Specification and Design:

Escape The House

Megan Loud

Gr 10

Contents:

1.1 Problem Summary:

In today's world it is very easy for people to get stressed. A good way to relieve yourself of stress is to play a game, which is exactly what my project is. I wanted to create a game that has many different factors within it. I wanted it to have a storyline that would keep the user interested in trying to figure out the mystery, some logical games to broaden the user's logical skills, and small mindless games that the user could enjoy playing while not needing to use too much brain power.

<u>I have three minigames: hangman, a puzzle, and a riddle:</u>
Hangman will be a letter guessing game based in a context of trying to get a jewellery box open. If the user guesses the right letters and finishes the word then they will get a clue that will help them finish the game and give more context about the mystery behind the game. The puzzle game will be based on trying to piece a ripped up picture together. There is a blank space in the puzzle and only the pictures that are directly next to the blank space can swap with the blank space. When the puzzle is correctly completed then the user will get another clue. There will also be a riddle in my game. The user will have to answer it correctly so they can get a clue. The user will be able to choose what games they want to play by choosing where they want to explore from a main screen that gives them a choice. On the main screen they can also choose to exit the game and check a data sheet that has all the information that they found on it.

My target audience for this game is anybody who enjoys playing mystery games and needs to escape from stress. My main goal is to help these people be relieved from stress.

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:

1.3.1 General:

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

1.3.2 Options screen (main screen):

- links everything (everything starts and ends here)
- Allows the user to choose where they want to go
- Has an exit button for the user to use to leave the game
- Gives the user access to the data sheet
- holds main method

1.3.3 Data sheet screen:

- Gives the user a choice where to go
- Links the user to the data collected screen
- Must have a button that allows the user to close the screen

1.3.4 Data collected:

- Displays information about the option selected in the data sheet screen(Only displays this information if the selected task has been collected)
- Will have a back button that will take the user back to the data sheet screen when clicked
- Must have a button that allows the user to close the screen

1.3.5 Area information screen:

- Will display information about the particular area that the user selected in the options screen
- Will have a next button that will take the user to the selected game and how to play it

1.3.6 How to play screen:

- Will display the instructions of how to play information from the permanent storage file of the game that the user selected in the options screen
- Must have a button that allows the user to close the screen

1.3.7 Broken picture frames screen (riddle):

- Displays riddle
- Allows the user to input an answer
- Must have an answer button that when pressed the user input is checked
- If the answer is correct then the the task completed screen must be opened
- If the answer is wrong the input area is cleared; after 3 wrong answers the task failed screen is opened

1.3.8 Door code screen:

- Will allow the user to input three numbers that will be displayed in three text areas(an area for each number)
- When the user is done they must be able to click an answer button which will check if the user input is the correct code
- If it is the correct code then the task completed screen will be opened
- If it is the wrong code the the 3 boxes will be cleared and the user can try again
- Will link the user to the how to play and main screens

1.3.9 Music box screen:

- 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 "_" (how many "_" there will be will depend on how many letters are in the word)
- Will display the right and wrong letter guesses in their respective places
- Links the user to the how to play screen and the main screen

1.3.10 Torn up pictures screen (puzzle):

- Links the user to the how to play screen and home screen
- Allows the user to play a puzzle game
- Must have 6 buttons that have icons and will run methods that swap two pictures when pressed

1.3.11 Storyline screen:

- Must display the obtained information and storyline about the area that the user selected in the options screen
- Will have a button which will take the user to the options screen

1.3.12 Task completed screen:

- Will display that the user completed the task
- Will have a button which will open the storyline screen

1.3.13 Task failed screen:

- Displays that the user failed the task
- needs a retry button that opens the previous game that the user failed
- Will have a button that takes the user back to the options screen

1.3.14 Completed game(end) screen:

- The user can select an exit button which will exit the game
- Will have a replay button which resets all the completed games variables to false and opens the main screen

1.4 Specifications of data storage:

1.4.1 brokenPicFramesInfo:

- **Fields:** how to play, area information, and storyline
- **When are fields accessed:** when displaying area information, how to play, and the storyline screens

1.4.2 doorInfo:

- **Fields:** how to play, area information, and storyline
- **When are fields accessed:** when displaying area information, how to play, and the storyline screens

1.4.3 musicBoxInfo:

- **Fields:** how to play, area information, and storyline
- **When are fields accessed:** when displaying area information, how to play, and the storyline screens

1.4.4 tornPicsInfo:

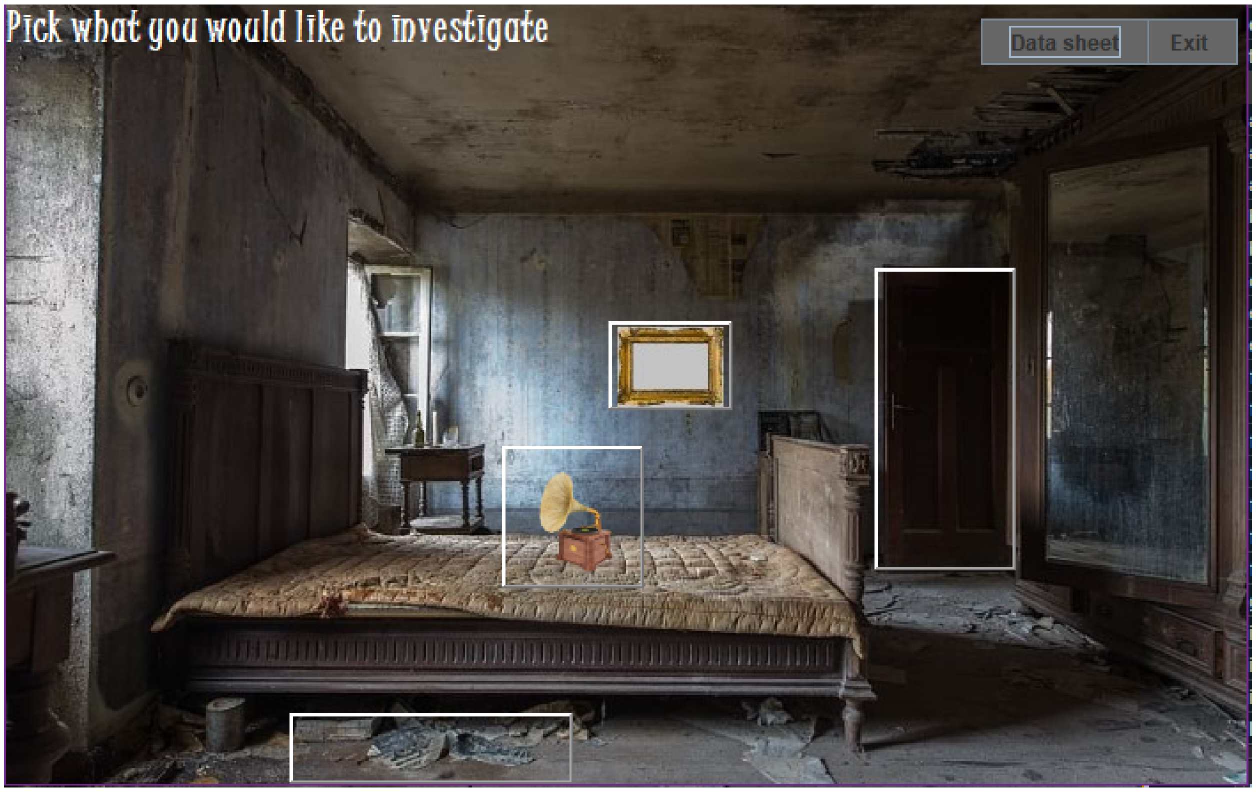
- **Fields:** how to play, area information, and storyline
- **When are fields accessed:** when displaying area information, how to play, and the storyline screens

1.4.5 completedTaskInfo:

- **Fields:** music box information, broken picture frames information, and torn up pictures information
- **When are fields accessed:** when displaying the area information screen

2.1 Interface Design:

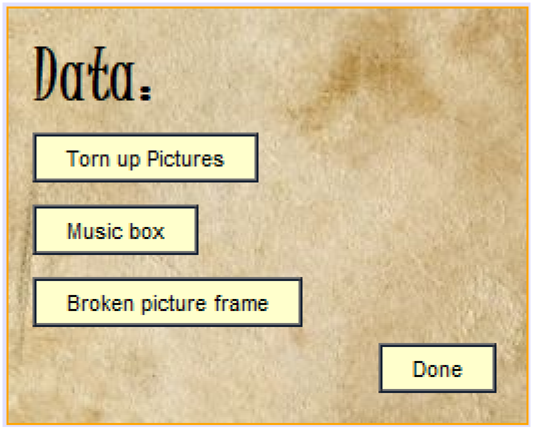
2.1.1 Main screen:



Description	The purpose of this screen is to give the user the option to choose what they want to investigate. This screen links all the other screens to it and displays the overall 'room'
Data in	- A background image is obtained from the resources folder to be presented - A music box picture is obtained from the resources folder to be presented - A picture frame image is obtained from the resources folder to be presented
Actions	**Data Sheet button** Opens the data sheet screen **Music box button** Opens the how to play screen and the music box screen **Broken picture frame button** Opens the how to play screen and the broken picture frame screen

	Door button Opens the how to play screen and the door screen **Exit button** Closes the screen (no more screens are running after this)

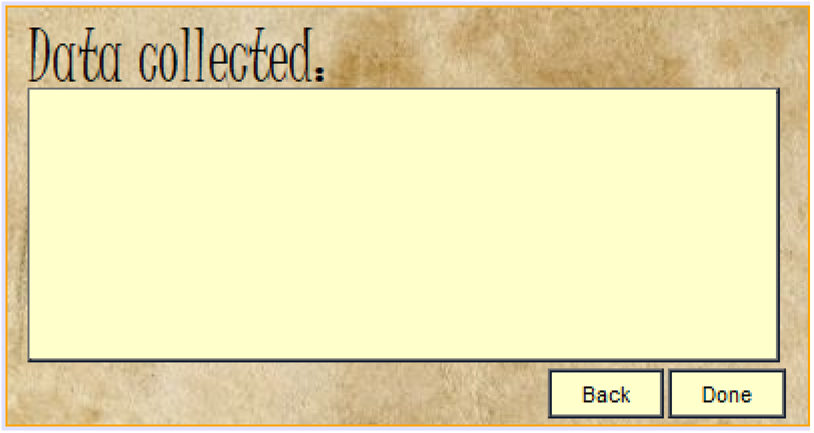
2.1.2 Data sheet screen:



Description	The purpose of this screen is to give the user a choice of what data they would like to see
Data in	- an image is retrieved from the resources folder to be used as a background
Actions	**Done button** Closes this screen **Torn up pictures button** Redirects the user to the data collected screen **Music box button** Redirects the user to the data collected screen **Broken picture frames button** Redirects the user to the data collected screen

---

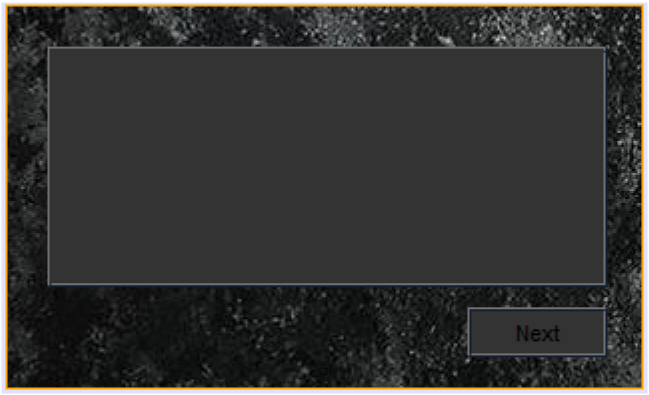
2.1.3 Data collected screen:



Description	The purpose of this screen is to display the data collected of the game they selected in the data sheet if they have completed the task
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 of the previously selected area area in the completedTasksInfo.txt text file
Actions	**Back button** Redirects the user to the data sheet screen **Done button** Closes this screen

---

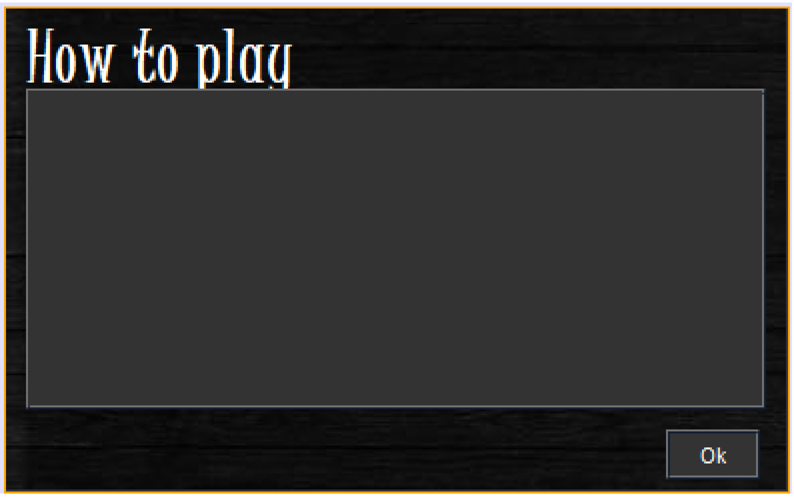
2.1.4 Area Information screen:



Description	The purpose of this screen is to display the area information of the game they selected
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 area in the brokenPicFramesInfo.txt/ doorInfo.txt /musicBoxInfo.txt /tornPicsInfo.txt text file
Actions	**Next button** Redirects the user to the how to play screen and the game screen of the option the user selected in the main screen

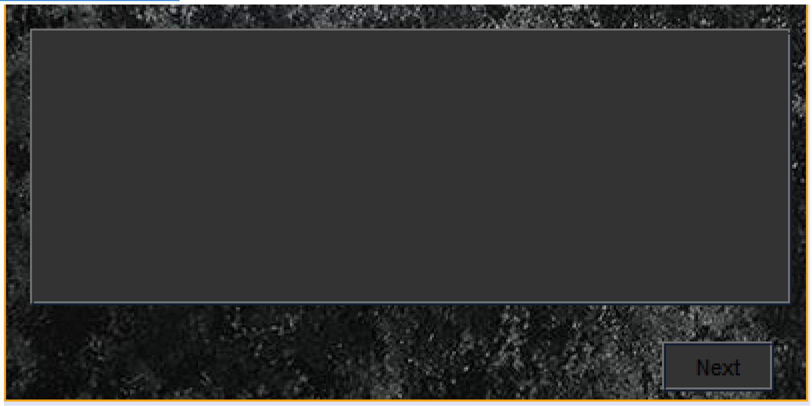
---

2.1.5 How to play screen:



Description	The purpose of this screen is to display how to play the game that the user selected
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 in the brokenPicFramesInfo.txt/ doorInfo.txt /musicBoxInfo.txt /tornPicsInfo.txt text file
Actions	**Ok button** Closes this screen

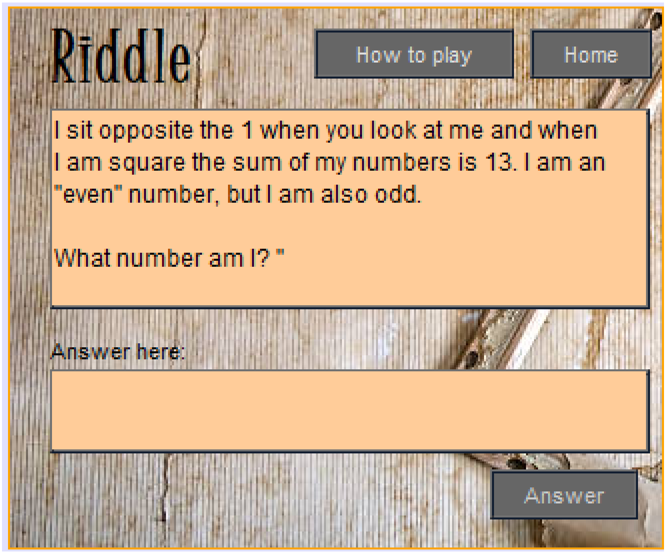
2.1.6 Storyline screen:



Description	The purpose of this screen is to display the storyline of certain 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 in the brokenPicFramesInfo.txt/ doorInfo.txt /musicBoxInfo.txt /tornPicsInfo.txt text file
Actions	**Next button** The user is redirected to the main screen; if the information displayed is from the doorInfo.txt then the user is redirected to the end screen

---

2.1.7 Riddle screen:

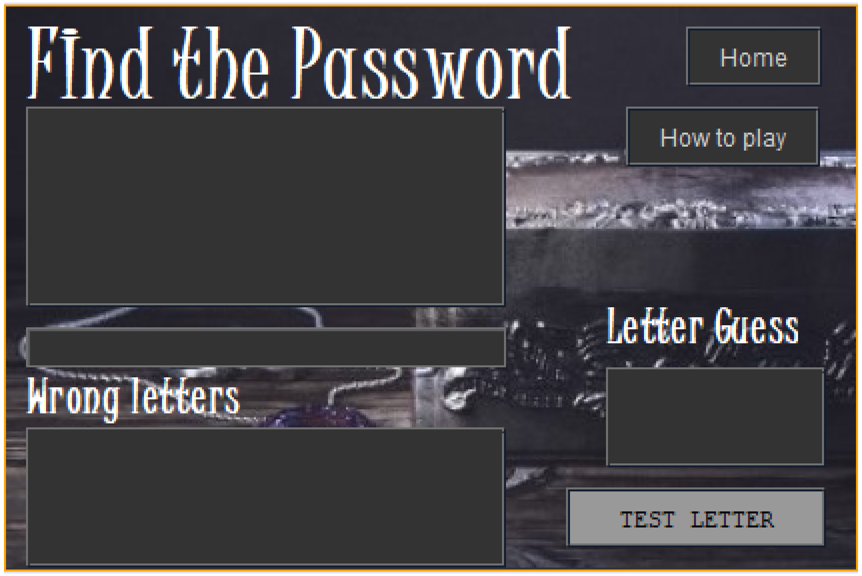


I sit opposite the 1 when you look at me and when I am square the sum of my numbers is 13. I am an "even" number, but I am also odd.

What number am I? "

Description	The purpose of this screen is to display a riddle and allows the user to give input to answer it.
Data in	- an image is retrieved from the resources folder to be used as a background
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 task completed screen. - If it is wrong, then the input area will be cleared. After 3 wrong answers the user will be redirected to the task failed screen. **How to play button** Opens the how to play screen. **Home button** Returns to the main screen.

2.1.8 Hangman screen:

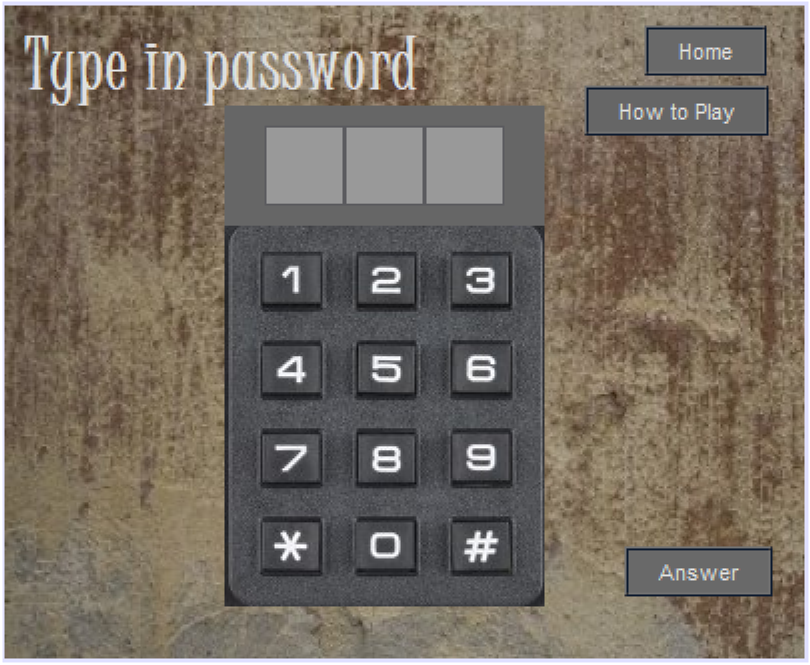


Description	The purpose of this screen is to display the hangman game. The user can interact with this screen by inputting letters and the letters guessed will be displayed in their respective text areas depending if the letter is right or wrong.
Data	- an image is retrieved from the resources folder to be used as a background

	- 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** Returns to the main screen **How to play button** Opens the how to play screen

---

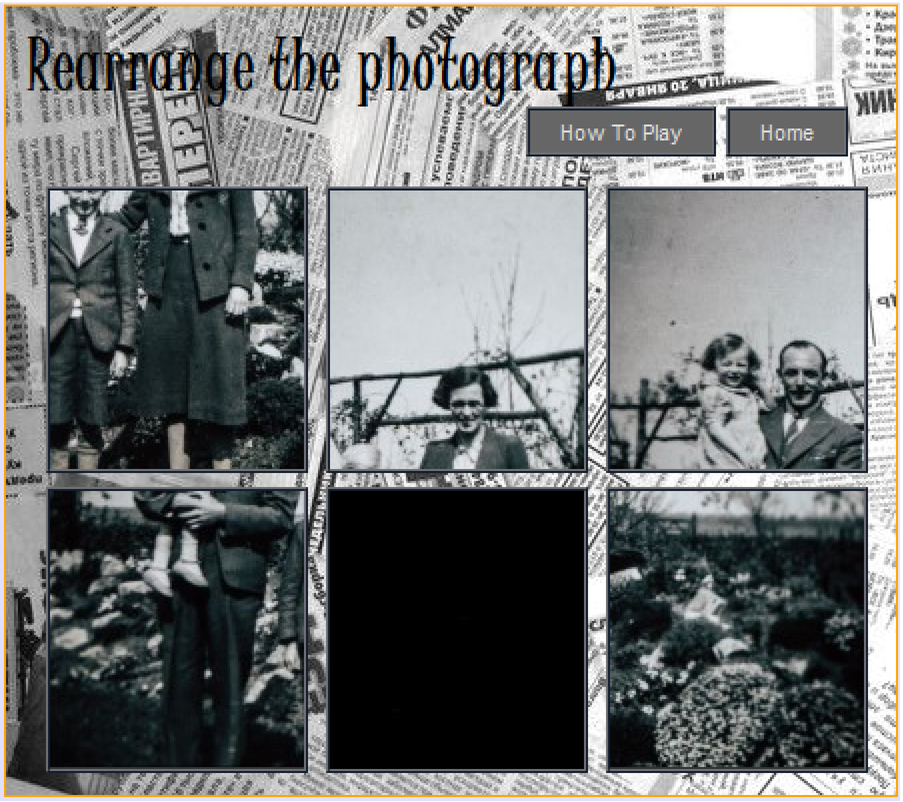
2.1.9 Door code screen:



Description	The purpose of this screen is to allow the user to input a door code.

Data in	- an image is retrieved from the resources folder to be used as a background - 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** Returns to the main screen **How to play button** Opens the how to play screen

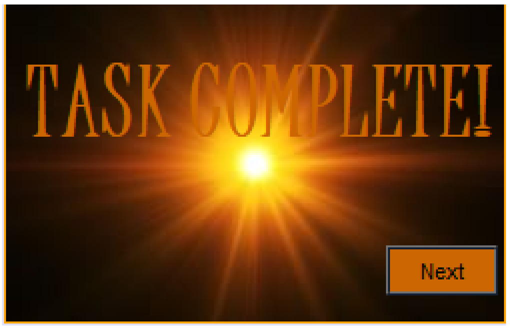
2.1.10 Puzzle screen:



Description	The purpose of this screen is to give the user an interface that they can play the puzzle on and move puzzle pieces on.

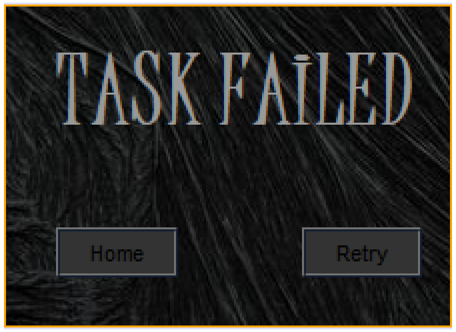
Data in	- an image is retrieved from the resources folder to be used as a background - 6 images are retrieved from the resources folder and are displayed in 6 different buttons
Actions	**Puzzle piece buttons** The puzzle piece clicked swaps with the black picture, if it is next to it. The picture order is checked to see if it the correct order - if the order is correct, them the task completed screen is opened **Home button** Returns to the main screen **How to play button** Opens the how to play screen

2.1.11 Task complete screen:



Description	The purpose of this screen is to display 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 storyline screen and closes this screen

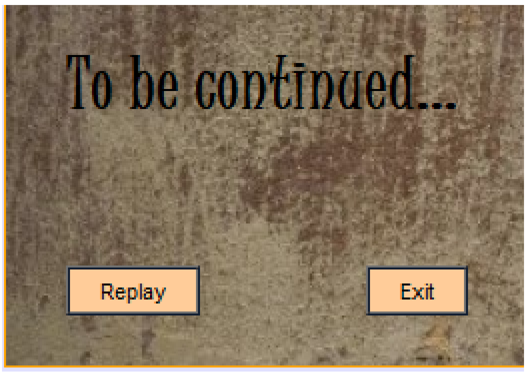
2.1.12 Task failed screen:



Description	The purpose of this screen is to display that the user failed the 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 main screen

---

2.1.13 End screen:



Description	The purpose of this screen is to display the end of the game and give the user an option to replay the game or close it
Data in	- an image is retrieved from the resources folder to be used as a background
Actions	**Replay button** Opens the options screen and resets the game **Exit button** Closes this screen

2.2 Program Flow:

2.2.1.Main screen - OptionsScreen:

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

```
OptionsScreen is disposed()
DataSheetScreen is launched
```

- When the **torn picture button** is pressed

```
OptionsScreen is disposed()
ChangingScreenMethods.tornUpPicturesOption <- true
TornPictureScreen is launched
```

- When the **music box button** is pressed

```
OptionsScreen is disposed()
ChangingScreenMethods.musicBoxOption <- true
MusicBoxScreen is launched
```

- When the **door button** is pressed

```
OptionsScreen is disposed()
ChangingScreenMethods.doorOption <- true
DoorScreen is launched
```

- When the **frame button** is pressed

```
OptionsScreen is disposed()
ChangingScreenMethods.brokenPictureFramesOption <- true
BrknPicFrameScreen is launched
```

- When the **Exit button** is pressed

```
OptionsScreen is disposed()
```

2.2.2.Data sheet - DataSheetScreen:

- When the **Done** button is pressed

```
DataSheetScreen is disposed()
```

- When the **Torn up pictures button** button is pressed

```
DataSheetScreen is disposed()
DataSheetMethods.game <- "tornUpPics"
InfoScreen is launched
```

- When the **Music box button** button is pressed

```
DataSheetScreen is disposed()
DataSheetMethods.game <- "musicBox"
InfoScreen is launched
```

- When the **Broken picture frame button** button is pressed

```
DataSheetScreen is disposed()
DataSheetMethods.game <- "brokenPicFrames"
InfoScreen is launched
```

2.2.3.Data sheet information - InfoScreen:

- When the screen is **initialised**

```
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

```
DataSheetScreen is launched
```

2.2.4.Area info - AreaInformationScreen:

- When the screen is **initialised**

```
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()
```

2.2.5.How to play - HowToPlay:

- When the screen is **initialised**

```
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()
```

2.2.6.Storyline - StorylineScreen:

- When the screen is **initialised**

```
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()
```

2.2.7.Riddle screen - BrknPicFrameScreen:

- 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
```

2.2.8.Door code - 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
```

2.2.9.Hangman screen - MusicBoxScreen:

- 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>
```

2.2.10.Picture swap game - TornPictureScreen:

- 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()
```

2.2.11.Winning screen - TaskCompletedScreen:

- When the **Next** button is pressed

```
TaskCompletedScreen is disposed()
StorylineScreen is launched
```

2.2.12.Losing screen - 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()
```

2.2.13.Ending screen - EndScreen:

- When the **Replay** button is pressed

```
Make a call to DataSheetMethods(reset)
      completedBrokenPicFrames <- false
      completedMusicBox <- false
      completedTornPics <- false
EndScreen is disposed()
OptionsScreen is launched
```

- When the **Exit** button is pressed

```
EndScreen is disposed()
```

2.3 Class Design:

ChangingScreensMethods	Description

+resetOptionVariables()
 	
+getTextfileFilepath() : String
+getAreaInfo() : String
+getHowToPlay() : String
+getStoryline() : String
+openGame()
 	
+openPreviousGame()
 	
+openNextScreen()
	Sets all the options variables(booleans) to false
	Gets the file path of a specific text file
	Gets the area info part of the text file
	Gets the how to play part of the text file
	Gets the storyline part of the text file
	Opens the game if the option variable is set to true
	Opens the game that the user was previously on
	Opens the next screen that would be needed depending on the option

DataSheetMethods	Description
+reset() +getCompletedGameData() : String	Sets all the class variables to false Gets the information of a selected option if the game is completed

DoorCodeMethods	Description

+addNumber(buttonPressed:int)
 	
+updateDisplay(num1Display:JTextField, num2Display:JTextField, num3Display:JTextField)
+reset()
+winLoseCheck()
	Adds a number to the next open space in the user's class variable
	Sets the numbers in the user's class variable to their display areas
	Resets the user's class variable to be clear
	Checks if the game has been lost or one(won = opens the winning screen; lost = reset())

HangmanMethods	Description

+reset()
 	
+getInput(inputTextArea:JTextArea) : String
+letterCheck(inputLetter:String, progressBar:JProgressBar)
	Sets all the class's class variables to their original values
	Fetches text from an input area
	Checks if the letter is right or wrong(right = adds the letter to the word display; wrong = adds the number to the wrong letters and

+updateScreen(rightAnswersTextArea:JTextArea, wrongAnswersTextArea:JTextArea , guessingTextArea: JTextArea , progressBar:JProgressBar)
+WinLoseCheck()
	increases the progress bar by 20%)
	Updates all the components on the screen with their new values
	
	Checks if the game has been won or lost(win = opens winning screen; lose = opens losing screen)

RiddleMethods	Description
+getInput(inputTextArea:JTextArea) : String +rightWrongCheck(userInput:String, inputTextArea:JTextArea)	Fetches text from an input text area Checks if the input is right or wrong

PuzzleMethods	Description

+getPic(screenStr:String) : String
 	
+getScreenNumber(screenStr:String) : int
 	
+pictureSwap(button1:JButton, button2:JButton, button1Str:String , button2Str:String)
+getBlankPicPos() : int
+framePicSwap(frameNum:int, blankPicFrame:int, frame0:JButton , frame1:JButton, frame2:JButton, frame3:JButton, frame4:JButton, frame5:JButton)
+winCheck()
 	
+reset()
	Gets the current picture on a specific jbutton
	Gets the number of a screen based on its name
	Swaps the pictures on two buttons
	
	Gets the position of the blank picture(pic2)
	Swaps only the pictures that are directly next to each other
	
	Checks if the picture order is correct; if it is correct then the task completed screen is opened
	Sets the picture order back to its original order

2.4 Secondary Storage Design

2.4.1.brokenPicFramesInfo.txt:

Format(in whole text file):

```
<Area information>
<How to play>
<storyline>
*note that each part can be multiple lines long and are separated with a “#”
```

Example:

```
I am going to the shops.
#Pick all the items needed to make cereal.
#Now I can eat a yummy breakfast.
```

2.4.2.doorInfo.txt:

Format(in whole text file):

```
<Area information>
<How to play>
<storyline>
*note that each part can be multiple lines long and are separated with a “#”
```

Example:

```
I am going to the shops.
#Pick all the items needed to make cereal.
#Now I can eat a yummy breakfast
```

2.4.3.musicBoxInfo.txt:

Format(in whole text file):

```
<Area information>
<How to play>
<storyline>
*note that each part can be multiple lines long and are separated with a “#”
```

Example:

```
I am going to the shops.
#Pick all the items needed to make cereal.
#Now I can eat a yummy breakfast
```

2.4.4.tornPicsInfo.txt:

Format(in whole text file):

```
<Area information>
<How to play>
<storyline>
*note that each part can be multiple lines long and are separated with a “#”
```

Example:

```
I am going to the shops.
#Pick all the items needed to make cereal.
#Now I can eat a yummy breakfast
```

2.4.5.completedTaskInfo.txt:

Format(in whole text file):

```
<Music box information>
<broken picture frames information>
<torn up pictures information>
*note that each part can be multiple lines long and are separated with a “#”
```

Example:

```
There is a diary  in this music box.
#This is a broken picture frame with a missing picture.
#This is an old family from long ago.
```