FORCEPAGE

Object-Oriented Programming Documentation

Jagunmolu B Oke - 001191460

Table of Contents

1. Project Concept

The idea for the game comes from Contra top-down mixed with Star Wars for a sci-fi look.

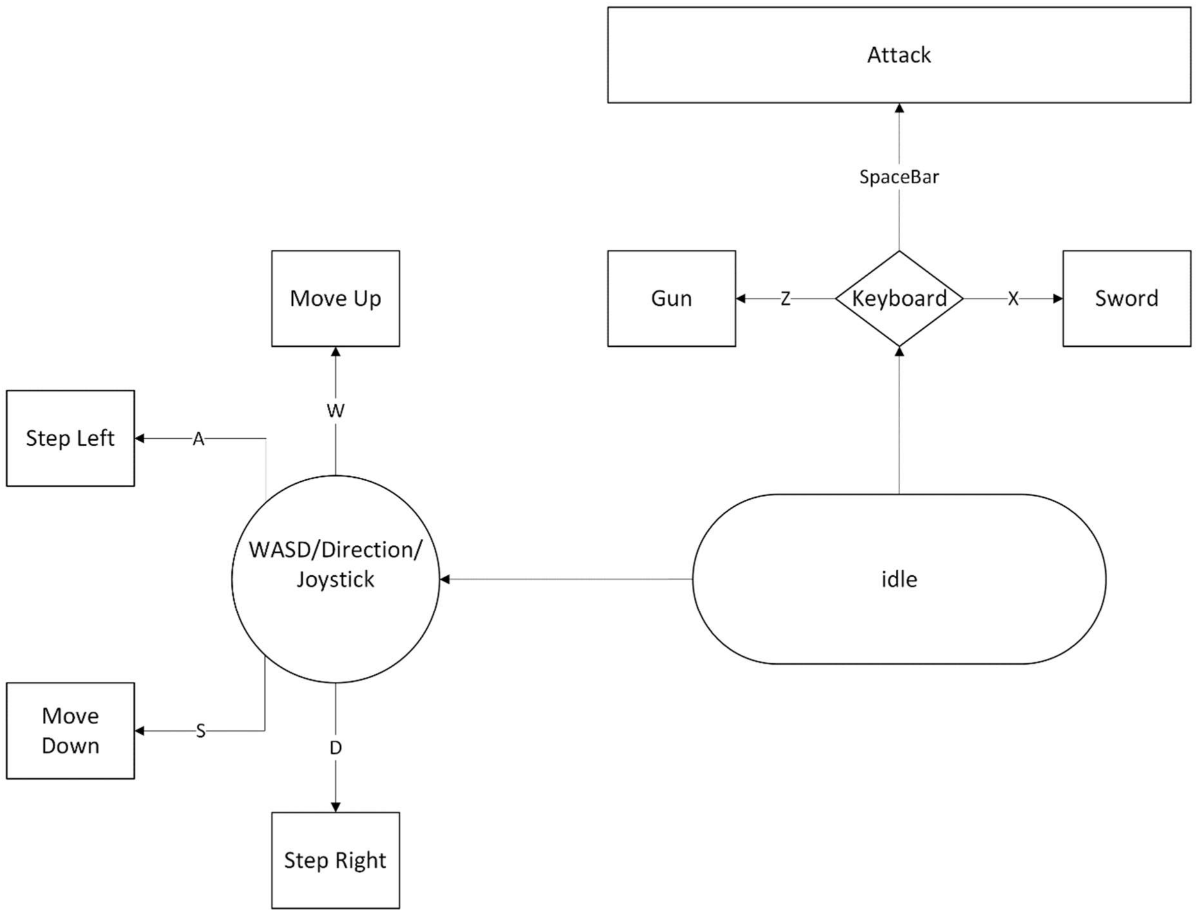


Figure 1.1:

The Player uses the WASD for navigation around the game world. The joystick can be an alternative to PlayStation (PS) and XBOX see table below

Joystick Inputs	Keyboard Input	Functions
Left Stick X-Axis	A and D (-1 or 1)	Left or right
Left Stick Y-Axis	W and S (1 or -1)	Up or down
Square (PS) or X (XBOX)	Space Bar	Attack
L1 (PS) or LB (XBOX)	Z	Switch to Gun
R1 (PS) or RB (XBOX)	X	Switch to Weapon
Option (PS and XBOX)	Esc	Pause and Un-pause

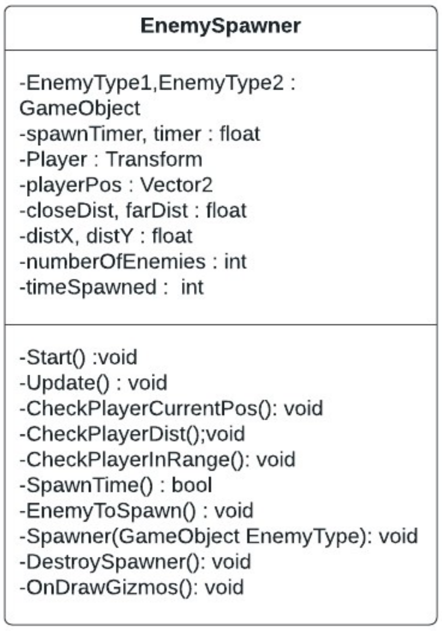
Table 1.1: Control Mapping

2. Graphics and Sounds

Some of the assets was downloaded from the asset store and sounds were downloaded from freesound, and some were edited using Audacity. See credits (p. 11). Walls assets were made in Photoshop idea from Games 1, Using a transparent Background a grid layout of even number 4 by 4 was created, a rectangle tool was used to create a rectangle shape occupying 2 columns and 1 row of the grid layout filled with a grey colour making the brick-like colour. The edge of the brick was created using Bevel and emboss. Which is then copied to the layer panel to duplicate the image and stack on top of each other using the grid layout for guides.

3. Programming

EnemySpawner one of the classes in the program.



```
EnemySpawner

-EnemyType1,EnemyType2 :
GameObject
-spawnTimer, timer : float
-Player : Transform
-playerPos : Vector2
-closeDist, farDist : float
-distX, distY : float
-numberOfEnemies : int
-timeSpawned :  int

-Start() :void
-Update() : void
-CheckPlayerCurrentPos(): void
-CheckPlayerDist();void
-CheckPlayerInRange(): void
-SpawnTime() : bool
-EnemyToSpawn() : void
-Spawner(GameObject EnemyType): void
-DestroySpawner(): void
-OnDrawGizmos(): void
```

Figure 3.1… Class Diagram of EnemySpawner

Attached to a gameObject called Spawner

Variables	Functions
EnemyType1 and EnemyType2	Storage for the enemy prefab variation.
spawnTimer and timer	Store and monitor time within the class.
Player	Used for player transform reference in the game world
closeDist and farDist	For monitoring player's closeness
distX and distY	Player's distance (X, Y) from the gameObject. EnemySpawner is attacked to on X and Y.
numberOfEnemies	Integer to store the number of enemies that can be spawned with Spawner
timeSpawned	Integer – keeps track of enemies spawned.

Table 3.1 Variables in EnemySpawner class

All variables in the table are declared within the class not in any method to make it globally accessible to all methods within the class.

All methods in the table below are void except SpawnTime ().

Methods	Functions
Start ()	Called once, to get the player transform reference.
Update ()	Called every frame and where most other methods are called
CheckPlayerCurrentPos ()	Called whenever Update () is evoked to check the player’s position.
CheckPlayerDist ()	checks player’s distance from the Spawner.
SpawnTime ()	Returns bool. Called in Update ().
CheckPlayerInRange ()	Checks the proximity of the player to farDistX and farDistY to decide to spawn an enemy.
EnemyToSpawn ()	Called in CheckPlayerInRange () when the player is in range and SpawnTime is met. Uses logic to decide when to call Spawner (GameObject EnemyType).
Spawner (GameObject EnemyType)	Takes in a gameObject type input. The input gameObject is used to know which type of enemy to instantiate. Keeps the number of Spawned GameObject
DestroySpawner ()	Ensures the number of timesSpawned is not greater than numberOfEnemies.
OnDrawGizmos ()	A unity engine method is used to draw gizmos in the scene for design to physically see the closeDist and farDist. With Gizmos.DrawWireSphere().

Table 3.2 Methods in EnemySpawner class

4. Object-Oriented Programming (OOP) Techniques

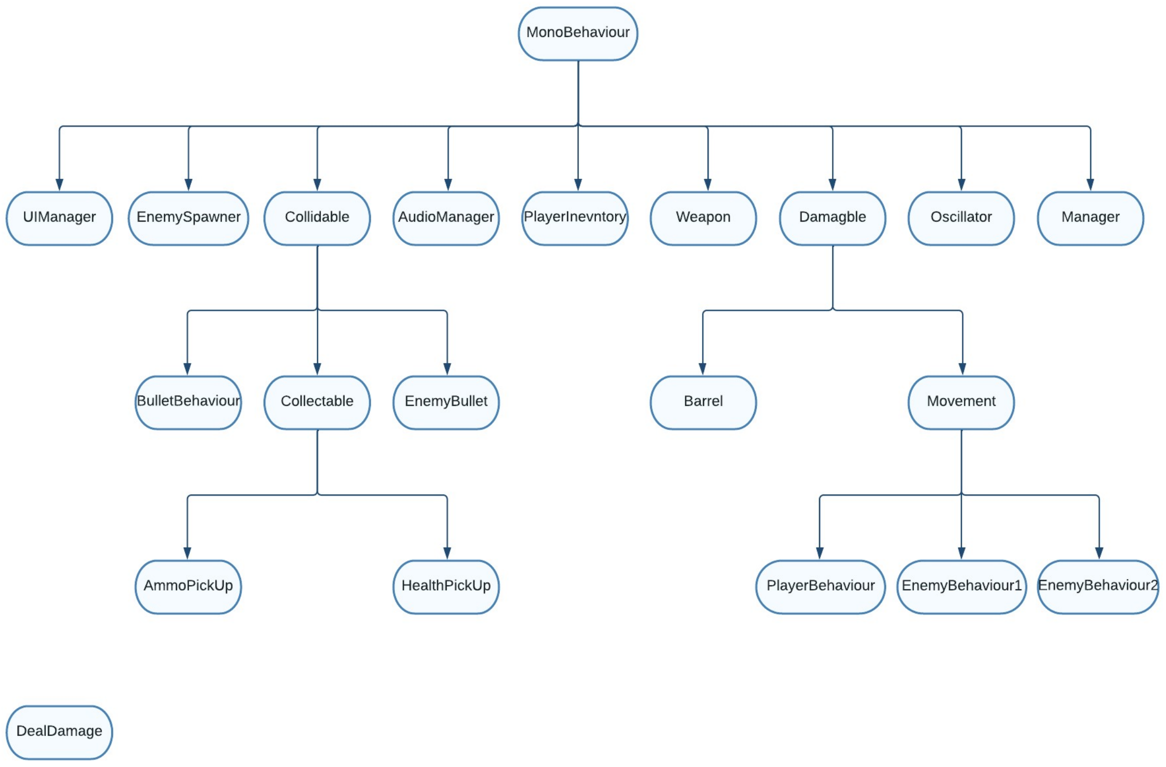
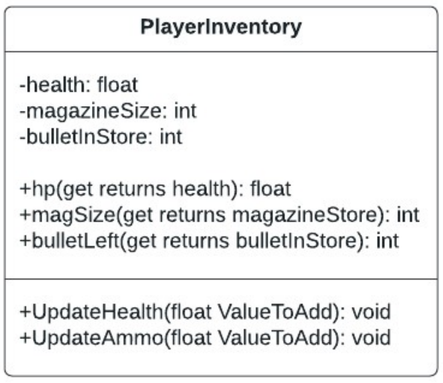


Figure 4.1 hierarchies of class in the program

4.1 Encapsulation



PlayerInventory

-health: float
-magazineSize: int
-bulletInStore: int

+hp(get returns health): float
+magSize(get returns magazineStore): int
+bulletLeft(get returns bulletInStore): int

+UpdateHealth(float ValueToAdd): void
+UpdateAmmo(float ValueToAdd): void

Figure 4.2 Class diagram of PlayerInventory

Encapsulation is the process in which no direct access is granted to the data; instead, it is hidden. If you want to gain access to the data, you must interact with the object responsible for the data. (Clark, 2013).

Variables with get property (read-only)	Function
hp	A get property that returns hidden health.
magSize	Returns magazineSize integer
bulletLeft	Returns bulletInStore

Table 4.1 Variables and their functions

The variables in the table return the values of stored variables that are hidden or out of scope to other classes. Examples are when the UIManager needs to display the player’s health, it interacts with hp that gives the health.

4.2 Inheritance

Inheritance in OOP is used to classify objects in a program according to common characteristics and functions. (Clark, 2013). A class that is been inherited from is called parent(base) class and one's inheriting is called child(derive) class.

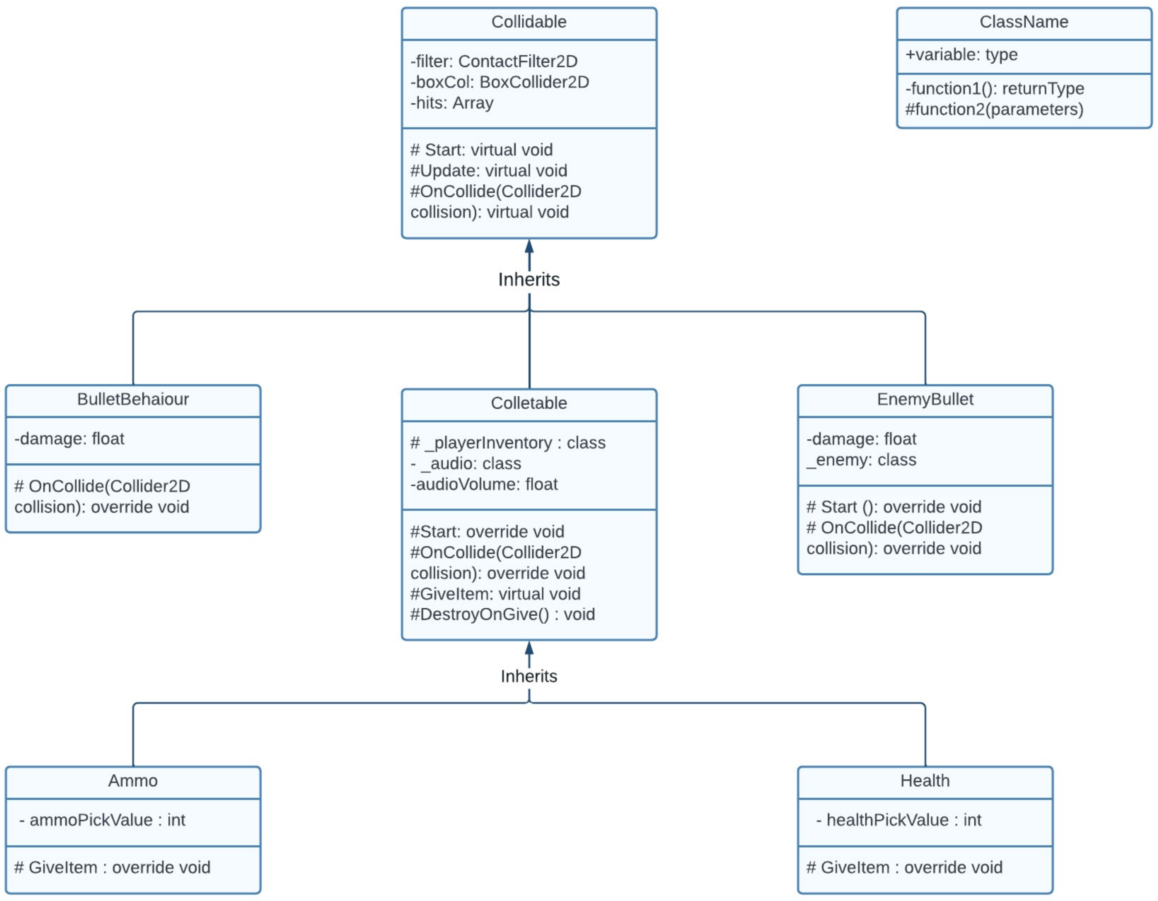


Figure 4.3: Shows the inheritance from base Collidable.

Class	Characteristic
Collectable (base Class)	Able to check collision, where classes that derive from it possess a similar function
BulletBehaviour, EnemyBullet, and Collectable (derived class)	All inherit from the base class, sharing collision detection property. While they possess extra characteristics that make them unique. Collectable awards player on collision while BulletBehaviour and deals damage.
Ammo and Health (sub-derive)	Awards players with unique items.

Table 4.2 Class in the group of objects that a collidable with functions

4.3 Polymorphism

Polymorphism is the ability of two objects to respond to the same request message in their own unique way (Clark, 2013).

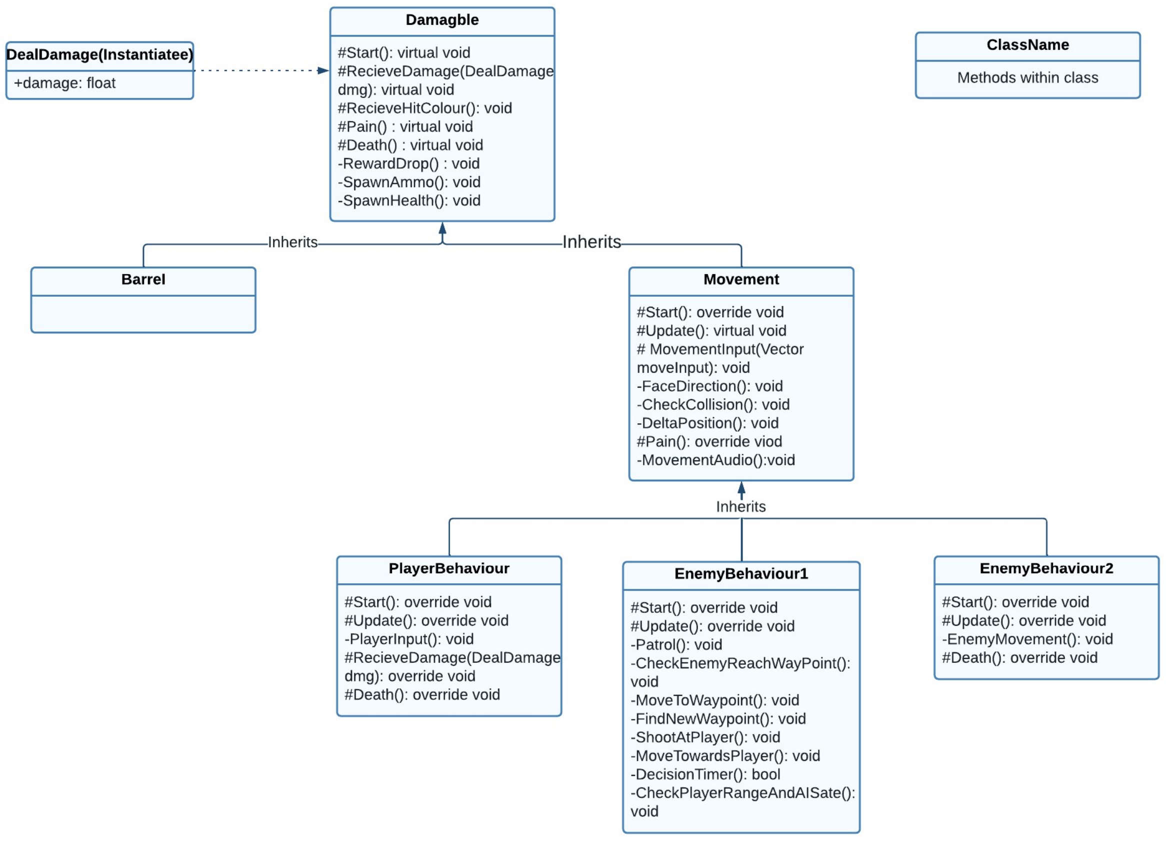


Figure 4.4 Class Diagram of Damagble (base) with its derived classes

Methods declared in a base and including the virtual keyword allow that method to be overridden in derived classes.

Base class = protected virtual returnType methodName(parameter) {block of code;}

Derived class = protected override returnType methodName(parameter) {unique block of code;}

Protected is used to make it private but visible deriving classes.

Virtual Methods(base)	Function in Barrel	Function in Movement (Enemy)	Function in Movement (Player)
Pain ()	Plays audio of breaking box	Plays audio of a male grunting.	Plays audio of a male grunting.
Death ()	Destroys gameObject and calls the method to award player	Destroys gameObject and award	Calls a method GameOver from Ui using reference.

Table 4.3 Methods that are overridden from the base class

5. Credits

i. Props (bench, lamp post,) by Caino from Unity Assets Store.
ii. Props (Swords) by LayerLab from Unity Assets Store.
iii. Wall, Characters, and Streets (graphics) made in Photoshop.
iv. Death (sound) by tonsil5 from freesound.
v. Wood destruction (sound) by Bertsz from freesound.
vi. Game (sound) by Xythe from freesound.
vii. Laser (shooting sound) by Bolkmar from freesound.
viii. Melee (sound) by Eminyidirium from freesound.
ix. Reward Pick Up (sound) by
x. Reward humming (sound) by Blukotek from freesound.
xi. Walking (sound) by Zepurple from freesound.

6. References

Borromeo, N. A., 2021. *Hands-On Unity 2021 Game Development.* Second Edition ed. Birmingham - Mumbai: Packt.

Clark, D., 2013. *Beginning C# Object-Oriented Programming.* Second Edition ed. New York: Apress.

Harrision, F., 2021. *Learning C# by Developong Games with Unity 2021.* Sixth ed. Birmingham: Packt.

McGrath, M., 2020 . *C# Programming.* 2nd ed. Warwickshire : In easy steps .

Sharp, J., 2018. *Microsoft Visual C# Step by Step.* Ninth Edition ed. U.S: Pearson Education, Inc.