A Word... with Death!

20.05.2023

Gavin Lampe

Changelog

Version	Date	Changes
1.0.0	08/05/2023	Initial Setup
1.1.0	10/05/2023	Adding basic information
2.0	15/05/2023	Completed research answers
2.1.0	15/05/2023	Added architecture and activity diagrams
2.5.0	20/05/2023	Finished diagrams – except State Diagram

Contents

Introduction

Rationale

The reason for this project is to create a hangman-style game to demonstrate my programming abilities to get a better job with a secondary goal of seeing if it is possible to make a fun and interesting hangman game; a considerable challenge considering this is one of the most played and re-made games of all time.

Background

Navigation through the game is managed through a menu systems. Menus have been used to provide a list of choices throughout history and continue to be the most efficient, clear, and concise method to navigate through a system via a series of choices. There is not reason to try and change what has always worked.

Player input in the game will be through the use of a mouse or touch screen clicking buttons for each letter or typing the chosen letter on a keyboard. Hangman is a traditionally a simple paper-based word game, so it is most suited to a simple point-and-click input or keyboard.

Graphics in the game will utilise pixel art, pixel art was not always called pixel art; until the late 90s, it was simply the best that computer graphics could achieve.

In the late 80s, EGA was 16 colours at 320x240 pixels. Then in the 90s went from VGA with 256 colors at 640x480 pixels to SVGA with ~16M colours and 800x600-1280x768. Today, pixel art is considered anything within these parameters as these resolutions all have a “blocky” look on modern displays. Although, the more blocky the look the more a sense of ‘nostalgia’ is created in young people who haven’t even been alive long enough to have any nostalgia.

Pixel art’s added benefit is that it is easily portable between platforms and devices and is relatively easy to make.

Terminology

Game Engine
: The base code used to create a game. The game engine handles the low-level tasks required to run a game, allowing the developer to focus on the higher-level programming the game itself. Game Engines usually also include tools and applications to assist with game creation. For example, the game engine Unity has an editor and many plugins to assist with game creation.

C#
: An object-orientated programming language used by Unity

UI
: User Interface, the way the user interacts with an application. Menus, buttons, and text displayed on the screen are all part of the User Interface

UX

And extension of UI, UX is User eXperience, it refers to everything the user sees and hears when playing a game. Any feedback the user receives whilst playing is part of the UX

GUI

Graphic User Interface is any UI that uses graphics rather than just text to interact with the user.

Canvas

Unity's way of handling the User Interface (UI). The canvas object holds all the UI elements that will be shown to the user. It uses direct pixel placement to make sure the UI appears to the user as expected

Software Development Platform

A single solution or collection of tools that help developers to quickly build, test, and release applications

Sprint

A sprint is a short amount of time, usually two weeks, where jobs in a project are assigned to team members to complete during that sprint. At the end of the sprint, a meeting is held to assess progress and decide how to proceed with the next sprint.

Library

A library is a collection of classes providing code for often-used functionalities such as displaying something on the screen, mathematical equations, file operations, etc.

Proposed Design

The proposed game is a hangman-style game that has a Main Menu, Game, and Pause Menu and is created within one Unity scene using C# and implementing a Canvas UI.

The game will be made according to the following pseudo-code:

```
//MAIN MENU
	//Play
		//Starts new Game
	//Exit
		//Quits to Desktop
//PAUSE MENU
	//Return to Game
		//Toggles on and off Pause Menu
	//Return to Main Menu
		//Hides Game and shows Main Menu
	//Exit Game
		//Quits the game

//GAME
// Have a collection of words that can be used
// Choose a word at random and store it in a variable
// Display the length of the word to the user
// Have several incorrect guesses that as they are marked off draw a hang man
// While the user has guesses left and has not guessed the word
	//Prompt the user to guess a letter
	//If the guess is correct
```

```
        //Increment correct guesses by 1
    //If the guess is incorrect
        //Increment incorrect guesses by 1
        //Draw the next part of the hangman
    //If the incorrect guesses have all been marked off tell the user
        //They lost and allow user to loop gameplay
    //If correct guess of the word
        //Tell the user they won
        //Prompt to play again
```

Non-Goals

Hangman is a game that everyone has played many times, so it is difficult to make it interesting.
There are non-programming goals that I doubt I will have time to implement, but if I could, I would like to implement:

- Graphics and animations to make the game more interesting.
  - Maybe instead of a hangman, animate another penalty such as someone trapped in a device that will kill them or a bomb that must be defused.
- Sound and voice effects would also have a big impact.

Programming goals:

- High-score system in which the player's score increases with each word.
- Word score based on scrabble scoring to allow difficulties (see below).
- High-score table – Player enters name on death or quitting with top 10 score.
- Difficulties – Based on online dictionaries for Beginner, Intermediate and Advanced speakers.

Software and Hardware Requirements

The software used for this project will be:

- Unity v2021.3.19f – Free with licensing restrictions
- Visual Studio 2022 – Free under a student licence (used at TAFE)
- Rider 2022.3.2 – Free under a student licence (used at home)
- Clip Studio Paint Pro v2.0 - $50USD

The game will made on Windows 10 and will be released on Windows; however, ideally the game will also be released on Android and Linux.

The project will not be released on Mac or iPhone due to content restrictions and a lack of testing hardware available.

Research

Software Development Platforms

Unity – A free (with restrictions) C#-based game engine with built-in tools for game development. It has been used by indie developers for many years due to it's generous licensing model, and because of this, has *many* tutorials and forum posts covering almost every subject and potential problem.

Unreal – Another free (with restrictions) game engine with built-in tools for development. Unreal is owned by Epic Games and as such has larger development budget which results in more advanced and cutting edge features than Unity allowing easier development AAA-style games with advanced graphics. However, Unreal uses C++ instead of C# which has a much larger learning curve and is not as accessible to beginner programmers. Also, it has not been available to indie developers for as long as Unity, so it does not have nearly as many online resources for learning. Although Epic is working to add as many tutorials and online resources as possible to catch up with Unity.

Godot – A completely free and open-source game engine without commercial licensing restrictions that is community supported and not-for-profit. Due to being community made and run, Godot is not as advanced as Unity or Unreal; however it can use C# or GDScript which is it's own light-weight language that may be more accessible to beginners.

All three of the above development platforms have built-in tools for making development easier, support both 2D and 3D games, and are able to building your game on multiple platforms.

Software Development Methodologies

1. Agile Development
    - Focuses on the project itself and allows for constant alterations based on feedback and internal changes. The Agile process is free of a rigid framework with the development process divided into short sprints to allow fast results and feedback.
2. Waterfall Development
    - Classic software development model. A rigid framework in which the project stages are done linearly with a stage not started until the last is finished, and the project only moves forward. Easy to stay on-task but very difficult to react to changes.
3. Extreme Programming (XP)
    - XP is an agile methodology which works best for creating software in an unstable environment. It allows for greater flexibility but requires constant meetings with team members and stakeholders. The development team meets to discuss plans and issues rather than writing documentation which makes them more personally committed but can make managing the project and providing estimates very difficult.

4. Lean Development
    - In lean development value for the client is the primary focus. If something is worth it, it should be implemented immediately; otherwise remove it. Lean development focuses on reducing loss. The project is examined to avoid wasted time and money. Feedback is important so value can be assessed and actions taken quickly. This method is excellent for low-budget and time-limited projects.
5. Scrum Development
    - Scrum Development is another agile methodology. It is easy to understand and effective at getting results. Like agile, development is broken into sprints. At the beginning of a sprint assignments are set and then at the end a meeting is held to discuss the results and decide what should be done in the next sprint. While there is still a plan for the overall project, scrum allows the plan to adapt to changes in requirements or working conditions (such as a sick coworker). The downside is that it can be harder to estimate time and budget, and it will not work with large projects. Also the test team must conduct regression testing (testing recent code changes) after every sprint which is time-consuming.
6. Spiral Development
    - The spiral model is based on identify risks early on. Development starts on a small scale such as a prototype. Then the risks to the project are assessed, and a plan is made to mitigate those risks. A decision is then made on whether to proceed with the next cycle. If the risks are low, they start the cycle again. This model is good for large projects that are too risky for a full budget, but it is costly in terms of development, and if a mistake is made during the risk assessment phase, the whole project could fail.
7. Prototyping Methodology
    - This is based on the waterfall method, but instead of doing full development, only a prototype is created and provided to the client to show functionality and obtain feedback before proceeding with the full project. This method can minimise risk as issues can be detected early and helps develop UI/UX and functionality through prototype feedback. Although, if the client requests multiple changes to the prototype, the budget and project length may increase.
8. Feature Driven Development (FDD)
    - FDD is an Agile development method that takes a client-valued feature perspective. This means that the project is assessed and broken down into features that are of value to the client; a feature is a function that the client wants such as “Verify this customer’s login credentials”. Features take no more than two weeks to complete and are assigned as classes to the programmers. FDD is most suited to large projects with multiple teams as it allows them to work concurrently without getting in each other’s way, but it does mean that each programmer or team must take ownership of their feature as the responsibility is not shared with others.
9. Rapid Application Development (RAD)
    - As the name implies, RAD is designed to achieve results fast. It focuses on quick prototype releases and iterations. Feedback is received quickly so errors can be fixed and results received quicker than other methodologies. The client is

encouraged to provide constant feedback for improvements and core functions can be easily changed during the prototyping phases. It is not suitable for small projects and requires experienced developers and strong collaboration. Development costs can also be higher than other methods.

10. Joint Application Development
    - Joint application development has everyone involved in the project development. Sessions are run involving a facilitator, the client, developers, end-users, mediators, and experts. A lot of effort is put into eliminating errors to avoid costly mistakes. This model creates close communication between all parties which can speed up development as well as get feedback and find issues quickly. However it requires a large budget to get started and needs highly experienced professionals.

Integrated Development Environment

An IDE is a software application that provides most, if not all, the tools required for programmers in software development. An IDE will at a minimum include a source code editor with tools to highlight different structures and objects with the code as well as a debugger for finding mistakes. They will often have tools for finding references in other scripts and documents as well as organising the code in a development project.

Two of the most popular IDE used in Game Development are Microsoft Visual Studio and JetBrains Rider.

S.O.L.I.D Design Principles

SOLID is an acronym used to help programmers of object-oriented languages write code that is clean, flexible, and efficient and with minimum redundancy. Although the acronym and it's principles were designed to be applied to object-oriented design, they can also be applied to agile software development.

The five principles of SOLID are:

The...

- Single Responsibility Principle
    - This principle states that a class should have just one responsibility so that only one type of change in the software will affect the class. It also makes it easier to collaborate because team members will not be working on the same script (generally), and it makes marge conflicts less likely to happen.

- Open-Closed Principle
    - This principle states that a class should be open to extension but closed to modification. This means that we should be able to add new functionality to a class without needing to modify the existing code.

- Liskov Substitution Principle
  - This principle states that a subclass or child class should be able to pass as its superclass or parent class. So if a method is expecting an object that is the superclass, we should also be able to send it the subclass without getting any errors. This is because the subclass should only extend the superclass and not change anything it has inherited.

- Interface Segregation Principle
  - This principle states that interfaces should be separated into component interfaces so that the interfaces can be extended or have functions removed without having to add workarounds or extensively modify existing code.

- Dependency Inversion Principle
  - This principle states that our classes should not depend on real classes and functions, but instead should depend on interfaces or abstract classes. This goes hand in hand with the Open-Closed Principle in that to extend a class without modifying it we can depend on an interface instead of the actual class itself.

Programming Methodologies

Procedural Programming

The original programming methodology which was widely used in the 80s and 90s before object-orientated became a thing with languages such as BASIC, Pascal, and C. In procedural programming, code is written and executed in order line by line. Because the focus is on writing the code in blocks of procedures that are executed line by line, it is relatively easy for beginners to learn and easier to implement compilers to parse the code. In some earlier and simpler procedural languages, all the code was written in one file and the only way to jump to another block of code was to use GoTo statements to 'jump' to another line, but too many GoTos created "Spaghetti Code" which is messy and hard to follow and debug.

Object-Oriented Programming

In OOP, a program is made up of objects. An object has a set of attributes or properties and a set of associated actions. It is like a blueprint of a thing. For example, a car object would have attributes such as 'colour' and 'model name' and also has actions such as start, stop, accelerate, and slow down. The main point of OOP is to communicate with objects using their public attributes. Using abstraction (showing only what is necessary outside the object) and encapsulation (wrapping the methods and related variables and properties together as a single unit), an object becomes a black box which can take information and return results but is otherwise hidden from the rest of the program. This makes reusing code and extending a program without having to re-write large amounts of code very easy. It is more efficient but harder to design as everything must be considered and mapped out before hand.

Functional Programming

Functional programming treats everything as a mathematical function. Functions are treated as data so you can use them as parameters, return them, and build other functions from them. But they are pure functions in that they do not change with the program state. When given an input, they will always return the same result regardless of what state the program or system is in. Because a function is not sharing states you don't need to know every state and the details of every shared variable to understand a functions effect.

Logical Programming

In logical programming, program statements are expressed as facts and rules about problems using formal logic such as "A = X" and "B is true if A = N" (Therefore B must be false). Some logic programming languages allow for statements about what should be accomplished but with step-by-step instructions on how to get there. Logic programming is especially useful for fault diagnosis, machine learning, and natural language processing, so it's heavily used in AI.

Open-Source Development tools

Open-source means that the code of a software project is publicly viewable. Open-source does not automatically mean that the code can be copied without restrictions. In fact, almost all open-source code has licensing restrictions of some kind with the most common being that any project that uses the open-source code must also be open-source and use a similar licence. Open-source tools and applications are almost always free to use and developed by the community rather than a business.

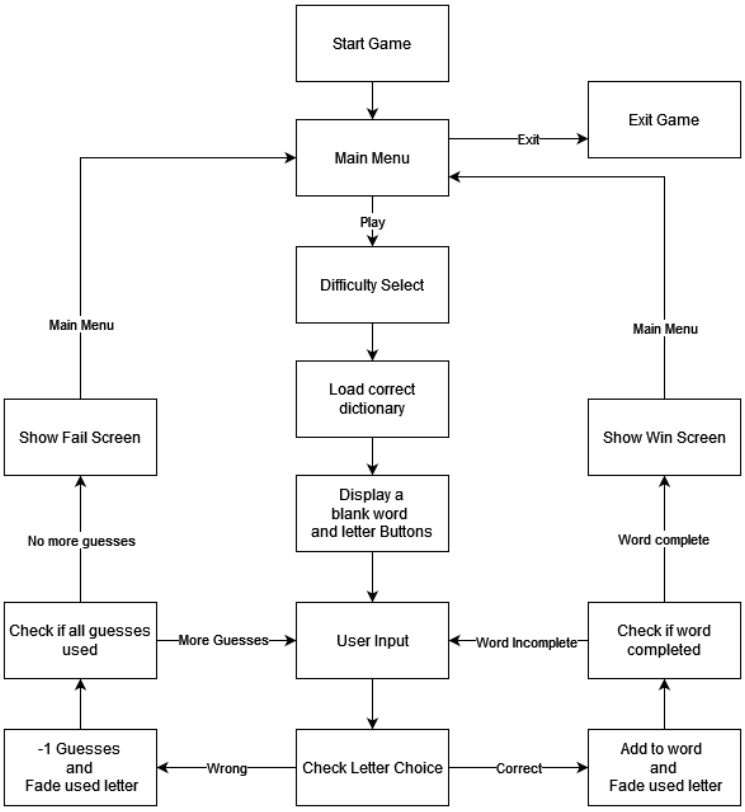
There are many open-source tools available to programmers such as:

- Godot – A game engine and integrated tools for 2D and 3D game development
- Adventure Game Studio – An IDE for making 2D point-and-click adventure games
- Twine – A stand-alone or browser-based app for writing non-linear fiction, which is storytelling with different possible outcomes as found in RPG and adventure games.
- GDevelop – A "game-making app" that allows you to create games without using code, which is particularly useful for getting beginners to focus on good game design without getting confused or slowed down by code.

Besides these application tools there are also many open source scripts and scripted frameworks available for Unity that save programmers time. For example, Ezy-Slice is a framework of C# scripts that allows you to cut meshes in-game in real-time. For example, a player could use a knife to cut a lemon and it will cut correctly creating two new objects. This would take a long time to program correctly and debug, so utilising a free open-source asset such as this allows a programmer to focus on other tasks.

System Architecture

Architecture Diagram



Data types

- Integer – Used to store whole numbers such as 1, 256, -10
- Float – Used to store numbers with a decimal place such as 1.596874 and 0.2
- Char – Used to store single alphanumeric characters such as “A”, “b”, or “&”
- String – Used to store a group of alphanumeric characters such as “hello!”
- Vector2 – Used to store a set of co-ordinates for X and Y planes; co-ordinates stored as floats
- Vector3 – Same as Vector2 but also includes co-ordinates for the Z plane
- GameObject – A data type for storing Gameobjects which are the base entities of a Unity scene
- List – A list is an ordered collection of elements which can be any data type.
- Array – An array is a collection similar to a list except that it has a predefined size, making it less flexible but more efficient.
- Dictionary – A dictionary is also a collection but each element has a key that maps to it’s value, like giving a name to each item. That key can be used to retrieve and assign the element.
- TextAsset – A TextAsset is a Unity type for any text file which in this project will be used to store a list of words that can be retrieved at run-time.

Data Model

difficulty: integer
– Created by GameManager(); passed to DictLoad() to choose correct textAsset
dictionaryFiles: textAsset array
– Created by DictLoad() to store 3 textAssets – text files that contain a list of words
wordList: string array
– Created by DictLoad() to store word list from textAsset; passed to GameManager()
wordPlayed: string
– Created by GameManager(); used to store random word from wordList array;
passed to Score() to calculate word score when completed
letterChosen: char
– Created by GameManager(); used to store letter chosen by player;
compared to against each letter in wordPlayed
guesses: integer
– Created by GameManager(); used to store how many guess the player is allowed
guessesUsed: integer
– Created by GameManager(); used to store current guesses used; compared against guesses
score: int
– Created by Score(); used to store word score once calculated and then displayed on screen
newGame: bool
– Created by GameManager(); used to determine if the player is continuing a game (just choose a new word) or starting a new one (needs to choose a difficulty first)

Interface/API Definitions

Scene (https://docs.unity3d.com/Manual/CreatingScenes.html)
– The main file where all objects are placed and the game takes place. The scene file literally holds the scene that the player will see as well as any elements that affect the scene such as cameras, lighting, and other gameplay (rather than visual) objects. A game can be one scene or split across multiple scenes.

GameObject (https://docs.unity3d.com/ScriptReference/GameObject.html)
– Any object in a unity scene. A scene is made of GameObjects whether they be the camera, the player model, environment, UI, or just an empty object that holds a script such as the Game Manager.

Canvas (https://docs.unity3d.com/Packages/com.unity.ugui@1.0/manual/class-Canvas.html)
– The canvas GameObject that holds UI elements such as buttons and text boxes. The canvas game object is different from other game objects in that instead of using Unity units for distance it uses pixels for exact positioning of UI elements on the screen in 2D as opposed to 3D space.

Image (https://docs.unity3d.com/Packages/com.unity.ugui@1.0/manual/script-Image.html)
– Image components will be used to hold PNG graphic files for the game art such as environment and characters an Image component is used with Canvas UI

Sprite – Sprites are similar to images in that they hold a 2D graphic but they are used in-game, are usually animated, and don't require Canvas UI

Animator Controller (https://docs.unity3d.com/Manual/class-AnimatorController.html)
– Allows you to arrange and maintain multiple animation clips and transitions for a GameObject. Even a single clip needs an Animator Controller to be attached to a GameObject.

UnityEngine (https://docs.unity3d.com/ScriptReference/index.html)
– UnityEngine is the core Unity library. It contains the classes and types that handle all Unity's core functions such as GameObject properties and methods for GameObject manipulation, methods for handling audio, camera, and lighting. As well as methods for handling interaction with the Unity Editor and Inspector.

UnityEngine.UI (https://docs.unity3d.com/Packages/com.unity.ugui@1.0/api/UnityEngine.UI.html)
– Contains classes and types to assist with adding and manipulating UI elements.

SceneManagement (https://docs.unity3d.com/ScriptReference/SceneManagement.Scene.html)
- Contains classes and types to assist with finding information about the current scene and actions such loading and closing scenes and switching between them.

System.IO (https://learn.microsoft.com/en-us/dotnet/api/system.io?view=net-7.0)
- A generic C# library that contains classes and types for reading and writing to files and data streams as well as basic file and directory manipulation.

Risks

There are no risks. The project can easily be completed to specification within the time frame.

Additional features such as a persistent high score table, animations, and a hint system have a high risk of not being added due to time and motivation constraints.

Alternatives

A basic UI drawn hangman using rectangle and circle gameObjects to draw the hangman was considered but abandoned as the only thing separating hangman-style games is the UX style, so if that is bland and uninteresting, no one will want to play the game for long.

Pseudocode

System Pseudocode

(Panel = Canvas UI Panel)

<Menu Panel enable by default when scene loads>
Quit Button_onClick > exit application
Play Button_onClick > enable Game Panel; enable Difficulty Panel; disable Menu Panel
Beginner/Intermediate/Advanced Button_onClick
    Score = 0
    Switch (Beginner, Intermediate, Advanced)
    parse matching dictionary file into a string array splitting on the carriage return

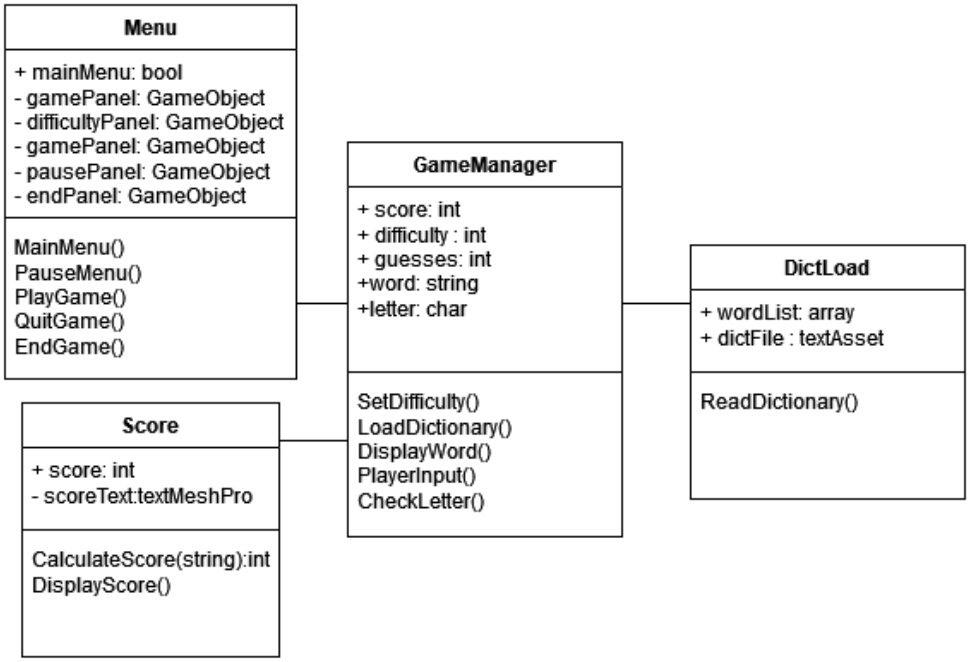
Randomly choose string from array; count characters
Instantiate correct number of letter spaces
Letter Button_onClick > check character against word string
    If character is found, put letter in correct space and fade letter button
    If all letters of word string guessed correctly,
        score += (scrabble scoring of letters); enable Win End-Game Panel
    Else, display failed guess image/animation, and fade letter button
    If number of guesses equals maximum, show Failed End-Game Panel

Win End-Game Panel:
    Display score
    Next Word Button_onClick > disable End-Game Panel; new random word chosen
    Quit Button_onClick > enable Menu Panel; disable Game Panel; disable End-Game Panel

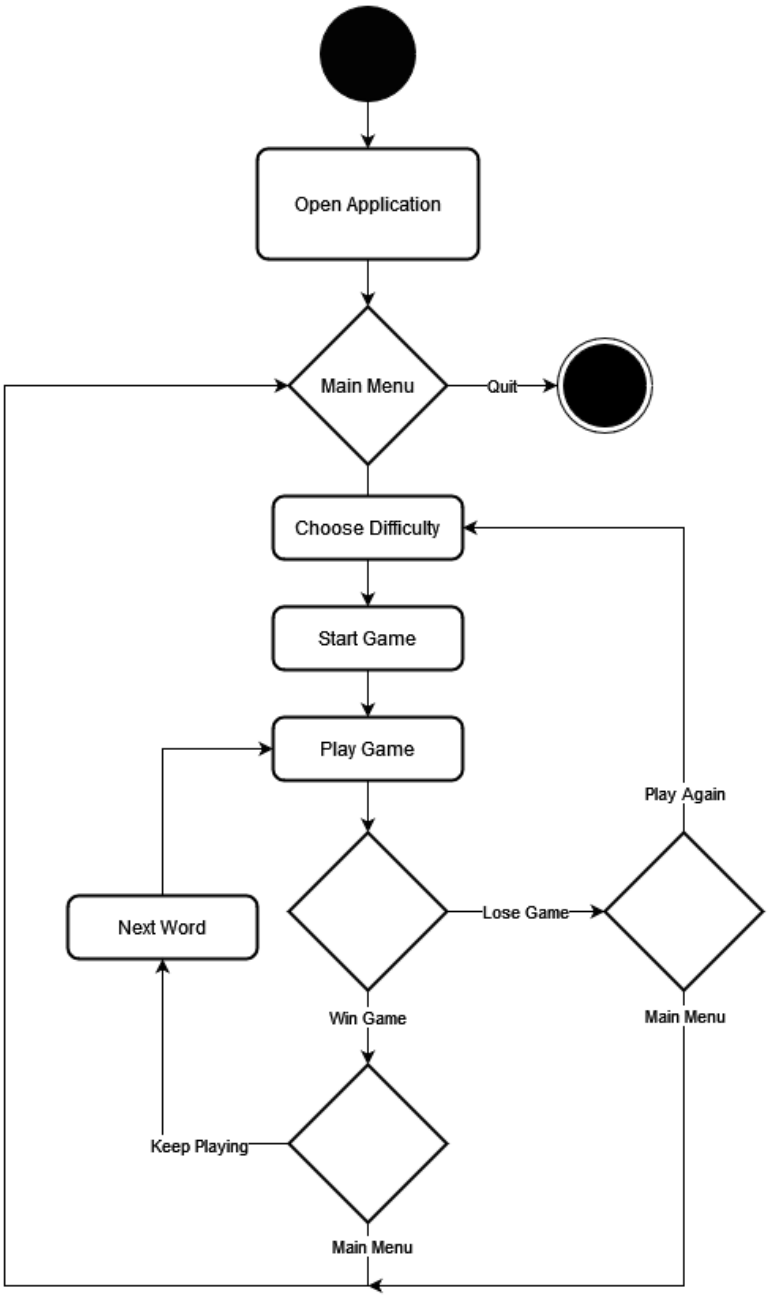
Failed End-Game Panel:
    Try Again Button_onClick > enable Difficulty panel; disable End-Game Panel
    Quit Button_onClick > enable Menu Panel; disable Game Panel; disable End-Game Panel

UML Diagrams

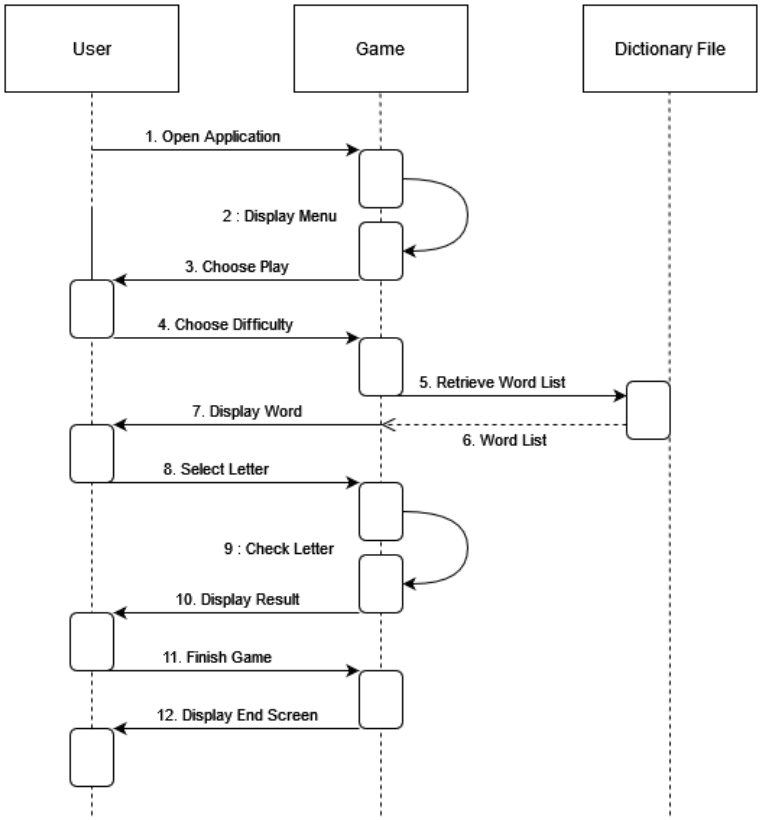
UML Class Diagrams



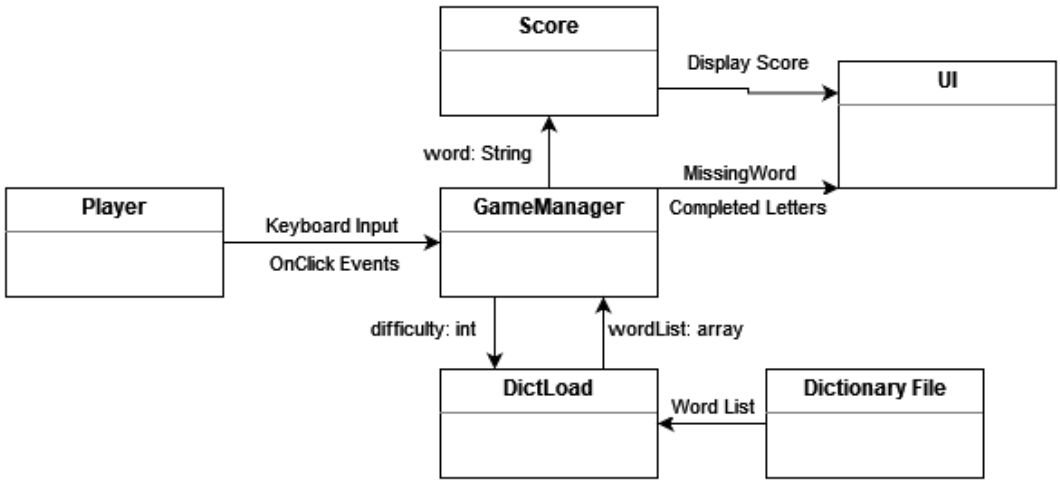
UML Activity Diagrams



UML Sequence Diagram



UML Communication Diagrams



UML State Diagrams

/ State diagram shows the different states of an entity. /

Sign Off

Name

James-Rae

Role

Lead Programmer

Signature

Date

[Click/tap to select date]

Testing

Errors and Bugs

Outline the test classes used. Add rows to table as required.

Class Name	Description of Error	Screenshots of testing	Solution
[Name of Class or Object that the error is connected to]	[Description of the error/error message]	[Add and resize relevant screen shots]	[Explain solution/fix to error]

Evaluation

Reflection

/Provide a self-reflection on your performance. /