Final Year Project

Sound-It-Out Spell-Checking Game

Iva Mechkarova

Student ID: 18345221

A thesis submitted in part fulfilment of the degree of

BSc. (Hons.) in Computer Science

Supervisor: Prof. Julie Berndsen

UCD School of Computer Science

University College Dublin

May 3, 2022

Table of Contents

Abstract

Sound-It-Out Spell-Checking Game is a mobile game targeted at children living in Ireland, with the purpose of gathering data on how they spell. Children who are unfamiliar with the correct spelling of a word are often encouraged to use a "sound-it-out" method, i.e., breaking the word down into its individual sounds and selecting letters to represent each identified sound [31]. Naturally, this leads to children commonly making misspellings that have a close phonemic distance to the intended word. Most widely-available spellchecking tools are not capable of correcting such spelling errors as they are usually developed to correct typographic errors [32], and therefore, do not check the phonemic distance. However, *S-Capade* [32] is a phonemic-based spellchecker aimed at correcting misspellings made by children, but it currently does not have sufficient data. *Sound-It-Out Spell-Checking Game* aims to be a means for the *S-Capade* team to efficiently gather the necessary data to better their spellchecker. This paper outlines how the game is designed to motivate children to play and how data is collected as a side effect of them playing.

Chapter 1: Project Specification

1.1 Problem Statement

The goal of this project is to devise an engaging and intuitive video game targeted at children living in Ireland, with the purpose of gathering data on how they spell. Video games have been used as a means of researching individuals for over 20 years [17] and nowadays, research suggests that 91% of children aged between 2 and 17 play video games in the United States [16]. Therefore, an intuitive and engaging game is an ideal method for gathering data from a large set of children. Throughout the game, the children will be prompted to spell a variety of words. In cases where the spelling varies greatly from the given word, they will be encouraged to use a "sound-it-out" method - hence the title, "*Sound-It-Out Spell-Checking Game*". We are interested in gathering this data as it will be used to better the accuracy of *S-Capade: Spelling Correction Aimed at Particular Deviant Errors*, which is a phonemic-based spellchecker aimed at correcting misspellings made by children [32]. Throughout the development of the game, ethics must be accounted for as ethics permission would be required before the game is deployed with children.

1.2 Background

Children who are unfamiliar with the correct spelling of a word are often encouraged to use a "sound-it-out" method, i.e., breaking the word down into its individual sounds and then selecting letters to represent each identified sound [31]. Research suggests that there is a positive correlation between children's ability to segment words into phonemes (sounds) and reading success [5]. However, the "sound-it-out" method often results in misspellings that, at first glance, differ greatly from the intended word [31]. This is because English spellings are not phonetic but lexical (standard spelling) in nature [47]. Therefore, if we were to calculate the phonemic difference between a child's spelling and the target word, we would see that they do not differ as much as first believed. [32]

Furthermore, spelling errors are generally one of two types: typographic or cognitive [23]. The misspelling made by a child due to using the "sound-it-out" technique would be an example of a cognitive error. On the other hand, an example of a typographic error would be pressing an adjacent character on the keyboard [31]. Most widely-available spellcheckers are based on typographic errors. Therefore, they are not fully capable of correcting children's spelling [32], especially if they have undergone the popular "sound-it-out" approach. However, *S-Capade* is a phonemic distance-based spellchecking tool aimed at correcting the misspellings made by children [32]. The project *Sound-It-Out Spell-Checking Game* is intended for gathering more data on how children spell in order to better the accuracy of phonemic-based spellcheckers, such as *S-Capade*.

1.3 Related Work

Nowadays, there is an abundance of video games and apps aimed at children. Two very famous examples are the games "*Fruit Ninja*" [13] and "*Color Switch*" [8]. The aim of *Fruit Ninja* is to slice as many fruits as possible. The player is awarded points for simply slicing fruits that appear on the screen. However, as the game progresses, the difficulty rapidly increases. Once the player misses three pieces of fruit or slices a bomb (sometimes bombs appear instead of fruit), the game ends. In *Color Switch*, the player controls a small circle, which is either blue, purple, orange, or pink. The player moves the circle up by tapping on the screen. Ahead of you are a variety of moving or rotating obstacles that alternate between the four colours. The goal is to move through these obstacles, but you can only pass through the colour that your circle resembles. If you pass through the wrong colour, the game ends.

Both of these games have one major similarity - the player accumulates points until a stopping condition is met, which ends the game and resets their score. These types of games are often monetised by giving the player the option to watch an advertisement in exchange for multiplying their score or continuing from where the game ended [51]. This idea can be applied to *Sound-It-Out Spell-Checking Game*. However, since our main objective is not monetisation (it is data collection), we can prompt the player to spell a collection of words in exchange for multiplying the player's score or continuing the game from where it ended.

1.4 Datasets

We will need a dataset with a variety of English words with a varying spelling difficulty - the exact dataset is to be decided at a later time.

1.5 Resources Required

- An off-the-shelf text-to-speech synthesizer to dictate each word.
- A plugin that can break a word into its phonemes. This will be made available by the creator of *S-Capade*, O'Neill.
- *S-Capade*'s phonemic edit-distance calculator. This will be used to calculate the phonemic distance between the target word and the inputted word. We can then set a threshold and only accept inputted words that have a distance below this threshold.

Chapter 2: **Introduction**

As mentioned in chapter 1, the goal of this project is to devise an engaging and intuitive video game targeted at children living in Ireland, with the purpose of gathering data on how they spell. It was decided to implement *Sound-It-Out Spell-Checking Game* as an Android application with a MySQL database. Storing the data produced by the game in a MySQL database ensures that the data can be easily extracted and used by the S-Capade [32] team that is currently lacking data for children living in Ireland. This report outlines how each aspect of the game was devised, implemented, and evaluated. This includes an exploration of various research conducted to help devise the game's design, an outline of how the project was planned to be approached, a thorough explanation of how each aspect of the application was implemented, an evaluation of the final implementation, and an outline of areas that should be improved in the future.

Chapter 3: Related Work and Ideas

In this chapter, we will focus on the research conducted for this project. The lack of data for phoneme-based spellcheckers, data collection methods utilised with children, games with a purpose (GWAP), the possible effects of screen time on children, and the educational benefits of video games will be discussed in this chapter. We will also explore different game development technologies and emphasise the importance of obtaining ethical approval before deploying a game with children.

3.1 Lack of Data for Phoneme-Based Spellcheckers

As mentioned in chapter 1, spelling errors are generally one of two types: typographic or cognitive [23] and most widely-available spellcheckers are based on typographic errors, which are not fully capable of correcting children's spelling [32]. On the other hand, since children are often encouraged to use the "sound-it-out" method when learning to spell [31], phonemic-based spellcheckers perform better on children's misspellings [32]. Therefore, we will focus on this type of spellchecker in this section.

An algorithm devised by French linguist and computer scientist Jean Veronis is an early example of an algorithm that uses weighted edit distances where the phonetic closeness of graphemes determines the cost of the operations [52]. Another early example of a phonemic-based spellchecking algorithm is *Soundex* [23]. This algorithm maps each word to an alpha-numeric code according to its characters and then assigns numeric values to groups of characters that are phonetically similar [32]. Although *Soundex* would likely perform better than a typographic spellchecker on misspellings made using the "sound-it-out" method, it has been found to be too general in evaluations carried out in both English [18] and Portuguese [29].

In recent years, other phonemic-based spellcheckers tailored for correcting misspellings made by children, such as *Kidspell* [10] and *S-Capade* [32], have emerged. As these spellcheckers have only recently been developed, they still lack data that could use their full potential. The figure 3.1 shows the datasets used by *S-Capade* and from this, we can see that the spellchecker used 9,754 target words between the five datasets. Moreover, the *Zeeko* dataset was the only dataset comprised of Irish children's misspellings, and as seen below, there were only 163 target words and 232 misspellings recorded.

Dataset	Misspellings	Misspellings used	Target words	Publicly available
Birkbeck	36,133	33,887	6,068	Yes
Holbrook	1,791	1,562	1,177	Yes
Wikipedia	2,455	2,230	1,909	Yes
Aspell	531	515	437	Yes
Zeeko	232	232	163	No

Figure 3.1: Datasets used by *S-Capade* [32]

However, according to *Merriam-Webster*, there are around 470,000 words in the English language

[19]. We can assume that the average child will only have a fraction of these words in their vocabulary, but the number of words will still be in the thousands. Therefore, it is clear that *S-Capade* is lacking data, especially for children living in Ireland. Furthermore, it is believed that the regional pronunciation of a child using the "sound-it-out" method would affect their spelling [31]. Hence, we are interested in devising an intuitive way to gather data on how children living in Ireland spell.

3.2 Data Collection Methods Utilised With Children

There are many methods that we could use to gather data from children. A paper by Flanagan et al. [12] explores the strengths and limitations of these various data collection methods, along with discussing other aspects that should be considered when collecting data from children, teenagers, or young people. For example, it is noted in the paper that ethics must always be considered when collecting data from children, regardless of the utilised data collection method. Moreover, the sensitivity of the data being collected should be considered when deciding on a method. This is also important to ensure compliance with GDPR when collecting any personal data, especially from children aged under 16 [4]. However, GDPR only covers personal data, which the European Commission defines as "any information that relates to an identified or identifiable living individual" [53]. Therefore, we must ensure that **only** non-personal data is stored, such as spelling attempts and age range, e.g., 2-5, 5-8, 8-11 years. Personal data such as name, date of birth, or location should never be requested or stored. Ethical approval will need to be obtained before collecting any data from children, regardless if it's personal or non-personal data.

Traditional data collection methods include 'pen and paper' questionnaires, interviews, and focus groups [12]. The authors compared these to new innovative methods, including digital technologies and the use of games, and found that online data collection methods have the advantage of being cheaper, easier to store and to analyse, and may offer a more representative sample. This is perhaps as it is easier to access and participants can complete it in their own time and place. Although remote means of data collection have these advantages, the paper [12] also highlights that face-to-face research allows the researcher to check the participants understanding of the questions/task, and that digital data collection carries the risk of excluding those from poorer socio-economic backgrounds. However, research suggests that 90% of Irish adults currently have a smart phone [38] (we can assume that most would allow their child to use it). Also, all Irish primary schools are expected to have IT resources that are accessible to pupils [20]. Therefore, if we were to use a digital means of data collection, we can assume that it would be possible to include the majority of Irish children.

3.3 Games With A Purpose

A very interesting form of digital data collection is a method devised by von Ahn called "*Games With A Purpose*" a.k.a "*GWAP*" [1]. GWAPs are video games that have more than just entertainment purposes, such as the purpose of researching individuals [17], solving computational problems [1] or gathering data [2]. GWAPs are developed to achieve these purposes, as a side effect of simply playing the game. Moreover, video games, whether they are played on a console, computer, or mobile phone, are estimated to be played by 91% of children aged between 2 and 17 in the United States [16], and we can assume that the number is similar in Ireland. Given that our goal is to gather data on how children spell, it is clear that an engaging and intuitive GWAP would be an

ideal method for gathering this data.

3.3.1 Improving Image Search Engines

The very first popular game with the purpose of collecting data is von Ahn's "*ESP Game*", where two players were randomly paired together, shown the same image, and then had to guess what label the other player would give to the image [1, 2]. If the two players guess the same label, they are awarded points, and the next image is shown [1]. A license for this game was later bought by Google and became known as the "*Google Image Labeler*", which was a powerful GWAP that significantly improved Google's image search engine [2]. The main reason behind the project's success is that the game was designed to be entertaining (some people played for over 40 hours per week) whilst ensuring that the collected data is free from errors. This was ensured as it was found that the label which two players agree on is typically an accurate one [1].

Figure 3.2: Gameplay from the *ESP Game* [2]

However, a different study has criticised this GWAP, stating that it does not add *informative* labels to the images [39]. The study suggests that players were inclined to label images with generic words, such as "car" instead of "red bmw". In an attempt to prevent this, some images had "taboo words" that players could not use [1, 2], as seen in figure 3.2. Moreover, the study also criticises the game as there were often a lot of synonyms present, such as "man" and "guy", and many labels were easily predictable given the existing labels, such as "clouds" if "sky" was present [39]. Nonetheless, the game is unarguably one of the most successful examples of a GWAP, collecting over 50 million labels by 2008 [2]. It has also inspired many similar games to be developed [24].

3.3.2 Identifying Social Information in Text Messages

One GWAP that was inspired by *ESP Game* is a game which aimed to gather data on how people interpreted social information in a text message [34, 33], for example, identifying *deception, embarrassment* or *intent to persuade*. In this GWAP, players were encouraged to write messages that reflect specific social information and label the social information in messages written by other players. The players were awarded points when writing a message that gets labeled correctly or when labeling a message correctly themselves, which encouraged players to provide accurate answers and consequently reduced noisy data. Moreover, the game had multiple other ways of reducing errors, such as "taboo words" which players could not use when writing a message, and having multiple players label each message. The data gathered from this GWAP was used to train a machine learning classifier to identify the social information in text messages, which, as seen in figure 3.3, exhibited similar behaviours to a human. In fact, it even performed better than a human at classifying certain social information. For example, it labeled 76% of messages containing *embarrassment* correctly, as opposed to 69% of humans. Similar to *ESP Game*, this project clearly shows the power of using a GWAP for large-scale data collection.

	deception	politeness	rudeness	embarrassment	confidence	disbelief	formality	persuading
deception	**.36**	.08	.19	.08	.08	.09	.06	.08
politeness	.05	**.49**	.12	.12	.05	.01	.12	.05
rudeness	.06	.06	**.63**	.04	.07	.07	.01	.07
embarrassment	.02	.01	.11	**.76**	.06	.03	.01	.00
confidence	.06	.01	.04	.08	**.68**	.02	.03	.08
disbelief	.08	.03	.08	.02	.09	**.56**	.02	.12
formality	.00	.26	.06	.03	.00	.06	**.43**	.15
persuading	.05	.06	.09	.03	.11	.03	.02	**.61**

(a) Confusion matrix for the machine learning classifier. [33]

	deception	politeness	rudeness	embarrassment	confidence	disbelief	formality	persuading
deception	**.45**	.05	.10	.01	.07	.07	.03	.21
politeness	.03	**.71**	.03	.00	.01	.00	.13	.09
rudeness	.03	.00	**.92**	.00	.01	.02	.02	.00
embarrassment	.04	.08	.05	**.69**	.00	.11	.01	.02
confidence	.01	.04	.02	.01	**.82**	.01	.01	.09
disbelief	.05	.03	.02	.02	.05	**.82**	.00	.02
formality	.02	.34	.02	.01	.03	.03	**.46**	.10
persuading	.03	.05	.01	.00	.05	.03	.01	**.82**

(b) Confusion matrix for the human participants, where the majority of participants agreed on a message's intended social information and at least two humans labeled the message. [33]

Figure 3.3: Machine Learning Classifier versus Human Participants

3.3.3 Annotating Music and Sound

Thus far, we have only mentioned successful examples of GWAPs. However, the "*TagATune*" prototype is an example of a GWAP which initially failed [24, 25]. Similar to *ESP Game*, this game paired two players together, played them the same sound, and encouraged players to agree on a description for the sound in exchange for points [25]. The reason behind the prototype's failure is that it used the same *output-agreement* method as *ESP Game*, i.e., the players had to agree on a description for the tune. Although the *output-agreement* method worked successfully for both image labeling [1] and text labeling [33], it was not ideal for audio data collection as it was very difficult for two players to agree on a description. This was made clear by a pilot study which showed that players chose to pass 36% of the time instead of attempting to describe the sound [25]. Consequently, the players did not enjoy the game as much as other GWAPs, and a questionnaire found the average enjoyment of the game to be 3.3/5. This was clearly an issue as, according to von Ahn (2008), "perhaps the most important aspect of GWAP is that the output is produced in a way that's designed to be enjoyable" [2].

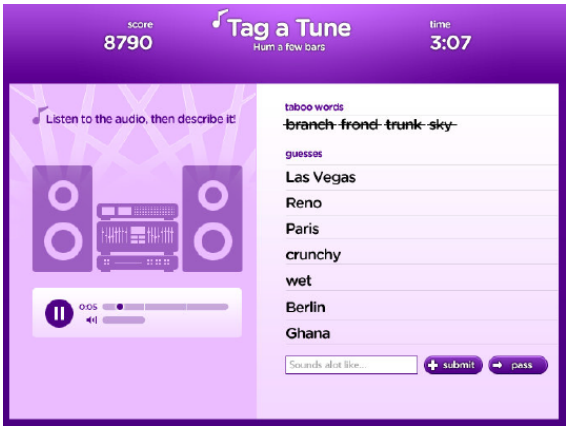
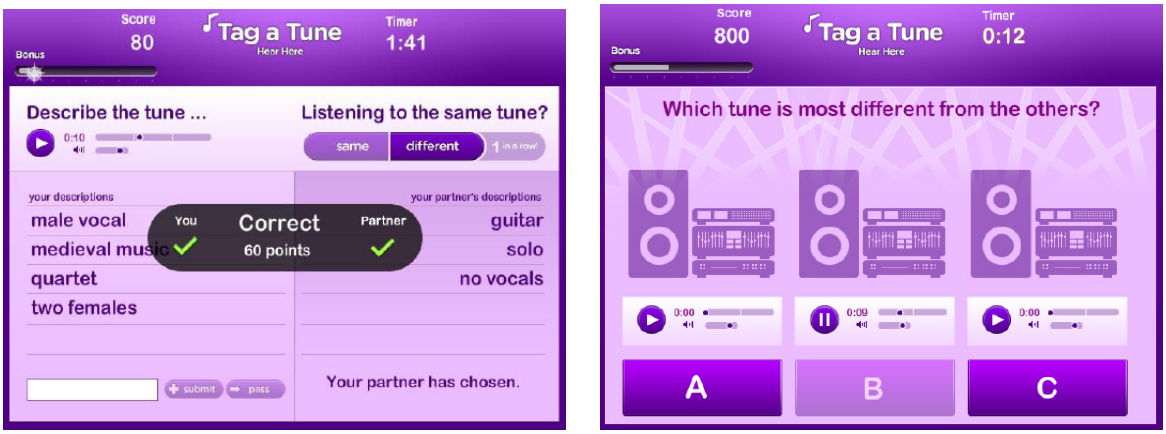


Figure 3.4: Prototype Design [24]

However, the lessons learned from this prototype led to the creators devising a different validation method called *input-agreement*, which was used for the final design of TagATune [24]. In this method, the two players either heard the same tune or different tunes, and as opposed to the prototype, the players can see the labels that their partner is giving to the audio. Then, the players must decide if they were listening to the same or different tune and were only awarded points if they *both* guessed correctly. This scoring mechanism was a clever method of ensuring that players labeled the audio accurately, as the points were only awarded if both guessed correctly. The creators also attempted to combat the difficulty of the game by adding a bonus round, where

players heard three different tunes and were asked to agree on which is the most different. The data collected from this round was used to tweak the game's difficulty, as two very similar tunes would likely require more specific labels for players to correctly guess if they are listening to the same audio. In turn, this increased the enjoyment of players.



(a) Regular Round [24]

(b) Bonus Round [24]

Figure 3.5: Snippets from TagATune's final design

3.4 Designing a GWAP for Children

According to von Ahn and Dabbish [2], there are multiple essential factors to consider when designing a GWAP. In the paper, it is stated that "perhaps the most important aspect of GWAP is that the output is produced in a way that's designed to be enjoyable" (p. 63). The **enjoyment of the game is key** to a successful GWAP, as players are not paid to play the game. Therefore, GWAPs rely on people playing the game solely based on their desire to be entertained. Of course, the terms "fun" and "enjoyable" are subjective, and the target audience (children) will need to be considered when designing our game. Moreover, the target audience should also be considered when deciding where the game should be deployed (desktop/laptop, mobile or, gaming device).

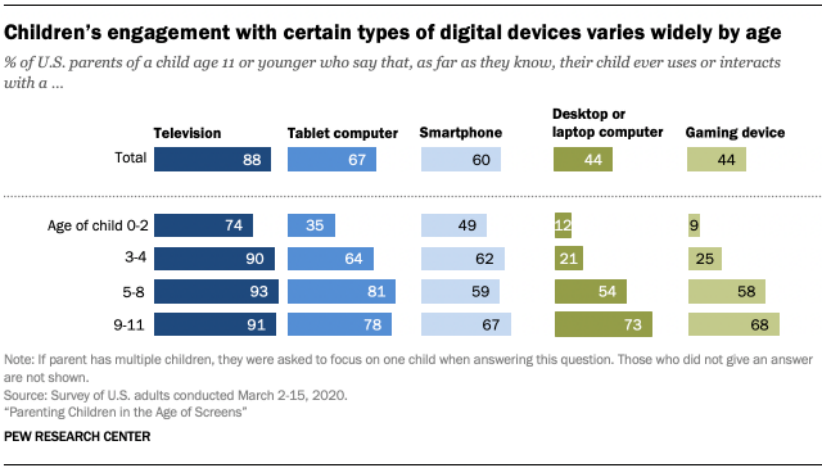


Figure 3.6: Children's engagement with digital devices [7]

As seen in figure 3.6, a study [7] found that children below the age of 11 engage with smartphones

and tablets more than desktops/laptops and gaming devices. 60% of the parents surveyed in the study said that their child started engaging with a smartphone before the age of 5, and 17% reported that their child has their own smartphone. Therefore, it is clear that our game should be **deployed as a mobile application** for it to be most enjoyable and engaging.

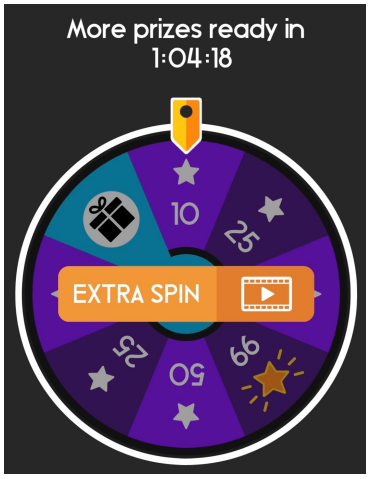


Figure 3.7: Screenshot of *Color Switch* Advertisement Prompt

Furthermore, in [2], it is declared that a game can be defined "successful" if enough hours are spent playing it and hence, we explored the famous mobile applications *Fruit Ninja* [13] and *Color Switch* [8] in Chapter 1. As previously mentioned, in these types of games, the player accumulates points until a stopping condition is met. Then, the stoppage of the game is often used for monetisation purposes by giving the player the option to watch an advertisement to multiply their score. Other rewards for watching advertisements are also often offered, such as in figure 3.7, where the player is given the option to watch an ad in order to get extra bonus "prizes". A variation of this principle where children are encouraged to spell different words, rather than watch advertisements, can be adopted by *Sound-It-Out Spell-Checking Game*. The stopping condition of both Fruit Ninja and Color Switch is failing to overcome the obstacles that occur. It makes sense that both of the games became so popular, as another paper explains that the key motivation behind playing games is voluntarily attempting to overcome unnecessary obstacles [28]. Therefore, it is clear that our game should have some **child-friendly obstacles** that increase in difficulty as the game progresses, and points should be awarded for overcoming them.

Another effective way to motivate players to continue playing the game and increase their enjoyment is *player skill levels* [2]. This means that as a player accumulates points, they are promoted to a higher *skill level*. We can incorporate this idea into our game. However, as we are designing our game for children, perhaps it would be better to allow them to **use their points to unlock new characters**. As seen in figure 3.8, this is a technique adopted by Color Switch [8], where the stars collected in the game can be used to unlock new balls, trails, and colors. Nonetheless, there should be a motivational factor for accumulating points so that we can adopt the variation of the monetisation principle. Then, suppose we design a game unrelated to spelling but enjoyable for children to play, and they are motivated to gather points. In that case, we can encourage them to **make spelling attempts to multiply their points gathered in the game**. This would likely be a lot more enjoyable for children instead of constantly prompting them to spell a variety of words.

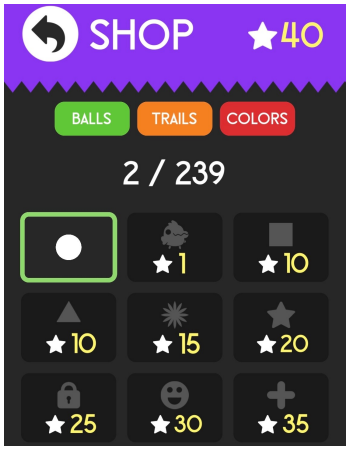


Figure 3.8: Screenshot of *Color Switch* Shop

However, according to von Ahn and Dabbish [2], the next most important aspect of GWAP is to **ensure the output is accurate**, i.e., adopt mechanisms to reduce the noise in the data collected. For example, in the ESP Game, the *output-agreement* method was used to validate the user input, and similarly, the *input-agreement* method was used in TagATune. However, Sound-It-Out Spell-Checking Game differs significantly from these examples as it would be too difficult for two children to agree on a spelling. Also, a multiplayer game poses an additional risk for children. Therefore, we have opted to develop a single-player game, and consequently, it will need a different mechanism for validating if the input is genuine. We will use a **phonemic edit-distance calculator**, provided by the creators of *S-Capade* [32]. O'Neill et al. [32] outline that the phonemic edit-distance is

calculated by "using a weighted edit distance algorithm akin to that of Wagner and Fischer [and then,] the cost for performing a substitution operation was defined as the distance between the two phonemes per the distance matrix" (p. 88). Using this, we will **set a threshold for the acceptable distance from the inputted word to the target word** and only accept the input if the phonemic distance between it and the target is below this threshold. For example, if the target word is "give", we may accept "gif", but not "bib".

	Character-level	S-capade
Misspelling	s i c h w e s h e n	S IH CH W EH SH AH N
Real-Word Target	s i t u a t i o n	S IH CH UW EY SH AH N
Edit-Distance	7	1.1

Figure 3.9: Character-level edit-distance vs S-capade's phonemic edit-distance [32]

3.5 Possible Effects of Screen Time on Children

Before the game is deployed, It is essential to consider the possible effects of screen time on children. A large population-based study [48] examined the associations between screen time and a variety of different measures of psychological well-being, such as emotional stability, relationships with caregivers, self-control, diagnosis of mood disorders, and treatment of mental health issues. Moreover, the study includes not only television but modern means of screen time such as electronic gaming, smartphones, tablets, and computers. On average, children aged 2 to 5 had 2.28 hours of screen time a day, children aged 6 to 10 had 2.78 hours, and those aged 11 to 13 had 3.80 hours. Moreover, the study found a strong correlation between the number of screen time hours a day and several measures of psychological well-being. As seen in figure 3.10, the more screen time

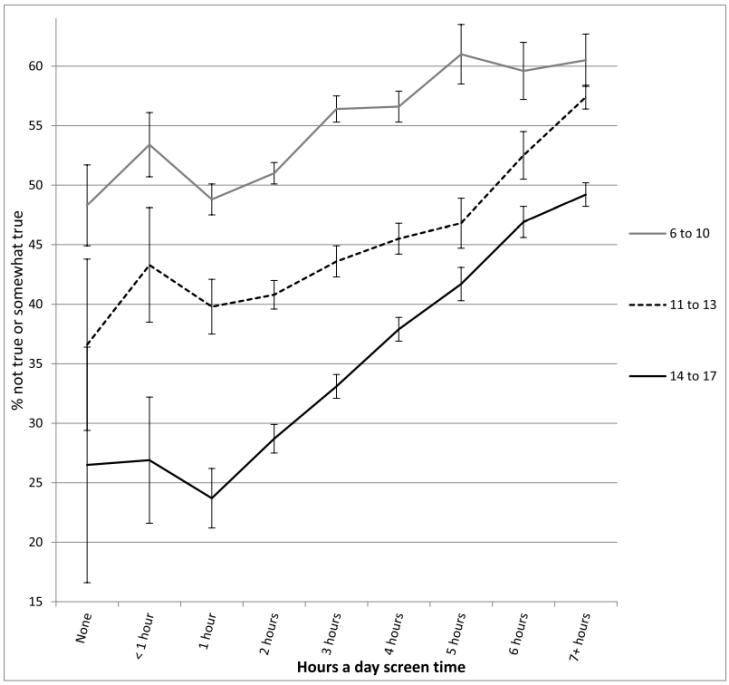


Figure 3.10: Percentage who do not stay calm when challenged [48]

hours a participant had, the more likely they were not to stay calm when challenged. Similarly, the participants that had more screen time tended to argue more with their caregivers, as seen in figure 3.11. In all of the evaluations carried out, including not staying calm, arguing with caregivers,

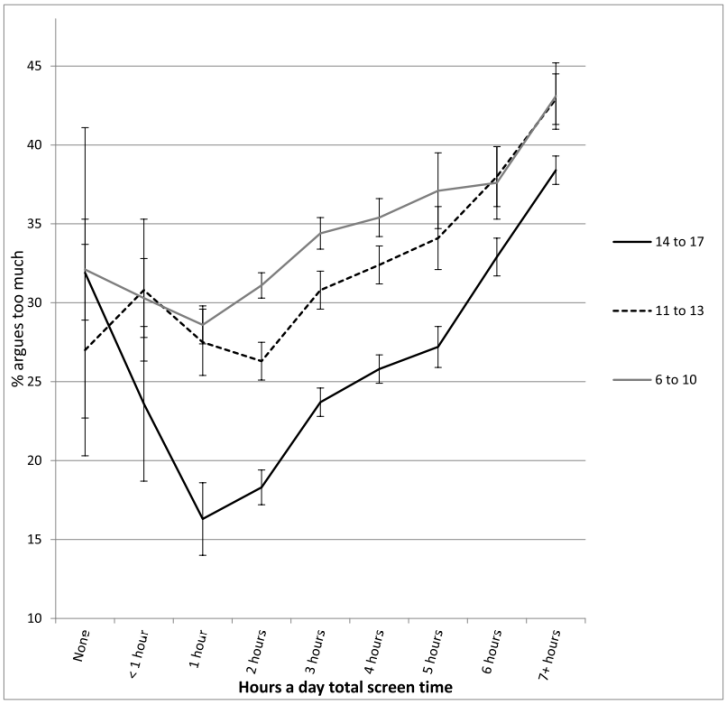


Figure 3.11: Percentage who argue too much with their caregivers [48]

not finishing tasks, and not being curious, up to an hour of daily screen time generally did not negatively affect any of the age groups. On the other hand, participants who had over 7 hours of screen time a day carried twice the risk of low well-being as those who had up to 1 hour.

3.6 Educational Benefits of Video Games

As opposed to the paper discussed in the previous section, Griffiths [17] outlines how children and adolescents can benefit from video games. He suggests that video games can teach children to set goals, ensure goals are rehearsed, and provide feedback. It is also suggested that video games may stimulate learning through elements of interactivity. Moreover, the paper states that video games may **enhance reading skills, basic maths skills, and social skills**. Reading skills may be enhanced by reading messages printed on the screen such as "Play", "Quit", and "Go". Similarly, basic maths skills may be enhanced if children interact with score counters. The paper explains that social skills may also be improved if the game is popular with other children, as they may share the same passion and engage in conversation about the game. For example, if our game had characters that could be unlocked with points, children would likely want to show off their unlocked characters to their peers. Thus, encouraging them to converse with each other. The concept of **unlocking new characters would also teach children to set goals**, and work towards achieving them. On top of this, a game that encourages children to spell would likely also **improve spelling skills**.

3.7 Technologies for Game Development

Finally, before developing the game for this project, we must decide on the technologies to use for the development. There are many technologies for developing games with Java, for example, JavaFX [22], LWJGL [27], and LibGDX [26]. JavaFX is an open source, client application platform for desktop, mobile, and embedded systems, which allows you to develop user interfaces and simple 2D games. However, it is not ideal for more complex games, and the support for mobile applications is limited. LWJGL (Lightweight Java Game Library) is an open source Java library that contains useful APIs for game development. However, it is a low-level technology, and therefore, it is recommended to use a framework that makes use of the library rather than using the library directly. LibGDX is an open source framework that makes use of LWJGL. The framework enables the development of both desktop and mobile games and is very fast. Although LibGDX seems to be the best candidate for developing a Java application, Unity [50] is a game engine for developing 2D and 3D games in the C# programming language, which is very similar to Java. The Unity game engine is rich in documentation, easy to use, and offers high-quality rendering for 2D games. Unity also has a large store for game assets [49], i.e, character sprites, backgrounds, and audio files. It is also easy to import these assets into a Unity game and build the scene. Testing a mobile application is also straightforward in Unity, as you can link your device directly and test as you develop the game. Due to these benefits, we will use Unity for the development of our game.

Chapter 4: Outline of Approach

This chapter outlines the considered approach to this project, specifically how and why each design choice was made. Chapter 6 outlines how each of the discussed elements in this chapter were implemented. As the core of this project is to devise a game with the purpose of gathering data on how children living in Ireland spell, the application needs to have both a carefully designed frontend and backend. It has already been discussed that the game will be a mobile application developed in Unity, but the specific details still need to be explored. This includes what the main game will entail, how the spelling functionality will work, which database management system to use, and where the database will be hosted.

4.1 Main Game Ideas

As previously discussed in section 3.4, there are multiple essential factors to consider when designing a GWAP (Game With A Purpose) for children. Most importantly, the game must be enjoyable as children must be motivated to keep playing the game for free. It was already decided in section 3.4 that this can be done by creating a game unrelated to spelling, where players are awarded points for overcoming some child-friendly obstacles and can use their points to unlock new characters. Although the main game will be unrelated to spelling, the player should be prompted to make spelling attempts for bonus points at the end of the game.

Since the game will rely on the players' motivation to earn points to unlock new characters, it is crucial that the characters appeal to children. Initially, the ideas that came to mind were mythical creatures, such as dragons, mermaids, and unicorns, as these are typically popular with children. However, playing as a mythical creature would mean that the game has little educational value. This led to the idea that the game should be somewhat educational whilst also being enjoyable. Therefore, it was decided that a more suitable approach would be to have the characters as a variety of child-friendly cartoon fish and the obstacles as plastic bottles. This would indirectly showcase the issue of 8 million tonnes of plastic ending up in the ocean each year [36].

This means that the player will play as a fish, plastic bottles will randomly spawn in front of the player, and the game will end once the player collides with a bottle. The player should start with one unlocked fish character and then have the option to unlock more fish that they can switch to use as their character. As mentioned, there should be a point system in place, and the player's accumulated points should be used to unlock new characters. A possible points system would be to have the player collect items, such as coins or stars, where each item is a point. However, considering this game should be suitable for children of a large age range, this may be too difficult for the younger children. Therefore, it was decided that the game will be an endless runner, i.e., the fish keeps swimming until it collides with a plastic bottle, where a point is earned every second that the game is running. To ensure that the game is not too difficult, the fish will continuously move forward, and the player's goal will be to exclusively move up and down, avoiding any plastic bottles that appear in the way.

4.2 Spelling Game for Bonus Points

Since this game aims to gather data on how children living in Ireland spell, the players should be prompted to make spelling attempts for bonus points at the end of the main game, i.e., after the player collides with an obstacle. As mentioned in section 3.4, players in popular mobile games, such as *Fruit Ninja* [13] and *Color Switch* [8], are often given the option to watch an advertisement at the end of the game to earn bonus points. This sparked the idea of having a spelling game/quiz for bonus points after the main game ends. Therefore, the approach of this project relies on the fact that the main game will be sufficiently enjoyable and that the motivation to earn points will be strong enough to make a good attempt at the spelling game. Although it is expected that children will mostly enjoy the main game and only play the spelling game as a necessity at the end, some children may wish to go directly to the spelling game to earn points. This would be ideal as more data would then be gathered. Therefore, there should also be an option to go directly to the spelling game without playing the main game.

4.2.1 Dictating Words With Text-To-Speech Plugin

The first aspect of the spelling game that needs to be decided is how the players will be asked each target word. The full words cannot be shown on screen as it would give away the correct spelling. A possibility would be to show an image and ask the children to label it, e.g., a picture of a house and prompt the player to type what they see. However, this would require an image for every target word that we wish to include. This restricts the target words that can be included as not every word has an associated visualisation. Children may also simply be unsure of what they see in the shown image. Moreover, if we wanted to add more target words in the future, there would have to be an image created for each, which would be quite a slow process.

This led to the decision that each target word should be dictated to the player with audio instead. Therefore, the target words are not restricted as any word can be asked, even if it does not have an associated image. However, recording a human voice for each target word would also be a slow process as data should be gathered on a large set of target words. Therefore, to ensure that it is a quick and easy process to add target words to the application, an off-the-shelf TTS (Text-To-Speech) plugin should be used instead. This means that any string can be dictated to the user, making the application highly scalable.

4.2.2 Target Words Dataset

Since TTS will be used for the spelling game, the game should select each target word from a text file, where each line is a separate word. However, it must be considered that not all children will be of the same age group and, therefore, will have different spelling abilities. As mentioned in section 3.2, the application must only store non-personal data to ensure compliance with GDPR. This means that date of birth cannot be stored, but age alone can be [53]. Therefore, when the application is loaded for the first time, the player should be prompted to select their age group, and there should be a separate text file of target words for each age group to ensure the words are of a suitable difficulty.

Consequently, it was decided to use the word lists provided by *ReadingRockets* [45] as the target words for the game. This is because the words are said to account for 80% of the vocabulary of children in American elementary schools. Since the game is intended to gather data on how children living in Ireland spell, any American spellings, such as "favorite", will need to be changed to the Irish version. Moreover, the words are separated into five lists of increasing difficulty; Grade

1 - Grade 5. This means that we should have five age groups and choose each target word based on the player's age group. Since the lists are separated based on the grade, it makes sense to have the age groups based on each grade's age. Therefore, the age groups should be; 0-5, 6-7, 8, 9, and 10+ [21], and each should have an associated word list (lists should be numbered 0-4, where list 0 is for the 0-5 group and list 4 is for the 10+ group).

4.2.3 Target Word Selection

Initially, it was thought to be a good idea to only ask a player target words from the word list associated with their age. However, this would mean that for the older age groups, data would only be gathered for a small subset of their entire vocabulary. For example, the vocabulary of a player who is 9 years old should include the words from the Grade 1 - Grade 4 lists (not exclusively the Grade 4 list). Therefore, it would be more suitable to randomly select a word from any of the lists associated with age groups younger than the player, as well as the list for the player's own age group.

However, this would mean that there will be significantly more data gathered for the easier words, as more age groups could be asked to spell them. For example, words from list 0 will be in the vocabulary of every player, whereas words from list 4 will be only in players that are 10+. If there is an equal probability of selecting a word from any of the lists in a player's vocabulary, the words from the easier lists will be asked more often than the others. To overcome this, we should prioritise the harder words to ensure sufficient data is collected for all of the word lists. For example, a 10-year-old player should be asked words from any of the word lists, but there should not be an equal probability of selecting from each list. Words from list 4 should have the highest probability, and the probability should decrease for each remaining list.

4.2.4 Repeat and Skip Functionality

As the game relies on TTS technology to dictate each word, the player may mishear a word and need to hear it again. Mishearing the word needs to be considered when designing the game, as only hearing the word once may frustrate the player. Consequently, the game should include a button to repeat a word. If the player clicks the repeat button, it can be assumed that they did not hear the word properly the first time. Therefore, if the spelling game is timed, adding more time to the player's timer would ensure they have sufficient time to process what they have heard and sound it out before attempting the spelling.

However, it should be considered that players may then abuse the repeat button for extra time instead of for the genuine need to hear the word again. This can be overcome by limiting the number of times the repeat button can be clicked for each word. If the repeat button adds 3 seconds to the timer and is limited to 3 clicks per word, this would allow for a maximum of 9 extra seconds per word, which would be reasonable. This is because this project aims to gather data on how children spell, so spending an extra 9 seconds per word will not matter if it leads to the child inputting a genuine spelling attempt. However, the number of repeats should be recorded along with each attempt as it may indicate that the child struggled to spell the word.

If the repeat button is limited to 3 clicks per word, this leads to the question of what would happen if the player still does not hear the word clearly after the three repeats. This can be overcome by including a button to skip the word and get a different word to spell. However, it should be considered that a player may keep skipping words until they hear an easy word. This poses the risk of players picking and choosing the easy words and may lead to insufficient data being gathered on the more challenging words. To avoid allowing the player to only choose easy words, the skip

button should be limited to 2 skips per game.

4.2.5 Points Mechanism

Finally, it is important to decide how points will be awarded for each spelling attempt. Naturally, points should be awarded if the target word is spelt correctly, and the player should be given another target word. However, considering that the goal is to encourage children to use the "sound-it-out" method and gather data on how they spell, it makes sense to award partial points for a genuine attempt at using the "sound-it-out" method to spell a word. This means that the application should check if the inputted word is phonetically similar to the target word, and if it is, award partial points and ask the next word. This can be done by checking the phonemic edit-distance between the inputted word and the target word and accepting attempts that have this distance below a certain threshold. In turn, players will likely be encouraged to keep sounding out the words when making attempts. It is unrealistic to expect children to spell every word correctly (and it is not the goal of this project), so accepting phonetically similar words allows for more data to be gathered as more words will be asked each game. It would be unethical to reward children for spelling words incorrectly without highlighting their mistakes, even if they are phonetically similar. This could reinforce their misspellings by making them believe they are spelling the words correctly. Although reasonable attempts should be accepted, a clear message should inform the player of the correct spelling.

4.3 Relational versus Non-Relational Databases

As the primary goal of this project is to gather data, it is crucial to design the database carefully. It must first be decided whether to use a relational or non-relational database as every database is typically one of these types [37]. Data in a relational database is organised in tables, and these tables may have relationships with one another. In contrast, data in a non-relational database is stored as key-value pairs, graphs, or documents, and relationships are not defined. The advantages of each architecture over the other must be explored to decide on which to use.

Advantages of Relational Databases over Non-Relational:

- Compliant with ACID (Atomicity, Consistency, Isolation, Durability) [40]. Reliability and integrity are higher in ACID-compliant databases.
- Use a standard query language known as SQL (Structured Query Language) [40]. This is a powerful, widely-accepted query language that can allow users to execute complex queries easily.
- Data is stored in a highly normalised, structured, and logical manner using tables [40]. This eliminates data redundancy and ensures the ease of analysing the data.

Advantages of Non-Relational Databases over Relational:

- Use horizontal scaling, whereas relational databases use vertical scaling [40]. Horizontal scaling is much cheaper and more efficient and, therefore, is more suitable for big data projects.

- More flexible as relational databases have a pre-defined schema that is difficult to alter and only supports structured data, whereas NoSQL databases have a dynamic schema that allows data to evolve rapidly [40].
- Data is replicated across multiple server clusters to prevent data loss [40].

After exploring the advantages of both types of databases, it is clear that a relational database would be more suitable for this project. Although NoSQL databases are more scalable and flexible, there is less structure to the data, no standard query language, and data integrity is not prioritised. Therefore, NoSQL databases are ideal for projects with unstructured and unrelated large data that is rapidly evolving. In contrast, the data produced from this project will be highly structured, related, and relatively small as we are solely interested in storing data related to the users' spelling attempts, and the users will only be children living in Ireland.

4.4 Choosing a Database Management System

A database management system consists of a database and software to utilise and maintain the database [40]. It has been decided to use a relational database, but there are many relational database management systems to choose from, such as Oracle, MySQL, Microsoft SQL Server, and PostgreSQL [37]. As seen in figure 4.1, Oracle and MySQL are the most popular database management systems worldwide by far. Due to their popularity, there are many online resources available. The availability of resources allows developers that may begin working on the project in the future to quickly learn how to use the systems or would likely be familiar with them already. However, Oracle requires a paid license and is intended for commercial use only, whereas MySQL is an open-source service that provides an immense amount of services for free [54]. Therefore, MySQL is the most suitable for this project and will be used as the database management system.

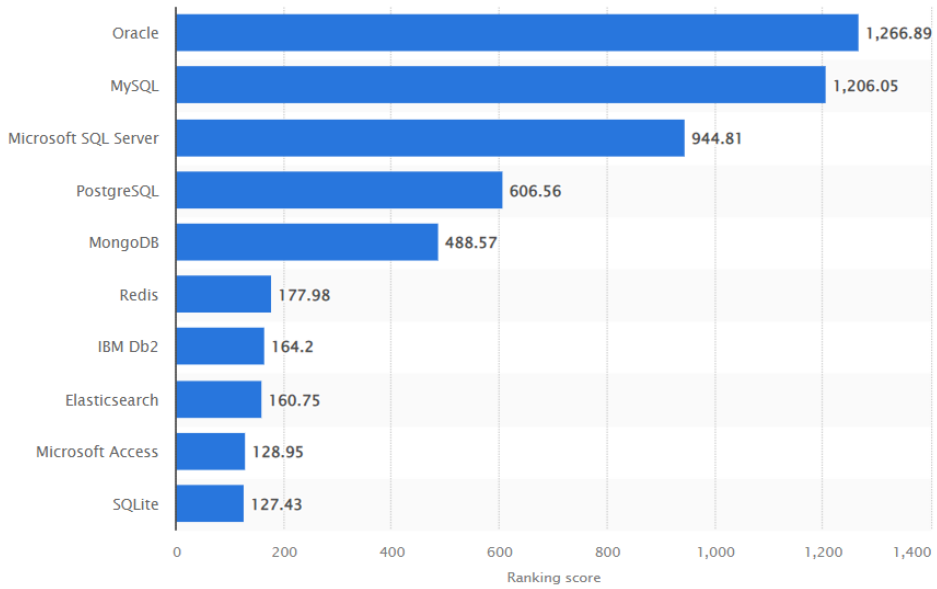


Figure 4.1: Most Popular Database Management Systems Worldwide in 2022 [30]

4.5 Hosting the Database on the Cloud

Finally, where to host the MySQL database should be decided. Naturally, a connection to the database will only be possible if the computer or server running the database is up and running. Therefore, it would not be ideal to have the database running on a local, personal computer as this would mean that the PC would have to be constantly running, which is unrealistic. Since the primary purpose of the application is to gather data, it is crucial to ensure that the application can establish a connection to the database at all times. Otherwise, no data would be sent to the database when a user plays the game, so valuable data may not be collected. Therefore, the database should be hosted via a cloud provider, such as Amazon Web Services (AWS). AWS are currently the leading cloud service provider as they offer hundreds of service, including Elastic Compute Cloud [6] which allows users to rent virtual machines. This is ideal for the application as the database can be left running continuously on a rented virtual machine. Moreover, AWS also offer many geographic regions, including Ireland, which is ideal for the project as hosting the database in Ireland means connections will be made quicker, and the data will be stored in compliance with GDPR.

Chapter 5: Project Workplan

The Gantt Chart below outlines how this project will be approached. Thus far, we have already completed the "Research and Plan" section. The next step is to gather the necessary game assets, such as character sprites, scene backgrounds, and audio files. Then, we will need to put these assets together in Photoshop to design the UI for the application. It is important to first design the UI in Photoshop to have a concrete idea about how to approach the development of the application. In February, we will begin the development by firstly developing the start screen, i.e., a screen with buttons "Play", "Characters", and possibly some other options. Then, we will develop the main game, spelling functionality, and 'unlock new characters' functionality. We will develop the database in parallel with this. It should only store non-personal data, such as age range, spelling attempts, high scores, and unlocked characters. The plan is to test the application in parallel with the development and fix bugs as they emerge. Finally, we will devise an evaluation plan for the project, which the research group will undergo after obtaining ethical approval. Perhaps, they could have a sample of children of varying ages play the game and answer a short questionnaire about their experience. However, testing the game (both manually and with NUnits) will be the primary evaluation for this project. The research group will carry out any further evaluations after the game is deployed. This is because ethical approval will need to be sought and obtained before children can play the game, which may take a while.

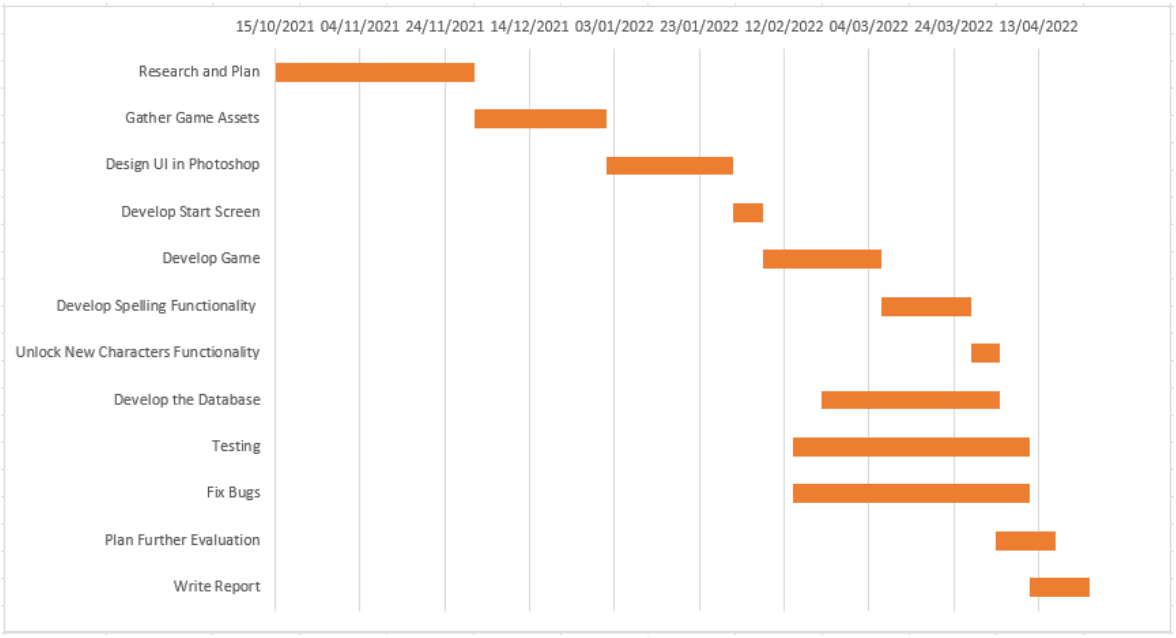


Figure 5.1: Plan

Chapter 6: **Implementation**

This chapter outlines how each development stage of the plan discussed in chapter 5 was implemented. This includes gathering the game assets, creating the Photoshop designs, and developing the start screen, main game, spelling game, characters shop, and database.

6.1 Game Assets and Photoshop Design

As per the Gantt Chart in 5.1, the first planned steps were to gather the necessary game assets and design the UI in Photoshop using the chosen assets. The gathered game assets included character sprites, an obstacle graphic, scene backgrounds, and UI graphics. A significant amount of time was dedicated to these tasks as the game assets are crucial to the game's visual aesthetics and the Photoshop designs provided a concrete plan on how to approach the development. Although the final version of the application slightly differs from the Photoshop designs (found on the GitLab repository), this was to be expected as the initial designs were solely a plan on how to begin the application. However, naturally, the designs evolved over the course of the game's development.

Popular asset stores, such as *Unity Asset Store* [49], *GraphicRiver* [15], and *GameDev Market* [14] were thoroughly searched for appropriate assets. When choosing the assets, it was considered whether children would find them appealing, whether each of the chosen assets would complement each other, and whether they were suitable for young children. It was important to consider whether the assets would complement each other as some assets may have been a good standalone choice but would have looked out of place when placed with the other assets. For example, it was considered to use [3] as the characters, but it was later decided that they were not sufficiently exciting and they did not suit the rest of the chosen assets well. Instead, it was decided that the characters seen in figure 6.1 would be a better choice as each fish is unique, and therefore, children will likely be motivated to unlock more of the characters. The figure below also shows the plastic bottle graphic that has been chosen as the obstacle for the game and some letters which can be used as decorations in the game to reinstate that it is a spelling game. The plastic bottle graphic was chosen as it is not too realistic-looking and complements the cartoonish characters and these letters were chosen as they fit the oceanic, child-friendly theme. A complete list of all the game assets used in the application can be found in section 9.1 of the appendix.

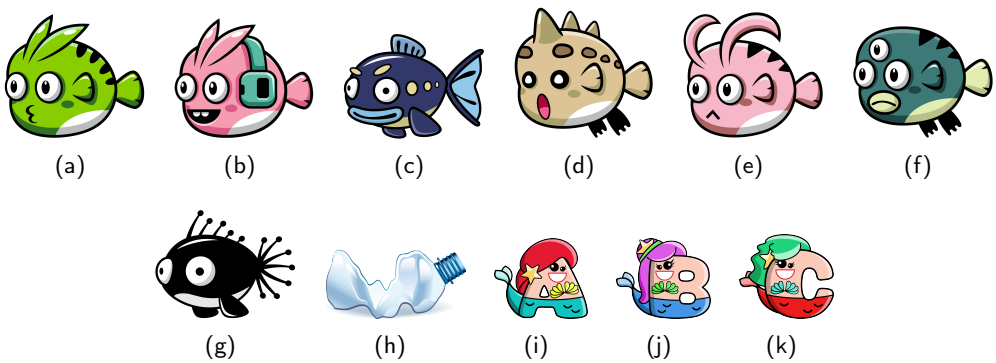


Figure 6.1: (a-g) are Characters, (h) is the Obstacle & (i-k) are Decorations

6.2 Start Screen and Age Selector

After the game assets were gathered and Photoshop designs were made, the next step was to develop the application's start screen (main menu). The purpose of the start screen is to allow the user to navigate to any of the application's components, such as the main game and the characters shop. As mentioned in section 4.2, there should also be an option to go directly to the spelling game without playing the main game. Therefore, the screen was designed to consist of the buttons *Play*, *Spell*, and *Characters*, which can be seen in section 9.3 of the appendix.

In Unity, a scene is where you can place content such as characters, obstacles, and UI, and each application is made up of at least one scene [44]. The start screen was the first scene that was created. The three buttons, game title, background, and some decorative letters were then added to the scene. The decorative letters were added as they reinstate the fact that the application is about spelling words, and they also fit the oceanic, child-friendly theme of the game. To add the functionality to the buttons, the scenes *MainGame*, *Spell*, and *Characters* were created, and a C# script containing a method for each button was developed. Each method was set as its corresponding button's "On Click" action in order to get executed when the button is clicked. The methods each call the built-in Unity function "LoadScene" to navigate to the relevant scene. For example, the code in the *Play* button's method is as follows:

```
SceneManager.LoadScene("MainGame");
```

Moreover, a pop-up to prompt the player to select their age group was also implemented as this was planned in section 4.2. The pop-up was made to only ever appear the first time that the application is loaded on a device. This was done by checking whether the player's age was set. If the age was not set already, the pop-up was activated when the application was opened. The player's parent/guardian must first select the consent box to indicate that the child has permission to play the game, and then the buttons for selecting the age group are enabled. Once a button for the age is pressed, the age is saved in "PlayerPrefs", which is a built-in Unity class that allows data to be saved between game sessions. Therefore, even after the application is terminated, the player's age will still be loaded once the application is started again. The player is also assigned a unique ID, and their ID and age are sent to the database (further discussed in section 6.6).

6.3 Main Game

This section outlines how the main game was implemented. This includes player animations, player movements, a looping parallax background, scoring functionality, and obstacle collisions. Screenshots of live gameplay of the main game can be found in section 9.4 of the appendix.

6.3.1 Player Animations

The first step was to add the player (fish) object to the scene. The player's position was set to be relative to the screen's width to ensure that the spawn position was the same on all devices. Next, animations were created for each character to add to the game's visual appeal. Animating the characters is essential as it is more exciting than having static images. Therefore, it would likely increase the children's motivation to unlock more characters. Each of the purchased character game assets included a series of images for creating a swimming animation. These were used in Unity's animation tool to create a seamless animation of each character swimming. The animation

of the chosen character was set to play once the game started.

6.3.2 Player Movements and Parallax Background

It was decided in section 4.1 that the fish should continuously swim forward, and the player should move up and down exclusively. To implement the continuous forward movement of the fish, a script (*CameraMovement.cs*) was developed, which made the camera move infinitely in an eastwards direction along the x-axis. The fish (player) was programmed to move along the x-axis at the same speed as the camera. Although the fish was now swimming forwards, it was hard to visualise until the background was added. The background is made up of five individual layers, which allows it to be given a parallax effect. This was done by developing a script (*LoopingBackground.cs*) that moved the back layers slower than the foreground, giving it a parallax effect that added depth to the game. This script also ensured that the background was continuously looped, allowing the game to continue endlessly until the player collided with an obstacle. The background was made to move in the opposite direction of the camera and player, and therefore, the player now appeared to be moving forwards.

Next, the functionality to allow the player to move up and down (along the y-axis) was added. This was done by creating a method that is called every frame of the game to check whether the screen has been touched or if a finger has been lifted off the screen. While the player was touching the screen, the fish was made to move up or down depending on whether the screen's top or bottom half was touched. To decide which half of the screen was pressed, it was checked whether the y-position of the touch was above or below half of the screen's height. Once the finger was lifted off the screen, the player's velocity was set to zero to stop moving vertically. Moreover, borders were added at the top and bottom of the screen to stop the player from moving outside the visible screen area.

6.3.3 Scoring Mechanism

As discussed in section 4.1, the player should earn one point for every second that the game is running to ensure that the game is not too difficult for the younger age groups. Once the player collides with an obstacle (plastic bottle), the score should stop increasing and be saved. The score needs to be saved so that it can later be added to the bonus score earned from the spelling game. The scoring functionality was implemented by creating a score counter that appears at the top of the screen, as seen in figure 6.2. The background bubble, star, and text were all added as separate game objects and positioned to match the planned Photoshop design. A script (*MainScore.cs*) was then created and attached to the score counter. The algorithm (which is called once per frame) in the script is as follows:

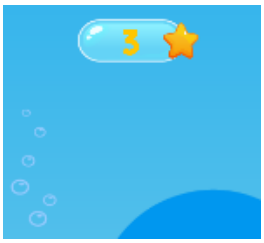


Figure 6.2: Snippet of Score Counter

1: $scoreText \leftarrow$ text of score counter
2: $score \leftarrow 0$
3: **if** $playerObject$ is not null **then** ▷ $playerObject$ will be destroyed when collision happens
4: $score \leftarrow score + deltaTime$ ▷ $deltaTime$ = seconds from last frame to current one
5: $scoreText \leftarrow$ Round $score$ to int and convert to string
6: **else**
7: Save $score$

8: **end if**

6.3.4 Obstacles and Game Over Pop-Up

The last step was to make plastic bottles (obstacles) spawn randomly throughout the game. The following method was created for this task (in script *SpawnObstacle.cs*):

1: $x \leftarrow screenWidth + 10$
2: $y \leftarrow$ Random Value Between 0 and $screenHeight$
3: Instantiate Obstacle at (x, y)

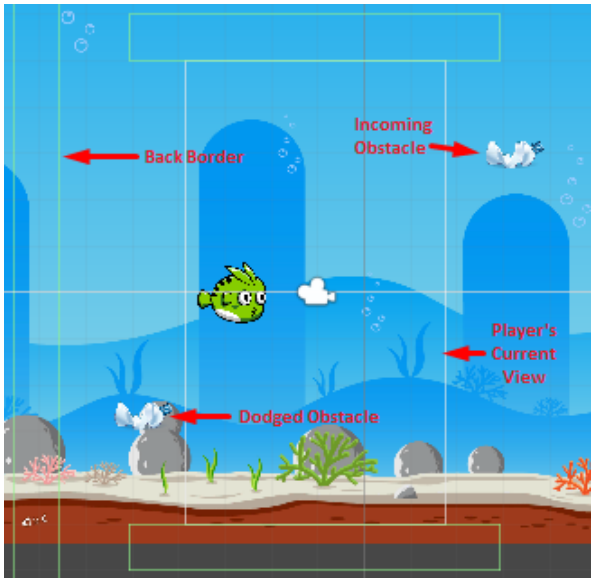


Figure 6.3: Developer's View of *MainGame*

The x-coordinate of the obstacle is the screen's width plus 10 to make the obstacle spawn in front of the current camera view, as seen in figure 6.3. This ensures that the player does not see the obstacle spawning. Therefore, when the player swims across to where the obstacle is, it appears to have been there the entire time. Moreover, the y-coordinate is a randomly generated number between 0 and the screen's height, which ensures that the obstacles are spawned randomly, and the player must move up and down to dodge them. This method is invoked every 2 seconds as it was found after thorough manual testing that this was the most appropriate rate at which to spawn the obstacles. The dodged obstacles are destroyed after they are no longer in the player's view to ensure that the game does not slow down if it proceeds for a long time. This was done by adding a back border and destroying the obstacle object once it collided with it.

Similarly, if there was a collision between the obstacle and the player, the player object was destroyed. Therefore, the stopping condition for the game is if the player object is null. This condition was used to stop increasing the score and trigger a pop-up which alerted the player that the game was over and that they had to move on to the spelling game for bonus points. As seen in section 9.4 of the appendix, the pop-up was made to include a button that the user must click to begin the spelling game. This button loads the *Spell* scene, which will be discussed in the next section.

6.4 Spelling Game

This section outlines how the spelling game was implemented. This includes the functionality to type words, using a TTS plugin to dictate words, checking the spelling attempt, timer to end the game, selecting a random word, skip and repeat buttons, and going straight to spell. Screenshots of live gameplay of the spelling game can be found in section 9.5 of the appendix.

6.4.1 Functionality to Type Words

Initially, the typing functionality was implemented by allowing the player to use their device's built-in keyboard to type their spelling attempt into an input field. However, it was found that Unity cannot disable the autocorrect functionality that comes with Android devices. Allowing the user to type words with autocorrect would ultimately defeat the purpose of the spelling game. Therefore, individual buttons were made for every letter of the alphabet to avoid using the built-in keyboard. A button for backspaces was also created to allow players to fix their typing errors. The input field was changed to be plain text that gets updated using the buttons. Two methods were created to do this - one is called when a letter is pressed, and the other is called when the backspace is pressed. The pseudocode for the two methods can be seen below:

```
1: OnLetterPressed(string letter):                    ▷ letter param is set by the pressed button
2:     typedWordText = typedWordText + letter          ▷ Add the letter to end of text
3:     Deactivate placeholderText                      ▷ Hide the "Type Below..." text
4: OnBackspacePressed():
5:     if typedWordText.length > 0 then
6:         typedWordText ← typedWordText.Substring(0, typedWordText.length − 1)
7:     end if
```

These methods are found in *Keyboard.cs*, along with another method that is called once per frame to check if the typedWordText's length is 0. If the length is 0, the method activates the placeholder text ("Type below..." text).

6.4.2 TTS Plugin to Dictate Strings

An open-source text-to-speech (TTS) plugin for Unity [46] was used to dictate each target word to the player. This plugin uses the native android and iOS TTS functionality to allow strings to be dictated in a mobile Unity application. This plugin was integrated by cloning the repository and importing the *TextToSpeech.cs* and *TextToSpeech.prefab* files in our Unity project. A prefab in Unity is a pre-configured, reusable GameObject [43], so this prefab was pre-configured to have the *TextToSpeech.cs* script attached to it already. Therefore, the prefab was added to the *Spell* scene and was ready to be used by the scene's other scripts.

Most of the logic for the spelling game was written in a script called *SpellController.cs*, including initialising the TTS plugin. On starting the game, the TTS plugin was initialised to use English and speak at a rate of 0.8 as it was found through testing that it was difficult to hear some words if the rate was any faster. The TTS plugin contains a method called *StartSpeak(string wordToSpeak)*, which speaks the *wordToSpeak* string. This method was called anytime the target word needed to be dictated, i.e., when the player gets a new target word or clicks the button to repeat the current word. The plugin was initially tested by hardcoding the *wordToSpeak* but was later updated to use the randomly selected target words.

6.4.3 Spellcheck Functionality

A button was added to allow the player to submit the word they typed. This button was made to send each attempt to the database and check whether the inputted spelling was correct, partially correct (phonetically similar), or incorrect. An appropriate message was displayed depending on each of these conditions, as seen in section 9.5 of the appendix. These messages were "Correct, Well Done! Keep Going", "Good Attempt at Sounding It Out! Correct answer was *Correct-Spelling*", and "INCORRECT: Time to...Sound It Out and Try Again!" respectively. The player's

score was also updated when the button was pressed, receiving 5 points for a correct answer, and 2 points for partially correct.

Checking whether the spelling was correct was easily done by checking whether the typed word and the target word were equal after converting both to lowercase to ensure the comparison is accurate. However, checking the phonetic distance between them posed some difficulty. This is because initially, the idea was to use S-Capade's phonemic edit-distance calculator, but it is written in Python and uses a tool [41] that was found to be unable to run on Android applications. It was also found during this phase of the implementation process that since Unity's primary language is C#, running Python scripts from within the application requires third-party plugins making it very difficult and unstable. Therefore, a different method for calculating the phonetic distances was used. However, S-Capade was still used offline in section 7.2 to demonstrate that it can use the data collected by the application.

It was decided to use the Soundex() function [11] that is built into MySQL to calculate the phonetic distance instead. This is because the application uses a MySQL database, making the Soundex function readily available to the application. Whenever a player pressed the submit button, the spelling attempt and target word were sent to the database, and the Soundex function was used to calculate the Soundex codes of the two words and send them back to the game. Then, the game calculates the difference between the two Soundex codes and stores this as the phonetic distance. If the difference between the Soundex codes was found to be 0, then the two words were pronounced the same, e.g., "party" and "parti" (both have Soundex code P630). Coroutines are used when communicating with the server to allow the request to execute concurrently with the rest of the application. The request for the phonetic distance was made to time out after 4s as the server may be down, so we cannot have the player indefinitely waiting for a response. Therefore, whenever the server is down, the player will only be awarded points if they spell a word correctly. This ensures that the application is fault-tolerant, as the game can still be played when the server is down. The algorithm for the spellcheck functionality is as follows:

```
1: CheckSpelling():                                   ▷ Called when submit btn pressed
2:     Send attempt to server as Couroutine           ▷ Request times out after 4s if no response
3:     Start CheckSpellingAfterPhoneticDistance() as Coroutine
4: CheckSpellingAfterPhoneticDistance():
5:     while Server hasn't responded with phonetic distance or timeout do
6:         Activate loading symbol and disable buttons
7:         Wait
8:     end while
9:     Deactivate loading symbol and enable buttons
10:    if phoneticDistance = 0 OR targetWord.Equals(typedWord.ToLower) then
11:        isCorrectSpelling ← targetWord.Equals(typedWord.ToLower)
12:        if isCorrectSpelling then
13:            score ← score + 5
14:            Set message colour to green
15:            Set message text to "correct" message
16:        else
17:            score ← score + 2
18:            Set message colour to orange
19:            Set message text to "good attempt" message
20:        end if
21:        Clear typedWord and enable repeatButton
22:        Get new target word
23:    else
24:        set message colour to red
25:        set message text to "incorrect" message
26:    end if
```

6.4.4 Timer to End Game

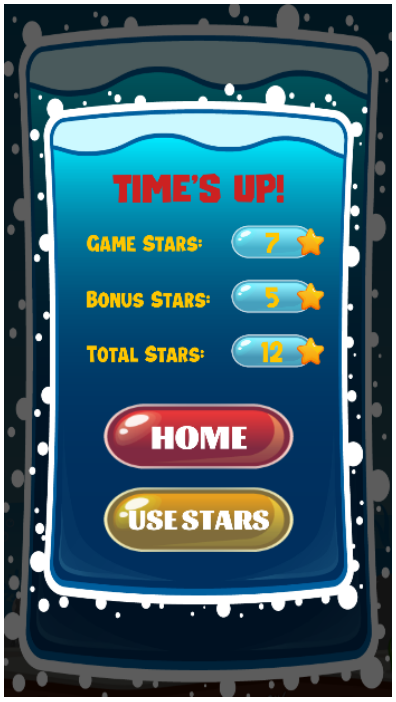


Figure 6.4: Pop-Up After Timer Ends

A timer was needed to end the spelling game. When the timer ends, a pop-up alert was made to appear, as seen in figure 6.4. The player is shown how many "stars" they earned and given the options to go to the start screen ("HOME") or the characters shop ("USE STARS"). The total score is added to the player's accumulated score, and this is saved to the "PlayerPrefs", which is a built-in Unity class that allows data to be saved between game sessions. Therefore, even after the application is terminated, the total accumulated score will still be saved and loaded once the application is started again.

It was decided that 100 seconds should be the time limit for the spelling game as a longer time may cause children to get bored. The timer was implemented by initialising a plain text object as "100" and using a script (*Timer.cs*) to decrement this time every second. The script was also made to pause the timer whenever the loading symbol appears as the player needs to wait for the server's response. Once the loading symbol disappears, the timer continues to decrement. It was also decided that whenever a player asks to repeat a word, the timer should be increased by 3 seconds as they likely require more time to make their spelling attempt. Increasing the timer when the repeat button is pressed ensures that we do not miss out on collecting data from a player who wanted to make a spelling attempt but did not have sufficient time.

6.4.5 Selecting a Random Target Word

The next stage of implementing the spelling game was to select from the *ReadingRockets* [45] lists that we have chosen to use instead of hardcoding target words. The five word lists were added to the project as text files, each containing one word per line. These lists were named targetWords_0.txt, targetWords_1.txt,..,targetWords_4.txt, where the number represents the age group that the words are for. As mentioned in section 4.2, a player should be asked words from their age group's list and any of the younger age groups' lists. However, they should most often be asked words from their age group's list and less often from the easier lists. This was implemented using the following algorithm:

```
1:  InitWordListProbabilities():                    ▷ Called when spelling game starts
2:      switch playerAgeGroup do
3:          case 1
4:              wordListProbabilities ← [0, 1, 1]              ▷ 33% list 0, 67% list 1
5:              break
6:          case 2
7:              wordListProbabilities ← [0, 1, 1, 2, 2, 2]     ▷ 17% list 0, 33% list 1, 50% list 3
8:              break
9:          case 3
10:             wordListProbabilities ← [0, 1, 2, 2, 2, 3, 3, 3, 3, 3]
11:             break
```

```
        case 4
            wordListProbabilities ← [0, 1, 2, 2, 3, 3, 3, 4, 4, 4, 4, 4]
            break
        case Default
            targetWordsList ← targetWords_0.txt   ▷ If no other condition, then group is 0
```

Then, if the player's age group was not 0, the $wordListProbabilities$ list was used every time a new word was needed to be chosen. This was done by randomly selecting an element in $wordListProbabilities$, and then selecting a random word from the list targetWords_$chosenElement$.txt. Naturally, the probability of reading from the easier lists is lower due to to the unequal distribution of the numbers in $wordListProbabilities$. If the player's age group was 0, they could only be asked words from targetWords_0.txt, so there was no need to select a random list each time.

A method called *GetRandomTargetWord()* was created, which was invoked every time a new target word was needed, i.e., when a player's spelling attempt was accepted or when the player skips a word. The method chooses the list to read from (by using the previously explained method) and selects a random word from that list. This is done by randomly generating a number between 0 and the number of words in the list, then reading the word at that line number. The method then checks if the selected word has been already asked during the game session. If the word has been asked, the process of selecting a random word is repeated. When the database was created, the method was also made to start a coroutine to update the number of times the last target word was repeated. Another coroutine was also added to update the database to reflect whether the last target word was skipped. Using coroutines for these tasks ensures that the game runs smoothly as communication with the server happens asynchronously.

6.4.6 Buttons to Skip and Repeat Word

Figure 6.5: Buttons

The buttons that were added to the spelling game can be seen in figure 6.5. The blue speaker icon was used for the repeat button, the red fast forward icon was used for the skip button, and the yellow checkmark icon was used for the submit button. When the repeat button was pressed, the TTS plugin's *StartSpeak(string wordToSpeak)* function was called, and a counter for the number of repeats was incremented. After the player had pressed the repeat button three times, the button was disabled as that is the maximum repeats allowed per word. When the player gets a new target word, the button is enabled, and the repeat counter is set back to 0 as they should be able to request repeats again for the new word.

When the skip button was pressed, a counter for the number of skips was incremented, and *GetRandomTargetWord()* was called. As previously mentioned, this method updates the database to reflect that the word has been skipped. When the skip counter reached 2, the skip button was disabled for the rest of the game as it was previously decided only to allow the player to skip two words per game. Figure 6.5 demonstrates how the skip button appears when it is disabled, but it can be seen in its enabled state in section 9.5 of the appendix.

6.4.7 Directly to Spell Without Playing Main Game

It was decided in section 4.2 that the player should have the option to go directly to the spelling game without playing the main game. Players may wish to play the spelling game to earn more points or practice their spelling. Hence, a button for this was added to the start screen. It would be

ideal for us if players often choose to go directly to the spelling game as it would lead to more data being gathered. Therefore, some changes were implemented to ensure appropriate pop-ups were displayed when the player went directly to the spelling game. For example, the "Game Over" text was removed from the initial pop-up containing the instructions, as the player did not play the main game. A screenshot of the instructions pop-up when the player went directly to the spelling game can be seen in section 9.5 of the appendix.

Moreover, the pop-up that appears at the end containing the scores was modified to only display the score earned from the spelling game, as seen in figure 6.6. This is because the player did not play the main game, so their score is made up of only the score from the spelling game. To check whether the player went straight to the spelling game, it was checked whether the player's score from the main game was greater than 0. This is because the main game cannot end in less than 3 seconds, as this is when the first obstacle appears. Therefore, if the score from the main game is 0, the player must have gone straight to the spelling game.

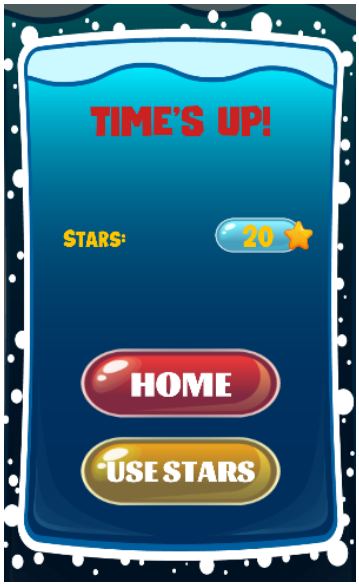


Figure 6.6: Time's Up Pop-Up

6.5 Characters Shop

As per the Gantt Chart in 5.1, the next feature to implement was the functionality to unlock new characters, i.e., the characters shop. This included allowing the player to browse all of the characters, spend their accumulated score to unlock new characters, and switch to play the main game with any of their unlocked characters. Screenshots of the characters shop can be seen in section 9.6 of the appendix.

Firstly, the functionality to browse the characters was implemented. This was done by adding two buttons to view the next character and the other to view the previous one. Before adding the functionality to the buttons, all seven characters' images were added to the scene and placed in the same position. However, all images except the first character's image were initialised to be hidden when the scene loads so that only the first character is displayed initially. Similarly, text for each character's cost and a "SELECTED" text was added to the scene, but all except the first character's cost were hidden. Then, to allow the user to browse to another character, an array containing these seven images was created in a script (*CharacterSelection.cs*). Another array containing each corresponding character's cost was also created. For example, the third element of the costs' array contained the cost of the third element of the characters' array. These arrays were used for the two buttons' methods in the following way:

1: $currCharacterIndex \leftarrow 0$
2: RightBtnPressed(): ▷ Called when right arrow btn pressed
3: **if** $currCharacterIndex < charactersArray.Length - 1$ **then**
4: $currCharacterIndex \leftarrow currCharacterIndex + 1$
5: **else**

```
        currCharacterIndex ← 0
    end if
    NextCharacter()
LeftBtnPressed():                          ▷ Called when left arrow btn pressed
    if currCharacterIndex > 0 then
        currCharacterIndex ← currCharacterIndex − 1
    else
        currCharacterIndex ← charactersArray.Length − 1
    end if
    NextCharacter()
NextCharacter():
    for i in [0, 1, .., 6] do
        if i ≠ currCharacterIndex then
            Deactivate characters[i]
            Deactivate costs[i]
        else
            Activate characters[i]
            Check whether to display cost, select btn, unlock button etc.
        end if
    end for
```

As seen above, if the player pressed the right arrow when they reached the final character, the first character was shown. Similarly, the final character was shown if they pressed the left arrow when viewing the first character.

Next, the functionality to unlock characters was implemented. This was done by checking if the player's accumulated score was greater than or equal to the current active cost price. If it was, then the "UNLOCK" button was set to be clickable. Otherwise, if a character was unaffordable, the button was not clickable. If the player clicks the button to unlock the character, a (key, value) pair ("fish-$currCharacterIndex$", 1) is saved in "PlayerPrefs", and the character's cost is subtracted from the player's total points. The value of the key "fish-$currCharacterIndex$" was used to check whether to display the cost or the "SELECT" button.

Lastly, the functionality to switch between characters was implemented. This was done by adding all of the character objects to the main game's scene but setting them all to be hidden. A (key, value) pair ("selectedCharacter", 0) was added to "PlayerPrefs" to set the character at index 0 to be the initially selected character. Then, a script (*LoadCharacter.cs*) was created to check the value of "selectedCharacter" each time the main game was loaded. The script contained an array of all the game objects and only enabled the one at the index corresponding to the value of "selectedCharacter". Then, a "SELECT" button was displayed in the characters shop when the player navigates to an unlocked character. When the button was clicked, the value of "selectedCharacter" was updated to $currCharacterIndex$. This ensures that the selected character is saved between game sessions and that only the selected character is visible when the game is played.

6.6 Database

This section outlines how the database was implemented. This included the AWS virtual machine, the database server on the virtual machine, the schema of the database (tables, columns, rows, relationships), and the APIs to connect to the database from the application.

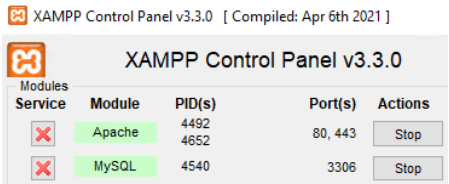
6.6.1 AWS Virtual Machine

It was decided in section 4.5 to host the database on an AWS Virtual Machine (VM) instead of a local personal computer so that it can constantly be up and running. Therefore, a free tier Windows 10 VM was rented on AWS EC2 (Elastic Compute Cloud). The free tier only allows the VM to be running for a limited number of hours before being charged, but this was sufficient as it was only run during testing and the user experience survey. A key pair (public key and private key) was created on AWS, and this was used to connect to the VM using Windows' built-in RDP (Remote Desktop Protocol) application. Moreover, an elastic IPv4 address was created for the VM, which ensures that the IPv4 address of the VM is static, i.e., it does not change after restarting. A screenshot of the VM can be seen below:

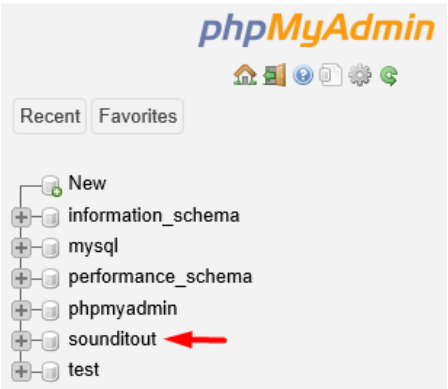
Figure 6.7: Screenshot of the VM Instance on AWS

6.6.2 Database Server using XAMPP

After connecting to the VM, the next step was to create the MySQL database. However, there also needed to be a means of connecting to the database on the VM. Therefore, it was decided to use XAMPP [55] to host the database and the APIs for connecting to the database. This is because XAMPP is a package containing an Apache Web Server, MySQL, and PHP. The package also contains phpMyAdmin [35], which allows MySQL databases to be easily created, queried, and managed. XAMPP was downloaded on the VM, and the Apache Server and MySQL were run. Then, phpMyAdmin was opened by navigating to *http://localhost/phpmyadmin/* on the VM, and a database called "sounditout" was created, as seen below.



(a) Running Apache and MySQL with XAMPP



(b) Created Database with phpMyAdmin

6.6.3 Schema Design

The next step was to design the schema of the database. This included the tables, columns, rows, and relationships between tables as MySQL is a relational database [37, 54]. The schema was planned using *DbDesigner* [9], which is an online tool that allows database schemas to be easily designed. The schema designed using the tool can be seen in figure 6.8. As seen from the figure, it was decided to create a relationship between the asked words table and both the other tables. Therefore, although a direct relationship between users and attempts does not exist, the tables can still be joined through the "asked_words" table, making it possible to retrieve attempts and the age group of the child that made each attempt.

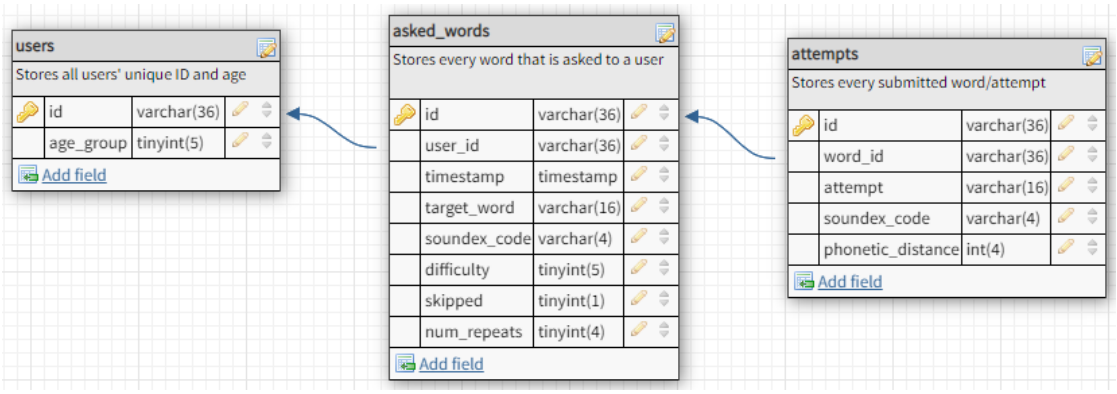


Figure 6.8: Screenshot of Schema Designed Using *DbDesigner*

6.6.4 APIs to Communicate With Database

Lastly, communication between the application and the database had to be established, i.e., send data to the database and retrieve the calculated phonetic distances. This was done by creating six PHP files to register a new user, record each asked word, record each attempt and retrieve the Soundex codes, update the number of repeats, update the number of skips, and update the phonetic distance. Each of the PHP files created a connection to the database and then executed an 'insert', 'update', or 'select' SQL query. The PHP files were hosted on the XAMPP server, allowing them to be requested remotely from the application. For example, the API *http://52.212.158.172/sounditout/RecordAttempt.php* was called to send each attempt to the database and respond with the Soundex code of the attempt and the target word. This was the URI because the public IPv4 address of the AWS VM is 52.212.158.172, and the name of the created PHP file for this task is *RecordAttempt.php*. Moreover, whenever an API was called in the application, it was done using a new coroutine. This ensured that the game did not freeze whilst waiting for a response from the server, as each request was made asynchronously.

Chapter 7: Testing and Evaluation

7.1 Manual Testing

The application was tested in parallel with the development process to catch bugs before they potentially cause significant issues. After all features were implemented, thorough testing was also undertaken, and any further discovered bugs were fixed. It was decided that manual testing was more suitable for testing this application as unit tests would not have caught issues with the text-to-speech synthesizer nor issues with the game's appearance. For example, it was found during testing that the text-to-speech synthesizer was difficult to understand sometimes, which was fixed by slowing down the rate at which it speaks. Moreover, Android devices have a wide range of screen sizes. Therefore, the application was tested using different screen sizes in Unity and also on two separate physical Android devices. Manual testing was ideal for this, as human eyes were needed to ensure that there were no issues with the UI on various screen sizes. For example, it was found that the character spawned in different locations on different screen sizes. This was fixed by using the screen's width to calculate the character's starting coordinates instead of hardcoding them. As the application was implemented over just three months, it was decided not to spend time creating unit tests in order to have sufficient time for thorough manual testing. However, if the application continues to be further developed in the future, it would be good to create unit tests that can be run every time a new feature is added to ensure that it does not break any existing features.

7.2 Using S-Capade with Collected Data

As previously mentioned, the purpose of the game is to gather data that can be used by S-Capade [32]. Therefore, data gathered from the game was fed to S-Capade's phonemic edit-distance calculator to ensure that the data produced by the application can be used with S-Capade. Although this was not planned, it was decided that it was crucial for evaluating the project's success as the gathered data must have the potential of being used by the S-Capade team to test or improve their tool. The calculator takes two words as input and returns the phonemic edit-distance between them. It was found that the calculator requires the two words to be lowercase, so the game was made to convert all attempts and target words to lowercase before storing them. This ensures that any collected data is ready to be used with minimal pre-processing. The following SQL query was used to extract the ten most recent attempts and their corresponding target words to a txt file:

```
1 SELECT asked_words.target_word, attempts.attempt
2 INTO OUTFILE 'attempts.txt'
3 FROM asked_words, attempts
4 WHERE asked_words.id = attempts.word_id
5 ORDER BY asked_words.timestamp DESC
6 LIMIT 10
```

Figure 7.1: SQL Query to Extract 10 Most Recent Attempts and Target Words

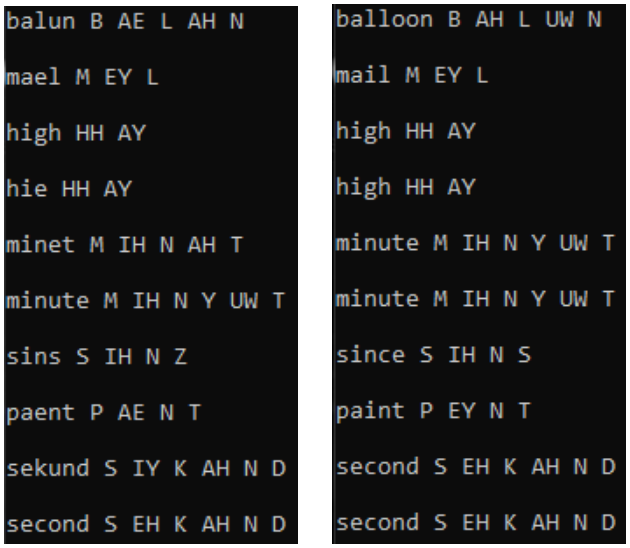
The 'attempts.txt' file containing the attempts and target words columns was copied to the local machine from the server. Then, each of the words were converted to their individual phonemes using the Sequence-To-Sequence Grapheme-To-Phoneme (G2P) [41] tool that S-Capade utilises. This step is necessary as S-Capade's calculator requires each word to be input as a list of phonemes. The G2P tool can convert words in bulk by providing it with a text file containing one word per line [41]. Therefore, the column containing the target words in the 'attempts.txt' file was moved to a separate file called 'targetwords.txt'. Now that the attempts and target words were in two separate files, both were converted to phonemes using the following commands:

Convert Attempts to Phonemes:

```
g2p-seq2seq --decode attempts.txt --model_dir
g2p-seq2seq-model-6.2-cmudict-nostress
```

Convert Target Words to Phonemes:

```
g2p-seq2seq --decode targetwords.txt --model_dir
g2p-seq2seq-model-6.2-cmudict-nostress
```



(a) Converted Attempts (b) Converted Target Words

Figure 7.2: Outputs Produced By G2P [41] Tool

After each word was converted to phonemes, the phonemic edit-distance between each attempt and its corresponding target word was calculated using S-Capade's calculator (the code for the calculator was kindly provided by one of S-Capade's creators, O'Neill). The calculator takes two lists of phonemes as input and returns two arrays. The last value of the first array is the value we are interested in, as it is the calculated phonemic edit-distance between the two inputs. Some examples of the gathered data being used with S-Capade's code can be seen in figure 7.3. This demonstrates that the game has clearly gathered data successfully. Therefore, S-Capade's team can use this data to test or improve their model.

```
phone_comp(['M', 'EY', 'L'], ['M', 'EY', 'L'])
```

```
(array([[0., 1., 2., 3.],
        [1., 0., 1., 2.],
        [2., 1., 0., 1.],
        [3., 2., 1., 0.]]),
 array([[(0, 0, 0), (0, 0, 1), (0, 0, 1), (0, 0, 1)],
        [(1, 0, 0), (0, 1, 0), (0, 0, 1), (0, 0, 1)],
        [(1, 0, 0), (1, 0, 0), (0, 1, 0), (0, 0, 1)],
        [(1, 0, 0), (1, 0, 0), (1, 0, 0), (0, 1, 0)]],
       dtype=[('del', 'i1'), ('sub', 'i1'), ('ins', 'i1')]))
```

(a) Phonetic distance between 'mael' and 'mail'

```
phone_comp(['S', 'IH', 'N', 'Z'], ['S', 'IH', 'N', 'S'])
```

```
(array([[0.        , 1.        , 2.        , 3.        , 4.        ],
        [1.        , 0.        , 1.        , 2.        , 3.        ],
        [2.        , 1.        , 0.        , 0.5       , 1.        ],
        [3.        , 2.        , 0.5       , 0.        , 1.        ],
        [4.        , 3.        , 1.        , 1.        , 0.33601807]]),
 array([[(0, 0, 0), (0, 0, 1), (0, 0, 1), (0, 0, 1), (0, 0, 1)],
        [(1, 0, 0), (0, 1, 0), (0, 0, 1), (0, 0, 1), (0, 1, 1)],
        [(1, 0, 0), (1, 0, 0), (0, 1, 0), (0, 0, 1), (0, 0, 1)],
        [(1, 0, 0), (1, 0, 0), (1, 0, 0), (0, 1, 0), (0, 0, 1)],
        [(1, 0, 0), (1, 0, 0), (1, 0, 0), (1, 0, 0), (0, 1, 0)]],
       dtype=[('del', 'i1'), ('sub', 'i1'), ('ins', 'i1')]))
```

(b) Phonetic distance between 'sins' and 'since'

Figure 7.3: Outputs Produced By S-Capade's Phonemic Edit-Distance Calculator

7.3 User Experience Survey

As discussed in chapter 5, ethical approval will need to be obtained before children can play the game. Although it was initially planned only to test the application and leave further evaluations to be completed when ethical approval is obtained, it was decided that a user experience survey should be completed with adults. This is because it was important to receive unbiased feedback before concluding if the project was successful and to see if any areas needed to be improved in the future. However, since the game is for children, the survey was designed to ask participants to answer questions about how they think a child would find their experience. Therefore, each participant was assigned an age group and was asked to answer questions about how they think a child of their assigned age group would find different aspects of the application.

The survey consisted of a sample of 15 adults that own an Android device. Each participant was assigned an age group and asked to download the application and spend time playing the game. Then, each filled out a Google form consisting of various questions about their experience (the results of the survey can be found here [42]). The questions asked the participants to rate various aspects of the application on a scale of 1 to 5. The average rating for each question can be seen in table 7.3. The majority of questions had an average rating above 4.5, which demonstrates that participants rated most aspects of the application very high. However, the survey showed that some areas of the application should be improved in the future. This includes the typing functionality of the spelling game (average rating was 3.8), the text-to-speech synthesizer (average rating was 4.0), and the loading speed of the application (average rating was 4.13). This will be further discussed in the following chapter.

Question	Avg out of 5
How do you think a child in this age group would find the navigability of the UI?	4.73
How appealing do you think the main game would be to a child of this age group?	4.53
How appropriate did you find the words in the spelling game for a child of this age group?	4.67
How easy did you find the typing functionality of the spelling game?	3.80
How well were you able to understand the text-to-speech?	4.00
How motivated do you think a child would be to unlock more of the characters?	4.53
How satisfied were you with the loading speed of the app?	4.13
Overall, how likely do you see children playing and enjoying the app as a whole?	4.60

Table 7.1: Average Responses in User Experience Survey

Chapter 8: Conclusions and Future Work

In conclusion, *Sound-It-Out Spell-Checking Game* was implemented as a Unity Android Application that communicates with a MySQL database hosted on an AWS virtual machine. The game sends data to the database using PHP every time a spelling attempt is made. Therefore, data on how children living in Ireland spell can be easily gathered by simply allowing children to play the game. The application complies with GDPR guidelines as it only stores non-personal data, such as spelling attempts and user IDs. The project's success was demonstrated by showing an example of how S-Capade can use data produced by the game. Additionally, a user experience survey showed that adults thought that children would be motivated to play the game. The survey was conducted with adults as ethical permission must be obtained before the game is deployed with children. Therefore, the next step for this project is to seek ethical permission so that children can begin playing the game, allowing actual data from children to be gathered.

Moreover, the user experience survey showed room for improvement in some areas of the application. This includes the typing functionality of the spelling game, the text-to-speech synthesizer, and the loading speed of the application. The typing functionality was likely rated lower than other aspects of the game as individual buttons were made for each letter of the keyboard due to the built-in Android keyboard having autocorrect. Therefore, the future work for this project should certainly include improving the keyboard. This could be done by searching various asset stores for a keyboard asset and replacing the individual buttons with a pre-configured keyboard asset.

Although the text-to-speech plugin worked well for dictating most words, it is naturally not as clear as hearing a human voice. The plugin also speaks words with an American accent. As this project aims to gather data from children living in Ireland, it would be better if the words were dictated using an Irish accent. This is because the regional pronunciation of a child often affects how they spell words [31], so dictating the words in an Irish accent would likely provide more accurate data as the child's spelling attempt will not be influenced by the American accent. Therefore, the future work for this project should involve recording a human's voice for each target word and playing the relevant recording for each word instead of using the text-to-speech plugin. Alternatively, a text-to-speech synthesizer that uses an Irish-English voice could be used to allow the children to hear the words in an Irish accent.

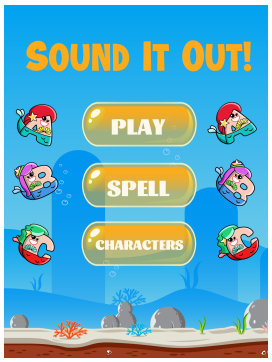
Lastly, the future work should also include improvements to the loading speed of the application. The user experience survey showed that the majority of participants were highly satisfied with the loading speed, but some were not. Likely, some were not as satisfied because the server may have been down when they were playing the game, so they had to wait for four seconds each time they made a spelling attempt, as this is how long it takes for the request for the Soundex codes to timeout. Therefore, the application's server should be moved from the free tier AWS virtual machine to a virtual machine that does not have a limited number of hours. For example, it could be moved to one of the servers in UCD in the future.

Chapter 9: Appendix

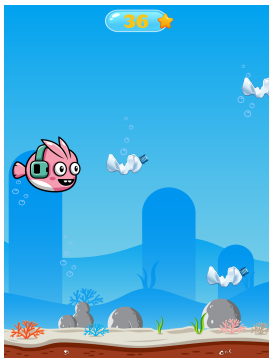
9.1 Purchased Game Assets

- **Characters:** 7 Villains Fish Sprite Sheets by Bevouliin https://graphicriver.net/item/7-villains-fish-sprite-sheets/9258963 (visited on 04/20/2022)
- **Characters:** Fish with Earphone Game Character Sprite Sheets by Bevouliin https://graphicriver.net/item/fish-with-earphone-game-character-sprite-sheets/11652858 (visited on 04/20/2022)
- **Obstacle:** Garbage Icons by Andegro4ka https://graphicriver.net/item/garbage-icons-/7733663 (visited on 04/20/2022)
- **Background:** Below the Sea Game Background by Bevouliin https://graphicriver.net/item/below-the-sea-game-background/19367836 (visited on 04/20/2022)
- **UI Pack:** Ocean Theme GUI Pack 08 by Maykwok0505 https://graphicriver.net/item/ocean-theme-gui-pack-08/27461799 (visited on 04/20/2022)
- **Mermaid Letters:** The Mermaid Alphabet by Partyhead Assets https://assetstore.unity.com/packages/2d/characters/the-mermaid-alphabet-139478 (visited on 04/20/2022)

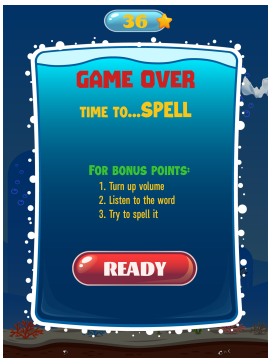
9.2 Photoshop Designs



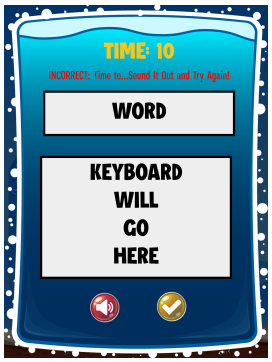
(a) Start Screen



(b) Main Game



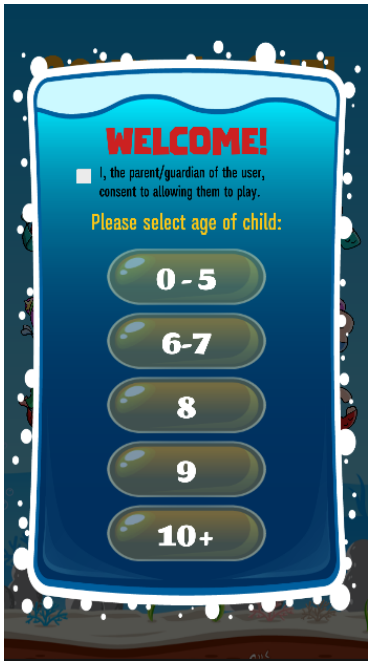
(c) Game Over Pop-Up



(d) Spelling Game

9.3 Start Screen Screenshots

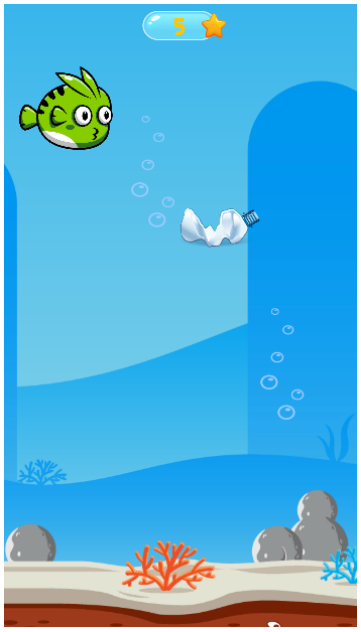
(e) Start Screen



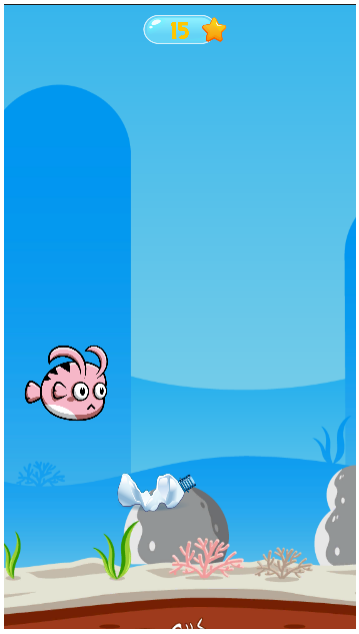
(f) Pop-Up When App is First Opened

(g) Pop-Up After Consent Checkbox is Selected

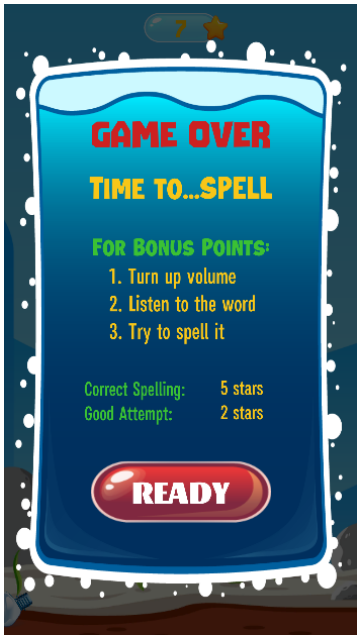
9.4 Main Game Screenshots



(h) Live Gameplay

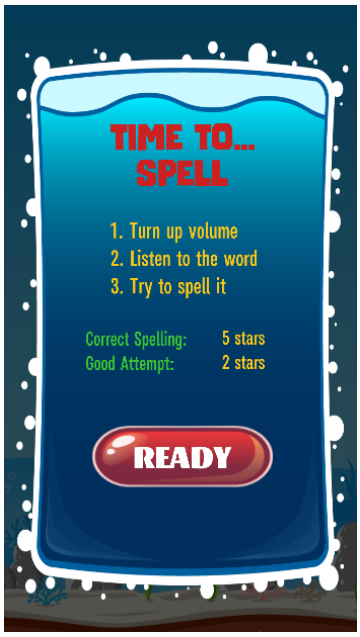


(i) Live Gameplay With Different Character (Character Switched in Characters Shop)



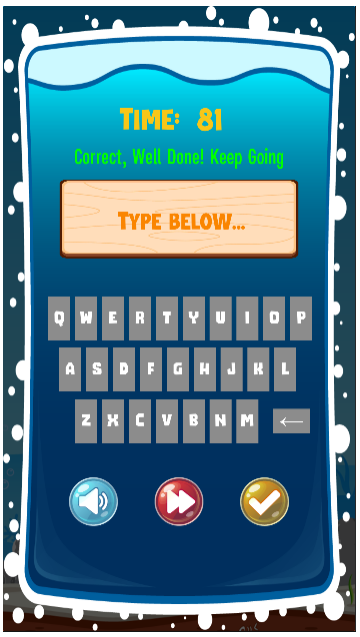
(j) Game Over Pop-Up

9.5 Spelling Game Screenshots



(k) Instructions Pop-Up When Player Went Directly to Spell

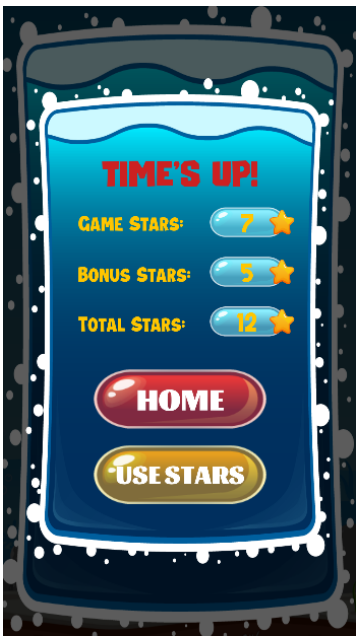
(l) Loading Symbol When Waiting For Phonetic Distance From Server



(m) Correct Answer Message

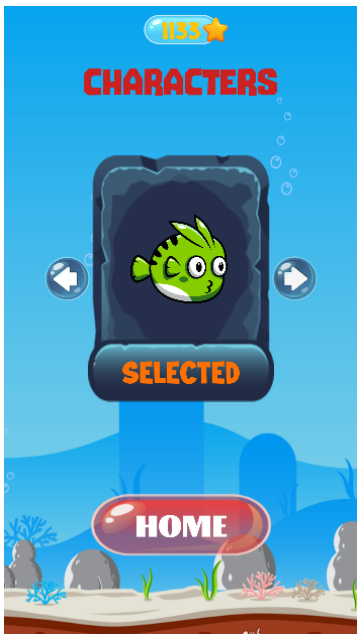
(n) Partially Correct Answer Message

(o) Incorrect Answer Message

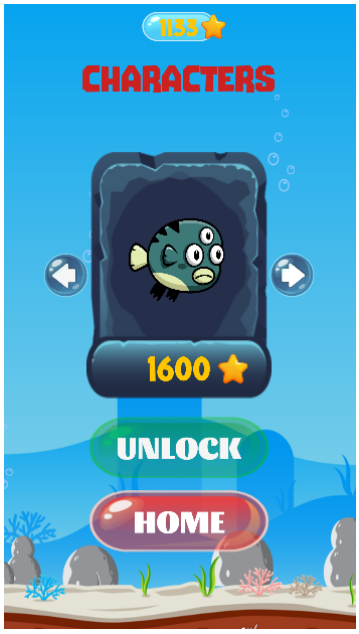


(p) Time's Up Pop-Up When Player Came From Main Game

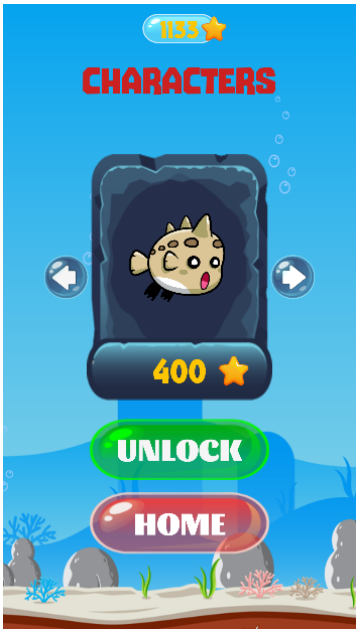
9.6 Characters Shop Screenshots



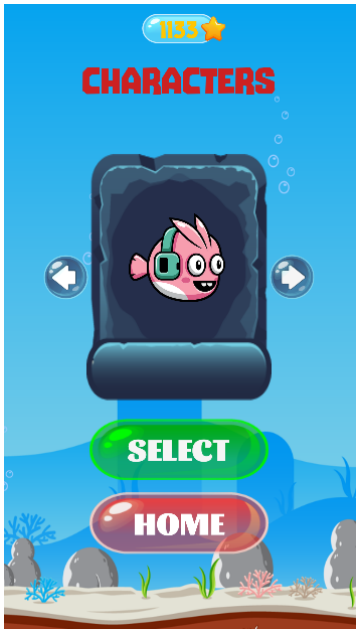
(q) Currently Selected Character



(r) Unaffordable Locked Character



(s) Affordable Locked Character



(t) Unlocked Character

Acknowledgements

I would like to give my warmest thanks to my supervisor, Prof. Julie Berndsen, who provided me with support and guidance throughout the entire project. Her weekly feedback helped me to stay motivated and reach all of my goals. I would also like to express my gratitude to Emma O'Neill (creator of *S-Capade* [32]) for providing constructive feedback that enhanced the final design of the spelling game. Lastly, I am also thankful to my family, friends, and classmates at UCD that downloaded the game and took the time to participate in the survey.

Bibliography

[1] Luis von Ahn. "Games with a Purpose". en. In: *Computer* 39.6 (June 2006), pp. 92–94. ISSN: 0018-9162. DOI: 10.1109/MC.2006.196.

[2] Luis von Ahn and Laura Dabbish. "Designing games with a purpose". en. In: *Commun. ACM* 51.8 (Aug. 2008), pp. 58–67. ISSN: 0001-0782, 1557-7317. DOI: 10.1145/1378704.1378719.

[3] *Animated friendly fish 6*. en-GB. URL: https://www.gamedevmarket.net/asset/animated-friendly-fish-6/ (visited on 04/20/2022).

[4] *Art. 8 GDPR – Conditions applicable to child's consent in relation to information society services*. en-US. URL: https://gdpr-info.eu/art-8-gdpr/ (visited on 11/27/2021).

[5] Eileen W. Ball and Benita A. Blachman. "Phoneme segmentation training: Effect on reading readiness". en. In: *Annals of Dyslexia* 38.1 (Jan. 1988), pp. 208–225. ISSN: 0736-9387, 1934-7243. DOI: 10.1007/BF02648257.

[6] Mahesh Chand. *Top 10 Cloud Service Providers In 2021*. URL: https://www.c-sharpcorner.com/article/top-10-cloud-service-providers/ (visited on 04/19/2022).

[7] *Children's engagement with digital devices, screen time*. URL: https://www.pewresearch.org/internet/2020/07/28/childrens-engagement-with-digital-devices-screen-time/ (visited on 11/26/2021).

[8] *Color Switch on Apple App Store*. URL: https://apps.apple.com/us/app/color-switch/id1314725881 (visited on 11/26/2021).

[9] *DbDesigner*. en-US. URL: https://www.dbdesigner.net/ (visited on 04/29/2022).

[10] Brody Downs et al. "KidSpell: A Child-Oriented, Rule-Based, Phonetic Spellchecker". English. In: *Proceedings of the 12th Language Resources and Evaluation Conference*. Marseille, France: European Language Resources Association, May 2020, pp. 6937–6946. ISBN: 979-10-95546-34-4. URL: https://aclanthology.org/2020.lrec-1.857 (visited on 11/18/2021).

[11] Hadi Fadlallah. *An overview of DIFFERENCE and SOUNDEX SQL functions*. en-US. Jan. 2022. URL: https://www.sqlshack.com/an-overview-of-difference-and-soundex-sql-functions/ (visited on 04/26/2022).

[12] Sarah M. Flanagan et al. "An exploration of the data collection methods utilised with children, teenagers and young people (CTYPs)". In: *BMC Research Notes* 8.1 (Mar. 2015). ISSN: 1756-0500. DOI: 10.1186/s13104-015-1018-y.

[13] *Fruit Ninja on Apple App Store*. URL: https://play.google.com/store/apps/details?id=com.halfbrick.fruitninjafree (visited on 11/26/2021).

[14] *Game Assets for Indie Developers*. en-GB. URL: https://www.gamedevmarket.net/ (visited on 04/20/2022).

[15] *Game Sprites & Sheet Templates from GraphicRiver*. en. URL: https://graphicriver.net/category/game-assets/sprites (visited on 04/14/2022).

[16] Isabela Granic, Adam Lobel, and Rutger C. M. E. Engels. "The benefits of playing video games". In: *American Psychologist* 69.1 (2014). Place: US Publisher: American Psychological Association, pp. 66–78. ISSN: 1935-990X. DOI: 10.1037/a0034857.

[17] Mark Griffiths. "The educational benefits of videogames". In: *Education and Health* 20 (Jan. 2002), pp. 47–51.

[18] Victoria Jane Hodge and Jim Austin. *An evaluation of phonetic spell checkers*. 2001.

[19] *How many words are there in English?* URL: https://www.merriam-webster.com/help/faq-how-many-english-words (visited on 11/18/2021).

[20] *Information and Communications Technology (ICT) in the Primary School Curriculum*. URL: https://www.curriculumonline.ie/getmedia/4adfbc22-f972-45a1-a0ba-d1864c69dff2/ICT-Guidelines-Primary-Teachers.pdf (visited on 11/22/2021).

[21] *International Students: Age/Grade Conversion | EXPLO*. URL: https://explore.explo.org/age-grade-conversion-chart (visited on 04/15/2022).

[22] *JavaFX*. URL: https://openjfx.io/ (visited on 12/01/2021).

[23] Karen Kukich. "Techniques for automatically correcting words in text". In: *ACM Comput. Surv.* 24.4 (Dec. 1992), pp. 377–439. ISSN: 0360-0300. DOI: 10.1145/146370.146380.

[24] Edith Law and Luis Von Ahn. "Input-agreement: a new mechanism for collecting data using human computation games". In: *Proceedings of the SIGCHI Conference on Human Factors in Computing Systems*. 2009, pp. 1197–1206.

[25] Edith LM Law et al. "TagATune: A Game for Music and Sound Annotation." In: *ISMIR*. Vol. 3. 2007.

[26] *LibGDX*. URL: https://libgdx.com/ (visited on 12/01/2021).

[27] *LWJGL*. URL: https://www.lwjgl.org/ (visited on 12/01/2021).

[28] Jane McGonigal and Inc OverDrive. *Reality Is Broken*. en. OCLC: 898986343. Penguin Group US, 2011. ISBN: 978-1-101-46715-2. URL: http://api.overdrive.com/v1/collections/v1L2BaQAAAJcBAAA1M/products/44205f90-d66a-4b2b-8483-8a0cdb6a8822 (visited on 11/17/2021).

[29] Gustavo Augusto de Mendonça Almeida et al. "Evaluating Phonetic Spellers for User-Generated Content in Brazilian Portuguese". en. In: *Computational Processing of the Portuguese Language*. Ed. by João Silva et al. Lecture Notes in Computer Science. Cham: Springer International Publishing, 2016, pp. 361–373. ISBN: 978-3-319-41552-9. DOI: 10.1007/978-3-319-41552-9_37.

[30] *Most popular database management systems 2022*. en. URL: https://www.statista.com/statistics/809750/worldwide-popularity-ranking-database-management-systems/ (visited on 04/19/2022).

[31] Emma O'Neill et al. "The Influence of Regional Pronunciation Variation on Children's Spelling and the Potential Benefits of Accent Adapted Spellcheckers". In: *Proceedings of the 25th Conference on Computational Natural Language Learning*. Online: Association for Computational Linguistics, Nov. 2021, pp. 674–683. URL: https://aclanthology.org/2021.conll-1.52 (visited on 11/19/2021).

[32] Emma O'Neill et al. "S-Capade: Spelling Correction Aimed at Particularly Deviant Errors". en. In: *Statistical Language and Speech Processing*. Ed. by Luis Espinosa-Anke, Carlos Martín-Vide, and Irena Spasić. Lecture Notes in Computer Science. Cham: Springer International Publishing, 2020, pp. 85–96. ISBN: 978-3-030-59430-5. DOI: 10.1007/978-3-030-59430-5_7.

[33] Lisa Pearl and Mark Steyvers. "Identifying Emotions, Intentions, and Attitudes in Text Using a Game with a Purpose". In: *Proceedings of the NAACL HLT 2010 Workshop on Computational Approaches to Analysis and Generation of Emotion in Text*. Los Angeles, CA: Association for Computational Linguistics, June 2010, pp. 71–79. URL: https://aclanthology.org/W10-0209 (visited on 11/17/2021).

[34] Lisa Pearl and Mark Steyvers. "Identifying Social Information in Text Using Human-Based Computation". en. In: *Computational Linguistics* (2005).

[35] *phpMyAdmin*. en. URL: https://www.phpmyadmin.net/ (visited on 04/29/2022).

[36] *Plastic in the Ocean*. June 2018. URL: https://www.natgeokids.com/uk/kids-club/cool-kids/general-kids-club/plastic-in-the-ocean/ (visited on 04/14/2022).

[37] *Relational vs. Non-Relational Database: Pros & Cons | The Aloa Blog*. URL: https://aloa.co/blog/relational-vs-non-relational-database-pros-cons (visited on 04/17/2022).

[38] *Research highlights the frequency of smartphone usage in Ireland | Irish Life Corporate Business*. URL: https://www.irishlifecorporatebusiness.ie/research-highlights-frequency-smartphone-usage-ireland (visited on 11/22/2021).

[39] Stephen Robertson, Milan Vojnovic, and Ingmar Weber. "Rethinking the ESP game". en. In: *CHI '09 Extended Abstracts on Human Factors in Computing Systems*. Boston MA USA: ACM, Apr. 2009, pp. 3937–3942. ISBN: 978-1-60558-247-4. DOI: 10.1145/1520340.1520597.

[40] Kosovare Sahatqija et al. "Comparison between relational and NOSQL databases". In: *2018 41st International Convention on Information and Communication Technology, Electronics and Microelectronics (MIPRO)*. May 2018, pp. 216–221. DOI: 10.23919/MIPRO.2018.8400041.

[41] *Sequence-to-Sequence G2P toolkit*. original-date: 2016-04-25T12:53:41Z. Apr. 2022. URL: https://github.com/cmusphinx/g2p-seq2seq (visited on 04/26/2022).

[42] *Sound-It-Out User Experience Survey Results*. en. URL: https://docs.google.com/forms/d/1UB74PUOFOsMe_CBiZA489AyftLlVNpw-yWiHMmRpAG0/viewanalytics (visited on 04/29/2022).

[43] Unity Technologies. *Unity - Manual: Prefabs*. en. URL: https://docs.unity3d.com/Manual/Prefabs.html (visited on 04/26/2022).

[44] Unity Technologies. *Unity - Manual: Scenes*. en. URL: https://docs.unity3d.com/Manual/CreatingScenes.html (visited on 04/21/2022).

[45] *The Basic Spelling Vocabulary List*. en. Apr. 2013. URL: https://www.readingrockets.org/article/basic-spelling-vocabulary-list (visited on 04/15/2022).

[46] Jimmy To. *Speech And Text in Unity iOS and Unity Android*. original-date: 2016-11-07T08:15:03Z. Apr. 2022. URL: https://github.com/j1mmyto9/Speech-And-Text-Unity-iOS-Android (visited on 04/25/2022).

[47] Duane R. Tovey. ""Sound-it-out": A reasonable approach to spelling?" In: *Reading World* 17.3 (Mar. 1978), pp. 220–233. ISSN: 0149-0117. DOI: 10.1080/19388077809557420.

[48] Jean M. Twenge and W. Keith Campbell. "Associations between screen time and lower psychological well-being among children and adolescents: Evidence from a population-based study". In: *Preventive Medicine Reports* 12 (2018), pp. 271–283. ISSN: 2211-3355. DOI: https://doi.org/10.1016/j.pmedr.2018.10.003.

[49] *Unity Asset Store*. URL: https://assetstore.unity.com/ (visited on 12/01/2021).

[50] *Unity's 2D game engine, evolved*. URL: https://unity.com/solutions/2d-game-engine (visited on 12/01/2021).

[51] Naser Valaei et al. "Ads in gaming apps: experiential value of gamers". In: *Industrial Management & Data Systems* ahead-of-print.ahead-of-print (Jan. 2021). ISSN: 0263-5577. DOI: 10.1108/IMDS-11-2020-0660.

[52] Jean Veronis. "Computerized correction of phonographic errors". en. In: *Comput Hum* 22.1 (Mar. 1988), pp. 43–56. ISSN: 1572-8412. DOI: 10.1007/BF00056348.

[53] *What is personal data?* URL: https://ec.europa.eu/info/law/law-topic/data-protection/reform/what-personal-data_en (visited on 11/27/2021).

[54] Mike Wolfe. *MySQL vs Oracle SQL*. en. Aug. 2021. URL: https://towardsdatascience.com/mysql-vs-oracle-sql-a97a7659f992 (visited on 04/19/2022).

[55] *XAMPP*. en. URL: https://sourceforge.net/projects/xampp/ (visited on 04/29/2022).