LINGOLUDUS - TECHNICAL GUIDE

Student 1: Fawwaz Kekere-Ekun **Student number:** 18408814

Student 2: Yann Ndjatang **Student number:** 18337813

Project Supervisor: Monica Ward

Completion Date:

Table Of Contents

1. **Introduction**
   1.1. Overview
   1.2. Motivation

2. **Research**
   2.1. Python & Pygame
   2.2. Language
   2.3. User Interface
   2.4. Game
   2.5. Sprites
   2.6. Sound

3. **Implementation & Design**
   3.1. System Architecture
   3.2. High Level Design

4. **Problems and Resolution**
   4.1. Changing Languages and Updating Stats
   4.2. Spawn enemies
   4.3. Sound not working

5. **Validation & Testing**
   5.1. Ad-hoc Testing
   5.2. User Testing
   5.3. Learning Outcomes

6. **Future Work**
   6.1 Future Work
   6.2 Conclusion

1.Introduction

1.1 Overview

LingoLudus is a language learning application which aims to gamify the language learning experience by making use of computer assisted language learning (CALL). Computer assisted learning is any learning that is facilitated by a computer and that does not require direct interaction between the user and a human teacher to function. CALL provides the user with an interface (created by a knowledgeable teacher in the subject) that allows the user to follow a lesson or learn a subject of interest. Such materials may be structured or unstructured, but they normally embody two important features: interactive learning and individualised learning. When compared to other more traditional education methodologies, CALL has been reported to improve information retention and achievement scores, improve judgement skills and minimise required teaching time.

Our application intends to adopt this technology and further enhance the learning process of languages by gamifying it. Gamification tackles the issue of monotony in normal teaching by giving the user a goal in which they have to reach and putting obstacles in front of it. This makes the process of learning more dynamic and enjoyable as users would have to interact in different ways to reach the goal they desire and would also be self-motivated to get new high scores upon playing.

LingoLudus consists of one game being a space shooter, where you play as a spaceship with the aim of shooting down the correct target displayed on the screen. This game allows you to enjoy practising your vocabulary for 3 different topics: numbers, clothes and Fruits.You will be able to enjoy learning these topics in two different popular languages; French and Spanish. As you play the game you’ll be able to see the spelling of the target in the language you’re learning and hear the pronunciation of the word as it approaches as well as get a visual representation of the word in the form of a sprite target. This makes use of the different forms of learning such as visual learning, auditory learning and finally, kinaesthetic learning. When combined, it makes the retention of information much easier. Getting a score for every correct target hit and a streak, also losing health for incorrect target hits makes the game engaging and gives the user a sense of accomplishment.

The game was designed to be as modular as possible to allow for future changes and additional content to be added easily without having to rewrite or heavily overhaul the already written code. This is achieved by having a base game which can be changed to take in different data which will in turn display different vocabulary to the user. This allowed us to work on one game and polish it rather than having many different games that may not work very well.

2.2 Motivation

The idea was proposed by our supervisor Monica, we decided to go with the idea because we have a strong interest in gaming and we’re both bilingual. With this combined we came to a conclusion that this would be a fun and challenging project to tackle as our final year project. As it would require learning a new module like Pygame and new concepts such as learning how to create a game from scratch without the aid of a game engine, learning how to design a game, how to build levels and how to build a complete mini game environment. It would require learning about how to interact with the pygame module and learning about what pygame can and can’t do. Most importantly, it would require trying to figure out what would be needed in the first place to create such a system and how to integrate all these different components together.

The challenges this project presented us with as well as the potential to build something that had useful real world applications was very inviting. We decided this was the type of project we would like to work on and hoped that over the course of trying to build this game we would learn a lot more than we had known previously.

2. Research

PYTHON & PYGAME

LingoLudus was developed using the python Pygame module. We decided to use python to develop this project as we enjoy the language and have worked with it for many other projects. At first the plan was to use pygame in conjunction with unity but after consulting with our supervisor we decided to use Pygame alone to develop this game as opposed to Unity because Unity is a game engine that makes game development easier by providing many pre-created assets and other features. We felt like this would be too much of a help and would have lowered the complexity of the overall project. We believed building this from the ground up with pygame would be adequate enough to qualify as a fourth year project.

LANGUAGE

When selecting the language to teach for our game this was a quick decision as Spanish and French are the most spoken and popular languages apart from English and Chinese. Plus we both have experiences with French and Spanish as one of us is fluent in French and the other has Spanish experience. We made this decision because it would help us with the pronunciation of words we planned to integrate into our game as sounds.

USER INTERFACE

For the user interface we went with a Pixelated design to give the application its own identity whilst also not being too extravagant. This meant that the game could still look appealing and allow us to also find assets to design it much easier. Taking skills we learned from the 3rd year module: CA357, We designed some mock interface with which we consulted classmates and supervisor for feedback on the design. We also researched different papers on colour choice in respect to enhancing learning to help us choose the colour palate that we wanted to go with. And made use of this colour UI site: https://mui.com/material-ui/customization/color/ to identify the right colour choices based on their contrast when combined with different colours. With the advice that we were able to get from user testers we were able to style the page in a simplistic way so that it wouldn't be a nuisance to navigate for the user.

GAME

While researching which game to create, we came up with several different ideas such as platformer, an rpg, quiz game, a quest game etc. But we narrowed it down based on thinking about how we were going to implement the learning aspect to it. This made our choice a bit easier as we wanted a game to implement the three forms of learning that being visual learning, auditory learning and kinesthetic learning. With this in mind, our game needed to be simple and not too complex to keep the emphasis on the learning of a language rather than learning on how to play a game.

SPRITES

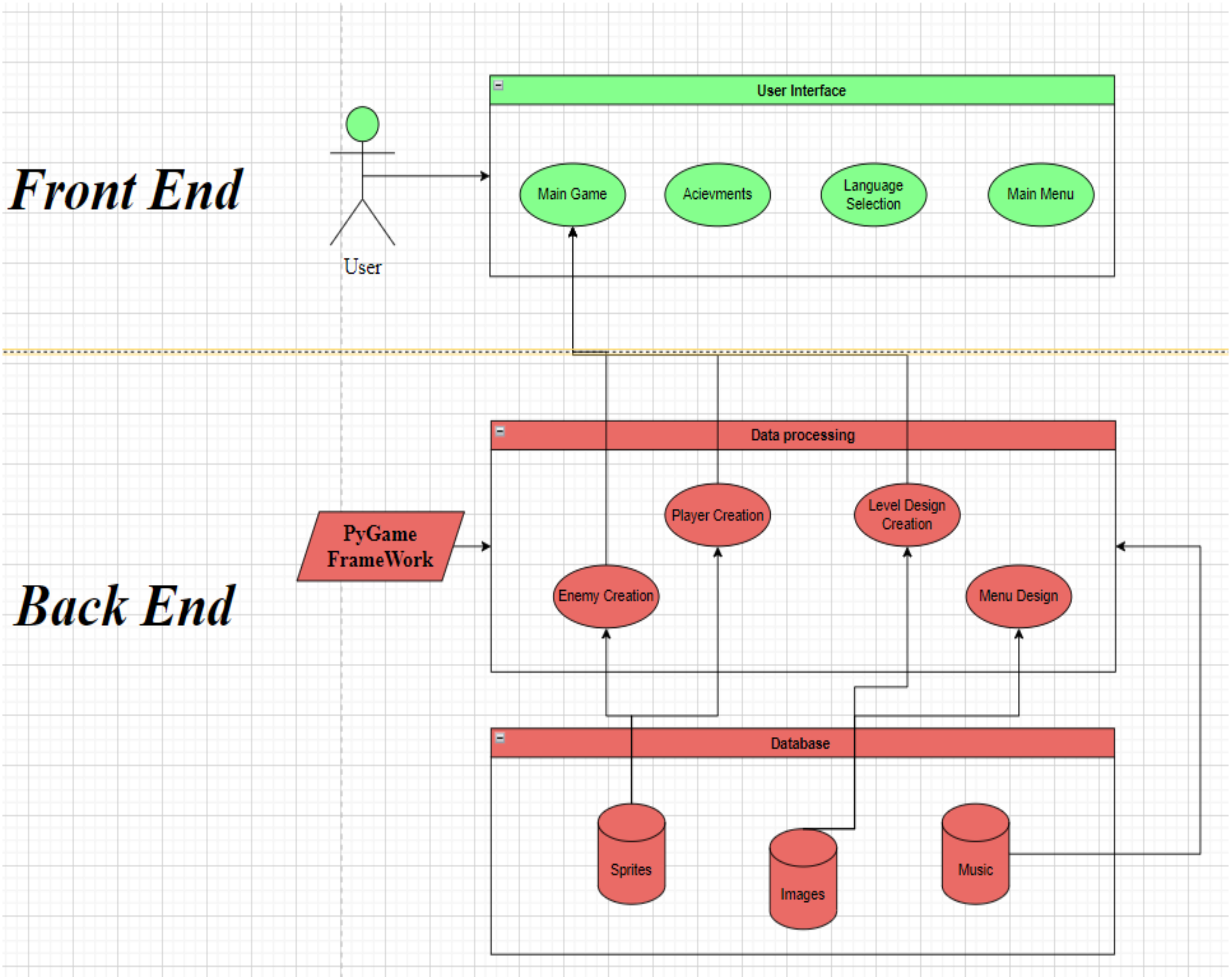
For the research of our sprites, google was our main source of information. We needed to learn about implementing sprites into our and limitations they have within the pygame module. We made use of sites like https://itch.io/game-assets/free/tag-pixel-art/tag-space and images on google to acquire royalty free sprites for use in our game.

SOUND

The pronunciation of words was recorded by us as we both have experience with French and spanish. This reduced the research of how we were going to find French and Spanish words being pronounced. For the game sound we used google to find them.

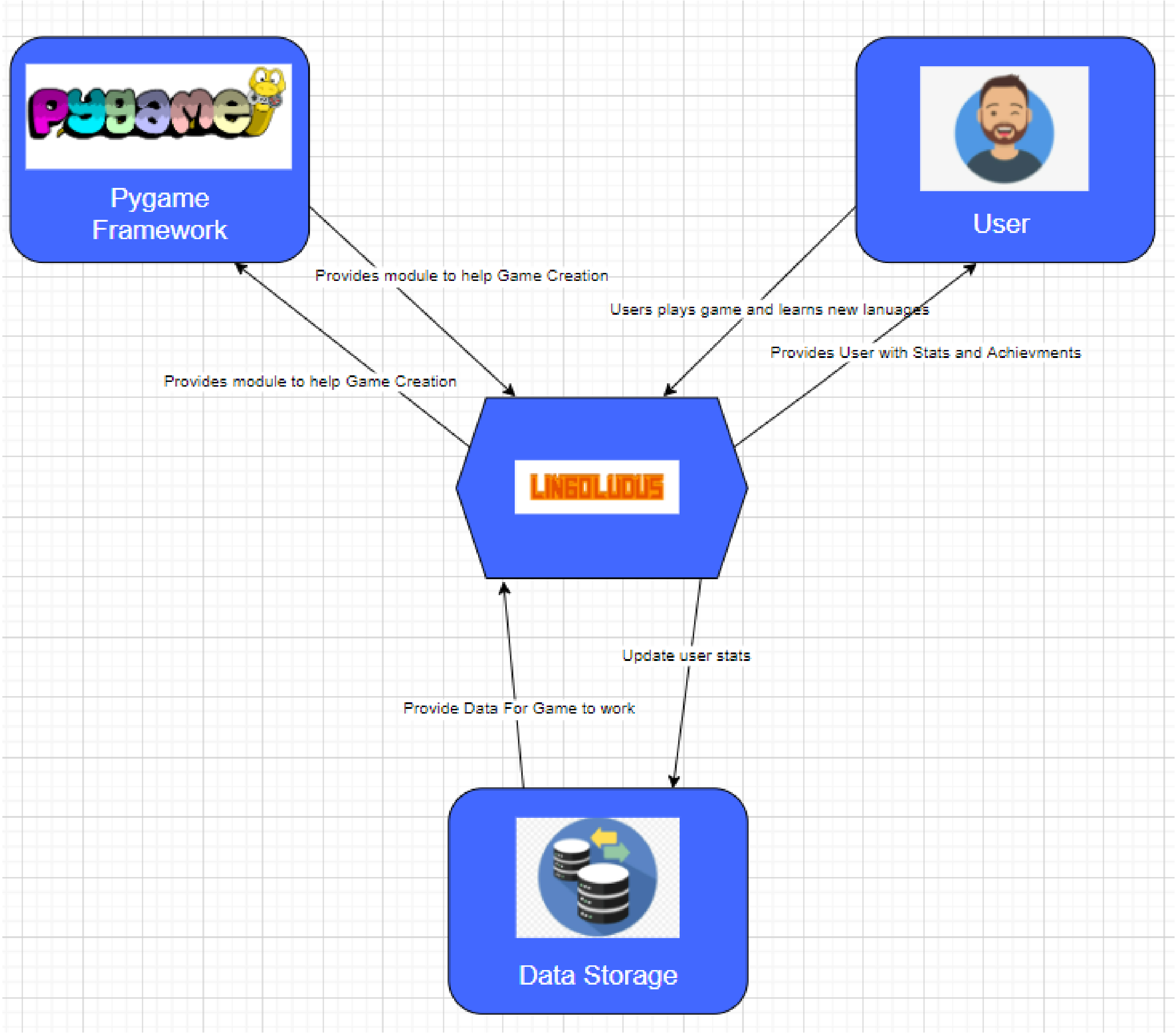
3. Implementation & Design

3.1 System Architecture Diagram



The System architecture that is implemented is one of a modular framework. This means that a lot of the functionality has been put into different classes and functions in which different attributes can be changed easily without having to rewrite a lot of the code. For example, we tried to make a lot of the placement of objects on the screen be based on the screen width as opposed to a particular point in space so that if we were to make a change to the screen size no other variables would need to be changed in the code.

3.2 High-Level Design


Pygame Framework
User
Provides module to help Game Creation
Users plays game and learns new lanuages
Provides User with Stats and Achievments
Provides module to help Game Creation
LINGOLUDUS
Update user stats
Provide Data For Game to work
Data Storage

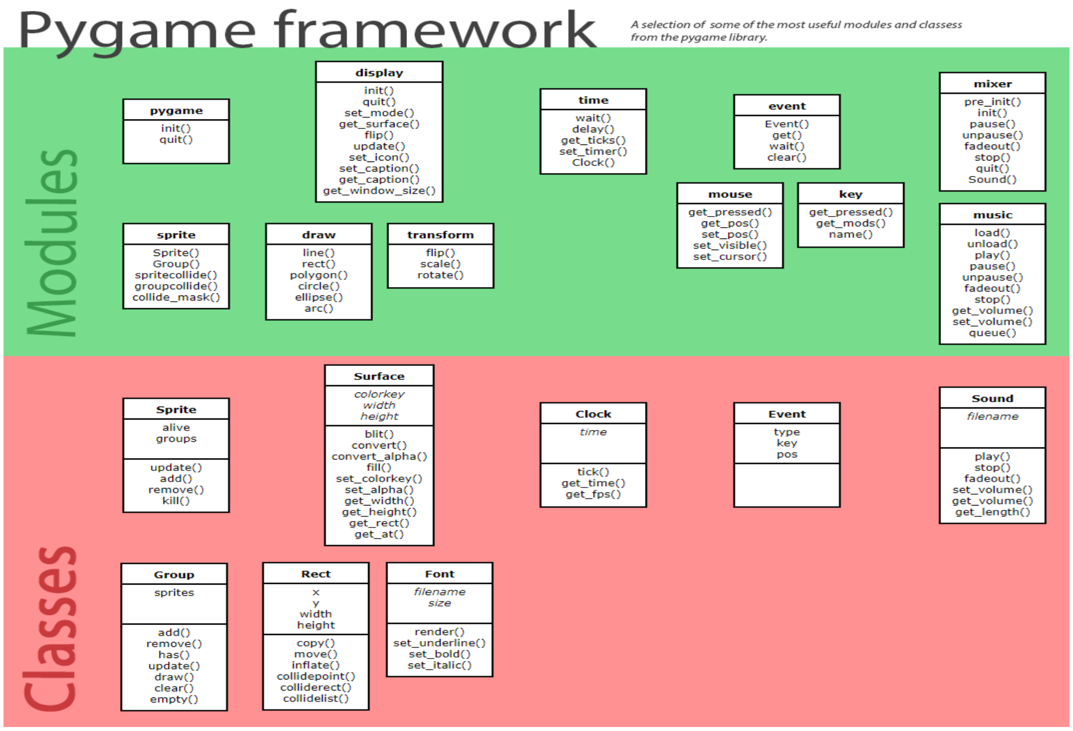
This Diagram shows the relationship between the Game and the software used to create and hold its data. It also shows the relationship between the Game and its users.

We see that the Game makes use of the Pygame framework to create the game. This is accomplished by using the modules functions to draw objects, create sprites, add music and much more.

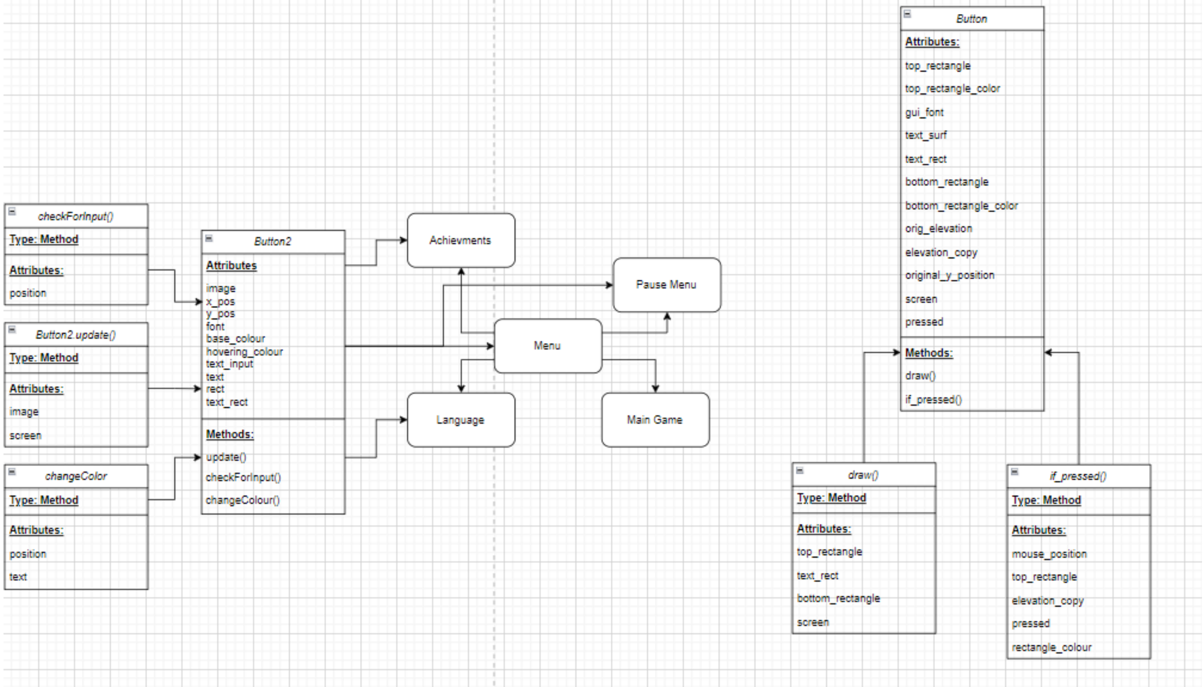
The Game also creates and stores its data within files and dictionaries created using python. The path for different pieces of data is stored within dictionaries which is to be loaded up by the Game when needed.

3.3 Class Diagram

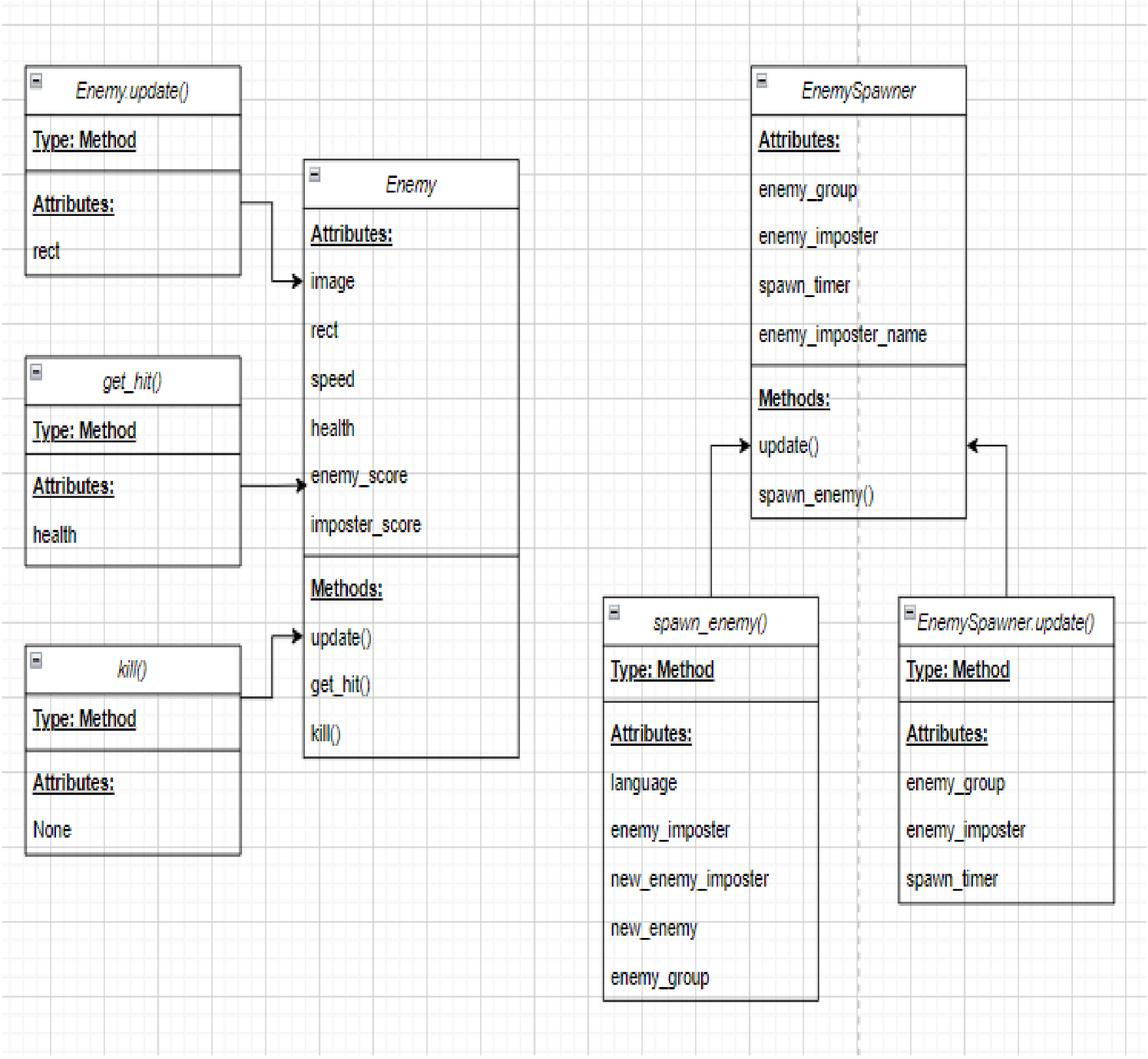
Pygame Class Diagram



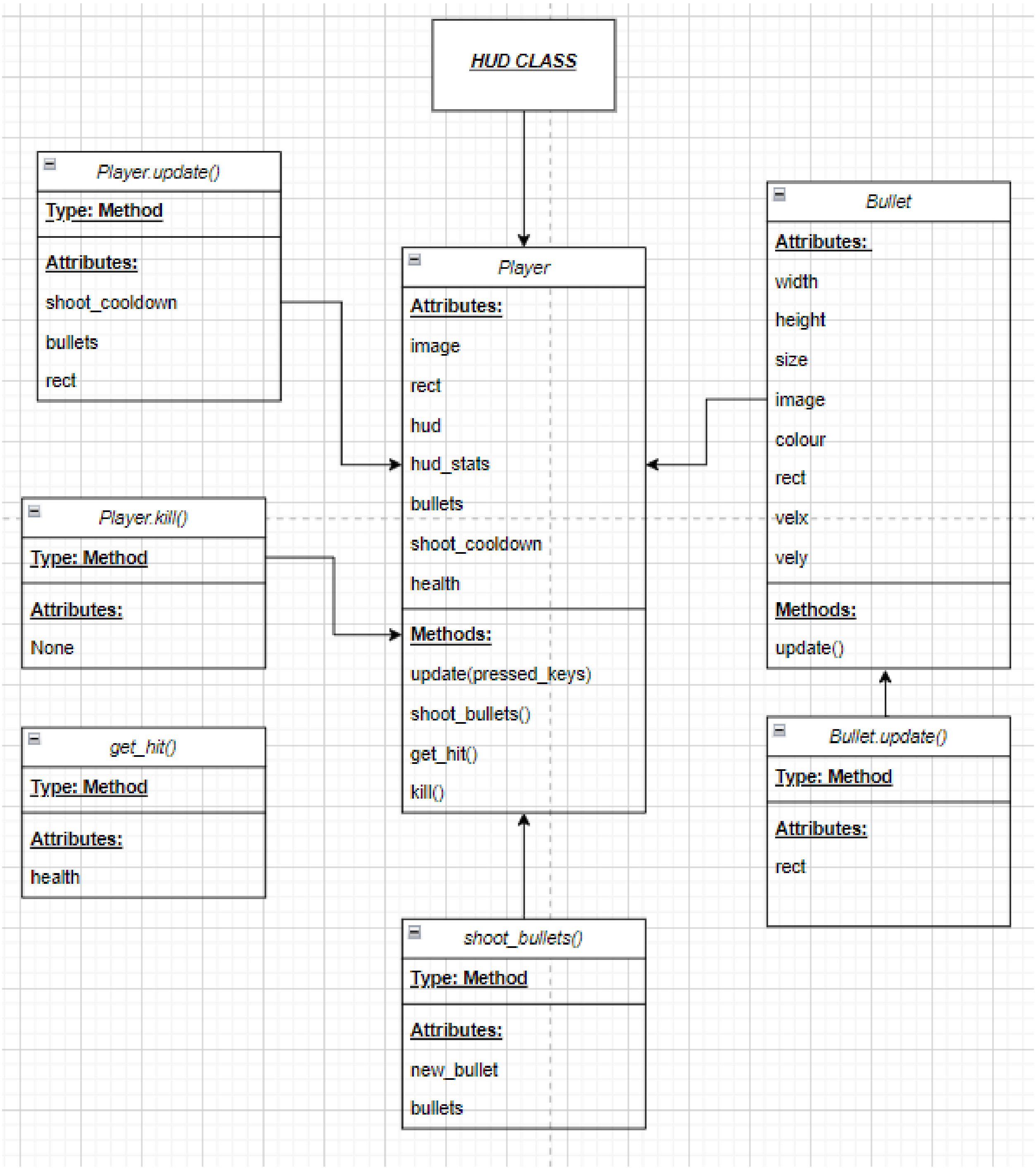
Menu Class Diagram



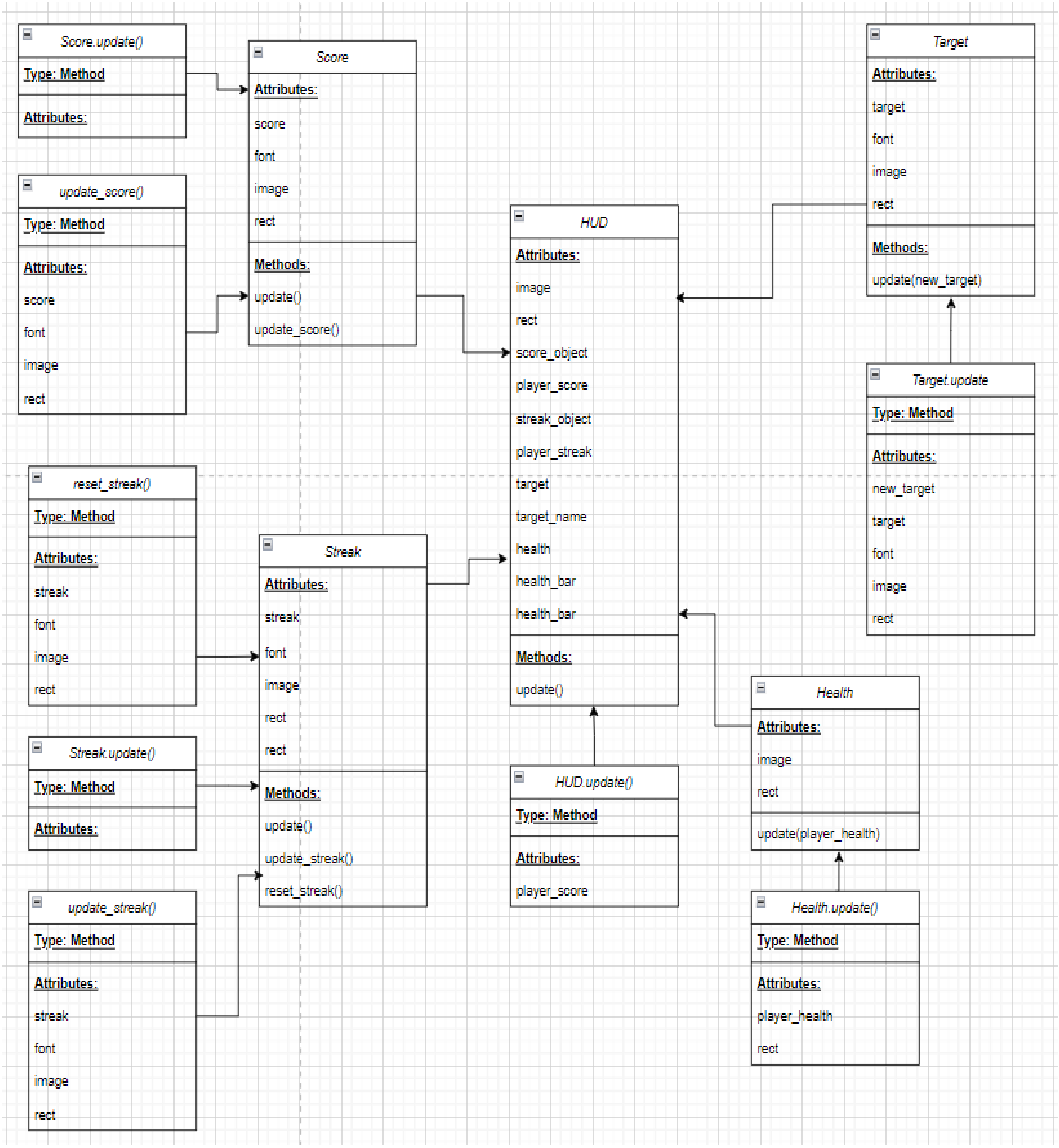
Enemy and Enemy Spawner Class Diagram

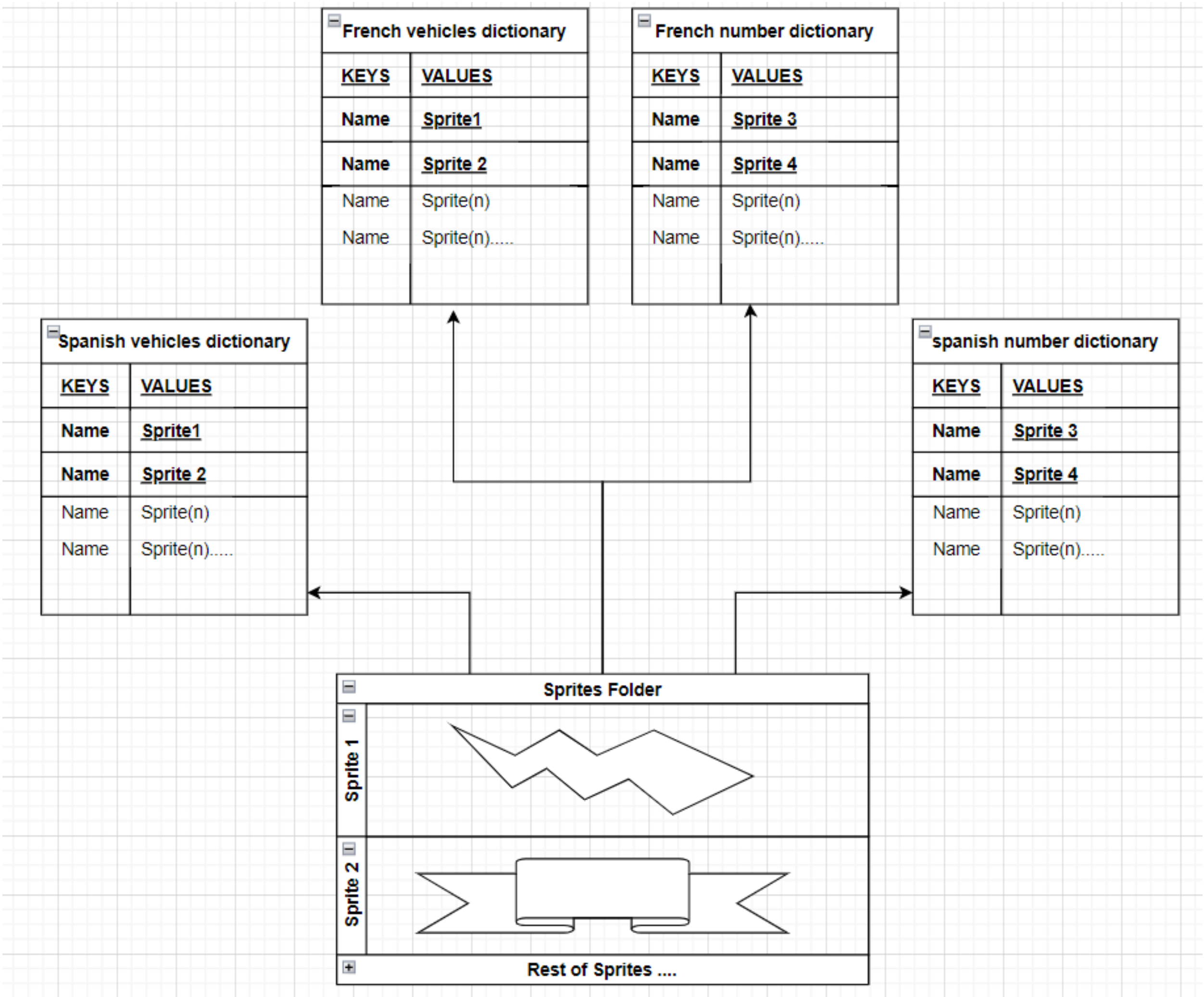


Menu Class Diagram



HUD Class Diagram





French vehicles dictionary

KEYS	VALUES
Name	Sprite1
Name	Sprite 2
Name	Sprite(n)
Name	Sprite(n).....

French number dictionary

KEYS	VALUES
Name	Sprite 3
Name	Sprite 4
Name	Sprite(n)
Name	Sprite(n).....

Spanish vehicles dictionary

KEYS	VALUES
Name	Sprite1
Name	Sprite 2
Name	Sprite(n)
Name	Sprite(n).....

spanish number dictionary

KEYS	VALUES
Name	Sprite 3
Name	Sprite 4
Name	Sprite(n)
Name	Sprite(n).....

Our code is spread out between multiple files and classes in which we make use of imports, class inheritance and polymorphism. We have many classes but the main ones which encompass others would be our Player, Enemy, Enemy Spawner, HUD, Achievements, Main Menu and Main Game classes.

The Menu Class is the class that starts up the game into the main menu. It imports the button classes to be used. These will run different files which encompass other menus and classes such as the achievements and language selection screen.

Moving to the achievements screen from the main menu will bring up the achievement's pages for French and Spanish which show the different stats that players have accumulated while playing the game. The achievements class takes in data from the stats files to be displayed on the screen while also making use of the Button class from the button files to move between the French and Spanish achievements pages.

Going back to the main menu, the language button when pressed will run the select_language file which allows the player to change the language of the game by editing the vocab file.

Finally the Play button on the main menu screen simply runs the main game file which starts up the main game. This game itself makes use of many different classes to operate. The Player, enemy spawner, alert box and button classes are all made use of all these different classes.

The game.py was mainly used to draw the game elements which consisted of objects made from these classes. The Player class was used to create the player ship that would be controlled by the user. This class made use of bullets class which was a class which controlled the bullets that the ship would shoot out and the HUD class which is the class that contained all the information about the players stats and health.

The HUD class displays a box of the user stat along with the target that they should be hitting. The different classes that the HUD uses are the Score, Streak, Target, and Health classes. Each of these classes do what the name implies. The objects are saved into the HUD class as variables to be drawn.

All these different classes being linked together means that it is much easier to change one aspect of the game without having to rewrite lots of code.

Data Storage

We stored the data for sprites locally in a sprites file rather than making use of a database. The sprite names were then saved into a python dictionary as values in which the corresponding target names were the keys. This allowed for data to be pulled simply by calling from dictionaries. It also allowed us to give a way to link the sprites to particular names which could then be used for the sake of creating target names.

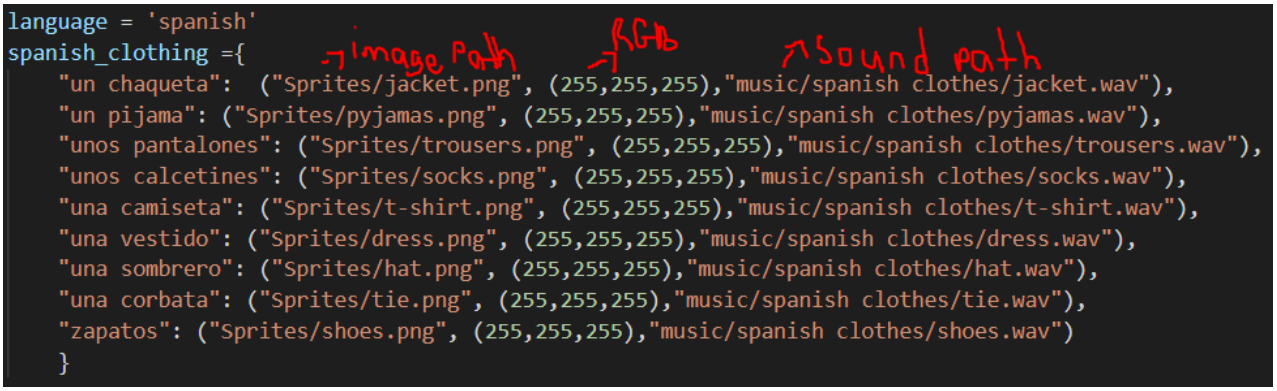
In creating the database models we had to refactor our initial design idea of using a database to store all our data and create a more simple structure for things to be stored. This structure might not be the most efficient but for the purpose of showcasing our idea for this project it would work.

The data that would need to be stored were the images that would be used as sprites to create enemy objects and player objects in the game, and also the sound files associated with all the vocab sprites and other functions in the game. Since we didn’t make use of a database we implemented this model by simply having a sprites file which contained all these images, and an image file which contained the sound files.

We took the name of each sprite and added it to a dictionary based on the language that it would be used for. Each sprite was given a name in the corresponding dictionaries language as a key. The value for each key was then made up of a three variable tuple. The first parameter contained the name of the file of the image, the second contained the rgb value of its background, and the third contained the name of the sound file it would use. The essence of giving them names was that we would later use those names to attach to images as target names to display in the game for the users.

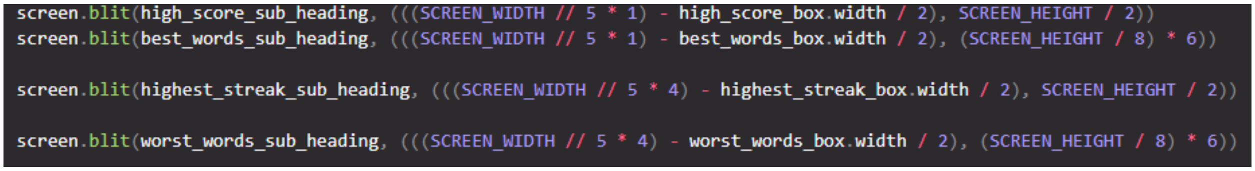
SAMPLE CODE

When creating an enemy the enemy's image would be loaded by finding the first parameter of the value tuple in the dictionary which contains the path of the image in the Sprite folder.
The second parameter would be used as the background rgb to make it transparent, and the third parameter would be used as the path to the sound file for that target. The target name was found by getting the index of the key that corresponded to the sprite in question then setting that at the target name to be displayed to the player.



```
language = 'spanish'
spanish_clothing ={
    "un chaqueta":  ("Sprites/jacket.png", (255,255,255),"music/spanish clothes/jacket.wav"),
    "un pijama": ("Sprites/pyjamas.png", (255,255,255),"music/spanish clothes/pyjamas.wav"),
    "unos pantalones": ("Sprites/trousers.png", (255,255,255),"music/spanish clothes/trousers.wav"),
    "unos calcetines": ("Sprites/socks.png", (255,255,255),"music/spanish clothes/socks.wav"),
    "una camiseta": ("Sprites/t-shirt.png", (255,255,255),"music/spanish clothes/t-shirt.wav"),
    "una vestido": ("Sprites/dress.png", (255,255,255),"music/spanish clothes/dress.wav"),
    "una sombrero": ("Sprites/hat.png", (255,255,255),"music/spanish clothes/hat.wav"),
    "una corbata": ("Sprites/tie.png", (255,255,255),"music/spanish clothes/tie.wav"),
    "zapatos": ("Sprites/shoes.png", (255,255,255),"music/spanish clothes/shoes.wav")
    }
```

We also tried to make it that changing the size of the game as a whole would not pose a problem if need were to arise. Therefore, we drew almost all the objects on the screen mathematically in relation to the screen size rather than giving them fixed positions. This also gave the sizes of objects good proportions as opposed to us drawing objects to different pixel locations based on look alone.



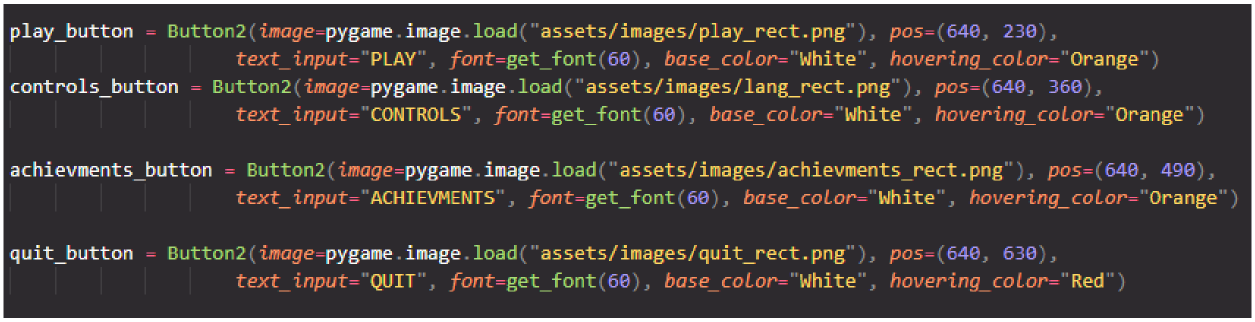
```
screen.blit(high_score_sub_heading, (((SCREEN_WIDTH // 5 * 1) - high_score_box.width / 2), SCREEN_HEIGHT / 2))
screen.blit(best_words_sub_heading, (((SCREEN_WIDTH // 5 * 1) - best_words_box.width / 2), (SCREEN_HEIGHT / 8) * 6))

screen.blit(highest_streak_sub_heading, (((SCREEN_WIDTH // 5 * 4) - highest_streak_box.width / 2), SCREEN_HEIGHT / 2))

screen.blit(worst_words_sub_heading, (((SCREEN_WIDTH // 5 * 4) - worst_words_box.width / 2), (SCREEN_HEIGHT / 8) * 6))
```

Frontend
The user interface was made up of a couple of different python files, one to handle each page that was displayed to the user. To reduce the amount of code that we would need to write. We made great use of the import function which allowed us keep our files structured and our code as clean as possible.

When designing the UI we wanted it to be consistent the reason why most of the menus reuses the same code structure keeping our code clean.

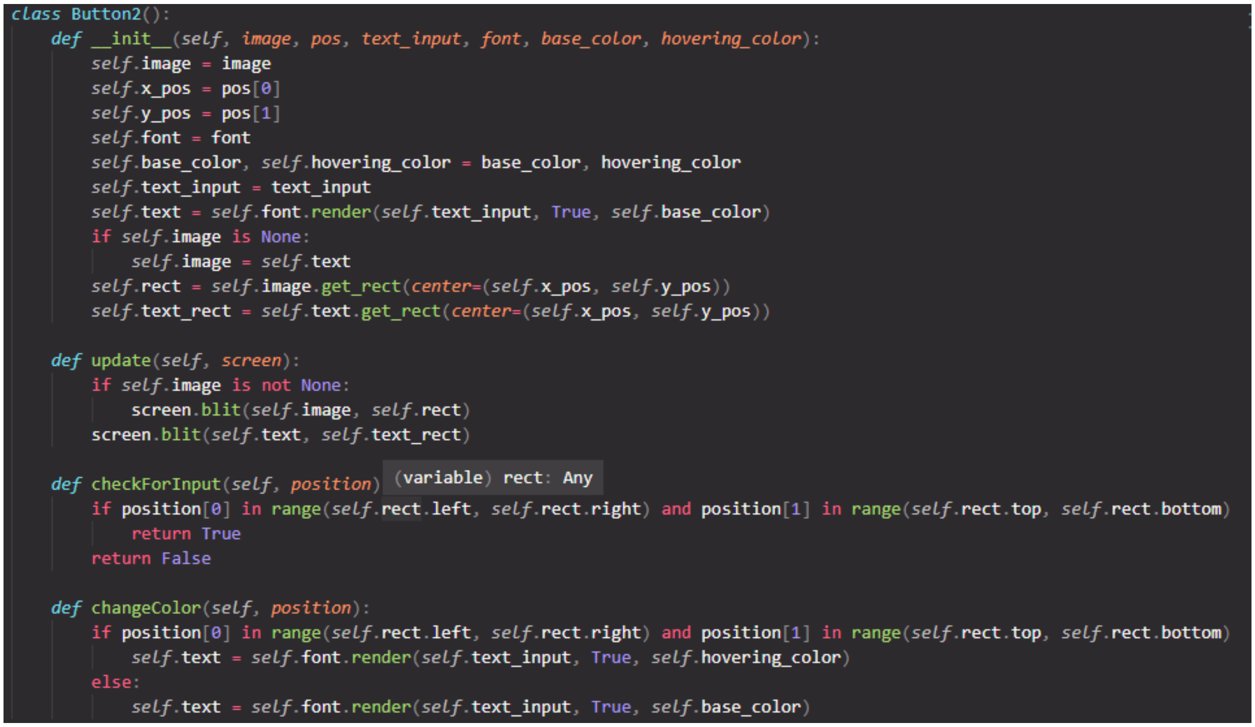


```python
play_button = Button2(image=pygame.image.load("assets/images/play_rect.png"), pos=(640, 230),
                    text_input="PLAY", font=get_font(60), base_color="White", hovering_color="Orange")
controls_button = Button2(image=pygame.image.load("assets/images/lang_rect.png"), pos=(640, 360),
                    text_input="CONTROLS", font=get_font(60), base_color="White", hovering_color="Orange")

achievments_button = Button2(image=pygame.image.load("assets/images/achievments_rect.png"), pos=(640, 490),
                    text_input="ACHIEVMENTS", font=get_font(60), base_color="White", hovering_color="Orange")

quit_button = Button2(image=pygame.image.load("assets/images/quit_rect.png"), pos=(640, 630),
                    text_input="QUIT", font=get_font(60), base_color="White", hovering_color="Red")
```

This is achieved by our button class that takes in 7 variables made in a separate file and imported. The first variable being a rectangle background for the button to sit on. The second variable takes in the position of the button, the third variable takes the button text, 4th variable takes the text font which is acquired from a separate function. 5th variable takes the colour of the text and final variable takes the hovering colour when the mouse goes over the button



```python
class Button2():
    def __init__(self, image, pos, text_input, font, base_color, hovering_color):
        self.image = image
        self.x_pos = pos[0]
        self.y_pos = pos[1]
        self.font = font
        self.base_color, self.hovering_color = base_color, hovering_color
        self.text_input = text_input
        self.text = self.font.render(self.text_input, True, self.base_color)
        if self.image is None:
            self.image = self.text
        self.rect = self.image.get_rect(center=(self.x_pos, self.y_pos))
        self.text_rect = self.text.get_rect(center=(self.x_pos, self.y_pos))

    def update(self, screen):
        if self.image is not None:
            screen.blit(self.image, self.rect)
        screen.blit(self.text, self.text_rect)

    def checkForInput(self, position)
        if position[0] in range(self.rect.left, self.rect.right) and position[1] in range(self.rect.top, self.rect.bottom)
            return True
        return False

    def changeColor(self, position):
        if position[0] in range(self.rect.left, self.rect.right) and position[1] in range(self.rect.top, self.rect.bottom)
            self.text = self.font.render(self.text_input, True, self.hovering_color)
        else:
            self.text = self.font.render(self.text_input, True, self.base_color)
```

4. Problems and Resolution

4.1 Changing Languages & Updating stats

The idea of this app is that it allows the user to select the language they want to learn which will then load in the corresponding words in the chosen language. Our languages vocabulary is stored in a vocab.p which consists of the two separate languages we provide them being french and spanish. One of our first problems we encountered was loading in the language vocab when the user selected a language to learn in the language selection menu. Every time the user selected a language the appropriate data was being loaded incorrectly or it would often load the opposite language. The same problems happened to our achievements page not getting updated after each game

We found out that the data wasn't being changed in real time because we had to exit the game and reload the game for the language and achievements to get updated. This issue was resolved by finding the reload() method from the importlib module. This allowed us to reload our previously imported module after being edited based on user selection.

4.2 Spawn enemies (vocab sprites)

One of the main features of the main game is the spawning of enemies which in our case are sprites of the corresponding word but along with other sprites in the same topics selected, randomly spawned to make it harder for the user, forcing them to shoot down the correct sprite or suffer a health loss.

The problem we encountered here was first being the sprite images not having a transparent background; this was fixed by converting the image to an alpha channel using the convert_alpha() function in pygame. Another problem we had with the vocab sprites was they overlapped each other making it hard for the user to shoot down the correct one, although we didn't completely fix the issue we compromised by making the sprites smaller allowing them to be distinguishable from one another.

4.3 Sound not working

Our application incorporates sound features to enhance the user experience. This was a problem for us when incorporating it at first as we had recorded all the vocab words and made use of the pygame.mixer to add the sounds and background music but which didn't work for us at first.

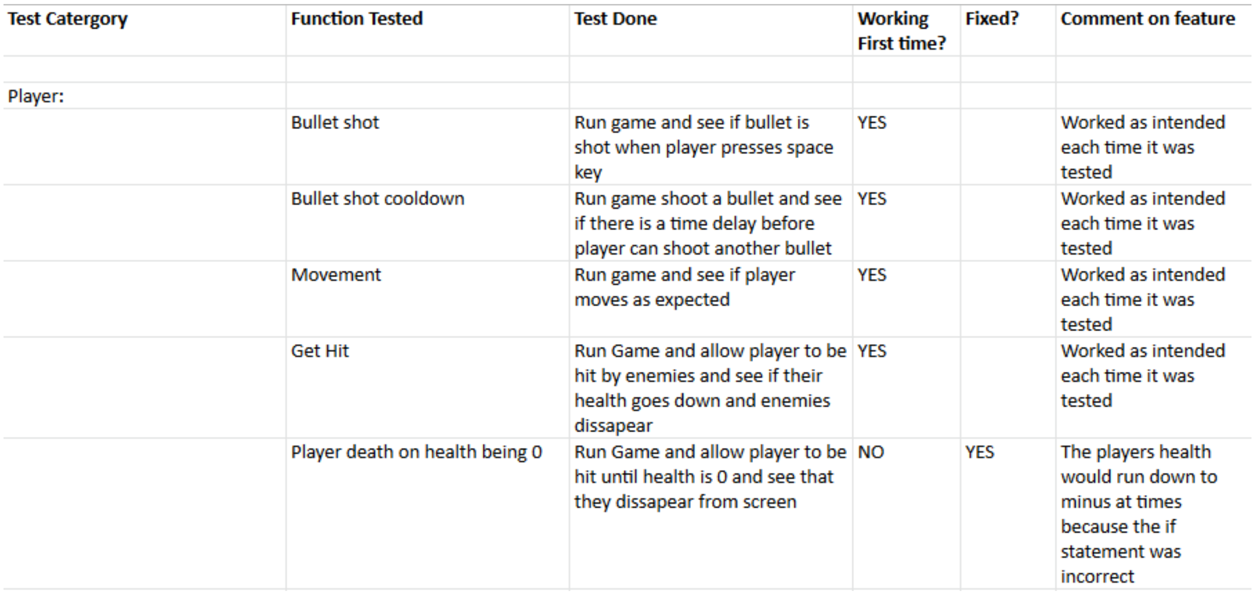
The is problem was resolved by doing some research in incorporating sounds in pygame where we found out our problem was a filetype was the issue. We recorded and converted our vocab words to .WAV files and where we saw some positive results and sounds working.

5. Validation & Testing

5.1 Ad-hoc Testing

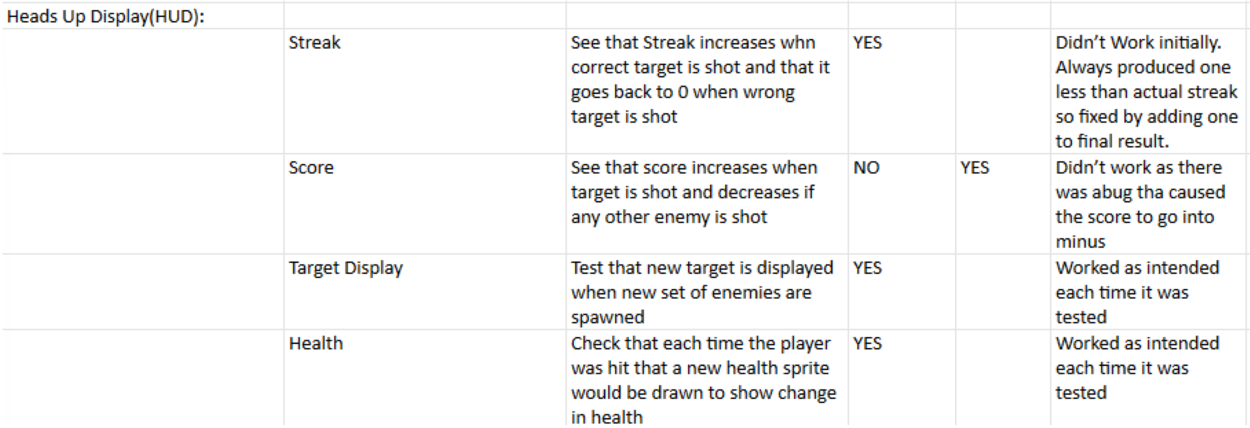
Throughout the development of the application there was frequent ad-hoc testing undertaken. This came in the form of specifically trying to break the application's functionality from an end-user's standpoint. We split this into different category and kept track of the results using google sheets.

First category was done on the user's functionality, with the columns describing the test procedures.



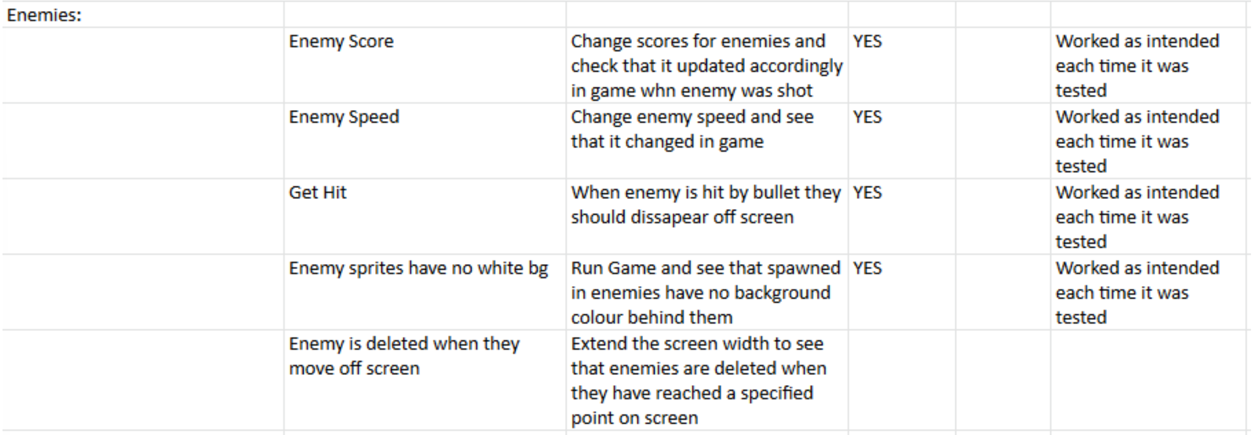
Test Catergory	Function Tested	Test Done	Working First time?	Fixed?	Comment on feature
Player:					
	Bullet shot	Run game and see if bullet is shot when player presses space key	YES		Worked as intended each time it was tested
	Bullet shot cooldown	Run game shoot a bullet and see if there is a time delay before player can shoot another bullet	YES		Worked as intended each time it was tested
	Movement	Run game and see if player moves as expected	YES		Worked as intended each time it was tested
	Get Hit	Run Game and allow player to be hit by enemies and see if their health goes down and enemies dissapear	YES		Worked as intended each time it was tested
	Player death on health being 0	Run Game and allow player to be hit until health is 0 and see that they dissapear from screen	NO	YES	The players health would run down to minus at times because the if statement was incorrect

Next was the HUD



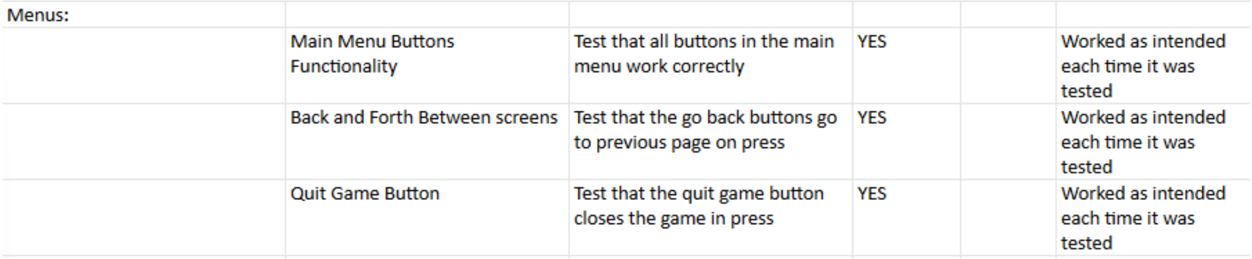
Heads Up Display(HUD):					
	Streak	See that Streak increases whn correct target is shot and that it goes back to 0 when wrong target is shot	YES		Didn't Work initially. Always produced one less than actual streak so fixed by adding one to final result.
	Score	See that score increases when target is shot and decreases if any other enemy is shot	NO	YES	Didn't work as there was abug tha caused the score to go into minus
	Target Display	Test that new target is displayed when new set of enemies are spawned	YES		Worked as intended each time it was tested
	Health	Check that each time the player was hit that a new health sprite would be drawn to show change in health	YES		Worked as intended each time it was tested

Following was the enemy spawner.



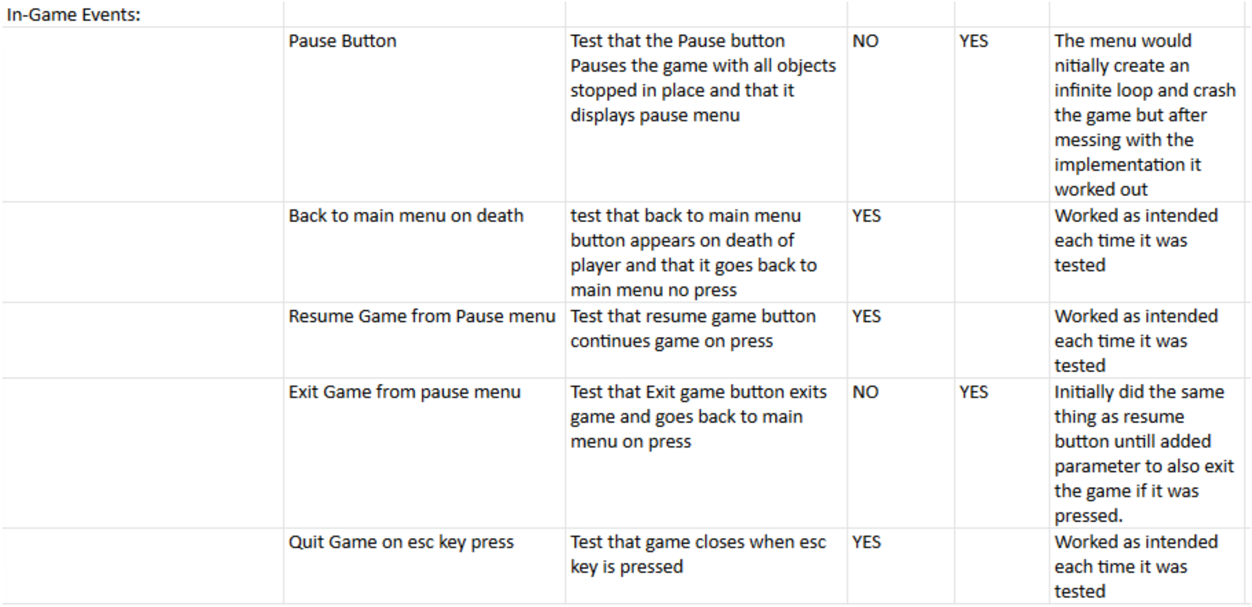
Enemies:					
	Enemy Score	Change scores for enemies and check that it updated accordingly in game whn enemy was shot	YES		Worked as intended each time it was tested
	Enemy Speed	Change enemy speed and see that it changed in game	YES		Worked as intended each time it was tested
	Get Hit	When enemy is hit by bullet they should dissapear off screen	YES		Worked as intended each time it was tested
	Enemy sprites have no white bg	Run Game and see that spawned in enemies have no background colour behind them	YES		Worked as intended each time it was tested
	Enemy is deleted when they move off screen	Extend the screen width to see that enemies are deleted when they have reached a specified point on screen			

Menu testing:



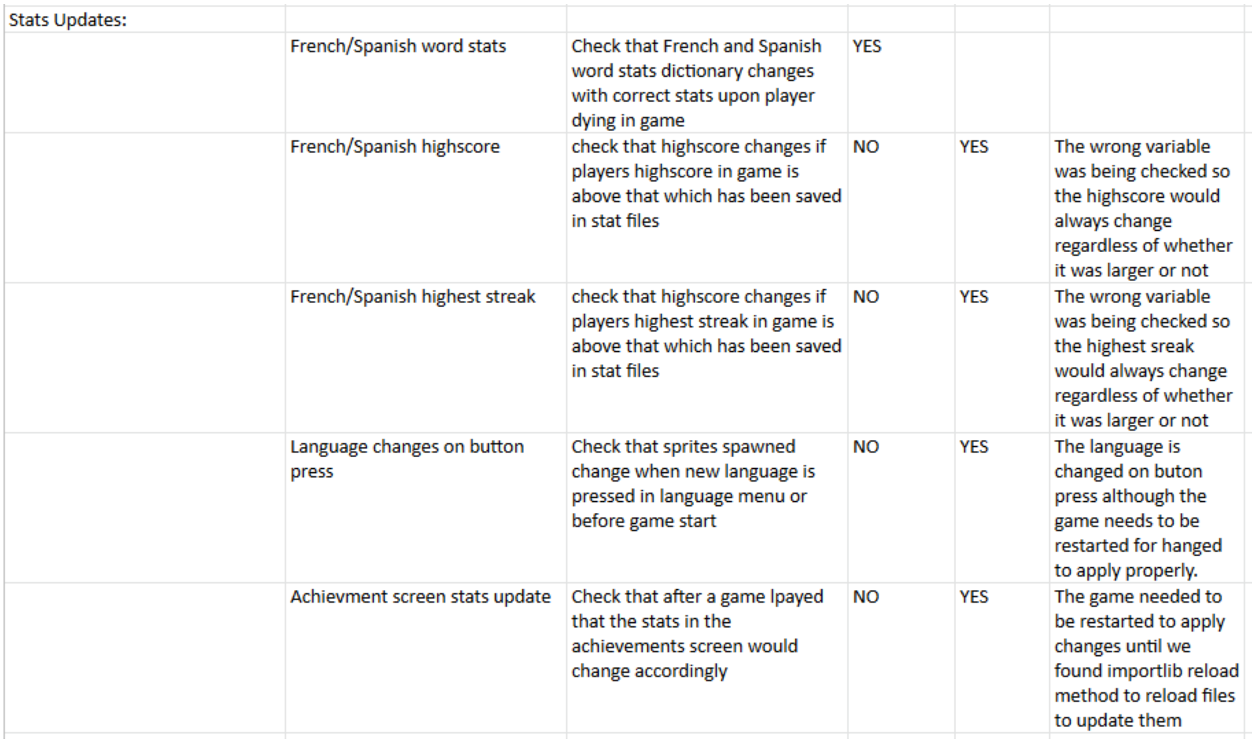
Menus:					
	Main Menu Buttons Functionality	Test that all buttons in the main menu work correctly	YES		Worked as intended each time it was tested
	Back and Forth Between screens	Test that the go back buttons go to previous page on press	YES		Worked as intended each time it was tested
	Quit Game Button	Test that the quit game button closes the game in press	YES		Worked as intended each time it was tested

In-game events testing:



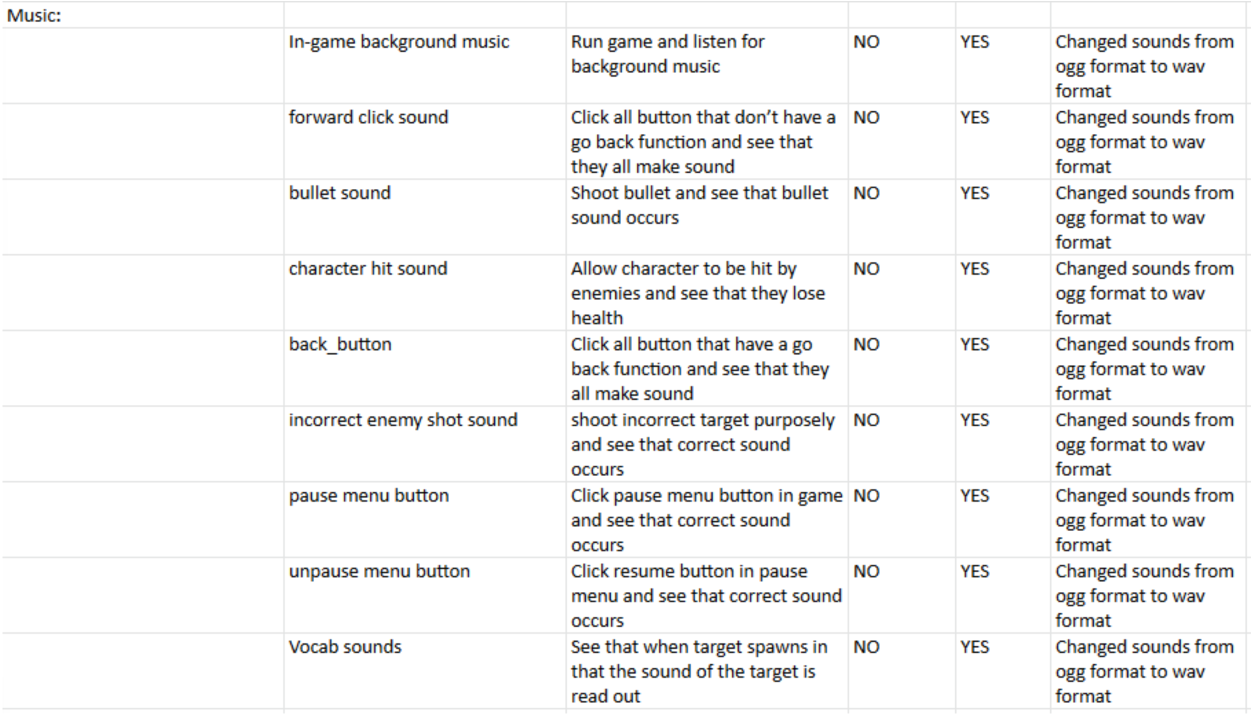
In-Game Events:					
	Pause Button	Test that the Pause button Pauses the game with all objects stopped in place and that it displays pause menu	NO	YES	The menu would nitially create an infinite loop and crash the game but after messing with the implementation it worked out
	Back to main menu on death	test that back to main menu button appears on death of player and that it goes back to main menu no press	YES		Worked as intended each time it was tested
	Resume Game from Pause menu	Test that resume game button continues game on press	YES		Worked as intended each time it was tested
	Exit Game from pause menu	Test that Exit game button exits game and goes back to main menu on press	NO	YES	Initially did the same thing as resume button untill added parameter to also exit the game if it was pressed.
	Quit Game on esc key press	Test that game closes when esc key is pressed	YES		Worked as intended each time it was tested

Testing that stats were being updated:



Stats Updates:					
	French/Spanish word stats	Check that French and Spanish word stats dictionary changes with correct stats upon player dying in game	YES		
	French/Spanish highscore	check that highscore changes if players highscore in game is above that which has been saved in stat files	NO	YES	The wrong variable was being checked so the highscore would always change regardless of whether it was larger or not
	French/Spanish highest streak	check that highscore changes if players highest streak in game is above that which has been saved in stat files	NO	YES	The wrong variable was being checked so the highest sreak would always change regardless of whether it was larger or not
	Language changes on button press	Check that sprites spawned change when new language is pressed in language menu or before game start	NO	YES	The language is changed on buton press although the game needs to be restarted for hanged to apply properly.
	Achievment screen stats update	Check that after a game lpayed that the stats in the achievements screen would change accordingly	NO	YES	The game needed to be restarted to apply changes until we found importlib reload method to reload files to update them

Finally music testing:

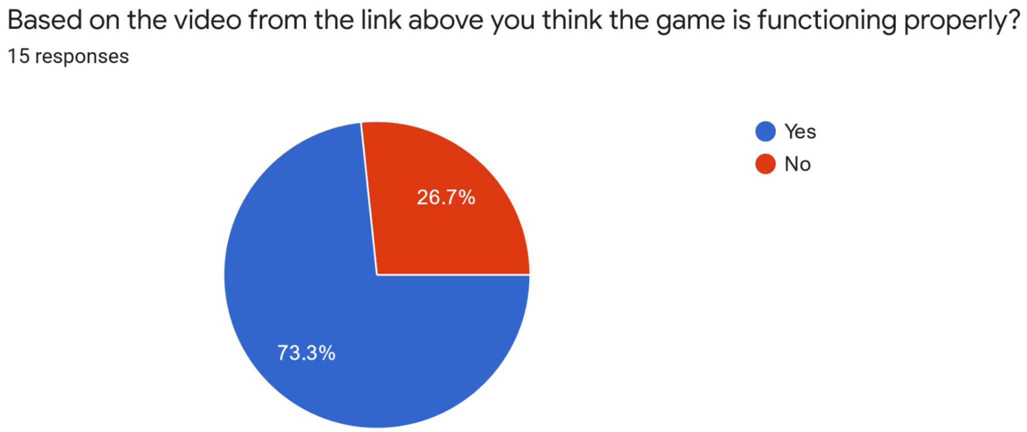


Music:					
	In-game background music	Run game and listen for background music	NO	YES	Changed sounds from ogg format to wav format
	forward click sound	Click all button that don't have a go back function and see that they all make sound	NO	YES	Changed sounds from ogg format to wav format
	bullet sound	Shoot bullet and see that bullet sound occurs	NO	YES	Changed sounds from ogg format to wav format
	character hit sound	Allow character to be hit by enemies and see that they lose health	NO	YES	Changed sounds from ogg format to wav format
	back_button	Click all button that have a go back function and see that they all make sound	NO	YES	Changed sounds from ogg format to wav format
	incorrect enemy shot sound	shoot incorrect target purposely and see that correct sound occurs	NO	YES	Changed sounds from ogg format to wav format
	pause menu button	Click pause menu button in game and see that correct sound occurs	NO	YES	Changed sounds from ogg format to wav format
	unpause menu button	Click resume button in pause menu and see that correct sound occurs	NO	YES	Changed sounds from ogg format to wav format
	Vocab sounds	See that when target spawns in that the sound of the target is read out	NO	YES	Changed sounds from ogg format to wav format

5.2 User Testing

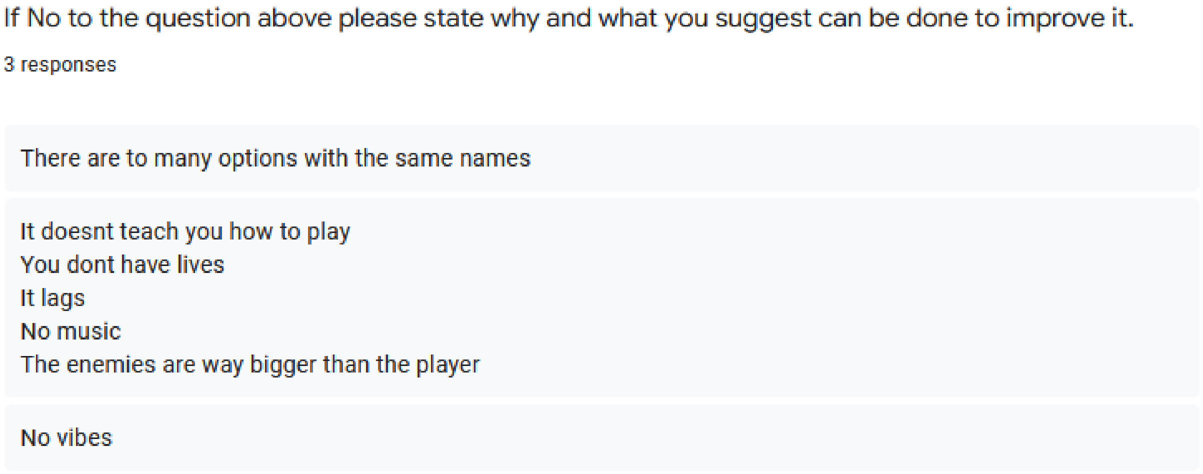
For user testing we went through 3 iterations of the app and for each iterations we did user testing where we decided to survey a sample of acquaintance, friends and family on their experience with our applications.

Some of the questions we asked and results gotten for the first iteration of our app included:



Based on the video from the link above you think the game is functioning properly?

15 responses



If No to the question above please state why and what you suggest can be done to improve it.

3 responses

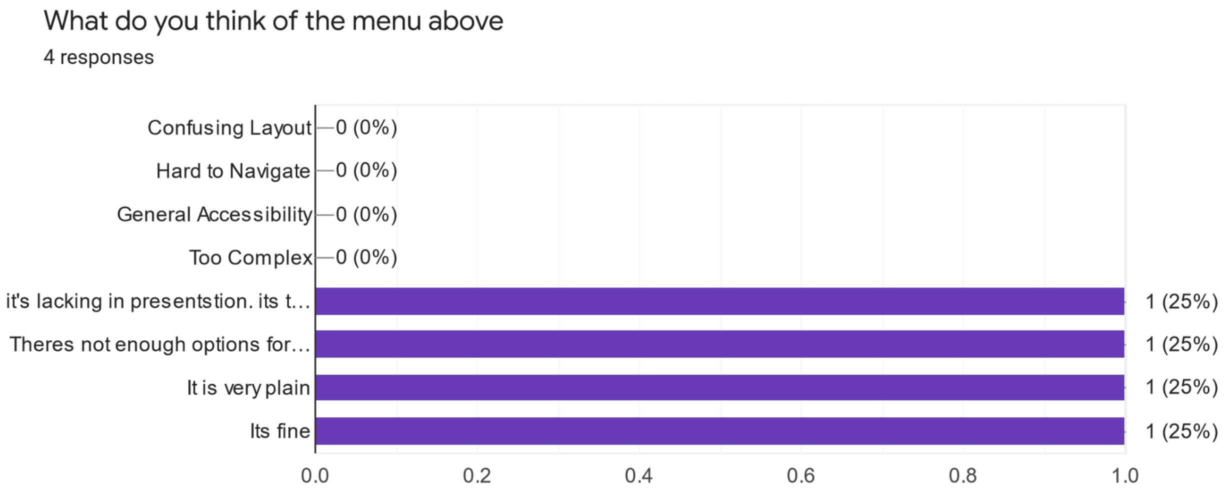
There are to many options with the same names

It doesnt teach you how to play
You dont have lives
It lags
No music
The enemies are way bigger than the player

No vibes

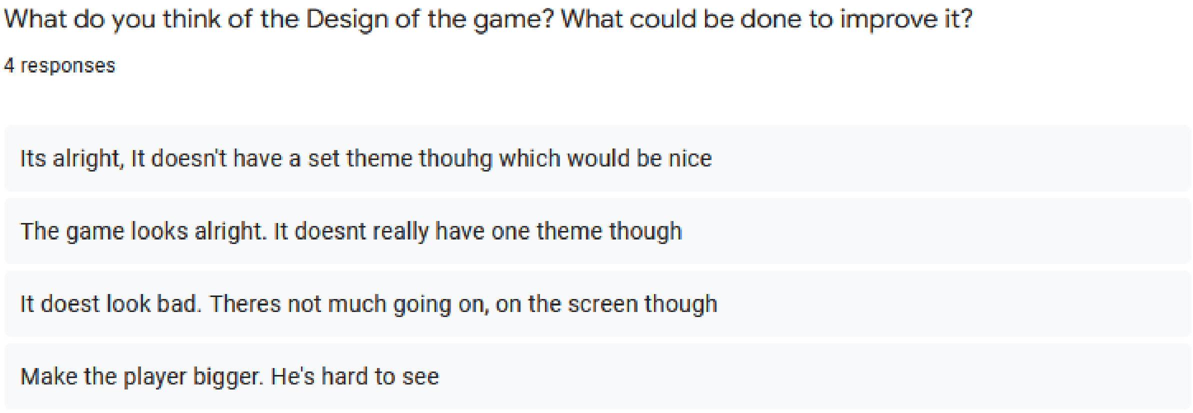
While discussing user interface used in the application, 73% found the game functioning properly and the ones that weren't satisfied suggested some improvements which was good for us as we could then take these into consideration for our next iteration.

And our next iteration questions included were:



What do you think of the menu above

4 responses



What do you think of the Design of the game? What could be done to improve it?

4 responses

Its alright, It doesn't have a set theme thouhg which would be nice

The game looks alright. It doesnt really have one theme though

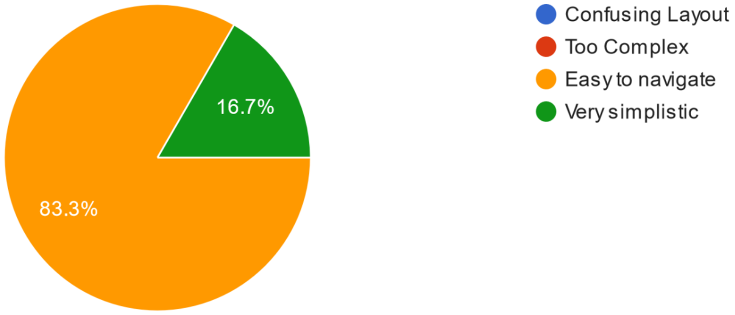
It doest look bad. Theres not much going on, on the screen though

Make the player bigger. He's hard to see

The results from this iteration of our app was another great help in leading us to our final iteration questionnaire. Here are some of the questions that were included in our final iteration.

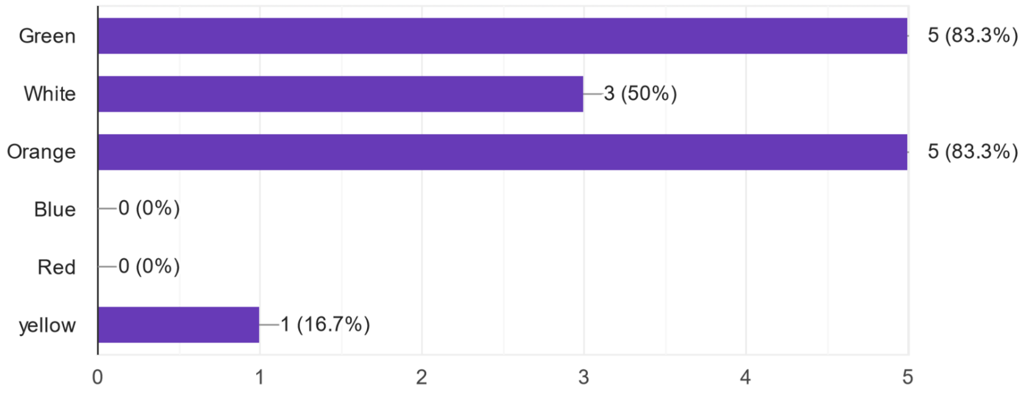
Do you find the menus easy to navigate?

6 responses



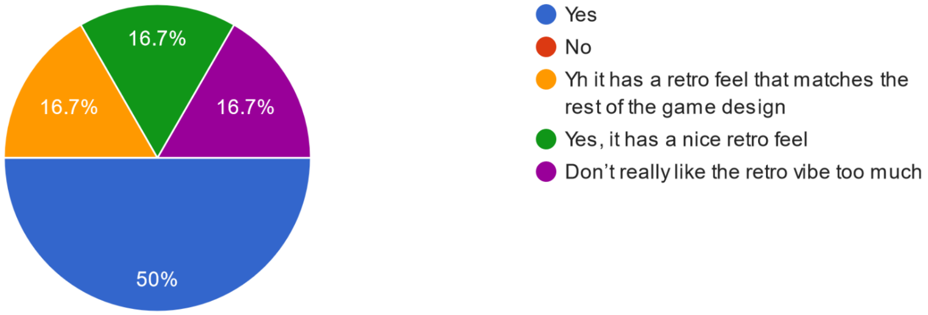
What colours do you remember most from the game

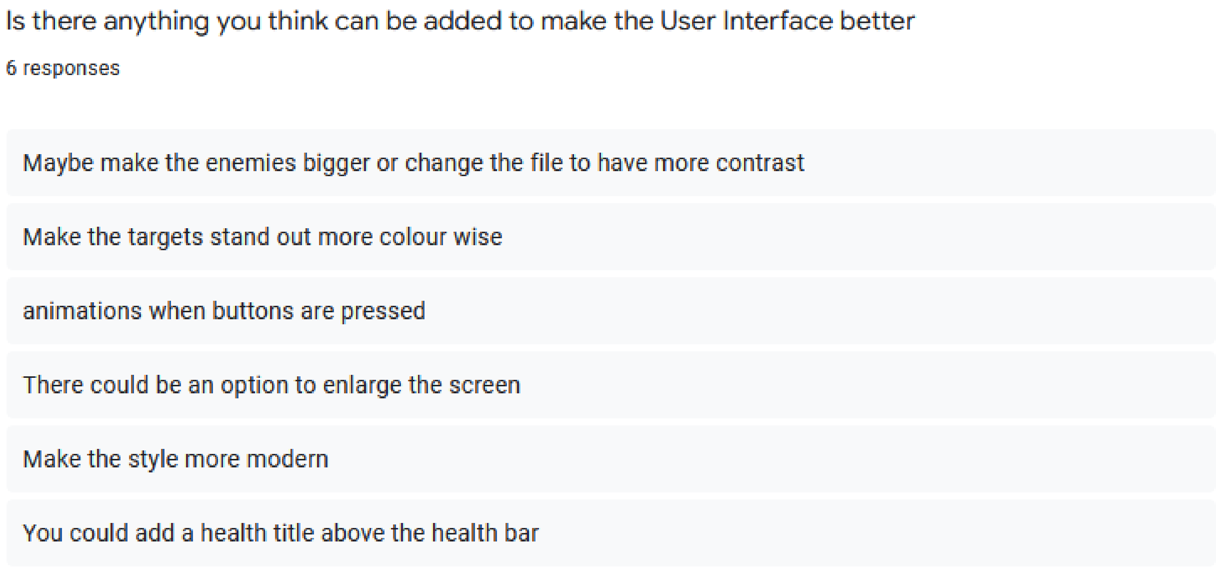
6 responses



Do you find the aesthetics of the game and logo nice

6 responses





Is there anything you think can be added to make the User Interface better

6 responses

Maybe make the enemies bigger or change the file to have more contrast

Make the targets stand out more colour wise

animations when buttons are pressed

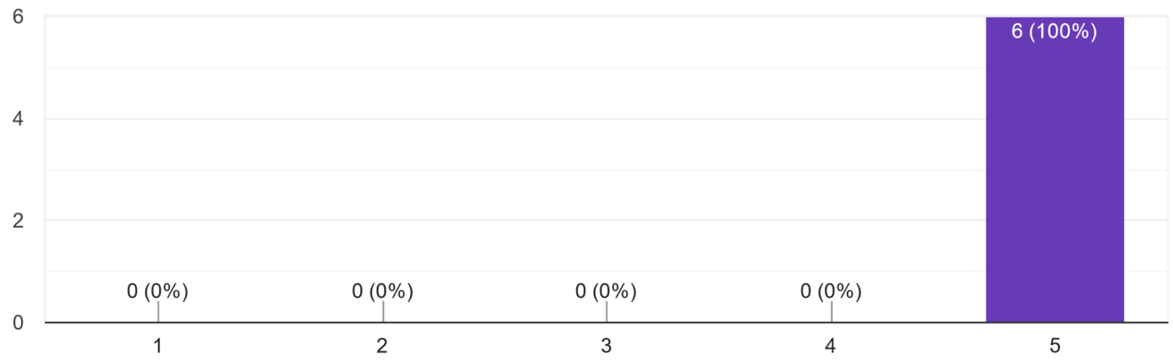
There could be an option to enlarge the screen

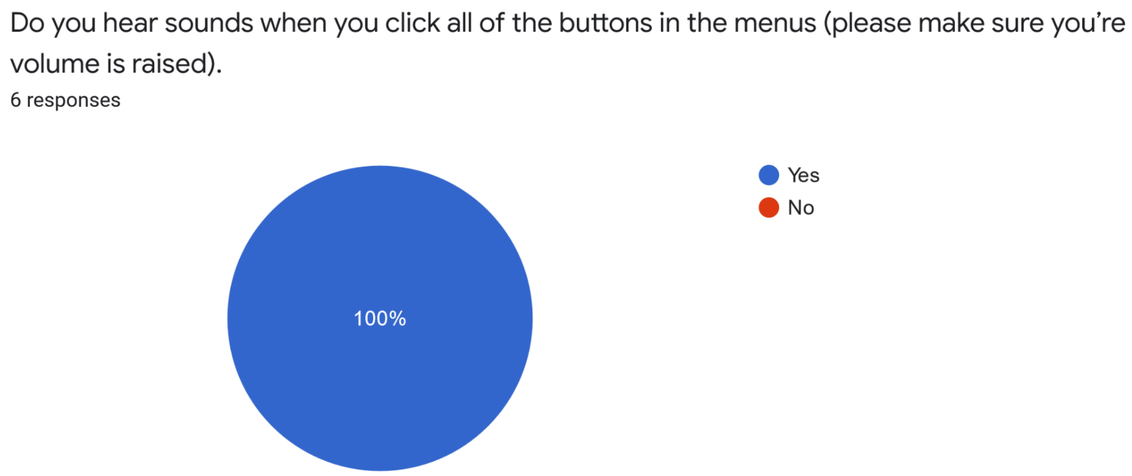
Make the style more modern

You could add a health title above the health bar

When in the main game, are the controls easy to use?

6 responses





Do you hear sounds when you click all of the buttons in the menus (please make sure you're volume is raised).

6 responses

The users from the study were far more positive about the role our application played in performing the tasks they were given. They felt that the tasks were easy to perform and each member of the sample was able to complete the provided tasks successfully. That said, the users did find room for improvement in the application: some users recommend implementing a splash screen of instructions to game as not all users would read the user manual of the application.

The users were also asked about their general thoughts about the idea as a whole. They were quite positive in their thoughts about the idea. They felt it was an effective way of learning new languages via gaming and that it was original.

Overall we found the process of this user study to be quite beneficial to the development process. It served as verification of the existing implementation of the application as well as scope for potential improvements in a future expansion of the Idea.

5.3 Learning Outcomes

In retrospect, we believe that a more Test-Driven Development strategy would have assisted our attempts to develop effective testing suites and appropriate code coverage percentages. We felt that our design decisions hampered this in some way, and it is certainly a far more important concern than we gave it credit for. Moving future, this is unquestionably a problem we'd like to address and better.

6. Future Work

6.1 Future Work / possible expansion of project

As a gaming education application there would be quite a lot of room for expansions and additional features. For our application we could add new forms of games based on the topic selected instead of having just one game, this would further focus on other forms of learning and allow the user to be more immersed.

Our first selected languages were based on our experience with the languages, future expansion would include adding more supported languages to be learned to our application. This would also open up the door to adding more support for teaching more than vocabulary for example teaching phrases or spelling. This would require creativity in the games aspect.

Updating the way we currently store our data such as sprites to a sophisticated database. This would allow for easier handling and updating of the data.

6.2 Conclusion

We made plenty of mistakes along the road, and there are some issues that we wish had been addressed earlier in the development process. However, we believe that this has been a great experience with a variety of learning outcomes to be digested, and that it is a good reflection of the knowledge we have received throughout the course of our undergraduate degree.