https://github.com/CuDevlin/UltimateGMDProject

Authors:

Devlin Onichuk (325592) aka CuDevlin

Adrian Bugiel (325618) aka adrianbugiel or Slardarz

Personal Reflections:

Devlin:

Reflecting upon my experience of the GMD final project, where we created a 2D, VIA Survivor game, to start with how the scripting went, it was an up and down ride. With starting the project early in the semester and seeing separation of responsibilities which was not taken into enough account early on. This had to be addressed, and I was able to separate the responsibilities, and make the components more modular.

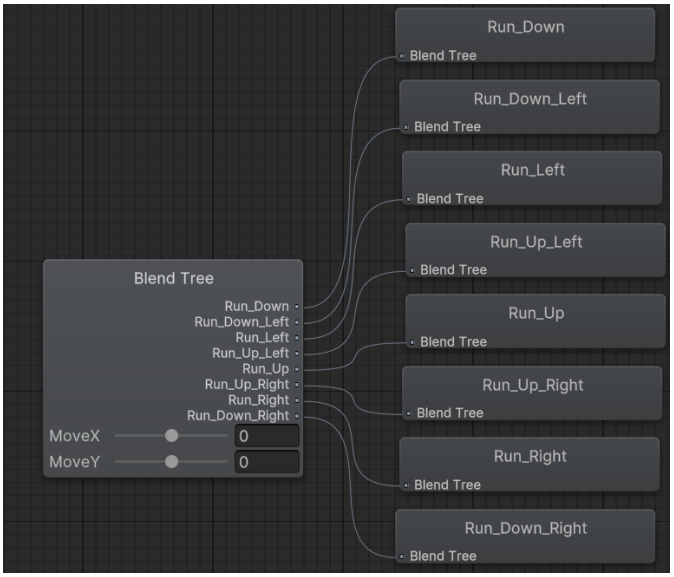
The scripting itself was normally straightforward but took an extensive amount of time and research of how certain gameplay mechanics would work, or how they should work. For example, the Damage Receiver was meant for both player and enemy, but as development continued it was decided this was unnecessary for the implementation of damage to enemies due to the damage scaling that Adrian implemented.

Looking back at these changes, we could have kept to a better separation as we continued, as some responsibilities could still be separated. Although I believe we handled with well given the time and scale for our ambition of the project.

To handle player movement, the built-in unity input system was used, where we could configure the movements and actions for multiple devices. So this aspect of the game for movements are configured by us but handled by unity.

```
12 references
private Vector2 moveInput;
6 references
private Vector2 lookInput;
3 references
private Vector2 lastMoveDirection = Vector2.up;
4 references
private Vector2 lastLookDirection = Vector2.right;
```

These vectors were used in the Player Controller. This was used to capture in current direction the player is moving, and if they stopped what was the last movement direction. This allows us to apply these vectors for the animations, since there is multiple animations and states where the player could be looking. It was important to handle how the players character would look with the given inputs.



The animations themselves focused more on the player, as seen above the player has a more in depth 8 directional animations, for movement as well as while idle. This significantly added life to the character from its original 2d box form. The enemies have simpler 3 frames to show some life but are not multi-directional based on their own movements.

Looking back, I wished we would have spent more time earlier on as this is one of the lasts sections we touched was with animations. As our focus was more on the gameplay and mechanics we wanted to implement, although I believed this still helped us make a more complete game.

Game physics in our case are minimal, there are collisions between the player, the environment, as well as the enemies. Which allows enemies to box in the player and gives the player obstacles to work around. This also works well with the current AI, as the enemies ai is as it stands only setup with a simple algorithm to track the player. This means the enemies do not avoid obstacles. I would like to in the future add either a more complex algorithm or make better use of the Nav-mesh. This would make pathing for the enemies better and give a higher difficulty with the enemies, so they do not just get stuck.

Our game uses game managers, to handle the UI state, as well as the game states. For example, the experience manager, handles all the interactions and uses for leveling up and the gaining of experience for the player.

I implemented the MainMenu, PauseMenu, and DeathScreen. While Adrian focused on the Level up mechanics and UI. Although the UI itself uses a RPG style. The buttons use multiple sprites to show when a button is hovered and selected by the player. This also allows on the arcade machine to show where you are in the menus. Currently the settings button has no function but, in the future, I would have liked to add a controls UI, as well as volume settings.

Overall, I believe we covered all areas required for the project in various degrees. Where we focused in some respects shows as we do well in game mechanics, UI, and the animations. We do lack in the AI aspects. But I believe this is to be expected and in the future, we would look to upgrade the AI and add more enemies, as well as different buffs and skills the player would be able to use. This has been an amazing opportunity, and I look forward to

Adrian:

During this project, I had the opportunity to work across multiple core Unity systems, gaining practical experience in various aspects of game development. Since the project took place during my final semester at VIA University College, it was particularly challenging to balance the limited available time with the demands of my bachelor project and other elective courses. Despite these constraints, I believe we made meaningful progress in several key areas of our *Vampire Survivors*-inspired game. This document serves as a reflection on the systems I contributed to and the lessons I gathered throughout the development process.

Scripting & Game Logic

Working with MonoBehaviours components, events and manager systems, I had an opportunity to develop core gameplay features such as the PowerUpManager, which coordinates upgrade choices across multiple classes tracking player's main statistics like health, damage or projectile count. This system was built around event-driven design, particularly in context of an ExperienceManager to which other components were subscribed to. This allowed for more modular code and helped me to use Unity's lifecycle methods, improving code clarity and debuggability.

```
15  public class PowerUpManager : MonoBehaviour
16  {
17      private HealthComponent healthComponent = null;
18      private DamageDealer damageDealer = null;
19      private PlayerController playerController = null;
20      private ProjectileShooter projectileShooter = null;
21
22      private void OnEnable()
23      {
24          ExperienceManager.Instance.OnLevelUp += HandleLevelUp;
25      }
26
27      private void OnDisable()
28      {
29          ExperienceManager.Instance.OnLevelUp -= HandleLevelUp;
30      }
31
32      public void RegisterPlayer(GameObject player)
33      {
34          healthComponent = player.GetComponent<HealthComponent>();
35          damageDealer = player.GetComponent<DamageDealer>();
36          playerController = player.GetComponent<PlayerController>();
37          projectileShooter = player.GetComponent<ProjectileShooter>();
38      }
39
40      private void HandleLevelUp(int level)
41      {
42          ShowUpgradeChoice();
43      }
```

Graphics & Audio

In terms of visual and audio feedback, I added ambient audio, sound effects for player actions and key events like levelling up or taking damage. I organized all game sounds through a central SoundManager that used separate two AudioSource components for music and SFX. This setup gave me a better understanding of audio layering and prioritization in Unity. Finding high-quality assets to match our game's style proved time-consuming, and in some cases, I recorded or extracted samples from online sources to meet our needs. For visuals, I used the Sora AI tool to generate custom sprite frames, enabling us to define a unique look for each enemy type.

Animation

I integrated together with my colleague sprite animations into enemy characters using Unity's Animator system. While the animations were relatively simple (3-4 different frames) , I gained experience in using animation controllers, creating transitions between states, and linking animation to game logic.

Game Architecture

My main contribution in this area was creating manager systems (EnemySpawner, SoundManager, ExperienceManager, PowerUpManager) that focus on organising code using single responsibility principle. Though I did not fully apply complete SOLID principles, I kept responsibilities clearly separated and used prefabs to improve reusability. This approach helped me think about relatively maintainable game structure under time constraints and constantly changing game vision.

User Interface

I built UI elements using Unity UI Toolkit, including a fully functional upgrade selection screen that appears during level-ups. The interface interacts with the gameplay systems, handles pause logic, and allows players to choose from one of two randomized upgrades, giving player a sense of progression. Integrating logic with visuals helped me understand user interaction in Unity and how it connects to game's main systems.

Conclusion

As the project allowed me to work across many key areas of Unity development like audio system or writing scripts that handle the game's state, it was a great learning opportunity despite significant time pressure. The result of our *Via Survivors* game is fully playable and makes me proud of the progress we achieved in this project. My final observation from this short project is that balancing the game's progression with the game's flow can be really difficult, and I can fully understand now how much iteration, testing, and player feedback is required to achieve a well-paced and engaging experience.