

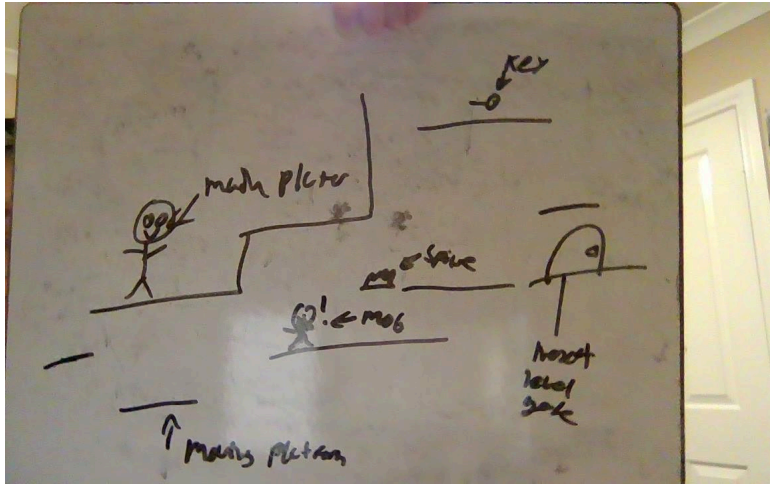
## Assessment 3: Game 6 States

### Planning and Development Process: Concept, Research/inspiration and Design:

The type of game I have chosen for this assignment is a 2D platformer as it allows for various save data from in-game items to levels and player hotkeys. I want this game to have a similar playthrough as Celeste. This game is fast-paced with little forgiveness for errors with satisfying movement. (Celeste, n.d.).

While implementing game features from Ghostrunner taking inspiration from the mobs in the game specifically the sniper and suicide bomber(Wikipedia, 2022). Considering all these aspects leaves me with a fast-paced 2D platformer containing obstacles, mobs and a simple progression system.

#### Basic design

Design of general game plan:	As I want the game to feel fast-paced with a similar feel to Ghost Runner and Celeste, I will design my game based on levels with keys required to move on to the next level.
Game Design General Photo	

### Usability Testing and Results:

#### Game Loop

As explained above, the game loop is based on collecting keys to 'unlock' the door to the next level. The player will encounter various obstacles from enemies to moving platforms. If all the keys have been collected on the level the door would be unlocked otherwise it would stay closed.

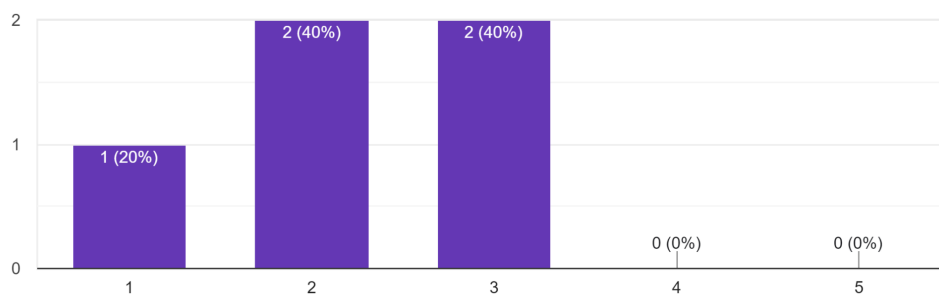
For testing purposes, I made a Google form inquiring about the following topics of my game

- [How easy was the game to understand on a scale from one to five](#)
- [Was the game fun if not why?](#)
- [What Features Need Improving](#)
- [What Did the Game Lack](#)
- [How difficult was the game on a scale from one to five](#)

## Question 1 Results and Implications

The statistics for the first question suggest that a majority of the players found the game loop hard to understand. Most people rated the game as a 2 or 3 in terms of understanding the game loop.

on a scale to 1-5 how easy was the game goal to understand  
5 responses



To solve this I added labels to print instructions for the player from how to complete the level to hotkeys based on the save states of the user. This was done by the following code

```
Python
extends Label
#Will print key bind instruction based on saved hotkey
func _ready():
    var dash = SaveSettings.config.get_value("keybinding", "pause")
    if dash != "Escape":
        self.text = "Press %s to pause"%(dash)
```

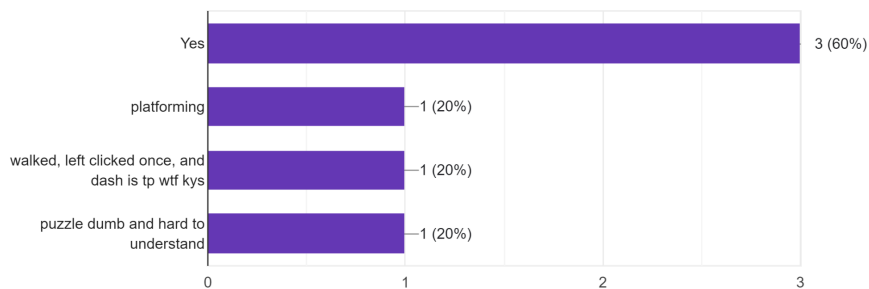
This calls the save setting with the keybinding and pauses to get the current hotkey for the save state. Information was given like this throughout the game to improve player experience regarding information.

## Questions 2, 3 and 4 Results and Implications

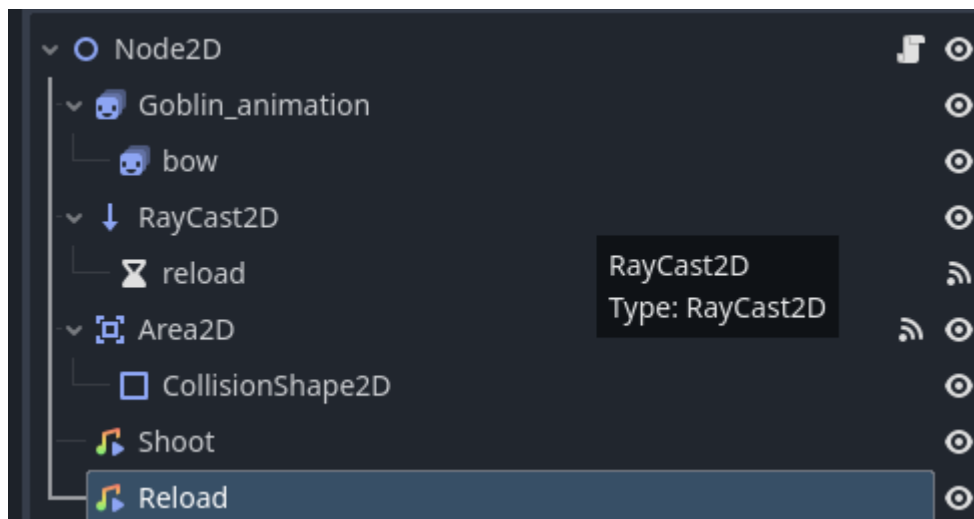
Due to the similarity of the next three questions they were all addressed in the same section.

50% of people enjoy the game, the other 50% showed dissatisfaction with some of the components of the game as shown below.

Was the game fun if not why?  
5 responses



The responses to the next question hinted at why only 50% of players enjoyed the game. Out of the responses a few responses spoke out to me. From “Enemy variation”, “the game difficulty” and the player’s dash. To add more variety to the game I introduced a bow-shooting goblin mob which would shoot the player every so often.



This goblin will send a ray cast until it detects the group “player” or the Player. Once it detects the player it adds child node ‘Arrow’ which moves in the direction of the player and if the Arrow collides it kills the player and the timer reload is called.

```
Python
func _physics_process(_delta):
    #Setting target to player a raycasting to player
    if target!= null:
```

```

        var angle_to_target: float =
            global_position.direction_to(target.global_position).angle()
            raycast.global_rotation = angle_to_target

        if raycast.is_colliding() and
            raycast.get_collider().is_in_group("Player"):
            bow.rotation = angle_to_target
            _playshoot()

```

Here is a snippet of the enemy's script and how this detection works. If the target is null or the Raycast is not colliding then the Raycast will move to the player. This is effectively like an area 2D detecting everywhere just a Raycast. Though this does look cleaner and ensures the enemy can't detect a player behind it makes the mob's arrows significantly harder to avoid by using the alternative method of an Area 2D.

Problem	Solution
The new mob caused the level system to break whenever the player was killed by it.	My global variables were not updated when the mob killed the player. To solve this I called the variable to change variables.

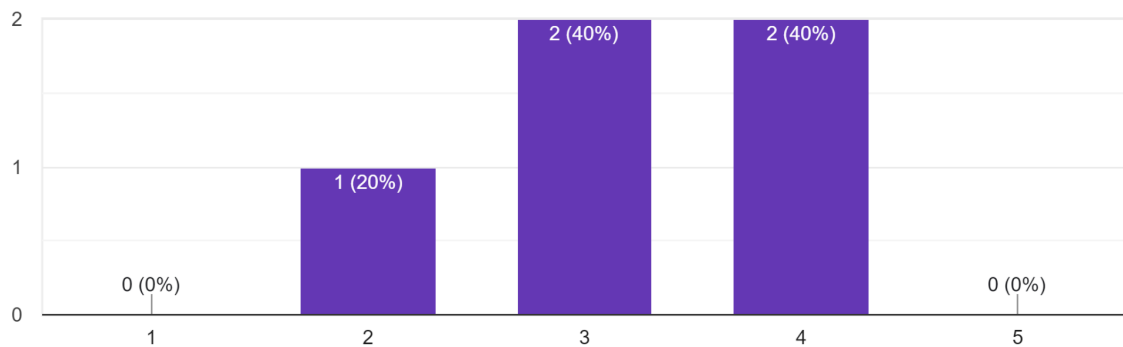
To solve the dash I changed how the movement would operate to affect the velocity of the player additionally adding tween nodes to give a ghost effect when the dash was clicked. The game is 'too hard' used save states addressed in [question 5](#)

## Question 5 Results and Implications

In regards to feedback on the difficulty of the game, more than 50% of people express their struggles in completing the game.

How difficult the game 1 being easy 5 being hard

5 responses



To solve this I added complex save states which would check the amount of deaths per level even if the game is closed and until you complete the level mobs would slowly become easier. An example of this is the Goblin Bow shooter above whose reload relies on a timer. This Save state would slowly change this reload timer over deaths. Setting the saved state as an autoloader lets it be used as the variable for the mob's attack speed.

Python

```
func _check_difficulty_complexSave():  
    #Checks how many times you die to change the difficulty complex save  
    states  
    print("Checking Difficulty", "Deaths: ", SaveFile.g_data["Deaths"])  
    if SaveFile.g_data["Deaths"] ==10:  
        SaveFile.g_data["bomb_attack_speed"]=30  
        SaveFile.g_data["bomb_speed"]=50  
        SaveFile.g_data["shooter_bow_attack_speed"]=2  
        SaveFile.save_data()  
    if SaveFile.g_data["Deaths"] ==20:  
        SaveFile.g_data["bomb_attack_speed"] =100  
        SaveFile.g_data["bomb_speed"]=40  
        SaveFile.g_data["shooter_bow_attack_speed"]=5  
        SaveFile.save_data()  
    if SaveFile.g_data["Deaths"] ==30:  
        SaveFile.g_data["shooter_bow_attack_speed"]=10  
        SaveFile.save_data()
```

This is a global function '`_check_difficulty_complexSave`' is called on player deaths, it will then check the current deaths and then change the mob speed and attack speed. The benefit of coding it through multiple if statements allow the mobs to be changed exactly to how I want with fixed values. Instead, though One check could have been done to check if the death increases and then slowly increment the change for the mob. This would have had shorter code initially but a lack of flexibility meant the code would take longer to create the desired effects. The way these save states were coded will be explained in the next section.

After editing my code based on feedback through testing the final product was [Evaluated](#)

## Save the file and analyse:

### Types of Save Data

Save data can vary from a variety of different information being stored. From the player's information, states of enemies, objects, game-level stats and hot keys or key binds. The idea of save states is to record this information after the game is closed. In simpler terms it "allows players to prevent the loss of progress in the game"(Wikipedia, 2024) They rely on an "external device [or] extract[ing] the data from the device". (Boyd, 2020) As external methods to save my data are tedious. Regarding Godot and its engine, I will be using either JSON, Config Files or Save Files.

Now that you know what types of saved data can be used I will explain which ones I will use for this assignment. When comparing these different save states in Godot JSON files can be seen as the weakest of the three. While Save Files are easy to implement unlike JSON there are more benefits in using config files. This is because JSON though the file can be read can only contain string values. This means it cannot contain numeric information for the level amount to vectors and volume(Godot Engine documentation, n.d.) Because of this Save Files will be used to store scene information such as the enemy's speed and the current level. However, Save files cannot be viewed manually making them a worse option than JSON files for settings saves. Thankfully Config Files have an integration system for saving key binds and can store numbers as well as string making them perfect for settings. (Godot Engine documentation, n.d.)

In summary, Save Files will be used over JSON due to their flexible nature. Config Files will be used for settings thanks to their integration system as well as being able to store different types of data.

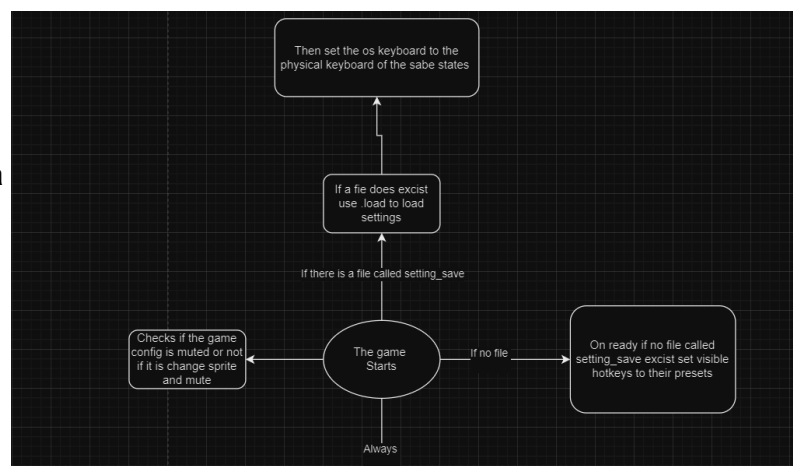
## Programming Save States:

These are the flow charts and explanations for each save time; one will be the setting while the other will be the game loop.

### Settings Save States:[[FlowChart](#)] also on Classroom

The flow chart on the right represents the start/menu and how it checks to Save data.

When the game loads on start it will check If a setting save file exists if it does not it will create one with the default settings. If the



save function is called it will take in the key values and save them.

```
Python
extends Node
const SETTING_SAVE= "user://setting.ini"
var config = ConfigFile.new()
func _ready():
    #SaveFile.g_data["Level"] = 3
    SaveFile.save_data()
    #If file does not exist create one
    if !FileAccess.file_exists(SETTING_SAVE):
        config.set_value("keybinding", "left", "A")
        config.set_value("keybinding", "right", "D")
        config.set_value("keybinding", "Dash", "shift")
        config.set_value("keybinding", "jump", "W")
        config.set_value("keybinding", "pause", "Escape")

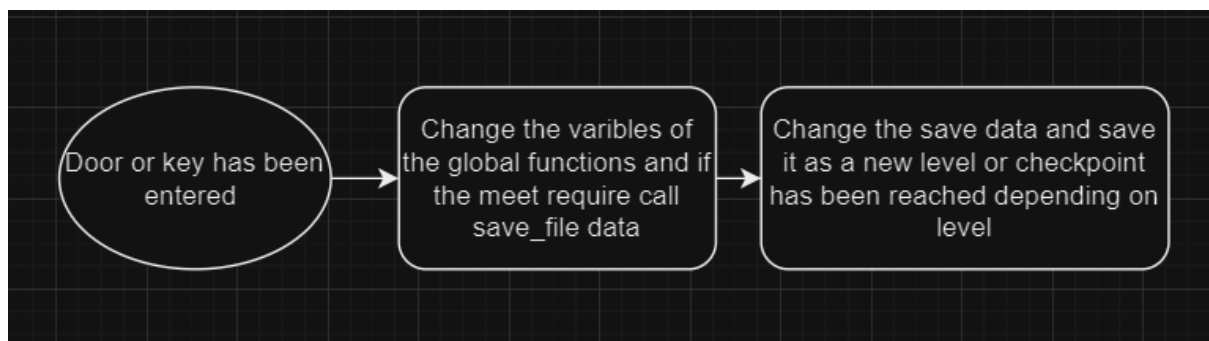
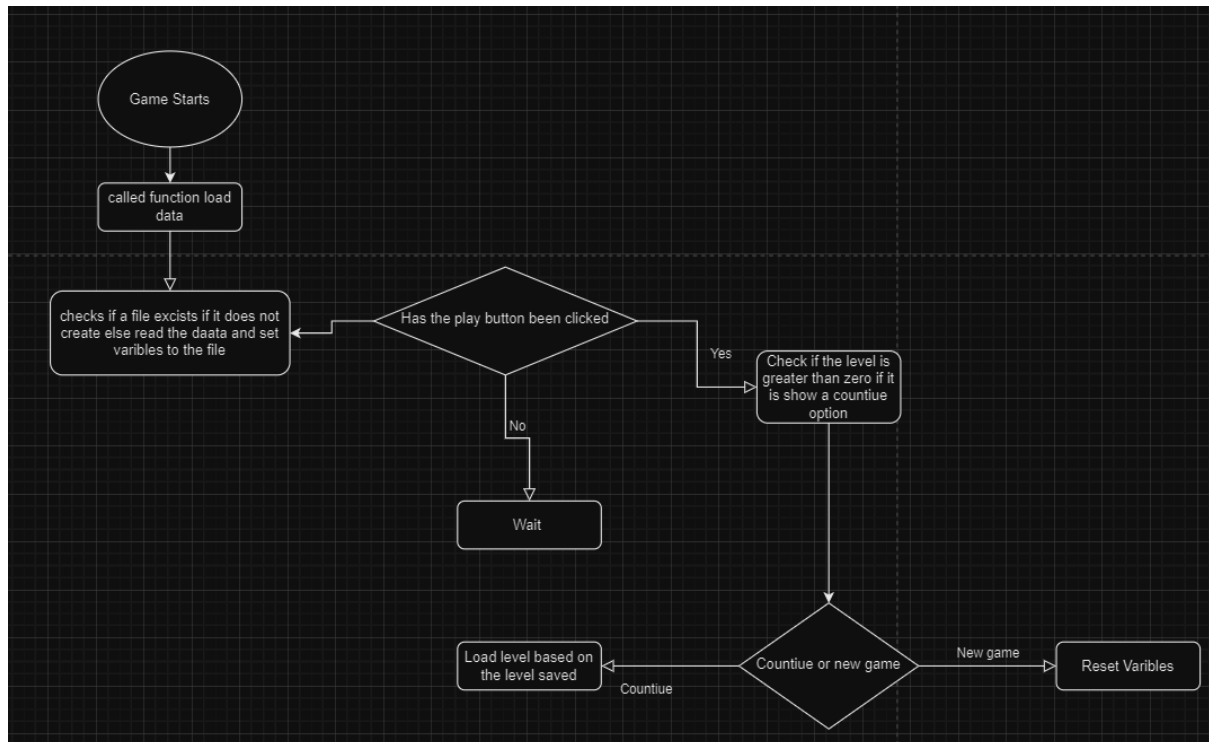
        config.set_value("audio", "Master", 1.0)
        config.set_value("audio", "Music", 1.0)
        config.set_value("audio", "SFX", 1.0)

        config.set_value("video", "Window Mode", true)
        config.set_value("video", "Full-screen", false)
        config.set_value("video", "Borderless Window", false)
        config.set_value("video", "Borderless Full-Screen", false)

        config.set_value("video_FPS", "Frames", 60)
        config.save(SETTING_SAVE)
    else:
        #If file exists load the file
        #config.save(SETTING_SAVE)
        config.load(SETTING_SAVE)
func save_frame(key: String, value):
    #Same as above
    config.set_value("video_FPS", key, value)
    config.save(SETTING_SAVE)
func load_frames():
    #Same as above
    var frame_amount = {}
    frame_amount=config.get_value("video_FPS", "Frames")
    return frame_amount
```

The save and load for each of the different types of setting types run on a similar code. For saving the value it uses the in-built `set_value` function which writes the information to the file path. For loading the data it makes a list of all the data in that category and returns that list when the autoloader function is called. Alternatively, to make the game runtime shorter these functions for each of the different data types could have been made into when function however this was not done due to a lack of readability.

Basic game loop Save States:[[Flow Chart](#)]or on classroom



```

Python
extends Node
#makes const so file can't be broken somehow
const SAVE_FILE = "user://save_file.save"
var g_data = {
    "Level": 0,
  
```



```

    "coins": 0,
    "PlayerLocation": Vector2.ZERO,
    "Deaths":0,
    "TotalDeath":0,
    "bomb_attack_speed":10,
    "bomb_speed":60,
    "shooter_bow_attack_speed":1,
}
func _ready():
    load_data()
func load_data():
    if not FileAccess.file_exists(SAVE_FILE):
        save_data() # Save the initial g_data to the file
    else:
        # If the file exists, load the data
        var file = FileAccess.open(SAVE_FILE, FileAccess.READ)
        file.open(SAVE_FILE, FileAccess.READ)
        g_data = file.get_var()
        print(g_data)
        file.close()

func save_data():
    #writes to file based on variables
    var file = FileAccess.open(SAVE_FILE, FileAccess.WRITE)
    file.store_var(g_data)
    file.close()

```

This uses the File Access godot function to read a write to a function. As it does not create a config and a JSON the file cannot be read manually and will appear as strange-looking text. This does not affect the player though as this data is not being displayed. If there were more complex save states being used and it required flexibility JSON would have been a better alternative. However, as this data is simple and what's more complex is the decisions made based on the saved data this was not needed.

After making this game code as well as the save states below it was [Evaluated](#)

## Evaluation:

## Usability Testing Results Justification

Overall as a product, I believe I followed the feedback given thoroughly. From adding a complex save system that changes the difficulty depending on deaths to fixing issues with the dash and navigating the levels. However, my process to get feedback could have been improved. Specifically, a few of the questions asked were very similar to questions 2,3 and 4. Combining these questions would have made the survey shorter and made it more focused yielding better results. (Johnson, 2019)

## Whole product and functionality

The concept of my game was rather simple and has been met with a game loop based around collecting keys to open doors as well as obstacles to stop the player. While I have implemented a variety of save states. From game save states to influence mobs based on deaths as well as config file save states for settings. As most of my code is modularized this also allows for more levels and features to be added in the future. Additionally, my game uses the universal fade addon as I wanted to implement something beyond the requirements per the assignment. However, I did not implement a dialogue system like I originally wanted to do this was due to time constraints on the project

## Programming

Overall my save state code is efficient and clean with minimal issues. Some aspects could be changed to make my save states more efficient however that would decrease the readability of the code. Other codes used in the game not in regards to the save state could have been combined and through export checks used only one code instead of five. For example, text displayed throughout the text was not done as it loads more variables increasing the memory unnecessarily.

Though most of my code is modularized some parts of my code do not include the player and the first enemy. By splitting the functionality of these scenes into functions new implementations could be added more easily.

## References

Boyd, C. (2020). A brief history of video game saves and data modification | Malwarebytes Labs. [online] Malwarebytes. Available at: <https://www.malwarebytes.com/blog/news/2020/06/a-brief-history-of-video-game-saves-and-data-modification>. [Accessed 29 May 2024].

Celeste. (n.d.). Celeste. [online] Available at: <https://www.celestegame.com/>. [Accessed 29 May 2024]

Godot Engine documentation. (n.d.). Saving games. [online] Available at: [https://docs.godotengine.org/en/stable/tutorials/io/saving\\_games.html](https://docs.godotengine.org/en/stable/tutorials/io/saving_games.html). [Accessed 10 June. 2024].

Godot Engine documentation. (n.d.). ProjectSettings. [online] Available at: [https://docs.godotengine.org/en/stable/classes/class\\_projectsettings.html](https://docs.godotengine.org/en/stable/classes/class_projectsettings.html) [Accessed 10 June. 2024].

itch.io. (n.d.). *Pixel Art Key by Frontend Pashtet*. [online] Available at: <https://drxwat.itch.io/pixel-art-key?download> [Accessed 12 May 2024].

itch.io. (2020). *Simple Dungeon Crawler 16x16 Pixel Art Asset Pack*. [online] Available at: <https://o-lobster.itch.io/simple-dungeon-crawler-16x16-pixel-pack>. [Accessed 11 May 2024].

itch.io. (n.d.). *Dungeon Platformer Tile Set (Pixel Art) by David G.* [online] Available at: <https://incolgames.itch.io/dungeon-platformer-tile-set-pixel-art>. [Accessed 20 May 2024].

itch.io. (n.d.). *Pixel UI buttons by TotusLotus*. [online] Available at: <https://totuslotus.itch.io/pixel-ui-buttons> [Accessed 24 May 2024].

Johnson, H. (2019). 10 tips to improve your online surveys | SurveyMonkey. [online] SurveyMonkey. Available at: <https://www.surveymonkey.com/curiosity/10-online-survey-tips/>. [Accessed 11 June. 2024].

Pixabay (2021). *Running 1*. [online] Pixabay. Available at: <https://pixabay.com/sound-effects/running-1-6846/> [Accessed 9 June. 2024].

Pixabay (2022). *Money Pickup 2*. [online] Pixabay. Available at: <https://pixabay.com/sound-effects/money-pickup-2-89563/> [Accessed 9 June. 2024].

Pixabay (2022). *mouse click*. [online] Pixabay. Available at: <https://pixabay.com/sound-effects/mouse-click-104737/> [Accessed 9 June. 2024].

Pixabay (2021). *Whoosh Transitions SFX 01*. [online] Pixabay. Available at: <https://pixabay.com/sound-effects/whoosh-transitions-sfx-01-118227/> [Accessed 9 June. 2024].

Pixabay (2022). *Bow Loading*. [online] Pixabay. Available at: <https://pixabay.com/sound-effects/bow-loading-38752/> [Accessed 9 June. 2024].

Pixabay (2021). *Goblin Death*. [online] Pixabay. Available at: <https://pixabay.com/sound-effects/goblin-death-6729/> [Accessed 9 June. 2024].

Pinterest. (n.d.). *Dungeon BG | Pixel art games, Pixel art design, Pixel art characters*. [online] Available at: <https://www.pinterest.com.au/pin/408138784957380983/> [Accessed 9 June. 2024].

www.deviantart.com. (2016). *Pixelated Dungeon Background by CakeEaterGames on DeviantArt*. [online] Available at: <https://www.deviantart.com/cakeeatergames/art/Pixelated-Dungeon-Background-625691333> [Accessed 9 June. 2024].

Pixabay (2022). *I am dreaming of a Final Fantasy menu kinda thing*. [online] Pixabay. Available at: <https://pixabay.com/sound-effects/i-am-dreaming-or-final-fantasy-menu-kinda-thing-29173/> [Accessed 9 June. 2024].

PixaBay (8AD). *dorm door opening*. [online] Pixabay. Available at: <https://pixabay.com/sound-effects/dorm-door-opening-6038/> [Accessed 9 June. 2024].

itch.io. (n.d.). *Simple Wooden Bow and Arrows by Arydian.G7*. [online] Available at: <https://arydian.itch.io/simple-wooden-bow-and-arrows> [Accessed 9 June. 2024].

Pixabay (2023). *Lost Alone Mysterious - Gentle Instrumental*. [online] Pixabay. Available at: <https://pixabay.com/music/ambient-lost-alone-mysterious-gentle-instrumental-175368/> [Accessed 9 June. 2024].

Pixabay (2024). *Rain sound*. [online] Pixabay. Available at: <https://pixabay.com/sound-effects/rain-sound-188158/> [Accessed 9 June. 2024].

Wikipedia. (2024). *Saved game*. [online] Available at: [https://en.wikipedia.org/wiki/Saved\\_game#:~:text=Game%20designers%20allow%20players%20to](https://en.wikipedia.org/wiki/Saved_game#:~:text=Game%20designers%20allow%20players%20to) [Accessed 29 May 2024].

Wikipedia. (2022). *Ghostrunner*. [online] Available at: <https://en.wikipedia.org/wiki/Ghostrunner>. [Accessed 29 May. 2024].