Specification and Design

Escape The House



Megan Loud

Grade 11

Contents:

[**Contents Page**](#_a9p5eyd0f5bv)[**2**](#_a9p5eyd0f5bv)**-**

[**1.1. Problem summary**](#_j5bpdavx1xj2) **4**

[**1.2. Motivation and Research**](#_h0gffwr4a6vt) **5**

[**1.2.1. Existing solution**](#_8nf1656v7ymc)**s**

**1.2.2. How my project is similar to the current available programs**

[**1.2.3. How my project will differ from the current available programs**](#_mcovg0yuy0ky)

[**1.3. Specs of program functionality**](#_ccpqdzxixali) **6-8**

[**1.3.1.**](#_2g8sm4x2t9wb) **General specifications 6**

[**1.3.2.**](#_kj9t6dc38nnm) **Options screen(main screen) specifications 6**

[**1.3.3. Data sheet screen specifications**](#_aqhv68dfgg42) **6**

[**1.3.4. Data collected specifications**](#_aqhv68dfgg42) **6**

[**1.3.5. Area information screen specifications**](#_aqhv68dfgg42) **6**

[**1.3.6. How to play screen specifications**](#_aqhv68dfgg42) **6**

[**1.3.7. Broken picture frames screen(riddle) specifications**](#_aqhv68dfgg42) **7**

[**1.3.8. Door code screen specifications**](#_aqhv68dfgg42) **7**

[**1.3.9. Music box screen specifications**](#_aqhv68dfgg42) **7**

[**1.3.10. Torn up pictures screen(puzzle) specifications**](#_aqhv68dfgg42) **7**

[**1.3.11. Storyline screen specifications**](#_aqhv68dfgg42) **7**

[**1.3.12. Task completed screen specifications**](#_aqhv68dfgg42) **7**

[**1.3.13. Task failed screen specifications**](#_aqhv68dfgg42) **8**

[**1.3.14. Completed game(end) screen specifications**](#_aqhv68dfgg42) **8**

[**1.4. Specs of data storage**](#_q5lpb5g17jbe) **9**

[**1.4.1.**](#_aqhv68dfgg42) **brokenPicFramesInfo 9**

[**1.4.2.**](#_aqhv68dfgg42) **doorInfo 9**

[**1.4.3.**](#_aqhv68dfgg42) **musicBoxInfo 9**

[**1.4.4.**](#_aqhv68dfgg42) **tornPicsInfo 9**

[**1.4.5.**](#_aqhv68dfgg42) **completedTaskInfo**

[**2.1. Interface design**](#_3zwasmfnfqxy) **10-19**

[**2.1.1.**](#_tepnd7bpxrw2) **Main screen 10**

[**2.1.2.**](#_4kt7w9m6wdtj) **Data sheet screen 11**

[**2.1.3.**](#_4kt7w9m6wdtj) **Data collected screen 12**

[**2.1.4.**](#_4kt7w9m6wdtj) **Area Information screen 12**

[**2.1.5.**](#_4kt7w9m6wdtj) **How to play screen 13**

[**2.1.6.**](#_4kt7w9m6wdtj) **Storyline screen 14**

[**2.1.7.**](#_4kt7w9m6wdtj) **Riddle screen 14**

[**2.1.8.**](#_4kt7w9m6wdtj) **Hangman screen 15**

[**2.1.9.**](#_4kt7w9m6wdtj) **Door code screen 16**

[**2.1.10.**](#_4kt7w9m6wdtj) **Puzzle screen 17**

[**2.1.**](#_4kt7w9m6wdtj)**11. Task collected screen 18**

[**2.1.12.**](#_4kt7w9m6wdtj) **Task failed screen 19**

[**2.1.13.**](#_4kt7w9m6wdtj) **End screen 19**

[**2.2. Program flow**](#_jpw6m2dpd17p) **20-29**

[**2.2.1.**](#_wu0iogzeemh) **Main Screen - OptionsScreen 20**

[**2.2.2.**](#_wu0iogzeemh) **Data sheet - DataSheetScreen 20**

[**2.2.3.**](#_wu0iogzeemh) **Data sheet information - InfoScreen 21**

[**2.2.4.**](#_wu0iogzeemh) **Area info - AreaInformationScreen 21**

[**2.2.5.**](#_wu0iogzeemh) **How to play - HowToPlay 22**

[**2.2.6.**](#_wu0iogzeemh) **Storyline - StorylineScreen 22**

[**2.2.7.**](#_wu0iogzeemh) **Riddle screen - BrknPicFrameScreen 22**

[**2.2.8.**](#_wu0iogzeemh) **Door code - DoorScreen 23**

[**2.2.9.**](#_wu0iogzeemh) **Hangman screen - MusicBoxScreen 24**

[**2.2.10.**](#_wu0iogzeemh) **Picture swap game - TornPictureScreen 25**

[**2.2.11.**](#_wu0iogzeemh) **Winning screen - TaskCompletedScreen 28**

[**2.2.12.**](#_wu0iogzeemh) **Losing screen - TaskFailedScreen 28**

[**2.2.13.**](#_wu0iogzeemh) **Ending screen - EndScreen 2**[**9**](#_wu0iogzeemh)

[**2.3. Class Design**](#_mrxpaz5p6e8u) **30-31**

**ChangingScreensMethods 30**

**DataSheetMethods 30**

**DoorCodeMethods 30**

**HangmanMethods 30**

**RiddleMethods 31**

**PuzzleMethods 31**

[**2.4. Secondary storage design**](#_h9nogm8by6q) **32-33**

[**2.4.1.**](#_wu0iogzeemh) **brokenPicFramesInfo.txt 32**

[**2.4.1.**](#_wu0iogzeemh) **doorInfo.txt 32**

[**2.4.1.**](#_wu0iogzeemh) **musicBoxInfo.txt 32**

[**2.4.1.**](#_wu0iogzeemh) **tornPicsInfo.txt 33**

[**2.4.1.**](#_wu0iogzeemh) **completedTaskInfo.txt 33**

1.1 Problem Summary:

My game is a mystery game, which includes puzzles, challenges and a storyline which ties all of these aspects together. My target audience of this game is anyone who enjoys mystery games, puzzles or just wants to have an escape from boredom. The goal of this game is to escape a house which the player has been trapped in. To do this the user must complete all the minigames within the room and find the clues along the way. While the player does this, small portions of the storyline will be revealed and the user must try to uncover the mystery of the house they are trapped in.

A brief summary of the storyline is the player gets stuck in the house with no recollection on how they got there. As the player works to find a way out of the house they will slowly uncover how they got there and the dark history of the house. This includes a muderous family member that now has diverted to kidnapping and making people try to escape the house with the little resources provided.

I want there to be three rooms: the bedroom (where the player will begin the game), the lounge (which is the next room after the player exits the bedroom) and the kitchen (which will lead the player to their final escape). This is part 1 of this project and, hence, consists only of the games in the bedroom.

**The bedroom:**

There will be three minigames in this room. The first game is hangman with the theme of trying to open a jewellery box which will be very much like the traditional game: the player must guess letters according to a topic before they run out of chances. There will be three levels in this minigame, each giving a certain clue to help escape and more context to the storyline. The second game is a puzzle, which will be themed as the user trying to piece together torn up photos. There will also be three levels here with each of them revealing pieces of the storyline once completed. The third game is a riddle. There will only be one of them and once completed the player will receive a clue. All of these clues will lead up to a door code that the player will have to input in order to move on to the next room.

1.2 Motivation and Research:

1.2.1 Existing solution:

There are similar games that currently exist:

* Escape game : 50 rooms 1 https://play.google.com/store/apps/details?id=com.coldapp.at50rooms1&hl=en\_ ZA&gl=US
* Escape game: 50 rooms 2 https://play.google.com/store/apps/details?id=com.coldapp.at50rooms2&hl=en\_ ZA&gl=US
* Escape game: 50 rooms 3 https://play.google.com/store/apps/details?id=com.coldapp.at50rooms3&hl=en\_ ZA&gl=US

1.2.2 How my project is similar to the current available programs:

My game is similar to these programs in the sense that the main goal is the same: to find clues to get out. One example of the similarities is the door code and being able to find the numbers to the door all throughout the game.

1.2.3 How my project will differ from the current available programs:

My motivation for this project was to create a game that had many mini games within it and to link them all together in one big storyline. This storyline is what makes my game different from the other games mentioned because it is more thought provoking and makes the user want to carry on playing. Another reason my game is different is because it has many parts to it, such as logical parts and mindless parts. It is an all in one game, which is exactly what I wanted it to be.

1.3 Specifications of Program Functionality:

# BASIC FUNCTIONALITY:

# 1.3.1 General:

* Each screen leads back to one of the main screens
* Each game has its own permanent storage file and backend class
* Changing between screens also has its own backend class and will use the permanent storage

# 1.3.2 UserScreen:

* Allows the player to make new users that will start from the beginning of the game
* Allows the player to pick between the existing users they have made
* Needs an exit button for the user to exit the game
* Will have a load button to load the game of the user selected
* Will have a new user button to link the user to the NewUserScreen
* Will have a delete button which will delete the selected user

# 1.3.3 NewUserScreen:

* Has a username input area that the user can use to name their player
* Has a create user button which will create a user with the entered username with all the rest of the values set to false and store it instorage
* Will have a back button to link the user back to the UserScreen

# 1.3.4 DataSheetScreen:

* Gives the user a choice about what data they want to view
* Links the user to DataCollectedScreen
* Must have a button that allows the user to close this screen

# 1.3.5 InfoScreen:

* Displays information about the option selected in the data sheet screen(will only display this information if the selected task has been completed and the user has obtained the information)
* Will have a back button that will lead back to DataSheetScreen
* Must have a button that allows the user to close this screen

# 1.3.6 AreaInfoScreen:

* Will display information about the previously selected area
* Must have a button that allows the user to close the screen

# 1.3.7 HowToPlayScreen:

* Displays information on how to play the selected game
* Needs a button to close the screen

# 1.3.8 StorylineScreen:

* Must display the obtained information and storyline about the area the user selected previously
* Will have a button which will take the user back to the OptionsScreen

# 1.3.9 TaskCompletedScreen:

* Displays that the user completed the task
* Will have a button which will open OptionsScreen

# 1.3.10 TaskFailedScreen:

* Displays that the user failed the task
* Will have a replay button which will take the user back to the previous game they just played
* Will have a home button which will take the user back to OptionsScreen

# **BEDROOM SCREENS:**

# **1.3.11 OptionsScreen:**

* Links everything in the game
* Allows the user to choose where they would like to go
* Has an exit button which will take the user to the StartScreen
* Gives the user access to the data sheet

# **1.3.12 RiddleScreen:**

* Displays the riddle for the first room(bedroom)
* Allows the user to input an answer
* Must have an answer button that will trigger an input check to see if the answer inputted is the correct answer
* If the answer is correct then TaskCompletedScreen will be opened
* If the answer is wrong then the input area is cleared and a progress bar will be addedd to. If the progress bar is completed(3 wrong answers), the TaskFailed screen will be opened
* Needs a home button that will open the OptionsScreen
* Will have a button which will open the HowToPlayScreen

# **1.3.13 HangmanScreen:**

* Allows the user to input an answer into the letter guess box
* Must have a button that the user can press to test their input
* Will present the user with a string of “\_”
* Will display the right and wrong letter guesses in their respective places
* Will have a progress bar that increases with every wrong letter inputted
* Links the user to HowToPlayScreen and Options screen via buttons

# **1.3.14 PuzzleScreen:**

* Allows the user to play a puzzle game (sorting out torn picture fragments)
* Must have 6 buttons that have pictures on them and will run methods that swap two pictures when pressed
* Links the user to HowToPlayScreen and OptionsScreen via buttons

# **1.3.15 DoorCodeScreen:**

* The door code screen for the first room (bedroom)
* Will allow the user to input three numbers that will be dispayed in three text areas(an area for each number)
* When the user is done they can click an answer button which will check if the input is correct
* If it is correct, then the task completed screen will be opened
* If it is incorrect, then the 3 boxes will be cleared and the user can try again
* Will link the user to HowToPlayScreen and OptionsScreen via buttons

# **1.3.16 EndScreen:**

* Will have an exit button
* Needs a home button that will link back to the UserScreen

1.4 Specifications of data storage:

# 1.4.1 userInfo:

* **FIELDS:** All the users (username and game stats)
* **WHEN ARE FIELDS ACCESSED:** from the start up of a user’s game (when you choose it in the start screen)

# 1.4.2 brokenPicFramesInfo:

* **FIELDS:** how to play, area information and storyline
* **WHEN ARE FIELDS ACCESSED:** when displaying AreaInfoScreen, HowToPlayScreen and StorylineScreen

# 1.4.3 musicBoxInfo:

* **FIELDS:** how to play, area information and storyline
* **WHEN ARE FIELDS ACCESSED:** when displaying AreaInfoScreen, HowToPlayScreen and StorylineScreen

# 1.4.4 tornPicsInfo:

* **FIELDS:** how to play, area information and storyline
* **WHEN ARE FIELDS ACCESSED:** when displaying AreaInfoScreen, HowToPlayScreen and StorylineScreen

# 1.4.5 doorInfo:

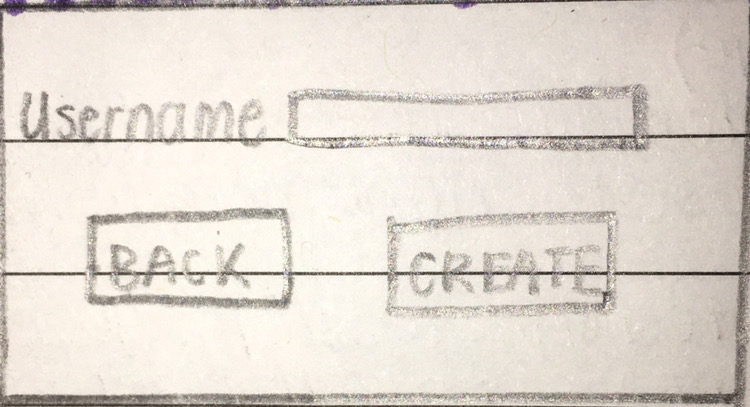
* **FIELDS:** how to play, area information and storyline
* **WHEN ARE FIELDS ACCESSED:** when displaying AreaInfoScreen, HowToPlayScreen and StorylineScreen

2.1 Interface Design:

# 2.1.1 UserScreen:

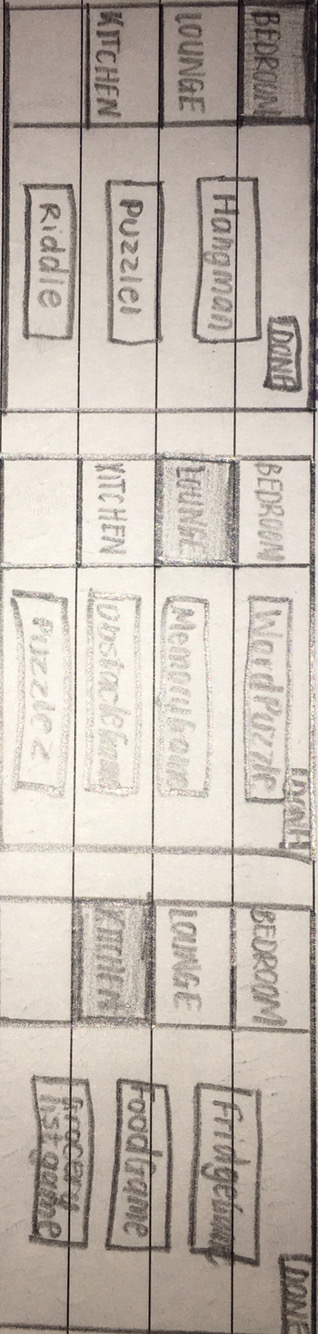
| **Description** | The opening screen when the application is started which allows the user to open different players and gives them access to be able to create new players |
| --- | --- |
| **Data In** | * A background image is obtained from the resources folder |
| **Actions** | **Player list**  Allows the user to select a singular player  **New User Button**  Opens the new user screen  **Load Button**  Obtains the game state of the selected player and loads one of the main screens depending on where the user finished last using that player  **Delete Button**  Deletes the selected player  **Exit Button**  Disposes the screen |

# 2.1.2 NewUserScreen:



| **Description** | Allows the user to create new player(which will have a game state of beginning of the game) |
| --- | --- |
| **Data In** | * A background image is obtained from the resources folder to be presented |
| **Actions** | **Username Field**  Allows the user to enter a name to call their player  **Create Button**  Retrieves the name in the username field and creates a new player object  **Back Button**  Closes this screen and opens the UserScreen |

# 2.1.3 DataSheetScreen:

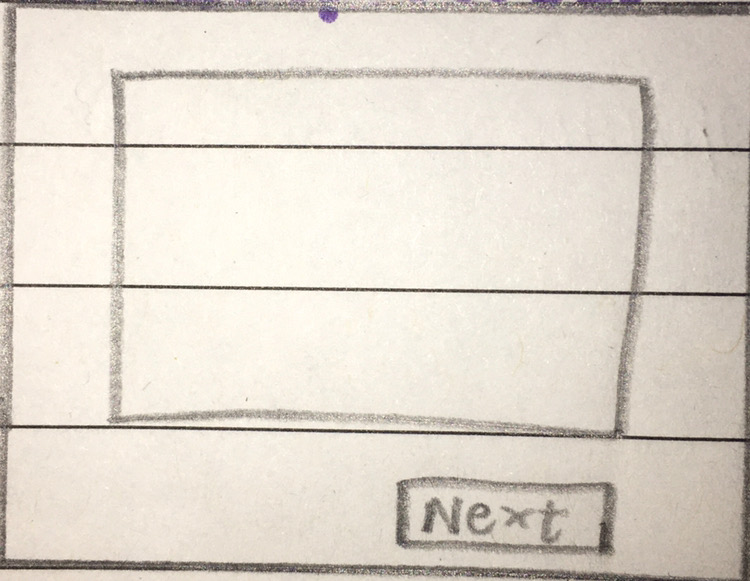


| **Description** | Gives the user a choice on which task’s data they would like to see |
| --- | --- |
| **Data In** | * A background image is obtained from the resources folder |
| **Actions** | **Done Button**  Disposes the screen  **Riddle Button**  Disposes the screen and opens the DataCollectedScreen  **Puzzle1 Button**  Disposes the screen and opens the DataCollectedScreen  **Hangman Button**  Disposes the screen and opens the DataCollectedScreen |

# 2.1.4 InfoScreen:

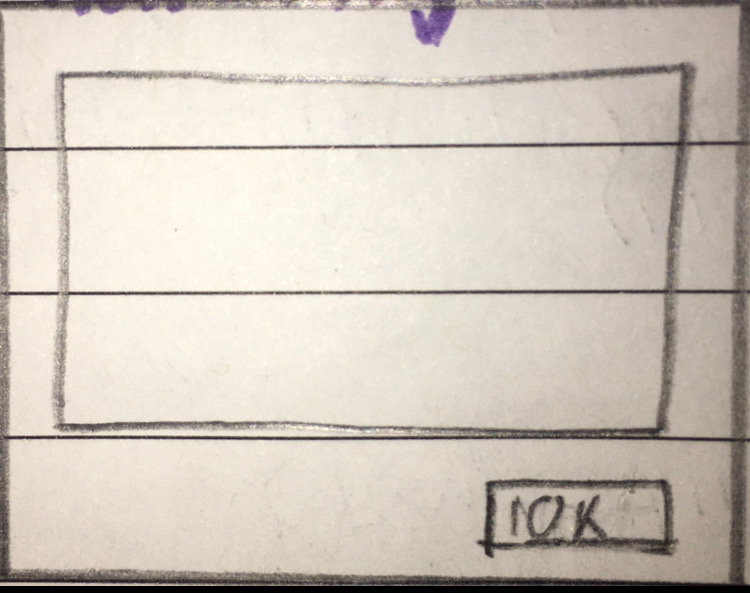
| **Description** | Retrieves and displays information about the task previously selected in the DataSheetScreen from storage |
| --- | --- |
| **Data In** | * an image is retrieved from the resources folder to be used as a background * If the task is completed, text will be retrieved through the DataSheetMethods class to access the information |
| **Actions** | **Back button**  Redirects the user to the DataSheetScreen  **Done button**  Closes this screen |

# 2.1.5 AreaInfoScreen:



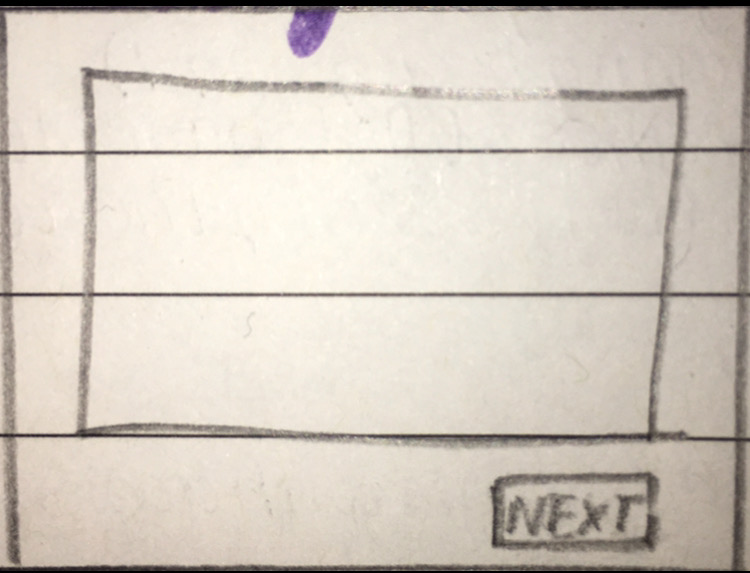
| **Description** | Displays information about the game selected in the OptionsScreen |
| --- | --- |
| **Data In** | * an image is retrieved from the resources folder to be used as a background * Text will be retrieved through the ChangingScreenMethods class to access the area information |
| **Actions** | **Next button**  Redirects the user to the HowToPlayScreen and the game screen of the option the user selected |

# 2.1.6 HowToPlayScreen:



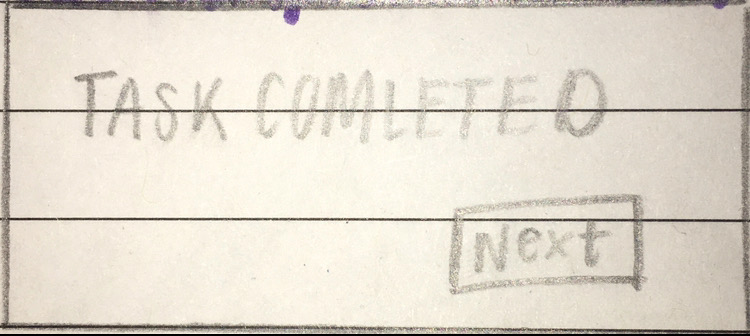
| **Description** | Displays how to play the game that the user selected in the OptionsScreen |
| --- | --- |
| **Data In** | * an image is retrieved from the resources folder to be used as a background * Text will be retrieved through the ChangingScreenMethods class to access the how to play area |
| **Actions** | **Ok Button**  Closes this screen |

# 2.1.7 StorylineScreen:



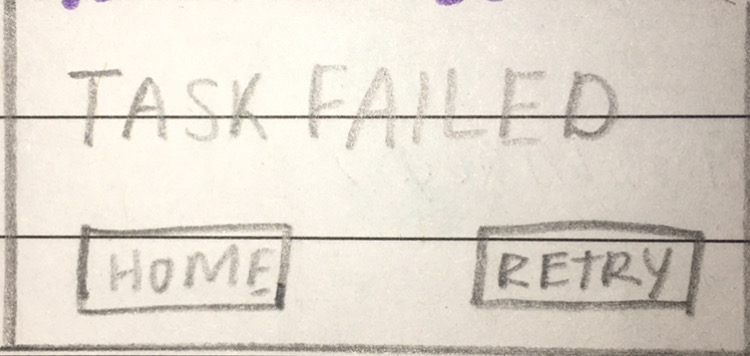
| **Description** | Displays the storyline of different parts of the game |
| --- | --- |
| **Data In** | * an image is retrieved from the resources folder to be used as a background * Text will be retrieved through the ChangingScreenMethods class to access the storyline area |
| **Actions** | **Next button**  The user is redirected to the MapScreen |

# 2.1.8 TaskCompletedScreen:



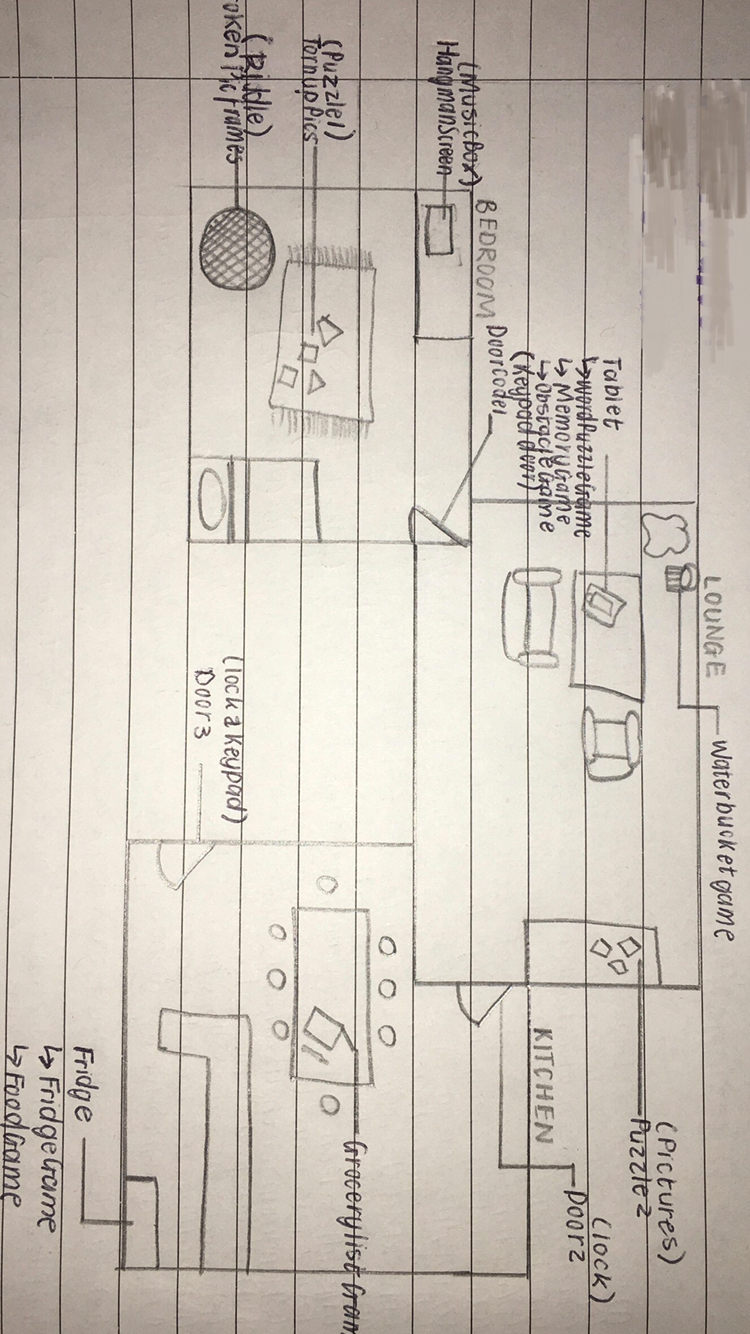
| **Description** | Displays that the user completed the task |
| --- | --- |
| **Data In** | * an image is retrieved from the resources folder to be used as a background |
| **Actions** | **Next button**  Opens the StorylineScreen and closes this screen |

# 2.1.9 TaskFailedScreen:



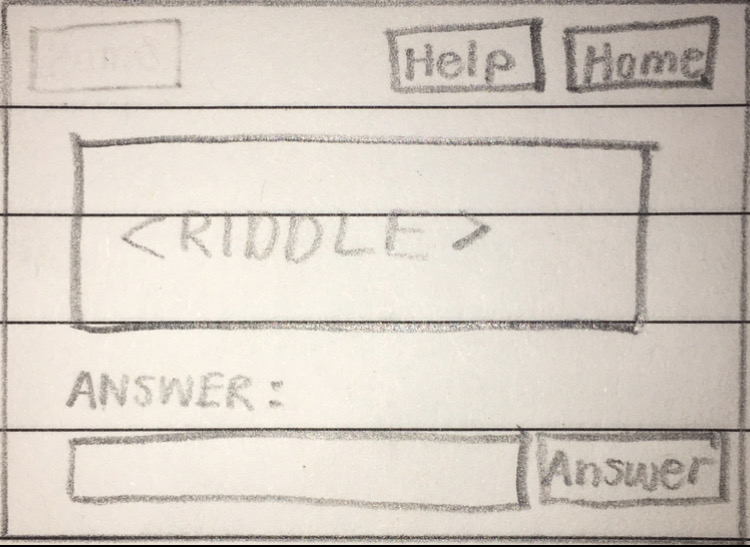
| **Description** | Displays that the user failed this task |
| --- | --- |
| **Data In** | * an image is retrieved from the resources folder to be used as a background |
| **Actions** | **Retry button**  Opens the previous game screen  **Home button**  Redirects the user to the OptionsScreen |

# 2.1.10 OptionsScreen:



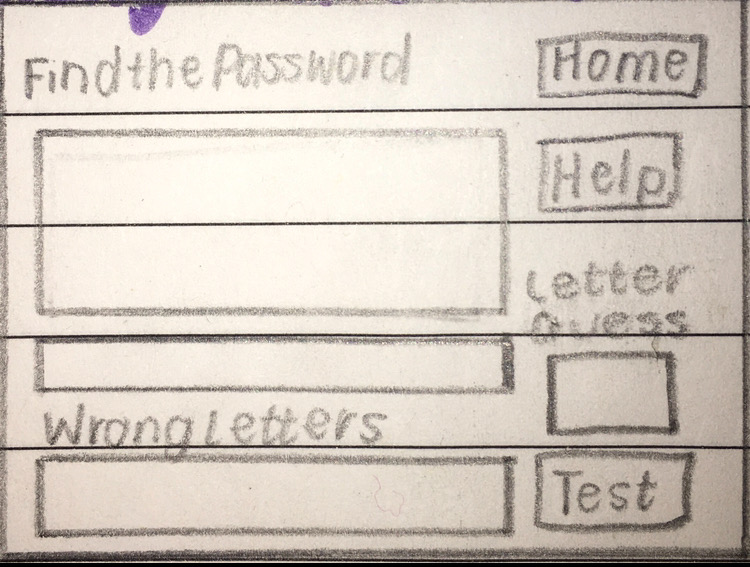
| **Description** | Gives the user a choice on where they would like to go |
| --- | --- |
| **Data In** | * an image is retrieved from the resources folder to be used as a background |
| **Actions** | **Home Button**  Closes this screen and opens the UserScreen  **Data sheet Button**  Closes this screen and opens the DataSheetScreen |

# 2.1.11 RiddleScreen:



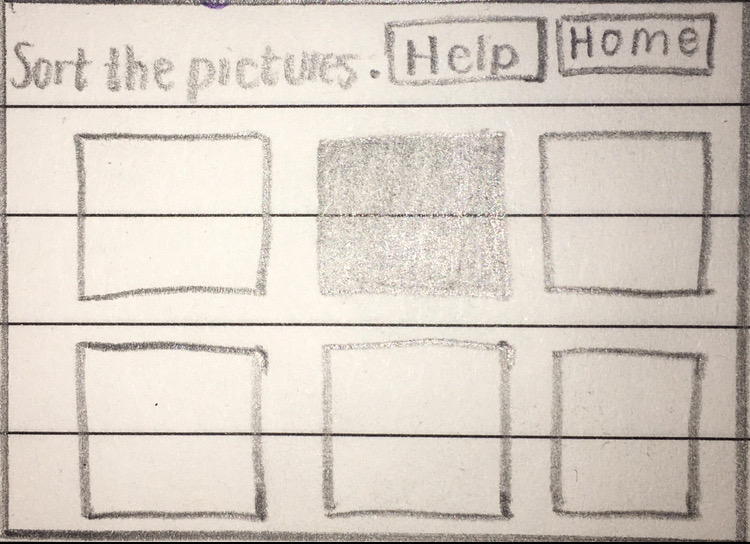
| **Description** | Displays a riddle and allows the user to input an answer |
| --- | --- |
| **Data In** | * A background image is obtained from the resources folder |
| **Actions** | **Answer button**  Retrieves the input from the input area and checks if it is the correct answer.   * If it is correct, then the user will be redirected to the TaskCompletedScreen * If it is wrong, then the input area will be cleared. After 3 wrong answers the user will be redirected to the TaskFailedScreen.   **Home Button**  Closes this screen and opens the OptionsScreen  **How to play Button**  Opens the HowToPlayScreen |

# 2.1.12 HangmanScreen:



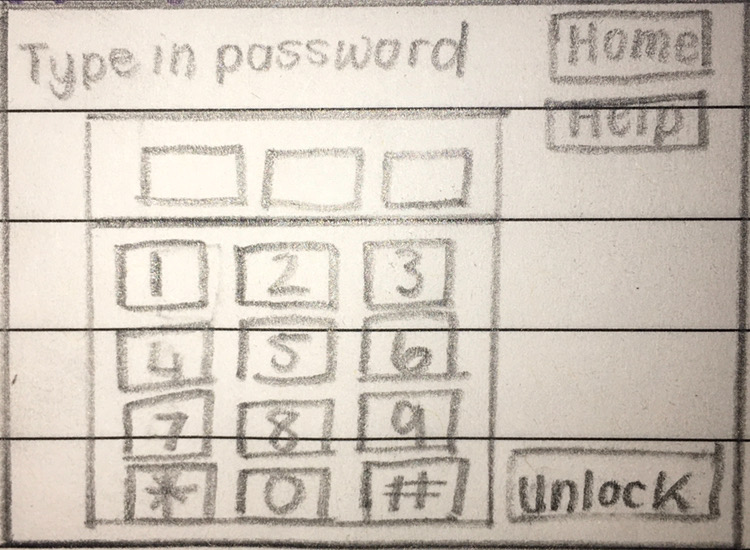
| **Description** | Displays the hangman game and allows the user to input letters which will be displayed in their respective text areas depending if the letter is right or wrong |
| --- | --- |
| **Data In** | * A background image is retrieved from the resources folder * the word the user is guessing will be displayed in the big box. It will be retrieved from the HangmanMethods class. * the wrong letters will be displayed in the wrong letters box. It will be retrieved from the HangmanMethods class. |
| **Actions** | **Test letter button**  Retrieves the user’s input and checks if it is a right letter or wrong letter   * if it is right, then it will be displayed in its respective place/s in the word * if it is wrong, then the letter will be added to the wrong letters and the progress bar increases by 20% * if the progress bar becomes fill, then the user is redirected to the task failed screen * if the word is successfully completed, then the user is redirected to the task completed screen   **Home Button**  Closes this screen and opens the MapScreen  **How to play Button**  Opens the HowToPlayScreen |

# 2.1.13 PuzzleScreen:



| **Description** | Gives the user an interface that they can play a puzzle on and move puzzle pieces on |
| --- | --- |
| **Data In** | * A background image is retrieved from the resources folder * 6 images are retrieved from the resources folder and are displayed in 6 different buttons |
| **Actions** | **Puzzle piece buttons**  The puzzle piece clicked swaps with the black picture, if it is next to it. The picture order is checked to see if it the correct order   * if the order is correct, them the task completed screen is opened   **Home Button**  Closes this screen and opens the OptionsScreen  **How to play Button**  Opens the HowToPlayScreen |

# 2.1.14 DoorScreen:



| **Description** | Allows the user to input a door code |
| --- | --- |
| **Data In** | * A background is retrieved from the resources folder * an image is retrieved from the resources folder to be displayed in the middle of the screen as a keypad |
| **Actions** | **1/2/3/4/5/5/6/7/8/9/0 button**  Adds the number to the door code, if there are any free slots left. The number is displayed in the next consecutive clear text area  Adds the number to the door code, if there are any free slots left. The number is displayed in the next consecutive clear text area  **Answer button**  The user input is checked  **Home Button**  Closes this screen and opens the MapScreen  **How to play Button**  Opens the HowToPlayScreen |

# 2.1.15 EndScreen:

| **Description** | Displays that the user has reached the end of the game |
| --- | --- |
| **Data In** | * A background image is retrieved from the resources folder |
| **Actions** | **Home Button**  Closes this screen and opens the UserScreen  **Exit Button**  Closes the game |

2.2 Program Flow:

# **2.2.1 UserScreen:**

* When the screen is **initialised**

Create a UserManager object called ‘users’

Populating the list with the names of the users

ArrayList<String> usersNames <- users.getListNames()

ArrayList<String> names <- new ArrayList<>()

for(User user : users)

names.add(user.getUsername())

END FOR

return names

DefaultListModel listModel <- new DefaultListModel()

listModel.addAll(usersNames)

userList.setModel(listModel)

* When the **New User Button** is pressed

UserScreen is disposed()

NewUserScreen is launched

* When the **Delete Button** is pressed

UserManager userArray <- new UserManager()

userArray.delete(userList.getSelectedIndex())

users.remove(selectedIndex)

String fileOutput <- “”

for(User user: users)

fileOutput <- fileOutput + user.toString() + “\n”

END FOR

try:

FileWriter fileWriter <- new FileWriter(filepath)

PrintWriter printWriter <- new PrintWriter(fileWriter)

printWriter.print(fileOutput)

fileWriter.close()

printWriter.close()

catch:

print(“File not found”)

END TRY-CATCH

Refreshing the list

ArrayList<String> usersNames <- users.getListNames()

ArrayList<String> names <- new ArrayList<>()

for(User user : users)

names.add(user.getUsername())

END FOR

return names

DefaultListModel listModel <- new DefaultListModel()

listModel.addAll(usersNames)

userList.setModel(listModel)

* When the **Exit Button** is pressed

UserScreen is disposed()

* When the **Load Button** is pressed

Checks if a user was selected

if(userList.selectionEmpty())

selectUserMessage.setText(“Please select a user”)

selectUserMessage.setOpaque(true)

else

int currentIndex = userList.getSelectedIndex()

UserManager.setCurrentUserIndex(currentIndex)

UserScreen is disposed()

OptionsScreen is launched

# **2.2.2 NewUserScreen:**

* When the **Back Button** is pressed

NewUserScreen is disposed()

UserScreen is launched

* When the **Create User Button** is pressed

User user <- new User(usernameTextArea.getText(), false, false, false, false)

UserManager users <- new UserManager()

users.createUser(user)

try:

FileWriter fileWriter <- new FileWriter(filpath,true)

PrintWriter addUser <- new PrintWriter(fileWriter)

addUser.println(user.toString())

addUser.close()

fileWriter.close()

catch:

print(“File not found”)

END TRY-CATCH

NewUserScreen is disposed()

UserScreen is launched

# **2.2.3 OptionsScreen:**

* When the screen is **inititialised**

ChangingScreenMethods screenMethods <- new ChangingScreenMethods()

screenMethods.resetOptionVariables()

brokenPicFramesOption <- false

tornPicsOption <- false

musicBoxOption <- false

doorOption <- false

* When the **Data Sheet** button is pressed

OptionsScreen is disposed()

DataSheetScreen is launched

* When the **Torn Picture Button** is pressed

OptionsScreen is disposed()

screenMethods.tornUpPicturesOption <- true

PuzzleScreen is launched

* When the **Music Box Button** is pressed

OptionsScreen is disposed()

screenMethods.musicBoxOption <- true

HangmanScreen is launched

* When the **Door Button** is pressed

OptionsScreen is disposed()

screenMethods.doorOption <- true

DoorScreen is launched

* When the **Broken Picture Frame Button** is pressed

OptionsScreen is disposed()

screenMethods.brokenPicFramesOption <- true

RiddleScreen is launched

* When the **Home button** is pressed

OptionsScreen is disposed()

UserScreen is launched

# **2.2.4 DataSheetScreen:**

* When the screen is **inititialised**

Games game <- new Games()

DataSheet dataSheet <- new DataSheet(game.getCurrentUser())

* When the **Done Button** is pressed

DataSheetScreen is disposed()

* When the **Puzzle Button** is pressed

DataSheetScreen is disposed()

dataSheet.setGame <- "tornUpPics"

InfoScreen is launched

* When the **Hangman Button** is pressed

DataSheetScreen is disposed()

dataSheet.setGame <- "musicBox"

InfoScreen is launched

* When the **Riddle Button** is pressed

DataSheetScreen is disposed()

dataSheet.setGame <- "brokenPicFrames"

InfoScreen is launched

# **2.2.5 InfoScreen:**

* When the screen is **initialised**

Games game <- new Games()

User currentUser <- game.getCurrentUser()

DataSheet dataSheet <- new DataSheet(currentUser)

Make a call to the DataSheetMethods(getCompletedGameData)

Scanner lineScanner <- completedTasksInfo.txt

if musicBox AND completedMusicBox

for(int <- 0, from 0 to 1, increase by 1)

info <- sc.nextLine()

END FOR

if brokenPicFrames AND completedBrokenPicFrames

for(int <- 0, from 0 to 2, increase by 1)

info <- sc.nextLine()

END FOR

if tornUpPics AND completedTornPics

for(int <- 0, from 0 to 3, increase by 1)

info <- sc.nextLine()

END FOR

Display info in the infoDisplay text area

* When the **Done** button is pressed

InfoScreen is disposed()

* When the **Back** button is pressed

InfoScreen is disposed()

DataSheetScreen is launched

# **2.2.6 AreaInfoScreen:**

* When the screen is **initialised**

ChangingScreenMethods screenMethods <- new ChangingScreenMethods()

Make a call to the ChangingScreenMethods(getAreaInfo)

filePath <- getTextfileFilepath

textfile <- filePath

lineScanner <- textfile

areaInfo <- lineScanner.next()

Display areaInfo in infoDisplay text area

* When the **Next** button is pressed

Make a call to the ChangingScreenMethods(getAreaInfo)

if(brokenPictureFramesOption)

BrknPicFrameScreen is launched

if(tornUpPicturesOption)

TornPictureScreenis launched

if(musicBoxOption)

MusicBoxScreen is launched

if(doorOption)

DoorScreen is launched

HowToPlay is launched

AreaInformationScreen is disposed()

# **2.2.7 HowToPlayScreen:**

* When the screen is **initialised**

ChangingScreenMethods screenMethods <- new ChangingScreenMethods()

Make a call to the ChangingScreenMethods(getHowToPlay)

filePath <- getTextfileFilepath

textfile <- filePath

lineScanner <- textfile

for(from 0-2, increase by 1)

htp <- lineScanner.next()

END FOR

Display htp in informationArea text area

* When the **Ok Button** is pressed

HowToPlay is disposed()

# **2.2.8 StorylineScreen:**

* When the screen is **initialised**

ChangingScreenMethods screenMethods <- new ChangingScreenMethods()

Make a call to the ChangingScreenMethods (getStoryline)

filePath <- getTextfileFilepath

textfile <- filePath

lineScanner <- textfile

for(from 0-3, increase by 1)

storyline <- lineScanner.next()

END FOR

Display storyline in informationArea text area

* When the **Next Button** is pressed

Make a call to the ChangingScreenMethods(openNextScreen)

if(doorOption)

EndScreen is launched

else

OptionsScreen is launched

StorylineScreen is disposed()

# **2.2.9 RiddleScreen:**

* When the **How to play** button is pressed

HowToPlay is launched

* When the **Home** button is pressed

BrknPicFrameScreen is disposed()

OptionsScreen is launched

* When the **Answer**  button is pressed

userInput(string) <- RiddleMethods.getInput(answerTextArea)

Gets the input using .getText() and returns it as a string

RiddleMethods.rightWrongCheck(userInput, answerTextArea)

\*\*Class variables:

lives(int) <- 3

winOrLose(boolean) <- false

if(userInput = 7/userInput = “seven”/userInput = “Seven”)

TaskCompletedScreen is launched

winOrLose <- true

else

answerTextArea.setText(“”)

Lives <- lives-1

if(lives = 0)

TaskFailedScreen is launched

winOrLose <- true

if(winOrLose)

BrknPicFrameScreen is disposed()

Make a call to RiddleMethods(reset)

lives <- 3

winOrLose <- false

# **2.2.10 DoorScreen:**

* When the **Home** button is pressed

DoorScreen is disposed()

OptionsScreen is launched

Make a call to DoorCodeMethods(reset)

\*\*userCode(string array) is a class variable

for(arrayPos <- from 0-3, increase by 1)

userCode[arrayPos] <- "";

END FOR

* When the **1/2/3/4/5/6/7/8/9/0**  buttons are pressed

Makes a call to DoorCodeMethods(addNumber)

for(arrayPos <- from 0-3, increase by 1)

if(userCode[arrayPos] = “”

userCode[arrayPos] <- buttonPressed

Break out of code loop

END FOR

Makes a call to DoorCodeMethods(updateDisplay)

num1Display.setText(userCode[0]);

num2Display.setText(userCode[1]);

num3Display.setText(userCode[2]);

* When the **Answer** button is pressed

\*\*class variables

correctCode(string array) <- 7, 9, 6

win(boolean) <- false

Makes a call to DoorCodeMethods(winLoseCheck)

numCorrect(int) <- 0

for(arrayPos <- from 0-3, increase by 1)

if(userCode[arrayPos] = correctCode[arrayPos])

numCorrect = numCorrect + 1

if(numCorrect = 3)

TaskCompletedScreen is launched

win <- true

else

Make a call to DoorCodeMethods(reset)

\*\*userCode(string array) is a class variable

for(arrayPos <- from 0-3, increase by 1)

userCode[arrayPos] <- "";

END FOR

# **2.2.11 HangmanScreen:**

* When the screen is **initialised**

wordDisplay.setText(“\_ \_ \_ \_ \_ \_ \_ \_ \_”)

* When the **Home** button is pressed

MusicBoxScreen is disposed()

OptionsScreen is launched

Make a call to HangmanMethods(reset)

\*\*class variables:

userArray(string array) <- \_, \_, \_, \_, \_, \_, \_, \_, \_

numWrongAnswers(int) <- 0;

winOrLose(boolean) <- false;

wrongAnswers(String) <- "";

for(arrayPos <- from 0-9, increase by 1)

userArray[arrayPos] <- “\_”

END FOR

numWrongAnswers <- 0;

winOrLose <- false;

wrongAnswers <- ""

* When the **How to play** button is pressed

HowToPlay is launched

* When the **Test Letter** button is pressed

inputLetter(string) <- letterGuess.getText

Make a call to HangmanMethods(letterCheck)

\*\*fields for letterCheck

inputLetter(String), progressBar(JProgressBar)

\*\*class variables

numWrongAnswers(int) <- 0

wrongAnswers(string) <- “”

completedWordArray(string array) <- b,u,t,t,e,r,f,l,y

completedWordStr(string) <- "butterfly"

completedWordLength <- completedWordStr.length()

numRight(int) <- 0

for(arrayPos <- from 0-completedWordLength, increase by 1)

if(completedWordArray[arrayPos] = inputLetter)

userArray[arrayPos] <- inputLetter

numRight = numRight + 1

END FOR

if(numRight = 0)

wrongAnswers = wrongAnswers inputLetter + ", "

numWrongAnswers = numWrongAnswers + 1;

progressBar.setValue(numWrongAnswers x 20)

Make a call to HangmanMethods(updateScreen)

\*\*fields for updateScreen

rightAnswersTextArea(JTextArea), wrongAnswersTextArea

(JTextArea),guessingTextArea(JTextArea,

progressBar(JProgressBar)

text(string) <- ""

for(arrayPosition <- from 0-9, increase by 1)

text <- text + userArray[arrayPosition] + " "

END FOR

rightAnswersTextArea.setText(text)

wrongAnswersTextArea.setText(wrongAnswers)

guessingTextArea.setText("")

progressBar.repaint()

Make a call to HangmanMethods(WinLoseCheck)

\*\*class variables

winOrLose(boolean) <- false

if(Arrays.equals(completedWordArray, userArray))

TaskCompletedScreen is launched

DataSheetMethods.completedMusicBox <- true

winOrLose <- true

if(numWrongAnswers = 5)

TaskFailedScreen is launched

winOrLose <- true

if(winOrLose)

MusicBoxScreen is disposed()

Make a call to HangmanMethods(reset)

<see home button action to see method>

# **2.2.12 PuzzleScreen:**

* When the **How to play** button is pressed

HowToPlay is launched

* When the **Home** button is pressed

OptionsScreen is launched

TornPictureScreen is disposed()

Make a call to PuzzleMethods(reset)

\*\*class variables:

currentPicOrder(string array) <- "/images/4.jpg",

"/images/1.jpg", "/images/0.jpg", "/images/3.jpg", "/images/2.jpg", "/images/5.jpg"

win(boolean) <- false

currentPicOrder[0] = "/images/4.jpg";

currentPicOrder[1] <- "/images/1.jpg"

currentPicOrder[2] <- "/images/0.jpg"

currentPicOrder[3] <- "/images/3.jpg"

currentPicOrder[4] <- "/images/2.jpg"

currentPicOrder[5] <- "/images/5.jpg”

win <- false

* When the **frame0/frame1/frame2/frame3/frame4/frame5** button is pressed

blankPicFrame <- PuzzleMethods.getBlankPicPos

blankPicPos(int) <- 0

blankPic(String) <- "/images/2.jpg"

for(i <- from 0-6, increase by 1)

if(blankPic = currentPicOrder[i]

blankPicPos <- i

END FOR

returns the blankPicPos

Make a call to PuzzleMethods(framePicSwap)

\*\*fields for framPicSwap

frameNum(int), blankPicFrame(int), frame0(JButton), frame1(JButton), frame2(JButton), frame3(JButton), frame4(JButton), frame5(JButton)

PuzzleMethods SwapPics

SwapPics <- new PuzzleMethods()

if(frameNum = 0)

switch(blankPicFrame)

Case 1: SwapPics.pictureSwap(frame0, frame1,

frameStr[0], frameStr[1])

(pictureSwap method, ran out of space)

\*\*fields for pictureSwap

button1(JButton), button2(JButton),

button1Str(String),button2Str(string)

button1Icon(String) <- getPic(button1Str)

button2Icon(String) <- getPic(button2Str)

button2.setIcon(new ImageIcon(getClass().getResource(button1Icon)));

button1.setIcon(new ImageIcon(getClass().getResource(button2Icon)))

btn1ScreenNumber(int) <- getScreenNumber(button1Str)

(getScreenNumber method)

Fields for getScreenNumber:

String screenStr

Class variable:

frameStr(string array) = {"frame0", "frame1", "frame2",

"frame3", "frame4", "frame5"}

screenNumber <- 0

switch (screenStr)

case "frame1" -> screenNumber = 1

case "frame2" -> screenNumber = 2

case "frame3" -> screenNumber = 3

case "frame4" -> screenNumber = 4

case "frame5" -> screenNumber = 5

END SWITCH

Returns the screenNumber

currentPicOrder[btn1ScreenNumber] <- button2Icon

int btn2ScreenNumber <- getScreenNumber(button2Str)

currentPicOrder[btn2ScreenNumber] <- button1Icon

Case 3: SwapPics.pictureSwap(frame0, frame3,

frameStr[0], frameStr[3])

END OF SWITCH

if(frameNum == 1)

switch(blankPicFrame)

Case 0: SwapPics.pictureSwap(frame1, frame0,

frameStr[1], frameStr[0])

Case 2: SwapPics.pictureSwap(frame1, frame2,

frameStr[1], frameStr[2])

Case 4: SwapPics.pictureSwap(frame1, frame4,

frameStr[1], frameStr[4])

END OF SWITCH

if(frameNum == 2)

switch(blankPicFrame)

Case 1: SwapPics.pictureSwap(frame2, frame1,

frameStr[2], frameStr[1])

Case 5: SwapPics.pictureSwap(frame2, frame5,

frameStr[2], frameStr[5])

END OF SWITCH

if(frameNum == 3)

switch(blankPicFrame)

Case 0: SwapPics.pictureSwap(frame3, frame0,

frameStr[3], frameStr[0])

Case 4: SwapPics.pictureSwap(frame3, frame4,

frameStr[3], frameStr[4])

END OF SWITCH

if(frameNum == 4)

switch(blankPicFrame)

Case 1: SwapPics.pictureSwap(frame4, frame1,

frameStr[4], frameStr[1])

Case 3: SwapPics.pictureSwap(frame4, frame3,

frameStr[4], frameStr[3])

Case 5: SwapPics.pictureSwap(frame4, frame5,

frameStr[4], frameStr[5])

END OF SWITCH

if(frameNum == 5)

switch(blankPicFrame)

Case 2: SwapPics.pictureSwap(frame5, frame2,

frameStr[5], frameStr[2])

Case 4: SwapPics.pictureSwap(frame5, frame4,

frameStr[5], frameStr[4])

END OF SWITCH

Make a call to PuzzleMethods(winCheck)

correctOrder(String[]) <- new String[6]

correctOrder[0] <- "/images/0.jpg"

correctOrder[1] <- "/images/1.jpg"

correctOrder[2] <- "/images/2.jpg"

correctOrder[3] <- "/images/3.jpg"

correctOrder[4] <- "/images/4.jpg"

correctOrder[5] <- "/images/5.jpg"

numCorrectPicPlace(int) <- 0

for(i <- from 0-6, increase by 1)

if(currentPicOrder[i] = correctOrder[i])

numCorrectPicPlace = numCorrectPicPlace + 1

END FOR

if(numCorrectPicPlace = 6)

TaskCompletedScreen is launched

DataSheetMethods.completedTornPics <- true

win = true

if(win)

Make a call to PuzzleMethods(reset)

TornPictureScreen is disposed()

# **2.2.13 TaskCompletedScreen:**

* When the **Next** button is pressed

TaskCompletedScreen is disposed()

StorylineScreen is launched

# **2.2.14 TaskFailedScreen:**

* When the **Home** button is pressed

TaskFailedScreen is disposed()

OptionsScreen is launched

* When the **Retry** button is pressed

Make a call to ChangingScreenMethods(openPreviousGame)

if(brokenPictureFramesOption)

BrknPicFrameScreen is launched

else if(musicBoxOption)

MusicBoxScreen is launched

TaskFailedScreen is disposed()

# **2.2.15 EndScreen:**

* When the **Home Button** is pressed

EndScreen is disposed()

UserScreen is launched

* When the **Exit** button is pressed

EndScreen is disposed()

2.3 Class Design:

| User | Description |
| --- | --- |
| -username : String  -completedBrokenPicFrames : boolean  -completedMusicBox : boolean  -completedTornPics : boolean  -completedDoor : boolean | Stores the username of the object  Stores if the user has completed the riddle or not  Stores if the user has completed hangman or not  Stores if the user has completed puzzle or not  Stores if the user has completed door or not |
| +constructor(username : String, completedBrokenPicFrames : boolean, completedTornPics : boolean, completedMusicBox : boolean, completedDoor : boolean)  +setBrokenPicFramesTrue()  +setMusicBoxTrue()  +setTornPicsTrue()  +setDoorTrue()  +getUsername() : String  +isCompletedBrokenPicFrames() : boolean  +isCompletedMusicBox() : boolean  +isCompletedTornPics() : boolean  +isCompletedDoor() : boolean  +toString() : String | Creates a User object with the required field  Sets completedBrokenPicFrames to true  Sets completedMusicBox to true  Sets completedTornPicsto true  Sets completedDoor to true  Gets the username  Gets completedBrokenPicFrames  Gets completedMusicBox  Gets completedTornPics  Gets completedDoor  Puts all the object’s data into a string |

| UserManager | Description |
| --- | --- |
| -users : ArrayList<User>  -filepath : String  -currentUserIndex : integer | A list of all the users  Filepath to where the users are stored  The index of the current user in *users* |
| +constructor()  +getUsers() : ArrayList<User>  +getCurrentUserIndex() : integer  getSelectedUser(selectedIndex : integer) : User  +setUsers(users : ArrayList<User>)  +setCurrentUserIndex(index : integer)  +createUser(user : User)  +getListNames() : ArrayList<String>  +delete(selectedIndex : integer)  +save(selectedIndex : integer, currentUser : User) | Creates an array list of the users in storage  Gets *users*  Gets *currentUserIndex*  Gets the user at the index provided  Sets *users* to the value of the userlist provided  Sets *currentUserIndex* to the index provided  Adds the user provided to *users* and adds it into storage  Creates a list of the usernames of all the objects  Deletes the user at the selected index from *users* and storage  Saves the users new information to storage |

| ChangingScreenMethods | Description |
| --- | --- |
| -brokenPicFramesOption : boolean  -tornPicsOption : boolean  -musicBoxOption : boolean  -doorOption : boolean  -filepath : String | Stores whether or not this option was selected  Stores whether or not this option was selected  Stores whether or not this option was selected  Stores whether or not this option was selected  Stores the filepath that is needed for a particular game |
| +setBrokenPicFramesOption(brokenPicFramesOption : boolean)  +setTornPicsOption(tornPicsOption : boolean)  +setMusicBoxOption(musicBoxOption : boolean)  +setDoorOption(doorOption : boolean)  +resetOptionVariables()  +getFilepath()  +getAreaInfo() : String  +getHowToPlay() : String  +getStoryline() : String  +openGame()  +openPreviousGame()  +openNextScreen() | Sets *brokenPicFramesOption* to boolean provided  Sets *tornPicsOption* to boolean provided  Sets *musicBoxOption* to boolean provided  Sets *doorOption* to boolean provided  Resets all the variables to false  Gets the filepath for the game selected  Gets area info for a particular game  Gets how to play info for a particular game  Gets storyline info for a particular game  Opens the game that the user selected  Opens the previous game the user selected  Opens the next screen that is required |

| DataSheet | Description |
| --- | --- |
| -completedBrokenPicFrames : boolean  -completedMusicBox : boolean  -completedTornPics : boolean  -game : String | Stores whether game has been completed or not  Stores whether game has been completed or not  Stores whether game has been completed or not  Stores the game that is selected |
| +constructor(currentUser : User)  +setCompletedBrokenPicFrames(completedBrokenPicFrames : boolean)  +setCompletedMusicBox(completedMusicBox : boolean)  +setCompletedTornPics(completedTornPics : boolean)  +setGame(game : String)  +getCompletedGameData(currentUser : User) | Gets the data from the user provided and sets the datasheet fields to its values  Sets *completedBrokenPicFrames* to boolean provided  Sets *completedMusicBox* to boolean provided  Sets *completedTornPics* to boolean provided  Sets *game* to string provided  Gets the data if the game task is completed |

| Games | Description |
| --- | --- |
| -userManager : UserManager  -currentArrayList : ArrayList<User>  -currentUserIndex : integer  -currentUser : User  -dataSheet : DataSheet  **HANGMAN**  -correctWordArray : String[] = {“b”, “u”, “t”, “t”, “e”, “r”, “f”, “l”, “y”}  -correctWordString : String = “butterfly”  -correctWordLength : int  -usersWordArray : String[]  -wrongAnswers : String  -numWrongAnswers : int  -progressBarValue : int  -hangmanWin : boolean  **RIDDLE**  -lives : integer  -riddleWin : boolean  **PUZZLE**  -currentPicOrder : String[]  -frameStr : String[]  -puzzleWin : boolean  **DOOR**  -userCode : String[]  -correctCode : String = {“7”, “9”, “6”}  -doorCodeWin | Creates a UserManager object  Creates an array list of the users  Stores the index of the current user in the arraylist  Stores the currentUser  Creates a DataSheet object  The correct array of letters for the word  The correct word as a string  The length of the word  The array the user will input letters into  A collection of all the wrong letters entered  The number of wrong letters entered  The value that will be displayed onto the progress bar  Stores if the game has been won or not  The number of lives the user has left  Stores if the game has been won or not  Stores the filepaths of the picture order  Stores the frames of the picture order  Stores if the game has been won or not  Stores the code entered by the user  The correct code  Stores whether the game has been won or not |
| +getCurrentUser() : User  -updateCurrentArrayList()  **HANGMAN**  +getDisplayString() : String  +getWrongAnswers() : String  +isHangmanWin() : boolean  +getProgressBarValue() : int  +resetHangman()  +letterCheck(inputLetter : String)  +hangmanWinCheck()  **RIDDLE**  +isRiddleWin() : boolean  +resetRiddle()  -rightWrongCheck(userInput : String)  **PUZZLE**  +getPic(screenStr : String) : String  +isPuzzleWin() : boolean  +getScreenNumber(screenStr : String) : integer  +getCurrentPicOrder() : String[]  +getBlankPicPos() : int  +pictureSwap(button1 : JButton, button2 : JButton, button1Str : String, button2Str : String)  +framePicWap(frameNum : integer, blankPicFrame : int, frame0 : JButton, frame1 : JButton, frame2 : JButton, frame3 : JButton, frame4 : JButton, frame5 : JButton)  +puzzleWinCheck()  +resetPuzzle()  **DOOR**  +getUserCode() : String[]  +isDoorCodeWin() : boolean  +addNumber(buttonPressed int)  +resetDoorCode()  +riddleWinCheck() | Gets *currentUser*  Updates *currentArrayList* to comply with new info  Converts *userArray* into a string  Gets *wrongAnswers*  *Gets hangmanWin*  Gets *progressBarValue*  Resets all hangman values to their default values  Checks if the letter is a part of the word and updates hangman values accordingly  Checks if the game has been won or lost  Gets if the riddle has been won or not  Resets all values to their original values  Checks if the answer is right or wrong  Gets the picture on a particular button  Gets *puzzleWin*  Gets the screen number of a particular frame  Gets *currentPicOrder*  Gets the position that the blank picture is in  Swaps the pictures of the buttons provided  Swaps the pictures depending on what frame they are in  Checks if the game has been won  Resets the puzzle to its original values  Gets *userCode*  Gets *doorCodeWin*  Adds a number to *userCode*  Resets door code values to their original values  Checks if the game is won or not |

2.4 Secondary Storage Design

# 2.4.1 users.txt

**Format**

| <username><riddleCompleted><puzzleCompleted><hangmanCompleted><doorCompleted>  <username><riddleCompleted><puzzleCompleted><hangmanCompleted><doorCompleted>  <username><riddleCompleted><puzzleCompleted><hangmanCompleted><doorCompleted> |
| --- |

**Example**

| Amy#true#false#false#false  Matthew#false#false#false#false  Henry#false#true#true#false |
| --- |

# 2.4.2 brokenPicFramesInfo.txt

**Format**

| <Area information>  <How to play>  <Storyline>  **\*\*Note: Each section can be multiple lines long and is separated using a #** |
| --- |

**Example**

| I am going to the shops  #Pick all the items needed to make cereal  #Now I can eat breakfast |
| --- |

# 2.4.3 musicBoxInfo.txt

**Format**

| <Area information>  <How to play>  <Storyline>  **\*\*Note: Each section can be multiple lines long and is separated using a #** |
| --- |

**Example**

| I am going to the shops  #Pick all the items needed to make cereal  #Now I can eat breakfast |
| --- |

# 2.4.4 tornPicsInfo.txt

**Format**

| <Area information>  <How to play>  <Storyline>  **\*\*Note: Each section can be multiple lines long and is separated using a #** |
| --- |

**Example**

| I am going to the shops  #Pick all the items needed to make cereal  #Now I can eat breakfast |
| --- |

# 2.4.5 doorInfo.txt

**Format**

| <Area information>  <How to play>  <Storyline>  **\*\*Note: Each section can be multiple lines long and is separated using a #** |
| --- |

**Example**

| I am going to the shops  #Pick all the items needed to make cereal  #Now I can eat breakfast |
| --- |