CGP503 AE2 Report

Q13068296 – James Coyle

Contents

[Design 2](#_Toc503530507)

[Main Menu Status Control 3](#_Toc503530508)

[Flowchart 3](#_Toc503530509)

[UML 3](#_Toc503530510)

[Player Control 4](#_Toc503530511)

[Flowchart 4](#_Toc503530512)

[UML 4](#_Toc503530513)

[Enemy AI Control 5](#_Toc503530514)

[Flowchart 5](#_Toc503530515)

[UML 5](#_Toc503530516)

[Projectile Control 6](#_Toc503530517)

[Flowchart 6](#_Toc503530518)

[UML 6](#_Toc503530519)

[Loot Control 6](#_Toc503530520)

[Flowchart 6](#_Toc503530521)

[UML 7](#_Toc503530522)

[Implementation and Source Code 7](#_Toc503530523)

[Code Clarity & Nesting 7](#_Toc503530524)

[Variable Naming & Comments 7](#_Toc503530525)

[OO Design 7](#_Toc503530526)

[Testing 7](#_Toc503530527)

[Test Plan 7](#_Toc503530528)

[Test Results 8](#_Toc503530529)

[Reflection 8](#_Toc503530530)

[Design 8](#_Toc503530531)

[How it was done 8](#_Toc503530532)

[Challenges 9](#_Toc503530533)

[Hindsight Changes 9](#_Toc503530534)

[Implementation 9](#_Toc503530535)

[How it was done 9](#_Toc503530536)

[Challenges 9](#_Toc503530537)

[Hindsight Changes 9](#_Toc503530538)

[Testing 10](#_Toc503530539)

[How it was done 10](#_Toc503530540)

[Challenges 10](#_Toc503530541)

[Hindsight Changes 10](#_Toc503530542)

[Self-Assessment 10](#_Toc503530543)

[Video Demo 10](#_Toc503530544)

[References 10](#_Toc503530545)

[Appendix 12](#_Toc503530546)

[ Source Code (a) 12](#_Toc503530547)

[Management Tap Script 12](#_Toc503530548)

[UI Functions Script 17](#_Toc503530549)

[Combat Script 26](#_Toc503530550)

[Enemy AI Script 37](#_Toc503530551)

[CannonBall Script 43](#_Toc503530552)

[CannonBallEnemy Script 44](#_Toc503530553)

[LootCrate Script 45](#_Toc503530554)

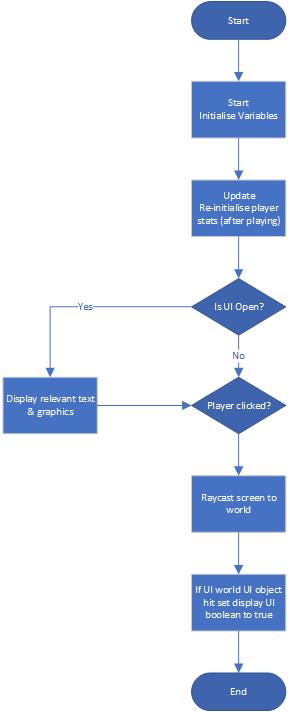
# Design

In relation to designing this project I believe there are 6 main items to be considered.

1. Main Menu status control
2. Player control
3. Enemy AI control
4. Game control
5. Projectile control
6. Loot control

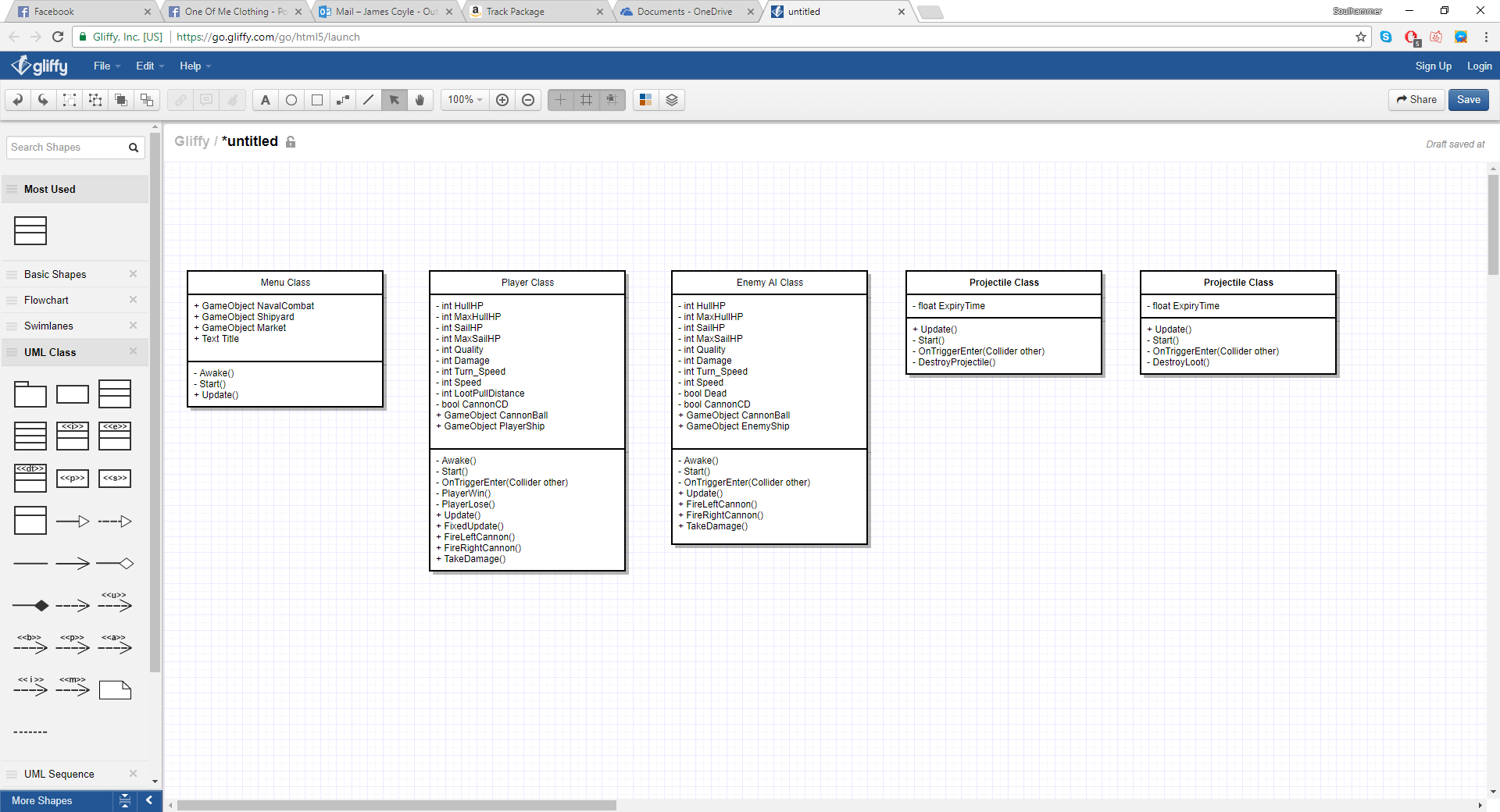
## Main Menu Status Control

### Flowchart



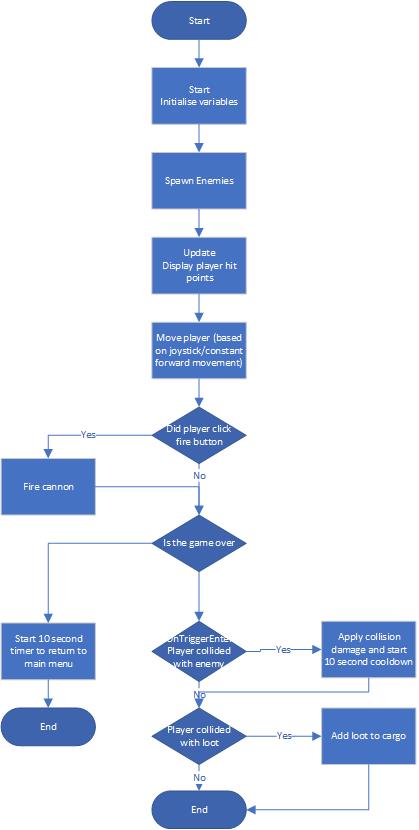
This design was created with in-game world objects having colliders attached to detect any player taps.

### UML

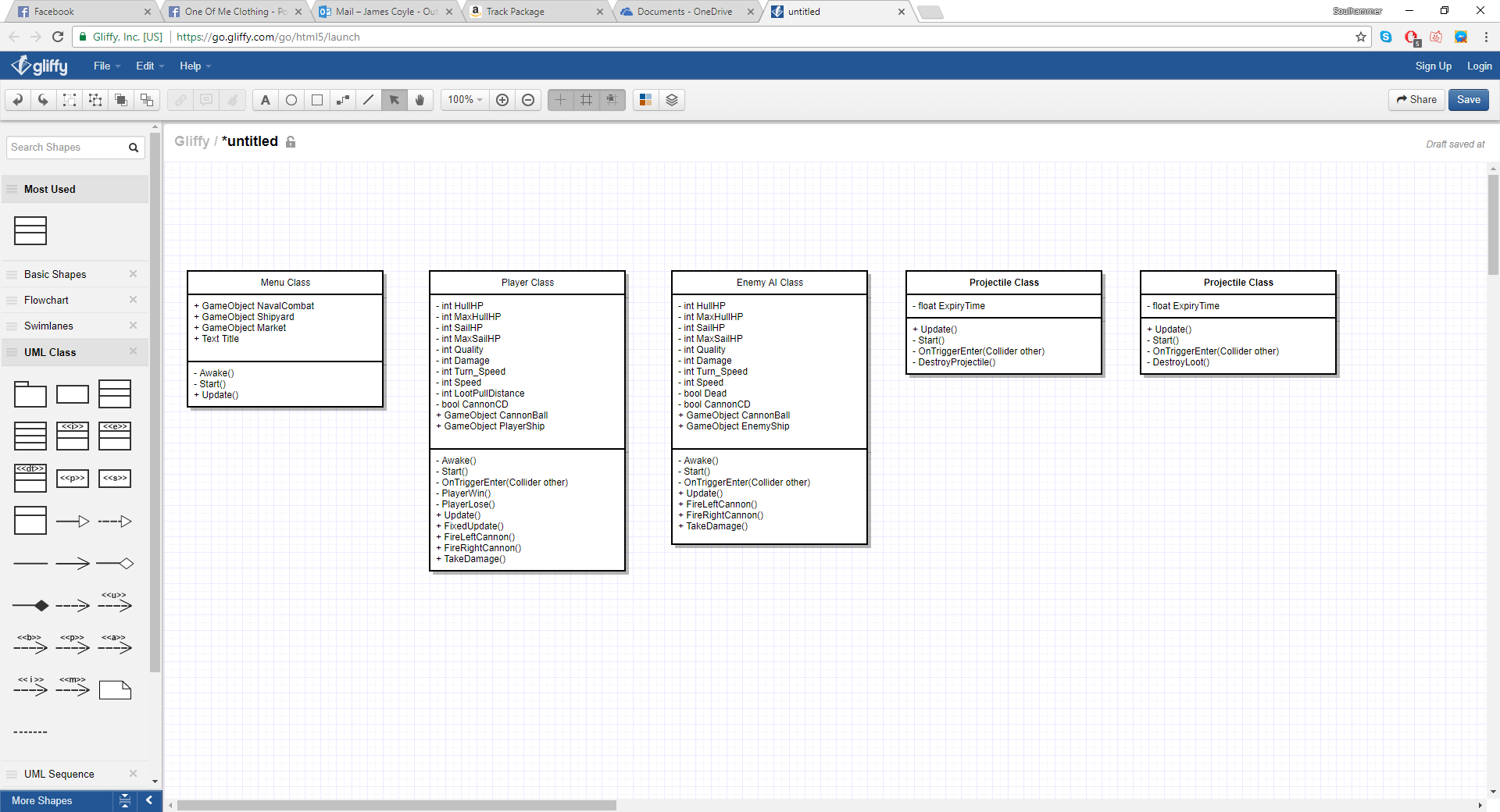
I elected to keep the class as basic as possible for this, in my experience with Games Programming’s last assignment I know that I will end up diverting drastically from this design and that it’s near-on impossible to understand everything you will need before you begin production.

## Player Control

### Flowchart

The player control script will double as the game control (after considering a Game Control script too). I think this is the right course because I don’t want to flood my Unity project with endless scripts if they aren’t necessary and I figure that the player object will never be deleted so this script will always be active.

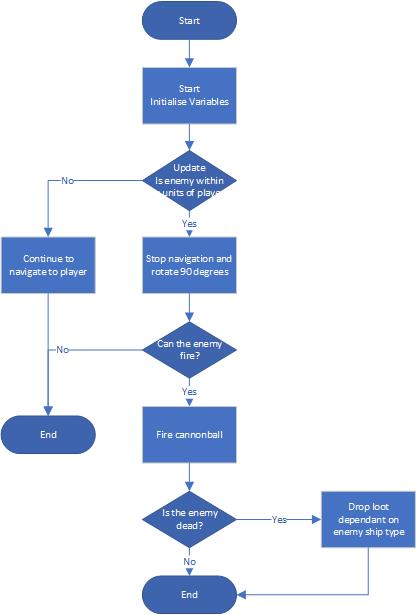
### UML

This class will ultimately be the biggest of the bunch with a lot of the game controls as well as all of the player interactions.

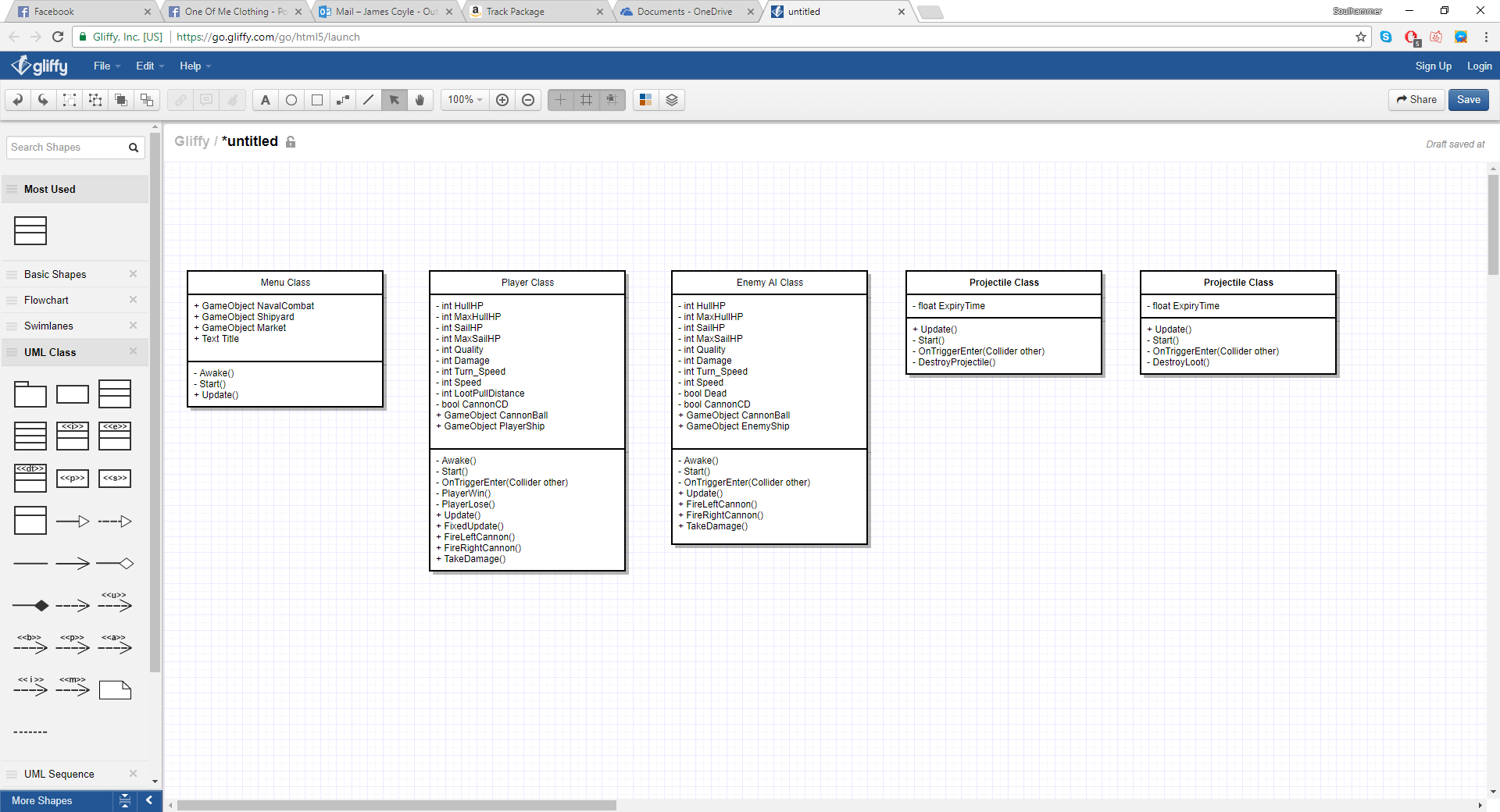
This design also keeps in mind certain brief criteria where appropriate such as loot pull distance, etc.

## Enemy AI Control

### Flowchart

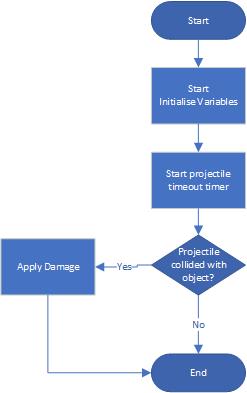
This design has the Unity standard Nav Mesh AI in mind, this is what I’ve had the most experience with so I figure it would be a good place to start

### UML

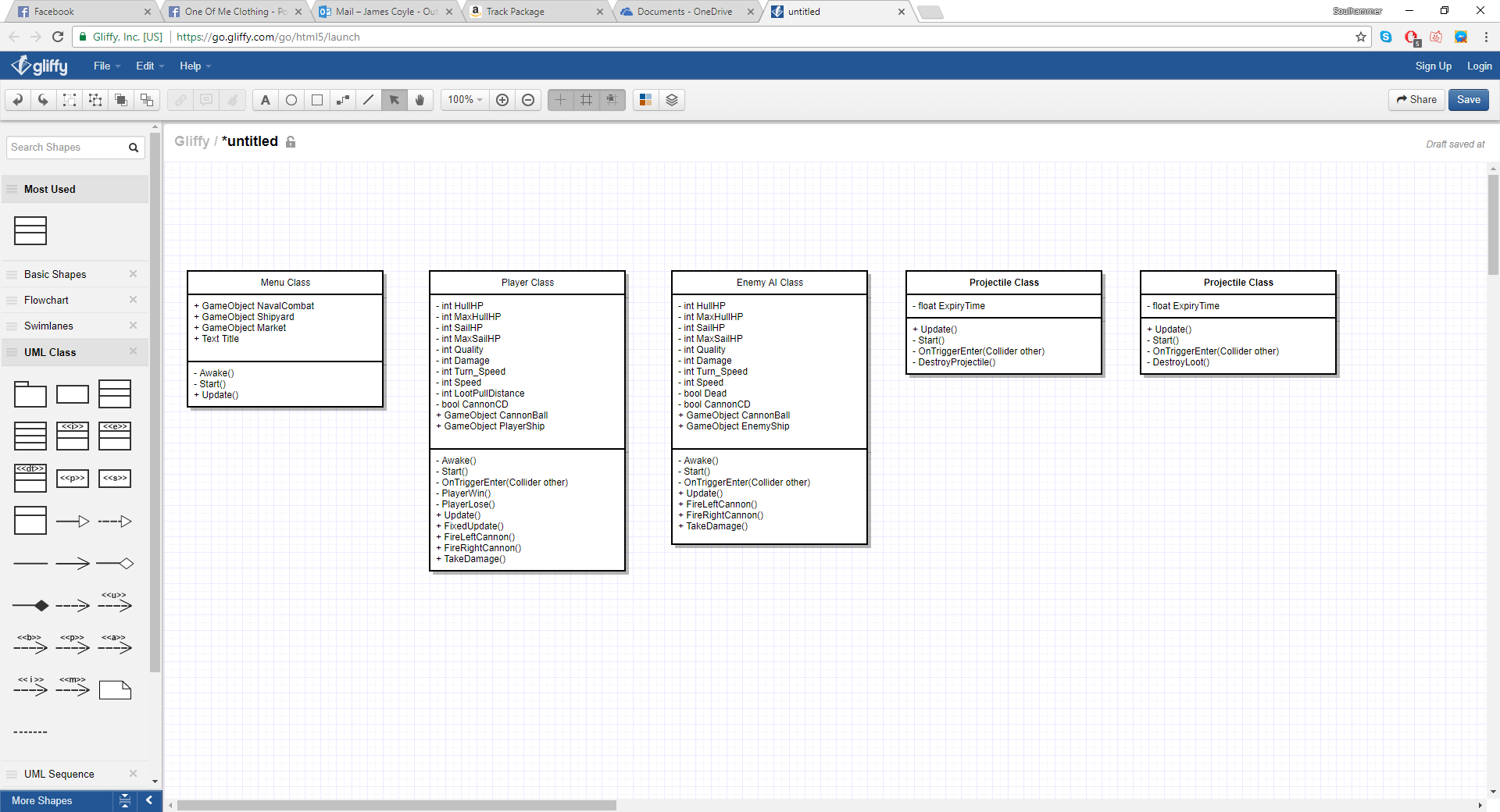
Again, this will be a fairly big script, with all of the enemy AI as well as things like cargo drop chances, animations and local stats involved.

## Projectile Control

### Flowchart

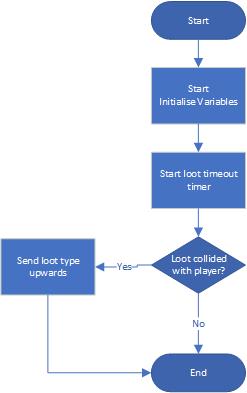
I designed this class to handle all projectiles fired, whether they’re from the player or enemy to allow them to be self-contained units.

### UML

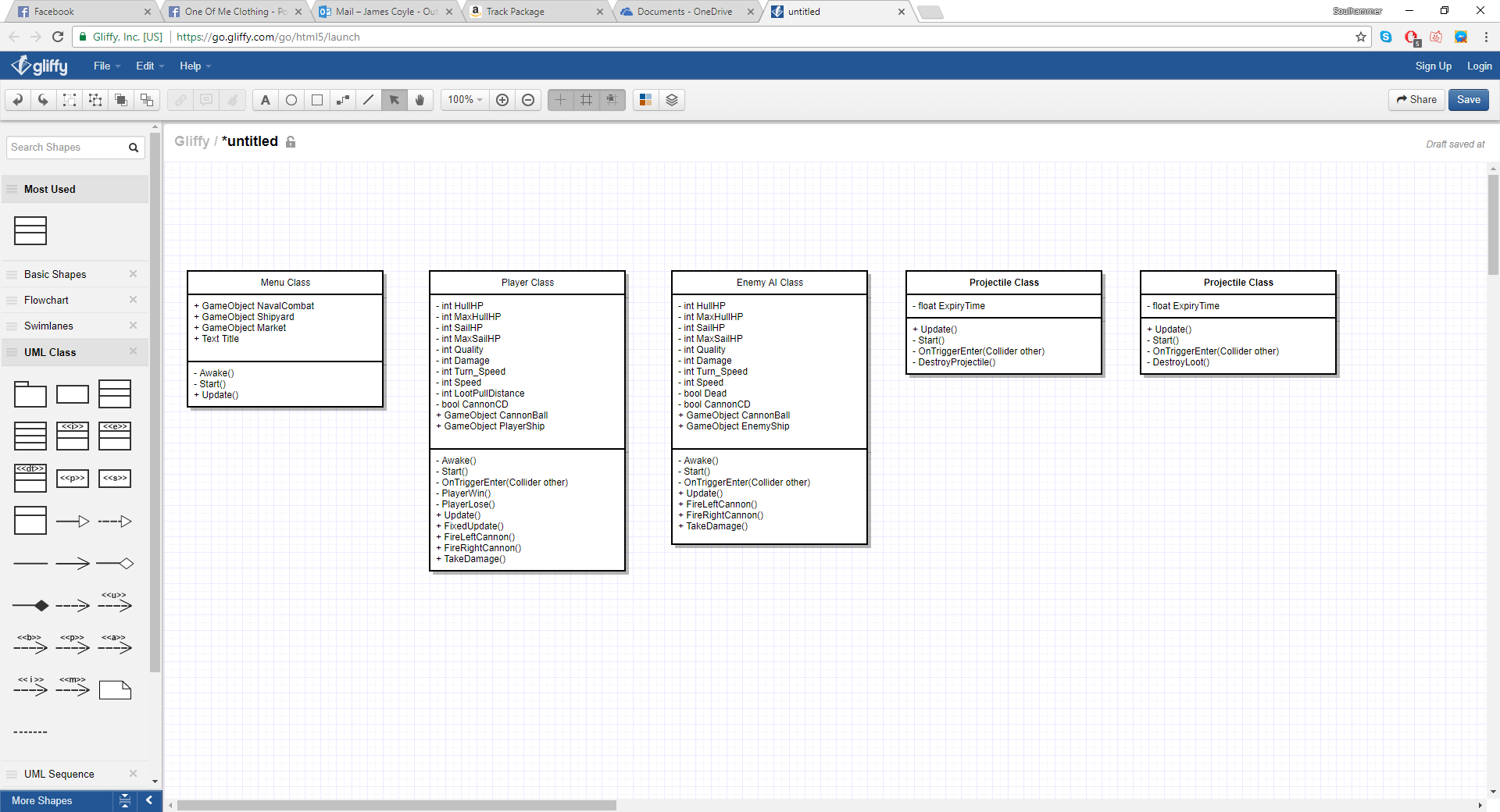
This UML is fairly basic (as is the Loot Class) however it should allow me to do so much.

## Loot Control

### Flowchart

As with the Projectile class, this will allow the dropped loot to have all of the automation it needs to be stand-alone.

### UML

Once again, this is a fairly simple class design but for what’s required it doesn’t need to be too complicated.

# Implementation and Source Code

## Code Clarity & Nesting

See Appendix (a).

## Variable Naming & Comments

See Appendix (a).

### OO Design

See Appendix (a).

# Testing

## Test Plan

The number/letter system below is the feature ID being tested.

1. Main menu
   1. Clicking on Naval Combat
   2. Clicking on Market
      1. Selling items
   3. Clicking on Shipyard
      1. Upgrading stats
2. Naval Combat
   1. Taking damage
   2. Firing cannons
   3. Receiving damage
   4. Turning
   5. Taking sail damage
   6. Colliding with enemy
   7. Hull health reaching 0 or less
   8. Collecting loot
   9. Killing all enemies
   10. Player joystick movement

## Test Results

For the bug severity I am going to use a RAG status (Red, Amber, Green) system.

|  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- |
| Feature ID | Bug | Severity | Cause | Reproduction | Solution |
| 1.a | None | N/A | N/A | N/A | N/A |
| 1.b | None | N/A | N/A | N/A | N/A |
| 1.b.i | None | N/A | N/A | N/A | N/A |
| 1.c. | None | N/A | N/A | N/A | N/A |
| 1.c.i | None | N/A | N/A | N/A | N/A |
| 2.a | None | N/A | N/A | N/A | N/A |
| 2.b | Left cannon was firing in the same direction as the Right cannon. | R | rFire function was using transform.right as the velocity direction. | Press Fire Right cannon button in Sea Raid combat. | Changed transform.right to -transform.right. |
| 2.c | None | N/A | N/A | N/A | N/A |
| 2.d | None | N/A | N/A | N/A | N/A |
| 2.e | Sail damage not being applied. | A | If statement in TakeDamage function was written incorrectly. | Get hit by enemy cannon ball whilst in Sea Raid. | Change the percentage float to 20.0f from 0.2f. |
| 2.f | None | N/A | N/A | N/A | N/A |
| 2.g | None | N/A | N/A | N/A | N/A |
| 2.h | None | N/A | N/A | N/A | N/A |
| 2.i | None | N/A | N/A | N/A | N/A |
| 2.j | Joystick disappeared after opening game file. | G | The mobile input joystick was setting the control input method to keyboard arrows and disabling the joystick. | Play game in the exe file. | No solution required, after researching the script attached there is still a valid and easier input method attached to the joystick class. |

# Reflection

## Design

The design phase for the game was short because I wasn’t as experienced with Unity as I am now. However, I feel like it went well considering the circumstances.

### How it was done

To lay out the design for the game I thought about the scripts which would be necessary for the game to work the way the brief described. I tried to keep it as simple as possible to ensure that my still in development skills in Unity wouldn’t be hindered by an overly complex design.

### Challenges

When designing the game, the main challenge was my inexperience with the game engine overall. I had some idea of what to do but looking back now it was very limited in terms to what was needed for the game itself.

### Hindsight Changes

In hindsight I could have done a lot differently with the design. For example, I could have changed the single-class designs to a structured inheritance design to make the code work better and more efficiently overall.

This in consideration, as I explained before with my limited skills at the beginning of this project I was unable to explore all the possible things that would be needed to complete the game, therefore the design was severely lacking and subsequently I added to the design while in production quite drastically.

I originally fully intended to use 2D sprites for all of my assets, however as production went into full swing I soon discovered that it would be a lot easier and more professional looking to just use 3D assets and art.

## Implementation

The implementation was a steep learning curve for myself as up until now I’ve only created small games with very basic mechanics in Unity and not had to do much with them to get them to work besides the basics.

Keeping this in mind though, I really enjoyed working on this Project once I got into the groove of it and I’m eager to continue developing games on this platform and build on the experience and learning I’ve developed thus far.

### How it was done

The implementation was done through trial and error primarily with a lot of research online when I encountered issues. As I mentioned above the end product was quite drastically changed from the original design due to the feasibility of it.

### Challenges

One of the biggest challenges I had was with the water. I tried and tested many different things, starting with basic sprite quads to then going to the end result as you see now. This is definitely one of the areas that the game falls down, because the water is not endless, and the enemy AI uses a Nav Mesh Agent instead of hand-written tracking code I was unable to achieve the endless-world effect.

Another challenge when implementing the design was transferring the variables used to store things like cargo and health from one scene to another. It took me a long time for figure out how awesome PlayerPref’s actually are and use them extensively and effectively.

One of the biggest challenges was how to take the cannon balls owners damage to the player when the cannon ball collided with the player. I found out that I was able to create an instance of an object, run a script to load damage onto it and then on collision take that damage and apply it to the players health before deleting it.

### Hindsight Changes

With the implementation I would definitely have liked to achieve an open world scenario whereby you cannot just run out of space, as well as this I would love to incorporate extra added features to the game such as a enemy tracking HUD or mini-map feature to allow the player to see enemies easier. But besides this I’m quite happy with the end result, all things considered.

## Testing

I elected to keep my test-plan as simple as possible so that It could cover as much as possible but not go to the extremes of testing each individual variable to see if it works or not.

### How it was done

The testing was fairly easily done, I simply set various variables required for the test to a set amount and launched the game as I would normally and see if everything checked out.

### Challenges

One of the main challenges was testing on mobile. I chose to go with a UI text box to display the stats in both scenes to ensure I would be able to check them live on Android, it took a very long time to implement and test on Android because of the time it takes to build and play the project.

### Hindsight Changes

I would like to have had left more time to test the game more thoroughly, but as I finished the game to a decent standard and then tested it I felt it wasn’t necessary because the bulk of the testing was done in the implementation stage using trial and error.

## Self-Assessment

I would say that confidently I would grade this assessment around a B1 or higher, I say this because the quality of the work is good. I’ve spent a lot of time developing the game and I’m very proud of it. There are very few places it falls down and overall, I feel as though it could legitimately be uploaded and sold on the app-store as is.

# Video Demo

<https://www.youtube.com/watch?v=kAqs-VMnf2A>

(Also in the Zip file containing the Assessment material)

# References

* Game Management Music:
  + http://soundimage.org/fantasywonder/
* Game Battle Music:
  + http://soundimage.org/wp-content/uploads/2017/09/Strangeness.mp3
* Ambient Music:
  + http://soundimage.org/sfx-environments/
* Music ON Image:
  + https://cdn1.iconfinder.com/data/icons/basic-vol-3/16/volume\_sound\_mute\_on\_off\_track-512.png
* Music Off Image:
  + https://cdn1.iconfinder.com/data/icons/basic-vol-3/16/volume\_sound\_mute\_on\_off\_track\_block-512.png
* Splashing Rock:
  + http://soundbible.com/2100-Splash-Rock-In-Lake.html
* Ship sinking:
  + https://www.youtube.com/watch?v=5kEZaBS167A
* Cargo pickup:
  + http://soundimage.org/sfx-ui/
* AuctionSell:
  + <http://www.wowhead.com/sounds/name:auction>

All other objects used have references to their creators in the unity editor’s Project Assets folders & can be found either on the Asset Store or are created by Unity.

# Appendix

## Source Code (a)

### Management Tap Script

using System.Collections;

using System.Collections.Generic;

using UnityEngine;

using UnityEngine.UI;

using UnityEngine.AI;

public class ManagementTap : MonoBehaviour {

//Main Management controller script (Attached to main camera object)

//Various items relating to the in-game UI (In-world and Canvas)

//Canvas Object(s):

public GameObject UI; //UI Canvas objects (Used to hide and display when clicking on the below world objects

//In-world Objects:

public Text UITitle; //Title 3D scene text object

public GameObject Combat; //In-game object for the Combat UI function

public GameObject Shipyard; //In-game object for the Shipyard UI function

public GameObject Market; //In-game object for the Market UI function

//Player AI-controlled ship variables:

private NavMeshAgent nav; //Nav Mesh agent variable for the non-controlled player AI object

public string currLocation; //Current location of the above player AI object

//These booleans are used to check to see if the ship in the menu has made it to the first checkpoint along the path to each destination:

private bool mVia = false; //Boolean used for the play AI route (Market route)

private bool sVia = false; //Boolean used for the play AI route (Shipyard route)

public bool cTar = false; //Boolean used for the play AI route (Starting combat route)

private void Awake()

{

//Initialise variables:

nav = GameObject.Find("MenuPlayer").GetComponent<NavMeshAgent>();

//First initialisation of PlayerPref variables:

if(!PlayerPrefs.HasKey("fDamage")) PlayerPrefs.SetFloat("fDamage", 20.0f);

if(!PlayerPrefs.HasKey("fQuality")) PlayerPrefs.SetFloat("fQuality", 0.95f);

//Set the current location of the AI-controlled player ship to the start:

currLocation = "Start";

}

private void OnApplicationQuit()

{

//When closing the application delete all player save-data:

PlayerPrefs.DeleteAll();

}

[RuntimeInitializeOnLoadMethod(RuntimeInitializeLoadType.BeforeSceneLoad)]

static void OnBeforeSceneLoadRuntimeMethod()

{

//When loading the scene check the PlayerPref settings and initialise them if necessary

if(!PlayerPrefs.HasKey("MHHP")) PlayerPrefs.SetFloat("MHHP", 50.0f); //Max Hull-HP

if(!PlayerPrefs.HasKey("MSHP")) PlayerPrefs.SetFloat("MSHP", 100.0f); //Max Sail-HP

if(!PlayerPrefs.HasKey("Turn")) PlayerPrefs.SetFloat("Turn", 1.1f); //Turn speed of in-game shit (Combat scene)

if(!PlayerPrefs.HasKey("fDamage")) PlayerPrefs.SetFloat("fDamage", 20.0f); //Cannon-ball base damage

if(!PlayerPrefs.HasKey("fQuality")) PlayerPrefs.SetFloat("fQuality", 0.95f); //Combat-scene player Quality

if(!PlayerPrefs.HasKey("Cargo")) PlayerPrefs.SetInt("Cargo", 10); //Combat-scene Cargo max capacity

//Shipyard Upgrade PlayerPref Variables:

if(!PlayerPrefs.HasKey("Sails")) PlayerPrefs.SetInt("Sails", 1);

if(!PlayerPrefs.HasKey("Damage")) PlayerPrefs.SetInt("Damage", 1);

if(!PlayerPrefs.HasKey("Hull")) PlayerPrefs.SetInt("Hull", 1);

if(!PlayerPrefs.HasKey("Storage")) PlayerPrefs.SetInt("Storage", 1);

if(!PlayerPrefs.HasKey("Quality")) PlayerPrefs.SetInt("Quality", 1);

//Initialise all Inventory Items to 0

if(!PlayerPrefs.HasKey("Gold")) PlayerPrefs.SetInt("Gold", 0);

if(!PlayerPrefs.HasKey("Grain")) PlayerPrefs.SetInt("Grain", 0);

if(!PlayerPrefs.HasKey("Fish")) PlayerPrefs.SetInt("Fish", 0);

if(!PlayerPrefs.HasKey("Oil")) PlayerPrefs.SetInt("Oil", 0);

if(!PlayerPrefs.HasKey("Wood")) PlayerPrefs.SetInt("Wood", 0);

if(!PlayerPrefs.HasKey("Brick")) PlayerPrefs.SetInt("Brick", 0);

if(!PlayerPrefs.HasKey("Iron")) PlayerPrefs.SetInt("Iron", 0);

if(!PlayerPrefs.HasKey("Rum")) PlayerPrefs.SetInt("Rum", 0);

if(!PlayerPrefs.HasKey("Silk")) PlayerPrefs.SetInt("Silk", 0);

if(!PlayerPrefs.HasKey("Silverware")) PlayerPrefs.SetInt("Silverware", 0);

if(!PlayerPrefs.HasKey("Emerald")) PlayerPrefs.SetInt("Emerald", 0);

//Save all created PlayerPref variables:

PlayerPrefs.Save();

}

private void Start()

{

//On starting the scene set the player's Hull HP and Sail HP to be their Max values

PlayerPrefs.SetFloat("HHP", PlayerPrefs.GetFloat("MHHP"));

PlayerPrefs.SetFloat("SHP", PlayerPrefs.GetFloat("MSHP"));

PlayerPrefs.Save();

}

// Update is called once per frame

void Update ()

{

//Test is primarily used to debug on android and displays all necessary variables on the Canvas at run-time which effect gameplay. (Currently in-active in both scenes)

GameObject.Find("Test").GetComponent<Text>().text = ("Quality: " + PlayerPrefs.GetFloat("fQuality") + ",\n Turn Speed: " + PlayerPrefs.GetFloat("Turn") + ",\n Damage: " + PlayerPrefs.GetFloat("fDamage") + ",\n Max Hull HP: " + PlayerPrefs.GetFloat("MHHP") +

",\n Hull HP: " + PlayerPrefs.GetFloat("HHP") + ",\n Max Sail HP: " + PlayerPrefs.GetFloat("MSHP") + ",\n Sail HP: " + PlayerPrefs.GetFloat("SHP"));

//Update the gold amount displayed to the player in the management scene

GameObject.Find("Gold").GetComponent<Text>().text = PlayerPrefs.GetInt("Gold") + " Gold";

//The following IF statements query whether the UI is active (displayed) as well as the sub-category and if so update the UI text:

//Update Market UI text with relevant PlayerPref variables:

if (UI.activeInHierarchy == true && Market.activeInHierarchy == true)

{

//Display the amount of loot in the ships cargo

GameObject.Find("Grain").transform.GetChild(0).GetComponent<Text>().text = PlayerPrefs.GetInt("Grain").ToString();

GameObject.Find("Fish").transform.GetChild(0).GetComponent<Text>().text = PlayerPrefs.GetInt("Fish").ToString();

GameObject.Find("Oil").transform.GetChild(0).GetComponent<Text>().text = PlayerPrefs.GetInt("Oil").ToString();

GameObject.Find("Wood").transform.GetChild(0).GetComponent<Text>().text = PlayerPrefs.GetInt("Wood").ToString();

GameObject.Find("Brick").transform.GetChild(0).GetComponent<Text>().text = PlayerPrefs.GetInt("Brick").ToString();

GameObject.Find("Iron").transform.GetChild(0).GetComponent<Text>().text = PlayerPrefs.GetInt("Iron").ToString();

GameObject.Find("Rum").transform.GetChild(0).GetComponent<Text>().text = PlayerPrefs.GetInt("Rum").ToString();

GameObject.Find("Silk").transform.GetChild(0).GetComponent<Text>().text = PlayerPrefs.GetInt("Silk").ToString();

GameObject.Find("Silverware").transform.GetChild(0).GetComponent<Text>().text = PlayerPrefs.GetInt("Silverware").ToString();

GameObject.Find("Emerald").transform.GetChild(0).GetComponent<Text>().text = PlayerPrefs.GetInt("Emerald").ToString();

}

//Update Shipyard UI text with relevant PlayerPref variables:

if (UI.activeInHierarchy == true && Shipyard.activeInHierarchy == true)

{

//Display the current upgrade level for each category

GameObject.Find("Sails").transform.GetChild(2).GetComponent<Text>().text = PlayerPrefs.GetInt("Sails").ToString() + " / 10";

GameObject.Find("CBDamage").transform.GetChild(2).GetComponent<Text>().text = PlayerPrefs.GetInt("Damage").ToString() + " / 10";

GameObject.Find("Hull").transform.GetChild(2).GetComponent<Text>().text = PlayerPrefs.GetInt("Hull").ToString() + " / 10";

GameObject.Find("Storage").transform.GetChild(2).GetComponent<Text>().text = PlayerPrefs.GetInt("Storage").ToString() + " / 10";

GameObject.Find("Quality").transform.GetChild(2).GetComponent<Text>().text = PlayerPrefs.GetInt("Quality").ToString() + " / 10";

//Display cost of upgrade for each upgrade category

GameObject.Find("Sails").transform.GetChild(4).GetComponent<Text>().text =

PlayerPrefs.GetInt("Sails") < 10 ? (PlayerPrefs.GetInt("Sails") \* 5).ToString() + "g" : "MAX";

GameObject.Find("CBDamage").transform.GetChild(4).GetComponent<Text>().text =

PlayerPrefs.GetInt("Damage") < 10 ? (PlayerPrefs.GetInt("Damage") \* 6).ToString() + "g" : "MAX";

GameObject.Find("Hull").transform.GetChild(4).GetComponent<Text>().text =

PlayerPrefs.GetInt("Hull") < 10 ? (PlayerPrefs.GetInt("Hull") \* 7).ToString() + "g" : "MAX";

GameObject.Find("Storage").transform.GetChild(4).GetComponent<Text>().text =

PlayerPrefs.GetInt("Storage") < 10 ? (PlayerPrefs.GetInt("Storage") \* 8).ToString() + "g" : "MAX";

GameObject.Find("Quality").transform.GetChild(4).GetComponent<Text>().text =

PlayerPrefs.GetInt("Quality") < 10 ? (PlayerPrefs.GetInt("Quality") \* 9).ToString() + "g" : "MAX";

}

//Move menu AI-controlled ship:

if (cTar) //If the Sea Raid is about to begin

{

//Set the destination to be the SeaRaid waypoint, placed in the scene

nav.SetDestination(GameObject.Find("SeaRaid").transform.position);

}

else //If the Sea Raid is not starting

{

if (currLocation == "Start")

{

//If the current location of the UI ship is the start set its destination to the start

nav.SetDestination(GameObject.Find("StartLocation").transform.position);

//Set both booleans to false

mVia = false;

sVia = false;

}

else if (currLocation == "Market1")

{

//If the location is the market then check whether the ship is close to the way-point and if so set the boolean to true

if (Vector3.Distance(GameObject.Find("MenuPlayer").transform.position, GameObject.Find("MarketViaLocation").transform.position) < 5.0f)

{

mVia = true;

}

if (!mVia)

{

//If the ship hasn't reached the waypoint go to it

nav.SetDestination(GameObject.Find("MarketViaLocation").transform.position);

}

else

{

//Else if the ship has reached the waypoint, dock at the Market

nav.SetDestination(GameObject.Find("MarketLocation").transform.position);

}

}

else if (currLocation == "Shipyard1")

{

//If the location is the Shipyard then check whether the ship is close to the way-point and if so set the boolean to true

if (Vector3.Distance(GameObject.Find("MenuPlayer").transform.position, GameObject.Find("ShipyardViaLocation").transform.position) < 5.0f)

{

mVia = true;

}

if (!mVia)

{

//If the ship hasn't reached the waypoint go to it

nav.SetDestination(GameObject.Find("ShipyardViaLocation").transform.position);

}

else

{

//Else if the ship has reached the waypoint, dock at the Shipyard

nav.SetDestination(GameObject.Find("ShipyardLocation").transform.position);

}

}

}

//PC Input:

if (Input.GetMouseButtonDown(0)) //If the mouse is clicked

{

//Initialise raycast variables

RaycastHit hit;

Ray mousePosition;

//Set the mouse position ray to be equal to the mouse position on the screen as a raycast location

mousePosition = Camera.main.ScreenPointToRay(Input.mousePosition);

if (Physics.Raycast(mousePosition, out hit) && UI.activeInHierarchy == false) //Fire the raycast if the menu isn't open

{

if (hit.transform.tag == "Market" || hit.transform.tag == "Shipyard" || hit.transform.tag == "Combat")

{

//If the hit object from the raycast is either the market, shipyard or combat then

GameObject.Find("UIManager").GetComponent<UIFunctions>().ClickObj(); //Play UI clicking sound effect

//Set the UI Parent GameObject to be visible

UI.SetActive(true);

//Set all of the child UI sub-items to be hidden

Combat.SetActive(false);

Shipyard.SetActive(false);

Market.SetActive(false);

}

if (hit.transform.tag == "Market")

{

//If player clicked the market

Debug.Log("Market");

UITitle.text = "Market";

//Set the AI-controlled player ship to the Market

currLocation = "Market1";

//Display the market UI sub-item

Market.SetActive(true);

}

else if (hit.transform.tag == "Shipyard")

{

//If player clicked shipyard

Debug.Log("Shipyard");

UITitle.text = "Shipyard";

//Set the AI-controlled player ship to the Shipyard

currLocation = "Shipyard1";

//Display the Shipyard UI sub-item

Shipyard.SetActive(true);

}

else if (hit.transform.tag == "Combat")

{

//If the player clicked combat

Debug.Log("Combat");

UITitle.text = "Combat";

//Set the AI-controlled player ship to the Combat area

currLocation = "Combat";

//Display the Combat UI sub-item

Combat.SetActive(true);

}

}

else if (UI.activeInHierarchy == false)

{

//Return the AI controlled ship to the start if nothing was selected

currLocation = "Start";

}

}

//Android Input:

if (Input.touchCount == 1 && Input.GetTouch(0).phase == TouchPhase.Ended)

{

//Initialise raycast variables

RaycastHit hit;

Ray mousePosition;

//Set the mouse position ray to be equal to the touch position on the screen as a raycast location

mousePosition = Camera.main.ScreenPointToRay(Input.GetTouch(0).position);

if (Physics.Raycast(mousePosition, out hit) && UI.activeInHierarchy == false) //Fire the raycast if the menu isn't open

{

if (hit.transform.tag == "Market" || hit.transform.tag == "Shipyard" || hit.transform.tag == "Combat")

{

//If the hit object from the raycast is either the market, shipyard or combat then

GameObject.Find("UIManager").GetComponent<UIFunctions>().ClickObj(); //Play UI clicking sound effect

//Set the UI Parent GameObject to be visible

UI.SetActive(true);

//Set all of the child UI sub-items to be hidden

Combat.SetActive(false);

Shipyard.SetActive(false);

Market.SetActive(false);

}

if (hit.transform.tag == "Market")

{

//If player clicked the market

Debug.Log("Market");

UITitle.text = "Market";

//Set the AI-controlled player ship to the Market

currLocation = "Market1";

//Display the market UI sub-item

Market.SetActive(true);

}

else if (hit.transform.tag == "Shipyard")

{

//If player clicked shipyard

Debug.Log("Shipyard");

UITitle.text = "Shipyard";

//Set the AI-controlled player ship to the Shipyard

currLocation = "Shipyard1";

//Display the Shipyard UI sub-item

Shipyard.SetActive(true);

}

else if (hit.transform.tag == "Combat")

{

//If the player clicked combat

Debug.Log("Combat");

UITitle.text = "Combat";

//Set the AI-controlled player ship to the Combat area

currLocation = "Combat";

//Display the Combat UI sub-item

Combat.SetActive(true);

}

}

else if (UI.activeInHierarchy == false)

{

//Return the AI controlled ship to the start if nothing was selected

currLocation = "Start";

}

}

}

}

### UI Functions Script

using System.Collections;

using System.Collections.Generic;

using UnityEngine;

using UnityEngine.SceneManagement;

using UnityEngine.UI;

using UnityEngine.AI;

public class UIFunctions : MonoBehaviour {

//UI Button controls and Misc UI control functions (Attached to UI Manager GameObject)

//Sound effect objects:

public AudioSource UpgradeStuff;

public AudioSource SellStuff;

public AudioSource Click;

public AudioSource CloseW;

//Canvas objects:

private GameObject UI;

private GameObject Music;

private GameObject MM;

//Floats to store & display the combat scene loading countdown

private float CurrTime = 0.0f;

private float StartTime = 0.0f;

//Boolean to check if the game is starting (combat scene being loaded)

private bool Starting = false;

// Update is called once per frame

void Update () {

if (Starting)

{

//If the Combat scene is about to be loaded display the countdown to it on the relevant UI element:

GameObject.Find("xCombat").transform.GetChild(2).GetComponent<Text>().text = "Starting Combat in.. " + Mathf.Round(StartTime - Time.time).ToString();

}

}

private void Start()

{

//Initialise game-object variable on start:

UI = GameObject.Find("UI");

MM = GameObject.Find("EventSystem");

Music = MM.transform.GetChild(0).gameObject;

}

//Button Controls:

//-------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------

// Miscellaneous button controls:

public void CloseUI()

{

//Function which closes the UI if it's open

//Play the close window sound effect

CloseW.Play();

Debug.Log("UI Closed");

//If the UI has been closed set the current location of the AI-controlled player ship to the start

GameObject.Find("Main Camera").GetComponent<ManagementTap>().currLocation = "Start";

//Hide the UI:

UI.SetActive(false);

}

public void StartCombat()

{

//Function to start combat (Activated when player clicks yes to starting combat)

//Play generic click UI element click sound

Click.Play();

//De-activate the buttons on the UI (Yes / No buttons)

GameObject.Find("xCombat").transform.GetChild(0).gameObject.SetActive(false);

GameObject.Find("xCombat").transform.GetChild(1).gameObject.SetActive(false);

//Set the countdown text initially (TBC by update in this class (above))

GameObject.Find("xCombat").transform.GetChild(2).GetComponent<Text>().text = "Starting Combat in.. 5";

//Set the starting boolean to true

Starting = true;

//Set the current time variable to be the time since the game started.

CurrTime = Time.time;

//Set the start time variable to be the time since the game started + 5 seconds. (primarily to allow the AI-controlled player ship to move upwards and give the player some time to prepare for battle)

StartTime = Time.time + 5.0f;

//Set the speed of the ship to a huge amount

GameObject.Find("MenuPlayer").GetComponent<NavMeshAgent>().speed = 100;

//Set the Moving to combat target waypoint boolean in the management class to true

GameObject.Find("Main Camera").GetComponent<ManagementTap>().cTar = true;

//switch scene to combat scene after 5 seconds

Invoke("BeginGame", 5.0f);

}

void BeginGame()

{

//Used in StartCombat to load the scene using Invoke.

SceneManager.LoadScene(1);

}

public void ClickObj()

{

//Used as a public function to allow other functions to use the attached click sound effect remotely.

Click.Play();

}

public void ToggleMusic()

{

//Toggles the music on/off

//Play generic click UI sound effect

Click.Play();

//Reverse the objects sound enabled status to enable or disable music

Music.SetActive(!Music.activeInHierarchy);

}

//-------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------

// Shipyard Button controls:

//Upgrade button generic comments:

//Click.Play(); - Plays Generic UI Button Click sound effect

//If statement - Checks if the users gold amount is greater than the cost of the item the player is trying to upgrade

//UpgradeStuff.Play(); - Plays the successful upgrade item sound effect

//SetInt #1 - Sets the players gold to be X amount lower than it is now dependant on the upgrade cost of that item

//SetInt #2 - Increments upgrade level of item

//SetFloat - Sets the float (which is used in the combat scene) to the appropriate new amount based on the upgrade value of that item

public void UpgradeSails()

{

Click.Play();

if ((PlayerPrefs.GetInt("Gold") >= (PlayerPrefs.GetInt("Sails") \* 5)) && (PlayerPrefs.GetInt("Sails") < 10))

{

UpgradeStuff.Play();

PlayerPrefs.SetInt("Gold", PlayerPrefs.GetInt("Gold") - (PlayerPrefs.GetInt("Sails") \* 5));

PlayerPrefs.SetInt("Sails", PlayerPrefs.GetInt("Sails") + 1);

}

PlayerPrefs.SetFloat("Turn", ((PlayerPrefs.GetInt("Sails") / 10) + 1));

}

public void UpgradeDamage()

{

Click.Play();

if ((PlayerPrefs.GetInt("Gold") >= (PlayerPrefs.GetInt("Damage") \* 6)) && (PlayerPrefs.GetInt("Damage") < 10))

{

UpgradeStuff.Play();

PlayerPrefs.SetInt("Gold", PlayerPrefs.GetInt("Gold") - (PlayerPrefs.GetInt("Damage") \* 6));

PlayerPrefs.SetInt("Damage", PlayerPrefs.GetInt("Damage") + 1);

}

PlayerPrefs.SetFloat("fDamage", (PlayerPrefs.GetInt("Damage") \* 5) + 15);

}

public void UpgradeHull()

{

Click.Play();

if ((PlayerPrefs.GetInt("Gold") >= (PlayerPrefs.GetInt("Hull") \* 7)) && (PlayerPrefs.GetInt("Hull") < 10))

{

UpgradeStuff.Play();

PlayerPrefs.SetInt("Gold", PlayerPrefs.GetInt("Gold") - (PlayerPrefs.GetInt("Hull") \* 7));

PlayerPrefs.SetInt("Hull", PlayerPrefs.GetInt("Hull") + 1);

}

PlayerPrefs.SetFloat("MHHP", ((PlayerPrefs.GetInt("Hull") - 1) \* 10) + (PlayerPrefs.GetInt("Hull") <= 1 ? 50 : 60));

PlayerPrefs.SetFloat("HHP", PlayerPrefs.GetFloat("MHHP"));

}

public void UpgradeStorage()

{

Click.Play();

if ((PlayerPrefs.GetInt("Gold") >= (PlayerPrefs.GetInt("Storage") \* 8)) && (PlayerPrefs.GetInt("Storage") < 10))

{

UpgradeStuff.Play();

PlayerPrefs.SetInt("Gold", PlayerPrefs.GetInt("Gold") - (PlayerPrefs.GetInt("Storage") \* 8));

PlayerPrefs.SetInt("Storage", PlayerPrefs.GetInt("Storage") + 1);

}

PlayerPrefs.SetInt("Cargo", (PlayerPrefs.GetInt("Storage") \* 5) + 5);

}

public void UpgradeQuality()

{

Click.Play();

if ((PlayerPrefs.GetInt("Gold") >= (PlayerPrefs.GetInt("Quality") \* 9)) && (PlayerPrefs.GetInt("Quality") < 10))

{

UpgradeStuff.Play();

PlayerPrefs.SetInt("Gold", PlayerPrefs.GetInt("Gold") - (PlayerPrefs.GetInt("Quality") \* 9));

PlayerPrefs.SetInt("Quality", PlayerPrefs.GetInt("Quality") + 1);

}

PlayerPrefs.SetFloat("fQuality", 1.0f - (PlayerPrefs.GetInt("Quality") \* 0.05f));

}

//-------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------

// Market Button Controls:

public void SellAll()

{

//This function sells all of the players cargo items based on the table found in the brief of the assignment

//Play the generic ui click sound effect

Click.Play();

//Checks if there is even anything to sell:

if (PlayerPrefs.GetInt("Grain") > 0 ||

PlayerPrefs.GetInt("Fish") > 0 ||

PlayerPrefs.GetInt("Oil") > 0 ||

PlayerPrefs.GetInt("Wood") > 0 ||

PlayerPrefs.GetInt("Brick") > 0 ||

PlayerPrefs.GetInt("Iron") > 0 ||

PlayerPrefs.GetInt("Rum") > 0 ||

PlayerPrefs.GetInt("Silk") > 0 ||

PlayerPrefs.GetInt("Silverware") > 0 ||

PlayerPrefs.GetInt("Emerald") > 0)

{

//Play the selling sound effect if there was something to be sold

SellStuff.Play();

//Sets the players gold amount to the current value plus whatever they've earned by selling the cargo

PlayerPrefs.SetInt("Gold", PlayerPrefs.GetInt("Gold") +

(PlayerPrefs.GetInt("Grain") \* 1) +

(PlayerPrefs.GetInt("Fish") \* 2) +

(PlayerPrefs.GetInt("Oil") \* 3) +

(PlayerPrefs.GetInt("Wood") \* 5) +

(PlayerPrefs.GetInt("Brick") \* 8) +

(PlayerPrefs.GetInt("Iron") \* 10) +

(PlayerPrefs.GetInt("Rum") \* 15) +

(PlayerPrefs.GetInt("Silk") \* 20) +

(PlayerPrefs.GetInt("Silverware") \* 30) +

(PlayerPrefs.GetInt("Emerald") \* 50)

);

//Set the current cargo amount to 0

PlayerPrefs.SetInt("Grain", 0);

PlayerPrefs.SetInt("Fish", 0);

PlayerPrefs.SetInt("Oil", 0);

PlayerPrefs.SetInt("Wood", 0);

PlayerPrefs.SetInt("Brick", 0);

PlayerPrefs.SetInt("Iron", 0);

PlayerPrefs.SetInt("Rum", 0);

PlayerPrefs.SetInt("Silk", 0);

PlayerPrefs.SetInt("Silverware", 0);

PlayerPrefs.SetInt("Emerald", 0);

}

}

//Generic Selling comments:

//Sell one:

//Click.Play(); - Play the generic UI clicking sound effect

//If statement - Check if there is any of the relevant item to be sold

//SellStuff.Play(); - Play the selling sound effect

//SetInt #1 - Decrease the relevant cargo item by 1

//SetInt #2 - Increase the gold based on that items value

//Set all:

//Click.Play(); - Play the generic UI clicking sound effect

//If statement - Check if there is any of the relevant item to be sold

//SellStuff.Play(); - Play the selling sound effect

//SetInt #1 - Increase the players gold based on the value of the item(s) sold and the worth of each item

//SetInt #2 - Set the amount of that cargo to 0

public void GrainSellOne()

{

Click.Play();

if (PlayerPrefs.GetInt("Grain") > 0)

{

SellStuff.Play();

PlayerPrefs.SetInt("Grain", PlayerPrefs.GetInt("Grain") - 1);

PlayerPrefs.SetInt("Gold", PlayerPrefs.GetInt("Gold") + 1);

}

}

public void GrainSellAll()

{

Click.Play();

if (PlayerPrefs.GetInt("Grain") > 0)

{

SellStuff.Play();

PlayerPrefs.SetInt("Gold", PlayerPrefs.GetInt("Gold") + PlayerPrefs.GetInt("Grain"));

PlayerPrefs.SetInt("Grain", 0);

}

}

public void FishSellOne()

{

Click.Play();

if (PlayerPrefs.GetInt("Fish") > 0)

{

SellStuff.Play();

PlayerPrefs.SetInt("Fish", PlayerPrefs.GetInt("Fish") - 1);

PlayerPrefs.SetInt("Gold", PlayerPrefs.GetInt("Gold") + 2);

}

}

public void FishSellAll()

{

Click.Play();

if (PlayerPrefs.GetInt("Fish") > 0)

{

SellStuff.Play();

PlayerPrefs.SetInt("Gold", PlayerPrefs.GetInt("Gold") + (PlayerPrefs.GetInt("Fish") \* 2));

PlayerPrefs.SetInt("Fish", 0);

}

}

public void OilSellOne()

{

Click.Play();

if (PlayerPrefs.GetInt("Oil") > 0)

{

SellStuff.Play();

PlayerPrefs.SetInt("Oil", PlayerPrefs.GetInt("Oil") - 1);

PlayerPrefs.SetInt("Gold", PlayerPrefs.GetInt("Gold") + 3);

}

}

public void OilSellAll()

{

Click.Play();

if (PlayerPrefs.GetInt("Oil") > 0)

{

SellStuff.Play();

PlayerPrefs.SetInt("Gold", PlayerPrefs.GetInt("Gold") + (PlayerPrefs.GetInt("Oil") \* 3));

PlayerPrefs.SetInt("Oil", 0);

}

}

public void WoodSellOne()

{

Click.Play();

if (PlayerPrefs.GetInt("Wood") > 0)

{

SellStuff.Play();

PlayerPrefs.SetInt("Wood", PlayerPrefs.GetInt("Wood") - 1);

PlayerPrefs.SetInt("Gold", PlayerPrefs.GetInt("Gold") + 5);

}

}

public void WoodSellAll()

{

Click.Play();

if (PlayerPrefs.GetInt("Wood") > 0)

{

SellStuff.Play();

PlayerPrefs.SetInt("Gold", PlayerPrefs.GetInt("Gold") + (PlayerPrefs.GetInt("Wood") \* 5));

PlayerPrefs.SetInt("Wood", 0);

}

}

public void BrickSellOne()

{

Click.Play();

if (PlayerPrefs.GetInt("Brick") > 0)

{

SellStuff.Play();

PlayerPrefs.SetInt("Brick", PlayerPrefs.GetInt("Brick") - 1);

PlayerPrefs.SetInt("Gold", PlayerPrefs.GetInt("Gold") + 8);

}

}

public void BrickSellAll()

{

Click.Play();

if (PlayerPrefs.GetInt("Brick") > 0)

{

SellStuff.Play();

PlayerPrefs.SetInt("Gold", PlayerPrefs.GetInt("Gold") + (PlayerPrefs.GetInt("Brick") \* 8));

PlayerPrefs.SetInt("Brick", 0);

}

}

public void IronSellOne()

{

Click.Play();

if (PlayerPrefs.GetInt("Iron") > 0)

{

SellStuff.Play();

PlayerPrefs.SetInt("Iron", PlayerPrefs.GetInt("Iron") - 1);

PlayerPrefs.SetInt("Gold", PlayerPrefs.GetInt("Gold") + 10);

}

}

public void IronSellAll()

{

Click.Play();

if (PlayerPrefs.GetInt("Iron") > 0)

{

SellStuff.Play();

PlayerPrefs.SetInt("Gold", PlayerPrefs.GetInt("Gold") + (PlayerPrefs.GetInt("Iron") \* 10));

PlayerPrefs.SetInt("Iron", 0);

}

}

public void RumSellOne()

{

Click.Play();

if (PlayerPrefs.GetInt("Rum") > 0)

{

SellStuff.Play();

PlayerPrefs.SetInt("Rum", PlayerPrefs.GetInt("Rum") - 1);

PlayerPrefs.SetInt("Gold", PlayerPrefs.GetInt("Gold") + 15);

}

}

public void RumSellAll()

{

Click.Play();

if (PlayerPrefs.GetInt("Rum") > 0)

{

SellStuff.Play();

PlayerPrefs.SetInt("Gold", PlayerPrefs.GetInt("Gold") + (PlayerPrefs.GetInt("Rum") \* 15));

PlayerPrefs.SetInt("Rum", 0);

}

}

public void SilkSellOne()

{

Click.Play();

if (PlayerPrefs.GetInt("Silk") > 0)

{

SellStuff.Play();

PlayerPrefs.SetInt("Silk", PlayerPrefs.GetInt("Silk") - 1);

PlayerPrefs.SetInt("Gold", PlayerPrefs.GetInt("Gold") + 20);

}

}

public void SilkSellAll()

{

Click.Play();

if (PlayerPrefs.GetInt("Silk") > 0)

{

SellStuff.Play();

PlayerPrefs.SetInt("Gold", PlayerPrefs.GetInt("Gold") + (PlayerPrefs.GetInt("Silk") \* 20));

PlayerPrefs.SetInt("Silk", 0);

}

}

public void SilverwareSellOne()

{

Click.Play();

if (PlayerPrefs.GetInt("Silverware") > 0)

{

SellStuff.Play();

PlayerPrefs.SetInt("Silverware", PlayerPrefs.GetInt("Silverware") - 1);

PlayerPrefs.SetInt("Gold", PlayerPrefs.GetInt("Gold") + 30);

}

}

public void SilverwareSellAll()

{

Click.Play();

if (PlayerPrefs.GetInt("Silverware") > 0)

{

SellStuff.Play();

PlayerPrefs.SetInt("Gold", PlayerPrefs.GetInt("Gold") + (PlayerPrefs.GetInt("Silverware") \* 30));

PlayerPrefs.SetInt("Silverware", 0);

}

}

public void EmeraldSellOne()

{

Click.Play();

if (PlayerPrefs.GetInt("Emerald") > 0)

{

SellStuff.Play();

PlayerPrefs.SetInt("Emerald", PlayerPrefs.GetInt("Emerald") - 1);

PlayerPrefs.SetInt("Gold", PlayerPrefs.GetInt("Gold") + 50);

}

}

public void EmeraldSellAll()

{

Click.Play();

if (PlayerPrefs.GetInt("Emerald") > 0)

{

SellStuff.Play();

PlayerPrefs.SetInt("Gold", PlayerPrefs.GetInt("Gold") + (PlayerPrefs.GetInt("Emerald") \* 50));

PlayerPrefs.SetInt("Emerald", 0);

}

}

}

### Combat Script

using System.Collections;

using System.Collections.Generic;

using UnityEngine;

using UnityEngine.UI;

using UnityStandardAssets.CrossPlatformInput;

using UnityEngine.SceneManagement;

public class Combat : MonoBehaviour {

//Combat script which controls everything related to the player as well as management of spawning and other scene elements

//Misc player/camera variables to control movement:

public Rigidbody shipRigid;

private Quaternion rotation;

private Vector3 pos;

//Camera GameObject:

private GameObject cam;

//Various UI variables:

private Text HHP;

private Text SHP;

private Text Cargo;

private GameObject button;

//Enemy Spawning Variables:

public Rigidbody CannonBall;

public Rigidbody Sloop;

public Rigidbody Caravel;

public Rigidbody Brigantine;

private Rigidbody Spawned;

//Sound effect variables:

public AudioSource CollisionAudio;

public AudioSource LootAudio;

//Ship's Cannon Variables:

private bool CannonCD1;

private bool CannonCD2;

private float CannonCD1Time;

private float CannonCD2Time;

private float CannonBallDamage;

public float CannonBallSpeed;

//Ship's Collision timing variables:

private bool canCollide = true;

private float canCollideTime = 0.0f;

//Game-End variables:

private bool GameOver = false;

private float GameOverTime = 0.0f;

private float GameEndTime = 0.0f;

//Ship's Current Inventory items:

public int Gold = 0;

private int Grain = 0;

private int Fish = 0;

private int Oil = 0;

private int Wood = 0;

private int Brick = 0;

private int Iron = 0;

private int Rum = 0;

private int Silk = 0;

private int Silverware = 0;

private int Emerald = 0;

//Current amount of cargo:

private int currentCargo = 0;

//Ship's Stats:

private float Quality;

private float turn\_speed;

public float speed;

public float LootPullDistance;

float mTurn = 0.0f;

// Use this for initialization

void Start () {

//Set both the left and right cannon's cooldown to ready

CannonCD1 = false;

CannonCD2 = false;

}

private void OnApplicationQuit()

{

//When the player closes the game, delete all PlayerPrefs

PlayerPrefs.DeleteAll();

}

private void FixedUpdate()

{

//Get the amount the player has moved the analog stick from the centre

mTurn = CrossPlatformInputManager.GetAxis("Horizontal");

//Turn the player ships rigid body using the above joystick amount using torque to simulate an actual ship

shipRigid.AddRelativeTorque(new Vector3(0, (mTurn \* (turn\_speed + 15) \* Time.deltaTime), 0), ForceMode.Force);

}

private void Awake()

{

if (CannonBallDamage != PlayerPrefs.GetFloat("fDamage"))

{

//Check if the Cannon ball damage has been initialised, if not initialise it

CannonBallDamage = PlayerPrefs.GetInt("fDamage");

}

if (Quality != PlayerPrefs.GetInt("fQuality"))

{

//Check if the Quality has been initialised, if not initialise it

Quality = PlayerPrefs.GetInt("fQuality");

}

//Set the Players sail hp to the max sail hp

PlayerPrefs.SetFloat("SHP", PlayerPrefs.GetFloat("MSHP"));

//Set the Players hull hp to the max hull hp

PlayerPrefs.SetFloat("HHP", PlayerPrefs.GetFloat("MHHP"));

//Set locale variables to the playerprefs variables for the ships stats

turn\_speed = PlayerPrefs.GetFloat("Turn");

CannonBallDamage = PlayerPrefs.GetFloat("fDamage");

Cargo = GameObject.Find("Cargo").GetComponent<Text>();

Quality = PlayerPrefs.GetFloat("fQuality");

//Get the camera object

cam = GameObject.Find("Main Camera");

rotation = cam.GetComponent<Rigidbody>().transform.rotation;

//Set the UI elements to show current hull and sail hp

HHP = GameObject.Find("Hull Health").GetComponent<Text>();

SHP = GameObject.Find("Sail Health").GetComponent<Text>();

//Used for testing purposes

Debug.Log("Quality: " + Quality + ", Turn Speed: " + turn\_speed + ", Damage: " + CannonBallDamage + ", Max Hull HP: " + PlayerPrefs.GetFloat("MHHP") +

", Hull HP: " + PlayerPrefs.GetFloat("HHP") + ", Max Sail HP: " + PlayerPrefs.GetFloat("MSHP") + ", Sail HP: " + PlayerPrefs.GetFloat("SHP"));

//Used for android testing purposes to track stats without a debugger

GameObject.Find("Test").GetComponent<Text>().text = ("Quality: " + PlayerPrefs.GetFloat("fQuality") + ",\n Turn Speed: " + PlayerPrefs.GetFloat("Turn") + ",\n Damage: " + PlayerPrefs.GetFloat("fDamage") + ",\n Max Hull HP: " + PlayerPrefs.GetFloat("MHHP") +

",\n Hull HP: " + PlayerPrefs.GetFloat("HHP") + ",\n Max Sail HP: " + PlayerPrefs.GetFloat("MSHP") + ",\n Sail HP: " + PlayerPrefs.GetFloat("SHP"));

//-----------------------------------------------------------------------------------------------------------------------------------------------------------------------------

// Enemy Spawning

//Get the random percentage which defines how many enemies will spawn

float SpawnPerc = Random.Range(0.0f, 100.0f);

if (SpawnPerc > 0.0f && SpawnPerc <= 60.0f)

{

//If the spawn percentage is 60%

//Set the random amounts of individual spawns for this percentage

float NSloop = Mathf.Round(Random.Range(2, 4));

float NCaravel = Mathf.Round(Random.Range(0, 1));

Vector3 SpawnPos;

for (int i = 1; i <= NSloop; i++)

{

//Cycle through the amount of Sloops to spawn dependant on the random numbers generated above

//Set the spawn position to the centre (where the player spawns)

SpawnPos = transform.position;

//Set x to be a random number between 5 and 20 positive or negative

SpawnPos.x += (Mathf.Round(Random.Range(0.0f, 100.0f)) <= 50.0f ? Random.Range(5.0f, 20.0f) : Random.Range(-5.0f, -20.0f));

SpawnPos.z += (Mathf.Round(Random.Range(0.0f, 100.0f)) <= 50.0f ? Random.Range(5.0f, 20.0f) : Random.Range(-5.0f, -20.0f));

//Set the Spawned variable to be equal to the new ship

Spawned = Instantiate(Sloop, SpawnPos, transform.rotation);

}

for (int i = 1; i <= NCaravel; i++)

{

//Cycle through the amount of Caravels to spawn dependant on the random numbers generated above

//Set the spawn position to the centre (where the player spawns)

SpawnPos = transform.position;

//Set x to be a random number between 5 and 20 positive or negative

SpawnPos.x += (Mathf.Round(Random.Range(0.0f, 100.0f)) <= 50.0f ? Random.Range(5.0f, 20.0f) : Random.Range(-5.0f, -20.0f));

SpawnPos.z += (Mathf.Round(Random.Range(0.0f, 100.0f)) <= 50.0f ? Random.Range(5.0f, 20.0f) : Random.Range(-5.0f, -20.0f));

//Set the Spawned variable to be equal to the new ship

Spawned = Instantiate(Caravel, SpawnPos, transform.rotation);

}

//For debugging purposes

Debug.Log("Spawned 60% chance (" + NSloop + " Sloops, " + NCaravel + " Caravels)");

}

else if (SpawnPerc > 60.0f && SpawnPerc <= 90.0f)

{

float NSloop = Mathf.Round(Random.Range(2, 6));

float NCaravel = Mathf.Round(Random.Range(1, 3));

float NBrigantine = Mathf.Round(Random.Range(0, 1));

Vector3 SpawnPos;

for (int i = 1; i <= NSloop; i++)

{

//Cycle through the amount of Sloops to spawn dependant on the random numbers generated above

//Set the spawn position to the centre (where the player spawns)

SpawnPos = transform.position;

//Set x to be a random number between 5 and 20 positive or negative

SpawnPos.x += (Mathf.Round(Random.Range(0.0f, 100.0f)) <= 50.0f ? Random.Range(5.0f, 20.0f) : Random.Range(-5.0f, -20.0f));

SpawnPos.z += (Mathf.Round(Random.Range(0.0f, 100.0f)) <= 50.0f ? Random.Range(5.0f, 20.0f) : Random.Range(-5.0f, -20.0f));

//Set the Spawned variable to be equal to the new ship

Spawned = Instantiate(Sloop, SpawnPos, transform.rotation);

}

for (int i = 1; i <= NCaravel; i++)

{

//Cycle through the amount of Caravels to spawn dependant on the random numbers generated above

//Set the spawn position to the centre (where the player spawns)

SpawnPos = transform.position;

//Set x to be a random number between 5 and 20 positive or negative

SpawnPos.x += (Mathf.Round(Random.Range(0.0f, 100.0f)) <= 50.0f ? Random.Range(5.0f, 20.0f) : Random.Range(-5.0f, -20.0f));

SpawnPos.z += (Mathf.Round(Random.Range(0.0f, 100.0f)) <= 50.0f ? Random.Range(5.0f, 20.0f) : Random.Range(-5.0f, -20.0f));

//Set the Spawned variable to be equal to the new ship

Spawned = Instantiate(Caravel, SpawnPos, transform.rotation);

}

for (int i = 1; i <= NBrigantine; i++)

{

//Cycle through the amount of Brigantine to spawn dependant on the random numbers generated above

//Set the spawn position to the centre (where the player spawns)

SpawnPos = transform.position;

//Set x to be a random number between 5 and 20 positive or negative

SpawnPos.x += (Mathf.Round(Random.Range(0.0f, 100.0f)) <= 50.0f ? Random.Range(5.0f, 20.0f) : Random.Range(-5.0f, -20.0f));

SpawnPos.z += (Mathf.Round(Random.Range(0.0f, 100.0f)) <= 50.0f ? Random.Range(5.0f, 20.0f) : Random.Range(-5.0f, -20.0f));

//Set the Spawned variable to be equal to the new ship

Spawned = Instantiate(Brigantine, SpawnPos, transform.rotation);

}

//For debugging purposes

Debug.Log("Spawned 30% chance (" + NSloop + " Sloops, " + NCaravel + " Caravels, " + NBrigantine + " Brigantines)");

}

else if (SpawnPerc > 90.0f && SpawnPerc <= 100.0f)

{

float NSloop = 7;

float NCaravel = Mathf.Round(Random.Range(2, 6));

float NBrigantine = 2;

Vector3 SpawnPos;

SpawnPos.y = 0.01000637f;

for (int i = 1; i <= NSloop; i++)

{

//Cycle through the amount of Sloops to spawn dependant on the random numbers generated above

//Set the spawn position to the centre (where the player spawns)

SpawnPos = transform.position;

//Set x to be a random number between 5 and 20 positive or negative

SpawnPos.x += (Mathf.Round(Random.Range(0.0f, 100.0f)) <= 50.0f ? Random.Range(5.0f, 20.0f) : Random.Range(-5.0f, -20.0f));

SpawnPos.z += (Mathf.Round(Random.Range(0.0f, 100.0f)) <= 50.0f ? Random.Range(5.0f, 20.0f) : Random.Range(-5.0f, -20.0f));

//Set the Spawned variable to be equal to the new ship

Spawned = Instantiate(Sloop, SpawnPos, transform.rotation);

}

for (int i = 1; i <= NCaravel; i++)

{

//Cycle through the amount of Caravels to spawn dependant on the random numbers generated above

//Set the spawn position to the centre (where the player spawns)

SpawnPos = transform.position;

//Set x to be a random number between 5 and 20 positive or negative

SpawnPos.x += (Mathf.Round(Random.Range(0.0f, 100.0f)) <= 50.0f ? Random.Range(5.0f, 20.0f) : Random.Range(-5.0f, -20.0f));

SpawnPos.z += (Mathf.Round(Random.Range(0.0f, 100.0f)) <= 50.0f ? Random.Range(5.0f, 20.0f) : Random.Range(-5.0f, -20.0f));

//Set the Spawned variable to be equal to the new ship

Spawned = Instantiate(Caravel, SpawnPos, transform.rotation);

}

for (int i = 1; i <= NBrigantine; i++)

{

//Cycle through the amount of Brigantine to spawn dependant on the random numbers generated above

//Set the spawn position to the centre (where the player spawns)

SpawnPos = transform.position;

//Set x to be a random number between 5 and 20 positive or negative

SpawnPos.x += (Mathf.Round(Random.Range(0.0f, 100.0f)) <= 50.0f ? Random.Range(5.0f, 20.0f) : Random.Range(-5.0f, -20.0f));

SpawnPos.z += (Mathf.Round(Random.Range(0.0f, 100.0f)) <= 50.0f ? Random.Range(5.0f, 20.0f) : Random.Range(-5.0f, -20.0f));

//Set the Spawned variable to be equal to the new ship

Spawned = Instantiate(Brigantine, SpawnPos, transform.rotation);

}

//For debugging purposes

Debug.Log("Spawned 10% chance (" + NSloop + " Sloops, " + NCaravel + " Caravels, " + NBrigantine + " Brigantines)");

}

}

// Update is called once per frame

void Update ()

{

//UI Meter upgrade modifications:

GameObject.Find("uHull").transform.GetChild(0).GetComponent<Text>().text = PlayerPrefs.GetInt("Hull").ToString();

GameObject.Find("uDamage").transform.GetChild(0).GetComponent<Text>().text = PlayerPrefs.GetInt("Damage").ToString();

GameObject.Find("uSails").transform.GetChild(0).GetComponent<Text>().text = PlayerPrefs.GetInt("Sails").ToString();

GameObject.Find("uStorage").transform.GetChild(0).GetComponent<Text>().text = PlayerPrefs.GetInt("Storage").ToString();

GameObject.Find("uQuality").transform.GetChild(0).GetComponent<Text>().text = PlayerPrefs.GetInt("Quality").ToString();

//Set the in-game slider to the upgrade level of each item

GameObject.Find("uHull").transform.GetChild(2).GetComponent<Slider>().value = PlayerPrefs.GetInt("Hull");

GameObject.Find("uDamage").transform.GetChild(2).GetComponent<Slider>().value = PlayerPrefs.GetInt("Damage");

GameObject.Find("uSails").transform.GetChild(2).GetComponent<Slider>().value = PlayerPrefs.GetInt("Sails");

GameObject.Find("uStorage").transform.GetChild(2).GetComponent<Slider>().value = PlayerPrefs.GetInt("Storage");

GameObject.Find("uQuality").transform.GetChild(2).GetComponent<Slider>().value = PlayerPrefs.GetInt("Quality");

if (Time.time >= canCollideTime)

{

//If the ship crash cooldown is finished set the boolean to true (allowed to take damage from collisions)

canCollide = true;

}

if (PlayerPrefs.GetFloat("HHP") <= 0.0f)

{

//Check to see if the player has died

//and if so set the hull hp to 0 to avoid going into negatives

PlayerPrefs.SetFloat("HHP", 0.0f);

//Display the lose text on the ui

GameObject.Find("Died").GetComponent<Text>().enabled = true;

//Call the return to management scene function after 2 seconds

Invoke("ManagementLose", 2.0f);

}

//Add a constant force to the player based on the speed and sail health

GetComponent<Rigidbody>().AddRelativeForce(((Vector3.forward \* speed \* Time.deltaTime) \* (PlayerPrefs.GetFloat("SHP") / 100)), ForceMode.Acceleration);

//Set the rotation of the camera to static to ensure it doesn't follow the rotation of the player

cam.GetComponent<Rigidbody>().transform.rotation = rotation;

//Get the position of the player and make it hover above the player by 50 units

pos = GetComponent<Rigidbody>().transform.position;

pos.y += 50;

cam.transform.position = pos;

//Set the UI's hull and sail HP to be equal to the player prefs relevant values

HHP.text = "Hull Health: " + Mathf.Round(PlayerPrefs.GetFloat ("HHP")) + " / " + Mathf.Round(PlayerPrefs.GetFloat("MHHP"));

SHP.text = "Sail Health: " + Mathf.Round(PlayerPrefs.GetFloat ("SHP")) + " / " + Mathf.Round(PlayerPrefs.GetFloat("MSHP"));

//Set the cargo UI text to be equal to the relevant player prefs values

Cargo.text = "Cargo: " + currentCargo + " / " + PlayerPrefs.GetInt("Cargo");

if (Time.time >= CannonCD1Time)

{

//Check if the first cannon has cooled down, if so set the boolean to false to allow firing again

CannonCD1 = false;

//Set the UI text to show that the cannon is ready to fire

GameObject.Find("CannonCD1").GetComponent<Text>().text = "Cooldown: Ready";

}

else

{

//If the cannon is still cooling down display the amount of time that is left

GameObject.Find("CannonCD1").GetComponent<Text>().text = "Cooldown: " + (Mathf.Round(Time.time - CannonCD1Time) \* -1) + "s";

}

if (Time.time >= CannonCD2Time)

{

//Check if the second cannon has cooled down, if so set the boolean to false to allow firing again

CannonCD2 = false;

//Set the UI text to show that the cannon is ready to fire

GameObject.Find("CannonCD2").GetComponent<Text>().text = "Cooldown: Ready";

}

else

{

//If the cannon is still cooling down display the amount of time that is left

GameObject.Find("CannonCD2").GetComponent<Text>().text = "Cooldown: " + (Mathf.Round(Time.time - CannonCD2Time) \* -1) + "s";

}

if (!GameOver)

{

//If the gameover bool is not true cycle through all of the objects with the Enemy tag and check if their corresponding boolean for being dead is set to false

var objects = GameObject.FindGameObjectsWithTag("Enemy");

var objectCount = objects.Length;

bool allDead = true;

foreach (var obj in objects)

{

if (!obj.GetComponent<EnemyAI>().Dead)

{

allDead = false;

}

}

if (allDead)

{

//If all of the enemies in the scene are dead, start the countdown to return to the main menu

GameOver = true;

GameOverTime = Time.time;

GameEndTime = GameOverTime + 10.0f;

GameObject.Find("Win").GetComponent<Text>().enabled = true;

//End the game after 10 seconds

Invoke("ManagementWin", 10.0f);

}

}

else

{

//If the Game is over and the player hasn't died display the return to base after 10 seconds ui textbox

GameObject.Find("Win").GetComponent<Text>().text = "All Enemies Dead!\nReturning to base in.. "

+ Mathf.Round((Time.time - GameEndTime) \* -1).ToString();

}

}

//Function to fire the right cannon

public void rFire()

{

if (!CannonCD2)

{

//If the right cannon is not on cooldown

Vector3 Arc;

Debug.Log("Fire! Right");

//Spawn the first cannon ball

Rigidbody BallClone1 = Instantiate(CannonBall, transform.position, transform.rotation);

//Give the cannon ball a velocity to the right

BallClone1.velocity = transform.right \* CannonBallSpeed;

Arc = transform.right;

Arc.z += 0.45f;

//Spawn the second cannonball

Rigidbody BallClone2 = Instantiate(CannonBall, transform.position, transform.rotation);

//Give the cannon ball a velocity to the right adding a 45 degree angle

BallClone2.velocity = (Arc) \* CannonBallSpeed;

Arc = transform.right;

Arc.z -= 0.45f;

//Spawn the third cannonball

Rigidbody BallClone3 = Instantiate(CannonBall, transform.position, transform.rotation);

//Give the cannon ball a velocity to the right minusing a 45 degree angle

BallClone3.velocity = (Arc) \* CannonBallSpeed;

//Set the cannonball fired boolean to true

CannonCD2 = true;

//Set the cooldown time to now plus 3 seconds

CannonCD2Time = Time.time + 3;

//Play the fire sound effect

gameObject.transform.GetChild(1).GetComponent<AudioSource>().Play();

//Give the cannonballs damage based on the formula supplied

BallClone1.GetComponent<CannonBall>().LoadDamage

((CannonBallDamage + (CannonBallDamage \* Random.Range(-0.3f, 0.03f))) \* Quality);

BallClone2.GetComponent<CannonBall>().LoadDamage

((CannonBallDamage + (CannonBallDamage \* Random.Range(-0.3f, 0.03f))) \* Quality);

BallClone3.GetComponent<CannonBall>().LoadDamage

((CannonBallDamage + (CannonBallDamage \* Random.Range(-0.3f, 0.03f))) \* Quality);

}

}

//Function to fire the left cannon

public void lFire()

{

if (!CannonCD1)

{

//If the left cannon is not on cooldown

Vector3 Arc;

Debug.Log("Fire! Left");

//Spawn the first cannon ball

Rigidbody BallClone1 = Instantiate(CannonBall, transform.position, transform.rotation);

//Give the cannon ball a velocity to the right

BallClone1.velocity = -transform.right \* CannonBallSpeed;

Arc = -transform.right;

Arc.z += 0.45f;

//Spawn the second cannonball

Rigidbody BallClone2 = Instantiate(CannonBall, transform.position, transform.rotation);

BallClone2.velocity = (Arc) \* CannonBallSpeed;

Arc = -transform.right;

Arc.z -= 0.45f;

//Spawn the third cannonball

Rigidbody BallClone3 = Instantiate(CannonBall, transform.position, transform.rotation);

//Give the cannon ball a velocity to the right minusing a 45 degree angle

BallClone3.velocity = (Arc) \* CannonBallSpeed;

//Set the cannonball fired boolean to true

CannonCD1 = true;

//Set the cooldown time to now plus 3 seconds

CannonCD1Time = Time.time + 3;

//Play the fire sound effect

gameObject.transform.GetChild(2).GetComponent<AudioSource>().Play();

//Give the cannonballs damage based on the formula supplied

BallClone1.GetComponent<CannonBall>().LoadDamage

((CannonBallDamage + (CannonBallDamage \* Random.Range(-0.3f, 0.03f))) \* Quality);

BallClone2.GetComponent<CannonBall>().LoadDamage

((CannonBallDamage + (CannonBallDamage \* Random.Range(-0.3f, 0.03f))) \* Quality);

BallClone3.GetComponent<CannonBall>().LoadDamage

((CannonBallDamage + (CannonBallDamage \* Random.Range(-0.3f, 0.03f))) \* Quality);

}

}

private void OnTriggerEnter(Collider other)

{

if (other.CompareTag("Enemy") && canCollide)

{

//If the player collided with an enemy

Debug.Log("Crash!");

//Minus half of the max hull hp from the current hp

PlayerPrefs.SetFloat("HHP",PlayerPrefs.GetFloat("HHP") - (PlayerPrefs.GetFloat("MHHP") / 2));

//Play the collision sound effect

CollisionAudio.Play();

//Toggle the collide boolean

canCollide = false;

//Set the cooldown float

canCollideTime = Time.time + 10.0f;

}

else if (other.CompareTag("Loot"))

{

//If the player collided with loot

//Play the loot audio clip

LootAudio.Play();

if (currentCargo < PlayerPrefs.GetInt("Cargo"))

{

//If there is space enough for cargo

Debug.Log("Looted!");

//Get the loot type from the object the player collided with

string LootType = other.GetComponent<LootCrate>().GiveType();

//Switch to find the loot type and add to the local loot variable counter

switch (LootType)

{

case "Grain":

++Grain;

break;

case "Fish":

++Fish;

break;

case "Oil":

++Oil;

break;

case "Wood":

++Wood;

break;

case "Brick":

++Brick;

break;

case "Iron":

++Iron;

break;

case "Rum":

++Rum;

break;

case "Silk":

++Silk;

break;

case "Silverware":

++Silverware;

break;

case "Emerald":

++Emerald;

break;

default:

break;

}

//Add one to the cargo amount

++currentCargo;

//Destroy the cargo item

Destroy(other.gameObject);

}

else

{

//For debugging purposes display that there is no room for the cargo looted

Debug.Log("Cargo Full!");

}

}

}

public void TakeDamage(float Amount)

{

//Called by enemy cannon balls when they collide with the players ship

if (Random.Range(0.0f,100.0f) <= 20.0f)

{

//If the random number generated is 20% then apply damage to the sail health instead of the hull hp

PlayerPrefs.SetFloat("SHP", PlayerPrefs.GetFloat("SHP") - Amount);

}

else

{

//if it's 80% then apply to the hull hp

PlayerPrefs.SetFloat("HHP", PlayerPrefs.GetFloat("HHP") - Amount);

}

//Play the take damage soundclip from the correct child

gameObject.transform.GetChild(3).GetComponent<AudioSource>().Play();

//Debug to display the amount the player was damaged for

Debug.Log("You were hit.. for " + Amount + " damage.");

if (PlayerPrefs.GetFloat("HHP") <= 0.0f)

{

//If the health is less than or equal to 0 the player died

//Set the hp to 0 to prevent it from going below 0

PlayerPrefs.SetFloat("HHP", 0.0f);

//Show the failed ui text component to signify failure

GameObject.Find("Died").GetComponent<Text>().enabled = true;

//Move to the management menu after 2 seconds

Invoke("ManagementLose", 2.0f);

}

if (PlayerPrefs.GetFloat("SHP") <= 0.0f)

{

//Prevent the sails hp from going below 0

PlayerPrefs.SetFloat("SHP", 0.0f);

}

}

void ManagementLose()

{

//Called when the player dies

//Set all local cargo items to 0 (delete them)

Grain = 0;

Fish = 0;

Oil = 0;

Wood = 0;

Brick = 0;

Iron = 0;

Rum = 0;

Silk = 0;

Silverware = 0;

Emerald = 0;

//Remove 25% from the current gold amount

float currGold = PlayerPrefs.GetInt("Gold") \* 0.75f;

//Set gold amount to new value

PlayerPrefs.SetInt("Gold", (int)Mathf.Round(currGold));

//Load the management scene

SceneManager.LoadScene(0);

}

void ManagementWin()

{

//Call when the player wins

//Set all relevant playerpref values to the local cargo (save them)

PlayerPrefs.SetInt("Gold", PlayerPrefs.GetInt("Gold") + Gold);

PlayerPrefs.SetInt("Grain", PlayerPrefs.GetInt("Grain") + Grain);

PlayerPrefs.SetInt("Fish", PlayerPrefs.GetInt("Fish") + Fish);

PlayerPrefs.SetInt("Oil", PlayerPrefs.GetInt("Oil") + Oil);

PlayerPrefs.SetInt("Wood", PlayerPrefs.GetInt("Wood") + Wood);

PlayerPrefs.SetInt("Brick", PlayerPrefs.GetInt("Brick") + Brick);

PlayerPrefs.SetInt("Iron", PlayerPrefs.GetInt("Iron") + Iron);

PlayerPrefs.SetInt("Rum", PlayerPrefs.GetInt("Rum") + Rum);

PlayerPrefs.SetInt("Silk", PlayerPrefs.GetInt("Silk") + Silk);

PlayerPrefs.SetInt("Silverware", PlayerPrefs.GetInt("Silverware") + Silverware);

PlayerPrefs.SetInt("Emerald", PlayerPrefs.GetInt("Emerald") + Emerald);

//Load the management scene

SceneManager.LoadScene(0);

}

}

### Enemy AI Script

using System.Collections;

using System.Collections.Generic;

using UnityEngine;

using UnityEngine.AI;

public class EnemyAI : MonoBehaviour {

//EnemyAI Script which is attached to each enemy spawned and controls the movement, stats and attacks of the enemy AI

//Local enemy stats:

private float HP; //Enemy HP

private float MHP; //Enemy Max HP

private float Quality; //Enemy Quality

private float CannonBallDamage; //Enemy Cannon Ball Damage

private float turn\_Speed; //Enemy turning speed

//Bool to check whether the local enemy is dead or not

public bool Dead;

//Variables to control the firing of the enemy cannons (only one cooldown because they only first in one direction)

private bool CannonCD;

private float CannonCDTime;

private float CannonBallSpeed;

//Rigidbody's which contain the prefab to spawn on either death or attack

public Rigidbody CannonBall;

public Rigidbody Loot;

//Local nav mesh agent variable

private NavMeshAgent nav;

//Misc variables:

private Quaternion rotation;

private void Awake()

{

//Initialise the nav mesh agent

nav = GetComponent<NavMeshAgent>();

//Store the spawn location for use later

Vector3 SpawnLocation = transform.position;

//More nav mesh agent initialisation (used to ensure the enemy is flat against the baked nav area)

nav.Warp(SpawnLocation);

switch (transform.name)

{

//Initialise the enemy ship's stats based on its name

case "Sloop(Clone)":

HP = 25.0f;

CannonBallDamage = 15.0f;

Quality = 0.8f;

turn\_Speed = 1.0f;

break;

case "Caravel(Clone)":

HP = 50.0f;

CannonBallDamage = 30.0f;

Quality = 0.75f;

turn\_Speed = 1.2f;

break;

case "Brigantine(Clone)":

HP = 90.0f;

CannonBallDamage = 50.0f;

Quality = 0.7f;

turn\_Speed = 1.5f;

break;

default:

break;

}

//Set the max hp to be the current hp

MHP = HP;

//Set the nav mesh agent's speed to be equal to the quality

nav.speed = Quality;

//Set the destination preliminarily to the player's position

nav.SetDestination(GameObject.Find("Player").transform.position);

//Set the dead boolean to false

Dead = false;

//Store the spawned enemies name 3d text rotation

rotation = gameObject.transform.GetChild(0).transform.rotation;

//Set the cannon ball speed

CannonBallSpeed = 10.0f;

}

// Update is called once per frame

void Update ()

{

if (Time.time >= CannonCDTime)

{

//Set the cannon ball cooldown to false if the cooldown has expired

CannonCD = false;

}

Vector3 pos;

//Set the enemy's 3d text tooltip to be equal to its current health

gameObject.transform.GetChild(0).GetComponent<TextMesh>().text = transform.name + "\nHP: " + Mathf.Round(HP) + " / " + Mathf.Round(MHP);

//Keep the rotation of the 3d text tooltip equal to the original rotation

gameObject.transform.GetChild(0).transform.rotation = rotation;

//Get the current position of the enemy ship

pos = transform.position;

pos.y += 1;

//Set the enemies name and health tooltip to be off-centre of the position of the ship itself

gameObject.transform.GetChild(0).transform.position = pos;

//Set the colour of the tooltip to be gray with slight transparency

gameObject.transform.GetChild(0).GetComponent<TextMesh>().color = new Color32(114, 114, 114, 138);

//Get the distance between the enemy ship and the player

float dist = Vector3.Distance(transform.position, GameObject.Find("Player").transform.position);

if (dist <= 10.0f && !Dead)

{

//If the position is within the stopping distance of the nav mesh agent and the enemy isnt dead

nav.enabled = false;

//Get the relative position of the direction of the player

Vector3 lookPos = GameObject.Find("Player").transform.position - transform.position;

lookPos.y = 0;

//Get the rotation of the players position compared to the enemy

Quaternion grotation = Quaternion.LookRotation(lookPos);

//Make the new rotation 90 degrees + on the y axis

grotation \*= Quaternion.Euler(0, 90, 0);

//Rotate the enemy ship to face the player

transform.rotation = Quaternion.Slerp(transform.rotation, grotation, Time.deltaTime \* turn\_Speed);

//Fire

if (!CannonCD)

{

//If the enemy is able to fire its cannon

Debug.Log("Fire! Left");

//Set the ball clone rigid body to a new instance of the cannonball prefab

Rigidbody BallClone1 = Instantiate(CannonBall, transform.position, transform.rotation);

//Give the new projectile a velocity

BallClone1.velocity = -transform.right \* CannonBallSpeed;

//Give the new projectile a damage based on this instance of the enemies stats

BallClone1.GetComponent<CannonBallEnemy>().LoadDamage

((CannonBallDamage + (CannonBallDamage \* Random.Range(-0.3f, 0.03f))) \* Quality);

//Set the cooldown of the cannon to true

CannonCD = true;

//Set the cooldown time to now plus 3 seconds

CannonCDTime = Time.time + 3;

}

}

else if(!Dead)

{

//If the enemy is not within the shooting range and isnt dead enable the nav mesh agent to move the enemy to the player

nav.enabled = true;

//Make the enemy move to the player using the nav mesh agent

nav.SetDestination(GameObject.Find("Player").transform.position);

}

}

public void TakeDamage(float Amount)

{

//This function is activated when the player collides with an enemy cannon ball

//Minus the damage amount from the current enemy hp

HP -= Amount;

//Used for debugging purposes

Debug.Log("You hit.. " + transform.name + " for " + Amount + " damage.");

//Play the cannon ball hit sound effect

gameObject.transform.GetChild(1).GetComponent<AudioSource>().Play();

if (HP <= 0 && !Dead)

{

//If the enemy hit points is less than or equal to 0 and the enemy isn't currently dead

//Set the dead boolean to true

Dead = true;

//Play the sinking sound effect

GetComponent<AudioSource>().Play();

//Disable the 3d text tooltip

gameObject.transform.GetChild(0).GetComponent<MeshRenderer>().enabled = false;

//Get the current rotation of the enemy ship

Vector3 tipped = transform.eulerAngles;

tipped.z += 180.0f;

//Tip the ship upside down

transform.eulerAngles = tipped;

//Disable the enemies collided when sunk

GetComponent<BoxCollider>().enabled = false;

//Initialise temporary variables used for loot spawning

int randLootMin = 0;

int randLootMax = 0;

int randLootAmount = 0;

switch (transform.name)

{

//Set the maximum amount of loot depending on what type of enemy is spawning it and add gold to the players cargo

case "Sloop(Clone)":

randLootMin = 0;

randLootMax = 2;

GameObject.Find("Player").GetComponent<Combat>().Gold =

GameObject.Find("Player").GetComponent<Combat>().Gold + (int)Random.Range(2, 8);

break;

case "Caravel(Clone)":

randLootMin = 1;

randLootMax = 4;

GameObject.Find("Player").GetComponent<Combat>().Gold =

GameObject.Find("Player").GetComponent<Combat>().Gold + (int)Random.Range(4, 15);

break;

case "Brigantine(Clone)":

randLootMin = 3;

randLootMax = 8;

GameObject.Find("Player").GetComponent<Combat>().Gold =

GameObject.Find("Player").GetComponent<Combat>().Gold + (int)Random.Range(8, 25);

break;

default:

break;

}

//Begin spawning random loot amounts

float randLoot = 0.0f;

randLootAmount = Random.Range(randLootMin, randLootMax);

for (int i = 0; i <= randLootAmount; i++)

{

//Get the current position of the enemy ship

Vector3 vlootPos = transform.position;

//Add a random amount to the x and y of the enemy ship

vlootPos.x += Random.Range(-5.0f, 5.0f);

vlootPos.z += Random.Range(-5.0f, 5.0f);

//Create a new instance of the loot prefab

Rigidbody newLoot = Instantiate(Loot, vlootPos, transform.rotation);

switch (transform.name)

{

//Give the new instance a random type based on their drop rates and enemy ship type

case "Sloop(Clone)":

randLoot = Random.Range(0.0f, 100.0f);

if (randLoot >= 0.0f && randLoot <= 25.0f) //Grain 25%

{

newLoot.GetComponent<LootCrate>().TakeType("Grain");

}

else if (randLoot > 25.0f && randLoot <= 47.0f) //Fish 22%

{

newLoot.GetComponent<LootCrate>().TakeType("Fish");

}

else if (randLoot > 47.0f && randLoot <= 65.0f) //Oil 18%

{

newLoot.GetComponent<LootCrate>().TakeType("Oil");

}

else if (randLoot > 65.0f && randLoot <= 78.0f) //Wood 13%

{

newLoot.GetComponent<LootCrate>().TakeType("Wood");

}

else if (randLoot > 78.0f && randLoot <= 88.0f) //Brick 10%

{

newLoot.GetComponent<LootCrate>().TakeType("Brick");

}

else if (randLoot > 88.0f && randLoot <= 96.0f) //Iron 8%

{

newLoot.GetComponent<LootCrate>().TakeType("Iron");

}

else if (randLoot > 96.0f && randLoot <= 100.0f) //Rum 4%

{

newLoot.GetComponent<LootCrate>().TakeType("Rum");

}

break;

case "Caravel(Clone)":

randLoot = Random.Range(0.0f, 100.0f);

if (randLoot >= 0.0f && randLoot <= 22.0f) //Grain 22%

{

newLoot.GetComponent<LootCrate>().TakeType("Grain");

}

else if (randLoot > 22.0f && randLoot <= 41.0f) //Fish 19%

{

newLoot.GetComponent<LootCrate>().TakeType("Fish");

}

else if (randLoot > 41.0f && randLoot <= 56.0f) //Oil 15%

{

newLoot.GetComponent<LootCrate>().TakeType("Oil");

}

else if (randLoot > 56.0f && randLoot <= 68.0f) //Wood 12%

{

newLoot.GetComponent<LootCrate>().TakeType("Wood");

}

else if (randLoot > 68.0f && randLoot <= 79.0f) //Brick 11%

{

newLoot.GetComponent<LootCrate>().TakeType("Brick");

}

else if (randLoot > 79.0f && randLoot <= 88.0f) //Iron 9%

{

newLoot.GetComponent<LootCrate>().TakeType("Iron");

}

else if (randLoot > 88.0f && randLoot <= 94.0f) //Rum 6%

{

newLoot.GetComponent<LootCrate>().TakeType("Rum");

}

else if (randLoot > 94.0f && randLoot <= 98.0f) //Silk 4%

{

newLoot.GetComponent<LootCrate>().TakeType("Silk");

}

else if (randLoot > 98.0f && randLoot <= 100.0f) //Silverware 2%

{

newLoot.GetComponent<LootCrate>().TakeType("Silverware");

}

break;

case "Brigantine(Clone)":

randLoot = Random.Range(0.0f, 100.0f);

if (randLoot >= 0.0f && randLoot <= 18.0f) //Grain 18%

{

newLoot.GetComponent<LootCrate>().TakeType("Grain");

}

else if (randLoot > 18.0f && randLoot <= 34.0f) //Fish 16%

{

newLoot.GetComponent<LootCrate>().TakeType("Fish");

}

else if (randLoot > 34.0f && randLoot <= 47.0f) //Oil 13%

{

newLoot.GetComponent<LootCrate>().TakeType("Oil");

}

else if (randLoot > 47.0f && randLoot <= 59.0f) //Wood 12%

{

newLoot.GetComponent<LootCrate>().TakeType("Wood");

}

else if (randLoot > 59.0f && randLoot <= 70.0f) //Brick 11%

{

newLoot.GetComponent<LootCrate>().TakeType("Brick");

}

else if (randLoot > 70.0f && randLoot <= 80.0f) //Iron 10%

{

newLoot.GetComponent<LootCrate>().TakeType("Iron");

}

else if (randLoot > 80.0f && randLoot <= 88.0f) //Rum 8%

{

newLoot.GetComponent<LootCrate>().TakeType("Rum");

}

else if (randLoot > 88.0f && randLoot <= 93.0f) //Silk 5%

{

newLoot.GetComponent<LootCrate>().TakeType("Silk");

}

else if (randLoot > 93.0f && randLoot <= 97.0f) //Silverware 4%

{

newLoot.GetComponent<LootCrate>().TakeType("Silverware");

}

else if (randLoot > 97.0f && randLoot <= 100.0f) //Emerald 3%

{

newLoot.GetComponent<LootCrate>().TakeType("Silverware");

}

break;

default:

break;

}

}

}

}

}

### CannonBall Script

using System.Collections;

using System.Collections.Generic;

using UnityEngine;

public class CannonBall : MonoBehaviour {

//CannonBall class which is attached to the player's instances of the cannon ball prefab

//Time the local cannon ball expires

public float ExpiryTime;

//Time the local cannon ball was spawned

private float StartTime;

//The amount of damage the cannon ball will deal on impact

private float DamageAmount;

// Use this for initialization

void Start () {

//Set the start time of the cannon ball to the current game time

StartTime = Time.time;

}

// Update is called once per frame

void Update () {

if (Time.time >= StartTime + ExpiryTime)

{

//If the cannon ball has expired

//Play the splash sound effect

GetComponent<AudioSource>().Play();

//Destroy the cannon ball

Destroy(gameObject);

}

}

public void LoadDamage(float Damage)

{

//Function to load damage into the cannon ball from the player when spawned

DamageAmount = Damage;

}

private void OnTriggerEnter(Collider other)

{

if (other.CompareTag("Enemy"))

{

//If the cannon ball collided with an enemy

//Run the enemy's take damage function, giving it this instance's damage amount

other.gameObject.GetComponent<EnemyAI>().TakeDamage(DamageAmount);

//Destroy the cannon ball

Destroy(gameObject);

}

}

}

### CannonBallEnemy Script

using System.Collections;

using System.Collections.Generic;

using UnityEngine;

public class CannonBallEnemy : MonoBehaviour

{

//CannonBall class which is attached to the enemy's instances of the cannon ball prefab

//Time the local cannon ball expires

public float eExpiryTime;

//Time the local cannon ball was spawned

private float eStartTime;

//The amount of damage the cannon ball will deal on impact

private float DamageAmount;

// Use this for initialization

void Start()

{

//Set the start time of the cannon ball to the current game time

eStartTime = Time.time;

}

// Update is called once per frame

void Update()

{

if (Time.time >= eStartTime + eExpiryTime)

{

//If the cannon ball has expired

//Play the splash sound effect

GetComponent<AudioSource>().Play();

//Destroy the cannon ball

Destroy(gameObject);

}

}

public void LoadDamage(float Damage)

{

//Function to load damage into the cannon ball from the enemy when spawned

DamageAmount = Damage;

}

private void OnTriggerEnter(Collider other)

{

if (other.CompareTag("Player"))

{

//If the cannon ball collided with the player

//Run the player's take damage function, giving it this instance's damage amount

GameObject.Find("Player").GetComponent<Combat>().TakeDamage(DamageAmount);

//Destroy the cannon ball

Destroy(gameObject);

}

}

}

### LootCrate Script

using System.Collections;

using System.Collections.Generic;

using UnityEngine;

public class LootCrate : MonoBehaviour {

//Script attached to the spawned loot object when an enemy has died

//Variable to store the type of loot the crate is holding

public string LootType;

private void Awake()

{

//Start the countdown to sink the cargo in 20 seconds of it being created

Invoke("Sink", 20.0f);

}

private void FixedUpdate()

{

//Get the distance between this instance of the cargo and the player

float dist = Vector3.Distance(transform.position, GameObject.Find("Player").transform.position);

if (dist <= GameObject.Find("Player").GetComponent<Combat>().LootPullDistance)

{

//If the distance is within the loot pulling distance defined in the player's combat script

//Move the cargo towards the player

transform.position = Vector3.MoveTowards(transform.position, GameObject.Find("Player").transform.position, 0.1f);

}

}

void Sink()

{

//After 20 seconds of not being collected enabled the cargo's gravity

GetComponent<Rigidbody>().useGravity = true;

//Begin the countdown to destroy the gameobject

Invoke("DestroyCrate", 5.0f);

}

void DestroyCrate()

{

//If the countdown of 5 seconds reached 0 destroy the gameobject

Destroy(gameObject);

}

public string GiveType()

{

//Return the loot type of this instance of the loot

return LootType;

}

public void TakeType(string LT)

{

//Give this instance of the loot crate a loot type

LootType = LT;

}

}