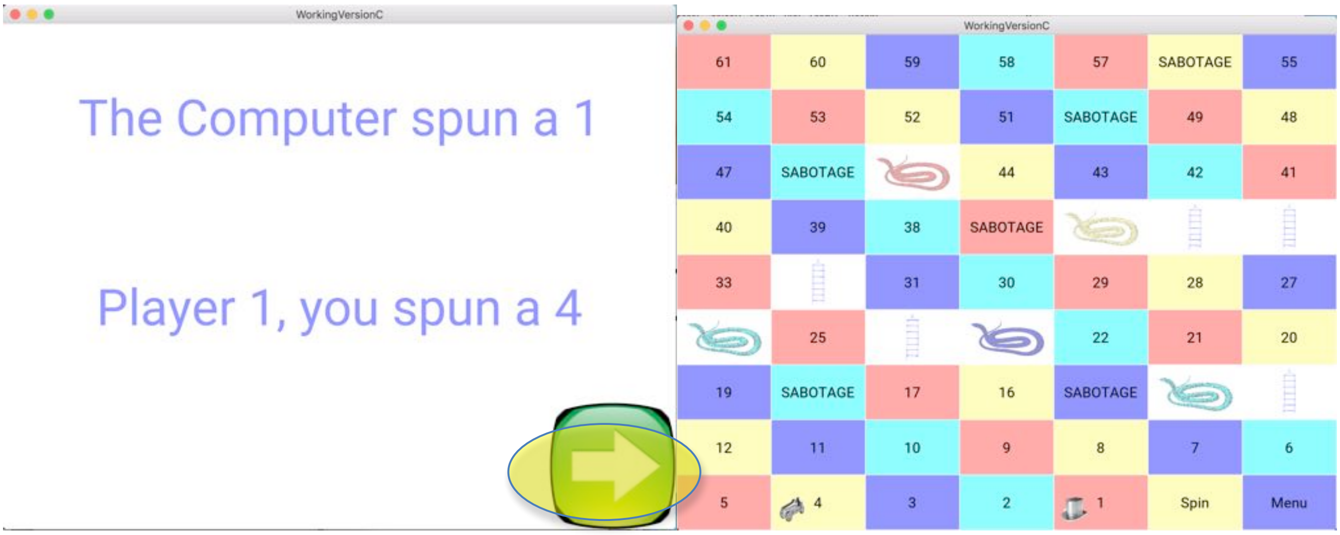


Player Two has rolled a 4

Oops, Player Two has landed on a snake

Fig 6.4

When player 2 landed on a snake, the notification screen appeared to tell them what they rolled and what happened. Successful



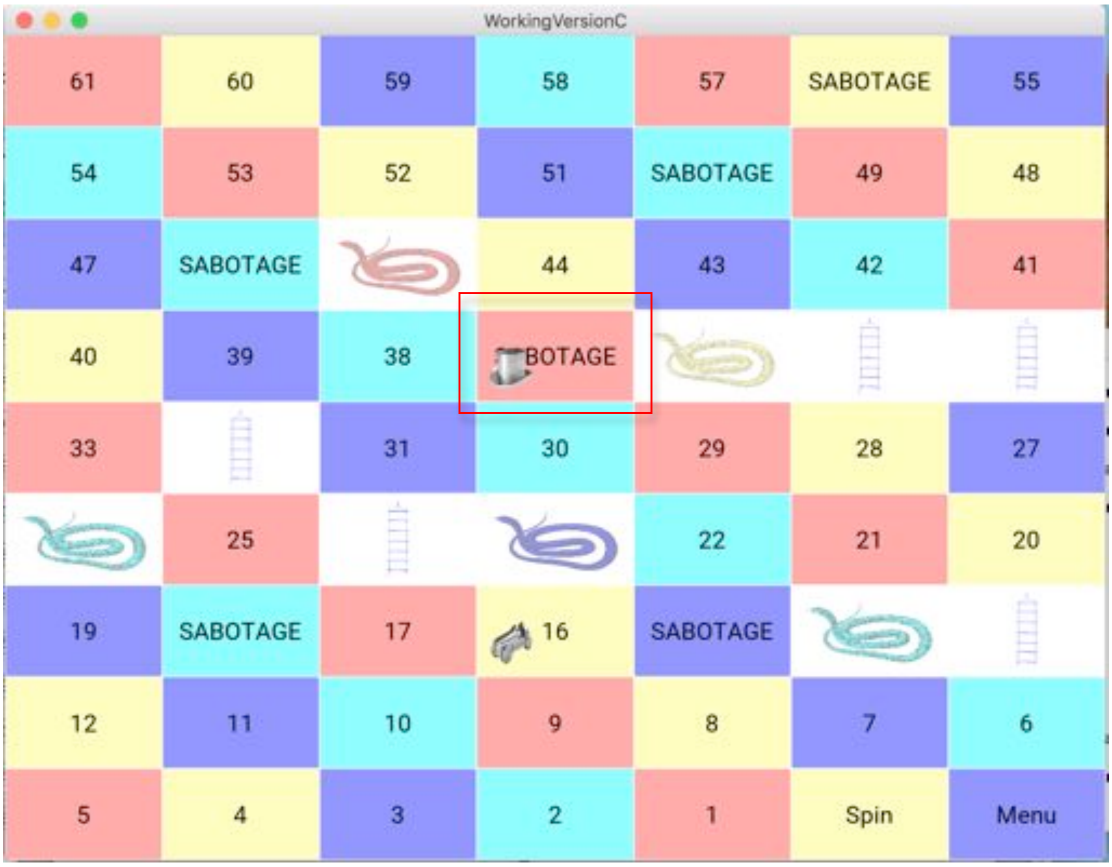
The Computer spun a 1

Player 1, you spun a 4

61	60	59	58	57	SABOTAGE	55
54	53	52	51	SABOTAGE	49	48
47	SABOTAGE		44	43	42	41
40	39	38	SABOTAGE			
33		31	30	29	28	27
	25			22	21	20
19	SABOTAGE	17	16	SABOTAGE		
12	11	10	9	8	7	6
5	4	3	2	1	Spin	Menu

Fig 7.1

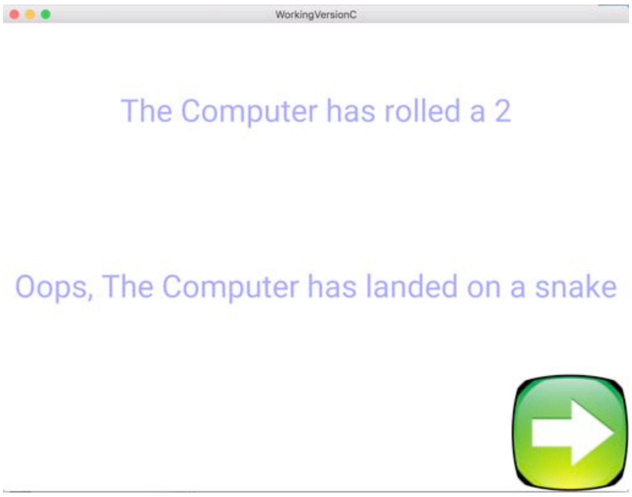
The pieces for player 1 and the computer moved simultaneously, as demonstrated by the positions of the pieces from the start position after 1 spin. Successful.



61	60	59	58	57	SABOTAGE	55
54	53	52	51	SABOTAGE	49	48
47	SABOTAGE		44	43	42	41
40	39	38	BOTAGE			
33		31	30	29	28	27
	25			22	21	20
19	SABOTAGE	17	16	SABOTAGE		
12	11	10	9	8	7	6
5	4	3	2	1	Spin	Menu

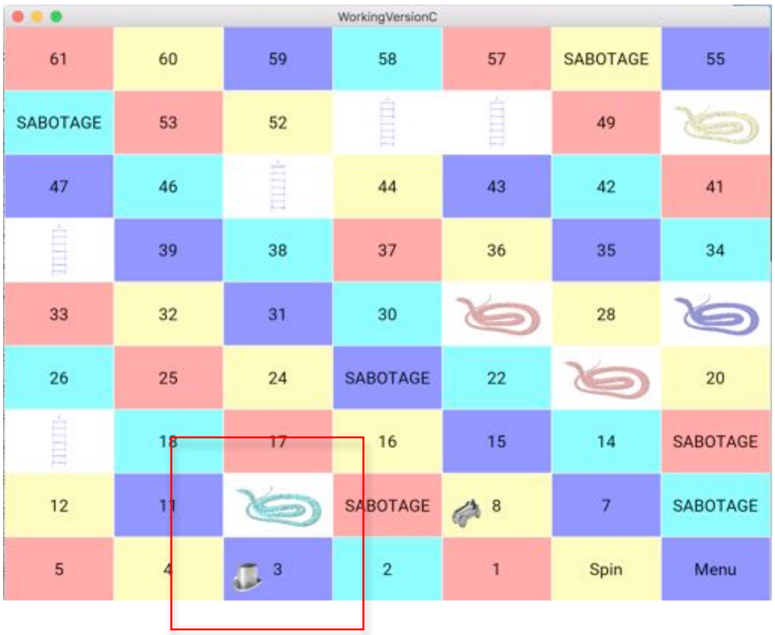
Fig 7.2

The computer landed on a sabotage space but no sabotage screen appeared and the game continued as normal. Successful.



The Computer has rolled a 2

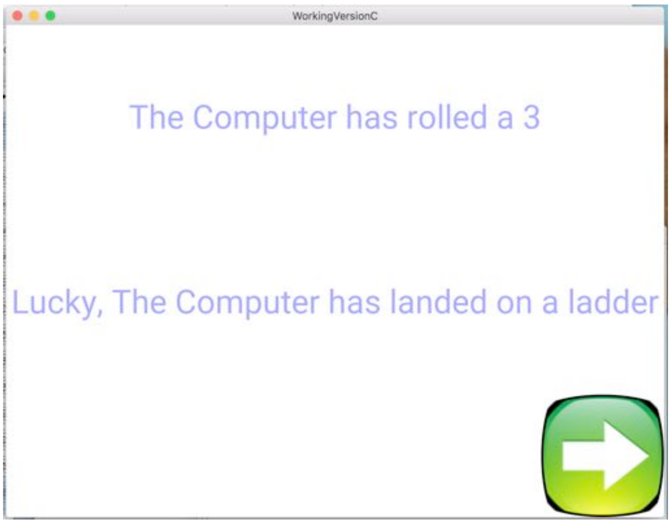
Oops, The Computer has landed on a snake



61	60	59	58	57	SABOTAGE	55
SABOTAGE	53	52			49	
47	46		44	43	42	41
	39	38	37	36	35	34
33	32	31	30		28	
26	25	24	SABOTAGE	22		20
	18	17	16	15	14	SABOTAGE
12	11		SABOTAGE	8	7	SABOTAGE
5	4	3	2	1	Spin	Menu

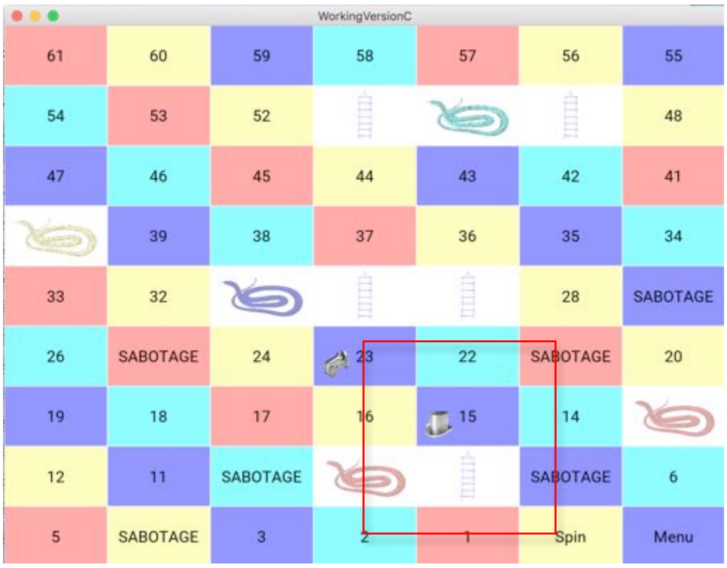
Fig 7.3

The computer landed on a snake and was moved 1 row down, as demonstrated. Successful.



The Computer has rolled a 3

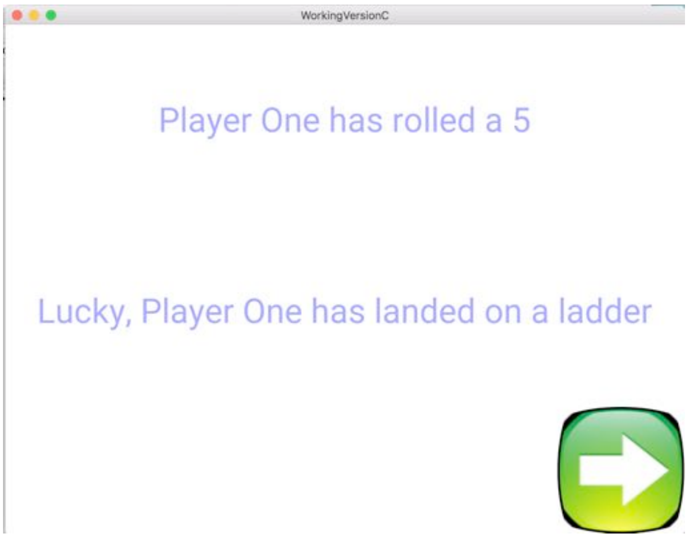
Lucky, The Computer has landed on a ladder



61	60	59	58	57	56	55
54	53	52				48
47	46	45	44	43	42	41
	39	38	37	36	35	34
33	32				28	SABOTAGE
26	SABOTAGE	24	23	22	SABOTAGE	20
19	18	17	16	15	14	
12	11	SABOTAGE			SABOTAGE	6
5	SABOTAGE	3	2	1	Spin	Menu

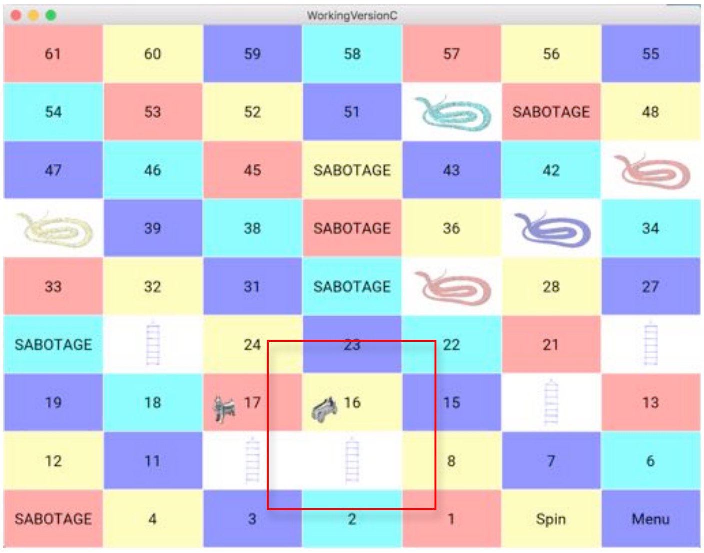
Fig 7.4

The computer landed on a ladder and was moved 1 row up, as demonstrated. Successful.



Player One has rolled a 5

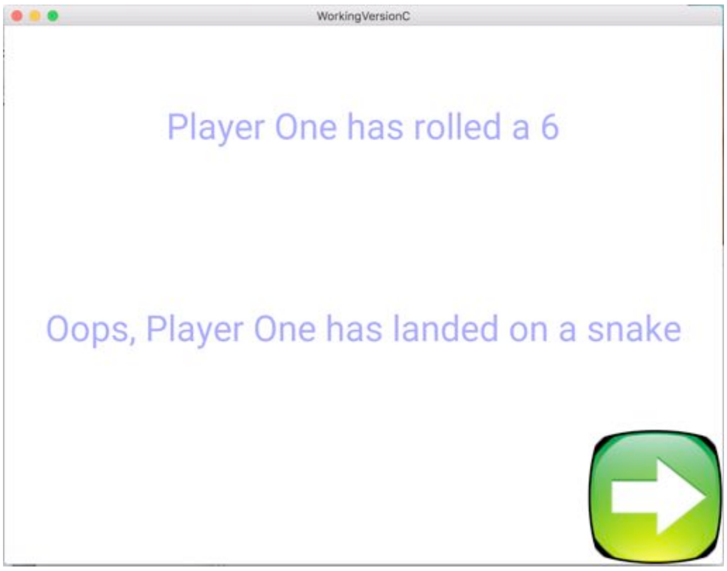
Lucky, Player One has landed on a ladder



61	60	59	58	57	56	55
54	53	52	51		SABOTAGE	48
47	46	45	SABOTAGE	43	42	
	39	38	SABOTAGE	36		34
33	32	31	SABOTAGE		28	27
SABOTAGE		24	23	22	21	
19	18	17	16	15		13
12	11			8	7	6
SABOTAGE	4	3	2	1	Spin	Menu

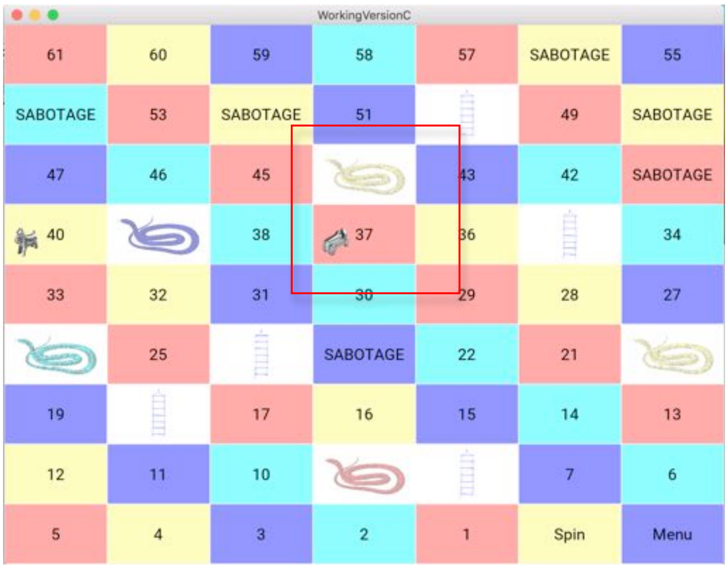
Fig 8.1

Player 1 landed on a ladder and was moved 1 row up, as demonstrated. Successful.



Player One has rolled a 6

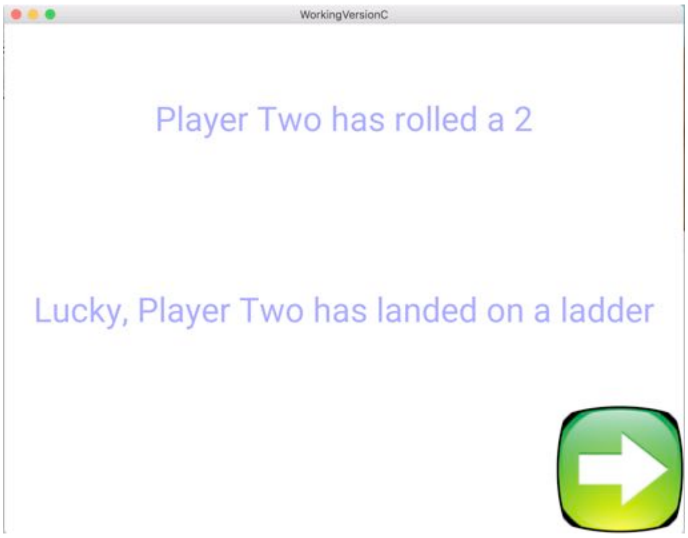
Oops, Player One has landed on a snake



61	60	59	58	57	SABOTAGE	55
SABOTAGE	53	SABOTAGE	51		49	SABOTAGE
47	46	45		43	42	SABOTAGE
40		38	37	36		34
33	32	31	30	29	28	27
	25		SABOTAGE	22	21	
19		17	16	15	14	13
12	11	10			7	6
5	4	3	2	1	Spin	Menu

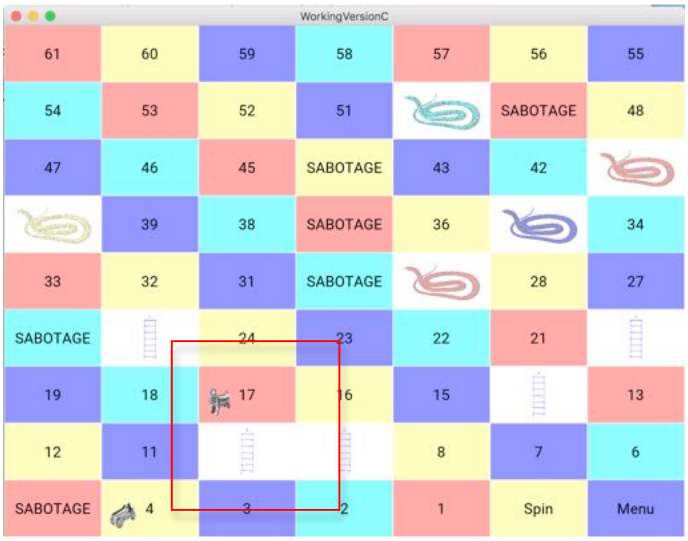
Fig 8.2

Player 1 landed on a snake and was moved 1 row down, as demonstrated. Successful.



Player Two has rolled a 2

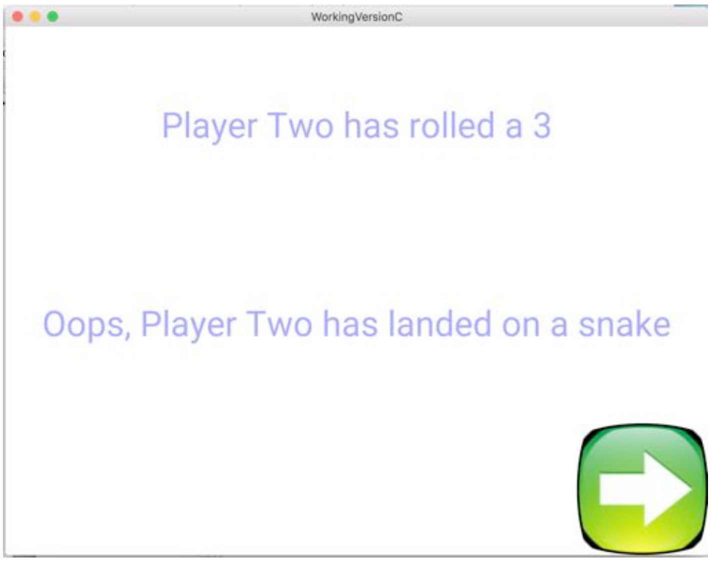
Lucky, Player Two has landed on a ladder



61	60	59	58	57	56	55
54	53	52	51		SABOTAGE	48
47	46	45	SABOTAGE	43	42	
	39	38	SABOTAGE	36		34
33	32	31	SABOTAGE		28	27
SABOTAGE		24	23	22	21	
19	18	17	16	15		13
12	11			8	7	6
SABOTAGE	4	3	2	1	Spin	Menu

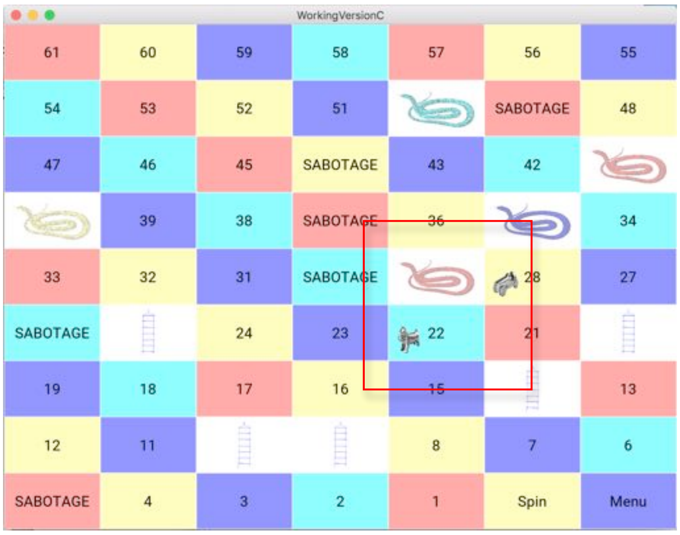
Fig 8.3

Player 2 landed on a ladder and was moved 1 row up, as demonstrated. Successful.



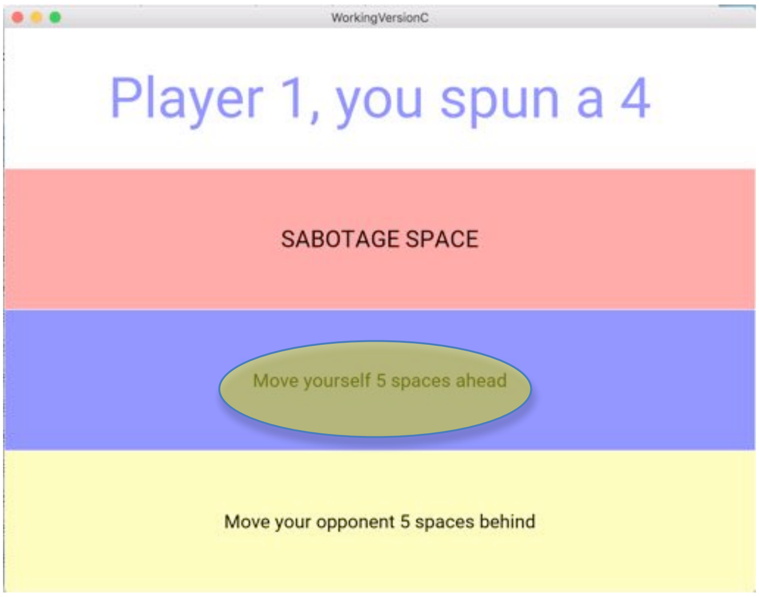
Player Two has rolled a 3

Oops, Player Two has landed on a snake



61	60	59	58	57	56	55
54	53	52	51		SABOTAGE	48
47	46	45	SABOTAGE	43	42	
	39	38	SABOTAGE	36		34
33	32	31	SABOTAGE		28	27
SABOTAGE		24	23	22	21	
19	18	17	16	15		13
12	11			8	7	6
SABOTAGE	4	3	2	1	Spin	Menu

Fig 8.4
Player 2 landed on a snake and was moved 1 row down, as demonstrated. Successful.

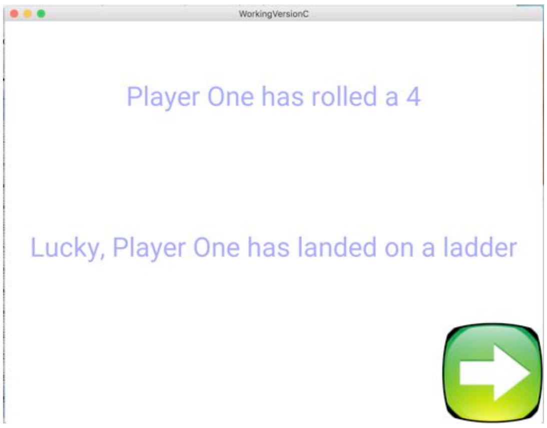


Player 1, you spun a 4

SABOTAGE SPACE

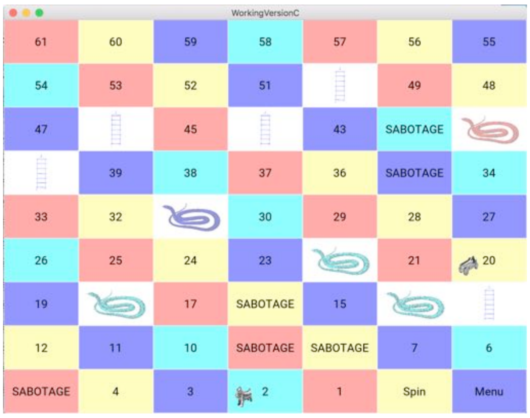
Move yourself 5 spaces ahead

Move your opponent 5 spaces behind



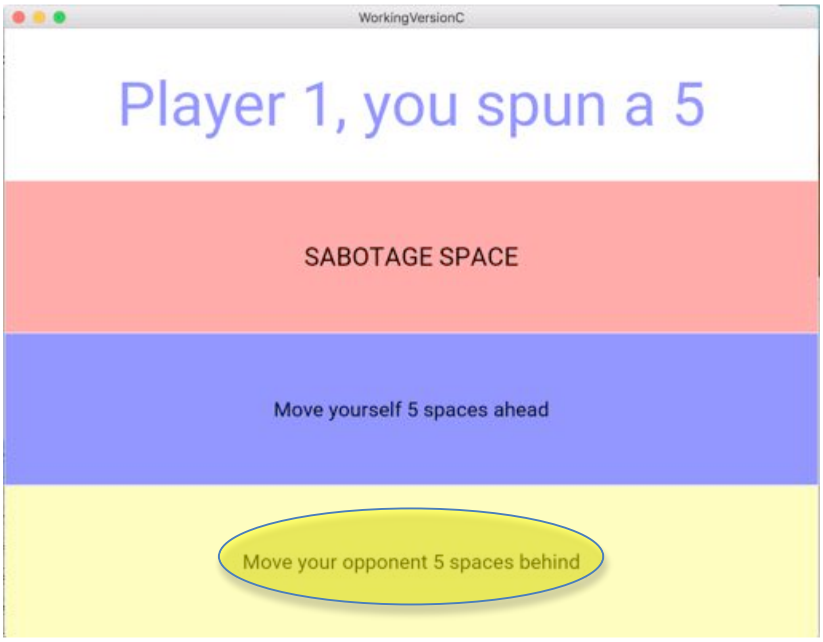
Player One has rolled a 4

Lucky, Player One has landed on a ladder



61	60	59	58	57	56	55
54	53	52	51		49	48
47		45		43	SABOTAGE	
	39	38	37	36	SABOTAGE	34
33	32		30	29	28	27
26	25	24	23		21	20
19		17	SABOTAGE	15		
12	11	10	SABOTAGE	SABOTAGE	7	6
SABOTAGE	4	3	2	1	Spin	Menu

Fig 9.1 Player 1 spun a 4 from space 4, which landed them on space 8. They ten chose to move themselves five spaces forward, which landed them on a ladder, which moved them one row up, which is why they are now on space 20. Successful, plus tested functionality of obstacles appearing after each other, which was also successful.

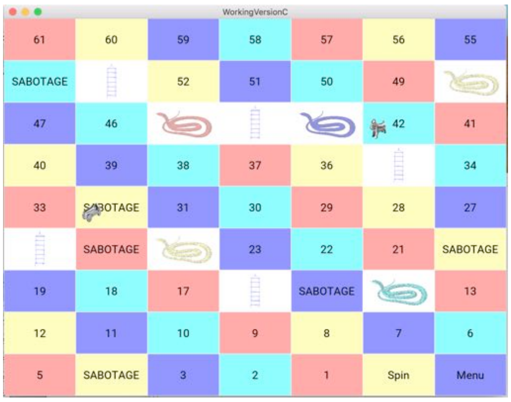


Player 1, you spun a 5

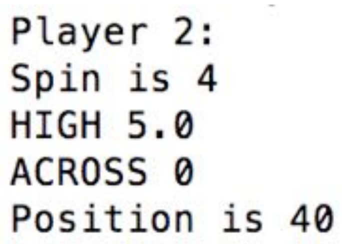
SABOTAGE SPACE

Move yourself 5 spaces ahead

Move your opponent 5 spaces behind

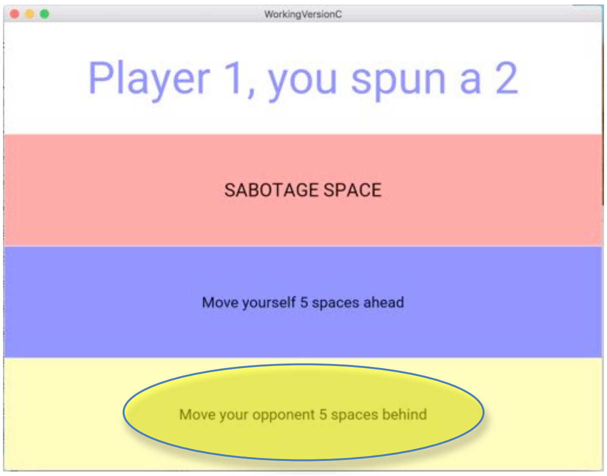


61	60	59	58	57	56	55
SABOTAGE		52	51	50	49	
47	46				42	41
40	39	38	37	36		34
33	SABOTAGE	31	30	29	28	27
	SABOTAGE		23	22	21	SABOTAGE
19	18	17		SABOTAGE		13
12	11	10	9	8	7	6
5	SABOTAGE	3	2	1	Spin	Menu



```
Player 2:
Spin is 4
HIGH 5.0
ACROSS 0
Position is 40
```

Fig 9.2 Player 1 landed on the sabotage space at position 32, while player 2 was at position 40. Player 1 chose to send P2 5 spaces behind, which landed them on space 35, which was a ladder, so P2 ended up on space 42, one row above. Fully successful.

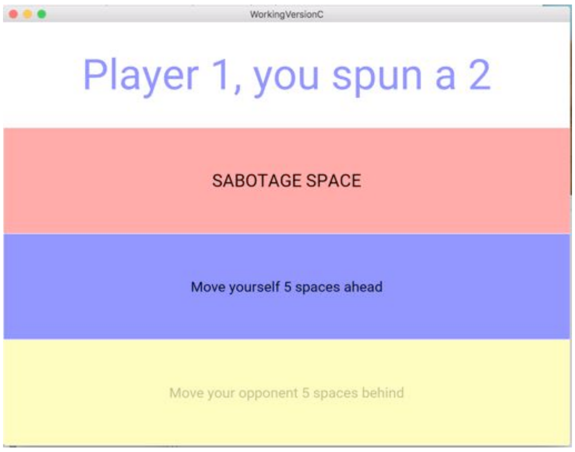


Player 1, you spun a 2

SABOTAGE SPACE

Move yourself 5 spaces ahead

Move your opponent 5 spaces behind

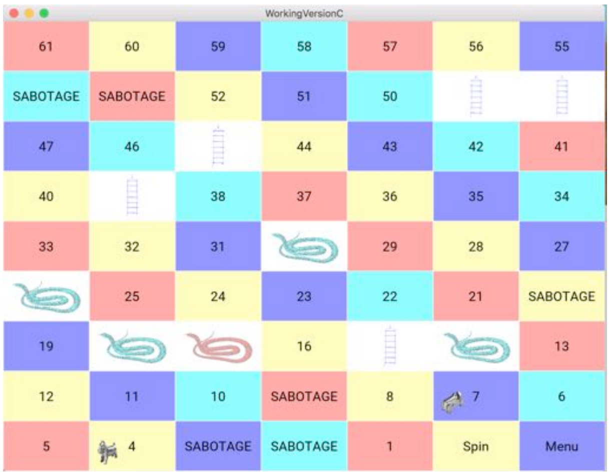


Player 1, you spun a 2

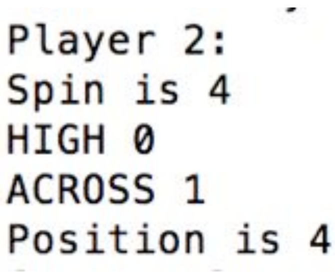
SABOTAGE SPACE

Move yourself 5 spaces ahead

Move your opponent 5 spaces behind

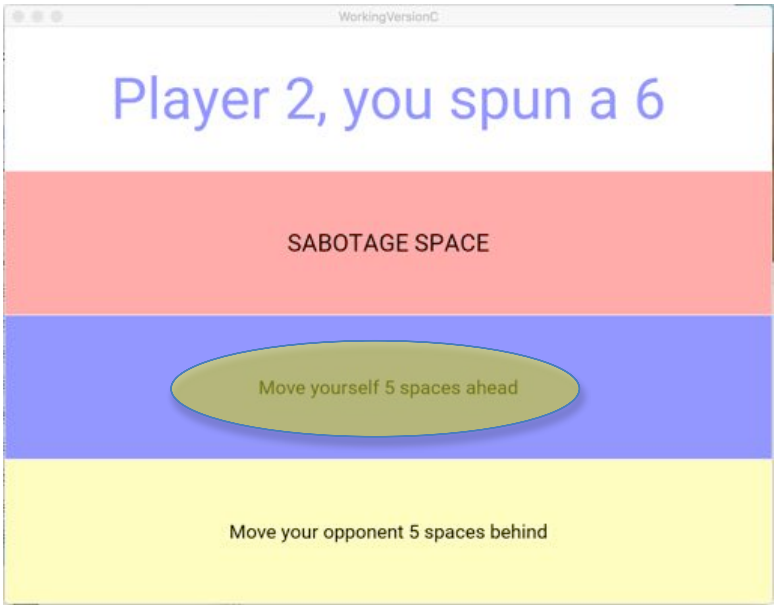


61	60	59	58	57	56	55
SABOTAGE	SABOTAGE	52	51	50		
47	46		44	43	42	41
40		38	37	36	35	34
33	32	31		29	28	27
	25	24	23	22	21	SABOTAGE
19			16			13
12	11	10	SABOTAGE	8	7	6
5	4	SABOTAGE	SABOTAGE	1	Spin	Menu



```
Player 2:
Spin is 4
HIGH 0
ACROSS 1
Position is 4
```

Fig 9.3 Player 1 landed on the sabotage space at position 2, while P2 was at position 4. P1 tried to move P2 5 spaces behind but could not because P2 was not 5 spaces into the board. P1 then chose to move 5 spaces ahead, landing them on space 7. Fully successful.

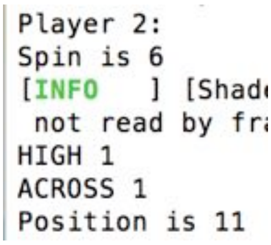


Player 2, you spun a 6

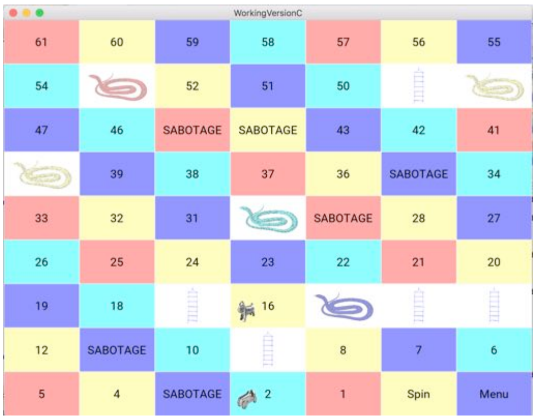
SABOTAGE SPACE

Move yourself 5 spaces ahead

Move your opponent 5 spaces behind

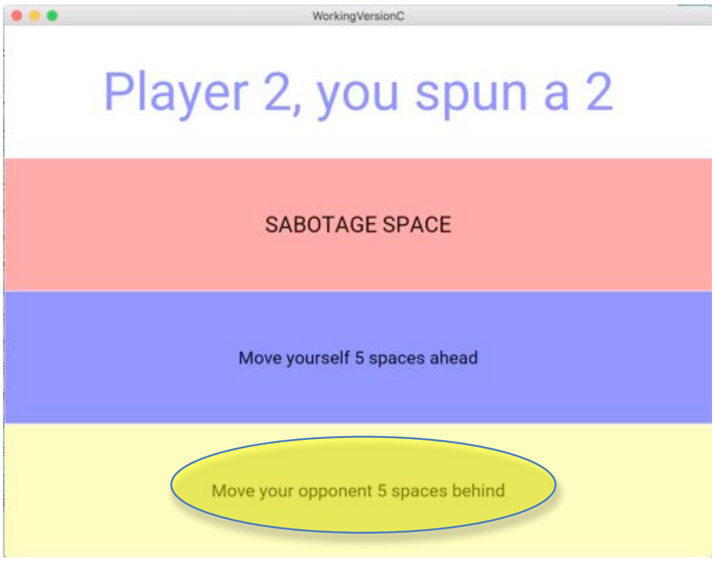


```
Player 2:
Spin is 6
[INFO   ] [Shad
 not read by fr
HIGH 1
ACROSS 1
Position is 11
```



61	60	59	58	57	56	55
54		52	51	50		
47	46	SABOTAGE	SABOTAGE	43	42	41
	39	38	37	36	SABOTAGE	34
33	32	31		SABOTAGE	28	27
26	25	24	23	22	21	20
19	18		16			
12	SABOTAGE	10		8	7	6
5	4	SABOTAGE	2	1	Spin	Menu

Fig 9.4 Player spun a six and land on a sabotage space at 11, chose to move five spaces ahead and now appears on space 16. Successful.



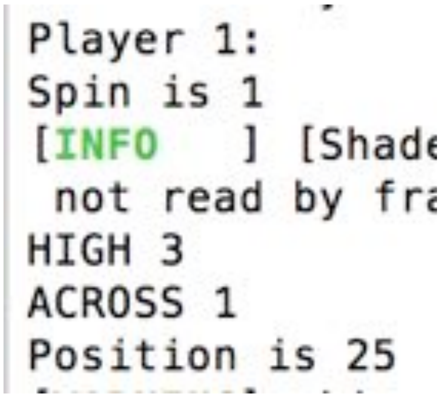
WorkingVersionC

Player 2, you spun a 2

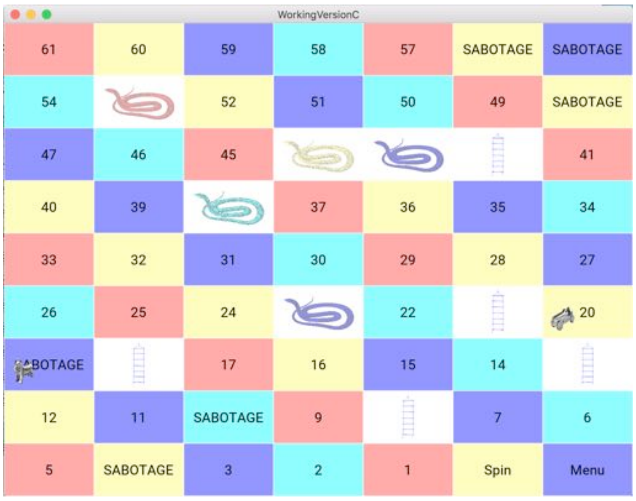
SABOTAGE SPACE

Move yourself 5 spaces ahead

Move your opponent 5 spaces behind



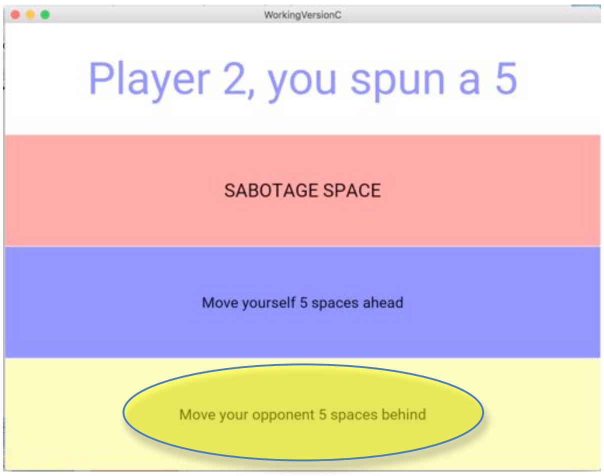
```
Player 1:
Spin is 1
[INFO    ] [Shade
 not read by fra
HIGH 3
ACROSS 1
Position is 25
```



WorkingVersionC

61	60	59	58	57	SABOTAGE	SABOTAGE
54		52	51	50	49	SABOTAGE
47	46	45				41
40	39		37	36	35	34
33	32	31	30	29	28	27
26	25	24		22		20
BOTAGE		17	16	15	14	
12	11	SABOTAGE	9		7	6
5	SABOTAGE	3	2	1	Spin	Menu

Fig 9.5 Player 2 landed on the sabotage space on space 19, and player 1 was on space 25. Player 2 chose to move player 1 back five, and player one's piece is now on space 20. Successful.



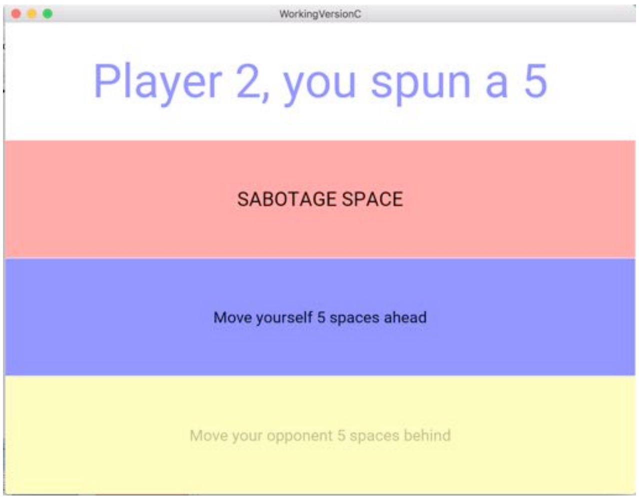
WorkingVersionC

Player 2, you spun a 5

SABOTAGE SPACE

Move yourself 5 spaces ahead

Move your opponent 5 spaces behind



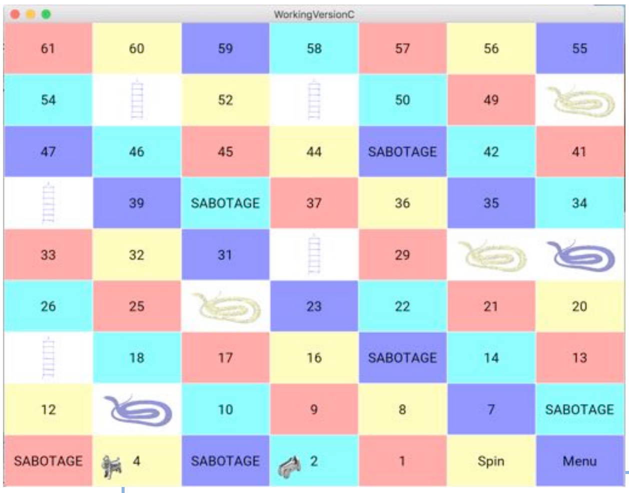
WorkingVersionC

Player 2, you spun a 5

SABOTAGE SPACE

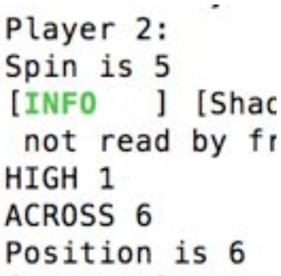
Move yourself 5 spaces ahead

Move your opponent 5 spaces behind



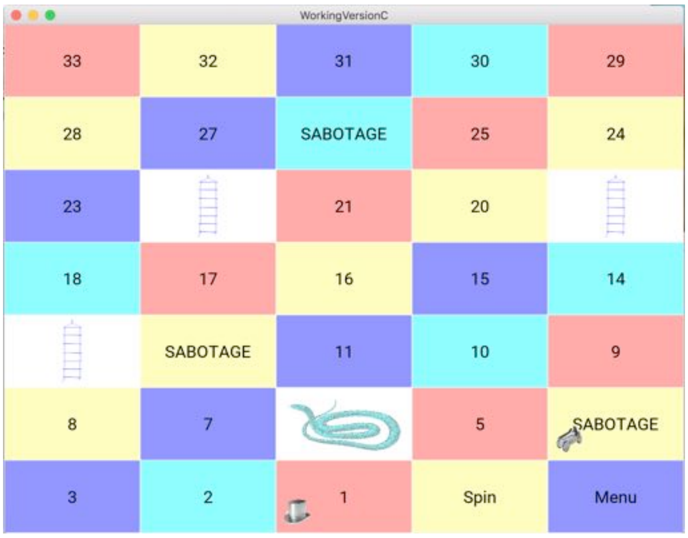
WorkingVersionC

61	60	59	58	57	56	55
54		52		50	49	
47	46	45	44	SABOTAGE	42	41
	39	SABOTAGE	37	36	35	34
33	32	31		29		
26	25		23	22	21	20
	18	17	16	SABOTAGE	14	13
12		10	9	8	7	SABOTAGE
SABOTAGE	4	SABOTAGE	2	1	Spin	Menu

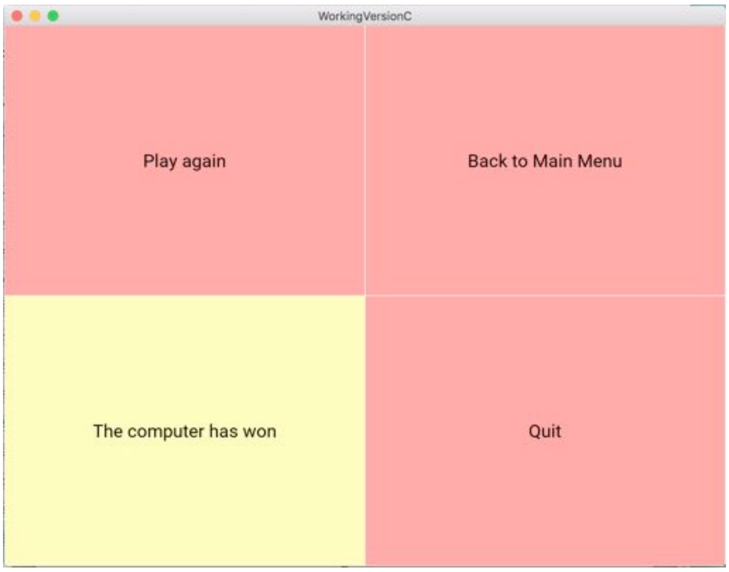


```
Player 2:
Spin is 5
[INFO    ] [Shac
 not read by fr
HIGH 1
ACROSS 6
Position is 6
```

Fig 9.6 Player 2 landed on the sabotage space at position 6, while player 1 was at position 2. Player 2 tried to send P1 5 spaces but the option was greyed out as they were not 5 spaces into the game. Instead, P2 moved 5 spaces ahead onto space 11, which was a snake, so P2 ended up on space 4, one row below. Fully successful.

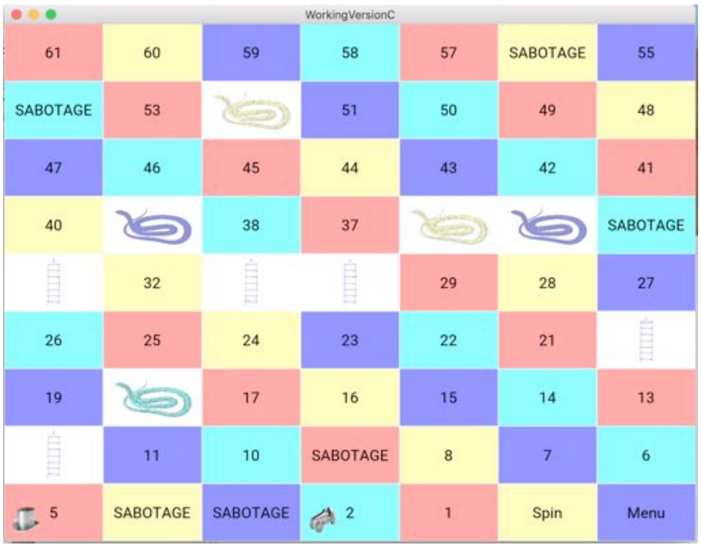


33	32	31	30	29
28	27	SABOTAGE	25	24
23		21	20	
18	17	16	15	14
	SABOTAGE	11	10	9
8	7		5	SABOTAGE
3	2	1	Spin	Menu

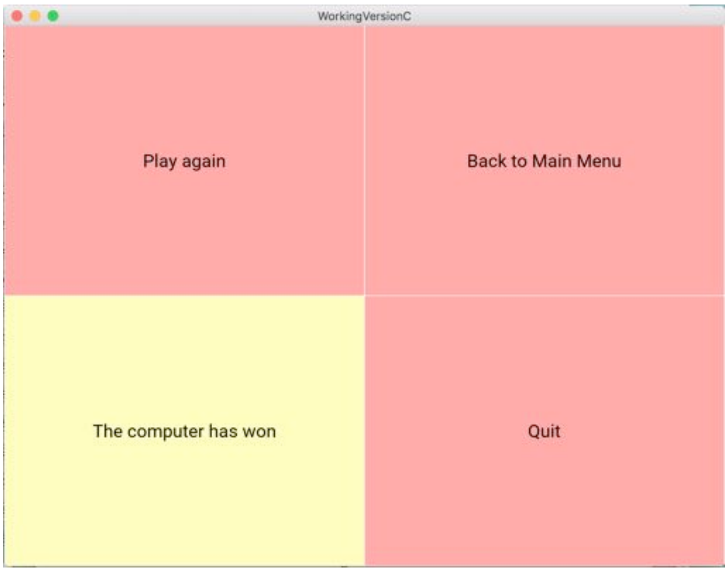


The computer has won

Fig 10.1 single player in easy

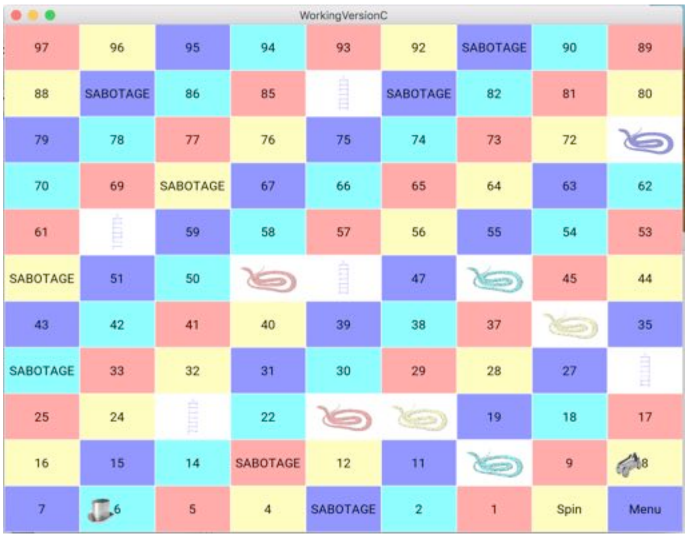


61	60	59	58	57	SABOTAGE	55
SABOTAGE	53		51	50	49	48
47	46	45	44	43	42	41
40		38	37			SABOTAGE
	32			29	28	27
26	25	24	23	22	21	
19		17	16	15	14	13
	11	10	SABOTAGE	8	7	6
5	SABOTAGE	SABOTAGE	2	1	Spin	Menu

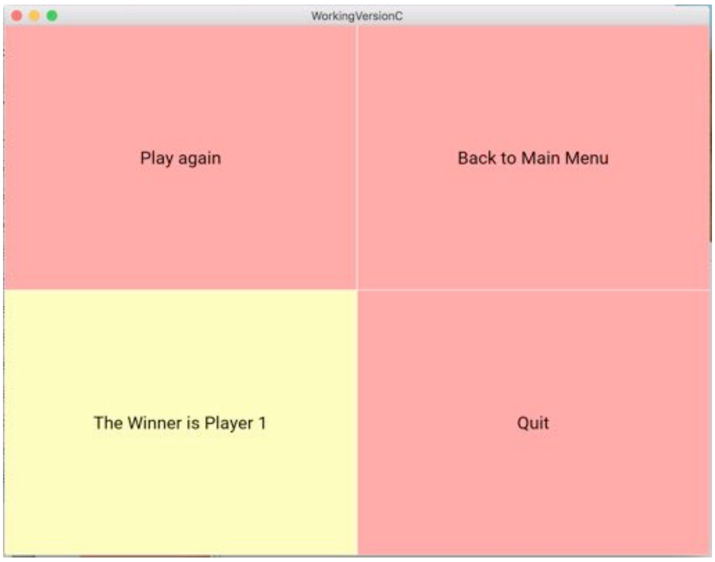


The computer has won

Fig 10.2 single player in intermediate, computer was able to win. Successful.

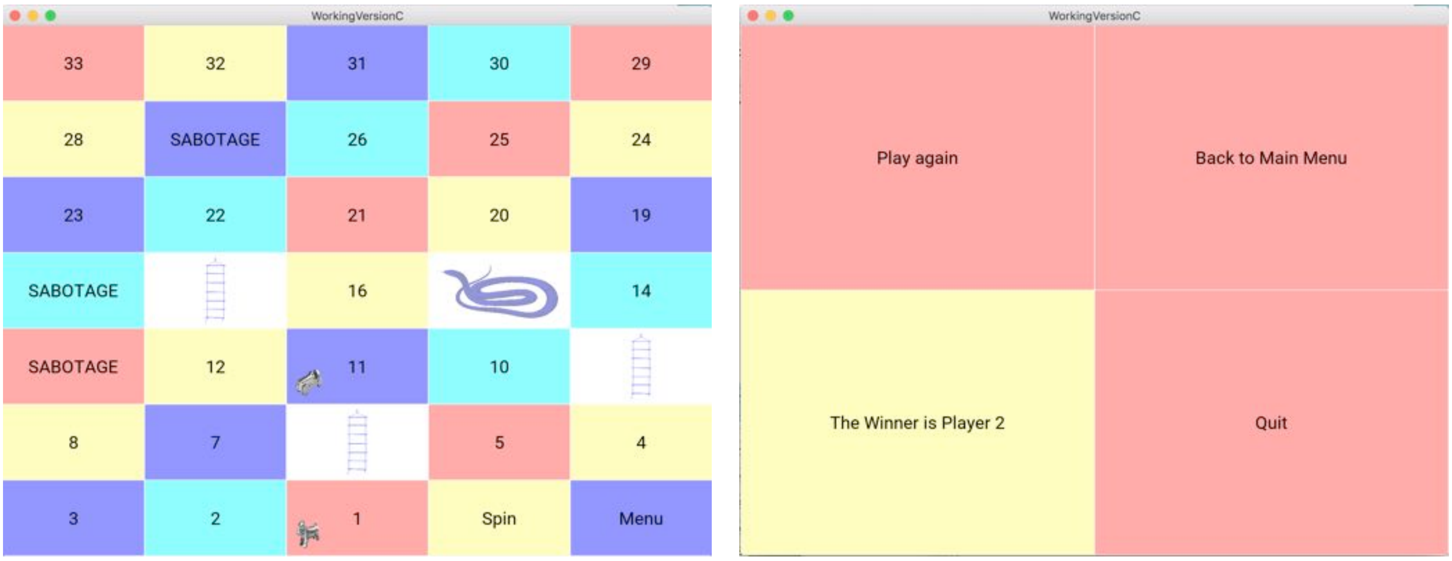


97	96	95	94	93	92	SABOTAGE	90	89
88	SABOTAGE	86	85		SABOTAGE	82	81	80
79	78	77	76	75	74	73	72	
70	69	SABOTAGE	67	66	65	64	63	62
61		59	58	57	56	55	54	53
SABOTAGE	51	50			47		45	44
43	42	41	40	39	38	37		35
SABOTAGE	33	32	31	30	29	28	27	
25	24		22			19	18	17
16	15	14	SABOTAGE	12	11		9	8
7	6	5	4	SABOTAGE	2	1	Spin	Menu



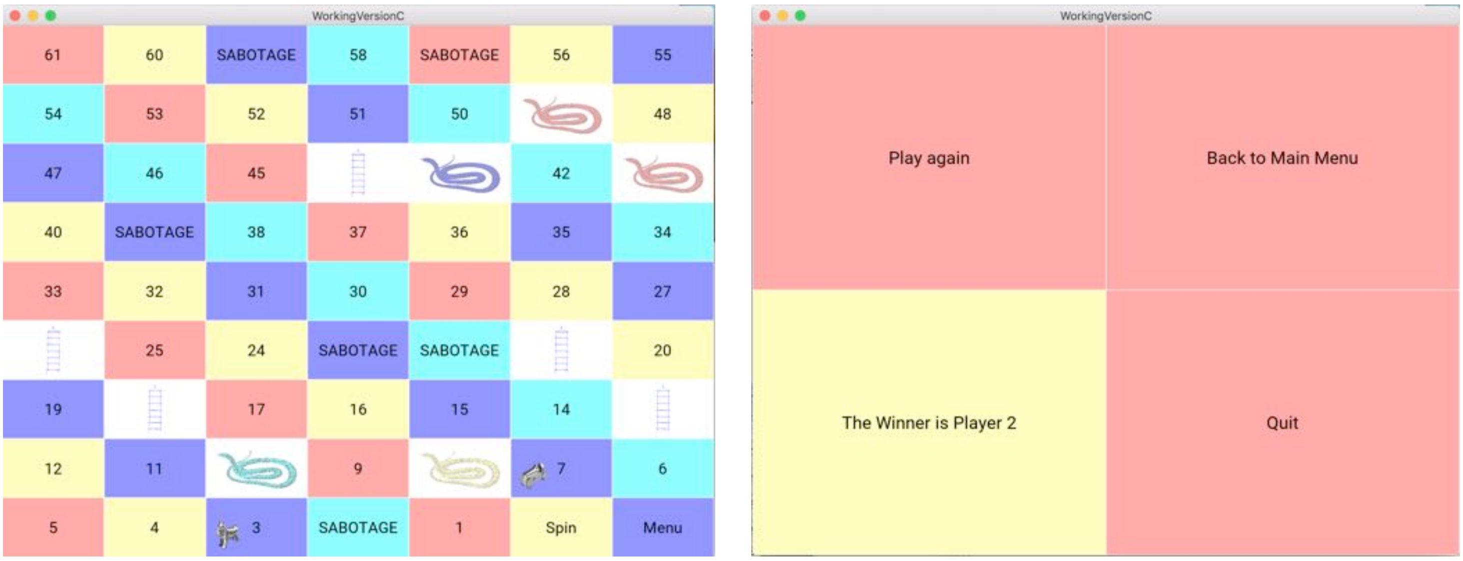
The Winner is Player 1

Fig 10.3 single player in hard, Player 1 was able to win. Successful.



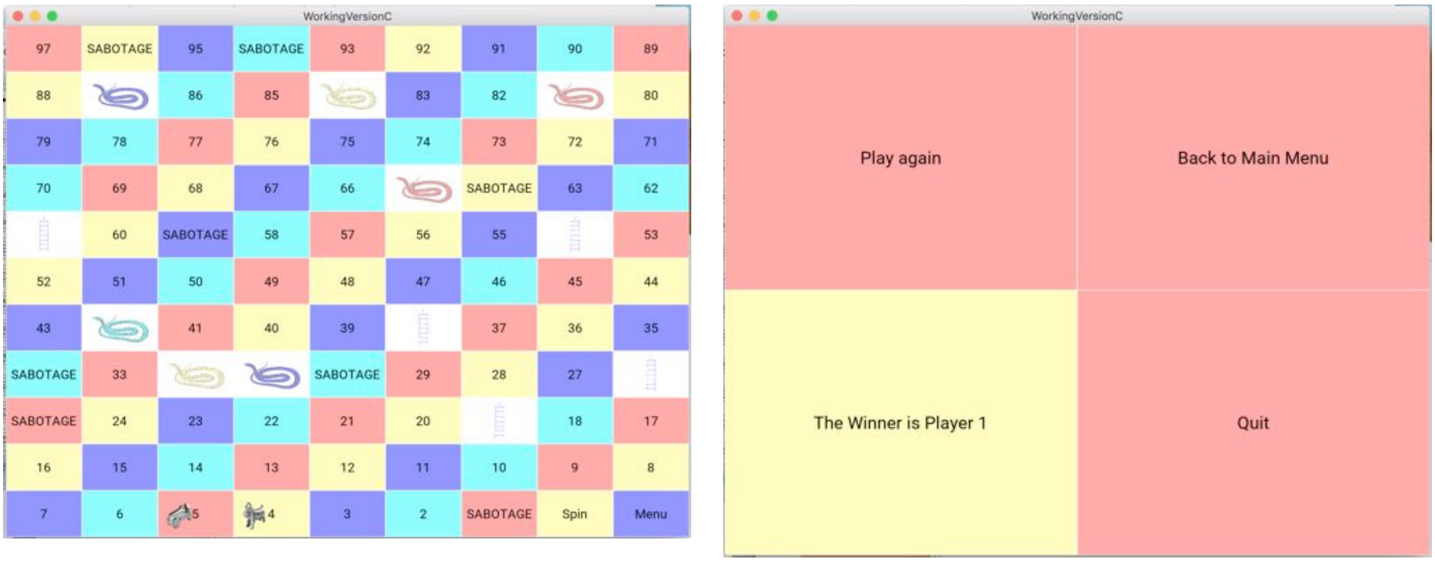
The Winner is Player 2

Fig 10.4 2 player in easy, Player 2 was able to win. Successful.



The Winner is Player 2

Fig 10.5 2 player in intermediate, Player 2 was able to win. Successful.


WorkingVersionC
97 SABOTAGE 95 SABOTAGE 93 92 91 90 89
88 86 85 83 82 80
79 78 77 76 75 74 73 72 71
70 69 68 67 66 SABOTAGE 63 62
60 SABOTAGE 58 57 56 55 53
52 51 50 49 48 47 46 45 44
43 41 40 39 37 36 35
SABOTAGE 33 SABOTAGE 29 28 27
SABOTAGE 24 23 22 21 20 18 17
16 15 14 13 12 11 10 9 8
7 6 5 4 3 2 SABOTAGE Spin Menu
WorkingVersionC
Play again
Back to Main Menu
Quit

The Winner is Player 1

Fig 10.6 2 player in hard, Player 1 was able to win. Successful.

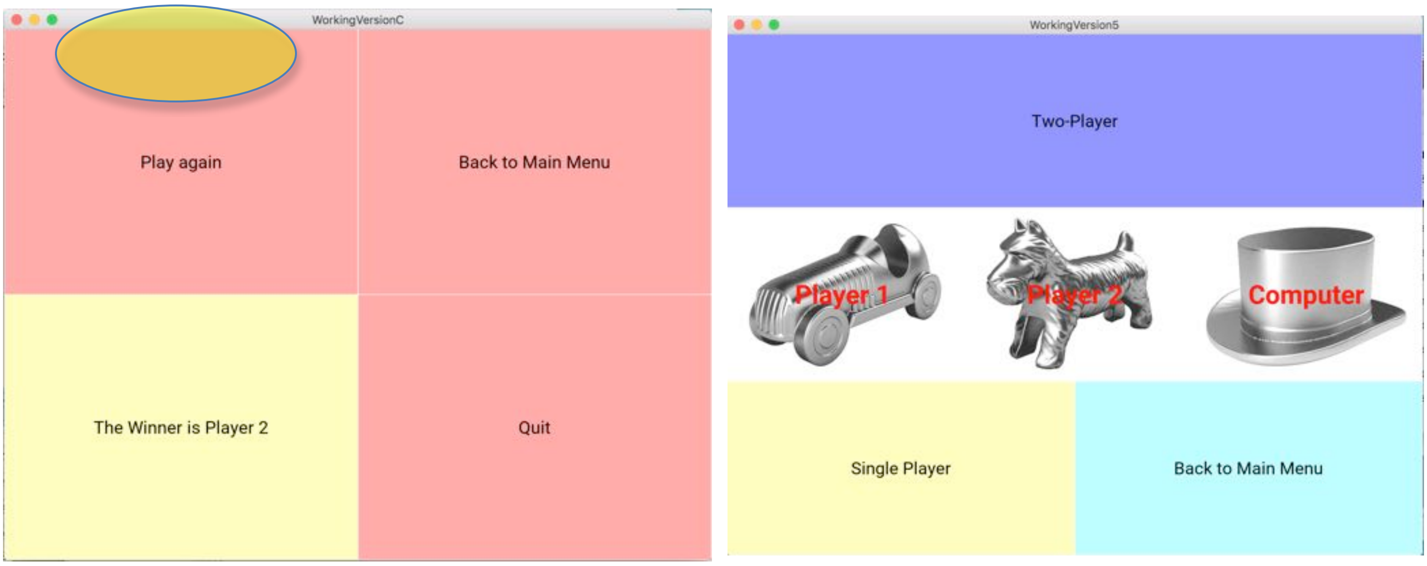


Fig 10.7 Transitioned from winner screen to game options screen. Successful.

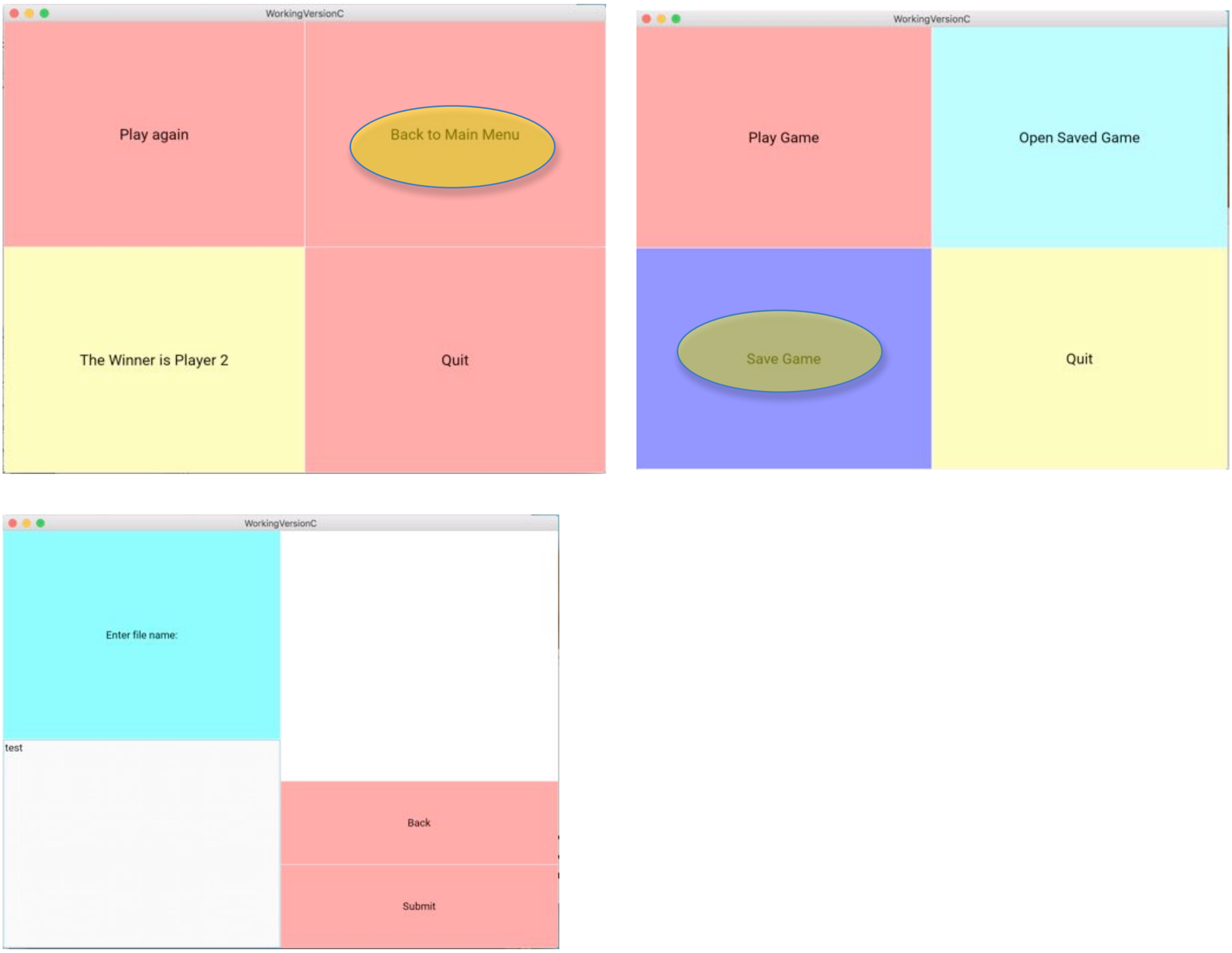


Fig 10.8 Transitioned from winner screen to main menu screen, but game can be saved when it shouldn't be able to- game is completed.

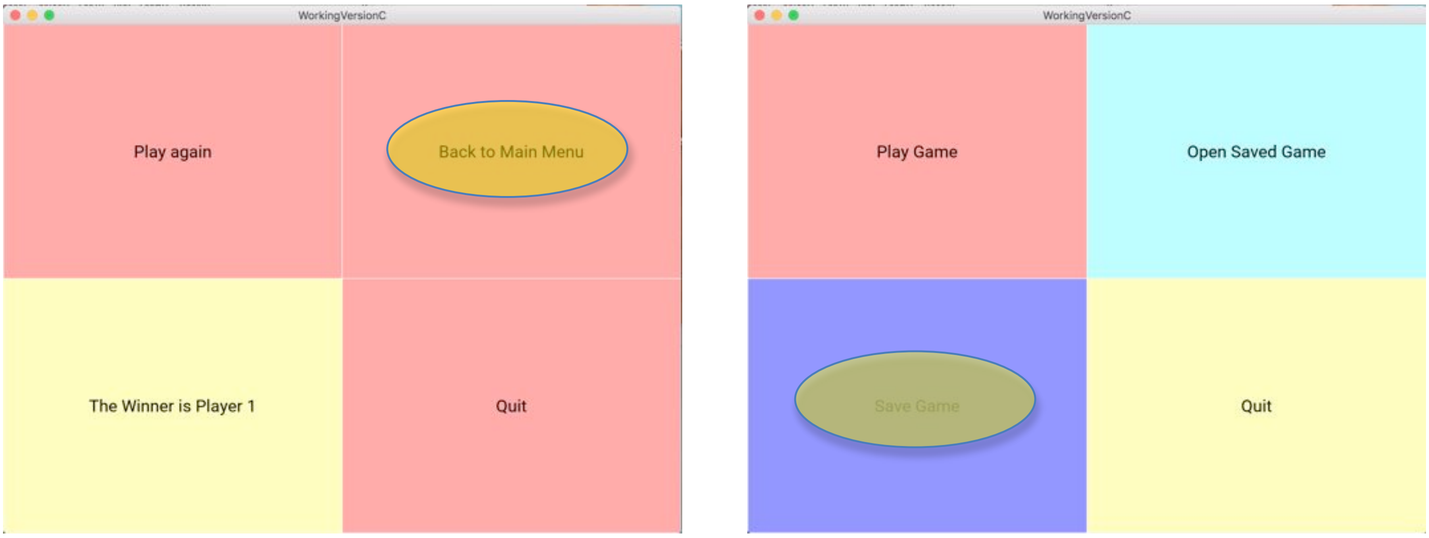


Fig 10.9 Once a game is won, ‘save game’ can not be selected and the menu screen stays until another option is selected. Successful.

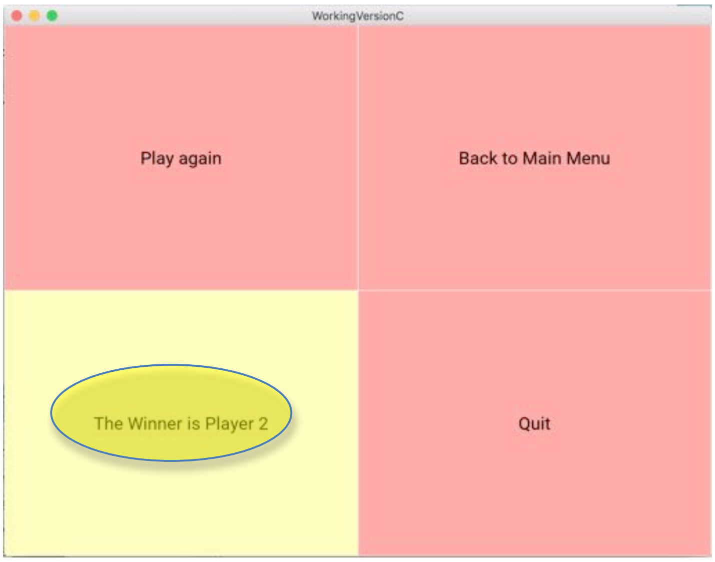
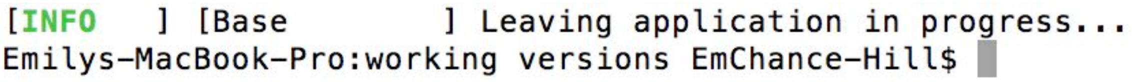


Fig 10.10 Window shut down when winner screen option ‘Quit’ was clicked. Successful.



```
[INFO   ] [Base        ] Leaving application in progress...
Emilys-MacBook-Pro:working versions EmChance-Hill$ 
```

NEA Evaluation

Evaluation Vs. Objectives

On a whole, I feel like I have met and in some places surpassed the **objectives** I set for myself in this project. I have created a functioning game with a colourful and interactive board with randomly generated **obstacles (see test 2.1) such as snakes (see tests 7.3, 8.2, 8.4), ladders (see tests 7.4, 8.1, 8.3)** and **sabotage spaces (see tests 7.2, 9.1, 9.2, 9.3, 9.4, 9.5, 9.6)** that can affect a player's piece. I have made a **menu screen (see tests 1.1 through 1.5)** and a **game board (see tests 1.6, 1.7, 1.8, 2.2)**, which all **players can win on (see tests 10.1 through 10.6)**. I have surpassed my objectives by making separate screens for the game, spinner, menus, winner screen, save and load screens, and display screens (see tests 6.1 through 6.3, 10.7 through 10.9), which all link to each other and back to the main menu so the app is fully traversable.

The **spinner (see test 2.2)** is graphical, colourful and interactive so that the players can click on it to produce the effect of generating the spin themselves, instead of it being randomly generated in the code, which it is. The **different players have fun and individual pieces (see test 2.2, 1.4)**, which are shown in one of the menu screens so that the players can see which belongs to whom. The **save (see tests 3.1 through 3.5) and load game (see tests 4.1 through 4.4, 5.1 through 5.3) functionality works well** and consistently, and the users can save an unlimited amount of games and can access them all whenever they please.

The **single player mode (see tests 7.1 through 7.4)** also works sufficiently, and the computer acts like a manual player in every way except for making decisions, such as on sabotage screen. I have managed to include all of my **additional gameplay features** including the sabotage spaces (see tests 7.2, 9.1, 9.2, 9.3, 9.4, 9.5, 9.6), where a player can choose to either progress their own piece or sabotage their opponent, the different difficulty levels (see tests 1.6, 1.7, 1.8) which affects the size of the board as well as the total number of obstacles and their proportions to each other.

Feedback From Users

Just after the analysis stage of my project, I produced a basic questionnaire (see questionnaire doc) asking for advice about the gameplay of my app, which I distributed to about ten people. My main objective for this exercise was to find out the general opinion of whether I should keep my game traditional or mix it up into a new kind of game.

For this, I asked people to play 2 of the games that I found in my research, one of which was a very basic online game of snakes and ladders where there were no real additional features, and the other an app loosely based around snakes and ladders which had extra features such as a slot machine where you had to get a certain amount of images matched up before you could take a turn. I then proposed three questions; which did you enjoy more, which was more intuitive to play and which felt more like snakes and ladders.

The results were about 60/40 for enjoying the game in favour of the slots game, but every single person said the basic game was easier and more intuitive to play, as well as feeling more like snakes and ladders. Since my game was to be aimed at children and families, the more intuitive the better, so I knew I had to make my game simpler than the slots game, with less radical changes. I also knew I wanted the game to be somewhat traditional, so being recognisable as a snakes and ladders game was imperative.

Based on this, and considering that how much the games were enjoyed was nearly equal, I decided to keep my game to a more traditional style, keeping the basic game and only adding features on top, rather than changing anything vital in the gameplay. I feel like I have done this, as the basic game is very essentially snakes and ladders, only with extra board options and sabotage spaces added in, which I like to think only add to the quality of gameplay. I have kept some features as close as I can to what playing the real life game would be like, such as having to manually make your own spin, and decide from the display screen when they want to move their piece and continue with the game.

To check that I had implemented the different features of my game in a user-friendly way, consistent with the style of game suggested in the preliminary questionnaire, I had three people play my game when it was near completion to get some verbal feedback. The conclusion from this was that the styling and approach to my game was good, and they enjoyed the gameplay, but the game was not easy to follow in some places, so any strategic playing was made difficult because it was difficult to track the different player's moves due to the relevant information not being provided all of the time. In response to this, I added in more display screens between turns and for each type of obstacle or interference to explain to the user what had happened, and most importantly making sure the user was always aware of what they rolled for this to come about.

Future Improvements

If I had had more time for the project, I would have liked to network the game so that two different users from different locations could have played against each other from different devices. As the project was nearing completion, however, I was still finishing my original functionality, which needed my attention. I would also have liked to create a better interface for the single player mode, by letting the player have their turn separately, then switching to the spinner screen and automatically generating a spin for the computer and displaying it to the user before continuing the game, as I think this would make the same easier to follow for the user. However, the way I created my app was through buttons, which require a user to click them to generate an action, so the only way to accomplish a separate turn for the computer would be to get the player to click through the screens for their opponent, which seemed counterintuitive.

Conclusion

Therefore, overall I am pleased with the style and approach of my application based on third party feedback from the analysis stage from existing systems, and I am confident that my game works sufficiently. I am also satisfied that I met all of my objectives when creating this game, and I have not strayed from my original path of design too much. The final set of third party feedback, and my response to it, has assured me that my game is now user-friendly and enjoyable to play. Although I would have developed and improved the application if the project was extended, I am happy with the results of what I have achieved.

NEA Questionnaire

1. Which of the two games, the online board game or the phone app slots game, did you enjoy most? (Please tick)

Online board game	Phone application

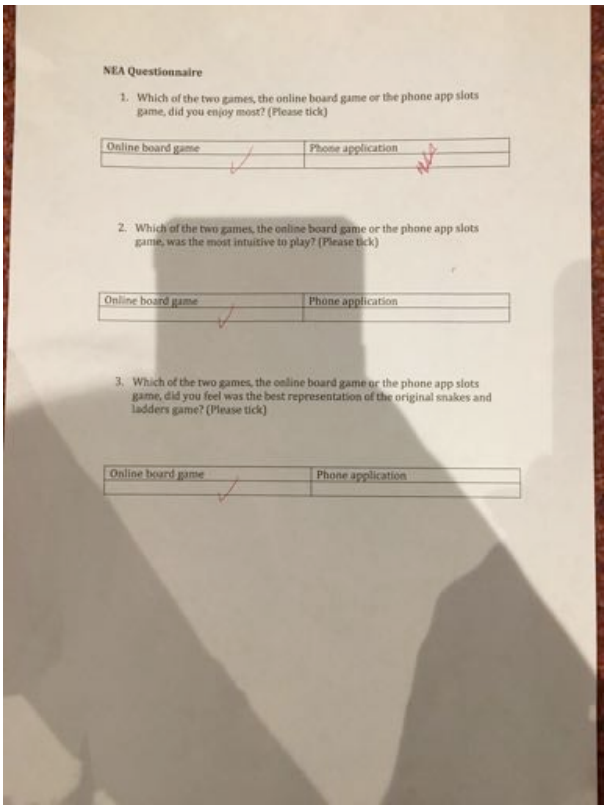
2. Which of the two games, the online board game or the phone app slots game, was the most intuitive to play? (Please tick)

Online board game	Phone application

3. Which of the two games, the online board game or the phone app slots game, did you feel was the best representation of the original snakes and ladders game? (Please tick)

Online board game	Phone application

Filled Questionnaires



NEA Questionnaire

1. Which of the two games, the online board game or the phone app slots game, did you enjoy most? (Please tick)

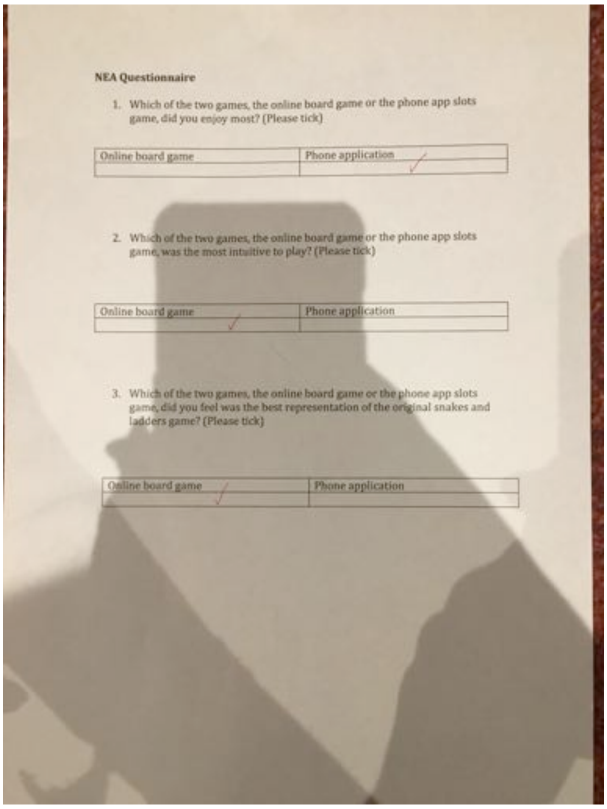
Online board game	Phone application
✓	

2. Which of the two games, the online board game or the phone app slots game, was the most intuitive to play? (Please tick)

Online board game	Phone application
✓	

3. Which of the two games, the online board game or the phone app slots game, did you feel was the best representation of the original snakes and ladders game? (Please tick)

Online board game	Phone application
✓	



NEA Questionnaire

1. Which of the two games, the online board game or the phone app slots game, did you enjoy most? (Please tick)

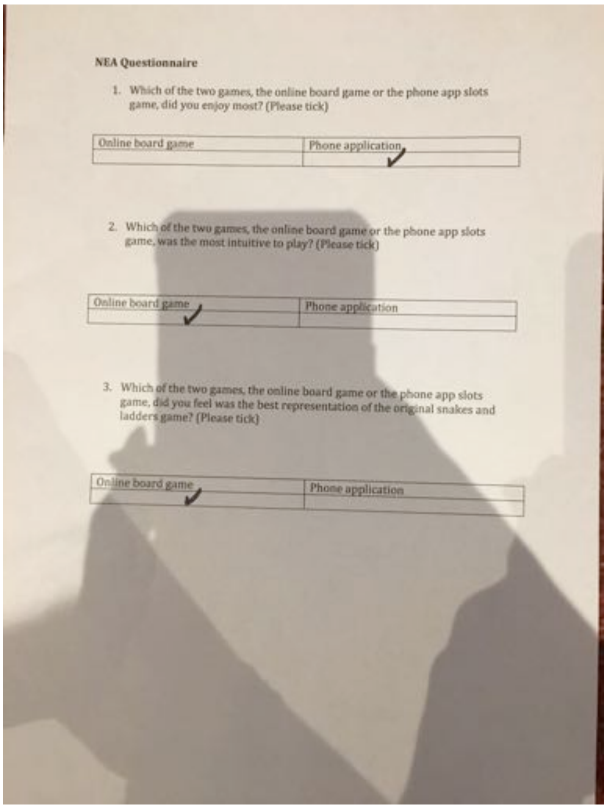
Online board game	Phone application
	✓

2. Which of the two games, the online board game or the phone app slots game, was the most intuitive to play? (Please tick)

Online board game	Phone application
✓	

3. Which of the two games, the online board game or the phone app slots game, did you feel was the best representation of the original snakes and ladders game? (Please tick)

Online board game	Phone application
✓	



NEA Questionnaire

1. Which of the two games, the online board game or the phone app slots game, did you enjoy most? (Please tick)

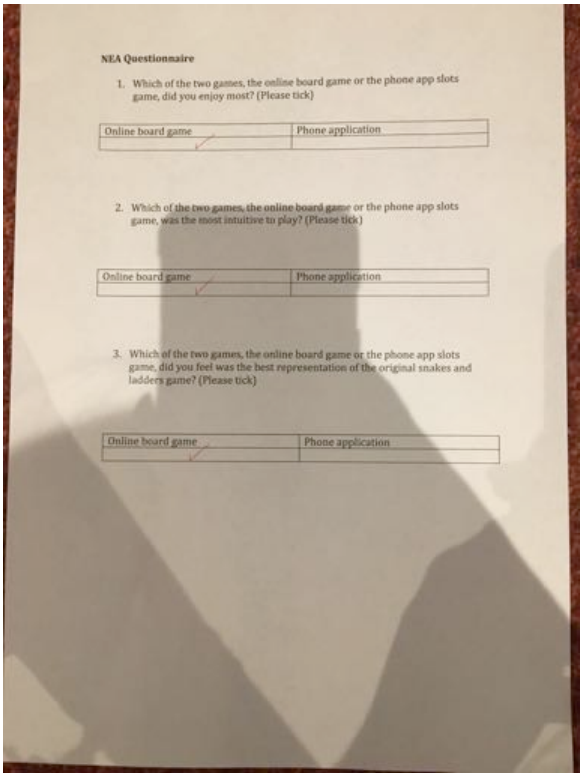
Online board game	Phone application
	✓

2. Which of the two games, the online board game or the phone app slots game, was the most intuitive to play? (Please tick)

Online board game	Phone application
✓	

3. Which of the two games, the online board game or the phone app slots game, did you feel was the best representation of the original snakes and ladders game? (Please tick)

Online board game	Phone application
✓	



NEA Questionnaire

1. Which of the two games, the online board game or the phone app slots game, did you enjoy most? (Please tick)

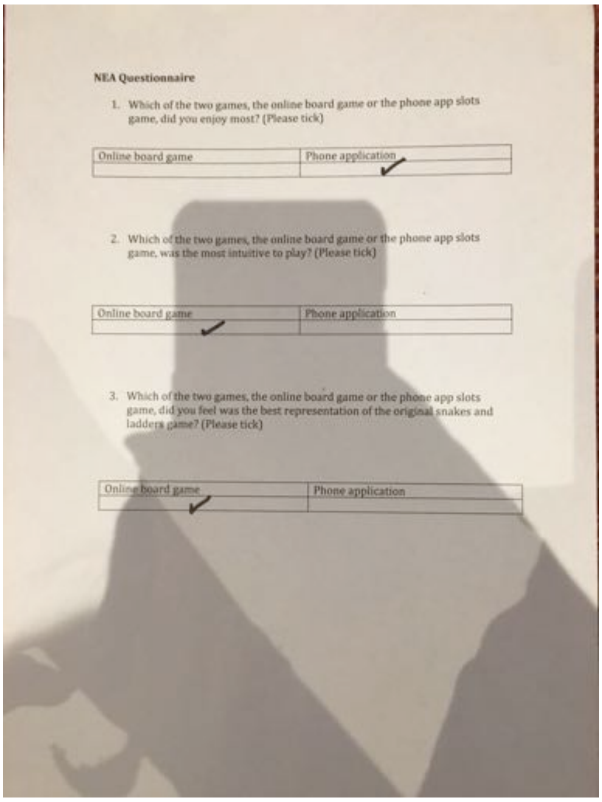
Online board game	Phone application
✓	

2. Which of the two games, the online board game or the phone app slots game, was the most intuitive to play? (Please tick)

Online board game	Phone application
✓	

3. Which of the two games, the online board game or the phone app slots game, did you feel was the best representation of the original snakes and ladders game? (Please tick)

Online board game	Phone application
✓	



NEA Questionnaire

1. Which of the two games, the online board game or the phone app slots game, did you enjoy most? (Please tick)

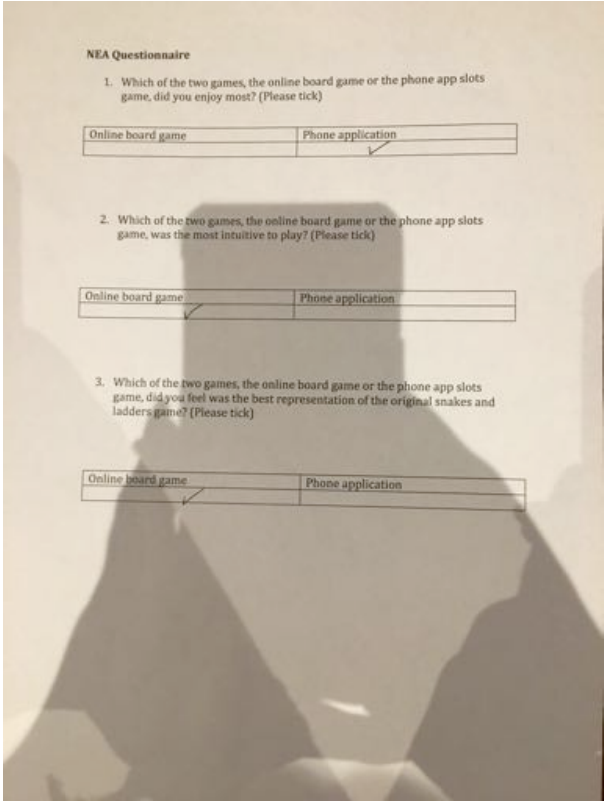
Online board game	Phone application
	✓

2. Which of the two games, the online board game or the phone app slots game, was the most intuitive to play? (Please tick)

Online board game	Phone application
✓	

3. Which of the two games, the online board game or the phone app slots game, did you feel was the best representation of the original snakes and ladders game? (Please tick)

Online board game	Phone application
✓	



NEA Questionnaire

1. Which of the two games, the online board game or the phone app slots game, did you enjoy most? (Please tick)

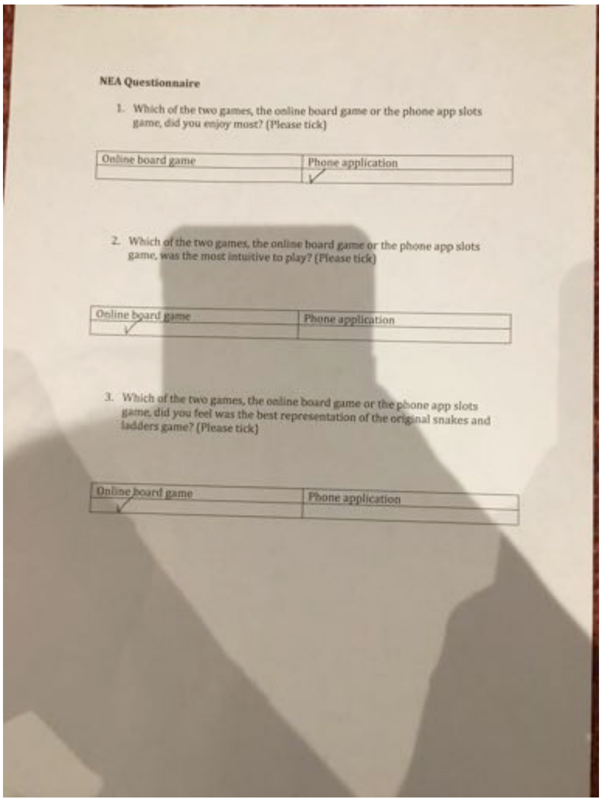
Online board game	Phone application
	✓

2. Which of the two games, the online board game or the phone app slots game, was the most intuitive to play? (Please tick)

Online board game	Phone application
✓	

3. Which of the two games, the online board game or the phone app slots game, did you feel was the best representation of the original snakes and ladders game? (Please tick)

Online board game	Phone application
✓	



NEA Questionnaire

1. Which of the two games, the online board game or the phone app slots game, did you enjoy most? (Please tick)

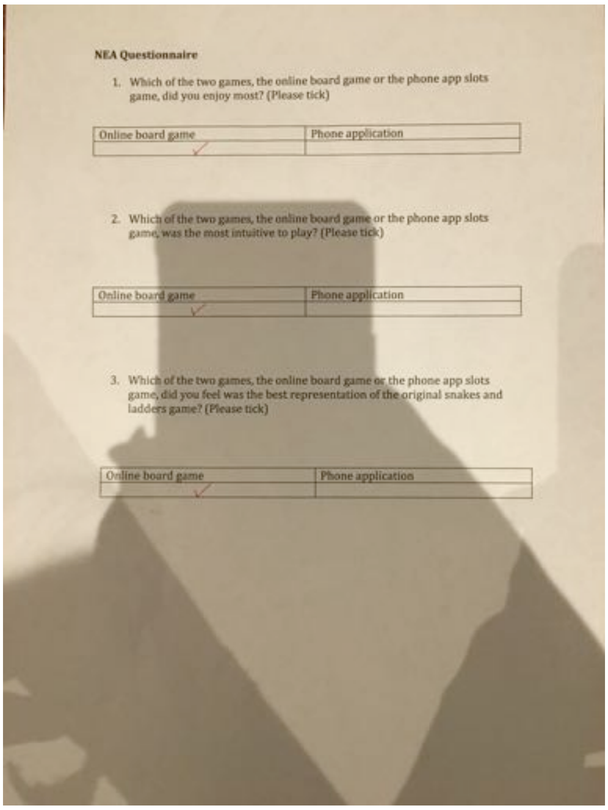
Online board game	Phone application
	✓

2. Which of the two games, the online board game or the phone app slots game, was the most intuitive to play? (Please tick)

Online board game	Phone application
✓	

3. Which of the two games, the online board game or the phone app slots game, did you feel was the best representation of the original snakes and ladders game? (Please tick)

Online board game	Phone application
✓	



NEA Questionnaire

1. Which of the two games, the online board game or the phone app slots game, did you enjoy most? (Please tick)

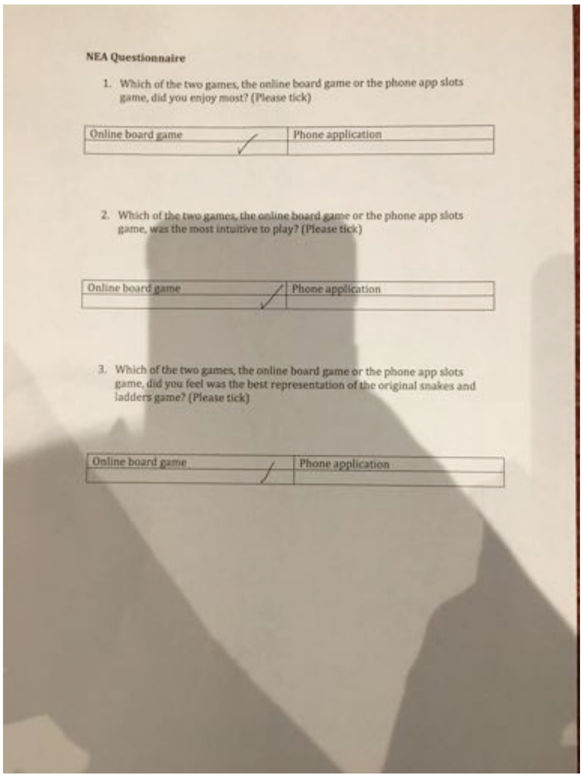
Online board game	Phone application
✓	

2. Which of the two games, the online board game or the phone app slots game, was the most intuitive to play? (Please tick)

Online board game	Phone application
✓	

3. Which of the two games, the online board game or the phone app slots game, did you feel was the best representation of the original snakes and ladders game? (Please tick)

Online board game	Phone application
✓	



NEA Questionnaire

1. Which of the two games, the online board game or the phone app slots game, did you enjoy most? (Please tick)

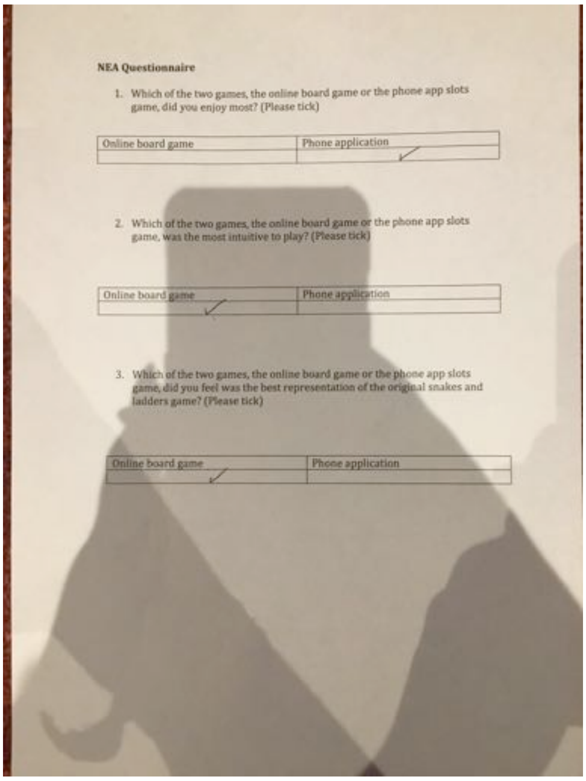
Online board game	Phone application
✓	

2. Which of the two games, the online board game or the phone app slots game, was the most intuitive to play? (Please tick)

Online board game	Phone application
✓	

3. Which of the two games, the online board game or the phone app slots game, did you feel was the best representation of the original snakes and ladders game? (Please tick)

Online board game	Phone application
✓	



NEA Questionnaire

1. Which of the two games, the online board game or the phone app slots game, did you enjoy most? (Please tick)

Online board game	Phone application
	✓

2. Which of the two games, the online board game or the phone app slots game, was the most intuitive to play? (Please tick)

Online board game	Phone application
✓	

3. Which of the two games, the online board game or the phone app slots game, did you feel was the best representation of the original snakes and ladders game? (Please tick)

Online board game	Phone application
✓	