**PROGRAM DESIGN METHODS FINAL PROJECT :**

**ANIMO**



**Student Informations:**

Fadhlan Muhammad Razan / 2440047463

**Class Informations:**

Class: L1BC

Lecturer: Ida Bagus Kerthyayana / D5757

**BINUS UNIVERSITY INTERNATIONAL**

**SCHOOL OF COMPUTER SCIENCE**

**2020**

**Chapter I: Introduction**

In the first few months of Binus University study, sudents were taught about python programming language for their programming courses. Based on the curriculum we as students are able to use python alongside with its external libraries. Students then were given with a final project to create anything by their own choosing. Students were given creative freedom to create anything using any external libraries as long as they use python and that they create something that incorporate at least the use of primitive data, instance variables and objects, custom-built classes & methods and detailed code commenting.

After the first time that the students are told about their final project, I started to brainstorm some idea that I could use for my final project. As a fan of dark souls and melee game in general, I came up with creating a 2-D platforming / sidescrolling game that uses a melee system as its primary combat system. I would then quickly start on designing on the premise and base look for the game before then starting to try and implement all those design and make it to an actual game application.

The project was started at the 10th of December 2020. I used Visual Studio Code for my IDE. The final projec, its codes and assets could be found in my GitHub, <https://github.com/Bumskee/ANIMO-Game-Project>.

**CHAPTER II: PROJECT SPESIFICATION**

**Project purpose:**

To create an entertainment product that could be enjoyed by people that used it. To learn python programming and the process of creating and designing a functional video game.

**Project audience:**

People that plays video games moderately or more. People that would like to dissect the innerworking of how a video game was made.

**Project Aim:**

To create a functional melee system that would work in the context of a platformer game that uses the element of the combat and platforming in harmony to create a pleasant gaming experience complete with enemies that can roam a platform and then detect the player when they come to close to the enemy. The game would then should have different screens for the start menu, main game application, pause menu, game over screen, and a final screen that would show when the player has meet all the requirements for their game completion.

**Project Requirements:**

* Functional melee system that uses hitbox spawning and despawning based on the animation frame of the characters
* A responsive control that felt good on the player hands
* Working enemy ai that could detect and run toward the player after detection
* Decent level designs for the player to experience.

**CHAPTER III: SOLUTION DESIGN**

**1. OVERVIEW**

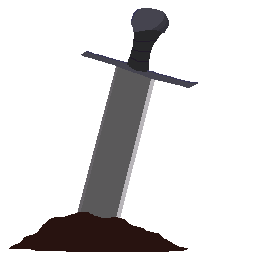
****

Image 1. “Animo” logo

“Animo” is a 2-D melee based platforming sidescrolling video game that challenges players ability in traversing areas and platform while still also focusing on timing and positioning in the combat gameplay.

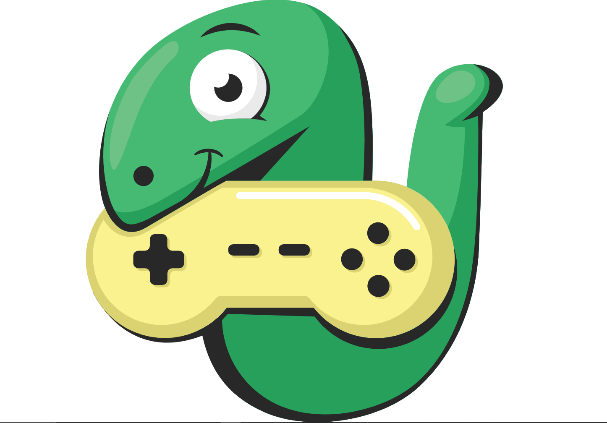


Image 2. Arcade logo

(<https://arcade.academy/_images/arcade-logo.svg>)

“ANIMO” is programmed in Python 3.8.5 and using the arcade library.

The only external library that is used on the project is only the arcade library. It is used for rendering graphics / images to the screen while also has nifty features useful for creating system in the nature of games.

**2. PROGRAM FILE DEPENDENCIES**

The program is not dependant on any other python files. All of the classes, objects, functions are all on the one ANIMO.py file. It is although dependant on the files located on the Assets folder as all of the sound and images are located on said folder.

**3. BASIC IDEA FOR CREATING THE GAME**

In a simplified form, to create the game using the arcade library, It is going to use custom classes that inherits from arcades built-in sprite class to instantiate an object that would store every single individual frame needed to form its animation.

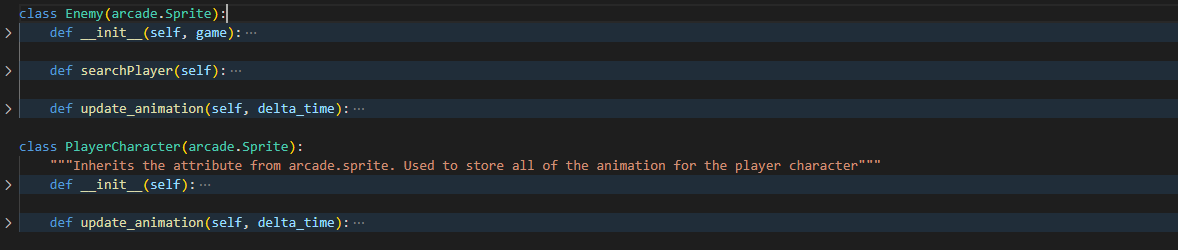


Image 3. Class for the game characters

Then add other necesarry methods or attributes depending on how it is wanted to behave later in the game. Then create another class that inherits from the arcades built-in view class to then store all the sprites, game logics, and draw images to the screen.

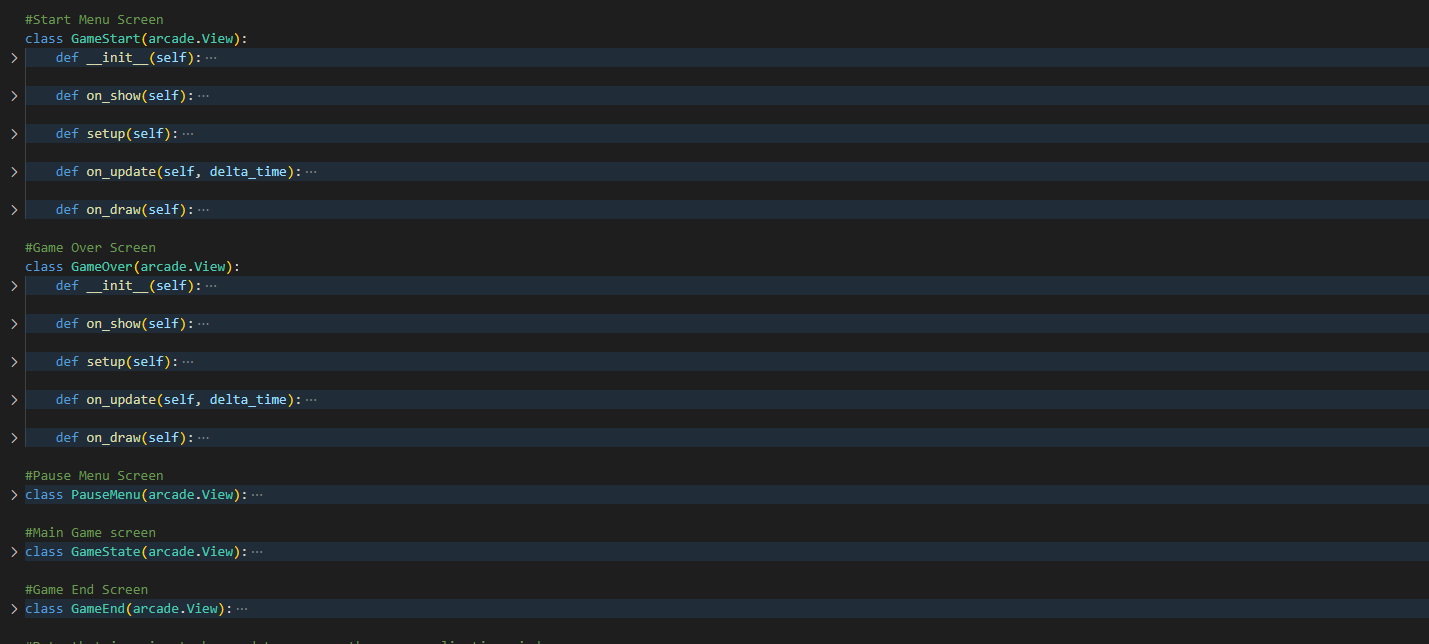


Image 4. Class for the game screens

**4. Layers**

Arcade view class have 2 methods that behaves like a loop that runs the program. The on\_draw() method is run after the on\_update() method. While the on\_update primarily focuses on the games logic, the on\_draw() method is run to control which sprites that is going to be drawn first.

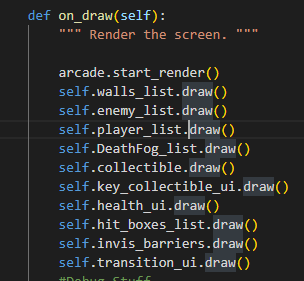


Image 5. Ordering the sprites in the on\_draw() method

Each sprite object in the game can be separated with arcades built-in SpriteList object. By sorting all of the sprites into the sprite lists, we can then use the on\_draw() method to specify which sprite list is going to be drawn first and then be stacked by the next drawn sprite list and so on.

**5. CHARACTER CLASSES**

|  |
| --- |
| PlayerCharacter |
| + Health: int  + character\_face\_direction: int  + animatonTextureFrames \* : int  + playerStates \* : bool  + weaponHitbox\* : object  + (attributes from super()\*) |
| + update\_animation() |

**Information for \*:**

1. animationTextureFrames :

Every character class would have different attributes specific for their each animation. For examples: there would be walk\_animation attributes and idle\_animation\_attributes

1. playerStates

This would have different states that the character would be in and then be updated to be processed on the game logic

1. attributes from super():

Since this class inherits Sprite class from arcade, it would also contains all of the attributes from the Sprite class.

The attributes can be seen here: [https://arcade.academy/arcade.html#arcade.Sprite](https://arcade.academy/arcade.html%23arcade.Sprite)

There is also a class for the enemy. It is similar to the PlayerCharacter class with the difference of less animationTextureFrames attributes and a new method called SearchPlayer()

|  |
| --- |
| Enemy |
| + Health: int  + character\_face\_direction: int  + animatonTextureFrames \* : int  + playerStates \* : bool  + weaponHitbox\* : object  + (attributes from super()\*) |
| + update\_animation()  + SearchPlayer() |

**6. ENEMYS AI**

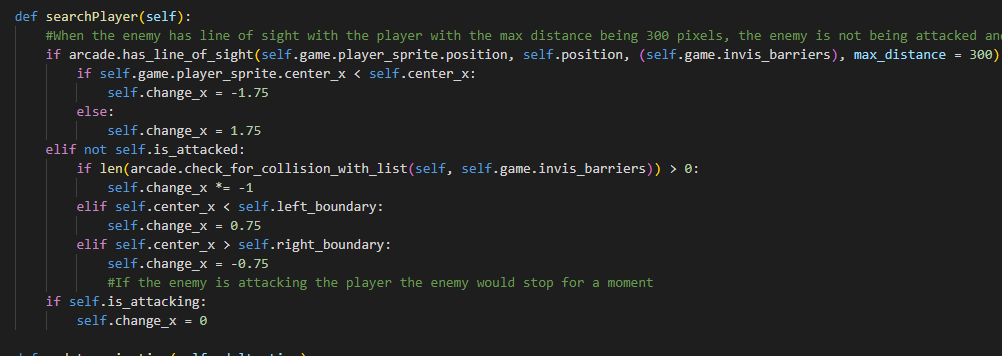
In the enemys class, there is a method for searching the player and their patrol/walk patterns.

Image 6. Enemys AI

Their patrol/walk pattern is just going to be them roaming a platform from side to side according to the predetermined boundary. Their left/right boundary is going to be determined when they are initiated at first by their x position on the map and adding or substracting 150 according to which sides boundary.

There is also a fail save in which case that they tried to walk of an edge of a platform due to bad positioning or level design. We could then place an invisible boundary that the enemy cant walk through during their roams / patrol while the player can just walk to it.

The invisibboundary also act as a block to prevent the enemy from seeing the player if they are behind the invisible boundaries to prevent the enemy from simply walking of an edge of a platform when they decide to run towards the player after they see the player.

For detecting the player, arcade already has a built-in feature that returns True or False with the has\_line\_of\_sight() method. The method takes 2 sprites object, barriers object, and max distance. It will then draw a line between those 2 sprite object. If the line is not obstructedby any barrier (barrier being the invisible barrier) object and its length is less than the max\_distance, it will then return True. Then it will be then be instructed to walk to to the player

**8. HANDLING THE COMBAT SYSTEM**

**8.1 Players Melee System**

What the melee system does is spawning an invisible sprite that is going to be used as a hitbox to detect if it comes in contact with an enemy. The hitbox spawn is going to be based on the frame of the attack animation. Every time the player attacks, the frame of the attackAnimation attribute from the PlayerCharacter class is going to be incremented based on the animation.

When it reaches a certain integer, the game would then append a new sprite to the games hit\_box\_list with the position being adjusted relative to the player x and y position.

After it reaches the end of the animation, the game would remove every hit\_box from the hit\_box\_list.

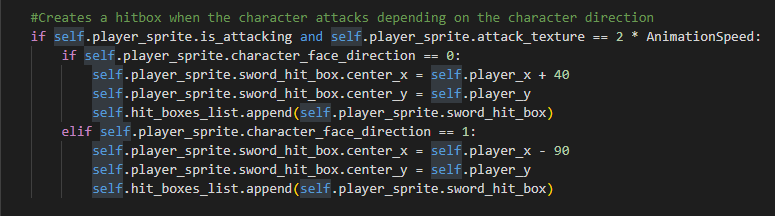


Image 7. Sword hitbox spawning

**8.2 COLLISION DETECTION**

Arcade has their builtin-method of checking collision with their check\_collision\_with\_list() method that takes in spriteObject and spriteList as the argument. It will then return the list of all the object in the spriteList that their hit\_box is in collision with one another.

Whenever the returned list length is more than zero, a set of code would be run that would reduce the attacked objects health by 1 and then launching them based on their relative x position with one another.

For example, if the player x position is less than the enemy x position, that means the player attacked the enemy from the left direction which in turn would make the enemy launched to the right.

**8.2 BEHAVIOUR WHEN BEING HIT**

When the player or enemy was hit, they are going to be launched back from the source of the attack. The way that this is implemented is that whenever they are attacked, their attributes are going to be changed to True for their isAttacked attribute.

And when their attribute for isAttacked is True, they would not be able to move according to their will. The game would then instruct them to move left or right based on their initial attack direction.

If they touches the ground, their isAttacked attribute would then be changed back to True to grant them back their control.

**9. MISCELLANEOUS NIFTY LITTLE FEATURES**

**9.1 SCROLLING EFFECT.**

Since the map is larger then the window size, the camera would have to be adjusted according to the players position. To do that, we could use arcades set\_viewport method to change what the camera capture. By passing the argument startwidth, endwidth, startheight, and endheight respectively.

We would have to specify the max distance between the player x and y position to those viewports widhts and heights. Every time the mainloop is reiterated, it would take the position of the player and then compare it to the current viewports data. It will substract the players x and y with the viewport data. Whenever the result is smaller then the specified max\_distances the camera would readjust itself by adding the result to the new viewport data according on which axis that needed to be readjusted.

**9.2 PARALLAXED SPRITES**

Since the camera is always readjusted, some sprites would also follows the camera. Those being the health UI and the collectible UI. To do that we update the position of those sprites according to the player location. By removing it and readding them based on the player location every main loop iteration.

**9.3 BUTTONS AND GUI**

Buttons could be added to the using arcade.gui Buttons classes. One of the button types that is used in the game is the arcade.gui UIImageButton class.

|  |
| --- |
| UIImageButton |
| + normal\_texture: object  + hover\_texture: object  + press\_texture: object  + center\_x: int  + center\_y: int |
| + on\_click() |

It would use the normal\_texture, hover\_texture, and press\_texture attributes as the sprites that would be used for the button every time when the cursor is not in the button, when the cursor is on the button, and when the button is being clicked respectively. After that it will use the center\_x and center\_y as the position of the button.

The on\_click() method is going to be called whenever the player would click on the button. The on\_click() method would be a custom method that we could make specific to be functionability of said button

**9.4 Sound Loader and Player.**

To load a sound we use the arcades built-in sound class. We intantiate the object with the sound file directory. The we could use the play() method with an optional looping: bool argument. It will then play the sound and return itself.

To stop the sound we can use the stop() method and passing the sound that we wanted to stop as the argument.

**CHAPTER III: PROJECT SCREENSHOTS**



Image 8. Player character is attacking the enemy

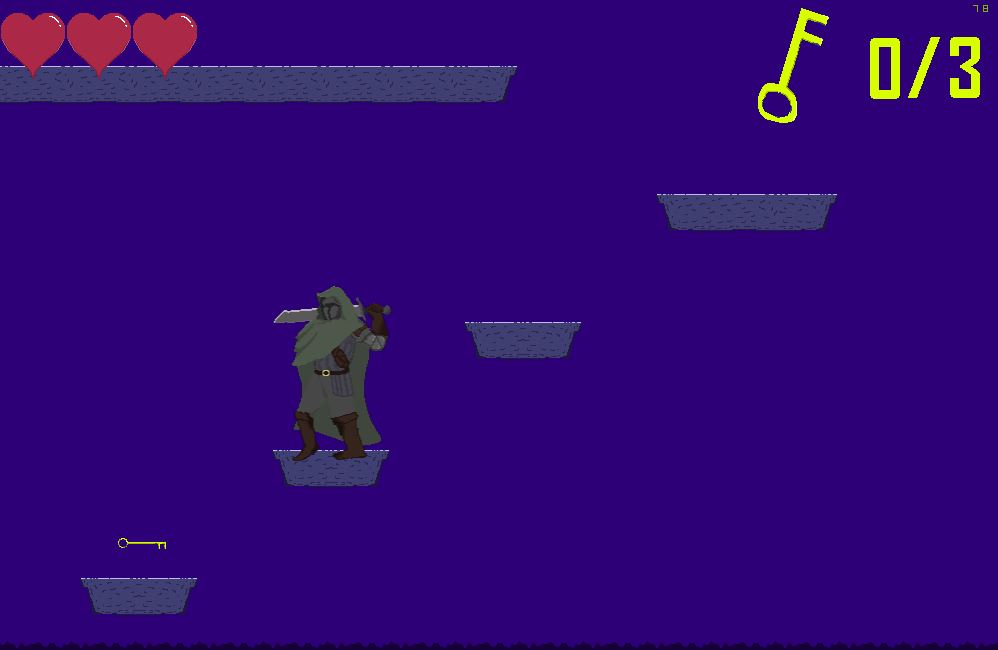


Image 9. Player character is exploring the map

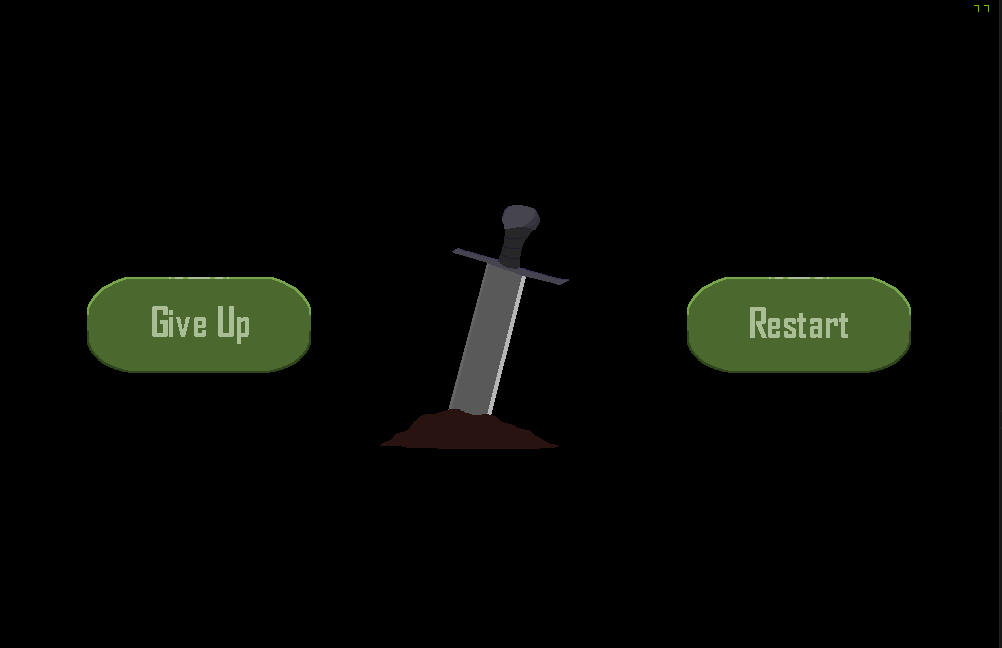


Image 10. Game over screen of the game

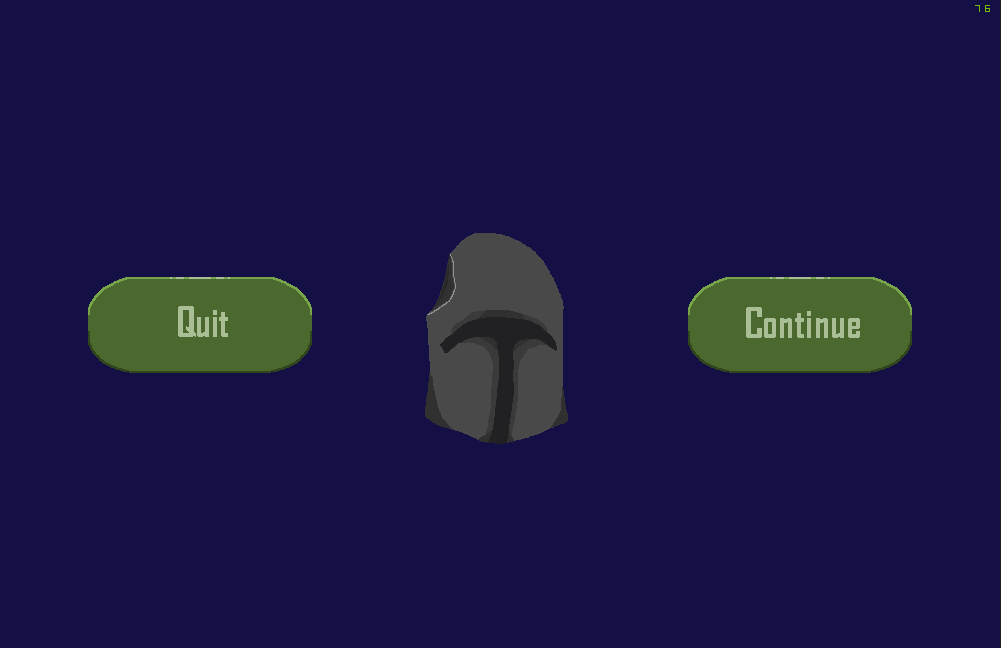


Image 11. Pause Menu screen of the game