



# UNDEAD SURVIVOR

**Course:** Algorithms & Data Structures

**Instructor:** Vi Chi Thanh



**Submitted by:** Mai Nguyễn Hoàng

**Student ID:** ITITIU21208

**Subject Code:** IT013IU Group 3 – Lab Group 2

## Table of Contents

I. Abstract.....	2
II. Introduction.....	2
III. Design chart .....	3
IV. Data Structure, Algorithms, and Design Pattern .....	3
1. Data structures.....	3
2. Algorithms.....	4
V. Demonstration .....	6
IV. Git explanation .....	8

## I. Abstract.

*Vampire Survivors* is a gothic horror casual game with roguelite and roguelike elements, developed and published by indie developer Luca Galante (a.k.a Poncle) using Unity – the popular and powerful game engine that supports cross-platform development, many features and tools for game creation.

The game was originally released as a browser game on itch.io in 2021 and later ported to various platforms, including Android, iOS, Windows, macOS, and Xbox.

In the game, the player has to survive until dawn (when the timer reaches 0), using various weapons, items, and skills. The game features pixel graphics, bullet hell mechanics, and randomized levels.

This project aimed to remake the game and also implement Data Structure and Algorithm (DSA) when making the game mechanic. The result of this project is a simple remake of the game that preserves the original simple control system and hardcore gameplay named **Undead Survivor**.

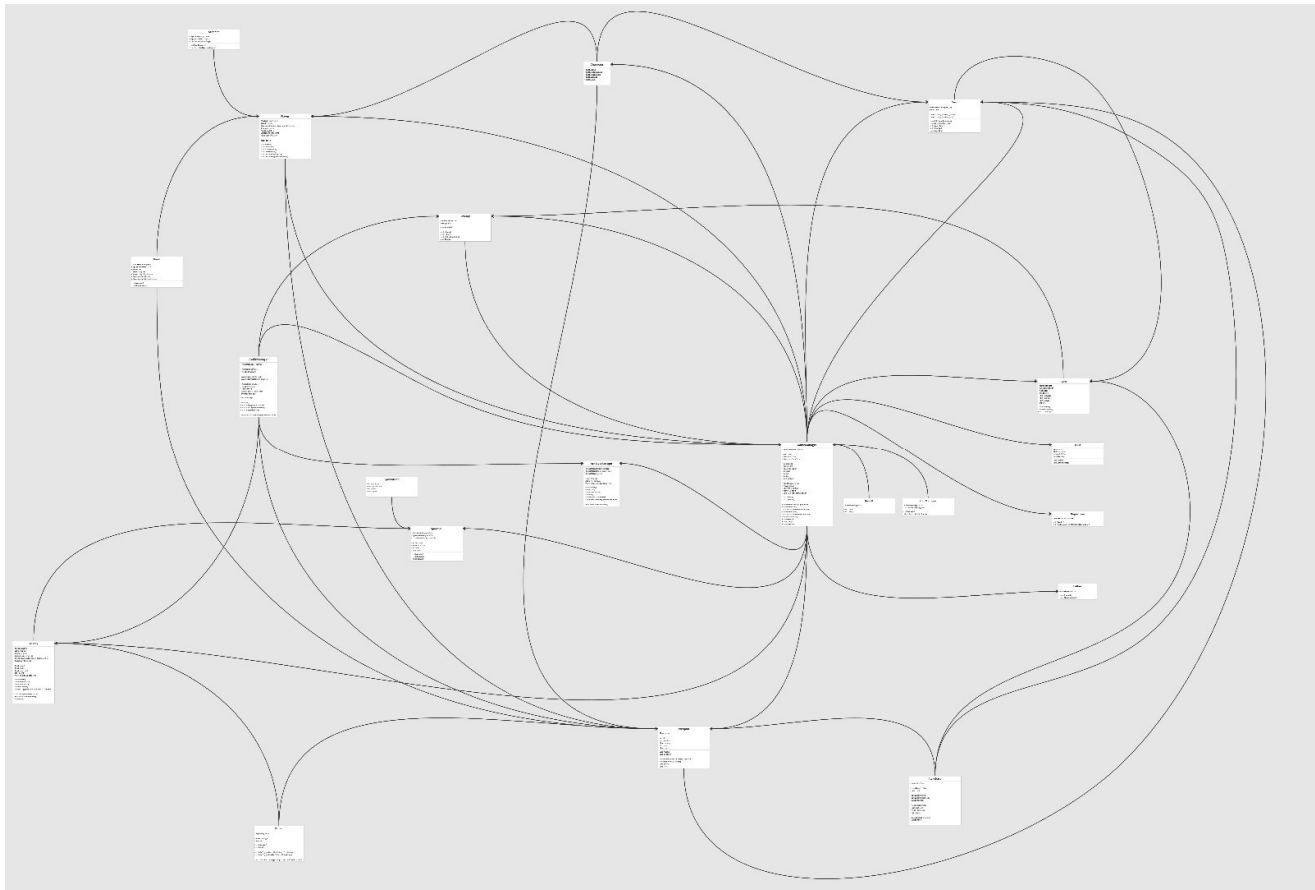
## II. Introduction

I choose to remake *Vampire Survivors* because, with my experience in making small games, Roguelike is one of the easiest genres to make. In that genre, *Vampire Survivors* is the first one that comes to my mind when thinking about a simple but addicting game. I also found a free game assets pack inspired by *Vampire Survivors*.

**Undead Survivor** uses the same Engine as the original *Vampire Survivors*, which is a personal preference since I also want to take this opportunity to learn and make a complete game using Unity. By using Unity, this project was made in C# with Unity MonoBehavior (a class that provides the framework for scripting in Unity)

The two (2) main goals of this project are to make a complete game (with mechanics, UI, sound,...) and use DSA in the game mechanics

### III. Design chart



For better navigation, use the link here: [https://miro.com/app/board/uXjVN\\_BPq2Y=/](https://miro.com/app/board/uXjVN_BPq2Y=/)

The diagram above shows all of the classes of the game and how they interact with each other.

The arrow **leaves from top/bottom** shows that **this class is being called** by the destination class

The arrow **point to the left/right** shows that **this class is call** the class point to it

### IV. Data structures and Algorithms

#### 1. Data structures

The game is mostly uses **Arrays** and **List** to store and manage multiple objects of the same type:

**Some Arrays used:**

DSA-Game - AchieveManager.cs

```
public GameObject[] lockCharacter;
public GameObject[] unlockCharacter;
```

DSA-Game - GameManager.cs

```
public int[] nextExp = { 3, 5, 10, 100, 150, 210, 280, 360, 450, 600 };
```

DSA-Game - LevelUp.cs

```
Item[] items;
```

DSA-Game - PoolManager.cs

```
public GameObject[] prefabs;
```

DSA-Game - Spawner.cs

```
public Transform[] spawnPoint; // Spawns's location
public SpawnData[] spawnData; // Enemy's data
```

DSA-Game - Scanner.cs

```
// array of targets around the player
public RaycastHit2D[] targets;
```

Those variables are defined as **Array** since the game logic that is needed is mostly based on the Array's length to scale/repeat the calculation.

### Some Lists used:

DSA-Game - PoolManager.cs

```
List<GameObject>[] pools;
```

DSA-Game - Spawner.cs

```
private List<SpawnData> spawnList; // List of Enemies to spawn
```

I intended to use **List** for spawnList instead of *Queue* or *Stack* although *Queue/Stack* can add more objects into the *spawnList* the same way List does.

```
DSA-Game - Spawner.cs

// when level reach the next level, add Stronger Enemy to spawnList
if (level != lastLevel)
{
    spawnList.Add(spawnData[level]);
    lastLevel = level;
}
```

It is because in the **Spawn()** function, the game does not clear the *spawnList* so it will grow bigger when the game goes on to spawn more **Enemy** into the game instead of **dequeue** or **pop** the data out to spawn.

```
DSA-Game - Spawner.cs

void Spawn()
{
    // For each Enemy in spawnList
    for (int i = 0; i < spawnList.Count; i++)
    {
        GameObject enemy = GameManager.instance.pool.Get(0); // 0 mean Enemy in Pool
        enemy.transform.position = spawnPoint[Random.Range(1, spawnPoint.Length)].position;
        enemy.GetComponent<Enemy>().Init(spawnList[i]); // Init Enemy with spawnData
    }
}
```

## 2. Algorithms

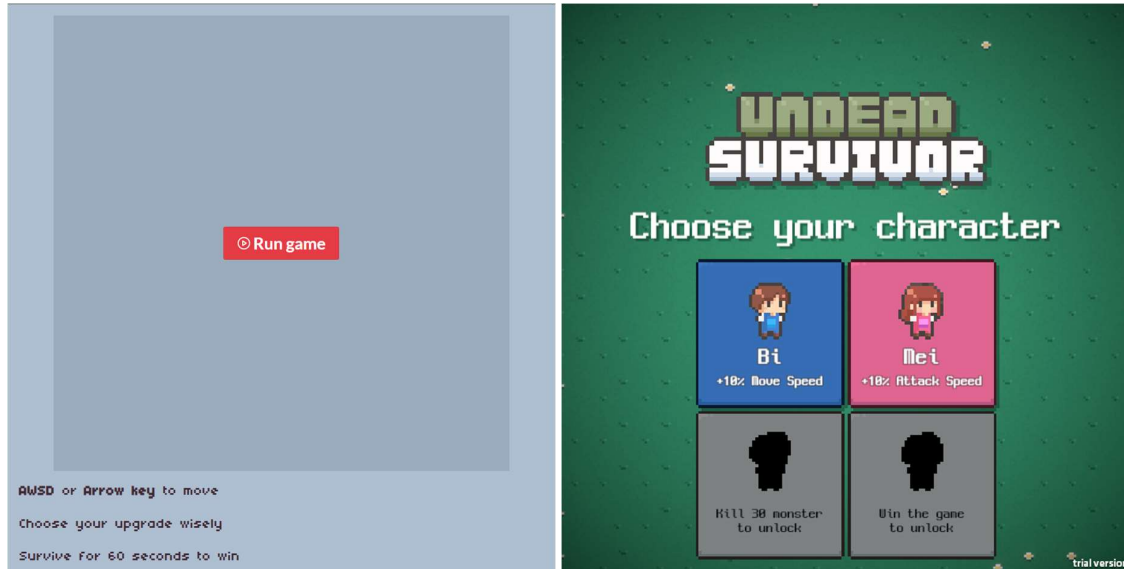
Since the game doesn't have any complex mechanics, I don't use any complex Algorithms. However, there are still some interesting common programming patterns and techniques:


- **Singleton Pattern:** the **GameManager** has a public static variable *that holds* a reference to the single instance of **GameManager** so that when other classes want to get any universal variable, the variables are taken from only one **GameManager**.
- **Game State Management:** The **GameManager** class manages the game state, including starting the game, winning the game, losing the game, and retrying the game.
- **Object Pooling:** This is a common optimization technique in game development. It reduces the overhead of frequently creating and destroying **GameObjects**, by reusing them instead.
- **Spawn Mechanism + Level Progression:** The Spawner class spawns enemies at random spawn points (defined in the Unity editor) based on the enemy's SpawnData, scaled with the current level (a.k.a difficulty scaling), which changes over time to add stronger enemies.

## V. Demonstration

**Disclaimer:** Since the game is made in Unity using C#, it's not as flexible as using Java. I have built the game and uploaded it to **itch.io** so that **Windows/MacOS devices** or even Mobile devices can open and play it (although Mobile is not intended, you can still move the player if you have a connected controller/keyboard).



Firstly, open the game using this link: <https://hoenmike.itch.io/undeadsurvival>

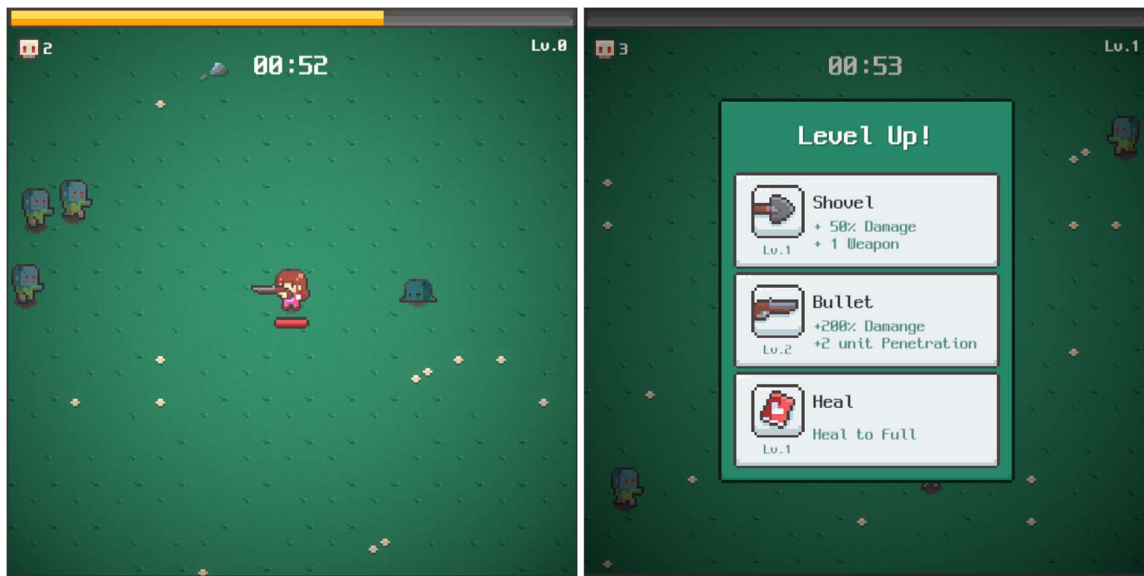


The website will look similar to this. Press the  button to start the game.

The game will start and you have two characters to choose from, each one gives you a different trait buff (even a different starting weapon) the other two characters will open when you play the game and reach its condition.

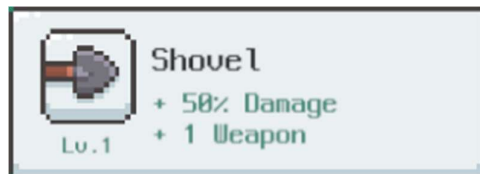
After choosing a character, the game will start, the game has minimal UI elements:

Beside a skull  icon is the **Kill count**. The text on the right  is the **Level** of the Player. A long bar on the top will fill up when you kill enemies, once it is full, you will **Level Up**.

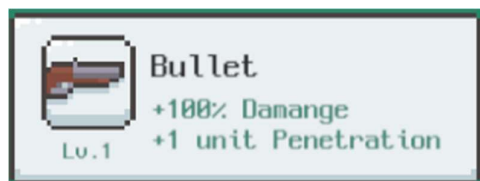


When **Level Up**, you can choose 1 over 3 **upgrades** which makes the player stronger to survive the later game (the game will pause so that you can choose carefully).

Those upgrades are:



Add more rotating **Shovel** around Player



Increase damage & penetration to **Gun**



Shovel **rotates faster**, Gun **shot faster**



Player **move faster**



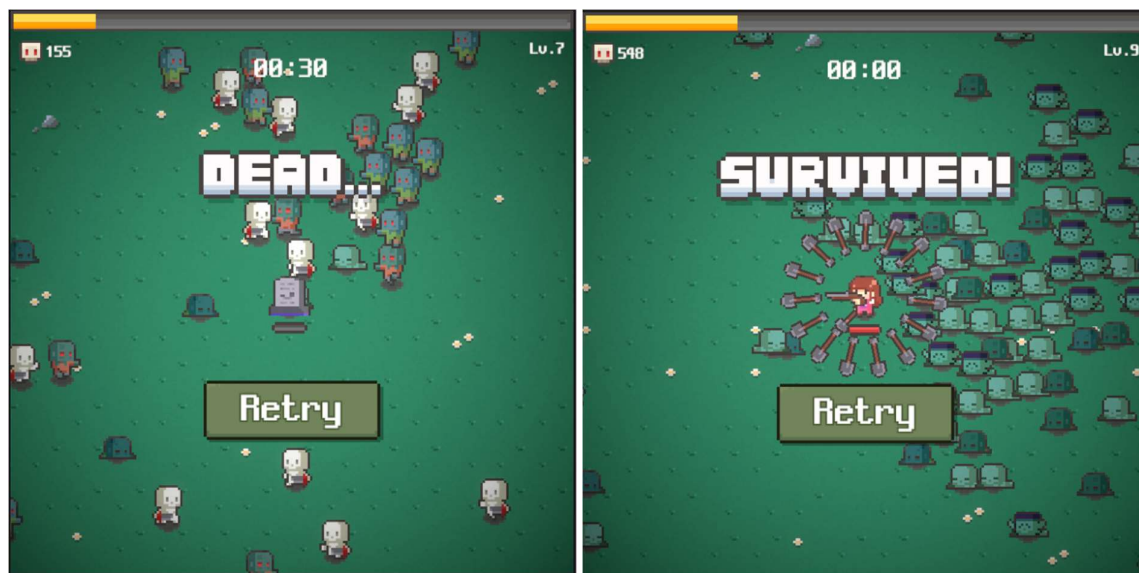
Fully **heal** Player



As mentioned earlier, when playing the game, you will unlock new Characters with stronger trait, the notification when you unlock new characters will be like this.



There is a small red bar under the Player, that's your **HP**. If the Enemy touch you, you will lose 1/100HP, when health is below 0, you will **Lose**. If you manage to survive until the **Timer** in the middle runs out, you will **Win**. When you Lose or Win, you can press **Retry** to get back to the characters selection and play again. **Have Fun!**



## VI. Git explanation

To save pages, I will crop out the Git commits and paste them side by side, the read order is from **left to right** (for each picture) and from **bottom to top** (for each commit)

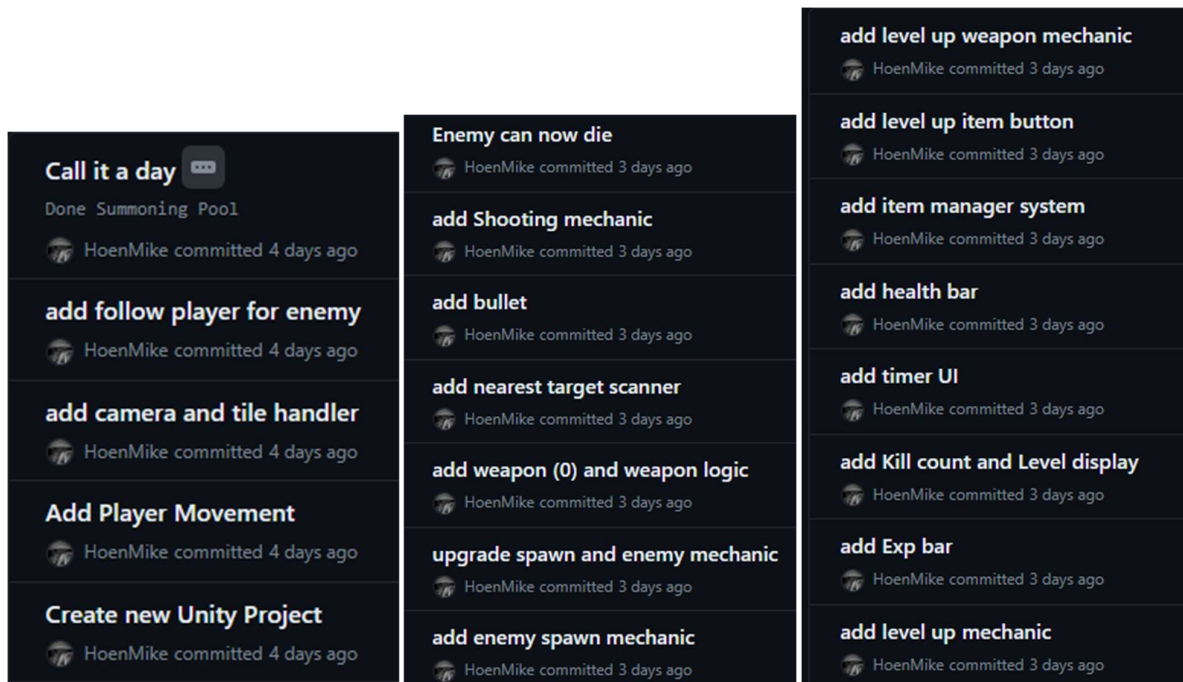
GitHub repository: <https://github.com/HoenMike/DSA-Game>

(1) Making the **Player that can move** with a **following Camera**. A **tile handler** to make the ground and **Enemy** that can follow the **Player**.

(2) After that I make the **Spawn mechanic, Weapon** (both Mele and Range) then to test the weapon, I make the Enemy can **detect bullet collision and die**

(3) Then I move to the UI aspect of the game. I first create prototypes of each button to test the level-up mechanic. I also add a **Health bar, Timer, Kill count, and player's Level display**.





(4) I create each type of **Gear** that increases Player's strength

(5) Making the **Start Scene** with **Character selection** (each Character has a different initial and stat scale)

(6) Add **Background music, Sound effects and sound manager**.

(-) The rest is mostly game **balancing**, and finalizing the project.

