OCR GCE A LEVEL

COMPUTER SCIENCE PROGRAMMING PROJECT

H446-03

Name: Jacob Braidley

Candidate Number: 4512

Institution Name: St Augustine’s Catholic College

Institution Centre Number: 66643

Title of Project: Picture This!

**Table of Contents**

[Analysis 7](#_Toc103268903)

[Problem Identification and Project Outline 7](#_Toc103268904)

[Identifying the Stakeholders 7](#_Toc103268905)

[In-depth Research 8](#_Toc103268906)

[Interview Plan 8](#_Toc103268907)

[Question Layout 8](#_Toc103268908)

[Interview Review 9](#_Toc103268909)

[Research into Current Applications 9](#_Toc103268910)

[Skribbl.io 9](#_Toc103268911)

[MS Paint 11](#_Toc103268912)

[Required Features for the Solution 13](#_Toc103268913)

[Interview Plan 13](#_Toc103268914)

[Question Layout 13](#_Toc103268915)

[Interview Review 14](#_Toc103268916)

[Underpinning Knowledge – Dynamic Game Difficulty Balancing 15](#_Toc103268917)

[Table of Features for the Solution 15](#_Toc103268918)

[Table of Additional Features for the Solution 17](#_Toc103268919)

[Identifying Limitations 17](#_Toc103268920)

[Additional Features 17](#_Toc103268921)

[Features Not Included 18](#_Toc103268922)

[Solving the Problem using Computational Methods 18](#_Toc103268923)

[Thinking Abstractly 19](#_Toc103268924)

[Thinking Ahead 19](#_Toc103268925)

[Thinking Procedurally 19](#_Toc103268926)

[Top-Down Modular Structure Diagram for ‘Picture This!’ 20](#_Toc103268927)

[Thinking Logically 20](#_Toc103268928)

[Thinking Concurrently 20](#_Toc103268929)

[Summary 21](#_Toc103268930)

[Identifying Hardware and Software Requirements 21](#_Toc103268931)

[Recommended Hardware Requirements 21](#_Toc103268932)

[Developer and Player 21](#_Toc103268933)

[Recommended Software Requirements 22](#_Toc103268934)

[Developer 22](#_Toc103268935)

[Player 22](#_Toc103268936)

[Success Criteria 23](#_Toc103268937)

[Design 26](#_Toc103268938)

[Structure Diagram of the Solution 26](#_Toc103268939)

[Design of User Interfaces and Considering Usability Features 28](#_Toc103268940)

[Menu 28](#_Toc103268941)

[Drawing Phase 28](#_Toc103268942)

[Guessing Phase 29](#_Toc103268943)

[Inputs and their Validation 29](#_Toc103268944)

[Algorithms 31](#_Toc103268945)

[Independent Explanations of Important Algorithms 32](#_Toc103268946)

[Random Word Selection 32](#_Toc103268947)

[Validating Guesses 33](#_Toc103268948)

[Dynamic Game Difficulty Balancing (DGDB) 35](#_Toc103268949)

[Drawing Tools 37](#_Toc103268950)

[Overall Program Flowchart 39](#_Toc103268951)

[Identifying Test Data 41](#_Toc103268952)

[Menu Graphical User Interface 41](#_Toc103268953)

[Creating or Joining Games 42](#_Toc103268954)

[Drawing Tools 43](#_Toc103268955)

[Chat Box for Guesses and Communication 44](#_Toc103268956)

[Required Data Structures for the Solution 45](#_Toc103268957)

[Class Diagrams and Required Variables 45](#_Toc103268958)

[Main Menu Interface 46](#_Toc103268959)

[Creator Lobby Interface 46](#_Toc103268960)

[Entering Game Code Interface 47](#_Toc103268961)

[Joiner Lobby Interface 47](#_Toc103268962)

[Drawing Phase Interface 48](#_Toc103268963)

[Drawing Canvas 48](#_Toc103268964)

[Guessing Phase Interface 49](#_Toc103268965)

[Chat Box 50](#_Toc103268966)

[Countdown Timer 51](#_Toc103268967)

[Leader Board Interface 52](#_Toc103268968)

[Required Files 52](#_Toc103268969)

[Word Selection - 2player.csv, 3player.csv, 4player.csv 52](#_Toc103268970)

[Game Codes to Join Games – codes.txt 53](#_Toc103268971)

[Blacklisted Words for Profanity Filter – censor.txt 53](#_Toc103268972)

[Leader Board Scores – scores.txt 53](#_Toc103268973)

[Known Data Structures for Picture This! 54](#_Toc103268974)

[Developing Coded Solution 56](#_Toc103268975)

[Module 1: ‘Main Menu’ Interface 56](#_Toc103268976)

[Development Aim of Module 1 56](#_Toc103268977)

[Interface Frame and Layout – [02/01/2022] 56](#_Toc103268978)

[Written Code 56](#_Toc103268979)

[Testing Performed 59](#_Toc103268980)

[Button Functionality – [05/01/22] 60](#_Toc103268981)

[Written Code 60](#_Toc103268982)

[Testing Performed 63](#_Toc103268983)

[Client Feedback on Module 1 65](#_Toc103268984)

[Module 2: ‘Joiner Lobby’ Interface 65](#_Toc103268985)

[Development Aim of Module 2 65](#_Toc103268986)

[Interface Frame and Layout - [08/01/22] 65](#_Toc103268987)

[Written Code 65](#_Toc103268988)

[Testing Performed 66](#_Toc103268989)

[Timer Implementation – [09/01/22] 66](#_Toc103268990)

[Written Code 66](#_Toc103268991)

[Testing Performed 67](#_Toc103268992)

[Client Feedback on Module 2 67](#_Toc103268993)

[Module 3: ‘Creator Lobby’ Interface 68](#_Toc103268994)

[Development Aim of Module 3 68](#_Toc103268995)

[Interface Frame and Layout – [11/01/22] 68](#_Toc103268996)

[Written Code 68](#_Toc103268997)

[Testing Performed 70](#_Toc103268998)

[Button Functionality – [12/01/22] 70](#_Toc103268999)

[Written Code 70](#_Toc103269000)

[Testing Performed 71](#_Toc103269001)

[Client Feedback on Module 3 72](#_Toc103269002)

[Module 4: ‘Drawing Phase’ Interface 72](#_Toc103269003)

[Development Aim of Module 4 72](#_Toc103269004)

[Interface Frame and Layout – [14/01/22] 72](#_Toc103269005)

[Written Code 73](#_Toc103269006)

[Testing Performed 76](#_Toc103269007)

[Drawing Canvas – [17/02/22] 77](#_Toc103269008)

[Code Written 77](#_Toc103269009)

[Testing Performed 79](#_Toc103269010)

[Button Functionality – [19/02/22] 79](#_Toc103269011)

[Code Written 79](#_Toc103269012)

[Test Performed 80](#_Toc103269013)

[Timer Implementation – [21/02/22] 81](#_Toc103269014)

[Code Written 81](#_Toc103269015)

[Testing Performed 82](#_Toc103269016)

[Saving Snapshot of Drawing – [22/02/22] 83](#_Toc103269017)

[Code Written 83](#_Toc103269018)

[Testing Performed 83](#_Toc103269019)

[Client Feedback on Module 4 83](#_Toc103269020)

[Module 5: ‘Guessing Phase’ Interface 84](#_Toc103269021)

[Development Aim of Module 5 84](#_Toc103269022)

[Interface Frame and Layout – [24/02/22] 84](#_Toc103269023)

[Code Written 84](#_Toc103269024)

[Testing Performed 86](#_Toc103269025)

[Chat Box Functionality – [25/02/22] 86](#_Toc103269026)

[Code Written 86](#_Toc103269027)

[Testing Performed 86](#_Toc103269028)

[Displaying Player Drawings – [26/02/22] 87](#_Toc103269029)

[Code Written 87](#_Toc103269030)

[Testing Performed 87](#_Toc103269031)

[Timer Implementation – [27/02/22] 87](#_Toc103269032)

[Code Written 88](#_Toc103269033)

[Testing Performed 89](#_Toc103269034)

[Client Feedback on Module 5 89](#_Toc103269035)

[Module 6: ‘Leader Board’ Interface 90](#_Toc103269036)

[Development Aim of Module 6 90](#_Toc103269037)

[Interface Frame and Layout – [11/03/22] 90](#_Toc103269038)

[Code Written 90](#_Toc103269039)

[Testing Performed 92](#_Toc103269040)

[Ordering and Displaying Leader Board Data – [15/03/22] 92](#_Toc103269041)

[Code Written 92](#_Toc103269042)

[Testing Performed 93](#_Toc103269043)

[Client Feedback on Module 6 93](#_Toc103269044)

[Module 7: Essential Algorithms 93](#_Toc103269045)

[Development Aim of Module 7 93](#_Toc103269046)

[Random Word Selection (Drawing Phase) – [17/03/22] 93](#_Toc103269047)

[Code Written 94](#_Toc103269048)

[Testing Performed 96](#_Toc103269049)

[Validation of Inputs Including Profanity Filter (Guessing Phase) – [24/03/22] 96](#_Toc103269050)

[Code Written 96](#_Toc103269051)

[Testing Performed 98](#_Toc103269052)

[Scoring/Dynamic Game Difficulty Balancing (Guessing Phase/Drawing Phase) – [31/03/22] 100](#_Toc103269053)

[Code Written 100](#_Toc103269054)

[Testing Performed 102](#_Toc103269055)

[Client Feedback on Module 7 102](#_Toc103269056)

[Evaluation 103](#_Toc103269057)

[Testing and Test Data (w/ Video) 103](#_Toc103269058)

[Review of Success Criteria 107](#_Toc103269059)

[Current Issues and Further Development of *Picture This!* 110](#_Toc103269060)

[Highlighting Usability Features and Improvements 112](#_Toc103269061)

[Menu Interface 112](#_Toc103269062)

[Joiner Lobby Interface 113](#_Toc103269063)

[Creator Lobby Interface 113](#_Toc103269064)

[Drawing Phase Interface 114](#_Toc103269065)

[Guessing Phase Interface 114](#_Toc103269066)

[Leader Board Interface 115](#_Toc103269067)

[Final Client Comment 115](#_Toc103269068)

[Appendices 116](#_Toc103269069)

[Interview Files 116](#_Toc103269070)

[Code Listings 118](#_Toc103269071)

[Menu.java 118](#_Toc103269072)

[JoinerLobby.java 122](#_Toc103269073)

[CreatorLobby.java 123](#_Toc103269074)

[DrawingPhase.java 127](#_Toc103269075)

[Canvas.java 134](#_Toc103269076)

[GuessPhase.java 137](#_Toc103269077)

[LeaderBoard.java 143](#_Toc103269078)

[Bibliography 147](#_Toc103269079)

# Analysis

## Problem Identification and Project Outline

The recent events of the COVID-19 pandemic have shown that online applications are an important part of people’s lives, keeping the public connected. Even after 10 months since the end of the second UK national lockdown, people are still using these online applications, showing their importance in everyday life.

The pandemic has also brought a lot of stress to young people, including secondary school students – particularly those studying A levels. As a student myself, I understand that school can be a large source of stress and anxiety for students. The addition of COVID-19 restrictions has significantly increased stress and damaged mental health in young people, in part due to the increased isolation that has been experienced.

For many young people, playing games with friends is a great way to partly relieve this stress and alleviate the pressure that accompanies education, whilst also developing social skills. I intend to create a casual game, called “Picture This!”, that is fun for all teenagers to play. It will be online so that friends can play with each other, no matter what social restrictions they face.

*Picture This!* will be a word guessing game that involves players drawing individual pictures that are combined to create a larger word, in the form of a rebus puzzle. Players will then have to the guess the word that has been made, points being awarded according to the amount of time it takes to guess correctly.

This software will provide many students a platform to enjoy themselves and have fun with their friends. It will be a benefit to their mental health, possibly reducing stress, contributing to a better performance in school. Having fun is very important for teenagers, especially when they face such pressure from education and the recent pandemic; social interaction is an important part of development.

The next step in my analysis is to identify the stakeholders and perform research to understand the popularity that this game would have among the demographic, ensuring that it will help solve the problem of stress among them. Part of this research will be through the form of interviews and questionnaires with my client so that I fully understand what is required.

## Identifying the Stakeholders

The game will require a good knowledge of common vocabulary so would potentially restrict gameplay for some younger players who are limited in their understanding of different words. *Picture This!* will be a PC game, using mouse movements for drawing and keyboard input for typing guesses. These are fairly easy controls for any age so does not limit the demographic. Moreover, the game can be adapted at a later stage to provide touchscreen controls. This would increase the player base of the game, allowing potentially younger players to try it on their touchscreen devices.

After the game works well with private games, I aim to implement online play with a queue system. This means that the game will be playable with anyone from anywhere (assuming an internet connection is present) so is more suitable to older players who understand online responsibility. However, private games will still be a part of *Picture This!* so these younger players will be able to play with just their friends. This factor does not limit the demographic, however, allows players to make new friends whilst playing.

Considering all the previous factors mentioned, the game should be designed and developed in a way that provides the best experience to the target audience of students aged between 11 and 18. This group would benefit from the game as it would help them to decrease stress and enjoy time with their friends online. It is a suitable game for this demographic since it is relaxing and is more about having fun than winning, whilst still including an element of competition. This is perfect for a player who is looking to enjoy themselves and relax in a fun environment.

Callum Macmahon is an 18-year-old student in my computer science class who enjoys playing online videogames with his friends regularly and has a large interest in new software. He takes three challenging A Levels so sometimes needs a way to reduce his stress and have fun. I have chosen Callum as a user who represents the whole of the game’s target audience. Since he is in my computer science class, I have regular contact with Callum allowing us to easily communicate about the design of the game.

## In-depth Research

### Interview Plan

The purpose of this interview is to discover what the client would want out of my game. Through this interview, the client should provide me with the information needed to comprehend what the demographic they represent would like in a game. My aim is to understand the client’s requirements, and how I can achieve them.

The interview will focus on 3 main topics: what does the client consider makes a good game, how hard can a game be whilst still maintaining an enjoyable environment and are online games with friends more fun than single-player games.

#### Question Layout

What does the client consider makes a good game?

* Does the easy usability of the GUI make a game better?
* Are music/sound effects important to a game?
* Does an aspect of competition against others make a game more fun?
* Do you think that repetitiveness within a game makes it boring?

How hard can a game be, whilst still maintaining an enjoyable environment?

* Does a higher difficulty of games make you want to not play them?
* Do you prefer games that are more relaxing rather than challenging?
* Are easier games hard to stay engaged with?
* Should games have a development of difficulty (e.g. become harder the better you do?)

Are online games with friends considered more enjoyable than single-player games?

* What do you prefer more, online games or single-player games?
* Do you regularly play and enjoy online games?
* Do you play these games alone or with others (either friends or strangers you meet)?
* Is communication required for an online game with others to be fun?

### Interview Review

This interview was constructed to produce an idea of what the client wants the game to be like. As the developer, I needed to know what was wanted and gain an idea of how to achieve this.

After interviewing my client Callum with the questions planned above, I have a better idea about what the game needs and how to satisfy the stakeholders.

For a good game, the client believes a user interface that provides easy navigation is an important aspect. I will focus on this when designing my GUI to ensure that I create a fluid and easy to use interface. In addition to this, the client believes that a good game includes some aspect of competition and shouldn’t be repetitive without a reward. Taking this into account, the game will have a scoreboard that is updated after each round so that players can compete against others and win by the end of the game – this introduces a reward. Moreover, I will ensure that words are not repeated within a game so that repetition isn’t an issue.

For sound, the client feels that music can be used well to create a game more immersive. I agree and will implement background music into my game.

The interview also helped me discover the client’s view on difficulty. They feel that difficulty isn’t the most important aspect of a game and would rather other aspects, like those mentioned above, be focused on more to make the game fun. Nevertheless, the client responded positively to the idea of dynamic game difficulty balancing being used to ensure that the game stays at a fair difficulty, maintaining engagement with the player. This involves making the words to draw harder/easier based on the average performance of the players.

I discussed with the client their personal enjoyment of online games so that I could understand how the game would fit within the demographics’ tastes. They concluded how online games are preferred due to their interactive capabilities, playing them every day themselves, and that games are more enjoyable when they are played with friends. This fits my game profile and suggests that it will be popular among the demographic. The client also suggests that communication is part of the fun when playing with friends. In my game, I will allow the chat box to be used for guesses and also talking to others, making players feel connected.

### Research into Current Applications

To better understand how my game can be developed, I looked at other applications and identified features that were positive but also the issues that it has.

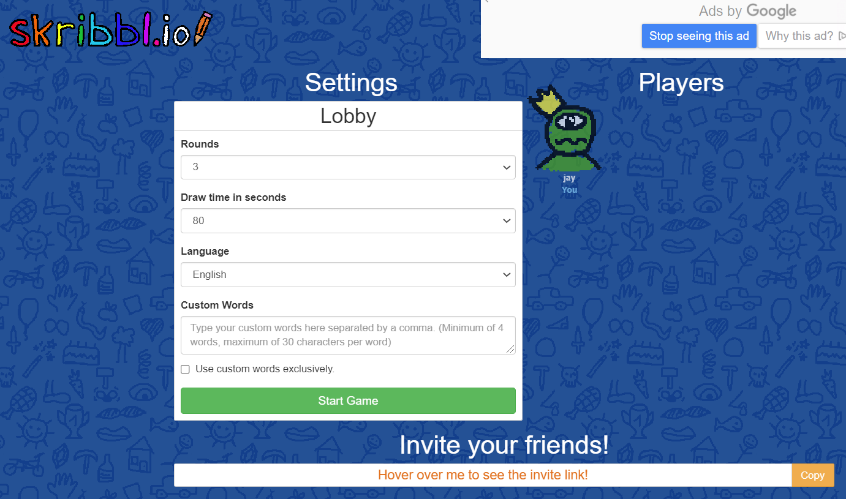
#### Skribbl.io

The first application that I researched into is a drawing game that shares similar features to my game design. It is a game where a single user chooses a word and draws it, whilst the other players try to guess the word. The role of drawer is then rotated through the players throughout rounds.

As the drawer, you have a selection of different tools including brushes of different thickness, different colours, eraser, and a fill option. These tools can be used to draw, with the mouse, on a set size canvas in the middle of the screen. After the round ends, the drawer is awarded points depending on how many people guessed the word correctly.

As the guesser, you must type in your guesses of what the drawing represents into the chat. Points are awarded depending on how fast you guess the word that is being drawn, with zero awarded if you don’t guess correctly before the round is over.

Features of the Game – Main Menu, Drawer, Guesser



All players in the current lobby.

Link that can be shared to others and used to access the private game.

Button to start the game.

Settings that can be changed by the game creator.



Amount of time left in the round along with the round number.

The chosen word that the player must draw.

The current leader board of all players in the game, including each player’s score.

Chat box where the drawer can see the guesses made by other players.

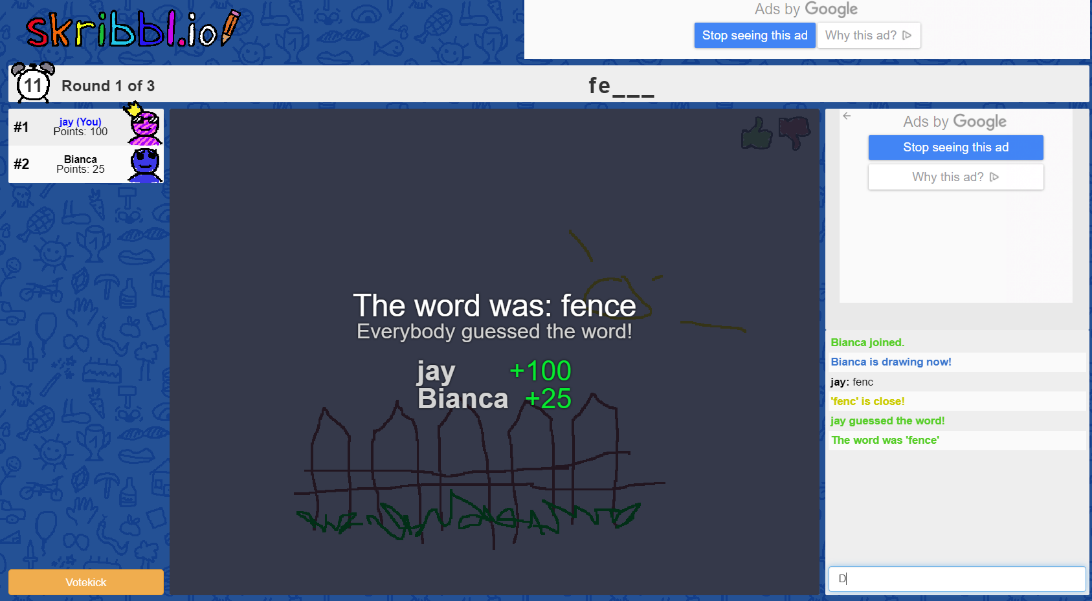
Textbox is disabled for the drawer during the round.

Colour selection toolbar.

Different brush types (Pen, Eraser, Fill), along with thicknesses.

Button to clear the canvas.

Drawing canvas where the player draws.



Word to guess, more letters are revealed after a certain amount of time.

Points achieved by each player in that round, appears after all players have guessed the word correctly, or the time has run out.

Chat box shows all previous guesses, whether a guess is close, and if the player guessed the word correctly.

Textbox for entering guesses. These guesses are then shown in the chat box.

*Skribbl.io* also provides the option for games with strangers. As a player, you are put into a queue, and subsequently entered into a random game that is in need of another player; The game then operates the same as described above. This game mode allows for players to have fun even when their friends are not available. It will increase the number of players that play the game, so is something that I will try to implement into my game design, after the private game mode is created and functions properly.

In this game, the number of rounds and time of each round is determined by the player. I like this feature and believe that giving the player this freedom will increase their enjoyment since they can adapt the game to their needs. I will discuss including this feature in my game with my client.

In *Skribbl.io*, I feel that the game idea of drawing a single word and others guessing it is too simple. I want to create a game that is more enticing to the audience, therefore, I will include the idea of rebus puzzles into the guessing phase of my game. Each player will draw a chosen word, these drawings from all players will then be combined into a rebus puzzle during a guessing phase, where all players try to guess what the phrase is.

#### MS Paint

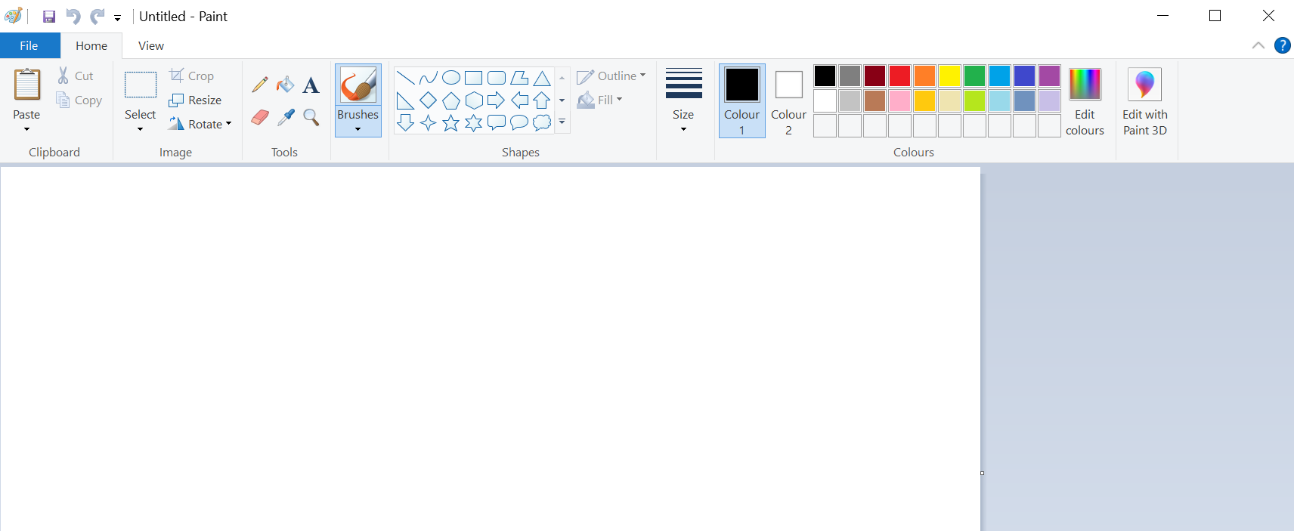
The second application that I researched is the common application *Paint* made by Microsoft. It is a default application on all computers that run Windows, with there being many copies for other operating systems.

My game is based around the drawing function, players should be able to use a variety of different brush types, colours, and sizes, etc. *Paint* is a good example of how this can be implemented within an application. It includes a large number of different tools that can be utilised by the user, something that I want my game to include.

*Paint* is used for many different purposes due to its versatility and large variety of tools. Consumers can use multiple brush types of various thickness and styles, shapes that can be styled to the current need, and different text options. The freedom of this application allows users to express themselves in any way they want.

The consumer selects a tool that they want to use, and then draw onto the canvas. They are provided with options to save and print their work, along with other features that improve the ease of use for the user.

Features of the Application



Selection for different shapes that can be used.

Different colour selection of typical colours, with the option to choose your own from a colour wheel.

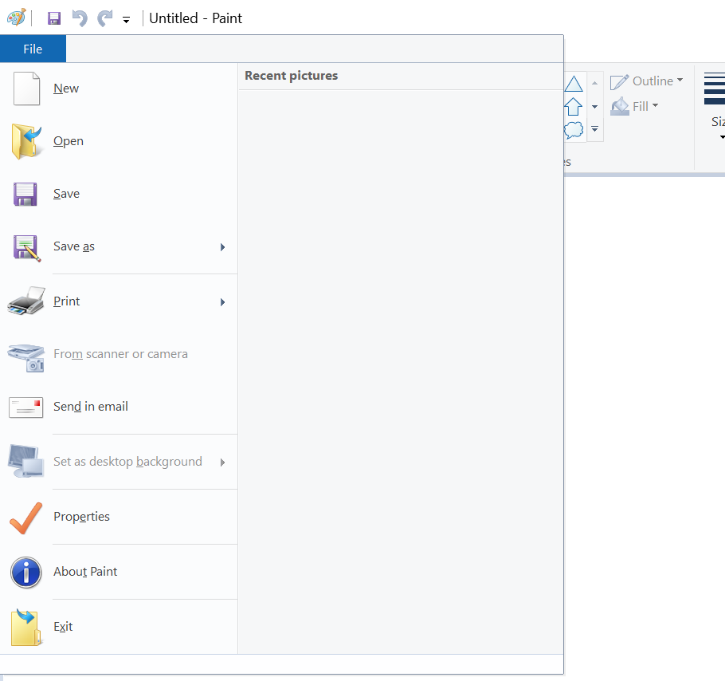
Cut, copy, and paste options.

Quick buttons for saving undo and redo.

Different tools other than the brush. E.g. Text, eraser, fill tool.

Canvas to draw on.

Drop down menu to change brush size.



New canvas file.

Open previous canvas file.

Save current canvas file.

Print current canvas file.

Attach current canvas file to email.

Exists application.

About the application.

Set properties of application.

Set current canvas drawing as desktop background.

Import image from scanner or camera.

The options that are provided to the user in *MS Paint* is a good example of what I want in my game; Multiple tools allow the player more creativity towards their drawing. I can take inspiration from this application and use it in the game design of *Picture This!*

I discussed with my client the toolbar used in the *Paint* application, and they agreed with my view that it is a good design and is easy to use for most people. In my game, I wish to implement a useful toolbar that contains a variety of different options, so the player is given a large amount of freedom to draw. To save space, I want to involve all the colour choices within one drop-down menu, something that I will discuss with my client.

*Paint* has lots of features that allow for the saving and opening of different canvas files. This is not needed for my game so I will not be implementing these options. However, I realise that the “File” tab at the top is a useful way to represent settings; I will adapt this design to fit the needs of my project, having a settings tab that can be used to set preferences.

For the development of my game, I am going to be using the Java programming language to create the client-side GUIs. After reviewing the drawing features involved in *MS Paint*, I feel that using the Swing toolkit (a class that is a part of Oracle’s Java Foundation Classes) is the best way to implement drawing features into my game.

## Required Features for the Solution

### Interview Plan

The purpose of this interview is to discover the features that the client wants to be a part of the final solution. Through this interview, I will begin to understand the more specific needs of the client, allowing me to create my set of requirements that are needed to solve the problem.

This interview will focus on the game mechanics, visuals, sounds, online capabilities, scoring system, and difficulty.

#### Question Layout

Game Mechanics:

* Should players have a limited number of guesses?
* Will players be able to change the round length?
* Will players be able to choose the number of rounds?

Visuals:

* For the drawing and guessing phase, should separate GUIs be used?
* Should the GUIs be a minimalist design, or more complex?

Sounds:

* Is background music important for the client?
* Should there be additional sound effects (e.g. when guessing correctly)?
* Will players be able to choose their own music?

Online Capabilities:

* Should both private and public games be available to the player?
* How should private games be accessible to players (e.g. through an email link)?
* Does the client want the possibility of friends in the game?

Scoring System:

* How should points be awarded to players?
* Will players be able to vote for their favourite drawing?
* Should the leader board be shown constantly, or only after each round?

Difficulty:

* Will the difficulty of the game change, depending on all the player’s performances?
* Is using dynamic game difficulty balancing an appropriate way of doing this?

### Interview Review

I carried out this interview, not only to understand the features that the client wants in the solution, but also to get the client’s opinion on my ideas as a developer.

After concluding the interview, I can create a list of required features that the client and I believe fit into the scope of the solution. In addition, since there is a time restraint to my project and I am currently studying two other A Level subjects, I can create a list of additional features that are not necessary for the game to be a finished solution but can be implemented if all the other features are achieved.

In terms of game mechanics, the interview clarified that the client wants players to be able to customise the number of rounds that are played, and the time length of the rounds. Also, the client initially doesn’t want players to have a limited number of guesses. However, they want players to have freedom over how they play so this is a feature that can be implemented at a later date.

For the visuals of *Picture This!*, the GUIs should be a minimalistic design with separate windows used for the ‘drawing’ and ‘guessing’ phase. The GUI should demonstrate an easy-to-read style that allows for seamless navigation by the user. It should be intuitive and not require instructions on how to use it.

When considering the sounds used in my game, the client believes that jovial background music will help create a friendly environment surrounding the game. In addition to this music, there should be sound effects used for multiple functions within the game, for example when the round ends or a player guesses the right answer. These sound bites can be used to help the user identify new sections of the game. I also discussed with the client whether they would want players to be able to choose their own music for the background. Since the client wants to emphasise the freedom that the players are given, he said that this could be an additional feature.

This interview allowed me to fully understand the client’s opinion on the online capabilities of my game. Private games are the priority for *Picture This!*, to ensure that the aspect of playing with friends is achieved. These games should be joinable by entering a code into a search bar, and once the code is validated then the player joins the game. Nevertheless, public games are an additional feature that can be added at a later date.

For the system of scoring within my game, the client wants players to be awarded points based on the time it takes for them to guess the drawings correctly. This means a player will be awarded more points if they take less time to guess correctly. Furthermore, the client wants the leader board to only be displayed after each round, and then again at the end of the match; This will assist in the simplicity of the GUI design.

My game is based around the idea of difficulty changing based on the performance of the players. In this interview, I discussed with the client the implementation of this in my game, and they agree that it is a good way of keeping the game exciting and preventing it feel repetitive.

As the developer, I have identified a process called Dynamic Game Difficulty Balancing (DGDB) that I can use to achieve this in my game. In the next sub-section, I explain this process in more detail.

#### Underpinning Knowledge – Dynamic Game Difficulty Balancing

Typically, games always had a linear development in difficulty with many modern games still adopting this trend. This meant that the difficulty could be controlled through steps such as levels or checkpoints where after each step, the game became harder; The difficulty of the first level would either be predetermined or chosen by the player from multiple options.

Changing the difficulty of a game through this method was reliant on the user starting out with little knowledge but getting better as the game progressed as they obtain more experience of how it works. However, this doesn’t work for all games since, firstly not every game has distinguishable steps where the difficulty can be set to increase, but also some games don’t follow the pattern where the player learns more throughout the game.

As the technology of computer games progressed, games were made where changes to the difficulty had to be made in real-time so that the player wasn’t becoming bored, if the game was too easy, or frustrated, if it was too challenging. Evidently, this cannot be achieved through the linear pattern that most previous games followed, so a new method was devised. Dynamic Game Difficulty Balancing (DGDB) helps to prevent the player becoming uninterested with a game due to a difficulty that does not match their skill level. It does this through changing the difficulty (through either parameters, scenarios, or behaviours) in real-time based on the player’s current performance.

In the context of my game, I wish to implement a simple version of the DGDB process so that the players do not become bored or frustrated. The parameter that I will change is the difficulty of the words that the players must draw, where words can be in multiple different difficulty categories. For example, the word “memory” is harder to draw than the word “apple”. If, on average, the players are performing well and guessing correctly in a fast time, then they will be provided with harder words.

### Table of Features for the Solution

Below is a list of the features that I will implement into my game.

|  |  |
| --- | --- |
| Feature | Explanation/Justification |
| Menu | This allows players to create a game, enter a code to join a game, enter their username, exit the application, and potentially access a settings tab (additional feature, see below). |
| Private Games | My game will focus on private games that can be played online with friends. One player will create a game, selecting different options about the game (see Game Lobby below), with others joining by entering a unique game code into a textbox in the main menu. |
| Validation of Codes | The game codes being entered should be validated to ensure they are in the correct format and also a valid code. After this validation, the players can enter the lobby for that game. |
| Game Lobby | After creating/joining a game, players are taken to a GUI where they are connected to the server before starting the game. The creator of the game can choose the settings of the game here (e.g. round length, number of rounds). It is important to note that the round length will be a combined time for both the drawing and guessing phases. |
| Separate GUIs | Separate GUIs will be used for each new page (i.e. menu, drawing phase, and guessing phase) in order to make the game seem more professional. It will also make the game easier to follow since each part will be distinguishable from the previous. |
| Drawing Phase | Each player will be assigned a different word of a certain difficulty, which they will then draw. The word is chosen at random. This round lasts a certain amount of time specified by the creator of each game, otherwise is set to a default value. |
| Canvas | A blank canvas will be provided to draw on, where players will draw their word. |
| Brush Tool | Allows the player to draw lines that can be used for drawing the word. I will be using the Swing toolkit, which allows me to implement this. |
| Different Colours of Brushes | Allowing players to use different colours for their drawings provides more freedom to how they play. It encourages more creativity in their drawings, making the game overall more enjoyable. |
| Different Sizes of Brushes | Allowing players to use different sizes of brushes provides more freedom to how they play. It encourages more creativity in their drawings, making the game overall more enjoyable. |
| Eraser Tool | The use of an eraser will allow players to remove parts of their drawing without having to start over, removing an element of frustration. An eraser can be implemented in my game by either: clearing the canvas on the area that the cursor is or drawing over with a white brush colour since the background of the canvas is white. |
| Word to Draw is Displayed | The word for players to draw is displayed at the top of the screen so that it is not forgotten. |
| Guessing Phase | Each player’s drawings get compiled into a larger image. Players then try to guess what the larger image represents, entering their guesses into the chat box. For example, the words “Bee”, “Four”, and “Hand” would be compiled into a single image that represents the word “Beforehand”. This round lasts a certain amount of time specified by the creator of each game, otherwise is set to a default value. |
| Chat Box | Players enter their guesses into the chat box, where the last x number of guesses is displayed as a chat history. When a correct guess is entered, the player will be notified in the chat box. This chat box can also be used for communicating with other players which would require the addition of a profanity filter to the chat box, ensuring no inappropriate language is sent to other players. |
| Scoring System | Players are awarded points based on the amount of time they take to guess what the word represents. This can be implemented through a variety of different ways, for example, using time boundaries such as p10 points for guessing within 30 seconds. However, I want to create a more accurate scoring system so will include this within my Design section. |
| Leader Board | The leader board is updated and displayed after each round so that players can see where they are in terms of score, it will also be displayed at the end of the game. This adds a competitive nature to the game, whilst still maintaining a casual fun feel. |
| Background Music | Background music helps create the feel of a polished game. The music in my game will be a jovial tune in order to further emphasise the purpose of destressing whilst playing my game. This music will play in the background of all games (however there will be an option to turn it off). |
| Sound Effects | Adding sound effects to my game will make the game more immersive. It also makes the game more polished, creating a feel of professionalism. These sound effects will be for actions such as guessing the word correctly and the end of rounds. |
| Dynamic Game Difficulty Balancing | My game is designed around the idea of developing the difficulty throughout the game by analysing the performance of the players on an average. In my game, I will use DGDB to change the difficulty through different levels of words. |

### Table of Additional Features for the Solution

Due to a time limit on this project, I have created a list of additional features that I will implement into my game.

|  |  |
| --- | --- |
| Feature | Explanation/Justification |
| Settings Tab | Allows players to choose their volume level and enable/disable music and sound effects. This will be accessed through the menu GUI. |
| Language Support | This feature will allow for the playing of my game in other countries that speak other languages. It could use a translator to convert words into the chosen language whilst playing, allowing players of different languages to play with each other. |
| Touchscreen Capabilities | Touchscreen would enable players to use their touchscreen devices to play. This would increase the number of players who play my game since more people will have a device capable of running it. |
| Limited Player Guesses | At the start of the game, the player who created the lobby can choose whether players have a limited number of guesses. This can be achieved through a counter on the client-side. |
| Public Games | Public games can be developed to allow players to play with anyone who is also searching for a game. I will have to develop a queue system that loads players into a game efficiently. |

## Identifying Limitations

### Additional Features

The additional features listed above were not included in the initial design, but as features that can be added at a later date. This is due to limitations that make the features harder to implement, with the features not being as important to my game. The limitations for each additional feature are as follow.

|  |  |
| --- | --- |
| Additional Feature | Limitation |
| Settings Tab | I have limited time to complete this project so am not including this in the initial design since it is not seen as important by the client. |
| Language Support | This would include using a real-time translator to translate each word as it is sent to the player. In addition, guesses will be in the language of the player and will need to be translated into English in order for the validation of the guess. This would increase the time it takes for guesses to register, decreasing the performance of *Picture This!*. |
| Touchscreen Capabilities | This would require the source code to be altered so that touchscreen can be used, requiring me to edit all of my code. This will take a long time so is a feature that should be implemented at a later date, after all other initial features are in my game. |
| Limited Player Guesses | Although this is a good feature, due to the time restraint on my project I would rather use my time to develop aspects of my game that the client feels are more important. |
| Public Games | Public games would require me, as a developer, to create a queue system that load balances players into a game. I would need to ensure that I have a suitable server and a way for players to join a queue for a game. For my game, the client prefers for games to be private and encourages playing with friends, so public games are an additional feature due to the workload I would need to put in. |

### Features Not Included

In addition, there are features that were not included in the initial design of my game and also not as additional features. This is due to limitations that make the feature unsuitable for my game. The limitations for each feature that was concluded to not be suitable for my game are as follow.

|  |  |
| --- | --- |
| Features Not Included | Limitation/Reason |
| Players Choose Own Background Music | This feature would mean that players could select their own music to be played in the background. This would require using an external API for an application that has a large music database such as Spotify or Youtube. Since I will be using third party software for this, the issue of copyright is prevalent. This makes the feature hard to deal with since it could cause copyright issues. |
| Automatic Connection to Server | This would allow clients to automatically connect to the server as soon as they open the application. However, I do not have access to a static IP address so this will change regularly. This means that the players will have to enter the IP for the server each time they want to play. |
| Additional Scoring | This feature would allow players to vote for their favourite drawing, at the end of the guessing phase. The player with the highest votes on their drawing would be awarded additional bonus points. This feature is not to be included in the game, however, due to the client claiming that it would distract players from the aim of the game. The client wants players to focus on guessing the correct word more than voting for the best drawing. |

## Solving the Problem using Computational Methods

Computational methods are important to consider when developing a solution; They help the solution become easier to design and create. *Picture This!* is a solution suitable to be solved through computational methods since it is a game that experiences scenarios involving executing multiple sub processes when handling clients on the server, and drawings that are heavily abstracted from reality.

### Thinking Abstractly

Abstraction is useful in the design of my game since I will need to know what to include and how representative of the real world my game is.

In my game, the third dimension will be removed to create a 2D plane where drawing tools can be used to draw on a blank canvas. It is an extreme abstraction where only the blank canvas, and the tools available to the player, are visible.

These drawing tools in my game will be an abstraction of real life since they only need to be a simplified representation. There are no real-life limitations on the tools, such as a limited amount of ink and the fact that the brush tool will not need to show the actual texture of a paintbrush, but only draw in a solid colour on the canvas. Moreover, the colour and sizes of the tools in *Picture This!* will be able to be changed easily to accommodate more freedom for the player. This is unrealistic of the real world, demonstrating how abstraction is used within my project.

Alongside the drawing features, abstraction can be used effectively when considering the sounds of my game. All sounds that would occur, in real life when painting, are removed since they are unnecessary to the game. Instead, only sound effects that indicate events to the user are included.

### Thinking Ahead

Every solution to a problem will require the developer to think ahead about how they can create their software. It is an important aspect of design and programming. An example of thinking ahead is the inputs and outputs of my game. These must be considered so that I can understand how the players can interact with the software.

In *Picture This!*, the player will use the mouse to complete their drawings (touchscreen may be used as well, if this additional feature is added later in development). They will also use their keyboard in order to type their guesses into the chat box. Again, touchscreen can be used here if I am able to add this capability.

There are certain preconditions that must be considered before developing the coded solution to a problem. For my game, as the developer, I do not know what the player’s level of understanding of the English language is. This is a precondition that must be taken into account; I should ensure that the words I use in my game are easily understandable to everyone who has a simple knowledge of English.

Code reuse is an important part of programming so is vital to think about when analysing computational methods. For my project, I will be using multiple sub-routines during development that I will only have to program once but can reuse them throughout. In addition, I will be using libraries for different processes, for example randomness when selecting a random word. These libraries are collections of code that are pre-developed and pre-tested so I can be sure that they are error free.

### Thinking Procedurally

With all problems, breaking it down into smaller, more manageable problems make the solution easier to develop. It will also increase the testing capabilities of my software, resulting in more efficient debugging. This is why procedural thinking is important for the development of software.

My game can be broken down into smaller aspects to consider, and by doing so it will make the development of the coded solution easier. By using a top-down modular design, I can begin to structure my solution as different sub-tasks that can be completed individually.

#### Top-Down Modular Structure Diagram for ‘Picture This!’

Diagram

Description automatically generated

This diagram shows how my game can be broken down into smaller sections that will be easier to develop and test individually. Once tested and debugged, they can be implemented into the final code. Thinking procedurally will lead to a more efficient development lifecycle of my software.

### Thinking Logically

Computers only make logical decisions and perform calculations; therefore, it is important to see how my game can be solved through logical computation. This is true, not only for the design of *Picture This!*, but also the development through programming. Every time a branch in code is approached, there is a logical decision that should be made. For example, the program will check whether a guess, entered by a player, is correct or incorrect and will pursue different branches depending on the logical outcome. No programs of this scale can exist without multiple logical decisions, emphasising the importance of this analysis.

There are multiple aspects of my game that require logical decisions throughout, including game code validation and checking player’s guesses. Through thinking logically, I have been able to identify points in my program where I will have logical decisions, further helping me understand the structure that my code development will have to take.

### Thinking Concurrently

My game will require multiple processes to occur at the same time, achievable through concurrency. However, concurrency itself does not allow for different tasks to be executed simultaneously, but only performing individual tasks in a small time slice at a fast rate. Therefore, I will use threading – an implementation of parallelism – that will allow my program to execute multiple processes simultaneously.

A clear example of where parallelism will be utilised in my game is when implementing the background music. This music will have to play continuously whilst all other actions and processes are being executed. Using threading, I can create a thread where the background music is played in a loop until a certain event (e.g. the end of the game). This allows the code of different processes to be separated, meaning the easier development, management, and testing of said code. In addition, code in these separate threads will not interfere with each other.

### Summary

All of the previous factors show that the solution to my project can be created through the use of computational methods. I have used the methods to construct an idea of how my project can be developed in an organised way.

## Identifying Hardware and Software Requirements

### Recommended Hardware Requirements

#### Developer and Player

|  |  |
| --- | --- |
| Requirement | Explanation |
| Keyboard | This peripheral will be needed by the player to type guesses of the what the drawings represent into the chat box. It is also required by me, the developer, to program the solution. |
| Mouse | Players will need to use this peripheral in order to direct through the GUI, and also draw their image. |
| 2D Display  (e.g. Monitor) | This peripheral will be used to display an image feed of the current game state to the player, allowing the player to view the current situation of the game in a real-time environment. |
| Speakers/Headphones | *Picture This!* will include audio in the form of music and sound effects. This means that audio output devices are required in order for the player to hear such audio. Speakers and headphones are the most common audio output devices; however, any other audio output device will work. |
| 500 Mb of Secondary Storage | This storage will be used to store the completed game program as an executable (.exe) file. It is not an extremely large project so the file will not require a large amount of secondary storage. |
| 1GHz CPU | All computers on the market will have a Central Processing Unit (CPU) with a speed above this, so will be sufficient enough to run my game. |
| 1GB RAM | All computers on the market will have Random Access Memory (RAM) with a size that is greater this, so will be sufficient enough to run my game. |
| Network Interface Card (NIC) | *Picture This!* is an online game so will require hardware that allows for the connection to a network. A Network Interface Card allows the user to connect to a network that the game will be held on. This hardware includes the NIC’s wireless variant, the Wireless Network Interface Card (WNIC) that allows users to connect to a network wirelessly. |
| Internet Access Point  (e.g. Router) | An online game requires connection to a network, so an Internet Access Point is needed for this to be possible. These access points can include routers, hubs, or switches. The player must have one of these hardware devices if they want to play *Picture This!* due to the network requirements. |

### Recommended Software Requirements

#### Developer

|  |  |
| --- | --- |
| Requirement | Explanation |
| Windows 10 Home/Pro Operating System | I will be using this operating system to complete my project. Windows has been the main operating system that I have used since first using a computer, so I am familiar with how it can be used. This means that it makes sense for me to use this operating system for my project. |
| Visual Studio Code | I am using this IDE due to the customisable ability that is provided to the developer. The extensions marketplace allows me to install certain features that assist me in my development of the solution. For example, I can install IntelliSense for Java which allows for the easier development of code through predictive code completion. In addition, I am familiar to the way Visual Studio Code works so using it will be easier than using a different IDE. |
| Oracle’s JavaTM Programming Language’s JDK | Java is a programming language based around the Object Orientated Programming (OOP) approach. It uses classes and objects, allowing for easier reusability of code. This suits my program well since I will have to repeat sections of my code for the different rounds in my game. |
| Java Swing | This is a lightweight toolkit – part of Oracle’s Java Foundation Classes (JFC) and a successor of the Abstract Window Toolkit (AWT) – that is used for creating GUIs in Java. I have chosen Java Swing as the way to make my GUIs as it is a simplistic, yet sophisticated way of programming interfaces within Java. |
| Java AWT | Java AWT (Abstract Window Toolkit) is an API to develop GUI or window-based applications in java. I am using it alongside the Swing library so that I can make the most out of my Graphical User Interface development. |

#### Player

|  |  |
| --- | --- |
| Requirement | Explanation |
| Windows 10 Home/Pro Operating System | My game will be developed on Windows 10, making this operating system the easiest OS to run the program on. Windows 10 is the most common operating system, running on the most devices compared to any other OS. This means that the people playing my game are most likely to play on Windows 10. |
| Any other Operating System | Since Java is a platform-independent language, it can be compiled and ran on any operating system. This means that the players can run my program on whichever operating system that they have, as long as the correct version of my game is downloaded. |
| Device Drivers | Drivers will be required for the peripherals to be used in my game. This includes drivers for audio, keyboard, and mouse. |
| *Picture This!* | The actual executable of my game will have to be installed in order for it to be played. This will be a single ‘.exe’ file that the user can download – versions for other operating systems can easily be added at a later date since Java is a programming language that can be compiled once and is then compatible on any device that has the Java Virtual Machine (JVM) installed, regardless of operating system. |

## Success Criteria

|  |  |  |
| --- | --- | --- |
| No. | Criteria | Explanation/Justification |
| 1. | Main menu that provides the user with different actions | This is the first GUI that the user will see. It shows the title of the game, and two buttons: ‘Create new game’ and ‘Join existing game’. In addition, there will be a ‘Quit’ button that allows the player to exit the game. It will also include the additional feature of a settings tab if that feature is implemented. |
| 2. | ‘Create new game’ button | This button provides users the option to create a new game that their friends can join. It will redirect the user to a new GUI where they can start to select their options for the game. |
| 3. | Round length chosen by the player | This is a parameter of the game, set by the user who is creating the game. They can choose this freely, however, it will have a maximum and minimum value in order to keep the game enjoyable within the eyes of the client. The time given will be a combined time to include both the drawing and guessing phases in each round: it will be split evenly between the two. |
| 4. | Number of rounds chosen by the player | This is another parameter of the game that is set by the user who creates the game. It will also have a maximum and minimum value. |
| 5. | New game thread on the server created for each new game | Each time a game is created, a new thread on the server is created so that players can join this game, and multiple games can run on the server at any time. Each game will be assigned a game code so that players can specify which game they want to join. This code is given to the creator to share. |
| 6. | Lobbies for players whilst waiting for the game to start | These are to different GUIs, one for the creator of the game and the other for the players joining the game. The ‘Creator lobby’ interface will allow the player to select settings that are wanted for the proceeding game (see **Success Criteria No. 3** and **4** above). The ‘Joiner lobby’ interface will simply be a GUI that the players will wait at until the creator starts the game. |
| 7. | ‘Join existing game’ button | This button, located on the Main menu, is used for when players want to join a game that already exists. It will redirect the user to a new GUI where they have the option to enter a new game code. |
| 8. | Game code entered and validated | This is where the player enters an existing game code that corresponds to a game that has been created by another player. Once entered and submitted, the code is validated. |
| 9. | Connects user to game thread on the server with corresponding game code | After the game code is entered and validated, the player is connected to the thread on the server that corresponds to the game. The player is then redirected to the game lobby GUI. |
| 10. | Random word assigned to each player | Depending on the number of players (2 - 4) and the difficulty determined, a random word is chosen from a certain file of predetermined words. This word can be split into smaller words, the amount of which is the same as the number of players. These smaller words are assigned to each player at the start of the drawing phase. |
| 11. | Players can draw with a brush tool | The brush tool allows users to paint lines with the mouse. The brush tool will change the colour of the canvas, where the mouse cursor is located, to the colour selected (see below). |
| 12. | Players can use different colours of the brush tool | The player will be able to select different colours for the brush tool, providing more creativity towards their drawings. |
| 13. | Players can use different thicknesses of the brush tool | The player will be able to select different thicknesses for the brush tool, providing more creativity towards their drawings. |
| 14. | Players can remove their drawings with an erase tool | The eraser tool is used to remove any drawings on the canvas, where the mouse cursor is located. This will work by changing the colour to white, instead of actually removing the drawings. |
| 15. | The drawing canvas can be cleared using a ‘clear canvas’ button | This button allows for players to completely clear their canvas drawing with a single click of a button. This is if the player wants to redo their drawing or approach the drawing from another angle. |
| 16. | After the drawing phase, all the drawings are combined to create the larger word | Each of the player’s drawings are combined to create a larger image of all drawings, which represents the original longer word. This image is displayed to all players. |
| 17. | A chat box allows players to enter guesses and communicate with a working profanity filter | Players are to guess this word and enter their guesses into the chat box where the guess will be checked. This chat box can also be used to communicate with other players, helping the game’s inclusive feel – however, requires a profanity filter. |
| 18. | After the guessing phase, player scores are updated | After each round, the score for each player is calculated by taking into account the time it took them to guess the word correctly. The players current score is then incremented by the score they got in that round. |
| 19. | A leader board is updated and displayed | After the guessing phase, the new player scores are checked, and the leader board updated accordingly. The leader board is then displayed in a separate GUI after each round. |
| 20. | During the game, Dynamic Game Difficulty Balancing (DGDB) is used | DGDB is used so that engagement with players is maintained. Based on the average performance of all players, the difficulty of the words will either increase or increase. Changing the difficulty keeps the players entertained but not frustrated. |
| 21. | Music plays in the background in a loop | Jovial background music is played on a loop in order to create a more friendly and casual feeling towards *Picture This!* |
| 22. | Sound effect for the start of each phase | This sound effect is used to notify the player of the start of the phase (whether it be the drawing or guessing phase). Sound effects also improve the professional feel of a game. |
| 23. | Sound effect for guessing word correctly | This sound effect is used to notify the player of the guessing the word correctly. Sound effects also improve the professional feel of a game. |
| 24. | Sound effect for the end of each phase | This sound effect is used to notify the player of the end of the phase (whether it be the drawing or guessing phase). Sound effects also improve the professional feel of a game. |
| 25. | Sound effect for results at the end of the game | This sound effect is used to notify the player of the end of the game results. This is where the leader board is displayed. Sound effects also improve the professional feel of a game. |

# Design

## Structure Diagram of the Solution

Diagram

Description automatically generated

This diagram shows the three main sections of my game (this excludes the server and focuses on the client-side experience). Each branch is a separate part that has been broken down into smaller sub-routines that are assigned to different functions in *Picture This!*, making the development of my game easier and the code more organised. Through this diagram, I can see where I need to create certain classes and functions to perform certain tasks.

The ‘Menu’ branch contains two sub-branches; ‘Create Game’ contains all the actions associated with creating a game, and ‘Join Game’ shows which actions are completed by the player when joining a pre-existing game. Inside the ‘Create Game’ sub-branch, customisation features are included that allow the player to select how their game will play out. These can be programmed individually from each other, being implemented into the game after testing individually. There is also an action that connects the client to the server – another feature that will be developed and tested in a separate environment before being used in my game. Included in the ‘Join Game’ sub-branch is the action of entering a game code that is then validated. This problem can be tackled as an individual problem where a code is entered and validated against multiple conditions. After testing different codes to see the required outputs, it can be implemented into the ‘Join Game’ function. The action of connecting the client to the server is also present here.

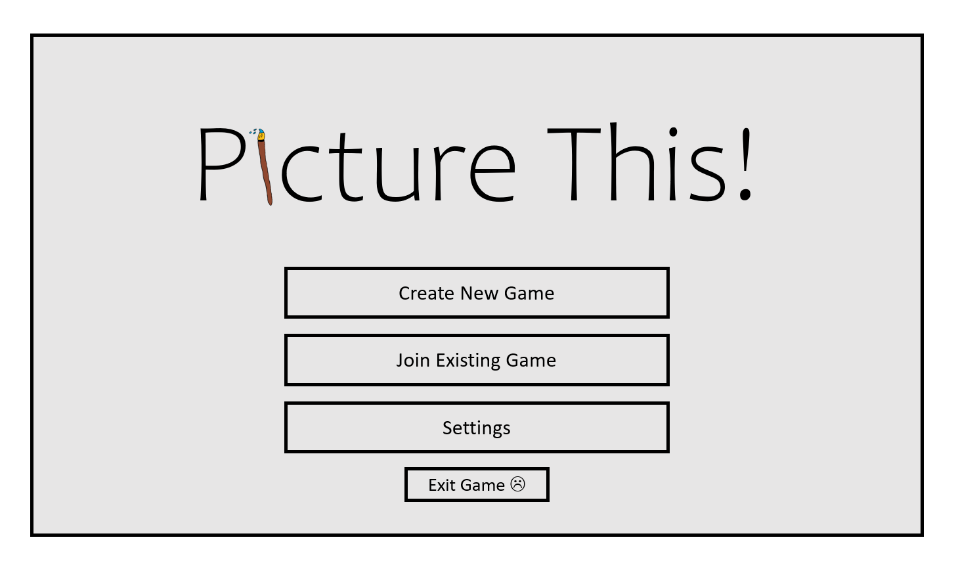
The ‘Game’ branch can be identified as having two sub-branches that represent the two phases of the gameplay of *Picture This!* – the drawing phase and guessing phase. For each phase, there are multiple problems that I must overcome so by creating this diagram, it makes the development of these phases more controllable. Inside the ‘Drawing Phase’ sub-branch, the first action is the random selection of a word from a file containing many different options. This word, although being a word by itself, can be a part of the larger word (e.g. the word ‘hand’ is a part of the word ‘beforehand’). By giving the other players words that are also part of this larger word, the drawings of these words can be combined to create a drawing that represents the larger word – a process completed at the start of the guessing phase. Once the word is selected, the other tasks identified include creating all the different tools that are to be used by the player in order to draw their given word. Each of these tools will be developed as individual functions and once thoroughly tested, implemented into *Picture This!* after. Developing each individual sub-routine independently allows for easier testing and identification of bugs; It means that I will deal with simpler programs at each given time and implement them into the more complex final software after I am assured that they are error-free. For the ‘Guessing Phase’ sub-branch, the first function is required to combine all the drawings from each player into one display that the players will be shown. The aim is now to guess the larger word that the image represents. This can be achieved by having each drawing stored as an image file, with each image imported into the guessing phase’s Graphical User Interface in the correct order. Proceeding through the sub-branch, another function to complete is the chat-box. This will require network programming for it to be implemented, where player’s guesses are shown to all other players who are in the game. Input validation will also be needed for when guesses are made to check against a guess being correct – if it is correct, the guess is not displayed to the other players, so the correct answer isn’t spoiled. After each guessing phase, points need to be incremented for each player based on their performance before a leader board can be updated and displayed. A timer from the start of the round through to when the player guesses correctly is a method that could be used to produce this, with each player’s score being stored in a file that is read from when the leader board is displayed.

Finally, the ‘Sounds’ branch shows all the sub-routines that should be developed in order to create an immersive experience for the player. The ‘Background Music’ sub-branch contains only one function that constantly loops through a single tune. This can be developed in isolation to the main game software, being implemented once testing and debugging is performed so that it works as expected. The other sub-branch of ‘Sound effects’ identifies all the different ways that sound effects are implemented into my game. These are simpler lines of code that can be written into the software after the main game is created since they are not necessary to the execution of other parts of the program.

## Design of User Interfaces and Considering Usability Features

### Menu

This is an initial design of the Menu interface.



Game Title is displayed in the centre of the interface since this is where the player’s eyes will be directed to most.

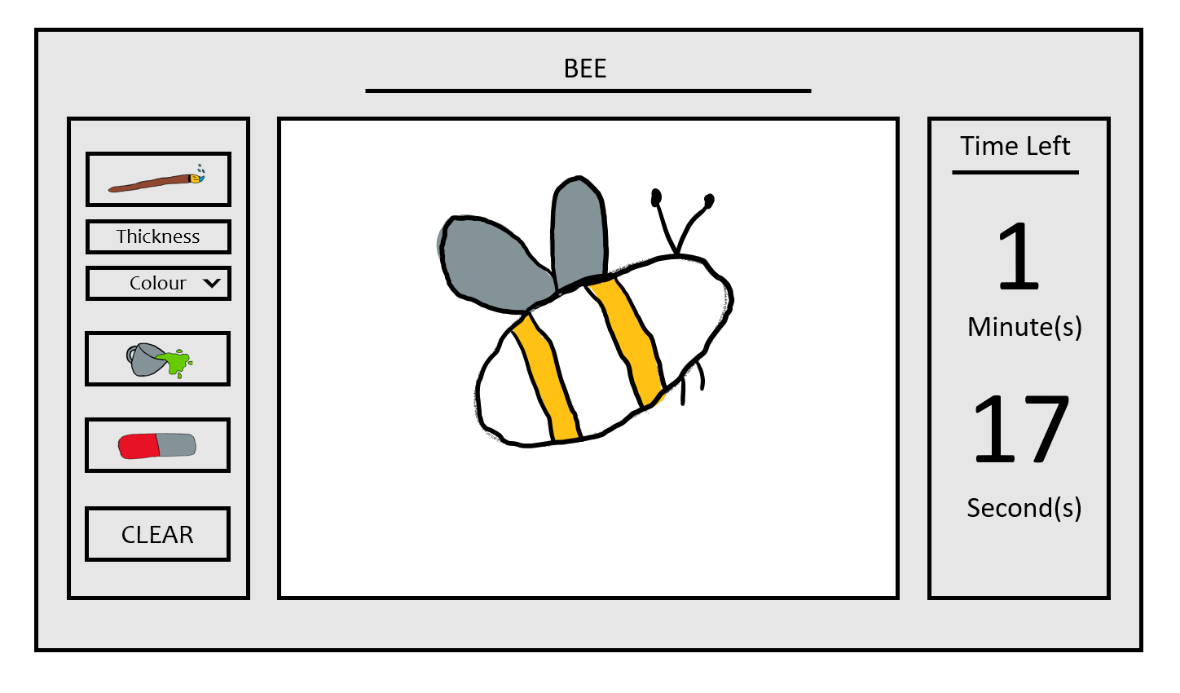
‘Create New Game’ and ‘Join Existing Game’ will be the two most popular selections so are above the other buttons, in order to provide an ease of access.

The button to exit the game is smaller than the others and at the bottom of the interface. This is so it takes up less space since it won’t be needed as much as the other buttons.

The settings button is an additional feature, but if implemented it will be used by the player to change settings that improves accessibility. For this reason, the button is large and in the centre of the screen.

The buttons are a large size with clear labels that identify their purposes. This means there is greater accessibility for the player, preventing any misunderstandings of a button’s function.

### Drawing Phase



The time remaining in the round is shown on the right in a large display so that it is clear for the player to see.

The different tools (i.e. brush, fill, eraser and clear) are displayed with unique icons that identify their function. They are located on the left as to not be in the way of drawing, whilst not being too distant.

The canvas, where the player draws, is a large area located in the centre of the screen. This is an intuitive location that makes sense since the player will naturally look towards the centre, making it more comfortable when drawing.

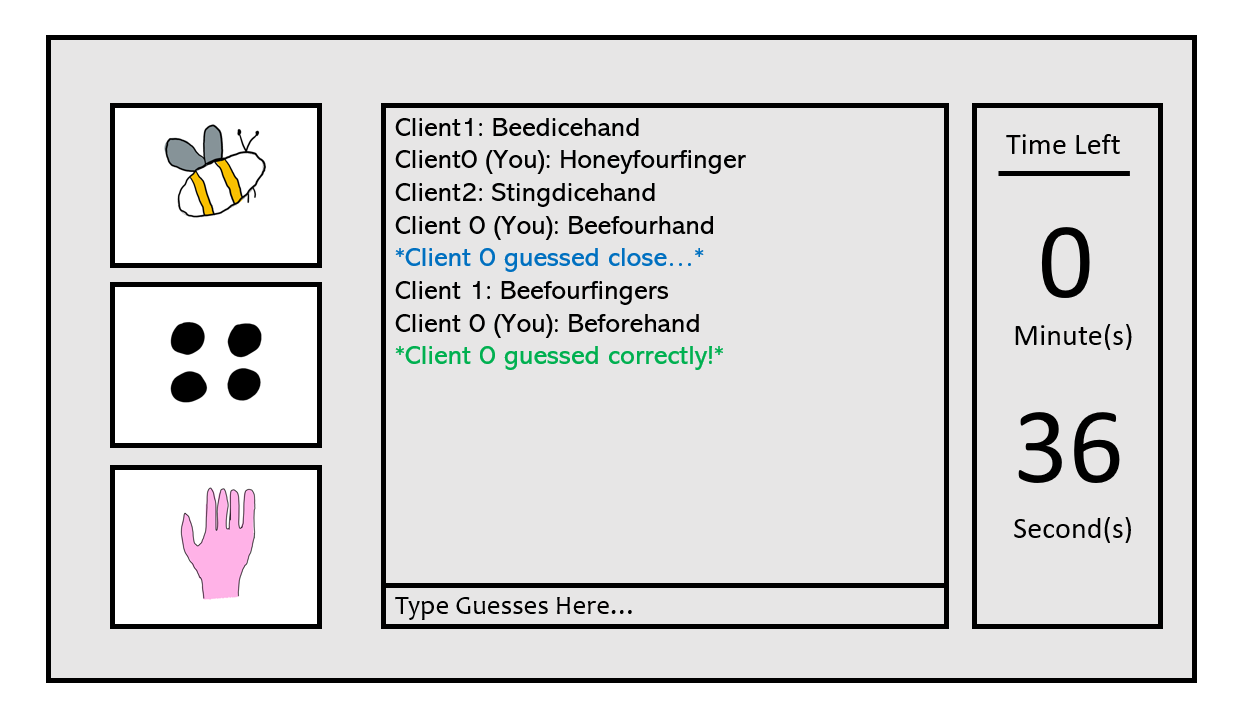
The colour of the brush can be changed through the ‘Colour’ drop-down box, and the thickness can be set by entering a value into the ‘Thickness’ textbox. These options are located by the Brush tool to prevent confusion.

The word to draw is displayed at the top to ensure the player doesn’t get confused about what they are drawing. It is not in the way of the other features whilst still being in a position that is easy to view.

This is an initial design of the Drawing phase interface.

### Guessing Phase

This is an initial design of the Guessing phase interface.



The time remaining in the round is shown on the right in a large display so that it is clear for the player to see.

Whilst not currently typing, there is a prompt for the player informing them of where to input their guesses. This is in a location that is easy for the player to see, preventing any misunderstanding.

The images are displayed in sequential boxes, allowing the distinction between each one. Usually, each drawing will represent a syllable within the answer, so having them separate helps players. They are located on the left since they do not need to be looked at constantly (i.e. the player can see it’s a drawing of a bee, its now about guessing the right word for it).

The history of guesses entered into the chat box are displayed here. It is in the centre so the player can easily see other player’s guesses – using them to help their own answers.

The player is informed when they guess close to the correct answer (e.g. their guess contains the correct word for at least one of the drawings). Also, it is important to note that when a player guesses correctly, the correct guess is not then displayed other players but only to the player who guessed it.

## Inputs and their Validation

*Picture This! r*equires the player to input data throughout the game at different points, therefore input validation (and input sanitation where necessary) is crucial to keeping my game functioning as intended. As an example, an input of the wrong format than the one expected could cause the game to crash or break, leading to the game being in an unplayable state for the player. Not only is this inconvenient to the players of *Picture This!* but also removes the professional feeling towards the game. My client agrees that crashes in a game can often show a lack of thought and preparation from the developer towards certain features, such as input validation and sanitation. Hence, I will ensure that all inputs are checked against certain conditions to make sure no issues are caused by unexpected inputs.

Below is a table of the different inputs of my game that require validation of different types. Each input is specified, and the type of validation is given, along with an explanation as to why this validation is performed.

|  |  |  |
| --- | --- | --- |
| Input | Validation Type | Explanation |
| Game code when joining an existing game | Presence Check | Ensures that data is entered into the textbox. Otherwise, an error is outputted to the player to tell them to enter appropriate data, in this case it is the game code. |
| Length Check | Checks the length of the data inputted to ensure the game code is the correct length. All game codes will follow a certain length so that it is clear that the code is for joining games and uniformity is met across *Picture This!*. |
| Format Check | The format of the data entered should be the correct format for a game code. It checks that the code entered is only made up of capital letters and numbers. This will increase security in my game. |
| Guessed answers to the players drawings | Presence Check | If a guess is entered that contains no text, the guess is not sent to the server. This is so the chat box is not filled with empty space. |
| Format Check | Checks that the guess entered contains only alphabetic characters. This is since the drawings will all be representative of words, so the player will not need to enter guesses that contain anything other than letters. |
| Input Sanitation | This is the process of converting an invalid input into one that is valid. In this case, guesses may be entered as capital letters or contain some capital letters within the word. Instead of flagging this as an error, the input can be completely converted to lowercase letters and then checked against the correct answer. |
| Profanity Filter | This will check the inputted guess against a list of blocked words to ensure that no inappropriate language is displayed in the chat box. The list of blocked words will most likely be stored in a file that can be searched and will also be updated regularly to ensure no exceptions are found. |
| Setting Round Length | Presence Check | Checks that data is entered into a textbox. Otherwise, a prompt is outputted to the player to tell them to enter appropriate data. |
| Format Check | The round length is an amount of time measured in seconds. This means that the data must be inputted in the correct format. Validation is performed to ensure that only integers are entered. |
| Range Check | Since the length of rounds should be between sensible values (for example, 10 seconds up to 120 seconds long). This validation will check that the data inputted is within a certain range of values, so the game is not ridiculously short, or overly long. |
| Setting Number of Rounds | Presence Check | A certain number of rounds must be played so a value must be entered in order for an error not to occur. This method of input validation checks that some data has been inputted, else, the user is prompted to enter data. |
| Format Check | Since the number of rounds is measured in integers, the user should input integers into the textbox. This validation ensures that only integers are inputted in order for errors not to be created. |
| Range Check | In *Picture This!*, it should not be possible for there to be no rounds played within a game. Following the same logic, it should also not be possible for there to be an extremely large amount of rounds played. This shows that an input validation technique of a range check, is required to ensure that a value is entered within an appropriate range. |
| Selecting Brush Colour | Lookup Table | In *Picture This!*, the colour choices of the brush tool will be limited to a set number of different colours. The player can then choose between these different colours. A lookup table ensures that the data from the input is restricted to a list of valid inputs, in this case the different colours. |
| Selecting Thickness of Brush | Presence Check | The brush must be a certain thickness, otherwise it will not work as intended. A presence check ensures that data has been entered into the thickness textbox, prompting the player to enter a value if not. |
| Format Check | The thickness of the brush will be proportional to a value entered as an integer. Therefore, an integer must be inputted so this validation is performed to ensure that the data is in the correct format. |
| Range Check | Larger and smaller values entered into the textbox would result in extreme thicknesses of the brush tool. This would cause the tool to be unusable since it would be harder to draw with. By completing an input validation range check, the data inputted is guaranteed to be between two values that keep the thickness at a sensible level. |

## Algorithms

This section will focus on making the development of the larger algorithms in *Picture This!* easier by explaining and planning them in a structured way. Below are the large algorithms in my game that will take the greatest time to implement since they contribute to the final game the most.

By explaining each algorithm in detail, I have allowed myself as the developer to understand the role of each set of steps clearly. This means that during development I can code the algorithms with greater ease since I recognise the purpose of the program well. Also, I have created pseudocode for each algorithm so that, when it comes to programming, I already have a basis to program upon. This will make it easier to develop each section of *Picture This!* since I know the logical structure and steps that each algorithm follows. I can use each pseudocode and convert it into Java code, rather than having to program the algorithms in Java from scratch, saving development time.

### Independent Explanations of Important Algorithms

#### Random Word Selection

The ‘Random Word Selection’ algorithm is a process, performed before every round, that randomly selects which words should be given to the players to draw, by taking into account: the number of players in the game, given as an integer value from 2 to 4; and the difficulty that has been chosen for that round as a string from 3 different options (“easy”, “medium”, “hard”), determined using Dynamic Game Difficulty Balancing.

**Random Word Selection**

String difficulty = “easy”

Int numPlayers = 3

String fileName = str(numPlayers) + “player.csv”

FUNCTION findWord(String filename) {

file = OPEN FILE filename

data = []

WHILE NOT file.ENDOFFILE:

line = file.READLINE()

words = line.SPLIT(“, ”)

IF words[0] == difficulty:

data.APPEND(words)

END IF

END WHILE

rand = RANDOM(0, data.LENGTH() - 1)

i = 0

FOR x IN data:

IF i == rand:

word1 = data[i][1]

word2 = data[i][2]

finalWord = data[i][-1]

switch(numPlayers) {

case 2:

RETURN word1, word2, finalWord

break

case 3:

word3 = data[i][3]

RETURN word1, word2, word3, finalWord

break

case 4:

word3 = data[i][3]

word4 = data[i][4]

RETURN word1, word2, word3, word4, finalWord

break

}

break

ELSE

i += 1

END IF

END FOR

file.CLOSE()

}

findWord(fileName)

In my game, the word that is being guessed depends on how many players can draw the different smaller words that make up the final word. For example, the word “beforehand” requires 3 players since it is split into the 3 words “bee”, “four”, and “hand”. Due to this, there will be different files for words that are appropriate for 2, 3 and 4 players.

In addition, I am also using Dynamic Game Difficulty Balancing during the play of *Picture This!* to determine if the difficulty needs to be changed by evaluating the average time to guess of all players. The difficulty relates to how hard it will be to draw and guess the word, with harder words being generally more complex and less resemblant of the final word. This means that each file must have a field specifying the difficulty of the word.

There will be a total of 3 different files containing words that could be selected, one for each number of players. Each file will be in a ‘.csv’ format where every new record (new line) consists of a string that defines the difficulty of the word, along with strings of the smaller words that it is split into and the word itself. In order to create appropriate file organisation, the files will be named after the number of players that the words are meant for (e.g. words that are split into 3 will be contained with a file called ‘3player.csv’). An example of the file is shown below.

Graphical user interface, text

Description automatically generated

In the pseudocode above, the difficulty is instantiated as a string variable with the value “easy”. In *Picture This!* the value of difficulty is actually determined by using Dynamic Game Difficulty Balancing but, for the purpose of demonstration for the algorithm, a simple declaration is used instead.

Using a Comma Separated Values (.csv) file the instead of a Text (.txt) file is a more professional method of storing values within a file. The type of data that I am storing in the file is suitable for a CSV file due to the fact that it is individual words, related to each other, on each line. This file type allows for a more defined structure to the file.

By parsing the required lines from the file into a 2D array, it allows for the faster querying of the list of words. Through the method I used, it means I can deal with arrays rather than searching through files, which is more efficient and easier in Java.

#### Validating Guesses

This algorithm is a process that validates each input, made by any user, that is entered into the chat box function. It is a necessary algorithm to ensure that any guesses entered are validated and, where required, sanitised.

The client specified that the chat box should be used for guesses but can also be used by players to communicate. Since this is a game for students aged between 11 and 18, there should be a profanity filter on the chat box that does not allow inappropriate language to be displayed to the other players. This can be achieved through creating a list of words, stored line by line in a text (.txt) file, that is searched after each input to check if it contains the inputted guess. If so, the input is replaced with a string containing the asterisk symbol (‘\*’), otherwise the input given is outputted to the chat box the same as it was entered. The file containing these words will be called censor.txt.

**Validating Guesses**

String clientName = “JayB”

String finalWord = “beforehand”

­­

FUNCTION profanityFilter(String word) {

file = OPEN FILE censor.txt

lines = []

FOR line IN file:

lines.APPEND(line)

END FOR

words = word.SPLIT(“ ”)

String sentence = “”

FOR x IN words:

FOR y IN lines:

IF x.TOLOWER().CONTAINS(y):

words[words.INDEX(x)].REPLACE(x, “\*” \* x.LENGTH())

END IF

END FOR

END FOR

file.CLOSE()

FOR item IN words:

sentence += item + “ “

END FOR

RETURN sentence

}

FUNCTION validation(String word) {

String toServer = “”

String toClient = “”

IF NOT any(char.ISDIGIT() FOR char IN word):

IF word.TOLOWER() == finalWord:

toServer = clientName + “ guessed correctly!”

play(“correctGuess.mp3”)

ELSE:

toServer = profanityFilter(guess)

END IF

ELSE:

toClient = “Invalid input, try again.”

END IF

IF NOT toServer == “”:

socket.SEND(toServer)

ELSE:

print(toClient)

END IF

}

WHILE True:

String guess = input(“Enter you guess…”)

validation(guess)

END WHILE

In the pseudocode above, the variable ‘ClientName’ variable is used to store a string value of the username that the client inputs earlier in the game. This username would be inputted at the start of the game and stored in a global variable, but for the purpose of the example algorithm it is instantiated here. Once a player guesses correctly, this username is outputted to the chat box for all players in a message stating the word has been guessed. A sound effect is then played to the player who got the answer correct.

In addition, the ‘finalWord’ variable stores the string of the word to be guessed. In the exemplar pseudocode it is instantiated as a variable and assigned a value here. However, in *Picture This!* this value would actually be attained from the **Random Word Selection** algorithm (see above). The value returned from this algorithm would be stored within a global variable for use in the algorithm below.

The pseudocode above demonstrates how Object Orientated Programming (OOP) can be used within the development of *Picture This!*. I have used multiple functions within one algorithm to allow easier readability of the code, leading to better testing and debugging.

In the algorithm above, I have prevented inputs that contain numbers within them. If a user attempts to enter a guess containing a number, a message is outputted to them explaining that it is an “invalid input” and to “try again” with a valid input. This is to ensure that no alterations of blacklisted words are entered, for example using the number ‘1’ instead of the letter ‘l’. This adds more security to the profanity filter that is to be used within the chat box.

#### Dynamic Game Difficulty Balancing (DGDB)

In *Picture This!*, the client wants the difficulty to be changed throughout the game at run-time so that it does not get too boring or too challenging for the players. This can be done through Dynamic Game Difficulty Balancing, by performing calculations after each round on the player’s scores to determine if the difficulty should be increased or decreased. After the calculations, the value will fall within a certain range corresponding to a difficulty. When finding a word to assign players, the **Random Word Selection** algorithm uses this difficulty (stored as a string in a variable) to only search for words that are of the difficulty determined in this algorithm.

A picture containing graphical user interface

Description automatically generatedSince this algorithm requires access to all the player’s scores, it can use the same file that is to be used for the leader board. This is a simple text file (an example of which is shown below) that stores the username of the player’s and their respective current scores. The algorithm will read this file, treating each line as its own array (split by the space in the middle), and parse the second item of each array (line) into a one dimensional (1D) array. The file is shown below:

To keep the difficulty up to date with the current performance of the players, this algorithm will be performed after every round before new words are assigned. It will take into account the average time it takes the players to guess the word correctly and compare it to boundaries that represent each difficulty.

**Dynamic Game Difficulty Balancing**

String difficulty = “medium”

Integer numPlayers = 3

Integer phaseLength = 60

FUNCTION findAvrg() {

file = OPEN FILE scores.txt

FOR line IN file:

data = line.SPLIT(“ “)

sum += float(data[1])

END FOR

Float avrg = sum / numPlayers

RETURN avrg

}

FUNCTION dgdb() {

percentTimeTaken = 1 – findAvrg() / 1000

IF percentTimeTaken >= 0 AND percentTimeTaken <= 0.3:

IF difficulty == “easy”:

difficulty = “medium”

ELSE IF difficulty == “medium”:

difficulty = “hard”

END IF

ELSE IF percentTimeTaken > 0.7 AND percentTimeTaken <= 1:

IF difficulty == “medium”:

difficulty = “easy”

ELSE IF difficulty == “hard”:

difficulty = “medium”

END IF

}

dgdb()

In *Picture This!*, the score awarded to each player is based on the time it takes to guess; the longer a player takes to guess correctly, the lower their score will be. Since all players are competing to guess the correct word that the drawings represent, the scores from all players can be taken to calculate an average score. This is due to all the players having an equal chance of getting the answer right, preventing any bias in the algorithm that favours a certain player. In the pseudocode above, a function is executed on the average player score in order to obtain a percentage of the time that it took the player to guess the word correctly. This percentage (given as a decimal) is then compared to set boundaries and the difficulty is changed depending on the outcome. In summary, if the average player score is very low, the difficulty is lowered and vice versa. If the average player score is determined to be at the right level, then the difficulty remains unchanged.

It is important to note that the boundaries shown in the above pseudocode are subject to change after testing. This will be involved in post-development testing where repetition of the algorithm with different sensible scores can be used to determine if the boundaries are either too low or too high. If it is found that the boundaries are inappropriate and are causing players to get either too easy or too hard words for their ability, then the boundaries can be adjusted accordingly.

In the pseudocode above, there are three variables that are instantiated at the start. These are all global variables that the values of which are set at the start of the game, however for the example I have shown them as part of this algorithm. The ‘difficulty’ variable of String data type is given a value of ‘medium’ by default and is adjusted throughout the game using this algorithm. The ‘numPlayers’ and ‘phaseLength’ variables are both of the Integer data type and are set at the start of the game by the creator of the game, in the ‘Creator Lobby’ interface; ‘numPlayers’ represents the number of players that are in the game (set to 3 for the example pseudocode), and ‘phaseLength’ is equal to half the value of the round length since this is the time (in seconds) of each phase.

#### Drawing Tools

Due to the nature of *Picture This!*, the drawing section of the game is key to the success of this project. For this reason, the development of the drawing tools requires great attention in order to create a fully functioning toolkit. In my game, there will be a brush, eraser, and clear tool that the players can use for their drawings.

The main tool to implement will be the brush tool, that can be used by the players to draw on the canvas. This is the tool that the players will use most so must be developed in a way that makes it easy to use to draw. It will work through placing small lines on the canvas in the mouse cursor’s current position, with the last line connecting to the previous. Due to the need for creativity, this brush tool will have two characteristics that can be set by the player: thickness and colour.

The drawings will require different colours to help make them recognisable, helping players to guess. This means that the option to select a variety of different colours is needed, preferably by using a drop-down selection menu. Each colour will be presented as an option in the drop-down box that can be selected by players. During development, if it is felt by the client that more colour options should be available to the users then the option to enter RGB values could be implemented so that players can use more specific colours than the basic pre-sets in the drop-down menu. The values could be entered through a textbox that is being presence, format and range checked to ensure a valid input, with the decision to do this being concluded after testing has been carried out.

In addition to players being provided with a choice of different colours to use with the brush tool, there will also be the capability to change the thickness of said tool. The thickness could be selected through different methods, one of which would be to select values from a drop-down box. However, this would lead to either a long drop-down menu or a limited variety of values for the thickness. Instead, a validated textbox should be used where players can enter the thickness they want within a certain range in order to not limit the choice of thickness used. If the method of a textbox is used, sufficient validation would be required. This would include a range check that would ensure the value is between two set boundaries, preventing any unsuitable values for the thickness of the brush.

The eraser tool is essential in *Picture This!* so that players are able to remove any mistakes they have made in their drawings. It will also include the capability of changing the thickness of the eraser to increase its usability. This tool will be a necessary implementation for *Picture This!* to be successful since otherwise the drawing mechanic would be too impractical and frustrating, causing the game to lack fun. By utilising this tool, players can redo parts of their drawings with ease. The development of the eraser can be completed through different methods. These include either: the eraser simply being a brush tool that is the same colour as the canvas, drawing over previous ‘paint’ to create the illusion of removing it; or actually removing the ‘paint’ that is currently under the cursor. The former would be easier to implement into my game since it would employ the use of inheritance from the brush tool code, and subsequently implements polymorphism. However, since both methods are feasible, I can use testing to conclude which would be more suitable to this project.

The final tool to implement is a simple clear tool that’s role is to clear the whole drawing canvas with the click of a button. This will be relatively simple to implement since it only involves removing all the ‘paint’ on the canvas. It is likely that this can be done with one or a few lines in the Swing library so shouldn’t take long to develop. By implementing this tool in *Picture This!*, it allows players to quickly restart their drawing for any reason such as making mistakes or thinking of new ideas.

**Drawing Tools**

Integer oldx, oldy, currentx, currenty

FUNCTION mousePressed(self) {

oldx = self.getx()

oldy = self.gety()

}

FUNCTION mouseDragged(self) {

currentx = self.getx()

currenty = self.gety()

drawLine(oldx, oldy, currentx, currenty)

}

FUNCTION clear() {

setColour(WHITE)

drawRect(0, 0, Canvas.getWidth(), Canvas.getHeight())

setColour(BLACK)

}

FUNCTION eraser() {

setColour(WHITE)

setThickness(10)

}

FUNCTION black() {

setColour(BLACK)

setThickness(5)

}

FUNCTION blue():

setColour(WHITE)

setThickness(5)

}

IF mousePressed == True:

mousePressed()

END IF

IF mouseDragged == True:

mouseDragged()

END IF

The pseudocode above consists of multiple functions that are used to implement the drawing tools into *Picture This!*. There are also four variables of the Integer data type that are defined at the start of the algorithm which are used to store the coordinates of the mouse position when the mouse is clicked, and subsequently dragged.

The ‘mousePressed’ function is called whenever the mouse’s left button is clicked on the canvas. When called, this function stores both the x and y coordinates of the current mouse position. Then when the ‘mouseDragged’ function is called the x and y coordinates of the new mouse position is called, and a line drawn between the two coordinates. This is constantly repeated when the mouse is being dragged, drawing lots of small lines across the canvas, resulting in a functioning brush tool. The if statement at the end of the code is a representation of the code that will check when there is an input, in the form of the mouse being either clicked or dragged, to the canvas. In reality, this will be a specific piece of code that is used in Java Swing applications, however an if statement is suitable for pseudocode since it portrays the logical structure that I will use.

Two functions, ‘black()’ and ‘blue()’, are used for the different colours that will be available to players as they draw. They are examples of the different functions that will be included for all the colours in my game. In the actual code of *Picture This!*, there will be many functions for all the different colours but for the simplicity of the example, I have only included two of them. In each colour function, the colour of the paint is set, along with the default thickness.

In addition, there is a function that is used for the eraser tool. This tool is simply a form of inheritance from the brush tool and follows the similar structure. When this function is called, the colour is set to white (the same colour as the canvas). This shows the eraser tool being implemented as being a brush tool and ‘painting’ over the previous ‘paint’. As mentioned above, however, the eraser tool can be implemented in another way, this is simply the pseudocode that proved easier to implement. The default thickness is set to twice that of the normal brush tool so that the eraser tool is more efficient at removing paint and players can be less accurate with the tool.

Another tool that is shown in the pseudocode is the clear tool. Shown in the example, this works by drawing a rectangle to the canvas that is the size of the canvas and the same colour (white). This has the effect of re-‘painting’ over the entire canvas so that it is clear for the player to restart their drawing. In this function, the dimensions of the canvas are retrieved by using methods of the **Canvas** class called ‘getWidth()’ and ‘getHeight()’; both are simple methods to implement.

In the pseudocode above, the ‘setThickness()’ method is called multiple times when setting the default thickness of all the different colours for the brush tool and also the eraser tool. Every time this function is called, it takes in an integer and uses it to set the thickness of the line that is drawn between the two coordinates. This causes the overall brush stroke to look more/less thick depending on what value the thickness is set to.

### Overall Program Flowchart

Along with explaining each algorithm and writing its respective pseudocode, I am also going to create a flowchart for the overall state of *Picture This!* that highlights the program flow of the finished game. This will help facilitate development and allow me to connect each algorithm together with greater ease due to knowing how each algorithm should lead to the others.

Since I will be using an iterative development methodology for this project, the overall finished product of my game could vary in logic and program flow to that displayed in the flowchart below. This is expected so the flowchart should not be followed strictly but used as a guideline.

Diagram, shape, polygon

Description automatically generated

In the flowchart above, it is important to explain the timer loops. In *Picture This!*, each round is a certain time length (this value is specified by the creator of the game in the ‘Creator Lobby’). The value of the round length is the time given each round meaning it’s the time for the drawing and guessing phase combined. Due to this, when instantiating the timer before each phase, the length of time is half of the value inputted. In addition, during development I plan to have the timer in a separate thread so that it can run in parallel to the other components of the game. The thread of the timer can be slept for one second and then the ‘timerLength’ value decremented by one. Once the time left reaches 0, it is the end of the round, so the program moves on.

The flowchart above also demonstrates how subroutines can be used within my project. This is evident through the use of the ‘guessValidation’ algorithm being included. This subroutine was shown in the flowchart because it links directly to the program flow of the final solution. However, although this algorithm was demonstrated, others were not since they didn’t directly affect the logic flow of the program. For example, the drawing tools are an algorithm that is very important to *Picture This!* but wasn’t included in the flowchart. This is because they do not determine the time or pathway of the program so are not worth implementing into the flowchart.

## Identifying Test Data

Since *Picture This!* is a large project and will take in many inputs throughout the whole game, it is imperative that testing is carried out so that, as the developer, I can be sure that the game’s features will perform as intended. By testing, it can be concluded that each section of the game is functioning in the correct way with the expected outcome.

For my project, I will use an iterative development method in order to keep the game’s development up to date when new issues are discovered. This method consists of constant and rapid changes to the code once testing has been performed and new errors/issues have been discovered. It aims to have each iteration of development being a small change that solves a small problem in order to work towards the final product. By using this method, development can follow a structured pattern leading to easier documentation of each new improvement that is made upon a newly found bug.

### Menu Graphical User Interface

The ‘Menu’ Graphical User Interface (GUI) includes three buttons, with a fourth being implemented as an additional feature if required, that should all perform certain tasks. Each button must be checked that it does what is intended upon a mouse click but remains inactive until pressed.

Since the GUI only consists of buttons and no other types of inputs, the testing only has to account for two different pathways: if the button is pressed or if it is not pressed.

|  |  |  |  |
| --- | --- | --- | --- |
| ‘Menu Graphical User Interface’ Test Data | | | |
| Test No. | **Aim of the Test** | **Test Input Data** | **Expected Outcome** |
| 1 | The player can select the ‘Create New Game’ button to create a new game | Mouse click on the button | The player is taken to the ‘Creator lobby’ GUI where they can select the settings for the game and are provided with a game code for their game (see the **Creating or Joining Games** section below) |
| No mouse click on the button | No action should be performed, and the player should remain on the ‘Menu’ GUI |
| 2 | The player can select the ‘Join Existing Game’ button to join an existing game | Mouse click on the button | The player is prompted to enter a game code that is then validated (see the **Creating or Joining Games** section below) |
| No mouse click on the button | No action should be performed, and the player should remain on the ‘Menu’ GUI |
| 3 | The player can select the ‘Quit’ button to quit *Picture This!* | Mouse click on the button | The *Picture This!* application is exited and closed |
| No mouse click on the button | No action should be performed, and the player should remain on the ‘Menu’ GUI |

### Creating or Joining Games

When the player is either joining a pre-existing game, or creating their own new one, there are multiple inputs and options that can be entered and selected. These inputs must be validated and tested to ensure that they function as intended. This can be done by testing inputs that are purposefully out of range, the wrong format, or other invalid attributes. They should also be tested against valid inputs to ensure that the correct function is performed when required.

|  |  |  |  |
| --- | --- | --- | --- |
| ‘Creating or Joining Games’ Test Data | | | |
| Test No. | **Aim of the Test** | **Test Input Data** | **Expected Output** |
| 4 | The player can enter a game code that allows them to join a pre-existing game | Game code with length less than 5 characters  (e.g. H8J, D, 2389) | The game code should be rejected, and a prompt is shown to the player to enter a valid game code with the correct length |
| Game code with length greater than 5 characters  (e.g. GW8J0L, I8DI310) | The game code should be rejected, and a prompt is shown to the player to enter a valid game code with the correct length |
| Game code with length of 5 characters  (e.g. AU7F3, 49137) | The game code’s length should be accepted, and the game code will undergo further validation |
| Game code containing both letters and numbers  (e.g. JE8PO, 89QW4) | The game code’s format should be accepted, and the game code will undergo further validation |
| Game code containing only letters  (e.g. HEOSD, KSPWH) | The game code’s format should be accepted, and the game code will undergo further validation |
| Game code containing only numbers  (e.g. 74923, 27593) | The game code’s format should be accepted, and the game code will undergo further validation |
| Game code containing no letters or numbers  (e.g. /#@]{, -/=+\_) | The game code should be rejected, and a prompt is shown to the player to enter a valid game code with the correct format |
| Game code containing a character that is not a letter or number  (e.g. H7S#P, WT3/X) | The game code should be rejected, and a prompt is shown to the player to enter a valid game code with the correct format |
| No input is given | A prompt should be shown to the player to enter a valid game code |
| 5 | The player is taken to the ‘Joiner lobby’ GUI once a valid game code is entered | A valid game code | The player should be taken to the ‘Joiner lobby’ GUI and waits until the game begins |
| An invalid game code | The game code should be rejected, and the player is not taken to the ‘Joiner lobby’ GUI, remaining on the GUI to enter a game code |
| 6 | The creator of the game can enter an integer for the number of rounds to be played | Integer within the range of 1 and 5  (e.g. 1, 5, 4, 3, 2) | The value should not be rejected, and this is the number of rounds in the game |
| Integer outside of the range of 1 and 5  (e.g. 0, 13, 6, 99) | The value should be rejected when the player tries to start the game, and a prompt is shown to the player to enter a valid value in the correct range |
| Non-integer data type  (e.g. 0.5, hello, True) | The value should be rejected when the player tries to start the game, and a prompt is shown to the player to enter a valid value of the correct data type |
| 7 | The creator of the game can enter an integer for the time length (in seconds) of each round | Integer within the range of 15 and 120  (e.g. 15, 100, 73, 119) | The value should not be rejected, and this is the time length (in seconds) of each round in the game |
| Integer outside of the range of 15 and 120  (e.g. 3, 14, 230, 140) | The value should be rejected when the player tries to start the game, and a prompt is shown to the player to enter a valid value in the correct range |
| Non-integer data type  (e.g. 0.5, hello, True) | The value should be rejected when the player tries to start the game, and a prompt is shown to the player to enter a valid value of the correct data type |

### Drawing Tools

In the drawing phase of *Picture This!*, there are many options for inputs by the player. These are notably the drawing tools that can be used, but also the adjustments to the thickness and colour of said tools.

For the actual tools themselves, they can take on two states: the tool is selected, or the tool is not selected. This makes testing simpler and easier since there are only two outcomes that are possible and need testing. However, for more complex inputs such as changing the thickness of the brush tool, more testing is needed. For example, valid data type and range checks.

|  |  |  |  |
| --- | --- | --- | --- |
| ‘Drawing Tools’ Test Data | | | |
| Test No. | **Aim of the Test** | **Test Input Data** | **Expected Outcome** |
| 8 | The player can draw on the drawing canvas by selecting the brush tool and moving the mouse | Mouse click and drag on the canvas whilst the brush tool is selected | ‘Paint’ should be outputted onto the drawing canvas at the mouse cursor’s position until the mouse button is released |
| Mouse-click and drag on the canvas whilst the brush tool is not selected | The drawing canvas should remain in its current state and no new ‘paint’ is outputted onto it |
| 9 | The player can change the thickness of the brush by inputting an integer | Integer within the range of 1 and 100  (e.g. 1, 99, 34, 67) | The value should not be rejected, and this is the thickness of the brush tool used when drawing |
| Integer outside of the range of 1 and 100  (e.g. 0, -72, 349, 115) | The value should be rejected, and a prompt is shown to the player to enter a valid value in the correct range. The thickness remains at its current value |
| Non-integer data type  (e.g. 0.5, hello, True) | The value should be rejected, and a prompt is shown to the player to enter a valid value of the correct data type. The thickness remains at its current value |
| 10 | The player can change the colour of the brush by selecting colours from a drop-down selection menu | Mouse click on a colour | The brush tool should draw in this colour on the canvas when being used |
| No mouse click on a colour | The brush tool colour should not change and remains the current colour |
| **NOTE:** This test should include testing each colour to ensure that it is labelled correctly | | |
| 11 | The player can select the ‘Clear’ button to clear the drawing canvas | Mouse click on the button | The whole drawing canvas should be cleared with no ‘paint’ left |
| No mouse click on the button | The whole drawing canvas should remain in its current state |
| 12 | The player can erase their drawings by selecting the eraser tool and moving the mouse | Mouse click and drag on the canvas whilst the eraser tool is selected | The ‘paint’ under the current position of the mouse cursor should be removed until the mouse button is released |
| Mouse click and drag on the canvas whilst the eraser tool is not selected | The drawing canvas should remain in its current state and no ‘paint’ is removed |

### Chat Box for Guesses and Communication

During the guessing phase of *Picture This!*, there is only one input that consists of submitting strings for guesses and communication. This means there are less inputs to test, however, the input that is present should be tested thoroughly since it can take many pathways.

Testing should include submitting invalid inputs of the wrong format, empty strings, and censored words so that all possible pathways are traversed and checked. Since the input will be a string, there will be multiple combinations of invalid inputs.

|  |  |  |  |
| --- | --- | --- | --- |
| ‘Chat Box for Guesses and Communication’ Test Data | | | |
| Test No. | **Aim of the Test** | **Test Input Data** | **Expected Outcome** |
| 13 | The player can type and submit text into the chat box for guesses and communication | Text containing no numbers that matches the correct guess | The input should not be rejected and a message to the player is outputted confirming they guessed correctly. The input is not outputted to the chat box |
| Text containing no numbers that does not match the correct guess | The input should not be rejected, and the input is outputted to the chat box for all players to see |
| Text containing numbers | The input should be rejected, and a prompt is shown to the player to enter a valid input of the correct format |
| Text containing no numbers that matches a blacklisted word | The input should not be rejected, and the input is outputted to the chat box for all players to see after the censored word has been replaced with a series of ‘\*’ characters |
| No input is given | A prompt should be shown to the player to enter a valid input |

## Required Data Structures for the Solution

*Picture This!* will be a big application so will contain many variables, constants, and data structures; making this section important to the design of my game. I will be able conclude the best data structures for a certain task due to the justifications I must give, leading to a better program overall.

It is important to note that this is still the design phase of the project, so during development it could be seen that another class or data structure is needed or vice versa, so the information shown below can be subject to change. However, this section gives a good layout of the most likely classes to be used.

### Class Diagrams and Required Variables

Text

Description automatically generatedSince I am going to be using an Object Orientated Programming (OOP) approach to develop my game, it is important that I make class diagrams so that, when it comes to implementing these classes, I can understand how the program should be laid out. These class diagrams are shown in the sections below with explanations about each. This is an example of how the class diagrams will be laid out:

Java is a language that has separate ‘.java’ files for each class, so by understanding the different classes that I will have, I can also ensure good file organisation. An Object Orientated Programming (OOP) approach is ideal for the language that I am using since Java is built upon the ideals of OOP and is specifically designed to be used as an OOP language.

Moreover, by developing my project in an Object Orientated way, I can make use of all the benefits of OOP in my code such as easy code reuse through inheritance, flexibility through polymorphism and modularity which will be helpful when testing. The decision to use an OOP approach in my project makes sense and is an obvious choice.

I will have classes for all Graphical User Interfaces (GUIs) and also other smaller components for the GUIs, along with a class for the server side of *Picture This!*.

#### Main Menu Interface

Text

Description automatically generated with medium confidenceThis class will contain all the code for the ‘Main Menu’ interface. This GUI consists of a title label and three buttons so will be relatively small, only containing a few methods and attributes.

For the class diagram shown above, it contains four attributes that are all Swing components consisting of a JFrame and three JButtons. The JButtons are simply the buttons that will be used for either creating a game, joining a game, or quitting the game. On the other hand, the JFrame is essentially the window itself and is how I can display these buttons to the user. In addition, there is a JLabel which will contain the title of the game, displaying the text “Picture This!”.

In terms of methods, there is the main() method required by Java so that the compiler knows the structure of the program. This will simply contain a line calling an object of the class, and then calling the method show() on that object. The show() method is what is used to set up the interface to be shown. It is where all the buttons are initialised and where the frame’s settings are set so that it has the required characteristics. This is also where the Swing components can be assigned an ActionListener to take in inputs to that component as a value.

The actionPerformed() method is used to listen to any ActionEvents that occur whilst running the program and performs a set piece of code that is determined through selection, where the condition is the type of ActionEvent. This is used to perform different actions to different inputs on different components and means that sometimes developers only need to use one ActionListener be used at times where the inputs are simpler (such as in *Picture This!*).

#### Creator Lobby Interface

This Graphical User Interface (GUI) is where the player will be taken to once they select the ‘Create New Game’ button on the ‘Menu’ GUI. This interface will consist of two textboxes and a button and will be similar to the **Main Menu Interface** class diagram.

A picture containing shape

Description automatically generated

In this class diagram, there are attributes that include Swing components and also variables that are used to store the game settings entered by the player. There is a JFrame that functions the same as in the **Main Menu Interface** class diagram and is used for the same purpose. There are also two JTextFields that allow the player to enter the values for the round length and number of rounds. These are text boxes that the player can type into. Moreover, there is a JButton that can be pressed to start the game. This is when all other players who are joining the game are also taken to the drawing phase Graphical User Interface (GUI).

For the methods, there are the similar methods of actionPerformed() and show() which will be the same as the **Main Menu Interface** class diagram, however, there is no main() method like in the previous diagram. This is because this interface will be loaded after a button is pressed from the ‘Menu’ interface, so the program does not start in this class.

The other methods that are involved in this class are used so that encapsulation is present for this class. This is good practice as a developer as to keep the variables within the class safe from being modified outside the class. The setNumRounds() and setLengthRounds() methods will allow the values of the number of rounds and the length of the rounds to be set respectively, without directly accessing the variables themselves. As for the getNumRounds() and getLengthRounds() methods, they allow the access to, and usage of the values set by the player from outside the class without accessing the variables directly and jeopardising the integrity of the class structure.

#### Entering Game Code Interface

Text

Description automatically generated with low confidenceThis is where the player will enter a game code of a pre-existing game in order to join it. This is done after the ‘Join Existing Game’ button is selected on the ‘Menu’ Graphical User Interface (GUI) and I have chosen to have this as a separate interface, and hence a separate class, so that the code organisation of my game is better. It will also make my game more intuitive to the player.

The attributes for this class are simple and are mostly Swing components of the type that has been explained from previous class diagrams. There is also a variable of the String data type that is used to store the result of the game code entered. This value is what is validated and checked to ensure that it is valid.

By using the setGameCode() and getGameCode() methods, a safer way to access the contents of the gameCode variable is provided. This is another example of encapsulation within my project. These methods are used to safely access the gameCode entered during the validation process.

#### Joiner Lobby Interface

After the player has entered a valid game code, they will be taken to this simple Graphical User Interface (GUI) where they wait until the creator of the game selects the ‘Start Game’ button in the ‘Creator Lobby’ interface (see **Creator Lobby** class diagram above). This is the most basic interface in *Picture This!* and will only contain a label informing the player to wait.

Text

Description automatically generated

Since this interface is a simple GUI that just displays a message (through the JLabel) notifying players to wait until the game begins, there are few attributes and methods. The attributes in this class are only the necessary Swing components: a JFrame to display the content, and a JLabel to display the message. In addition, there is only one method which is the same one used for every Graphical User Interface. It sets all the pre-sets required for the interface to be displayed and is called when the GUI should be loaded and shown.

#### Drawing Phase Interface

A picture containing shape

Description automatically generatedThis is one of the two main Graphical User Interfaces (GUIs) that are important to the gameplay of *Picture This!*, so contains many parts. For larger interfaces, however, the supposed interface itself can be split into sections that can be programmed as their own class. For this specific situation, the drawing phase interface can consist of a class for all the buttons and labels required for the GUI, whilst another class handles the drawing canvas since this is a big section (see **Drawing Canvas** class diagram below).

In the class diagram above, the attributes consist of a JFrame (its uses being aforementioned), JButtons for all the drawing tools and the colours that the player can use; and a JLabel that displays the word that has to be drawn. These attributes are all simple Swing components that can be implemented without hassle.

For the methods, they are the same as ones seen in most of the previous classes. The show() method helps initialise the JFrame and assigns all required characteristics to the window, whilst the actionPerformed() method allows code to be executed upon an ActionEvent of an input to any component that has the ActionListener associated with it.

The setThickness() and getThickness() methods are more examples of encapsulation within my game code. They are used to set the thickness of the brush tool and then allow other objects to retrieve the value of the thickness of the brush and use it.

#### Drawing Canvas

This class is one that is to be implemented into the ‘Drawing Phase’ interface. It is the canvas that players can draw on and will be in the centre of the GUI when implemented. Since I am using the Swing library to develop my game, this class will contain methods from the Swing library that already exist. Due to this class only being concerned with the drawing of ‘paint’ onto the canvas, it will only A picture containing shape

Description automatically generatedinclude a few attributes that are necessary to this action.

The attributes are variables of the Integer data type that represent the position of cursor in x and y coordinates separately. These are used whilst drawing to draw a line between the old x position and the current x position, and the same for the y values. The oldxPos and oldyPos variables are then assigned the values of the currentxPos and currentyPos variables and the process repeats until the mouse button is released.

The thickness attribute will be used to store the result of the getThickness() method from the **Drawing Phase Interface** class (shown above). This value will then be used to change the thickness of the brush/eraser tool.

The methods mousePressed() and mouseDragged() are methods from the Swing library that listen for MouseEvents of a specific type. These methods are called so that the program knows when the mouse is being used to draw. Although they are not methods that I, as the developer, have created myself, they are still important methods that I have used so I have included them in the class diagram.

This class also overrides the pre-existing method paintComponent() with code that specifies the type of drawing that is wanted. This method must be overridden so that the program knows what to ‘paint’ onto the canvas. This is a key method that must be included, otherwise the whole class would not function as intended. In addition, the constructor method canvas() is called to set specific rules upon initialisation of an object of that class.

Along with the methods that are a part of the Swing library, the class contains methods for each colour and tool specifying what actions are performed when these components are selected. The clear() method will empty the canvas of all previous ‘paint’, whilst the erase() method will work similar to the brush tool but paint over in the colour of the canvas (most likely white) at the curser’s current position. The [colour]() method is used in my class diagram to represent methods for all of the different colours that I will have in *Picture This!*. These methods will simply set the colour to the selected one.

#### Guessing Phase Interface

This is the other of the two main Graphical User Interfaces (GUIs) that are important to the gameplay of *Picture This!*, so it contains many parts. The main aspect of guessing is the chat box so this can be programmed in another class in order to break up the program better and allow for more modularity.

Text

Description automatically generated with medium confidenceMeanwhile for this class, it will contain the rest of the components that are needed in this interface, such as the collection of all the drawings made by the players. This should be done through using a JPanel, which groups the images together so that they can be displayed in an organised way.

For the attributes, the JFrame is used for the same purpose in this class as explained above for the previous class diagrams. The JPanel is used to collate and group all the drawings from players together into one component that allows for easier layout and formatting in the Graphical User Interface (GUI).

The JTextField will be used by the player to enter guesses into the chat box. The inputs received by this Swing component will be used in the **ChatBox** class so that the appropriate output can be made to either the chat box or the player. The text inputted into the JTextField is submitted by clicking the ‘Submit’ JButton.

The methods setInput() and getInput() are present in order for encapsulation within this class. They will allow objects to call this method without directly accessing the ‘input’ variable of data type String which stores the value entered by the player.

The only other methods that need to be present in this class is the show() method and the actionPerformed() method that are both previously aforementioned and explained in detail about their purpose.

#### Chat Box

Text

Description automatically generated with medium confidenceThis class is one to be implemented into the ‘Guessing Phase’ interface. It will contain the code for the chat box that is to be used by players to guess what the drawings represent. An object of this class can then be made within the **GuessingPhase** class, and this chat box implemented into the ‘Guess Phase’ interface.

The JTextArea is a Swing component that displays text in a large area. In my game, this attribute is to be used to display all the previous guesses made by players, capturing the recent history of the chat box. After guesses are inputted, they are validated, and the appropriate output is displayed on this JTextArea.

The two String variables (‘finalWord’ and ‘username’) hold the values of the word to guess and the player’s name respectively. These are used within the guessValidation() method when checking if the guess is correct and outputting the statement indicating a correct guess by the player.

For the methods in this class, there is an actionPerformed() that performs the same as explained in most of the above class diagrams. Also, there are two methods responsible for the validation and censorship of the inputted guesses by the player. The profanityFilter() method checks the input against a list of blacklisted words, listed in a Text (.txt) file, and censors any words that are not permitted to be displayed. The guessValidation() method then takes the subsequent input and checks if it is valid, correct or if it is neither. The output to the JTextField is dependent on the outcome of this method.

The methods toServer() and toClient() are the methods that are responsible for outputting the player’s guess appropriately. Based on the outcome of the validation, if a message is only to be outputted to the player who made the input then toClient() is called. Otherwise, if there is an output needing to be displayed to the whole chat box, to Server() is called and this is responsible for sending the message to the server, which in turn outputs the message to every player.

#### Countdown Timer

Text

Description automatically generatedIn *Picture This!*, the rounds are timed so this means there is a requirement for a timer. This class will be responsible for creating the timer so that the classes that handle the Graphical User Interfaces (GUIs) can implement this component. Both the **DrawingPhase** and **GuessingPhase** classes will make use of this timer, displaying the remaining time left in the round.

This class will have two attributes. Firstly, the variable ‘counter’ of data type Integer is assigned the value of half of that retrieved from the getLengthRounds() from the **Creator Lobby Interface**. This is because the round length entered by the player is for the drawing phase and guessing phase combined, so each implementation of the timer should cover half of this time.

The other attribute is a Timer component from the Swing library that is used to create a countdown timer. This is the main essential part of this class since it defines the whole purpose of this section of code.

The only method that is required for this class is a constructor method that is used to initialise, start, and iterate the timer so that it functions as intended. This is the only method required since this class is to be implemented into other interfaces, so the program starts from there.

#### Leader Board Interface

This class contains the code for the ‘Leader board’ interface that is displayed to the players after every round. Each time, the player’s scores are updated, and the leader board is reorganised so that players are ranked in terms of best score to worst.

Text

Description automatically generated

There are two attributes for this class. The JFrame attribute shares the same function as the other JFrames from the other class diagrams that have been explained. The other attribute is a JTextArea that is used to show the ranking of players according to their score. This is how the leader board will actually be displayed.

The methods consist of the show() method, which has been previously explained in the other classes, and the order() method which is responsible for ordering the players on the leader board with respect to their scores. The highest scoring players are displayed at the top of the leader board, and the lowest scoring players at the bottom.

### Required Files

Along with all the necessary classes for my game, it is important to outline all the files that I will include in the development of *Picture This!*. For each file, I will provide an example of the file and justifications about them.

Since this is still the design of the game, there is a likelihood that the files explained below could change, not be used, or others be used instead during the actual development of *Picture This!*. This could be due to finding an alternative, more efficient method of programming my project. However, all changes and adaptations to the structure of my game will be fully documented in the **Developing the Coded Solution** section (see below) since it is all a part of the development process.

#### Word Selection - 2player.csv, 3player.csv, 4player.csv

Graphical user interface, text, application

Description automatically generatedThese are three files that are used for the storage of the words that are assigned to players to draw. The files contain final words that can be split into either two, three or four constituent sub words. Each final word is assigned a difficulty to it, that allows the **Random Word Selection** algorithm to only search words of the required difficulty. Below are three examples of the files, one for each:

Graphical user interface, text, application

Description automatically generatedGraphical user interface, text

Description automatically generated

I have used a Comma Separated Values (.csv) file since it allows for the words to easily be parsed into an array, making it easier to search through. This is easier than using a Text (.txt) file since each record in the file can be a new word and the words it can be split into.

#### Game Codes to Join Games – codes.txt

Graphical user interface

Description automatically generated with low confidenceThis file is to be used to store all the current game codes of existing games. This is so that when players enter a game code, it can be checked against the list of codes in this file to ensure that it is valid and does exist. Below is an example of this file:

This will be a simple file since there is only one code per line, and there will not be many codes in the file at a time due to the estimated popularity of *Picture This!*, so a Text (.txt) file is suitable. All that is needed is to read and write to the file in order to keep the codes updated and to check codes against the one submitted. A Text file is suitable for these requirements.

#### Blacklisted Words for Profanity Filter – censor.txt

A picture containing graphical user interface

Description automatically generatedWhen players submit an input into the chat box, it must be checked to ensure that no blacklisted words are outputted to the chat box for all players to see. This means that the inputted guess must be checked against a list of blacklisted words. This file contains those words, and an example is shown below:

It is important to note that in my game, the blacklisted words will be inappropriate words that should be censored to avoid players being exposed to rude and offensive language. However, for the example file above, test words have been used in order to not show inappropriate language in this documentation for *Picture This!*.

Due to the simplicity of the file, there is no need to use a more complicated and detailed file type. I will only be performing read actions on this file, so I have used a Text (.txt) file which is easier to implement and use.

#### Leader Board Scores – scores.txt

This file will consist of all the current player’s scores and will be updated after each round to adjust the scores after the player’s recent performances. Each line in the file will consist of the username of the player, followed by the current score that they have. Since this file will change each game, it is created at the start of the game and is deleted after the game so that no duplicates are made. An example of the file is shown below:

A picture containing graphical user interface

Description automatically generated

I have used a Text (.txt) file since it is a relatively small and simple file containing a maximum of four lines at any given time. Also, I will have to read and write to the file, and this is easiest for a Text (.txt) file.

After each round the file will be written to and updated with the player’s scores. When the leader board is to be shown, the file is parsed into an array (another reason why using a Text file would be simpler and more efficient) where each line is a new index in the array. This array can then be sorted to ensure that it is in order and loaded into the JTextField for the leader board to be outputted.

Above is an example of the scores that have been calculated. These are example scores that have been estimated from what the scores of a normal game are thought to look like. Please note that the actual scores could vary depending on the game and the players in the game.

In order to fully understand the file, I will explain the scoring system so that the scores and where they came from can be understood. In *Picture This!*, the score the player is awarded is based on the time it takes them to guess the word, that the drawings represent, correctly. This is done by calculating the percentage of time left on the timer after guessing correctly and using this to take a percentage out of the maximum points that a player can be awarded. The steps of the calculation are as follows:

1. Take the time it took the player to guess and divide by the length of the guessing phase (this value is half of the value chosen by the player for the round length). This leaves a percentage, in decimal format, of the time it took the player to guess correctly out of the total time possible.
2. Subtract this decimal percentage from 1 to give the percentage of the time left after guessing correctly.
3. Multiply this percentage, in decimal format, by the maximum score of 1000. This calculation takes the percentage of the maximum score that the player could achieve.

I have chosen 1000 as the maximum score since it is more visibly pleasing to the players than a score as a decimal. It is possible that the player could get a score of 0 if they fail to guess correctly, and on the other hand it is also possible for players to get the maximum score of 1000 if they guess instantly, however, this is extremely unlikely.

### Known Data Structures for Picture This!

Below is a table of data structures that are thought to be needed for the development of *Picture This!*. For each data structure, there is an explanation of its purpose and a justification for the type of data structure used. These reasonings are important to consider and highlight so that a conclusion can be made to the most efficient and effective of developing my game. It will allow me, as the developer, to identify which data structures would be most suitable to its purpose.

|  |  |  |
| --- | --- | --- |
| Data Structure | Explanation of Purpose | Justification |
| 2D Array: Leader board | The data in the scores.txt file is parsed into a two dimensional array so that every i index contains two j indexes for the username of the player and their respective score. This can then be sorted in descending order according to the score and outputted to the JTextField on the ‘Leader board’ interface. | I have decided to use a 2D array since I will know how many indexes there should be because it will directly correlate to the number of players. In Java, I can also easily sort an array using a lambda function which will make of the leader board easier. A two-dimensional array makes the data structure more organised and allows me to easily parse the data. |
| 1D Array:  Score | This one dimensional array is a data structure that’s contents will be constantly changing. It will hold the contents of the new line read in the scores.txt file. | I am storing the line as an array so that I can split the line into indexes, allowing me to only pick out the player’s score (and not their username) with ease. This means that I can efficiently perform calculations on the player’s scores during the **Dynamic Game Difficulty Balancing** algorithm. |
| 1D Array:  Words | This one dimensional array is a data structure that’s contents will be constantly changing. It will hold the contents of the different words chosen that will be provided to the players to draw. | I am using a one dimensional array to store the different words that are to be outputted to the players since I know the number of words there are so can set the defined size initially. Also, using an array will allow me to easily query the data structure in order to retrieve the words during the **Random Word Selection** algorithm. |
| Lists | This includes the variety of different temporary lists that are instantiated during different executions of code, most commonly in the different algorithms. They are used for storing different values for different purposes (e.g. storing words from a file when reading the file). | I have used lists for this since they are dynamic so their length can be changed at run time. This makes it useful for when I am parsing file content into them line by line where I do not know the length of the file. Lists are also mutable so I can query them and manipulate them easily in the Java programming language. |

# Developing Coded Solution

## Module 1: ‘Main Menu’ Interface

### Development Aim of Module 1

The development of this module focuses on creating the first, and simplest, Graphical User Interface (GUI) that the user will be shown - the Main Menu. From here, the players can select to either create a new game, join an existing game, or exit to desktop. These will all be buttons that must be developed and placed onto the screen so that the player can select them and perform certain actions. In addition, this interface should include a label displaying the title of the game (*Picture This!*). Moreover, there will be an aspect of validation when it comes to submitting a game code after choosing to join an existing game.

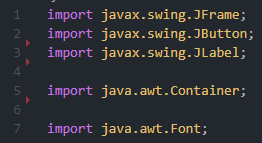
### Interface Frame and Layout – [02/01/2022]

This section of the development involves creating the basic foundation for the Graphical User Interface. In Java, this is achieved through creating a ‘JFrame’ and adding a ‘Content Pane’ container to it that handles all Swing components placed onto the frame. This container can use different layout managers to determine how the components are arranged on the frame.

#### Written Code

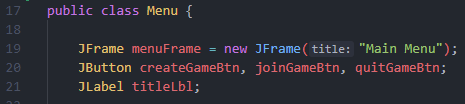
Below is the code that is used to create the backbone of the interface that will become the main menu of *Picture This!*.

This code makes use of multiple Swing components so will require imports of such components. At first, I used fewer lines for the imports as I simply imported the whole Swing library (using ‘import javax.swing.\*’) and the same for the AWT library. However, although this used less lines it was less efficient since it would load the whole library including parts that I did not need. Instead, I altered the code to only import the sections of the library that were required for the program.



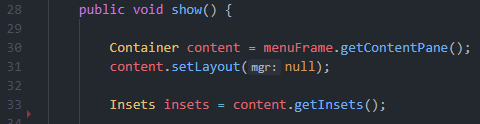
Imports part of the Swing library that allows me to use these components.

Imports part of the AWT library that allows me to use the Container component and change the font of text.



Instantiates a JFrame which will be used for the main menu. This frame is given a title, providing context to the player into what the interface is for.

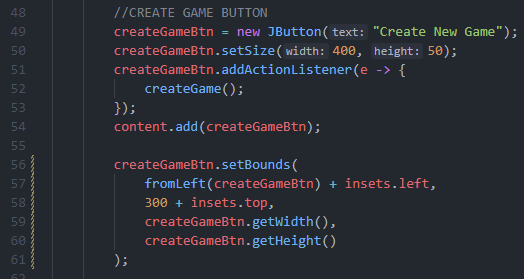
Instantiates Swing components that are to be used in the interface. Three different buttons are created for the different functions that can be performed by the player, and a label created to display the title of the game.



The Container component is instantiated and set a value of the frame’s content pane. It is set a value of null for the layout manager, meaning that I can manually set the position of the components within the Container. (See item 1 in *Testing Performed* below)

Since I am using manual assignments for the positions of the components, I need to set insets to determine how far from the borders the components are.

**Note:** The rest of the code snippets in this section are for the show() method.



Sets text and size of the button.

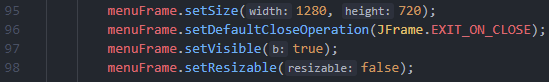
Adds an action listener to the button so that whenever the button is pressed, the method ‘createGame()’ is called. (See **Button Functionality** section below).

Adds the button to the container of the frame’s components.

Sets the position of the button within the container by taking in the new x and y coordinates and new dimension of the component. This controls how the button is displayed in the interface. (See item 2 in *Testing Performed* below).

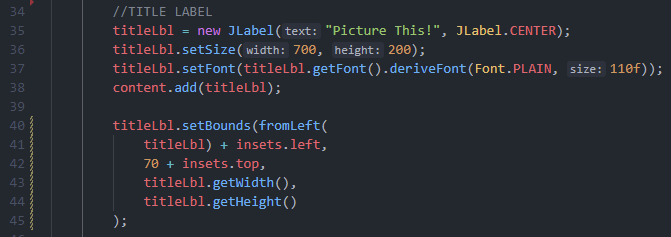
**Note:** This code is the same for the other two buttons, differing only in the method called when the button is pressed.

Sets the size of the frame (interface), specifying its width and height when it is created.



Sets the size of the frame (interface), specifying its width and height when it is created.

These lines, in order: allow the program to be stopped when the window is closed; allow the window to be seen when it is created; disable the ability to resize the window.



Sets the position of the label within the container by taking in the new x and y coordinates and new dimension of the component. This controls how the label is displayed in the interface. (See item 2 in *Testing Performed* below).

Adds the label to the container of the frame’s components.

Assigns text to display to the label, centring that text within the label. It also sets the size of the label itself and the size of the font used.

Main method is the first thing that is called when this Java (.java) file is compiled and ran.

Text

Description automatically generated

Creates new object of the Menu() class and calls the method show(). This creates the interface and launches it, with all components.

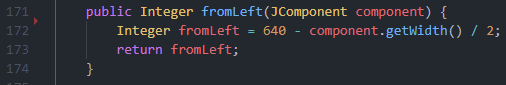
Graphical user interface

Description automatically generatedThe code above creates the resulting interface – a simplistic main menu that provides the user with a title and three buttons.

#### Testing Performed

There were a few aspects of the development of this section that required testing, which then led to the code being altered and improved. They are listed below:

1. **Layout Manager:** Initially I used the BorderLayout() manager to organise how the components are displayed on the frame. However, after testing how this looked, it was clear that this would not work since it was not providing me with the flexibility to set the components on the interface how I wanted. Due to this, I switched to not using a layout manager, which is more complicated but suitable for this interface since it is simplistic.
2. **Bounds of Components:** For the bounds of each button and label component, an integer for the new x coordinate must be taken in as a parameter. Since I want each component to be central horizontally, I had to manually trial and error the correct integer to get them all centralised. After repeated trials of testing to see if the components were central, I instead developed a new method to calculate the distance from the left that the component should be. This method is shown below:



Takes the width of the component passed into the method as a parameter and divides it by two, taking it way from the width of the frame.

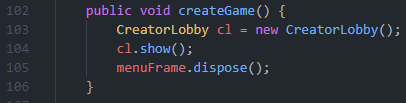
Returns the integer value representing how far away the component should be from the left border.

### Button Functionality – [05/01/22]

This section of the development of the ‘main menu’ interface consists of programming the functions that the buttons will carry out. There are three buttons that should provide a way for the user to select three different tasks: create a new game, join an existing game, quit to main menu.

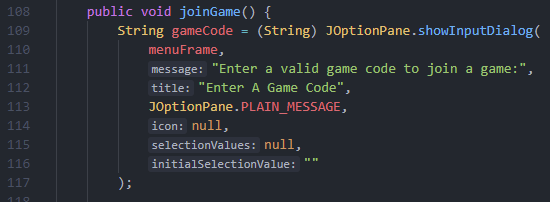
#### Written Code

The following is the code that I have developed to give the buttons functions, creating two new methods for when the plyer creates a new game, or joins an existing one. The simpler button – ‘Quit to Main Menu’ – will be easy to implement since all it involves is closing any open interface windows and stopping the program.



Closes the ‘main menu’ interface window and stops the program.

Creates a new object of the Creator Lobby class (see **Module 3: Creator Lobby Interface** below) and calls the show() method to display the new interface.



Creates a plain input prompt to the user from the menuFrame interface asking them to enter a game code. The string returned is saved in the variable gameCode.

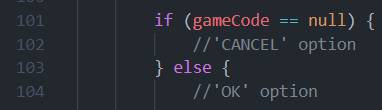
Settings selectionValues to ‘null’ means that it turns the combo-box into a textbox so that the players can type a game code. In addition, the initial value of the textbox is set to an empty string.

**Note:** The rest of the code snippets in this section are for the joinGame() method.



Closes the ‘main menu’ interface window and stops the program.

Adds an action listener to the button so that whenever the button is pressed the enclosed code is run.



Checks if the string entered from the input prompt is null, meaning that the ‘cancel’ button was selected.

The ‘ok’ button is selected since a string has been submitted.

(See item 1 in *Testing Performed* below).



Checks that the game code entered is a valid length of 5 characters and uses a regex to check that the string only contains letters and integers. (See item 2 in *Testing Performed* below).

Handles the two different exception errors that can be thrown.

Loops through the file using a while loop, reading each line (trimming the line so that all whitespace is removed) and checking if the line is equal with the game code entered. (See item 3 in *Testing Performed* below).

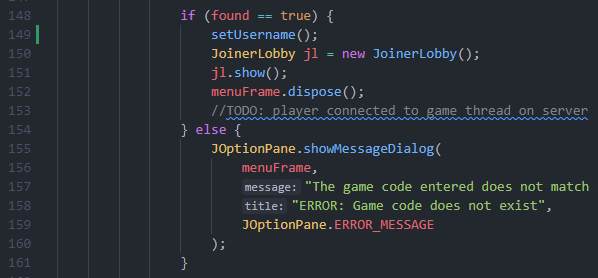
To reduce the workload that this while loop would provide, once the Boolean ‘found’ is set to true I could use the line ‘break;’ to exit out of the while loop, preventing it to loop through an unnecessary number of times.

I could add break to reduce usage of resources.

This code had to be placed into a try-catch block due to the errors that could be thrown.

Creates a FileReader and BufferedReader object so that the file can be read.

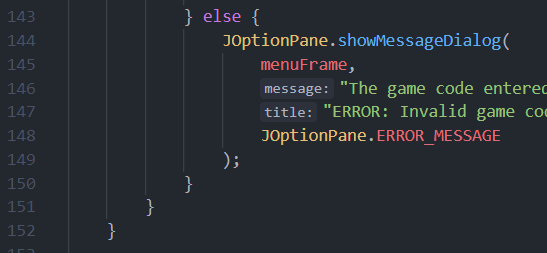
Closes the BufferedReader object to ensure there is no resource leak.



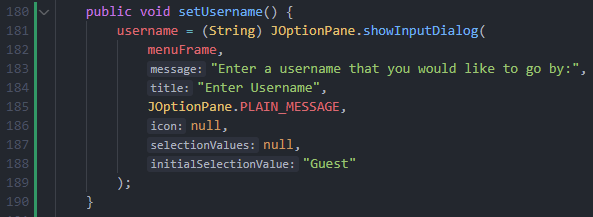
If the game code entered matches a game code in the file, create an object of the JoinerLobby class and call the show() method. Also, call the setUsername() method to store the player’s username.

If the game code entered does not match an existing game code, an error dialog box is displayed to the player.

Close the current window interface and connect the player to the thread on the server that corresponds to the game code.



If the game code entered is not valid, an error dialog box is displayed to the user.



If the game code is accepted and found to exist, this method is called in which the player is prompted to enter a username that they would like to go by. The initial value is set to the string “Guest” so if the player does not enter a value, this is what is used instead.

#### Testing Performed

There were a few aspects of the development of this section that required testing, which then led to the code being altered and improved. They are listed below:

|  |  |  |  |
| --- | --- | --- | --- |
| Purpose | Input | Expected Output | Observed Outcome |
| Ensure that the correct program path is followed | Null String | The input prompt is closed, and the player taken back to the main menu interface. | Test was successful and the outcome was as expected. |
| Ensure that the correct program path is followed | Extant String | Further validation is performed on the game code string entered since the user has selected the ‘ok’ button. | Test was successful and the outcome was as expected. |

1. **Null Values for Game Code:** When shown the dialog box prompt to enter a game code, the user could either enter a code and select ‘ok’ to submit it, or press ‘cancel’ to go back to the main menu. Testing was performed to ensure that the correct program branch was followed if a null string is entered (the player selected ‘cancel’).
2. **Game Code Validation:** Since the user can enter any string that they want into the input prompt, the inputted value must undergo validation. For security reasons, the game codes must follow set rules about their character length and format – they should only contain capital letters and numbers. Below is a table of tests to ensure correct functionality.

|  |  |  |  |
| --- | --- | --- | --- |
| Purpose | Input | Expected Output | Observed Outcome |
| Ensure that only valid game codes are accepted | ‘ABC12’ | The game code is accepted and the program proceeds to read the file to compare the game code. | Test was successful and the outcome was as expected. |
| Ensure that only valid game codes are accepted | ‘ABCDE’ | The game code is accepted and the program proceeds to read the file to compare the game code. | Test was successful and the outcome was as expected. |
| Ensure that only valid game codes are accepted | ‘12345’ | The game code is accepted and the program proceeds to read the file to compare the game code. | Test was successful and the outcome was as expected. |
| Ensure that only valid game codes are accepted | ‘aBC12’ | The game code is rejected and the program proceeds to output an error prompt to the user. | Test showed that there was an error with my regex. Since I only wanted capital letters allowed, restricting lowercase letters, I must change my regex to ‘^[A-Z0-9]\*$’ |
| Ensure that only valid game codes are accepted | ‘ABC123’ | The game code is rejected and the program proceeds to output an error prompt to the user. | Test was successful and the outcome was as expected. |
| Ensure that only valid game codes are accepted | ‘ABC$2’ | The game code is rejected and the program proceeds to output an error prompt to the user. | Test was successful and the outcome was as expected. |
| Ensure that only valid game codes are accepted | ‘ ‘ | The game code is rejected and the program proceeds to output an error prompt to the user. | Test was successful and the outcome was as expected. |

Below is an example of one of the invalid test inputs, and the expected error output when the string is submitted - the test being clearly successful.

Graphical user interface, text, application, Word

Description automatically generatedGraphical user interface

Description automatically generated

1. **Searching codes.txt File:** Each line of the file is checked against the inputted game code in order to determine if the entered string matches that of an existing game code in the file. This search can undergo testing to ensure that it is working properly so that codes are not wrongly identified as correct, and vice versa.

|  |  |  |  |
| --- | --- | --- | --- |
| Purpose | Input | Expected Output | Observed Outcome |
| Ensure only pre-existing game codes are accepted | Game code from codes.txt | The game code is accepted and the program proceeds to load the ‘Joiner Lobby’ interface and closes the ‘Main Menu’ interface. | Test showed that even codes from the file were being rejected. This was because I was using the ‘==’ operator to compare the line and game code. Instead, I should have been using the ‘.equals()’ method so that the actual data of the strings were being compared. |
| Ensure only pre-existing game codes are accepted | Game code not from codes.txt | The game code is rejected and the program proceeds to output an error prompt to the user. | Test was successful and the outcome was as expected. |

Graphical user interface, application

Description automatically generatedGraphical user interface, application, Word

Description automatically generatedBelow is an example of an invalid test input, and the expected error output when the string is submitted – the test is clearly successful.

### Client Feedback on Module 1

Since I had carried out a lot of testing for this module – developing the ‘Main Menu’ interface – it was important to discuss with my client all of the tests and changes that had occurred in the development so far.

I discussed with my client how I had to change multiple aspects of my code including the regex, that was used to define the characters permitted in the game code string, and also the way I compared the two game code strings, one from the codes.txt. file and the other being inputted by the player. For both of these instances, the client agreed that it was the right decision.

When looking at the code, my client suggested that I use a different icon for the error prompts, rather than the red ‘error’ icon, so that the player feels less confronted and more relaxed. I thought this was a good idea and plan to look into using a more friendly ‘warning’ sign icon for my error prompts.

## Module 2: ‘Joiner Lobby’ Interface

### Development Aim of Module 2

The development of this module focuses on creating the Graphical User Interface (GUI) for when the player tries to join an existing game. This interface is a lobby where the players are taken to after entering a valid game code. There will be a label, displaying a message to the players to wait until the creator starts the game, and a timer that shows the elapsed time since the player entered the lobby. I will have to use testing to ensure that the timer is working as intended.

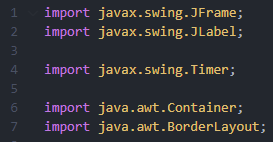
### Interface Frame and Layout - [08/01/22]

This section of the development involves creating the basic foundation for the Graphical User Interface (GUI). In Java, this is achieved through creating a ‘JFrame’ and adding a ‘Content Pane’ container to it that handles all Swing components placed onto the frame. This container can use different layout managers to determine how the components are arranged on the frame.

#### Written Code

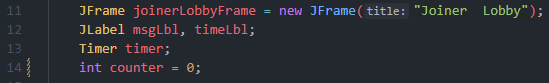
Below is the code that I have developed for the ‘Joiner Lobby’ interface in *Picture This!*. It is one of the simplest GUIs compared most of the other interfaces in my game, in terms of layout of components, so will not require as much programming.

Since this interface will contain Swing components and other features of both the Swing and AWT library, it will require a few imports at the start of the code.



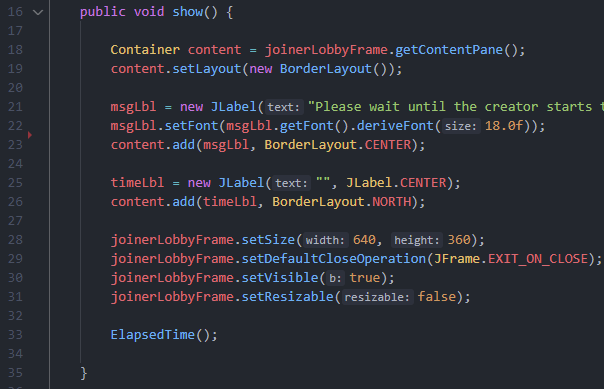
Imports part of the Swing library that allows me to use these components.

Imports part of the AWT library that allows me to use the Container component and set the layout manager of the Container to BorderLayout.



Instantiates the required Swing components, setting a title for the frame of the interface.

Creates a variable of data type Integer and sets it a value of 0. This counter is used for incrementing the timer. (See item 1 in *Testing Performed* in **Timer Implementation** below).



The Container component is instantiated and set a value of the frame’s content pane. It is set a value of a new BorderLayout() object for the layout manager since this is a simple interface consisting of few components, so I do not need toto manually position the components.

Assigns text to the label to display (this is simply a message informing players to wait for the creator to start the game), sets the font size of the text and then adds the label to the frame’s content pane container with a centre position.

Assigns empty text to the timer label (this will be overwritten with the elapsed time as soon as the ElapsedTime() method is called to start the timer) and adds the label to the frame’s content pane container with a north position.

Calls ElapsedTime() method which initialises and starts the timer. (See **Timer Implementation** section below).

Sets the size of the frame (interface), specifying its width and height when it is created. Also, these lines allow the program to be stopped when the window is closed; allow the window to be seen when it is created; disable the ability to resize the window.

#### Testing Performed

Since this section of the module simply consists of displaying two labels on the interface, there isn’t any testing required.

### Timer Implementation – [09/01/22]

The aim of this section of the module is to develop and implement a timer that can track the time elapsed since the player was taken to the ‘Joiner Lobby’ interface. In order to ensure that the timer is working effectively and keeping time, testing can be carried out.

#### Written Code

This code is for the timer component to be implemented into the ‘Joiner Lobby’ interface and will consist of one method that is then called when the frame is being loaded. By doing it this way, the timer will always display the time since the start of the interface, since it is started when the interface starts.



Creates a new timer and sets the delay between each count to be 1000 milliseconds. During each count, the counter variable is incremented by one and outputted in a formatted string to the label.

Sets the timer’s initial delay to 0 milliseconds (meaning the timer starts straight away) and starts the timer.

#### Testing Performed

There was one aspect of the development of this section that required testing, which then led to the code being altered and improved. It is listed below:

1. **Timer Start Integer:** Since the timer is started immediately once the ‘Joiner Lobby’ interface is initialised and started, having the timer count from zero (the counter variable being set an integer value of 0) means that the counter immediately shows that one second has passed. After testing I realised that this was occurring and formulated a solution where I can instantiate the counter variable with a value of negative one instead of zero. All other aspects of the timer work as intended.

### Client Feedback on Module 2

Since I had carried out a lot of testing for this module – developing the ‘Joiner Lobby’ interface – it was important to discuss with my client all of the tests and changes that had occurred in the development so far.

After discussing the change to the timer that I made (beginning the incrementation of the counter variable from negative one instead of zero), my client agreed that it was the right choice since it improved the visual look of the timer and made it clearer to the player how much time had elapsed.

Whilst talking about the visible display of the components in the interface, my client suggested that I could potentially make the timer label larger so that the players could see it easier. However, he agreed with my counterargument when I stated that the timer is not the main feature of the interface so does not need to be the centre of the player’s attention. My client felt that this was the right decision, and the label should stay the size that it is.

I informed my client that this interface was only shown until the creator of the game chose to begin the game. My client was pleased that this feature of the lobby – that he requested – had been implemented and summarised how this section was well developed.

## Module 3: ‘Creator Lobby’ Interface

### Development Aim of Module 3

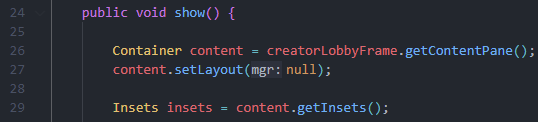
The development of this module focuses on creating the Graphical User Interface (GUI) for when the player tries to create a new game. This interface is a lobby where the player is taken to after selecting the ‘create new game’ button on the ‘main menu’ interface. It will consist of: two labels, used to inform the player of the different inputs required; two text boxes, allowing the player to enter the different inputs required; a button, used to submit the values and continue to the ‘drawing phase’ interface.

### Interface Frame and Layout – [11/01/22]

This section of the development involves creating the basic foundation for the Graphical User Interface (GUI). In Java, this is achieved through creating a ‘JFrame’ and adding a ‘Content Pane’ container to it that handles all Swing components placed onto the frame. This container can use different layout managers to determine how the components are arranged on the frame.

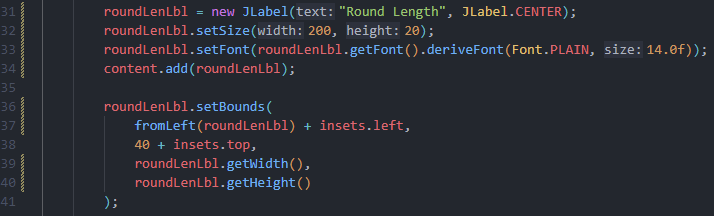
#### Written Code

Below is the code that I have written for the ‘creator lobby’ interface in *Picture This!*. This is a simpler Graphical User Interface (GUI) so will not require as much programming as the other interfaces. However, the development of this interface will still require multiple imports from the Java Swing library and other associated libraries from the language.



Creates an object of the Container class that stores value of the content pane for current frame. The layout of this content pane is set to null. This means I can manually set the layout of the components on the interface.

Instantiates an Object of the Insets class, which stores the insets for the content pane. This allows me to use these insets to manually set the layout of the different Swing components.

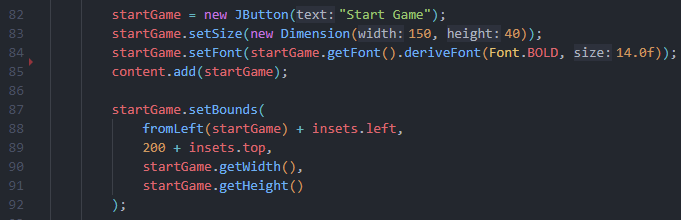


Creates a label with the text “Round Length”, centring the text in the centre of the label. The size of the label is also set, and the font size is chosen.

The label component is added to the interface’s frame.

Sets the bounds to be used by the label when added to the interface. This is how I manually position the component onto the frame. This uses the same method as shown in as shown in **Method 1**’s *Testing Performed* section.

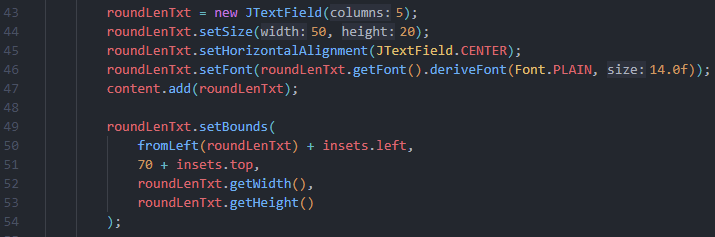
**Note:** This is the same code used for the other label.



Creates a new button with the text “Start Game”, specifying the size. The font size is also set.

The button component is added to the interface’s frame.

Sets the bounds to be used by the button when added to the interface. This is how I manually position the component onto the frame. This uses the same method as shown in as shown in **Method 1**’s *Testing Performed* section.

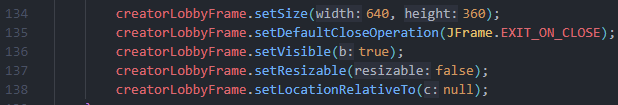


Creates a new text box, specifying the size. The text is set to be centred in the middle of the textbox, and the font size set.

The text box component is added to the interface’s frame.

Sets the bounds to be used by the text box when added to the interface. This is how I manually position the component onto the frame. This uses the same method as shown in as shown in **Method 1**’s *Testing Performed* section.

**Note:** This is the same code used for the other text box.



Sets the size of the frame (interface), specifying its width and height when it is created.

These lines allow the program to be stopped when the window is closed; allow the window to be seen when it is created; disable the ability to resize the window.

#### Testing Performed

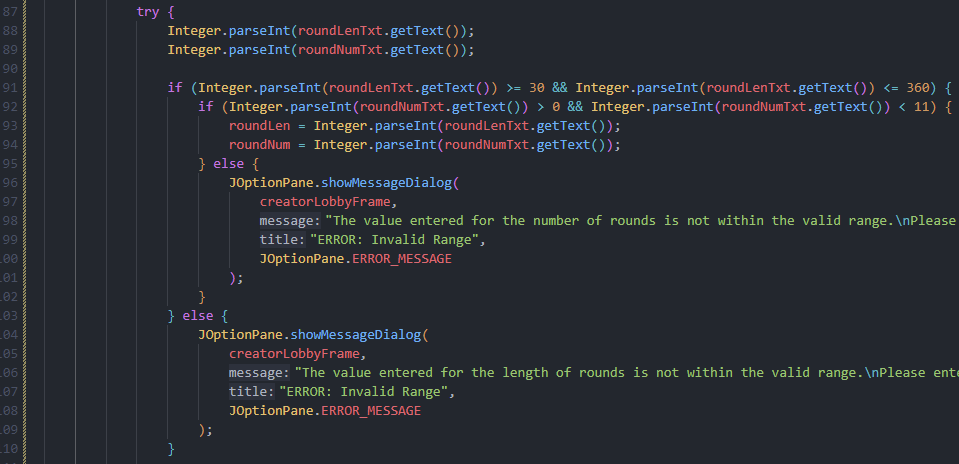
Since this section of the module simply consists of displaying a few Swing components on the interface, there isn’t any testing required.

### Button Functionality – [12/01/22]

This section of the development of the ‘creator lobby’ interface consists of programming the function that the button will carry out. This is the process that should occur when the player chooses to start the game.

#### Written Code

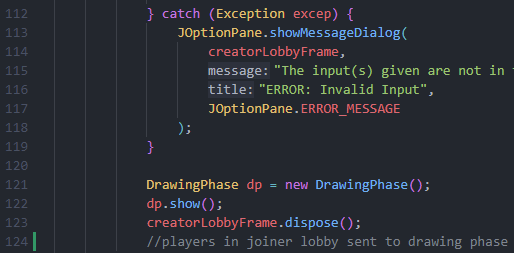
To add functionality to the button, I added an Action Listener object to the button. In doing so, I would have to make sure I validated any inputs to ensure the security of my game. Below is the code I wrote for this:



Parses the value inputted by the player creating the game to an Integer data type. This essentially checks if the inputted value is an Integer, generating an error if not (shown in the code annotation below). (See item 1 in *Testing Performed* below).

Checks the different inputs that they are within the required range, if so then storing the values inputted into static variables. (See item 2 in *Testing Performed* below).

If the inputted values are not within the required range, output error messages are displayed to the player to inform them to input values in the required range.



If an error is thrown (the value entered is not an integer data type) then an error output message is displayed to the player to inform them to input a valid data type.

Once the inputs have been validated, the ‘drawing phase’ interface is shown, and the current ‘creator lobby’ interface is closed. This is also where the players in the ‘joiner lobby’ interface will be taken to the ‘drawing phase’ interface.

#### Testing Performed

During the development of this module, there were many aspects of my code that I tested and modified. Below is a summary of the testing carried out and what changed within my code.

1. **Validating Integer Data Type of Inputs:** Since the inputs from the creator of the game for both the length of the rounds and the number of rounds played must be integers, this must be checked and appropriately dealt with if errors arise. Below is a table of testing that I carried out to ensure that this was working as intended.

|  |  |  |  |
| --- | --- | --- | --- |
| Purpose | Input | Expected Output | Observed Outcome |
| Check that Integer data types are accepted | Integer values for both inputs | The values are accepted and the program proceeds to check that the inputs are between the required ranges. | Test was successful and the outcome was as expected. |
| Check that non Integer data types are not accepted | Non Integer values for both inputs | The values are not accepted and the program proceeds to output an error message informing the player to input values of the correct data type. | Test was successful and the outcome was as expected. |
| Check that non Integer data types are not accepted | Non Integer values for only one input | The values are not accepted and the program proceeds to output an error message informing the player to input values of the correct data type. | Test was successful and the outcome was as expected. |

1. **Validating Range of Inputs:** Once validated for the correct data type, the inputs must be validated to ensure that they are within the required range. These ranges are used so that game rounds are not insensibly quick or frustratingly long, or there is not an insensible amount of rounds. This validation must be tested to ensure that it is working as intended.

|  |  |  |  |
| --- | --- | --- | --- |
| Purpose | Input | Expected Output | Observed Outcome |
| Ensure only values within the correct range are accepted | Both values within the correct range | The values are accepted and the program proceeds to show the player the ‘drawing phase’ interface, closing the current interface. | Test showed that for values on the boundary (e.g. a round length of 30), the program rejects the values. This was since I was checking if the value was > 30 and not >= 30. This has been changed within my code. |
| Ensure values outside the required range are not accepted | Both values outside the correct range | The values are not accepted and the program proceeds to output an error message informing the player to input the values within the correct range. | Test was successful and outcome was as expected. |
| Ensure values outside the required range are not accepted | One value outside the correct range | The values are not accepted and the program proceeds to output an error message informing the player to input the values within the correct range. | Test was successful and outcome was as expected. |

### Client Feedback on Module 3

Due to the changes made within my code for this module, it is important to discuss the module with my client; I held a meeting with him and discussed the whole of the module.

Firstly, we talked about the changed to the range validation where I edited my code to start comparing the inputs against >= 30 rather than > 30. My client agreed that this was the best solution to the issues discovered during testing and praised my solution.

Another topic of this module that we discussed was how I was validating the data type of the input by the player, ensuring that only Integers were supplied to the code. My client pointed out that the implementation of a ‘try catch’ statement to validate this was not the most professional, or optimised, method of achieving the function required. This is since in professional code, exceptions should try to be avoided and code should be built so that it reduces the probability that an exception occurs. If exceptions do occur, they should be handled in an appropriate way. However, although my code shows unprofessional characteristics, due to the context in which I am developing this game, and the time constraint that I face, the client felt that it was not necessary to spend time finding an alternative method to such a small issue that does not affect the overall function of my code.

## Module 4: ‘Drawing Phase’ Interface

### Development Aim of Module 4

The development of this module focuses on creating the Graphical User Interface (GUI) for when the player is set a word to draw. It will be a more complicated interface than the previous ones I have developed, consisting of more components and some that I have not used before. The player will be able to use a range of tools to draw the word including a brush tool, eraser tool and clear tool. The tools will all be selected through buttons, and the thickness of the brush can be changed using a JSpinner component. There will be a label at the top of the interface indicating to the player what word to draw, and the player will then draw that word on the canvas component in the centre of the interface. Included within the drawing phase interface will also be a timer that indicates the time left in the phase.

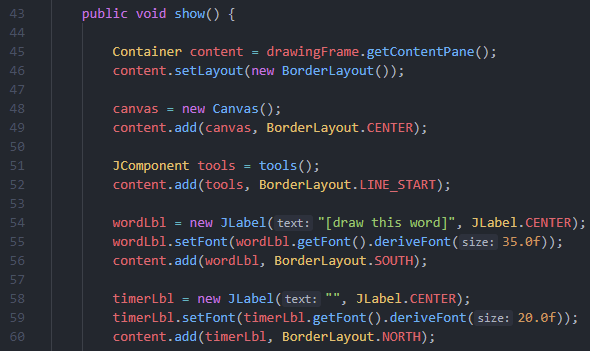
### Interface Frame and Layout – [14/01/22]

This section of the development involves creating the basic foundation for the Graphical User Interface (GUI). In Java, this is achieved through creating a ‘JFrame’ and adding a ‘Content Pane’ container to it that handles all Swing components placed onto the frame. This container can use different layout managers to determine how the components are arranged on the frame.

#### Written Code

Below is the code that I have developed for the ‘Drawing Phase’ interface in *Picture This!*. This is the most complicated Graphical User Interface (GUI) that is in my game since it contains many different Swing components that all require a lot of programming for their implementation.

Due to the size of this section in terms of code and the amount of components used, I have not included the imports that I have written. To see these, find the **Code Listings** section at the bottom of this report and view the ‘DrawingPhase.java’ file.



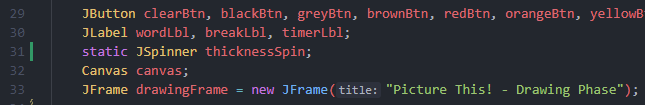
The Container component is instantiated and set a value of the frame’s content pane. It is set a value of a new BorderLayout() object for the layout manager since this works best for the interface due to each component residing on the borders. This means that I do not have to manually position the components myself. (See item 1 in *Testing Performed* below).

Creates a new object of the Canvas class so that I can implement this component into my interface. This is what the players will draw on. I have added this component to the window in the centre.

Creates JComponent object of the returned value from the tools() method. This method (shown below) creates a JPanel of all the buttons and the spinner used for drawing. This component is then added to the interface on the left hand side border.

Creates a JLabel Swing component that will display the word to be drawn to the players. It is set a font size and then added to the interface at the bottom border.

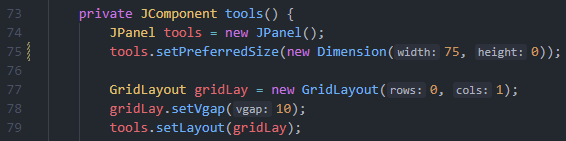
Creates a JLabel Swing Component that will display the time left in the drawing phase to the players. It is set a font size and then added to the interface at the top border.



Instantiates Swing components that are to be used in the interface including all the buttons for the tools and any necessary labels. Also, an object of the Canvas class is created so that it can be implemented onto the interface (see **Drawing Canvas** section below).

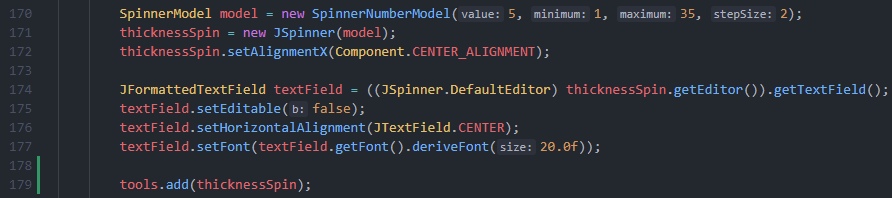
Instantiates a JFrame which will be used for the main menu. This frame is given a title, providing context to the player into what the interface is for.

Instantiates a static spinner component that is used to change the thickness of the brush tool. The keyword static means that it can be used anywhere in the class.



Creates a JPanel Swing component to contain all of the different drawing tool components. I also set the size of this panel by giving a specified width and height as two parameters. However, since this JPanel is being displayed on the left hand border, it means that the height of the component is dependent on the height of the interface, so is set to 0 here.

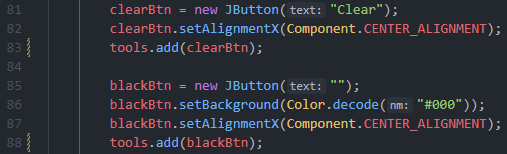
Creates a new layout manager of the type GridLayout for the JPanel. This is to structure the buttons and spinner within the JPanel itself. It is instantiated with 0 rows and 1 column, meaning that it has one column but any number of rows since zero means unspecified. I have also set the vertical gap between each tool component to be 10 pixels. The layout manager is then applied to the JPanel. (See item 2 in *Testing Performed* below).



Creates a JSpinner Swing component using a ‘model’ that sets the initial value, minimum value, maximum value, and the amount the value increases or decreases when selecting the spinner buttons. I have centrally aligned this along the x axis so that it is central in the JPanel. (See item 3 in *Testing Performed* below).

To make the process of validating the inputs of thickness easier, I created a formatted text field of the spinner, which essentially takes just the textbox part of the spinner and allows me to customise it how I like. I could then prevent it being editable, meaning I would not have to validate the inputs being inputted, since players could only choose from a range of values. Also, I set the text to be central in the spinner since it looked better, and I also set the font so that it took up more of the area.

Adds the JSpinner component to the JPanel so that it is displayed on the interface.



Creates a JButton Swing component that will be used to clear the canvas, removing all the current ‘paint’ from it. It is set the text of “Clear” so that the players are clear of its purpose. It is given a centre alignment along the x axis so that it appears in the horizontal centre of the JPanel. The tool is then added to the JPanel.

Creates a JButton Swing component that will be used to select the black colour ‘paint’. Once selected, the brush tool will draw in this colour until another colour is selected. The text of the button is set to a blank string and the background colour set to the same colour as the colour it will draw as. This is so the players can easily identify the purpose of the button, and also makes the interface look nicer. Like the button above, this button is given the centre alignment along the x axis. The tool is then added to the JPanel.

**Note:** All the other buttons used for colours (including the eraser tool which is simply a white colour) follow the same structure and use the code so it is unnecessary for me to show the code for them. This also makes it easy for me to implement more colours quickly.

After all the code above, the current state of the ‘Drawing Phase’ interface is show below. It is clear and is easy for the user to see where everything is, making it an intuitive Graphical User Interface (GUI). My client agrees and personally really likes the minimalistic design.

Table

Description automatically generated with medium confidence

#### Testing Performed

There were a few aspects of the development of this section that required testing, which then led to the code being altered and improved. They are listed below:

1. **Frame Layout Manager:** Originally, I was displaying and organising the components on the interface manually (like in the ‘Main Menu’ interface) however, this was a more complicated interface, and I was getting lots of errors when it came to how the different components reacted to each other when they overlapped etc. Due to these issues arising constantly, I thought it was sensible to adopt a layout manager and after debating the various possibilities of ones I could use, I decided to use the BorderLayout one. This was because the layout that I wanted for this Graphical User Interface (GUI) already followed a border-like pattern.
2. Graphical user interface, text, application, Word

   Description automatically generated**Panel Layout Manager:** Originally, I was using the BoxLayout manager for the buttons since I thought that it would be the easiest to use to make the buttons stack upon one another. However, after failed attempts of this method, I researched the different alternative options and concluded that using a separate JPanel to store all the ‘tools’ would be a more sensible idea. Using a layout manager separately on this panel to organise the components within. Before switching this is what the interface looked like:

It was clearly the right decision since it prevented me from trying to manipulate and organise the components with a convoluted layout manager such as the BoxLayout manager.

1. **Thickness Spinner Validation:** Since I have selected for the JSpinner component that is responsible for the thickness of the brush tool to not be editable and also have both a maximum and minimum value, I must test that this is actually the case, and the spinner is working as intended.

****

Shown above, when I loaded the interface, the spinner was more greyed out than before suggesting that it was not editable. To further prove that the code was functioning as intended, I tried to click on the display of the number yet could not select it. Even after trying to type numbers into what used to be the textbox part of the spinner, I could not change the value. The only way to change the value was to use the spinner buttons, therefore the spinner was working as intended.

In addition, after selecting both the ‘up’ and ‘down’ spinner buttons repeatedly, increasing and lowering the number respectively by a value of two (another characteristic I had chosen), the value of the thickness would stop at what I had set the maximum and minimum to be. This proved that the spinner was working as intended.

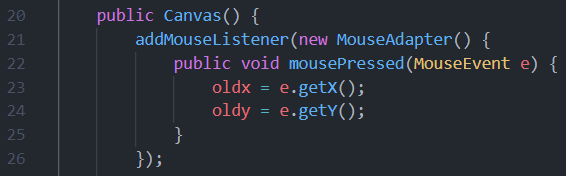
### Drawing Canvas – [17/02/22]

This section of the development involves creating a separate class for the drawing aspect of the ‘Drawing Phase’ interface. This is so that I can easily implement this component into the interface by creating an object of this class and placing it onto the Graphical User Interface (GUI).

#### Code Written

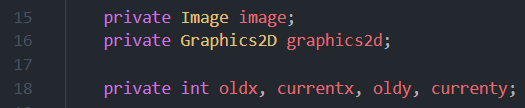
This is the code that I have written for the canvas component. The actual drawing on the canvas will require me to track the mouse movements made by the player. I can then draw lines onto the canvas between the previous and new coordinates of the mouse, creating the illusion of drawing. However, I am unsure if drawing a line between the two points would be the best method. Potentially drawing something else between the two coordinates of the mouse position (for example, a small circle) would perform better. This is something that I will have to test to determine.

This is a more complex interface that I am going to develop since it consists of using new features of the AWT library that I have not used in my previous development of *Picture This!*. This means that I will have many imports for this Canvas class.



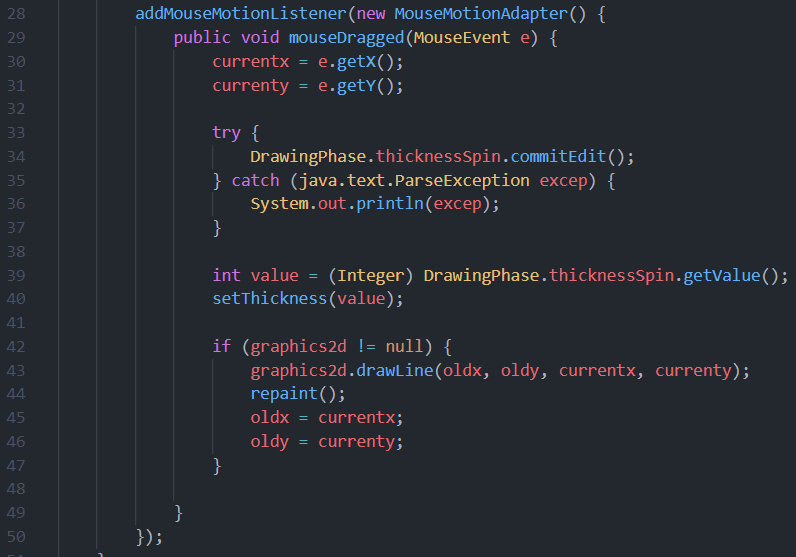
Adds an action listener to the canvas to listen for when the mouse is clicked. When it is clicked, the x and y coordinates of the mouse cursor are retrieved and stored in the oldx and oldy variables.

This is the Canvas classes Constructor method which is the method that is called whenever an object of that class is instantiated.



Instantiates variables that will be used to draw onto the canvas.

Instantiates variables of data type Integer to store the old and current coordinates of the mouse, with the x and y components separate.



Adds an action listener to the canvas to listen for when the mouse is dragged. When the mouse is dragged, the following code is run.

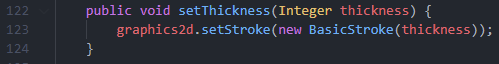
**Note:** This method is also set within the Constructor method shown above.

Retrieves the x and y coordinates of the mouse cursor after being dragged and stores the values in the currentx and currenty variables.

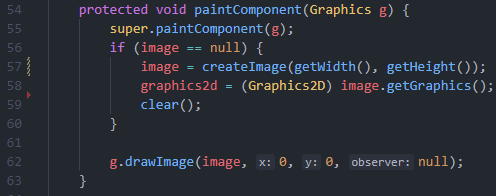
When the value of the thickness is changed by the player, this change must be committed to the editor of the JSpinner component so that I can then use the new value. If I didn’t do this, the old value could still be stored instead.

This stores the value of the JSpinner component, corresponding to the thickness of the brush tool, in a variable of data type Integer. This value is then parsed into the setThickness() method shown below.

This code is responsible for drawing a line onto the screen between the two sets of coordinates that represent the mouse cursor’s new and old position. After the line is drawn, the repaint() method is called in order to apply the change to the canvas. The current x and y coordinates are now stored in the ‘oldx’ and ‘oldy’ variables so that the next line drawn can be from this point.



Sets the size of the stroke to be the value of the thickness chosen by the player in the JSpinner component. This stroke size is what is used when the lines are drawn onto the canvas.



This is an overridden method that is essential for drawing in Java Swing. I must call the parent (using super) of the method so that I do not override all of the method, but just the section I need to override.

If an image isn’t already created, one is created and set the size of canvas. By using ‘getWidth()’ and ‘getHeight()’, I can easily obtain the dimensions of the canvas. This image allows ‘paint’ to be drawn to the canvas.

The graphics of the image are obtained and cast into the Graphics2D data type. I also call the clear() method to ensure there is no ‘paint’ on the canvas.

The image is now drawn onto the canvas at the top left (both the x and y coordinates set to 0).

#### Testing Performed

Since this is a feature that does not run as an individual script of code, the only way to test this component is to implement it into the actual program and test it there. Due to this, I will be testing the drawing capabilities in the **Button Functionality** section below whilst testing the different colours.

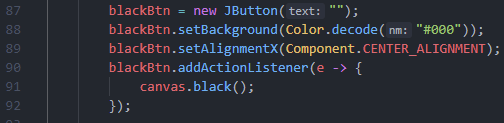
### Button Functionality – [19/02/22]

This section of the development of the ‘drawing phase’ interface consists of programming the functions that the buttons will carry out. There are buttons for all the available colours that the player can select and also a button each for the eraser and clear tool.

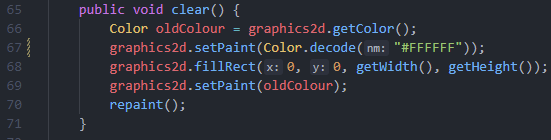
#### Code Written

Since I have already developed the layout of buttons, I simply need to create functions for each button that can be called when the button selected and also add an Action Listener object to each button so that the input can be detected.

Going back to the code from the ‘DrawingPhase’ class shown above, I can add a line of code to each button where I create them. Shown below is an example of this being implemented to the button for the black colour.



This new line adds and Action Listener object to the button. When an input to the button is detected (i.e. the button is clicked), the method in the ‘Canvas’ class for the corresponding button is called.

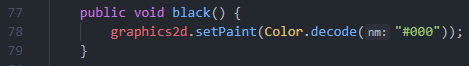


This code is executed once the ‘clear’ button has been selected. Firstly, the current colour of the ‘paint’ is stored in a variable of data type Colour so that this colour can be set again once the clear tool has performed its intended function. Then the paint is set to white since this is the same colour as the canvas.

Creates a rectangle that is the size of the canvas and covers the whole of the component. It then draws this rectangle to cover all previous drawings.

The ‘paint’ colour is set back to the colour that it was before the clear() method was called. The repaint() method is then called to apply the new changes.

**Note:** This code is from the ‘Canvas’ class and is the same for all colours (the eraser tool is simply white).



Sets the colour of the ‘paint’ to the hexadecimal value of “#000” which represents the colour black. This code is repeated for all the other colours with their respective hexadecimal values. Using hex to represent the colours instead of the limited built in values from the Swing library provides me more freedom to get the colours that I and the client want.

#### Test Performed

This testing is concerned about the selection of the different buttons to ensure that they function as intended. This includes all of the different colours, and also the two other tools (i.e. the eraser tool and clear tool). I have already tested the input to the thickness JSpinner component; however, I have not ensured that the thickness is actually set for the ‘paint’. I will cover this in my tests too.

Firstly, I tested each colour to ensure that, after the button was pressed, the colour was changed. In addition, I checked that the colour I last selected was still selected once I cleared the paint from the canvas. This also proved that the clear() method was working. Both of these tests proved valid, and the components were working as intended (see the snapshot of the canvas below).

Secondly, I tested that the thickness values set by the player would change the thickness of the lines drawn. This worked as intended since the thickness of the brushes increased proportionally to the size set in the spinner component. However, as evident through the snapshot of the canvas below, the thicker lines would be more jagged, not creating a straight line. This was because the lines drawn were much larger and sometimes were created at an angle, resulting in the corners showing. Due to this, I drew circles instead of lines to see if this would look better. Instead, there were multiple gaps between the circles when moving the mouse faster than a crawling speed, so this was not an appropriate change. The results of both these tests are shown below.

A picture containing timeline

Description automatically generatedThis was the canvas state after testing the colour buttons, eraser tool button and the thickness spinner:

Table

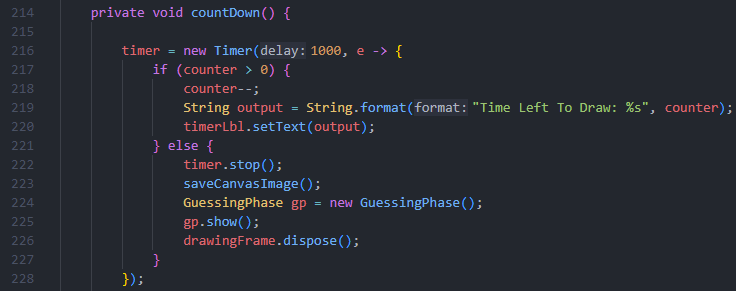
Description automatically generatedThis was the canvas state after testing drawing circles instead of lines:

### Timer Implementation – [21/02/22]

This section of the development involves creating a timer that is used to track the amount of time left that the player has to draw. Once this timer reaches zero, the drawing phase will end and the player will be taken to the guessing phase, where they can begin guessing what the drawings represent. This timer will allow players to see how long they have left and know when the phase is close to ending.

#### Code Written

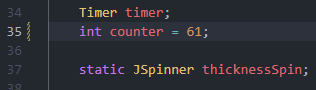
The code written below is all provided within one method (except the instantiation of the timer component and the ‘counter’ variable of data type Integer) that can be called when the interface is first loaded and shown. This method will start the timer and handle what happens once the timer is finished, meaning that the show() method does not need to handle anything with the timer.



Creates a new Timer object and sets the delay of the timer to 1000 milliseconds (one second) meaning that the code within the timer is run every one second. (See item 1 in *Testing Performed* below).

If the counter is still above zero (there is still time left in the drawing phase), decrement the counter by one and output the new value of the counter. (See item 2 in *Testing Performed* below).

If the counter is zero (there is no time left in the drawing phase): stop the timer; load the guessing phase interface, closing the current interface also; call the saveCanvasImage() method (see **Saving Snapshot of Drawing** section below).



Instantiates Swing timer component and variable for the counter used in the timer. The value of this variable is set to half the round length (chosen by the player) plus one. I add one since the timer will immediately decrement the counter by one when it starts. For purposes of testing the drawing phase independently, I have set the counter value to an integer.

#### Testing Performed

This is a smaller section to develop and is not a standalone independent class so cannot simply easily be tested individually. However, I can add temporary lines to the code to check the state of variables and outputs throughout the running state of the code. This is shown through the use of the ‘System.out.println()’ line in my testing.

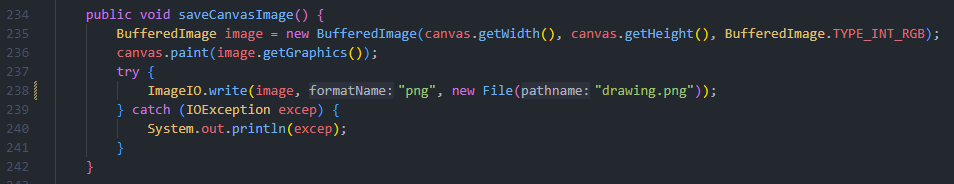
1. **One Thousand Millisecond Intervals:** Since this timer is meant to run iterations of the code every 1000 milliseconds, I can use the ‘System.out.println(counter)’ to output the value of the counter to the terminal in my Integrated Development Environment (IDE) every iteration. I can check this against a stopwatch to ensure that the counter is being decremented every second. When running the program with the temporary line of code, I could confirm that the timer was running at the correct delay.
2. **Label Output:** From the previous test, I have confirmed that then timer is running with a delay of one second, yet I must also test that the label is outputting the counter at the same rate, otherwise the timer will not work properly. I can do this by simply using a stopwatch and checking the time it takes the label to change**.** When doing so, I could validate that the label was displaying the counter at correct intervals - the timer is working as intended.

### Saving Snapshot of Drawing – [22/02/22]

This section of the development focuses on the final aspect of the drawing phase, occurring once the timer is finished and the players have finished drawing – saving a snapshot of the drawing on the canvas. This is necessary so that the images can then be outputted to the interface for the players in the guessing phase.

#### Code Written

The following code is responsible for the saving of the player’s drawing. The code for this process is all within a single method that can be called once the timer is over. Being all within one method can help me post-development if maintenance needs to be performed since my code will be more organised.



Creates a variable of the data type BufferedReader and stores the value of an image that is the width and height of the canvas. The image is also set 8-bit RGB colour. The image is then drawn to the canvas.

A file is created, and the image is written to it, unless there is an IOException error in which the error stack is printed to the output terminal in my Integrated Development Environment (IDE). This file is overwritten if it already exists.

#### Testing Performed

This is a smaller development section than the others in this module so will require less testing, however, whilst developing I encountered an error where the whole screen was being screen shotted and saved to the file. This was an issue, so I did some research into taking snapshots in Java Swing and came to the conclusion that I wasn’t correctly specifying the component that I wanted to take a snapshot of. After realising this, I could edit my code to ensure that I was only taking a snapshot of the canvas.

In order to ensure that the code is working as intended and the player’s drawings are being saved to the file, I did multiple trials of drawing my own image on the ‘drawing phase’ interface and seeing the resulting file saved. After repeating this process many times, there were zero times at which an error occurred which suggests the method is working correctly.

### Client Feedback on Module 4

This has been one of the larger modules that I have developed and documented so far so it is important to get the client’s feedback on the development and testing of my code.

Firstly, we discussed the layout of the interface to ensure that my client was happy with the chosen design and organisation of the components on the interface. He supported the decision that I had chosen to use the BorderLayout layout manager claiming that it made the interface have a more structured and organised feel. My client felt that the components (e.g. the buttons and spinner) were the right size and helped add to the minimalistic feel of the interface.

In terms of changes to my code, there weren’t many since most testing came out valid. However, we did discuss the way I was drawing the ‘paint’ onto the drawing canvas and whether the client was happy with my decision to use lines and not change to using circles. The client felt that the circles made for a smoother line when drawing at a moderate speed but claimed that it was not worth changing since drawing at faster speeds made the game look less professional and more unpolished.

Overall, the client was feeling confident with my progress on the project and felt like the development so far showed the right characteristics of achieving the product to the client’s standards.

## Module 5: ‘Guessing Phase’ Interface

### Development Aim of Module 5

The development of this module focuses on creating the Graphical User Interface (GUI) for when the player is set a guess what the drawings represent. It will be a more complicated interface than the previous ones I have developed since it will consist of a Swing component that I have not used before – a JTextArea. Within this interface, please can view the drawings that with players have created since they will be displayed at the top of the screen. They can then input their guesses into a textbox, pressing a button to submit the guess. A timer, displayed in the bottom left of the screen, will indicate the time left for the player to guess the word.

### Interface Frame and Layout – [24/02/22]

This section of the development involves creating the basic foundation for the Graphical User Interface (GUI). In Java, this is achieved through creating a ‘JFrame’ and adding a ‘Content Pane’ container to it that handles all Swing components placed onto the frame. This container can use different layout managers to determine how the components are arranged on the frame.

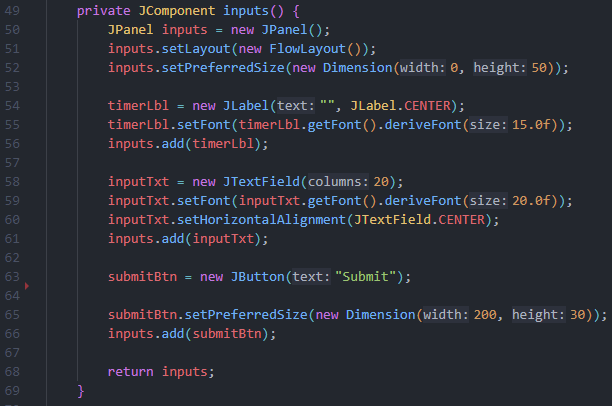
#### Code Written

Below is the code I have written in order to achieve the layout and organisation of the components on the guessing phase interface. This interface will consist of a panel at the top in order to display all the drawings made by the players, a text box where the player can enter their guesses, a button to submit the guesses made by a player, and a text area to display the outputs of the chat box.



The Container component is instantiated and set a value of the frame’s content pane. It is set a value of a new BorderLayout() object for the layout manager since this works best for the interface due to each component residing on the borders. This means that I do not have to manually position the components myself.

Creates and instantiates different panels that contain the different components. These methods are shown below. The containers are then added to the interface’s frame.

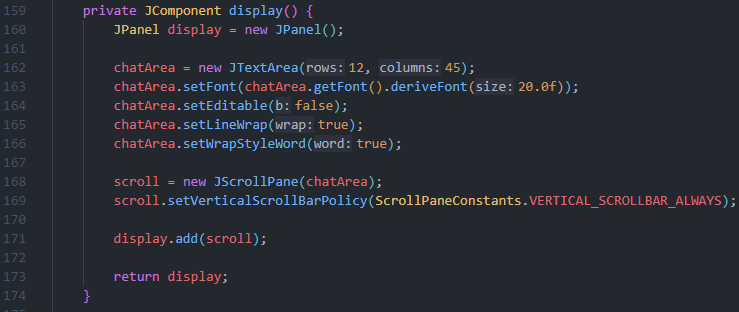


Creates a new JPanel object that is used to store the components used for inputs to the chat box. The FlowLayout layout manager is used so that each component follows after the other, and the size of the panel set.

Creates a new label used to display the current time left to guess. This timer is referenced in the method used for the counter (see **Time Implementation** below). The font size is set, and the label is added to the panel.

Creates a new text box used to input the player’s guesses. The font size is set, and the text specified to be central in the text box. This component is then added to the panel.

Creates a new button that is used to submit the player’s guesses. The size of the button is set, and the button is added to the panel.



Creates a new JPanel object that is used to store the chat area component where all the player’s guesses will be displayed. A layout manager does not need to be specified since there is only one component displayed.

Creates a new text area setting both the sizes of the component and the font used. The ability to edit the text area is disabled and the characteristic to wrap text inside the text area is enabled.

Creates a new JScrollPane object for the text area that enables a vertical scroll bar to be shown on the text area. This component is then added to the interface’s frame.

The other JPanel used in my interface is for displaying the player’s drawings to the other players. This is covered in another section of the module below since it is a more specific panel than the previous aforementioned ones. (See **Displaying Player Drawings** section below).

#### Testing Performed

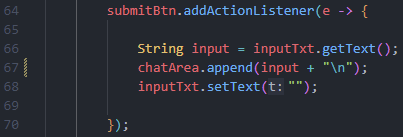
Since this section of the module simply consists of displaying different panel components on the interface, there isn’t any testing required. I have already tested the layout mangers that I have used in the other modules above, so it is not necessary to test them in this module.

### Chat Box Functionality – [25/02/22]

This section of the module consists of creating the basic functionality of the chat box component in the interface. In the **Interface Frame and Layout** section above, I developed code to display the chat box component to the player. In this section, I aim to add functionality to this component and develop a way for players to input their guesses, as to what the drawings represent, which are then displayed to the other players.

#### Code Written

The code shown below shows the development of the chat box feature within this interface. When the ‘submit’ button is pressed, it will read the text inputted into the textbox by the player and output this text to the other players. This development of code does not include creating the validation algorithm that will be used in the final solution. This algorithm is to be developed at a later date and the development of it explained at a later date (see **Module 7: Essential Algorithms** below).



I added an Action Listener object to the button so that when it is selected, I can specify code to perform.

Creates a variable of data type String to store the text that is retrieved from the text box. This text is appended to the text area, with a new line ending so that every new input is on a new line. (See item 1 in *Testing Performed* below). This is also where the text will be sent to the server to be displayed to the rest of the players.

The text box is cleared ready for another input.

#### Testing Performed

This is a rather simple development of code but still requires testing since it involves inputs from the player. This is more the case since I am using a component – JTextArea – that I have not used before so do not have experience with.

1. **Displaying Inputs to the Text Area:** I ran a simple test of just inputting different values into the text box to check that the strings were being outputted to the text area. After performing this test, I was encountering an error in that the text was being displayed and appended onto a single line of the text area, and not a new line for each item. After researching this common issue, I realised that I must include a new line (“\n”) tag after the text I want to display so that the next item will start on a new line.

### Displaying Player Drawings – [26/02/22]

This section of the module involves the development of the code to display the different drawings, in order, onto the interface so that players can try to guess what they represent.

#### Code Written

The code below shows the method that creates a new panel and adds all of the drawings to it. Since this section of development only consists of displaying components onto the frame, it can be completed using a single method that is called during the show() method shown above.



Creates a new JPanel object that is used to store the different player drawings, displaying them to the players. The size of this panel is set along with the background colour, and the panel is given padding.

Creates new Image Icon objects that retrieve the image files of the drawings and scales them to the correct size. The image icons are then displayed by parsing them into labels, and then adding these labels to the panel.

#### Testing Performed

Since this section of the module simply consists of displaying different components on the interface, there isn’t much testing required. However, I wanted to ensure that the files were being read properly so played through *Picture This!* a few times to test that the drawing I drew in the drawing phase would be the same drawing displayed in the guessing phase. For each time I tested it, the test was successful so I can be assured that my code for this section is error free.

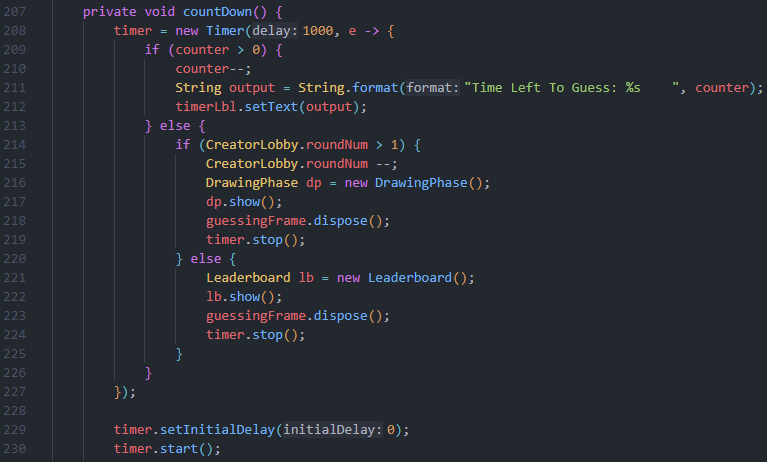
### Timer Implementation – [27/02/22]

This section of the development involves creating a timer that is used to track the amount of time left that the player has to guess. Once this timer reaches zero, the guessing phase will end and depending on the number of rounds already played, the player will be taken to the drawing phase again for another round or taken to the ‘leader board’ interface where they can see the current scores of all the players. This timer will allow players to see how long they have left and know when the phase is close to ending.

#### Code Written

The code written below is all provided within one method (except the instantiation of the timer component and the ‘counter’ variable of data type Integer) that can be called when the interface is first loaded and shown. This method will start the timer and handle what happens once the timer is finished, meaning that the show() method does not need to handle anything with the timer.

Instantiates Swing timer component and variable for the counter used in the timer. The value of this variable is set to half the round length (chosen by the player) plus one. I add one since the timer will immediately decrement the counter by one when it starts.



Creates a new Timer object and sets the delay of the timer to 1000 milliseconds (one second) meaning that the code within the timer is run every one second.

If the counter is still above zero (there is still time left in the guessing phase), decrement the counter by one and output the new value of the counter.

If the counter is zero (there is no time left in the guessing phase) then check whether or not there are more rounds to be played (see item 1 in *Testing Performed* below). If there are more rounds to be played then the player is taken to the ‘drawing phase’ interface again, otherwise the player is taken to the ‘leader board’ interface.

Start the timer, ensuring it starts immediately.

#### Testing Performed

This section is similar to a section in module 4 – the ‘drawing phase’ interface – where I use the same method but this time I have made slight moderations for its use within the ‘guessing phase’ interface. Due to this, most of the testing I would perform on this method has already been performed in the earlier section for that module. (See *Timer Implementation* in **Module 4: ‘Drawing Phase’ interface** above). However, there is still a section of code unique to this module which must be tested.

1. **Number of Rounds:** Since I am executing two very different pathways of code depending on the result of a variable form another class, it is important to test that the pathways are reached when they should so that the method is working as intended. To test this, I let my game run through after specifying the number of rounds to be 3 in the creator lobby. After going through 3 iterations of the drawing and guessing phase it was expected that I, as the player, would be taken to the ‘leader board’ interface. However, the ‘drawing phase’ interface loaded again, and it went through another round iteration. Each time I tested this; my game would play one more round than the value specified by the player. Since I was checking the number of rounds left was greater than 0 (“roundNum > 0”), to fix this issue I simply changed the condition in the if statement to be “roundNum > 1”. After more testing, this seemed to fix the issue and my code was now working as intended.

### Client Feedback on Module 5

This has been one of the larger modules that I have developed and documented so far so it is important to get the client’s feedback on the development and testing of my code.

Firstly, we discussed the layout of the interface to ensure that my client was happy with the chosen design and organisation of the components on the interface. He supported the decision that I had chosen to use the BorderLayout layout manager claiming that it made the interface have a more structured and organised feel. My client felt that the components (e.g. the buttons and text boxes) were the right size and helped add to the minimalistic feel of the interface.

When reviewing the code that I had programmed, my client pointed out how I had only developed the implementation of displaying the drawings to all the players when there were three players only. I explained how this was due to getting a basic structure of the whole program before then redeveloping sections of code in order to add those certain features. However, my client was pleased with me only developing then code for displaying three drawings since adding the capabilities to display the amount for different players could be easily added post development due to the simplicity of the code. However, my client would rather I focus on the development of other aspects of *Picture This!* due to the time constraints that I face.

Another topic that I discussed with my client was how I was scaling the images when displaying them. My client raised the issue that using this method of scaling would reduce the quality of the images and make them harder to see, being a detriment to the players trying to guess. However, although it is true that the method my code uses will reduce the quality of the image files, it does not matter since when actually viewing the drawings in game, the quality is not bad in a way that negatively affects the player’s guessing ability. My client agreed with this and claimed that my method was suitable for this project.

In terms of the changes to my code, my client felt that my solution to the fact that too many rounds would play was the most efficient and sensible method that I could’ve chosen.

## Module 6: ‘Leader Board’ Interface

### Development Aim of Module 6

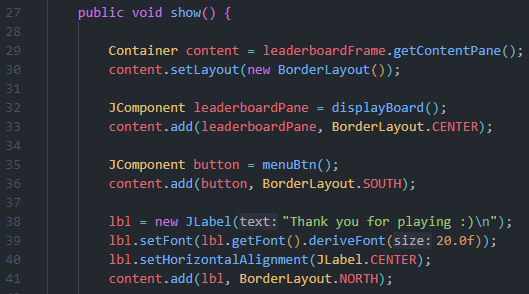
The development of this module focuses on creating the Graphical User Interface (GUI) that the plyer is taken to when they finish the game. This interface will be simpler than the two previous ones since it will only consist of: a label, used to display a thank you message for playing the game; a button, used to take the player back to the ‘main menu’ interface; a text area, to display the leader board scores in descending order according to points.

### Interface Frame and Layout – [11/03/22]

This section of the development involves creating the basic foundation for the Graphical User Interface (GUI). In Java, this is achieved through creating a ‘JFrame’ and adding a ‘Content Pane’ container to it that handles all Swing components placed onto the frame. This container can use different layout managers to determine how the components are arranged on the frame.

#### Code Written

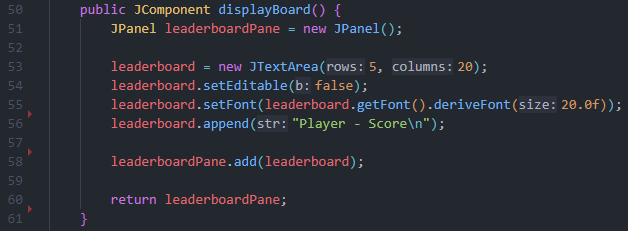
The code shown below is from the development of the interface used to display the leader board to the players. This code is responsible for the organisation and layout of the Swing components on the interface, not being concerned about the functionality of the interface at this current time.



Creates and instantiates different panels that contain the different components. These methods are shown below. The containers are then added to the interface’s frame.

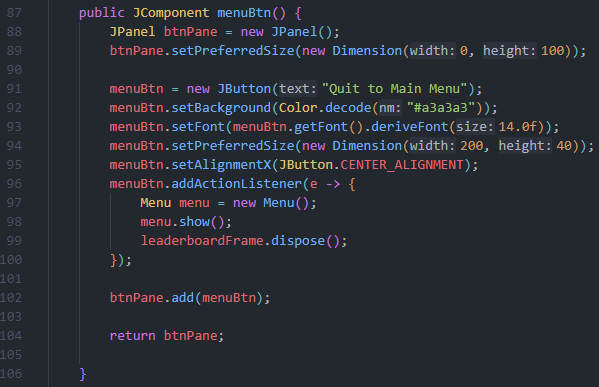
Creates a new label that is used to display a thank you message to the player. The label text font size is set along with the horizontal alignment to be central. The label is then added to the frame.

The Container component is instantiated and set a value of the frame’s content pane. It is set a value of a new BorderLayout() object for the layout manager since this works best for the interface due to each component residing on the borders. This means that I do not have to manually position the components myself.



Creates a new JPanel object that is used to contain the text area Swing component for the leader board to be shown.

The characteristics of the text area are set, including font size and initial text. The text area is also set so that it cannot be edited. The component is then added to the panel.



Creates a new JPanel object that is used to contain the button Swing component for the ‘quit to min menu’ button to be shown.

The characteristics of the button are set, including font size, text shown, background colour and component size. The x alignment is also set so that the button is in the middle of the panel.

An ActionListener object is added to the button so that a pathway of code can be executed upon interaction with the button. When the button is pressed, the ‘main menu’ interface is shown, and the current frame closed.

The button is added to the panel.

#### Testing Performed

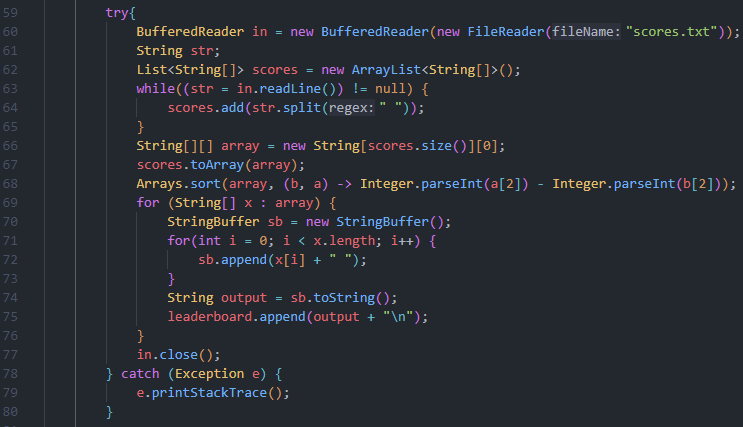
Since this section of the module simply consists of displaying different components on the interface, there isn’t any testing, that I have not already performed, required.

### Ordering and Displaying Leader Board Data – [15/03/22]

This section of the module is responsible for the development of how the data and scores of the players will be displayed onto text area for the leader board. The data in question will be the username and the score of the player with that username. This data should be sorted into descending order according to the score so that it is easy for players to see who is winning and losing in respective of the other players.

#### Code Written

The code shown below is for the process of displaying player’s username and score to the text area for the leader board. Since this will include data stored by all the players, it will utilise file reading and parsing this data from the file into an array in order to sort it. The code below is all programmed to execute within the displayBoard() method that is run when the leader board text area is instantiated.



A buffered reader object is created and passed a file reader object that is then passed the file path of the file where the scores of the players are stored. A null variable of data type String is also instantiated. This is to be used when reading the file, representing each line in the file. An ArrayList object is also created that will be used to store an array of each line in the file.

Creates a loop that continues until the whole file is read. With each iteration, the next line is read and split by its spaces, creating a two dimensional list consisting of the player’s username and score in separate indexes. An array of this length is then created, and the list converted to an array.

Sorts the array based on the third index which is where the scores are. The array is sorted so that the scores are in descending order (largest first and smallest last). (See item 1 in *Testing Performed* below).

Loops through each i index in the two dimensional array and combine the one dimensional array indexes to a larger string. This re-forms the lines of the files, but they are now ordered. The string is outputted to the text area with a new line tag at the end, so each score is outputted onto a new line.

Avoids a resource leak by closing the buffered reader.

Catches any exception that is thrown and handles it. This will most likely be an IOException due to the file reading code that I have implemented.

#### Testing Performed

This is a smaller section for the module so does not consist of much testing. However, the purpose of the code above is to order the scores in descending order so I should perform testing to ensure that this is functioning as intended.

1. **Sorting a 2d array:** Testing must be carried out to ensure that the sorting of the two dimensional array is being sorted correctly. To do this, I entered different scores in different orders into the ‘scores’ text file so that the program would have to order these numbers. The test was successful with the order being outputted as expected, showing that this code is working and executing as intended.

### Client Feedback on Module 6

This has been a smaller module so there is less to discuss with my client, however, it is good to keep constant communication, so I set up a meeting and we talked about the different aspects of the development for this module.

Firstly, we discussed the layout of the ‘leader bord’ interface and my client stated how they thought it was simple and intuitive to the player whilst also fitting the style of the other interfaces in *Picture This!*. My client suggested that more customisations could be made to the interface’s design such as adding a logo for my game, however we both agreed that this could be something for post development due to the current time constraints that I face.

In addition, we discussed the leader board text area and the information that is displayed to the player. My client agreed that the right amount of data is shown to the player on the leader board – this being the username of the player and their respective scores, with a hyphen (“-“) character splitting the two pieces of information in order to increase the readability of the text area.

At the end of the meeting, the client stated how he was pleased with all the interfaces that I have developed for *Picture This!*. He stated how they all followed the same look and feel, creating the feeling of a complete piece of software.

## Module 7: Essential Algorithms

### Development Aim of Module 7

The development of this module focuses on the creation of the three different key algorithms that are used throughout *Picture This!* and making up a significant amount of the development. These three algorithms programmed below are the ones that appear in the **Algorithms** section in the **Design** chapter earlier in this dissertation.

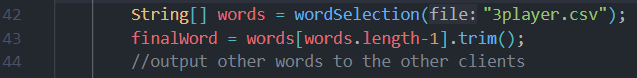
### Random Word Selection (Drawing Phase) – [17/03/22]

The ‘Random Word Selection’ algorithm is a process, performed before every round, that randomly selects which words should be given to the players to draw, by taking into account: the number of players in the game, given as an integer value from 2 to 4; and the difficulty that has been chosen for that round as a string from 3 different options (“easy”, “medium”, “hard”), determined using Dynamic Game Difficulty Balancing (see **Scoring and Dynamic Game Difficulty Balancing** section below).

#### Code Written

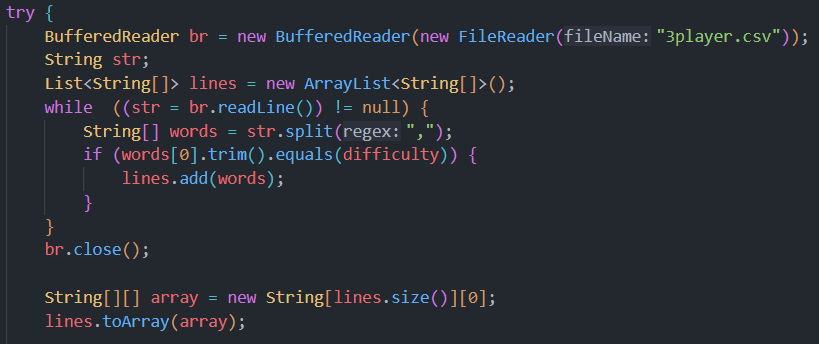
The code shown below is form the development of the **Random Word Selection** algorithm. This code will make use of many arrays and lists, and other data structures. It will also utilise the file reading capabilities within the Java programming language, having to read the different words from the ‘2player.csv’, ‘3player.csv’ and ‘4player.csv’ Comma Separated Values (CSV) files. I have developed this algorithm so that it can be used for any numbers of players playing *Picture This!*. This is done through using conditional statements to determine the number of players that there are. This code is executed before every new round of the drawing phase so I have included it within the ‘drawing phase’ interface class, however, this could change to being placed within the server class.

For the following code, I call the wordSelection() method within the show() method so that a word is found before every round of the drawing phase, and the rest of the code is from the ‘wordSelection()’ method.



Instantiates a new one dimensional String array that holds the returned values of the words.

Stores the valuable of the last index of the array into the String variable ‘finalWord’



Creates a BufferedReader object that is passed a FileReader object that is passed the name of the file. This is so that I can read the file.

Creates a String variable and a two dimensional ArrayList data type that are then used to search through the file. The file is read line by line, with only lines that match the correct difficulty being added to the list. The line is split by commas, creating an array.

The BufferedReader object is closed to avoid a potential resource leak. Also, the list data structure containing the lines of possible words is converted to a two dimensional array allowing for easier data manipulation.

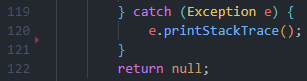


A random integer between 0 and the length of the array is created. This will be used to fetch a random word from the array.

Creates a loop, searching each i index in the array checking if its line number corresponds to the random integer created.

If the random integer created matches the list number of the current i index then the words to be given to the player are set and returned, depending on the amount of players that there are in the game.

If the random integer and line number do not match, the value of the line number is incremented, and the next i index checked.



Exceptions (most likely an IOException from using the buffered reader and file reader) are caught and handled.

Returns a value if no other values are returned.

#### Testing Performed

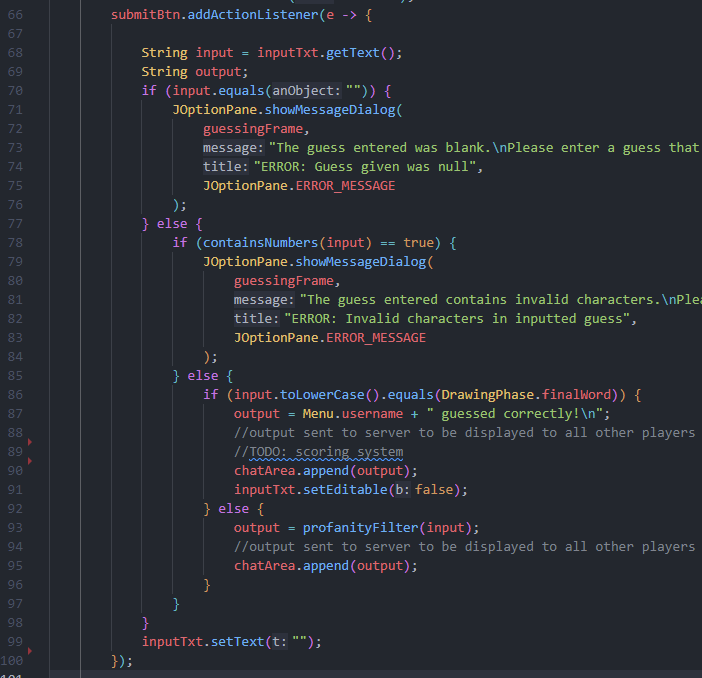
This algorithm is based upon randomness and its outcomes are determined mainly from the random integer created. Due to this, the best method of testing the different outcomes of my code is to run the code multiple times and get an idea of the different possibilities and the probability that they occur. In doing so, I can ensure that different words are being provided to the players each round.

### Validation of Inputs Including Profanity Filter (Guessing Phase) – [24/03/22]

This algorithm is a process that validates each input, made by any user, that is entered into the chat box function. It is a necessary algorithm to ensure that any guesses entered are validated and, where required, sanitised. It includes using a custom made profanity filter to prevent inappropriate language reaching the chat box where other players would be able to view it.

#### Code Written

The code shown below achieves the validation and profanity filter through String sanitation and manipulation, using conditional statements to determine which program pathway should be followed. The main block of code below (this excludes the additional methods called) is all included within an ActionListener object added to the ‘submit’ button in the ‘guessing phase’ interface.



Instantiates and input and output variable, both of the String data type. The input variable is set a value of that which is in the text box. This is what the player is trying to input.

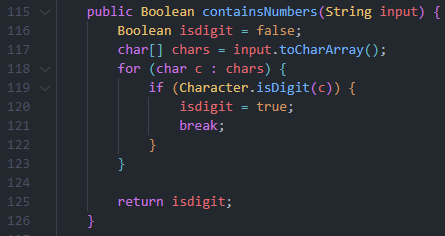
If the input provided by the player is empty then an error output message is shown to the player, informing them to enter a valid input. (See item 1 in *Testing Performed* below).

If the input provided by the player is not empty then the input is checked if it contains numbers. If so, an error output message is shown to the player, informing them to enter a valid input. (See item 2 in *Testing Performed* below).

If the input provided by the player contains does not contain numbers then the input is checked if it equals the same as the correct word. If so, the output message is set to a value that will inform all players that this player guessed correctly. This is also where their score will be recorded. (See item 3 in *Testing Performed* below).

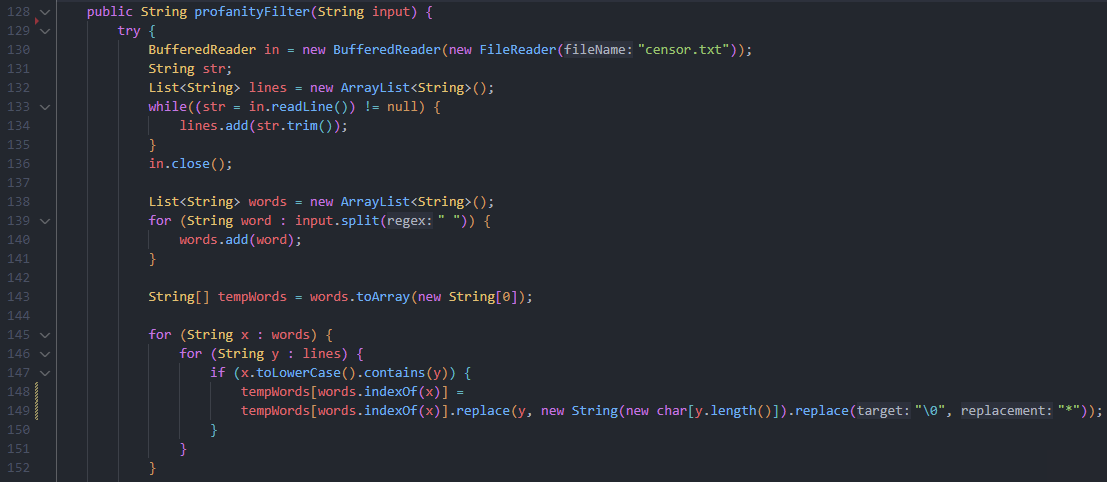
If the inputted value is valid but does not match the correct word, the output is set to the value of this input once passed through the profanity filter, which is performed through a method. (See item 4 in *Testing Performed* below).

Sets the text box to be empty, ready for another input.



Creates a one dimensional array of the characters in the input.

Iterates through each character in the array, checking if it is a number. If so, the loop is break and the Boolean variable ‘isdigit’ is set to true.



A buffered reader object is created and passed a file reader object that is then passed the file path of the file where the blacklisted words are stored. A null variable of data type String is also instantiated. This is to be used when reading the file, representing each line in the file. An ArrayList object is also created that will be used to store an array of each line in the file.

Creates a loop that continues until the whole file is read. With each iteration, the next line is read, trimmed (removing whitespace) and added to the list, creating a list consisting of the blacklisted words.

Creates a new ArrayList object that is to be used to store the words that have been inputted. A for loop is used to store each word from the input in a separate index of a one dimensional array.

Converts List data structure to Array data structure ready for manipulation.

Uses nested for loops to compare every inputted word against every blacklisted word, checking if the inputted word contains the blacklisted word.

If the inputted word does contain a blacklisted word, the word in the array is replaced with a series of ‘\*’ characters with the same length as the blacklisted word. (See item 4 in *Testing Performed* below).

#### Testing Performed

This section of the module consists of many inputs of all different kinds, increasing the need for testing. There are four selection conditions that can be met or not which determines the logic pathway pursued by the program. I will need to test each of these to ensure that the validation algorithm (including the profanity filter) is working as intended and not letting inappropriate language through so that it is displayed to all the other players.

1. **Empty Inputs:** This will test what occurs when empty Strings are submitted to the chat box and check that they are being rejected. Below is a table of test data that I have used to come to a conclusion as to whether this section of the algorithm is functioning correctly.

|  |  |  |  |
| --- | --- | --- | --- |
| Purpose | Input | Expected Output | Observed Outcome |
| Check that empty Strings are being rejected | Empty String | The value is not accepted, and an error message is outputted to the player informing them to enter a valid input. | Test showed that empty Strings were being accepted even though they shouldn’t be. After reviewing my code, I realised it was a similar mistake that I had made in earlier development where I was using the ‘==’ comparator operator when I should actually be using ‘.equals()’ |
| Check that empty Strings are being rejected | Non Empty String | The value is accepted, and the program proceeds to check if the input contains numbers. | Test was successful and the outcome was as expected. |

1. **Inputs Containing Numbers:** Since there are lists of blacklisted words, it is safer to reject inputs with numbers so that variations of these banned words are not created. For example, it prevents the use of the number one (1) instead of the letter L in order to bypass the profanity filter. Due to this, it is important to ensure that inputs with numbers are being rejected. Below is table of test data that I used to check this.

|  |  |  |  |
| --- | --- | --- | --- |
| Purpose | Input | Expected Output | Observed Outcome |
| Check that inputs with numbers in are not accepted | String containing numbers | The value is not accepted, and an error message is outputted to the player informing them to enter a valid input. | Test was successful and the outcome was as expected. |
| Check that inputs with numbers in are not accepted | String not containing numbers | The value is accepted, and the program proceeds to check if the input matches the correct word. | Test was successful and the outcome was as expected. |

1. **Valid Inputs That Match Correct Word:** After all the format checks, it is important to check if the, now validated, inputted word matches the correct word. If so, the player’s score can be recorded and a message explaining how they guessed correctly is outputted. Otherwise, another conditional selection statement has to be run. Checking that the inputted word matches the correct word is relatively simple, consisting of using the ‘.equals()’ method on the inputted value after undergoing input sanitation (i.e. the inputted value is converted to lower case so that it is the same format as the word to guess). Below is a table of the test data that I used to check this validation was working as intended.

|  |  |  |  |
| --- | --- | --- | --- |
| Purpose | Input | Expected Output | Observed Outcome |
| Check that correct guesses are dealt with properly | String that matches the correct word | The value is accepted, and the program proceeds to output a message to all player claiming the player guessed correctly. The score is also recorded. | Test was successful and the outcome was as expected. |
| Check that correct guesses are dealt with properly | String that does not match the correct word | The value is not accepted, and the program proceeds to perform a profanity filter check on the input. The return value is now outputted to all players. | Test was successful and the outcome was as expected. |

1. **Valid Inputs That Do Not Match Correct Word:** Since players can input guesses that are perfectly valid but not correct, there must be a logic pathway of my program that deals with valid inputs that do not match the correct word. This will also include using the profanity filter on any inputs given that are valid and do not match the correct guess. Through testing I can ensure that this program pathway is functioning as intended with no bugs. Below is the table of test data that I have used for testing.

|  |  |  |  |
| --- | --- | --- | --- |
| Purpose | Input | Expected Output | Observed Outcome |
| Check valid inputs that are not correct proceed to be filtered and outputted | Valid String that is not correct | The value is checked with the profanity filter and outputted to the chat box as it is. | Test was successful and the outcome was as expected. |
| Check valid inputs that are not correct proceed to be filtered and outputted | Valid String that is not correct, containing one blacklisted word | The value is checked with the profanity filter and outputted to the chat box with the blacklisted word censored. | Test was successful and the outcome was as expected. |
| Check valid inputs that are not correct proceed to be filtered and outputted | Valid String that is not correct, containing more than one blacklisted word. | The value is checked with the profanity filter and outputted to the chat box with the blacklisted words censored. | Test showed that the blacklisted words after the first blacklisted word are not censored. This was a big issue but after reviewing my code, I moved the line where I converted the list to an array to above where I checked for blacklisted words. This solved my issue and allowed me to complete further testing and confirm the algorithm was now working as intended. |

### Scoring/Dynamic Game Difficulty Balancing (Guessing Phase/Drawing Phase) – [31/03/22]

In *Picture This!*, the client wants the difficulty to be changed throughout the game at run-time so that it does not get too boring or too challenging for the players. This can be done through Dynamic Game Difficulty Balancing, by performing calculations after each round on the player’s scores to determine if the difficulty should be increased or decreased. After the calculations, the value will fall within a certain range corresponding to a difficulty.

To use this algorithm, the scoring system must also be established and programmed. This section also concerns itself with completing such a task, meaning this section of the development also includes creating the code responsible for the scoring of all the players.

#### Code Written

In *Picture This!*, the score the player is awarded is based on the time it takes them to guess the word, that the drawings represent, correctly. This is done by calculating the percentage of time left on the timer after guessing correctly and using this to take a percentage out of the maximum points that a player can be awarded. To calculate this programmatically, I will need to put this into an equation that I can include within my code.



Performs calculations on the time it took the player to guess correctly, finding the percentage of time left.

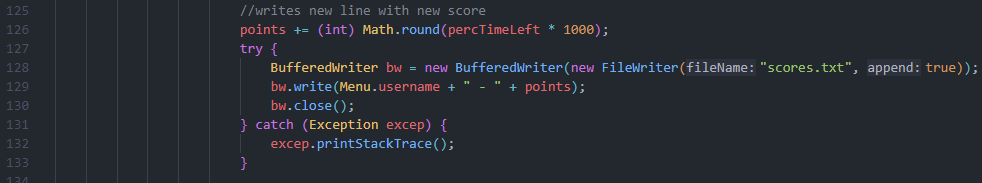
Instantiates both the scores file and a temporary scores file as File data types.

Creates a BufferedReader and BufferedWriter object used to read and write to files. Also, the target line to remove is set. This is so the new score can be written instead.

Iterates through file line by line, writing all the lines to the temp scores file unless it is the targeted line.

The original scores file is deleted and the temp scores file is renamed to the original name. The reader and writer are also closed to avoid resource leaks.

Catches and handles exceptions (most likely IOException from file reading).



Increments player’s score by the amount of points they achieved in the most recent round.

Creates BufferedWriter object, appends the player’s new score and then closes the writer to prevent a resource leak.

Catches and handles exceptions (most likely IOException from file reading).

In addition, the code for the Dynamic Game Difficulty Balancing algorithm can be developed and implemented into the software by reversing an altered version of the equation and using conditional statements to determine the boundary that the average player score falls between. This can be used to set the difficulty. The code of such is shown below.



Creates BufferedReader object in order to read the scores file line by line, adding up all the scores from each line to get a total score.

Calculates the average time taken (as a percentage) to guess the word correctly.

If the average time to guess of all players is between certain boundaries (checked through using conditional selections statements) then the difficulty is changed accordingly by either increasing or decreasing it by one level – sometimes maintaining the difficulty if the Dynamic Game Difficulty Balancing Algorithm deems it to be the correct difficulty for the current players.

Closes reader to ensure no resource leak and catches and handles any exceptions thrown.

#### Testing Performed

This algorithm does not include taking in inputs but simply altering pre-existing variables to ensure that they hold the appropriate value that my simple AI system deems necessary. Since this is the case, to test this software so that I can be certain as the developer that there are no prevalent issues in my code, I must run the program through multiple times and simply observe the outcome. Whilst doing this, I can alter made up scores for players in order to test if the difficulty is changed accordingly. When I tested the program three times with extremely high scores, I observed that the Dynamic Game Difficulty Balancing (DGDB) algorithm altered the difficulty to be one lower than the current setting. This was converse when testing with extremely low values for the player’s scores. This shows that the algorithm is functioning as intended and there are no issues. It is doing its purpose.

### Client Feedback on Module 7

This module was different to the other modules in that it only involved sections of code that fit elsewhere into the program and not into their own interface. These are key algorithms that are vital to the performance of *Picture This!* and are deemed necessary by the client in terms of requirements. This means that it was important to discuss the development of this module with my client.

To start off the meeting, we discussed the algorithms as a whole and how the client felt about them. He started by explaining how we he was pleased with their development and liked how they perform. However, he had a few queries about them that he wanted to discuss with me. These are explained below.

The first thing my client wanted to discuss was the way I represented the difficulty values within the Dynamic Game Difficulty Balancing (DGDB) algorithm and in *Picture This!* as a whole. He suggested that instead of using the characters that the different difficulties begin with, I could use the integers one, two and three so that I could easily increase and decrease the values of them during the DGDB algorithm. It would also reduce the amount of conditional statements that I would have to include within my code. However, I stated how I have already started using the characters so it would increase development time if I went back to change them. This is not optimal since I am facing severe time constraints. Also, code readability is increased when using the characters since it is clear what the values represent. The client agreed with this counter argument and suggested that I could alter this after release of *Picture This!*.

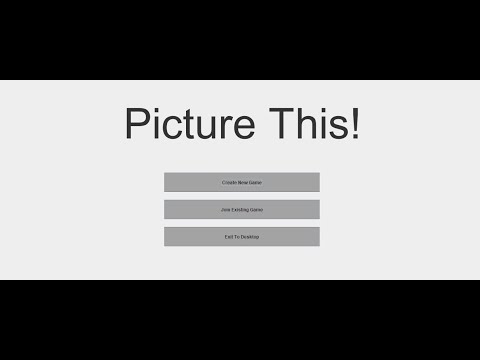
The other query that the client had about my development of this module was about the method I used to develop the process of updating the player’s score. This included copying over the scores file without the line with the player’s score and then writing the new score, replacing the file. My client was concerned that this was inefficient and would cause performance issues within my code. Although I understand the client and recognise that my method is not as efficient as it could be, I explained how the performance of my game was not affected negatively by this method so due to the timed deadlines that I face, it would be better for the project to move on and come back to this code post development, altering it then. My client understood and agreed that this would be better for a post development addition to *Picture This!*.

# Evaluation

This evaluation section will include many sections that allow me to reflect upon the development of *Picture This!* including testing, and its associated test data, a review of the success criteria from the analysis section, highlighting the current issues of my game, followed by a section on how to improve these issues and fix them after the initial release. I will also look at the usability features of my game and how the user experience can be improved.

## Testing and Test Data (w/ Video)

Below is a table of different tests that should be performed in order to ensure that *Picture This!*,as a whole program, functions as intended. The tests below are demonstrated in a video embedded and linked here: <https://youtu.be/jj8RNjQvedA>

[](https://www.youtube.com/embed/jj8RNjQvedA?feature=oembed)

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| ‘Menu Graphical User Interface’ Test Data | | | | |
| Test No. | **Aim of the Test** | **Test Input Data** | **Expected Outcome** | **Timestamp + Pass/Fail** |
| 1 | The player can select the ‘Create New Game’ button to create a new game | Mouse click on the button | The player is taken to the ‘Creator lobby’ GUI where they can select the settings for the game and are provided with a game code for their game (see the **Creating or Joining Games** section below) | 1.05  Pass |
| No mouse click on the button | No action should be performed, and the player should remain on the ‘Menu’ GUI | N/A  Pass |
| 2 | The player can select the ‘Join Existing Game’ button to join an existing game | Mouse click on the button | The player is prompted to enter a game code that is then validated (see the **Creating or Joining Games** section below) | 0.14  Pass |
| No mouse click on the button | No action should be performed, and the player should remain on the ‘Menu’ GUI | N/A  Pass |
| 3 | The player can select the ‘Quit’ button to quit *Picture This!* | Mouse click on the button | The *Picture This!* application is exited and closed | 0.06  Pass |
| No mouse click on the button | No action should be performed, and the player should remain on the ‘Menu’ GUI | N/A  Pass |

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| ‘Creating or Joining Games’ Test Data | | | | |
| Test No. | **Aim of the Test** | **Test Input Data** | **Expected Output** | **Timestamp + Pass/Fail** |
| 4 | The player can enter a game code that allows them to join a pre-existing game | Game code with length less than 5 characters  (e.g. H8J, D, 2389) | The game code should be rejected, and a prompt is shown to the player to enter a valid game code with the correct length | 0.19  Pass |
| Game code with length greater than 5 characters  (e.g. GW8J0L, I8DI310) | The game code should be rejected, and a prompt is shown to the player to enter a valid game code with the correct length | 0.27  Pass |
| Game code with length of 5 characters  (e.g. AU7F3, 49137) | The game code’s length should be accepted, and the game code will undergo further validation | 0.50  Pass |
| Game code containing a character that is not a letter or number  (e.g. H7S#P, WT3/X) | The game code should be rejected, and a prompt is shown to the player to enter a valid game code with the correct format | 0.37  Pass |
| No input is given | A prompt should be shown to the player to enter a valid game code | 0.43  Pass |
| 5 | The player is taken to the ‘Joiner lobby’ GUI once a valid game code is entered | A valid game code | The player should be taken to the ‘Joiner lobby’ GUI and waits until the game begins | 0.55  Pass |
| An invalid game code | The game code should be rejected, and the player is not taken to the ‘Joiner lobby’ GUI, remaining on the GUI to enter a game code | N/A  Pass |
| 6 | The creator of the game can enter an integer for the number of rounds to be played | Integer within the range of 1 and 10  (e.g. 1, 5, 4, 7, 9) | The value should not be rejected, and this is the number of rounds in the game | 2.11  Pass |
| Integer outside of the range of 1 and 10  (e.g. 0, 13, 25, 99) | The value should be rejected when the player tries to start the game, and a prompt is shown to the player to enter a valid value in the correct range | 1.47/1.53  Pass |
| Non-integer data type  (e.g. 0.5, hello, True) | The value should be rejected when the player tries to start the game, and a prompt is shown to the player to enter a valid value of the correct data type | 1.59  Pass |
| No input is given | The value should be rejected when the player tries to start the game, and a prompt is shown to the player to enter a valid value | 2.04  Pass |
| 7 | The creator of the game can enter an integer for the time length (in seconds) of each round | Integer within the range of 30 and 360  (e.g. 30, 100, 73, 119) | The value should not be rejected, and this is the time length (in seconds) of each round in the game | 2.11  Pass |
| Integer outside of the range of 30 and 360  (e.g. 3, 14, 360, 702) | The value should be rejected when the player tries to start the game, and a prompt is shown to the player to enter a valid value in the correct range | 1.13/1.20  Pass |
| Non-integer data type  (e.g. 0.5, hello, True) | The value should be rejected when the player tries to start the game, and a prompt is shown to the player to enter a valid value of the correct data type | 1.27  Pass |
| No input is given | The value should be rejected when the player tries to start the game, and a prompt is shown to the player to enter a valid value | 1.34  Pass |

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| ‘Drawing Tools’ Test Data | | | | |
| Test No. | **Aim of the Test** | **Test Input Data** | **Expected Outcome** | **Timestamp + Pass/Fail** |
| 8 | The player can draw on the drawing canvas by selecting the brush tool and moving the mouse | Mouse click and drag on the canvas whilst the brush tool is selected | ‘Paint’ should be outputted onto the drawing canvas at the mouse cursor’s position until the mouse button is released | 2.12  Pass |
| Mouse-click and drag on the canvas whilst the brush tool is not selected | The drawing canvas should remain in its current state and no new ‘paint’ is outputted onto it | N/A  Pass |
| 9 | The player can change the thickness of the brush by changing the thickness spinner | Spinner value changed, being both incremented and decremented. | The thickness changes accordingly to the value set in the spinner component | 2.18/2.23  Pass |
| 10 | The player can change the colour of the brush by selecting colour buttons | Mouse click on a colour | The brush tool should draw in this colour on the canvas when being used | 2.28  Pass |
| No mouse click on a colour | The brush tool colour should not change and remains the current colour | N/A  Pass |
| **NOTE:** This test should include testing each colour to ensure that it is labelled correctly | | | |
| 11 | The player can select the ‘Clear’ button to clear the drawing canvas | Mouse click on the button | The whole drawing canvas should be cleared with no ‘paint’ left | 2.53  Pass |
| No mouse click on the button | The whole drawing canvas should remain in its current state | N/A  Pass |
| 12 | The player can erase their drawings by selecting the eraser tool and moving the mouse | Mouse click and drag on the canvas whilst the eraser tool is selected | The ‘paint’ under the current position of the mouse cursor should be removed until the mouse button is released | 2.45  Pass |
| Mouse click and drag on the canvas whilst the eraser tool is not selected | The drawing canvas should remain in its current state and no ‘paint’ is removed | N/A  Pass |

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| ‘Chat Box for Guesses and Communication’ Test Data | | | | |
| Test No. | **Aim of the Test** | **Test Input Data** | **Expected Outcome** | **Timestamp + Pass/Fail** |
| 13 | The player can type and submit text into the chat box for guesses and communication | Text containing no numbers that matches the correct guess | The input should not be rejected and a message to all the players is outputted confirming they guessed correctly. The input is not outputted to the chat box as it is | 4.01  Pass |
| Text containing no numbers that does not match the correct guess | The input should not be rejected, and the input is outputted to the chat box for all players to see | 3.18/3.22/3.35  Pass |
| Text containing numbers | The input should be rejected, and a prompt is shown to the player to enter a valid input of the correct format | 3.25  Pass |
| Text containing no numbers that matches a blacklisted word | The input should not be rejected, and the input is outputted to the chat box for all players to see after the censored word has been replaced with a series of ‘\*’ characters | 3.41  Pass |
| Text containing no numbers that matches more than one blacklisted word | The input should not be rejected, and the input is outputted to the chat box for all players to see after all the censored words have been replaced with a series of ‘\*’ characters | 3.51  Pass |
| No input is given | A prompt should be shown to the player to enter a valid input | 3.28  Pass |

## Review of Success Criteria

Below are my success criteria that I set myself as shown in the **Analysis** section earlier in this documentation. In this evaluation I will determine how far I met these points of criteria.

|  |  |  |
| --- | --- | --- |
| No. | Criteria | Explanation/Justification |
| 1. | Main menu that provides the user with different actions | This is the first GUI that the user will see. It shows the title of the game, and two buttons: ‘Create new game’ and ‘Join existing game’. In addition, there will be a ‘Quit’ button that allows the player to exit the game. It will also include the additional feature of a settings tab if that feature is implemented. |
| 2. | ‘Create new game’ button | This button provides users the option to create a new game that their friends can join. It will redirect the user to a new GUI where they can start to select their options for the game. |
| 3. | Round length chosen by the player | This is a parameter of the game, set by the user who is creating the game. They can choose this freely, however, it will have a maximum and minimum value in order to keep the game enjoyable within the eyes of the client. The time given will be a combined time to include both the drawing and guessing phases in each round: it will be split evenly between the two. |
| 4. | Number of rounds chosen by the player | This is another parameter of the game that is set by the user who creates the game. It will also have a maximum and minimum value. |
| 5. | New game thread on the server created for each new game | Each time a game is created, a new thread on the server is created so that players can join this game, and multiple games can run on the server at any time. Each game will be assigned a game code so that players can specify which game they want to join. This code is given to the creator to share. |
| 6. | Lobbies for players whilst waiting for the game to start | These are to different GUIs, one for the creator of the game and the other for the players joining the game. The ‘Creator lobby’ interface will allow the player to select settings that are wanted for the proceeding game (see **Success Criteria No. 3** and **4** above). The ‘Joiner lobby’ interface will simply be a GUI that the players will wait at until the creator starts the game. |
| 7. | ‘Join existing game’ button | This button, located on the Main menu, is used for when players want to join a game that already exists. It will redirect the user to a new GUI where they have the option to enter a new game code. |
| 8. | Game code entered and validated | This is where the player enters an existing game code that corresponds to a game that has been created by another player. Once entered and submitted, the code is validated. |
| 9. | Connects user to game thread on the server with corresponding game code | After the game code is entered and validated, the player is connected to the thread on the server that corresponds to the game. The player is then redirected to the game lobby GUI. |
| 10. | Random word assigned to each player | Depending on the number of players (2 - 4) and the difficulty determined, a random word is chosen from a certain file of predetermined words. This word can be split into smaller words, the amount of which is the same as the number of players. These smaller words are assigned to each player at the start of the drawing phase. |
| 11. | Players can draw with a brush tool | The brush tool allows users to paint lines with the mouse. The brush tool will change the colour of the canvas, where the mouse cursor is located, to the colour selected (see below). |
| 12. | Players can use different colours of the brush tool | The player will be able to select different colours for the brush tool, providing more creativity towards their drawings. |
| 13. | Players can use different thicknesses of the brush tool | The player will be able to select different thicknesses for the brush tool, providing more creativity towards their drawings. |
| 14. | Players can remove their drawings with an erase tool | The eraser tool is used to remove any drawings on the canvas, where the mouse cursor is located. This will work by changing the colour to white, instead of actually removing the drawings. |
| 15. | The drawing canvas can be cleared using a ‘clear canvas’ button | This button allows for players to completely clear their canvas drawing with a single click of a button. This is if the player wants to redo their drawing or approach the drawing from another angle. |
| 16. | After the drawing phase, all the drawings are combined to create the larger word | Each of the player’s drawings are combined to create a larger image of all drawings, which represents the original longer word. This image is displayed to all players. |
| 17. | A chat box allows players to enter guesses and communicate with a working profanity filter | Players are to guess this word and enter their guesses into the chat box where the guess will be checked. This chat box can also be used to communicate with other players, helping the game’s inclusive feel – however, requires a profanity filter. |
| 18. | After the guessing phase, player scores are updated | After each round, the score for each player is calculated by taking into account the time it took them to guess the word correctly. The players current score is then incremented by the score they got in that round. |
| 19. | A leader board is updated and displayed | After the guessing phase, the new player scores are checked, and the leader board updated accordingly. The leader board is then displayed in a separate GUI after each round. |
| 20. | During the game, Dynamic Game Difficulty Balancing (DGDB) is used | DGDB is used so that engagement with players is maintained. Based on the average performance of all players, the difficulty of the words will either increase or increase. Changing the difficulty keeps the players entertained but not frustrated. |
| 21. | Music plays in the background in a loop | Jovial background music is played on a loop in order to create a more friendly and casual feeling towards *Picture This!* |
| 22. | Sound effect for the start of each phase | This sound effect is used to notify the player of the start of the phase (whether it be the drawing or guessing phase). Sound effects also improve the professional feel of a game. |
| 23. | Sound effect for guessing word correctly | This sound effect is used to notify the player of the guessing the word correctly. Sound effects also improve the professional feel of a game. |
| 24. | Sound effect for the end of each phase | This sound effect is used to notify the player of the end of the phase (whether it be the drawing or guessing phase). Sound effects also improve the professional feel of a game. |
| 25. | Sound effect for results at the end of the game | This sound effect is used to notify the player of the end of the game results. This is where the leader board is displayed. Sound effects also improve the professional feel of a game. |

To determine how successful the development of *Picture This!* was, I can evaluate my success criteria and establish if I met the points I set out to complete. Below I am listing the success criteria number and explaining the point.

1. This criterion was fully met (excluding the additional feature listed). I have a fully functioning menu that allows players to select the action they want to perform.
2. This criterion was fully met. I have a button, that players can select if they want to create a new game, which functions as intended.
3. This criterion was fully met. The round length can be chosen by the player in the ‘creator lobby’ interface.
4. This criterion was fully met. The number of rounds can be chosen by the player in the ‘creator lobby’ interface.
5. This criterion was not met. I decided to only allow a single game at a time so that it was simpler to develop. Multiple games handled by the server can be implemented in the future.
6. This criterion was fully met. The players are taken to certain lobbies depending on what button they selected in the menu.
7. This criterion was fully met. I have a button, that plyers can select if they want to join an existing game, which functions as intended.
8. This criterion was fully met. Each game code entered, after selecting to join a game, is validated, and checked against a list of valid game codes.
9. This criterion was not met. I decided to only allow a single game at a time so that it was simpler to develop. Multiple games handled by the server can be implemented in the future.
10. This criterion was fully met. I created an algorithm that randomly selects words to provide players with depending on the number of players in the game.
11. This criterion was fully met. The brush tool works as intended, with players being able to paint the word easily.
12. This criterion was fully met. Different colours can be selected and drawn in by selecting the different buttons that represent the colours.
13. This criterion was fully met. The thickness of the brush (and eraser) tool can be changed by using the spinner component.
14. This criterion was fully met. By implementing a white colour of the brush tool, players can erase their previous drawings by painting over it with this tool.
15. This criterion was fully met. The player can select the clear button to empty the canvas component, removing all their previous paint.
16. This criterion was partially met. I have developed code to combine three images of the drawings and display them to the players in the guessing phase but have not tested it with three newly created drawings.
17. This criterion was fully met. I have developed a functioning chat box, displaying player’s guesses, and have also created a validation and profanity filter algorithm that works with this.
18. This criterion was fully met. After each round, the players score is added onto their previous score and the file rewritten to in order to store the new score.
19. This criterion was fully met. The scores file is read from, sorted in descending order according to score and displayed to all the players.
20. This criterion was fully met. I created an algorithm that automatically updates the difficulty of the words, before the words are selected, each drawing phase.
21. This criterion was not met. Due to the time constraints that I faced with this project; background music was a feature that did not take priority for development.
22. This criterion was not met. Due to the time constraints that I faced with this project; sound effects were a feature that did not take priority for development.
23. This criterion was not met. Due to the time constraints that I faced with this project; sound effects were a feature that did not take priority for development.
24. This criterion was not met. Due to the time constraints that I faced with this project; sound effects were a feature that did not take priority for development.
25. This criterion was not met. Due to the time constraints that I faced with this project; sound effects were a feature that did not take priority for development.

## Current Issues and Further Development of *Picture This!*

Since this is an incomplete piece of software that requires more updates and alterations to the code, there are obviously errors in the code. These issues can be overlooked when reviewing the project as what it is – a project. However, if this software wanted to reach true success, these issues and bugs would have to be addressed and solved. Below is a list of such issues and potential methods of fixing them that could be used in future development.

1. In terms of inefficient code, I have a large section of code that spans a lot of lines used to instantiate components in the drawing phase interface. In specifics, these are all the colour buttons that all have properties about their visual look and an action listener that defines the action performed when they are selected. This long sequence of similar code is inefficient and increases the load time of the interface since all the components are being instantiated when the interface is loaded. To reduce this inefficiency and create more professional code, it would be better to use a method of some kind that instantiates a button that can be used for a colour, with each colour being specified. This would significantly reduce the amount of lines needed to implement the buttons for the colours.
2. During the validation of the inputs in the creator lobby interface, I use a try catch block to ensure that the inputted values are only of the Integer data type. However, this is inefficient and against best practice when programming since try catch blocks should be used to prevent errors only where it isn’t possible to prevent an exception error. Using it for validation, where the exception is expected to be thrown, is bad practice in the Java programming language since the chances of an exception are not being reduced and an attempt to prevent the error are not being made. Instead of validating the inputs like this, I could spend more development time on creating a series of conditional selection statements that only allow Integer values through. However, this would require more time programming which is unsuitable for me since I am facing a tight time deadline. This could be something for the future development of *Picture This!*.
3. When scaling the images of the drawings to display them to all the players in the guessing phase interface, I am using a single line that scales the images to a more appropriate size. This single line solution to scaling the image files is not the most optimal method since it negatively impacts the quality of the images. There is a method that does not affect the quality of the image files when scaling them to be smaller, but this would require a lot more lines and since the quality of the images are not detrimental to the players ability to guess what they represent, there is minimal need to spend longer developing his solution.
4. Another section of my software that shows the characteristics of inefficient and unprofessional code is where I replace the players scores after each guessing phase. This includes copying over a file but not including the line that I want to replace, writing the new score to the new file and then renaming the new file to the name of the previous file, overwriting the old file. Due to the amount of file querying and handling that this method involves, it will add significant run time to the project, making the performance slower than it could be. I am unsure how I would redevelop this code to produce a more efficient solution but after research I would be able to determine the right method. This is something that I can develop in the future past release of *Picture This!*.
5. Moving onto the first of the issues that are not to do with poor code efficiency, it is important to further develop my **Random Word Selection** algorithm to include the capabilities of preventing players being shown the same word more than once per game. This could be achieved through, once a player is assigned a randomly chosen word, the word being added to a temporary file containing the words that that player has been shown that game. When assigning players words, this temporary file can be checked to ensure a word that has already been given to the player, is not given again. Once the game is over, the file can be cleared and reset. This is a great feature to develop after the release of *Picture This!* and can be implemented through an update to the software.
6. Another issue that should be fixed with further development of my game after release would be the drawing of lines on the canvas and how they can appear jagged and not straight. This makes the drawing mechanics seem less professional and polished. This issue occurs for larger thicknesses of the brush tool since lines are being drawn between the two sets of mouse cursor coordinates. This line can sometimes be drawn on an angle that obtrudes over the line drawn. This makes the line look jagged. This can be altered in further development by drawing smoother shapes to the canvas between the cursor coordinates instead of large lines. However, although this works in theory, during the development of the canvas class for the drawing phase interface, I tested using circles instead of lines and it proved to not work well enough, let alone better than the current implementation, to be used. Nevertheless, with more research and development, this could be a good future development.
7. Eventually, I want *Picture This!* to expand into a larger online game. Although this isn’t an issue at this moment in time, only being able to run one game at a time would become a problem if the game wanted to attract success. It would not be able to handle the different game unless I implemented significant threading into the server. This is a future development that could boost the success and playability of my game.

## Highlighting Usability Features and Improvements

### Menu Interface

This looks at the usability features of the Menu interface.



Clear title in large font is helpful for all players to see.

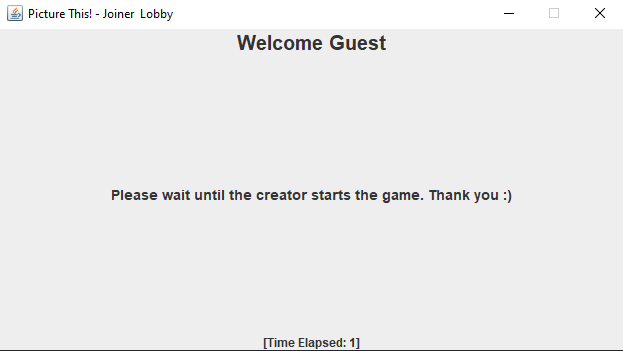
Large buttons with helpful positioning so that players can easily select them.

**Improvements:**

* Larger text on the buttons so they are easier to see.
* More colour makes the interface look nicer.

### Joiner Lobby Interface

This looks at the usability features of the Joiner Lobby interface.



Welcome message makes interface feel more welcoming and personal.

Clear, polite, and informative message informing the player what to do.

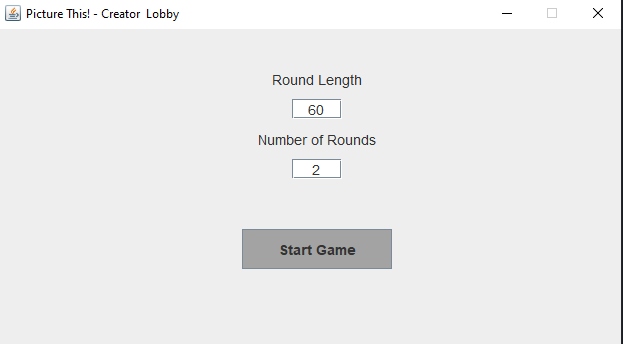
Timer visible but not obstructive of the rest of the interface.

**Improvements:**

* Larger text on all components so it is easier to read.
* More colour makes the interface look nicer.

### Creator Lobby Interface

This looks at the usability features of the Creator Lobby interface.



Central components allow for easier identification for the players.

Large button with large bold text provides easy identification to the players.

**Improvements:**

* Larger components so it is easier to read and less empty space.
* More colour makes the interface look nicer.

### Drawing Phase Interface

This looks at the usability features of the Drawing Phase interface.



Clear timer which is well labelled and in a good position for the players to see.

Largely displayed word to draw is obvious to the player.

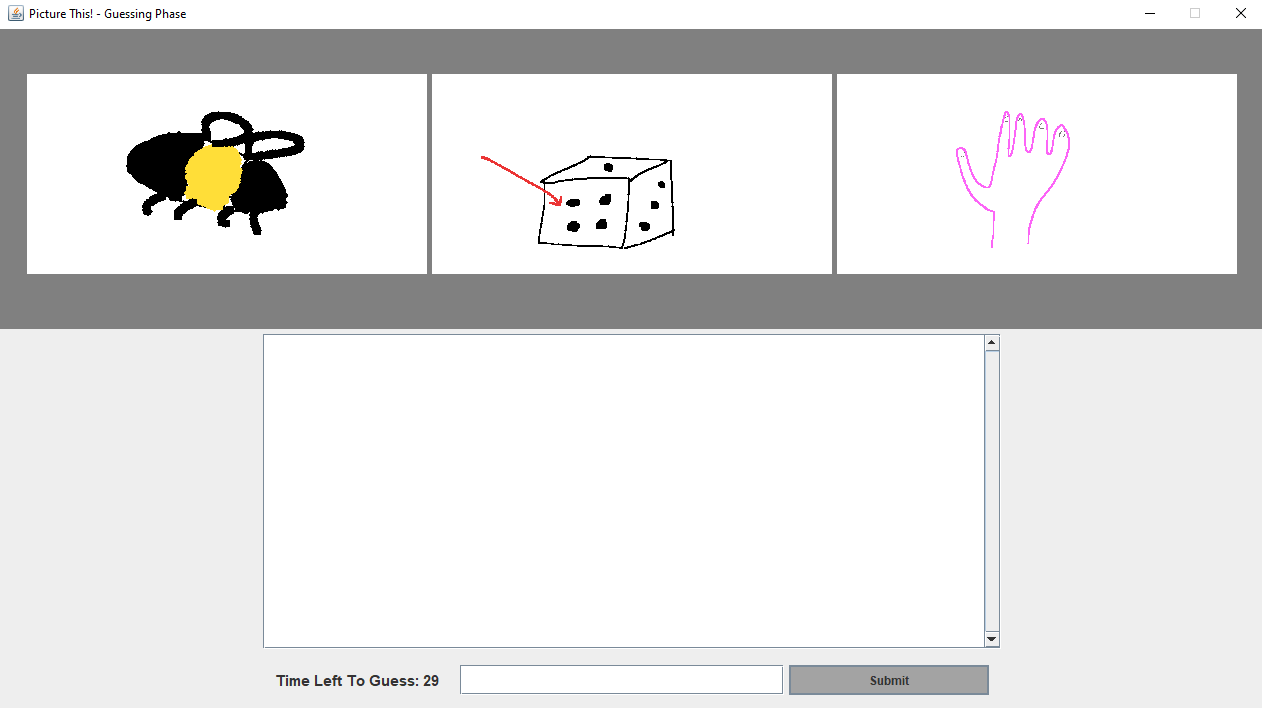
Buttons using colours to represent what they do instead of text, making it more intuitive to the players.

**Improvements:**

* Smoother lines when drawing would make the game feel more professional.
* A larger variety of colours would increase freedom when drawing.
* More visible thickness spinner.

### Guessing Phase Interface

This looks at the usability features of the Guessing Phase interface.



JayB: hello

Callum: test

Tommy: hey \*\*\*\*\*\*!

Guest guessed correctly!

Clearly displayed drawings visible to the player.

Large font size for all guesses that are shown in the chat box are clear to the player.

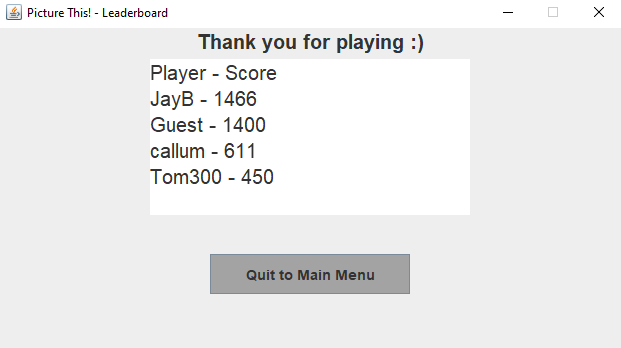
Input components in an intuitive position.

**Improvements:**

* Large input components so the user can clearly identify how to submit guesses.
* More colour makes the interface look nicer.

### Leader Board Interface

This looks at the usability features of the Leader Board interface.



Large font size for thank you message is clear for the players to see and makes them feel valued.

Clear display of scores with a large font in the centre of the screen is clear for the players.

Large button with large bold text is clear for the players.

**Improvements:**

* More colour makes the interface look nicer.
* Better layout of the leader board so it is easier to read (e.g. columns).

## Final Client Comment

In the final meeting I had with my client, Callum Macmahon said the following last words that we both agreed on:

“To see the development of *Picture This!* has been great and it really shows the journey that was taken to create such a great piece of software. This program works fully on its own in its current state, however it also provides future development opportunities that is exciting. I am looking forward to seeing the progression made in *Picture This!* in the future and am pleased to have been a part of the development of a well-rounded piece of software.”

# Appendices

### Interview Files

These are the files for the interviews that I held with my client during the analysis section earlier in the documentation of the project.

**INTERVEIW TO DISCOVER CLIENT REQUIREMENTS** DATE: 01/11/21

What does the client consider makes a good game?

* Does the easy usability of the GUI make a game better?
* Yes, I think that a good UI is important when making a game because it allows for easier navigation for the user.
* Are music/sound effects important to a game?
* I think they are because it can make the game more immersive.
* Does an aspect of competition against others make a game more fun?
* Definitely, I enjoy games where I play against others.
* Do you think that repetitiveness within a game makes it boring?
* Depending on the game, repeating the same actions can’t be fun without a reward.

How hard can a game be, whilst still maintaining an enjoyable environment?

* Does a higher difficulty of games make you want to not play them?
* Difficulty isn’t the most important factor for the game to be fun. I think that games can be enjoyable no matter the difficulty.
* Do you prefer games that are more relaxing rather than challenging?
* Depends on how I am feeling and who I am playing with. I enjoy both.
* Are easier games hard to stay engaged with?
* Yes, however, if I am playing with friends then it is enjoyable.
* Should games have a development of difficulty (e.g. become harder the better you do?
* I think so because otherwise games can become boring and repetitive.

Are online games with friends considered more enjoyable than single-player games?

* What do you prefer more, online games or single-player games?
* Online games, it is more interactive and enjoyable
* Do you regularly play and enjoy online games?
* Yes, I play them every day and love them
* Do you play these games alone or with others (either friends or strangers you meet)?
* Both, I like to discover new players and also enjoy playing with my friends
* Is communication required for an online game with others to be fun?
* Yes, part of the fun is the communication with others

**INTERVEIW TO DISCOVER REQUIRED FEATURES** DATE: 05/11/21

Game Mechanics:

* Should players have a limited number of guesses?
* I think, by default, players should not have a limit on the number of guesses that they can make. However, I feel that it would be cool if players could choose at the start of the game if a limit is applied.
* Will players be able to change the round length?
* Yes, this should be chosen before the game starts.
* Will players be able to choose the number of rounds?
* Yes, this should be chosen before the game starts.

Visuals:

* For the drawing and guessing phase, should separate GUIs be used?
* I feel that this will help separate the separate phases of the game.
* Should the GUIs be a minimalist design, or more complex?
* I prefer a minimalist design and want the GUI to reflect this.

Sounds:

* Is background music important for the client?
* It is not important, but I still feel that it would be good for the game and should be a jovial tune.
* Should there be additional sound effects (e.g., when guessing correctly)?
* Yes, I want this as it will make the game feel more polished.
* Will players be able to choose their own music?
* This is a feature that I feel is something that can be added after the main game is complete.

Online Capabilities:

* Should both private and public games be available to the player?
* Yes, but public games should not be a priority. I would prefer private games initially and public games can be potentially added later.
* How should private games be accessible to players (e.g., through an email link)?
* I think players should enter a code of the game to enter.
* Does the client want the possibility of friends in the game?
* This should be a feature that is added after the initial requirements are met.

Scoring System:

* How should points be awarded to players?
* Players should be awarded points based on the time it takes for them to guess what the images represent.
* Will players be able to vote for their favourite drawing?
* I believe that this is a feature that distracts players from the purpose of the game. I want players to focus more on guessing the correct word, than voting for the best drawing.
* Should the leader board be shown constantly, or only after each round?
* After each round, I want the leaderboard to be shown. This is something that is only at the end of each round, and at the end of the game.

Difficulty:

* Will the difficulty of the game change, depending on all the player’s performances?
* Yes, this is a good idea from the developer. This will keep the game exciting and prevent a repetitive feeling.
* Should the difficulty change depending on the words or limits on time etc.?
* Since I want time limits to be decided by the player, I think that the difficulty should be changed by choosing harder words.

### Code Listings

Below are the whole code listings for *Picture This!*

#### Menu.java

import javax.swing.JFrame;

import javax.swing.JButton;

import javax.swing.JComponent;

import javax.swing.JLabel;

import javax.swing.JOptionPane;

import java.awt.Container;

import java.awt.Insets;

import java.awt.Font;

import java.awt.Color;

import java.io.BufferedReader;

import java.io.FileReader;

import java.io.FileNotFoundException;

import java.io.IOException;

public class Menu {

    private JFrame menuFrame = new JFrame("Picture This! - Main Menu");

    private JButton createGameBtn, joinGameBtn, quitGameBtn;

    private JLabel titleLbl;

    static String username;

    static int numPlayers = 3; //set as three for example

    public static void main(String[] args) {

        new Menu().show();

    }

    public void show() {

        Container content = menuFrame.getContentPane();

        content.setLayout(null);

        Insets insets = content.getInsets();

        //TITLE LABEL

        titleLbl = new JLabel("Picture This!", JLabel.CENTER);

        titleLbl.setSize(700, 200);

        titleLbl.setFont(titleLbl.getFont().deriveFont(Font.PLAIN, 110f));

        content.add(titleLbl);

        titleLbl.setBounds(

            fromLeft(titleLbl) + insets.left,

            70 + insets.top,

            titleLbl.getWidth(),

            titleLbl.getHeight()

        );

        //CREATE GAME BUTTON

        createGameBtn = new JButton("Create New Game");

        createGameBtn.setSize(400, 50);

        createGameBtn.setBackground(Color.decode("#a3a3a3"));

        createGameBtn.addActionListener(e -> {

            createGame();

        });

        content.add(createGameBtn);

        createGameBtn.setBounds(

            fromLeft(createGameBtn) + insets.left,

            300 + insets.top,

            createGameBtn.getWidth(),

            createGameBtn.getHeight()

        );

        //JOIN GAME BUTTON

        joinGameBtn = new JButton("Join Existing Game");

        joinGameBtn.setSize(400, 50);

        joinGameBtn.setBackground(Color.decode("#a3a3a3"));

        joinGameBtn.addActionListener(e -> {

            joinGame();

        });

        content.add(joinGameBtn);

        joinGameBtn.setBounds(

            fromLeft(joinGameBtn) + insets.left,

            370 + insets.top, joinGameBtn.getWidth(),

            joinGameBtn.getHeight()

        );

        //QUIT GAME BUTTON

        quitGameBtn = new JButton("Exit To Desktop");

        quitGameBtn.setSize(400, 50);

        quitGameBtn.setBackground(Color.decode("#a3a3a3"));

        quitGameBtn.addActionListener(e -> {

            menuFrame.dispose();

        });

        content.add(quitGameBtn);

        quitGameBtn.setBounds(

            fromLeft(quitGameBtn) + insets.left,

            440 + insets.top,

            quitGameBtn.getWidth(),

            quitGameBtn.getHeight()

        );

        menuFrame.setSize(1280, 720);

        menuFrame.setDefaultCloseOperation(JFrame.EXIT\_ON\_CLOSE);

        menuFrame.setVisible(true);

        menuFrame.setResizable(false);

        menuFrame.setLocationRelativeTo(null);

    }

    public void createGame() {

        setUsername();

        CreatorLobby cl = new CreatorLobby();

        cl.show();

        menuFrame.dispose();

    }

    public void joinGame() {

        String gameCode = (String) JOptionPane.showInputDialog(

            menuFrame,

            "Enter a valid game code to join a game:",

            "Enter A Game Code",

            JOptionPane.PLAIN\_MESSAGE,

            null,

            null,

            ""

        );

        if (gameCode == null) {

            //CANCEL selected

        } else {

            if (gameCode.length() == 5 && gameCode.matches("^[A-Z0-9]\*$")) {

                Boolean found = false;

                try {

                    FileReader freader = new FileReader("codes.txt");

                    BufferedReader breader = new BufferedReader(freader);

                    String line = null;

                    while ((line = breader.readLine()) != null) {

                        if (line.trim().equals(gameCode)) {

                            found = true;

                        }

                    }

                    breader.close();

                } catch (FileNotFoundException e) {

                    e.printStackTrace();

                } catch (IOException e) {

                    e.printStackTrace();

                }

                if (found == true) {

                    setUsername();

                    JoinerLobby jl = new JoinerLobby();

                    jl.show();

                    menuFrame.dispose();

                } else {

                    JOptionPane.showMessageDialog(

                        menuFrame,

                        "The game code entered does not match an existing game code.\nPlease enter a game code of a current game.",

                        "ERROR: Game code does not exist",

                        JOptionPane.ERROR\_MESSAGE

                    );

                }

            } else {

                JOptionPane.showMessageDialog(

                    menuFrame,

                    "The game code entered is not in the correct format.\nPlease enter a game code of a valid format.",

                    "ERROR: Invalid game code",

                    JOptionPane.ERROR\_MESSAGE

                );

            }

        }

    }

    public Integer fromLeft(JComponent component) {

        //Determines where the component should be placed horizontally so that it appears in the middle

        Integer fromLeft = 640 - component.getWidth() / 2;

        return fromLeft;

    }

    public void setUsername() {

        username = (String) JOptionPane.showInputDialog(

            menuFrame,

            "Enter a username that you would like to go by:",

            "Enter Username",

            JOptionPane.PLAIN\_MESSAGE,

            null,

            null,

            "Guest"

        );

    }

}

#### JoinerLobby.java

import javax.swing.JFrame;

import javax.swing.JLabel;

import javax.swing.Timer;

import java.awt.Container;

import java.awt.BorderLayout;

public class JoinerLobby {

    private JFrame joinerLobbyFrame = new JFrame("Picture This! - Joiner  Lobby");

    private JLabel welcomeLbl, msgLbl, timeLbl;

    private Timer timer;

    private int counter = -1;

    public void show() {

        Container content = joinerLobbyFrame.getContentPane();

        content.setLayout(new BorderLayout());

        welcomeLbl = new JLabel(String.format("Welcome %s", Menu.username), JLabel.CENTER);

        welcomeLbl.setFont(welcomeLbl.getFont().deriveFont(20.0f));

        content.add(welcomeLbl, BorderLayout.NORTH);

        msgLbl = new JLabel("Please wait until the creator starts the game. Thank you :)", JLabel.CENTER);

        msgLbl.setFont(msgLbl.getFont().deriveFont(14.0f));

        content.add(msgLbl, BorderLayout.CENTER);

        timeLbl = new JLabel("", JLabel.CENTER);

        content.add(timeLbl, BorderLayout.SOUTH);

        joinerLobbyFrame.setSize(640, 360);

        joinerLobbyFrame.setDefaultCloseOperation(JFrame.EXIT\_ON\_CLOSE);

        joinerLobbyFrame.setVisible(true);

        joinerLobbyFrame.setResizable(false);

        joinerLobbyFrame.setLocationRelativeTo(null);

        elapsedTime();

    }

    public void elapsedTime() {

        timer = new Timer(1000, e -> {

            counter++;

            String output = String.format("[Time Elapsed: %S]", String.valueOf(counter));

            timeLbl.setText(output);

        });

        timer.setInitialDelay(0);

        timer.start();

    }

    public void startGame() {

        DrawingPhase dp = new DrawingPhase();

        dp.show();

        joinerLobbyFrame.dispose();

    }

    //creator selects start game, sends game-start msg to server

    //server sends game-start msg to all players

    //joiner lobby socket listens for msg and starts drawing phase once recieved

}

#### CreatorLobby.java

import javax.swing.JButton;

import javax.swing.JComponent;

import javax.swing.JFrame;

import javax.swing.JLabel;

import javax.swing.JOptionPane;

import javax.swing.JTextField;

import java.awt.Container;

import java.awt.Dimension;

import java.awt.Insets;

import java.awt.Font;

import java.awt.Color;

public class CreatorLobby {

    private JFrame creatorLobbyFrame = new JFrame("Picture This! - Creator  Lobby");

    private JButton startGame;

    private JLabel roundLenLbl, roundNumLbl;

    private JTextField roundLenTxt, roundNumTxt;

    static Integer roundLen, roundNum;

    public void show() {

        Container content = creatorLobbyFrame.getContentPane();

        content.setLayout(null);

        Insets insets = content.getInsets();

        roundLenLbl = new JLabel("Round Length", JLabel.CENTER);

        roundLenLbl.setSize(200, 20);

        roundLenLbl.setFont(roundLenLbl.getFont().deriveFont(Font.PLAIN, 14.0f));

        content.add(roundLenLbl);

        roundLenLbl.setBounds(

            fromLeft(roundLenLbl) + insets.left,

            40 + insets.top,

            roundLenLbl.getWidth(),

            roundLenLbl.getHeight()

        );

        roundLenTxt = new JTextField("60", 5);

        roundLenTxt.setSize(50, 20);

        roundLenTxt.setHorizontalAlignment(JTextField.CENTER);

        roundLenTxt.setFont(roundLenTxt.getFont().deriveFont(Font.PLAIN, 14.0f));

        content.add(roundLenTxt);

        roundLenTxt.setBounds(

            fromLeft(roundLenTxt) + insets.left,

            70 + insets.top,

            roundLenTxt.getWidth(),

            roundLenTxt.getHeight()

        );

        roundNumLbl = new JLabel("Number of Rounds", JLabel.CENTER);

        roundNumLbl.setSize(200, 20);

        roundNumLbl.setFont(roundNumLbl.getFont().deriveFont(Font.PLAIN, 14.0f));

        content.add(roundNumLbl);

        roundNumLbl.setBounds(

            fromLeft(roundNumLbl) + insets.left,

            100 + insets.top,

            roundNumLbl.getWidth(),

            roundNumLbl.getHeight()

        );

        roundNumTxt = new JTextField("2", 5);

        roundNumTxt.setSize(50, 20);

        roundNumTxt.setHorizontalAlignment(JTextField.CENTER);

        roundNumTxt.setFont(roundNumTxt.getFont().deriveFont(Font.PLAIN, 14.0f));

        content.add(roundNumTxt);

        roundNumTxt.setBounds(

            fromLeft(roundNumTxt) + insets.left,

            130 + insets.top,

            roundNumTxt.getWidth(),

            roundNumTxt.getHeight()

        );

        startGame = new JButton("Start Game");

        startGame.setSize(new Dimension(150, 40));

        startGame.setBackground(Color.decode("#a3a3a3"));

        startGame.setFont(startGame.getFont().deriveFont(Font.BOLD, 14.0f));

        startGame.addActionListener(e -> {

            try {

                Integer.parseInt(roundLenTxt.getText());

                Integer.parseInt(roundNumTxt.getText());

                if (Integer.parseInt(roundLenTxt.getText()) >= 30 && Integer.parseInt(roundLenTxt.getText()) <= 360) {

                    if (Integer.parseInt(roundNumTxt.getText()) > 0 && Integer.parseInt(roundNumTxt.getText()) < 11) {

                        roundLen = Integer.parseInt(roundLenTxt.getText());

                        roundNum = Integer.parseInt(roundNumTxt.getText());

                    } else {

                        JOptionPane.showMessageDialog(

                            creatorLobbyFrame,

                            "The value entered for the number of rounds is not within the valid range.\nPlease enter an integer in the range 1 - 10.",

                            "ERROR: Invalid Range",

                            JOptionPane.ERROR\_MESSAGE

                        );

                    }

                } else {

                    JOptionPane.showMessageDialog(

                        creatorLobbyFrame,

                        "The value entered for the length of rounds is not within the valid range.\nPlease enter an integer in the range 30 - 360.",

                        "ERROR: Invalid Range",

                        JOptionPane.ERROR\_MESSAGE

                    );

                }

            } catch (Exception excep) {

                JOptionPane.showMessageDialog(

                    creatorLobbyFrame,

                    "The input(s) given are not in the correct format.\nPlease only enter integers.",

                    "ERROR: Invalid Input",

                    JOptionPane.ERROR\_MESSAGE

                );

            }

            DrawingPhase dp = new DrawingPhase();

            dp.show();

            creatorLobbyFrame.dispose();

            //players in joiner lobby sent to drawing phase

        });

        content.add(startGame);

        startGame.setBounds(

            fromLeft(startGame) + insets.left,

            200 + insets.top,

            startGame.getWidth(),

            startGame.getHeight()

        );

        creatorLobbyFrame.setSize(640, 360);

        creatorLobbyFrame.setDefaultCloseOperation(JFrame.EXIT\_ON\_CLOSE);

        creatorLobbyFrame.setVisible(true);

        creatorLobbyFrame.setResizable(false);

        creatorLobbyFrame.setLocationRelativeTo(null);

    }

    public Integer fromLeft(JComponent component) {

        //length to middle

        Integer fromLeft = 320 - component.getWidth() / 2;

        return fromLeft;

    }

}

#### DrawingPhase.java

import javax.swing.\*;

import java.awt.\*;

import java.awt.image.BufferedImage;

import javax.imageio.ImageIO;

import java.io.BufferedReader;

import java.io.File;

import java.io.FileReader;

import java.io.IOException;

import java.util.ArrayList;

import java.util.List;

import java.util.concurrent.ThreadLocalRandom;

public class DrawingPhase {

    private JButton clearBtn, blackBtn, greyBtn, brownBtn, redBtn, orangeBtn, yellowBtn, greenBtn, lightBlueBtn, darkBlueBtn, pinkBtn, purpleBtn, eraserBtn;

    private JLabel wordLbl, timerLbl;

    private Canvas canvas;

    private JFrame drawingFrame = new JFrame("Picture This! - Drawing Phase");

    static String finalWord;

    static String difficulty = "m";

    static JSpinner thicknessSpin;

    private Timer timer;

    int counter = CreatorLobby.roundLen / 2;

    public void show() {

        Container content = drawingFrame.getContentPane();

        content.setLayout(new BorderLayout());

        canvas = new Canvas();

        content.add(canvas, BorderLayout.CENTER);

        JComponent tools = tools();

        content.add(tools, BorderLayout.LINE\_START);

        String[] words = wordSelection("3player.csv");

        finalWord = words[words.length-1].trim();

        //output other words to the other clients

        wordLbl = new JLabel(words[0], JLabel.CENTER);

        wordLbl.setFont(wordLbl.getFont().deriveFont(35.0f));

        content.add(wordLbl, BorderLayout.SOUTH);

        timerLbl = new JLabel("", JLabel.CENTER);

        timerLbl.setFont(timerLbl.getFont().deriveFont(20.0f));

        content.add(timerLbl, BorderLayout.NORTH);

        drawingFrame.setSize(1280, 720);

        drawingFrame.setDefaultCloseOperation(JFrame.EXIT\_ON\_CLOSE);

        drawingFrame.setVisible(true);

        drawingFrame.setResizable(false);

        drawingFrame.setLocationRelativeTo(null);

        countDown();

    }

    private String[] wordSelection(String file) {

        dgdb(); //sets difficulty

        try {

            BufferedReader br = new BufferedReader(new FileReader("3player.csv"));

            String str;

            List<String[]> lines = new ArrayList<String[]>();

            while ((str = br.readLine()) != null) {

                String[] words = str.split(",");

                if (words[0].trim().equals(difficulty)) {

                    lines.add(words);

                }

            }

            br.close();

            String[][] array = new String[lines.size()][0];

            lines.toArray(array);

            int randomNum = ThreadLocalRandom.current().nextInt(0, array.length); //min and (max + 1)

            int i = 0;

            for (String[] x : array) {

                if (i == randomNum) {

                    String word1 = array[i][1];

                    String word2 = array[i][2];

                    String finalWord = array[i][array.length - 1];

                    if (Menu.numPlayers == 2) {

                        String[] returns = new String[3];

                        returns[0] = word1;

                        returns[1] = word2;

                        returns[2] = finalWord;

                        return returns;

                    } else if (Menu.numPlayers == 3) {

                        String word3 = array[i][3];

                        String[] returns = new String[4];

                        returns[0] = word1;

                        returns[1] = word2;

                        returns[2] = word3;

                        returns[3] = finalWord;

                        return returns;

                    } else if (Menu.numPlayers == 4) {

                        String word3 = array[i][3];

                        String word4 = array[i][4];

                        String[] returns = new String[5];

                        returns[0] = word1;

                        returns[1] = word2;

                        returns[2] = word3;

                        returns[3] = word4;

                        returns[4] = finalWord;

                        return returns;

                    }

                } else {

                    i += 1;

                }

            }

        } catch (Exception e) {

            e.printStackTrace();

        }

        return null;

    }

    private JComponent tools() {

        JPanel tools = new JPanel();

        tools.setPreferredSize(new Dimension(75, 0)); //Height value is negligable since since the panel will take the max height it can in the interface

        GridLayout gridLay = new GridLayout(0, 1);

        gridLay.setVgap(10);

        tools.setLayout(gridLay);

        clearBtn = new JButton("Clear");

        clearBtn.setBackground(Color.decode("#a3a3a3"));

        clearBtn.setBorder(BorderFactory.createLineBorder(Color.decode("#000")));

        clearBtn.setAlignmentX(Component.CENTER\_ALIGNMENT);

        clearBtn.addActionListener(e -> {

            canvas.clear();

        });

        blackBtn = new JButton("");

        blackBtn.setBackground(Color.decode("#000"));

        blackBtn.setBorder(BorderFactory.createLineBorder(Color.decode("#000")));

        blackBtn.setAlignmentX(Component.CENTER\_ALIGNMENT);

        blackBtn.addActionListener(e -> {

            canvas.black();

        });

        greyBtn = new JButton("");

        greyBtn.setBackground(Color.decode("#757575"));

        greyBtn.setBorder(BorderFactory.createLineBorder(Color.decode("#000")));

        greyBtn.setAlignmentX(Component.CENTER\_ALIGNMENT);

        greyBtn.addActionListener(e -> {

            canvas.grey();

        });

        brownBtn = new JButton("");

        brownBtn.setBackground(Color.decode("#8c4420"));

        brownBtn.setBorder(BorderFactory.createLineBorder(Color.decode("#000")));

        brownBtn.setAlignmentX(Component.CENTER\_ALIGNMENT);

        brownBtn.addActionListener(e -> {

            canvas.brown();

        });

        redBtn = new JButton("");

        redBtn.setBackground(Color.decode("#eb3434"));

        redBtn.setBorder(BorderFactory.createLineBorder(Color.decode("#000")));

        redBtn.setAlignmentX(Component.CENTER\_ALIGNMENT);

        redBtn.addActionListener(e -> {

            canvas.red();

        });

        orangeBtn = new JButton("");

        orangeBtn.setBackground(Color.decode("#ff7912"));

        orangeBtn.setBorder(BorderFactory.createLineBorder(Color.decode("#000")));

        orangeBtn.setAlignmentX(Component.CENTER\_ALIGNMENT);

        orangeBtn.addActionListener(e -> {

            canvas.orange();

        });

        yellowBtn = new JButton("");

        yellowBtn.setBackground(Color.decode("#ffde38"));

        yellowBtn.setBorder(BorderFactory.createLineBorder(Color.decode("#000")));

        yellowBtn.setAlignmentX(Component.CENTER\_ALIGNMENT);

        yellowBtn.addActionListener(e -> {

            canvas.yellow();

        });

        greenBtn = new JButton("");

        greenBtn.setBackground(Color.decode("#17e658"));

        greenBtn.setBorder(BorderFactory.createLineBorder(Color.decode("#000")));

        greenBtn.setAlignmentX(Component.CENTER\_ALIGNMENT);

        greenBtn.addActionListener(e -> {

            canvas.green();

        });

        lightBlueBtn = new JButton("");

        lightBlueBtn.setBackground(Color.decode("#0fcfff"));

        lightBlueBtn.setBorder(BorderFactory.createLineBorder(Color.decode("#000")));

        lightBlueBtn.setAlignmentX(Component.CENTER\_ALIGNMENT);

        lightBlueBtn.addActionListener(e -> {

            canvas.lightBlue();

        });

        darkBlueBtn = new JButton("");

        darkBlueBtn.setBackground(Color.decode("#1f1bf7"));

        darkBlueBtn.setBorder(BorderFactory.createLineBorder(Color.decode("#000")));

        darkBlueBtn.setAlignmentX(Component.CENTER\_ALIGNMENT);

        darkBlueBtn.addActionListener(e -> {

            canvas.darkBlue();

        });

        pinkBtn = new JButton("");

        pinkBtn.setBackground(Color.decode("#fc65f7"));

        pinkBtn.setBorder(BorderFactory.createLineBorder(Color.decode("#000")));

        pinkBtn.setAlignmentX(Component.CENTER\_ALIGNMENT);

        pinkBtn.addActionListener(e -> {

            canvas.pink();

        });

        purpleBtn = new JButton("");

        purpleBtn.setBackground(Color.decode("#b005ff"));

        purpleBtn.setBorder(BorderFactory.createLineBorder(Color.decode("#000")));

        purpleBtn.setAlignmentX(Component.CENTER\_ALIGNMENT);

        purpleBtn.addActionListener(e -> {

            canvas.purple();

        });

        eraserBtn = new JButton("Eraser");

        eraserBtn.setBackground(Color.decode("#FFFFFF"));

        eraserBtn.setBorder(BorderFactory.createLineBorder(Color.decode("#000")));

        eraserBtn.setAlignmentX(Component.CENTER\_ALIGNMENT);

        eraserBtn.addActionListener(e -> {

            canvas.erase();

        });

        SpinnerModel model = new SpinnerNumberModel(5, 1, 35, 2);

        thicknessSpin = new JSpinner(model);

        thicknessSpin.setAlignmentX(Component.CENTER\_ALIGNMENT);

        JFormattedTextField textField = ((JSpinner.DefaultEditor) thicknessSpin.getEditor()).getTextField();

        textField.setEditable(false);

        textField.setHorizontalAlignment(JTextField.CENTER);

        textField.setFont(textField.getFont().deriveFont(20.0f));

        tools.add(thicknessSpin);

        tools.add(blackBtn);

        tools.add(greyBtn);

        tools.add(brownBtn);

        tools.add(redBtn);

        tools.add(orangeBtn);

        tools.add(yellowBtn);

        tools.add(greenBtn);

        tools.add(lightBlueBtn);

        tools.add(darkBlueBtn);

        tools.add(pinkBtn);

        tools.add(purpleBtn);

        tools.add(eraserBtn);

        tools.add(clearBtn);

        return tools;

    }

    private void countDown() {

        timer = new Timer(1000, e -> {

            if (counter > 0) {

                counter--;

                String output = String.format("Time Left To Draw: %s", counter);

                timerLbl.setText(output);

            } else {

                timer.stop();

                saveCanvasImage();

                GuessingPhase gp = new GuessingPhase();

                gp.show();

                drawingFrame.dispose();

                //send msg to server to take imgs and order

            }

        });

        timer.setInitialDelay(0);

        timer.start();

    }

    public void saveCanvasImage() {

        BufferedImage image = new BufferedImage(canvas.getWidth(), canvas.getHeight(), BufferedImage.TYPE\_INT\_RGB);

        canvas.paint(image.getGraphics());

        try {

            ImageIO.write(image, "png", new File("word1.png"));

        } catch (IOException excep) {

            System.out.println(excep);

        }

    }

    public void dgdb() {

        try {

            BufferedReader br = new BufferedReader(new FileReader("scores.txt"));

            String str;

            int sum = 0;

            while ((str = br.readLine()) != null) {

                String[] words = str.split(" ");

                sum += Integer.parseInt(words[2]);

            }

            double avrg = sum / Menu.numPlayers;

            double percTimeTaken = 1 - avrg / 1000;

            if (percTimeTaken >= 0 && percTimeTaken <= 0.3) {

                System.out.println("[difficulty increased]");

                if (difficulty.equals("e")) {

                    difficulty = "m";

                } else if (difficulty.equals("m")) {

                    difficulty = "h";

                }

            } else if (percTimeTaken > 0.3 && percTimeTaken <= 1) {

                System.out.println("[difficulty unchanged]");

            } else if (percTimeTaken > 0.7 && percTimeTaken <= 1) {

                System.out.println("[difficulty decreased]");

                if (difficulty.equals("m")) {

                    difficulty = "e";

                } else if (difficulty.equals("h")) {

                    difficulty = "m";

                }

            }

            br.close();

        } catch (Exception e) {

            e.printStackTrace();

        }

    }

}

#### Canvas.java

import javax.swing.JComponent;

import java.awt.Image;

import java.awt.Graphics2D;

import java.awt.Graphics;

import java.awt.Color;

import java.awt.BasicStroke;

import java.awt.event.MouseAdapter;

import java.awt.event.MouseMotionAdapter;

import java.awt.event.MouseEvent;

public class Canvas extends JComponent {

    private Image image;

    private Graphics2D graphics2d;

    private int oldx, currentx, oldy, currenty;

    public Canvas() {

        addMouseListener(new MouseAdapter() {

            public void mousePressed(MouseEvent e) {

                oldx = e.getX();

                oldy = e.getY();

            }

        });

        addMouseMotionListener(new MouseMotionAdapter() {

            public void mouseDragged(MouseEvent e) {

                currentx = e.getX();

                currenty = e.getY();

                try {

                    DrawingPhase.thicknessSpin.commitEdit();

                } catch (java.text.ParseException excep) {

                    //error should not be reached due to disabling of keybaord input on JSpinner

                    System.out.println(excep);

                }

                int value = (Integer) DrawingPhase.thicknessSpin.getValue();

                setThickness(value);

                if (graphics2d != null) {

                    graphics2d.drawLine(oldx, oldy, currentx, currenty);

                    repaint();

                    oldx = currentx;

                    oldy = currenty;

                }

            }

        });

    }

    protected void paintComponent(Graphics g) {

        super.paintComponent(g);

        if (image == null) {

            image = createImage(getWidth(), getHeight());

            graphics2d = (Graphics2D) image.getGraphics();

            clear();

        }

        g.drawImage(image, 0, 0, null);

    }

    public void clear() {

        Color oldColour = graphics2d.getColor();

        graphics2d.setPaint(Color.decode("#FFFFFF"));

        graphics2d.fillRect(0, 0, getWidth(), getHeight());

        graphics2d.setPaint(oldColour);

        repaint();

    }

    public void erase() {

        graphics2d.setPaint(Color.decode("#FFFFFF"));

    }

    public void black() {

        graphics2d.setPaint(Color.decode("#000"));

    }

    public void grey() {

        graphics2d.setPaint(Color.decode("#757575"));

    }

    public void brown() {

        graphics2d.setPaint(Color.decode("#8c4420"));

    }

    public void red() {

        graphics2d.setPaint(Color.decode("#eb3434"));

    }

    public void orange() {

        graphics2d.setPaint(Color.decode("#ff7912"));

    }

    public void yellow() {

        graphics2d.setPaint(Color.decode("#ffde38"));

    }

    public void green() {

        graphics2d.setPaint(Color.decode("#17e658"));

    }

    public void lightBlue() {

        graphics2d.setPaint(Color.decode("#0fcfff"));

    }

    public void darkBlue() {

        graphics2d.setPaint(Color.decode("#1f1bf7"));

    }

    public void pink() {

        graphics2d.setPaint(Color.decode("#fc65f7"));

    }

    public void purple() {

        graphics2d.setPaint(Color.decode("#b005ff"));

    }

    public void setThickness(Integer thickness) {

        graphics2d.setStroke(new BasicStroke(thickness));

    }

}

#### GuessPhase.java

import javax.swing.\*;

import java.awt.\*;

import java.io.BufferedReader;

import java.io.BufferedWriter;

import java.io.File;

import java.io.FileReader;

import java.io.FileWriter;

import java.util.ArrayList;

import java.util.List;

import javax.swing.border.EmptyBorder;

public class GuessingPhase {

    private JFrame guessingFrame = new JFrame("Picture This! - Guessing Phase");

    private JButton submitBtn;

    private JTextField inputTxt;

    private JTextArea chatArea;

    private JScrollPane scroll;

    private JLabel word1Img, word2Img, word3Img, timerLbl;

    static int timeTaken = CreatorLobby.roundLen / 2; //max time taken, assumes players do not guess

    static int points = 0;

    private Timer timer;

    private int counter = CreatorLobby.roundLen / 2;

    public void show() {

        Container content = guessingFrame.getContentPane();

        content.setLayout(new BorderLayout());

        JComponent inputs = inputs();

        content.add(inputs, BorderLayout.SOUTH);

        JComponent display = display();

        content.add(display, BorderLayout.CENTER);

        JComponent pics = pics();

        content.add(pics, BorderLayout.NORTH);

        guessingFrame.setSize(1280, 720);

        guessingFrame.setDefaultCloseOperation(JFrame.EXIT\_ON\_CLOSE);

        guessingFrame.setVisible(true);

        guessingFrame.setResizable(false);

        guessingFrame.setLocationRelativeTo(null);

        guessingFrame.getRootPane().setDefaultButton(submitBtn); //focuses button allowing enter to be used to submit guesses

        countDown();

    }

    private JComponent inputs() {

        JPanel inputs = new JPanel();

        inputs.setLayout(new FlowLayout());

        inputs.setPreferredSize(new Dimension(0, 50));

        timerLbl = new JLabel("", JLabel.CENTER);

        timerLbl.setFont(timerLbl.getFont().deriveFont(15.0f));

        inputs.add(timerLbl);

        inputTxt = new JTextField(20);

        inputTxt.setFont(inputTxt.getFont().deriveFont(20.0f));

        inputTxt.setHorizontalAlignment(JTextField.CENTER);

        inputs.add(inputTxt);

        submitBtn = new JButton("Submit");

        submitBtn.setBackground(Color.decode("#a3a3a3"));

        submitBtn.addActionListener(e -> {

            String input = inputTxt.getText();

            String output;

            if (input.equals("")) {

                JOptionPane.showMessageDialog(

                    guessingFrame,

                    "The guess entered was blank.\nPlease enter a guess that contains at least one letter character.",

                    "ERROR: Guess given was null",

                    JOptionPane.ERROR\_MESSAGE

                );

            } else {

                if (containsNumbers(input) == true) {

                    JOptionPane.showMessageDialog(

                        guessingFrame,

                        "The guess entered contains invalid characters.\nPlease enter a guess that does not contain numbers.",

                        "ERROR: Invalid characters in inputted guess",

                        JOptionPane.ERROR\_MESSAGE

                    );

                } else {

                    if (input.toLowerCase().equals(DrawingPhase.finalWord)) {

                        output = Menu.username + " guessed correctly!\n";

                        //output sent to server to be displayed to all other players

                        int timeTaken = (CreatorLobby.roundLen / 2) - counter;

                        double percTimeTaken = ((double) timeTaken) / (CreatorLobby.roundLen / 2);

                        double percTimeLeft = 1 - percTimeTaken;

                        //deletes current line of old score

                        File inputFile = new File("scores.txt");

                        File tempFile = new File("tempScores.txt");

                        try {

                            BufferedReader reader = new BufferedReader(new FileReader(inputFile));

                            BufferedWriter writer = new BufferedWriter(new FileWriter(tempFile));

                            String lineToRemove = Menu.username + " - " + points;

                            String currentLine;

                            while((currentLine = reader.readLine()) != null) {

                                //trim newline when comparing with lineToRemove

                                String trimmedLine = currentLine.trim();

                                if(trimmedLine.equals(lineToRemove)) continue;

                                writer.write(currentLine + System.getProperty("line.separator"));

                            }

                            writer.close();

                            reader.close();

                            inputFile.delete();

                            tempFile.renameTo(new File("scores.txt"));

                        } catch (Exception excep) {

                            excep.printStackTrace();

                        }

                        //writes new line with new score

                        points += (int) Math.round(percTimeLeft \* 1000);

                        try {

                            BufferedWriter bw = new BufferedWriter(new FileWriter("scores.txt", true));

                            bw.write(Menu.username + " - " + points);

                            bw.close();

                        } catch (Exception excep) {

                            excep.printStackTrace();

                        }

                        chatArea.append(output);

                        inputTxt.setEditable(false);

                    } else {

                        output = profanityFilter(input);

                        //output sent to server to be displayed to all other players

                        chatArea.append(output);

                    }

                }

            }

            inputTxt.setText("");

        });

        submitBtn.setPreferredSize(new Dimension(200, 30));

        inputs.add(submitBtn);

        return inputs;

    }

    public Boolean containsNumbers(String input) {

        Boolean isdigit = false;

        char[] chars = input.toCharArray();

        for (char c : chars) {

            if (Character.isDigit(c)) {

                isdigit = true;

                break;

            }

        }

        return isdigit;

    }

    public String profanityFilter(String input) {

        try {

            BufferedReader in = new BufferedReader(new FileReader("censor.txt"));

            String str;

            List<String> lines = new ArrayList<String>();

            while((str = in.readLine()) != null) {

                lines.add(str.trim());

            }

            in.close();

            List<String> words = new ArrayList<String>();

            for (String word : input.split(" ")) {

                words.add(word);

            }

            String[] tempWords = words.toArray(new String[0]);

            for (String x : words) {

                for (String y : lines) {

                    if (x.toLowerCase().contains(y)) {

                        tempWords[words.indexOf(x)] =

                        tempWords[words.indexOf(x)].replace(y, new String(new char[y.length()]).replace("\0", "\*"));

                    }

                }

            }

            StringBuffer sb = new StringBuffer();

            for(int i = 0; i < tempWords.length; i++) {

                sb.append(tempWords[i] + " ");

            }

            String output = sb.toString() + "\n";

            return output;

        } catch (Exception e) {

            e.printStackTrace();

            return null;

        }

    }

    private JComponent display() {

        JPanel display = new JPanel();

        chatArea = new JTextArea(12, 45);

        chatArea.setFont(chatArea.getFont().deriveFont(20.0f));

        chatArea.setEditable(false);

        chatArea.setLineWrap(true);

        chatArea.setWrapStyleWord(true);

        scroll = new JScrollPane(chatArea);

        scroll.setVerticalScrollBarPolicy(ScrollPaneConstants.VERTICAL\_SCROLLBAR\_ALWAYS);

        display.add(scroll);

        return display;

    }

    private JComponent pics() {

        JPanel pics = new JPanel();

        pics.setPreferredSize(new Dimension(0, 300));

        pics.setBackground(Color.GRAY);

        pics.setBorder(new EmptyBorder(40, 10, 10, 10));

        //server orders images according to word and sends to all clients

        //the words are then displayed by the clients, shown below

        ImageIcon word1Icon = new ImageIcon(new ImageIcon("word1.png").getImage().getScaledInstance(400, 200, Image.SCALE\_DEFAULT));

        word1Img = new JLabel();

        word1Img.setIcon(word1Icon);

        pics.add(word1Img);

        ImageIcon word2Icon = new ImageIcon(new ImageIcon("word2.png").getImage().getScaledInstance(400, 200, Image.SCALE\_DEFAULT));

        word2Img = new JLabel();

        word2Img.setIcon(word2Icon);

        pics.add(word2Img);

        ImageIcon word3Icon = new ImageIcon(new ImageIcon("word3.png").getImage().getScaledInstance(400, 200, Image.SCALE\_DEFAULT));

        word3Img = new JLabel();

        word3Img.setIcon(word3Icon);

        pics.add(word3Img);

        return pics;

    }

    private void countDown() {

        timer = new Timer(1000, e -> {

            if (counter > 0) {

                counter--;

                String output = String.format("Time Left To Guess: %s    ", counter);

                timerLbl.setText(output);

            } else {

                if (CreatorLobby.roundNum > 1) {

                    CreatorLobby.roundNum --;

                    DrawingPhase dp = new DrawingPhase();

                    dp.show();

                    guessingFrame.dispose();

                    timer.stop();

                } else {

                    Leaderboard lb = new Leaderboard();

                    lb.show();

                    guessingFrame.dispose();

                    timer.stop();

                }

            }

        });

        timer.setInitialDelay(0);

        timer.start();

    };

}

#### LeaderBoard.java

import java.io.BufferedReader;

import java.io.FileReader;

import java.util.ArrayList;

import java.util.Arrays;

import java.util.List;

import javax.swing.JButton;

import javax.swing.JComponent;

import javax.swing.JFrame;

import javax.swing.JLabel;

import javax.swing.JPanel;

import javax.swing.JTextArea;

import java.awt.BorderLayout;

import java.awt.Container;

import java.awt.Dimension;

import java.awt.Color;

public class Leaderboard {

    private JFrame leaderboardFrame = new JFrame("Picture This! - Leaderboard");

    private JTextArea leaderboard;

    private JButton menuBtn;

    private JLabel lbl;

    public void show() {

        Container content = leaderboardFrame.getContentPane();

        content.setLayout(new BorderLayout());

        JComponent leaderboardPane = displayBoard();

        content.add(leaderboardPane, BorderLayout.CENTER);

        JComponent button = menuBtn();

        content.add(button, BorderLayout.SOUTH);

        lbl = new JLabel("Thank you for playing :)\n");

        lbl.setFont(lbl.getFont().deriveFont(20.0f));

        lbl.setHorizontalAlignment(JLabel.CENTER);

        content.add(lbl, BorderLayout.NORTH);

        leaderboardFrame.setSize(640, 360);

        leaderboardFrame.setDefaultCloseOperation(JFrame.EXIT\_ON\_CLOSE);

        leaderboardFrame.setVisible(true);

        leaderboardFrame.setResizable(false);

        leaderboardFrame.setLocationRelativeTo(null);

    }

    public JComponent displayBoard() {

        JPanel leaderboardPane = new JPanel();

        leaderboard = new JTextArea(5, 20);

        leaderboard.setEditable(false);

        leaderboard.setFont(leaderboard.getFont().deriveFont(20.0f));

        leaderboard.append("Player - Score\n");

        try{

            BufferedReader in = new BufferedReader(new FileReader("scores.txt"));

            String str;

            List<String[]> scores = new ArrayList<String[]>();

            while((str = in.readLine()) != null) {

                scores.add(str.split(" "));

            }

            String[][] array = new String[scores.size()][0];

            scores.toArray(array);

            Arrays.sort(array, (b, a) -> Integer.parseInt(a[2]) - Integer.parseInt(b[2]));

            for (String[] x : array) {

                StringBuffer sb = new StringBuffer();

                for(int i = 0; i < x.length; i++) {

                    sb.append(x[i] + " ");

                }

                String output = sb.toString();

                leaderboard.append(output + "\n");

            }

            in.close();

        } catch (Exception e) {

            e.printStackTrace();

        }

        leaderboardPane.add(leaderboard);

        return leaderboardPane;

    }

    public JComponent menuBtn() {

        JPanel btnPane = new JPanel();

        btnPane.setPreferredSize(new Dimension(0, 100));

        menuBtn = new JButton("Quit to Main Menu");

        menuBtn.setBackground(Color.decode("#a3a3a3"));

        menuBtn.setFont(menuBtn.getFont().deriveFont(14.0f));

        menuBtn.setPreferredSize(new Dimension(200, 40));

        menuBtn.setAlignmentX(JButton.CENTER\_ALIGNMENT);

        menuBtn.addActionListener(e -> {

            Menu menu = new Menu();

            menu.show();

            leaderboardFrame.dispose();

        });

        btnPane.add(menuBtn);

        return btnPane;

    }

}

## Bibliography

Below is a list of references that are of websites and other resources that I have used during the different aspects of creating this project.

1. skribbl - Free Multiplayer Drawing & Guessing Game. (n.d.). skribbl - Free Multiplayer Drawing & Guessing Game. [online] Available at: https://skribbl.io/. [Accessed 12 May 2022].
2. 2021. Microsoft Paint. Redmond, Washington: Microsoft.
3. docs.oracle.com. (n.d.). Java 2D Rendering (The JavaTM Tutorials > 2D Graphics > Overview of the Java 2D API Concepts). [online] Available at: https://docs.oracle.com/javase/tutorial/2d/overview/rendering.html [Accessed 12 May 2022].
4. docs.oracle.com. (n.d.). How to Use Buttons, Check Boxes, and Radio Buttons (The JavaTM Tutorials > Creating a GUI With JFC/Swing > Using Swing Components). [online] Available at: https://docs.oracle.com/javase/tutorial/uiswing/components/button.html.
5. docs.oracle.com. (n.d.). How to Make Dialogs (The JavaTM Tutorials > Creating a GUI With JFC/Swing > Using Swing Components). [online] Available at: https://docs.oracle.com/javase/tutorial/uiswing/components/dialog.html.
6. docs.oracle.com. (n.d.). How to Use Formatted Text Fields (The JavaTM Tutorials > Creating a GUI With JFC/Swing > Using Swing Components). [online] Available at: https://docs.oracle.com/javase/tutorial/uiswing/components/formattedtextfield.html.
7. docs.oracle.com. (n.d.). How to Use Labels (The JavaTM Tutorials > Creating a GUI With JFC/Swing > Using Swing Components). [online] Available at: https://docs.oracle.com/javase/tutorial/uiswing/components/label.html.
8. docs.oracle.com. (n.d.). How to Use Scroll Panes (The JavaTM Tutorials > Creating a GUI With JFC/Swing > Using Swing Components). [online] Available at: https://docs.oracle.com/javase/tutorial/uiswing/components/scrollpane.html.
9. docs.oracle.com. (n.d.). How to Use Text Areas (The JavaTM Tutorials > Creating a GUI With JFC/Swing > Using Swing Components). [online] Available at: https://docs.oracle.com/javase/tutorial/uiswing/components/textarea.html.
10. docs.oracle.com. (n.d.). Doing Without a Layout Manager (Absolute Positioning) (The JavaTM Tutorials > Creating a GUI With Swing > Laying Out Components Within a Container). [online] Available at: https://docs.oracle.com/javase/tutorial/uiswing/layout/none.html [Accessed 12 May 2022].
11. docs.oracle.com. (n.d.). How to Use BorderLayout (The JavaTM Tutorials > Creating a GUI With JFC/Swing > Laying Out Components Within a Container). [online] Available at: https://docs.oracle.com/javase/tutorial/uiswing/layout/border.html.
12. docs.oracle.com. (n.d.). How to Use FlowLayout (The JavaTM Tutorials > Creating a GUI With JFC/Swing > Laying Out Components Within a Container). [online] Available at: https://docs.oracle.com/javase/tutorial/uiswing/layout/flow.html.
13. docs.oracle.com. (n.d.). How to Use GridLayout (The JavaTM Tutorials > Creating a GUI With JFC/Swing > Laying Out Components Within a Container). [online] Available at: https://docs.oracle.com/javase/tutorial/uiswing/layout/grid.html.
14. docs.oracle.com. (n.d.). How to Use Swing Timers (The JavaTM Tutorials > Creating a GUI With Swing > Using Other Swing Features). [online] Available at: https://docs.oracle.com/javase/tutorial/uiswing/misc/timer.html [Accessed 12 May 2022].
15. Oracle.com. (2019). Painting in AWT and Swing. [online] Available at: https://www.oracle.com/java/technologies/painting.html [Accessed 12 May 2022].
16. Hillyard, D. and Sargent, C. (2020). Documenting Defold Programming Projects : OCR A Level Computer Science H446. Gloucester: Craigndave Ltd.