## Assignment Task

We need to create a "Metal Slug 6" game using Unity3D. The game should be a standalone application with at least three levels, featuring smooth graphics and a gradual increase in difficulty. We're encouraged to be creative and add at least two new features. The game will include basic mechanics like player movement, shooting, a health bar, different types of enemies, sound effects, and displaying player stats like health, score, and current weapon.

**Key Requirements:**

1. **Player Controls:** Our game should let the player move left/right, jump, crouch, and shoot.
2. **Health System:** The player will have a health bar that decreases when hit by enemies.
3. **Enemies:** We need to include a variety of enemies like aliens, soldiers, and mechanical foes.
4. **Display Elements:** The game should show the player's health, current weapon, and score on the screen.
5. **Sound Effects:** We should incorporate sounds for shooting, explosions, enemy attacks, power-ups, etc.

**Level Design:**

- We can use themes like jungle, base, alien ship, and fortress for our levels.
- Each level should include platforms, traps, and moving hazards.
- The levels will end with a boss battle.

**Bonus Features:**

- Players will start with a basic rifle and can pick up power-up weapons like spread shots and laser beams.
- Power-ups can be collected from defeated enemies or found in hidden areas.
- We can implement a system where players have limited lives and can earn extra lives.
- If we want, we can include multiplayer mode and let players choose different characters.

## Installing unity:

- Download Unity Hub from the official Unity website.
- Open Unity Hub and go to the "Installs" tab.
- Click "Add" to choose and install a Unity version.
- Select necessary components (e.g., editor, platform support, Visual Studio).
- Click "Next" and wait for the installation to complete.

- Log in with your Unity account through Unity Hub.
- Activate the license (choose the personal license if applicable).
- After activation, start creating projects in Unity.


## Introduction:

In this assignment, we developed a 2D Unity game called Ninja-Run, inspired by the classic Metal Slug series. The game's primary objective was to create an engaging and immersive experience for players, complemented by visually appealing graphics and smooth gameplay. We aimed to provide players with an enjoyable adventure that keeps them coming back for more.

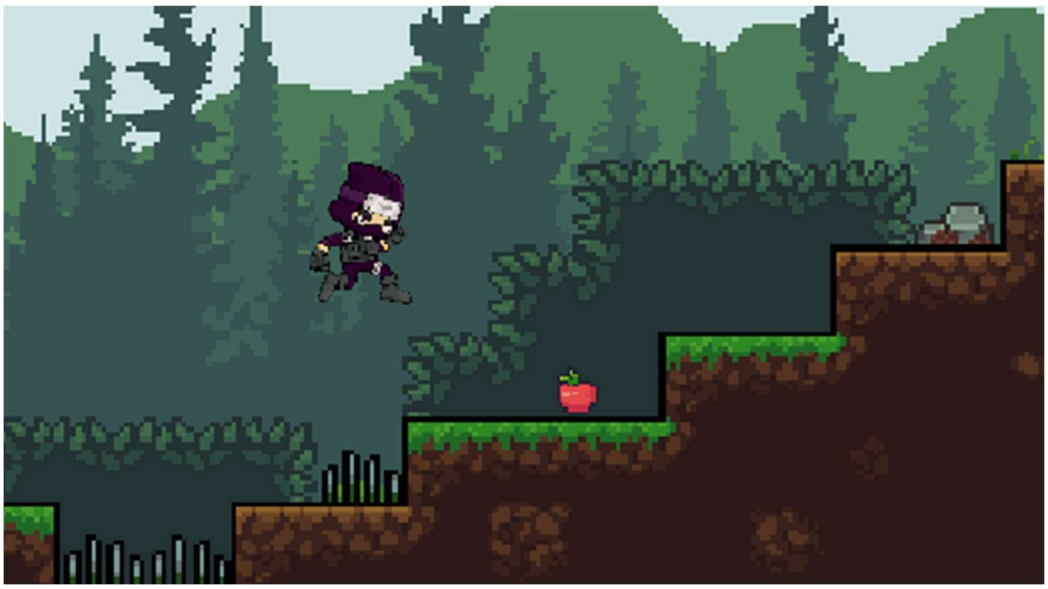
**Game Concept:**

Ninja-Run revolves around a skilled ninja character named Naruto, who embarks on an epic journey through various levels, ranging from serene natural landscapes to the fiery depths of hell. Throughout this journey, Naruto faces a variety of formidable enemies, including samurai warriors, powerful mages, and relentless zombies. Each level introduces unique challenges and environmental elements that add depth to the gameplay.

At the climax of each level, Naruto encounters a challenging Boss enemy. These bosses are not just ordinary foes; they are designed to chase the player within a certain range, providing an exhilarating chase experience. With high attack power and robust defense, these bosses present a significant challenge that requires players to strategize and adapt their gameplay to defeat them.

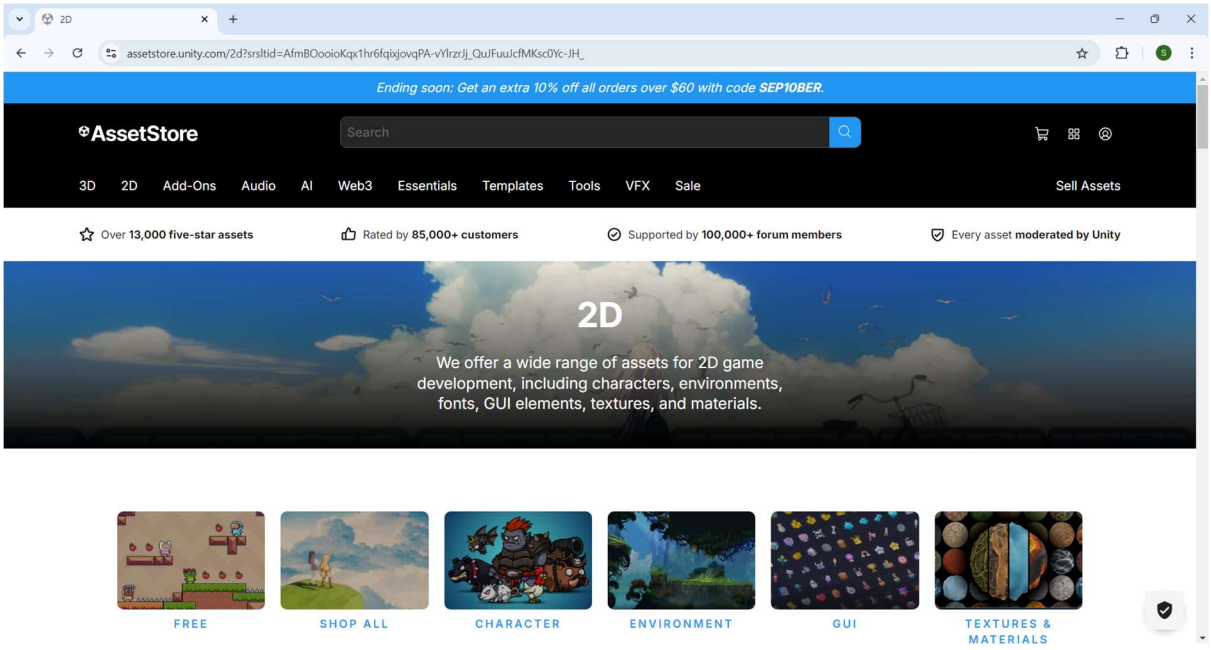
To enhance the overall experience, all enemies in the game are equipped with high-quality animations. These animations include various actions such as attacking the player, performing death animations upon defeat, and displaying hurt reactions when struck by the player's attacks. This attention to detail in animation ensures that players are fully immersed in the game, making each encounter feel dynamic and engaging.

Overall, Ninja-Run aims to combine thrilling gameplay mechanics with captivating visuals, creating a memorable experience for players as they guide Naruto through this exciting adventure.

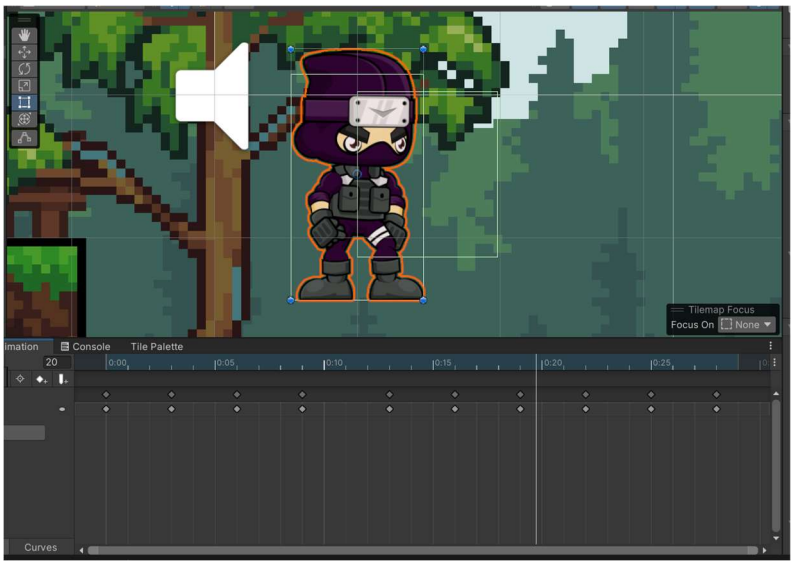


# Development Process

In the development process, we began by sourcing the necessary sprites for the player and different enemies from various online platforms, such as the Unity 2D Asset Store and other internet resources. After gathering these sprites, some of them required adjustments using Unity's Sprite Renderer to ensure they retained high-quality visuals and were correctly displayed in the game environment.

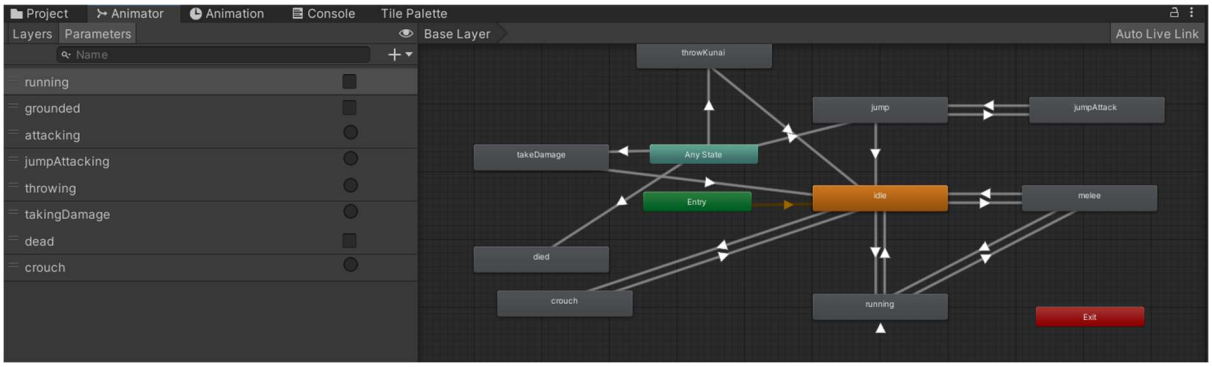



With all the sprites in place, our first step was to create animations for the player character. We started by setting up basic movement animations like walking, running, and jumping. Following that, we developed animations for different attack moves—such as slashing with a knife, swinging a sword, throwing kunai, and shooting fireballs. Each attack animation was carefully crafted to look distinct and fluid, matching the fast-paced nature of a ninja's combat style. Alongside these, we also designed animations for non-combat states, such as getting hurt (for when the player takes damage) and a death animation (triggered when health reaches zero).



Once these animations were ready, we moved on to using the Animator component in Unity to connect and control them. We created various states within the Animator, setting up transitions based on specific parameters like speed, attack type, and health status. For example, we linked the player's walk animation to trigger whenever horizontal movement was detected and set conditions to switch from one attack animation to another depending on the input command.

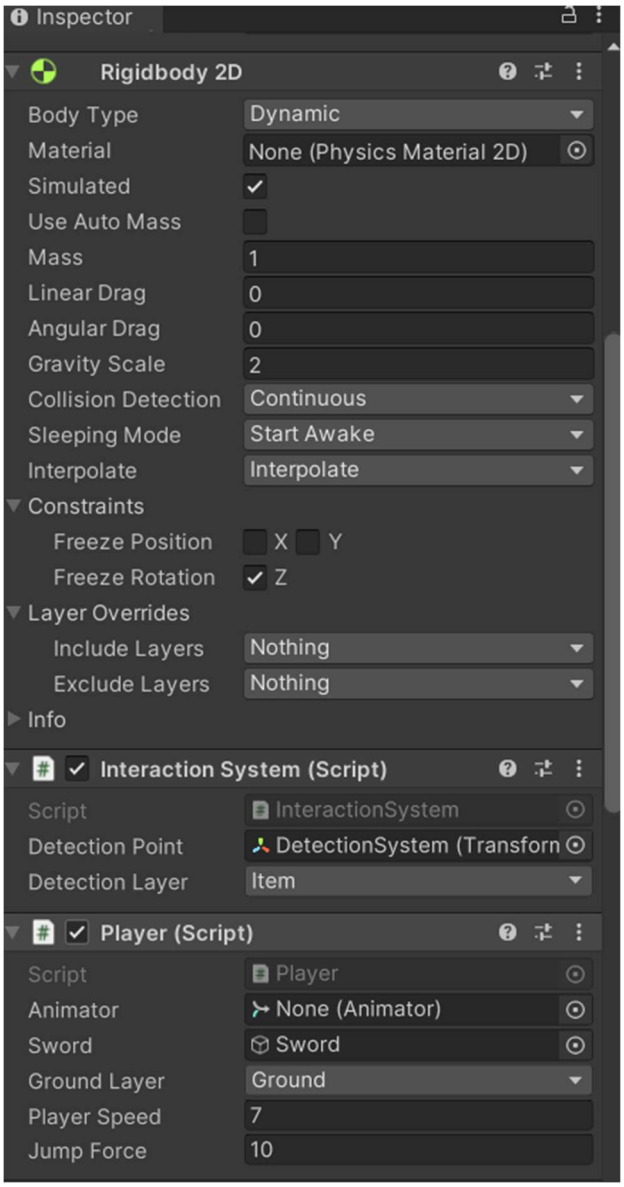
We then coded the player's behavior to ensure a seamless interaction between gameplay and animations. This included implementing health-based triggers—such as switching to the hurt animation when the player takes damage from enemies. For the death animation, we set up a condition in the code that monitors the player's health, ensuring the character smoothly transitions to the death state once health is depleted. This way, the animation plays out correctly and stops any further player actions until the game resets or respawns.




To facilitate player movement, we implemented a dedicated script that governs the player's control mechanics. This script not only handles the character's movement but also manages various actions such as jumping, crouching, and attacking enemies.

The movement code is designed to respond to player inputs from the keyboard or game controller, enabling smooth and intuitive navigation across different terrains in the game. We incorporated physics-based calculations to ensure realistic movement dynamics, allowing the player to accelerate, decelerate, and change directions fluidly.

For instance, the script uses Unity's Rigidbody component to apply forces for jumping and falling, creating a more immersive experience. In the Rigidbody settings, we froze the Z-axis to prevent the player from rotating, ensuring that the movement remains strictly two-dimensional, which is essential for a 2D game. Furthermore, to enhance realism, we utilized the dynamic command in the Rigidbody, allowing the player to respond naturally to forces applied during movement, such as gravity and collisions. This approach contributes to a smoother and more engaging gameplay experience.



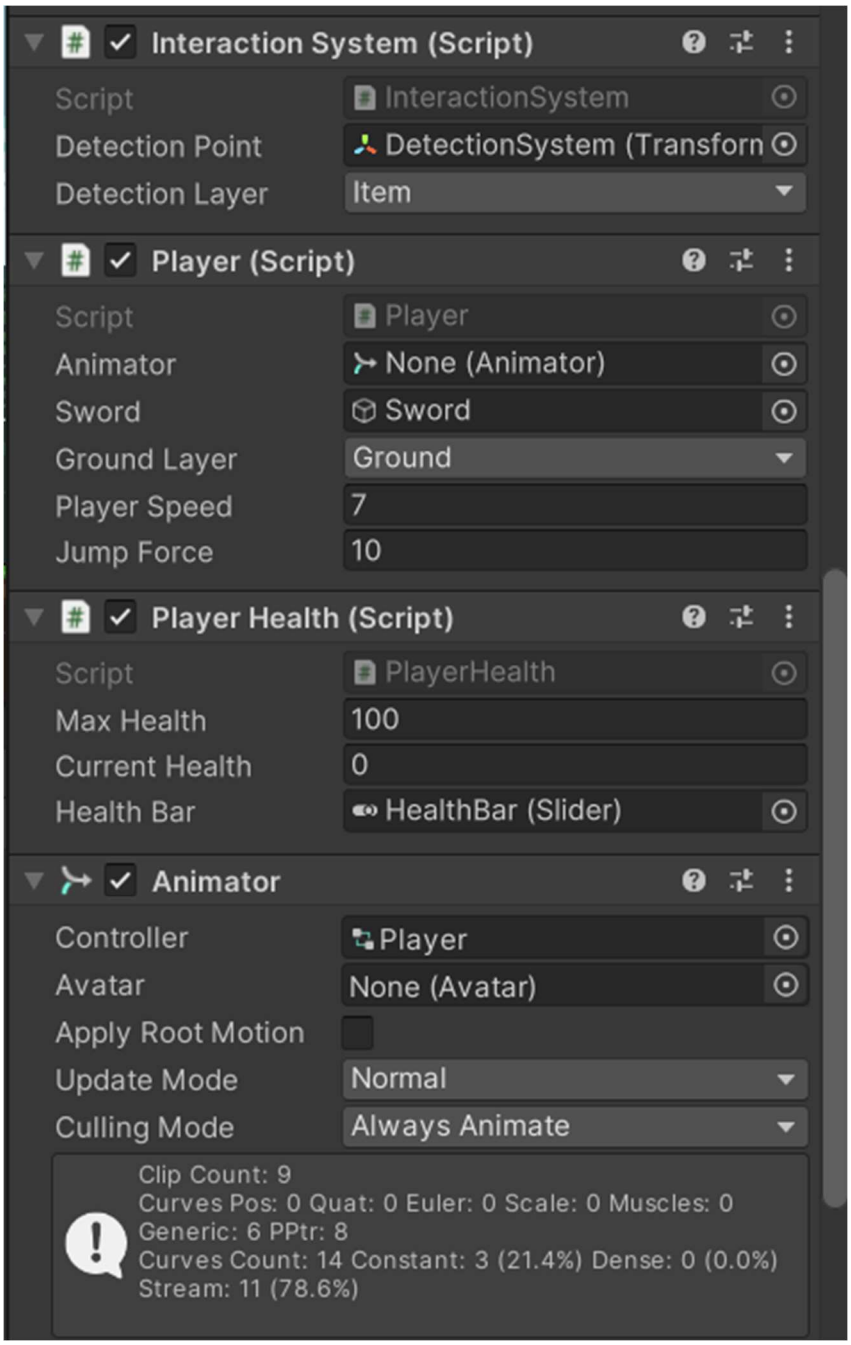

In addition to movement, the script also controls the player's attack mechanics. Depending on the input commands, the player can execute different attack animations—such as slashing with a sword, throwing kunai, or using fireballs. Each attack reduces the health of enemies based on the type of weapon used and the player's current stats. We implemented a hit detection system that checks for collisions between the player's attacks and the enemies, ensuring that the combat feels responsive and satisfying.

To track the player’s health, we developed a separate health management script. This script monitors the player's health status and updates it whenever the player takes damage from enemies or environmental hazards. The health script also triggers the appropriate animations when the player is hurt and controls the transition to the death animation when health reaches zero. Furthermore, we included a health pickup system where the player can regain health by collecting items scattered throughout the levels, adding another layer of strategy to the gameplay.

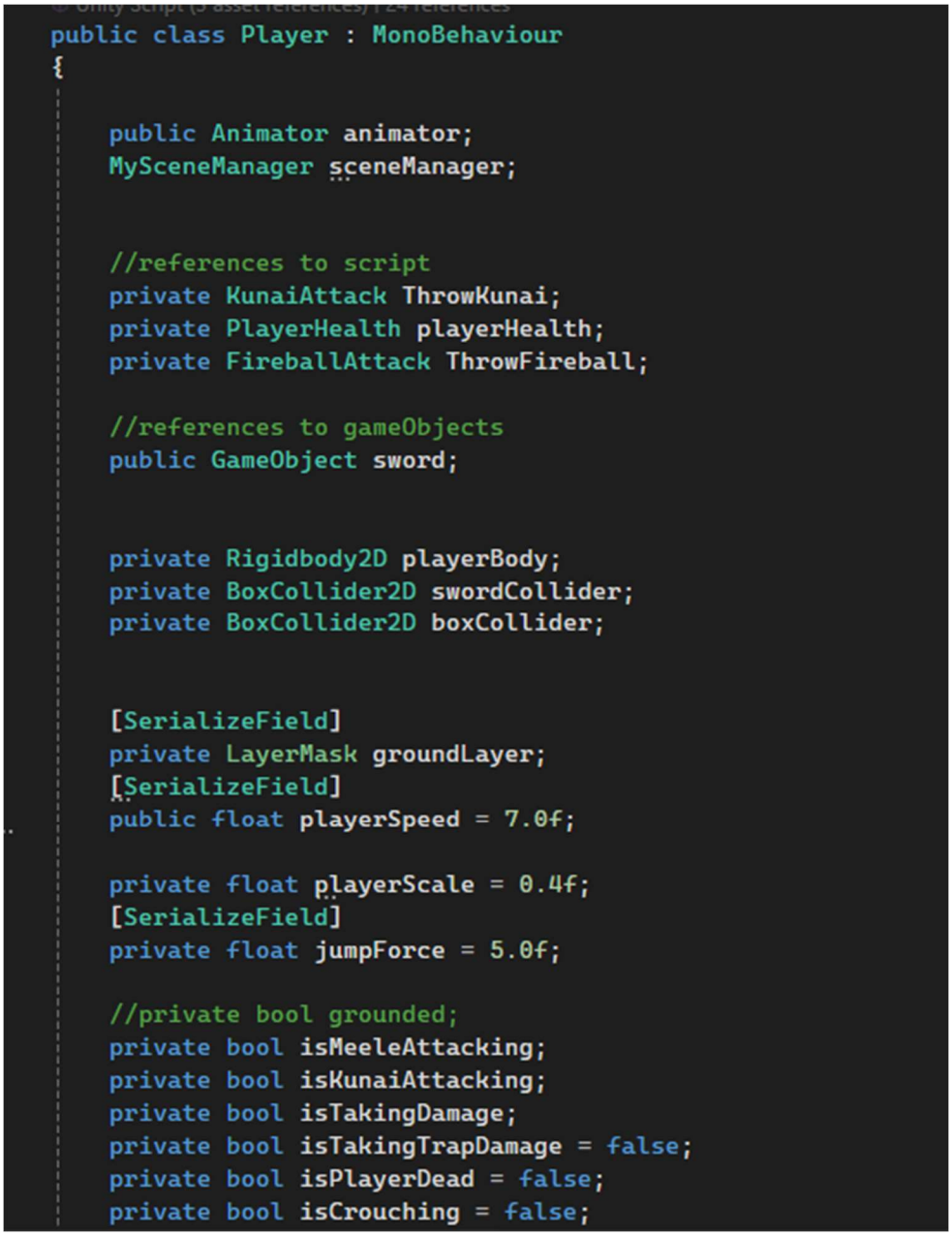
```
    // Call this method to apply damage to the player
    7 references
    public void TakeDamage(float damage)
    {
        currentHealth -= damage;
        currentHealth = Mathf.Clamp(currentHealth, 0f, maxHealth); // Ensures health never goes below 0
        if (currentHealth > 0)
        {
        }

        UpdateHealthBar();

    }

    // Call this method to heal the player
    1 reference
    public void Heal(float healAmount)
    {
        currentHealth += healAmount;
        currentHealth = Mathf.Clamp(currentHealth, 0f, maxHealth);

        UpdateHealthBar();
    }




    // Update the health bar to reflect the current health
    2 references
    void UpdateHealthBar()
    {


        healthBar.value = currentHealth;
    }
}
```

Together, these scripts create a cohesive framework for player interactions, ensuring that movement and combat mechanics work seamlessly together. The combination of responsive controls, engaging animations, and robust health management enhances the overall gameplay experience, keeping players engaged and challenged throughout their journey.

We have also aimed to keep most of the features in a serializable format, allowing us to make adjustments directly within the Unity interface without needing to repeatedly delve into the code. This approach not only streamlines our workflow but also adheres to sound design principles. By utilizing Unity's serialization, we can easily modify parameters such as player speed, enemy health, enhancing flexibility and efficiency during development. This practice also helped in better collaboration among ourselves and allowed for quicker iterations.




For the enemies in our game, we began by creating all their animations, ensuring they have smooth and engaging movements. After designing the animations, we used Unity's Animator to set up transitions between different states, utilizing triggers and boolean values to control how and when these transitions occur. We also developed a health script for the enemies, which keeps track of their health points and determines when they are defeated.



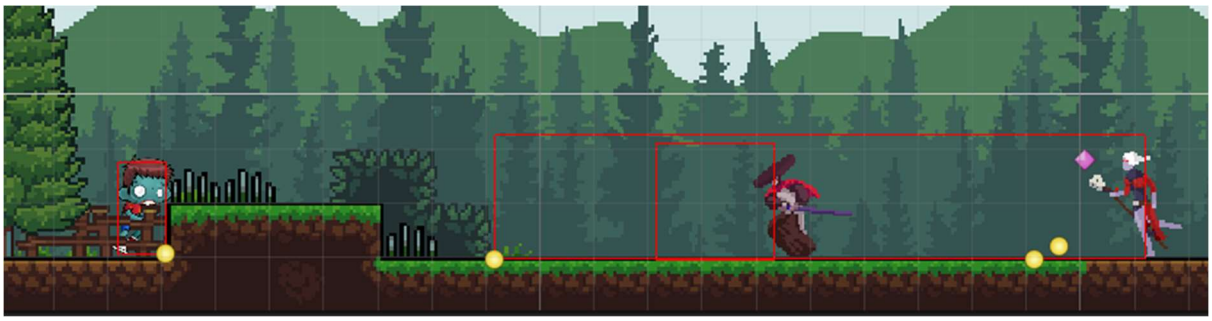
```
public class Player : MonoBehaviour
{

    public Animator animator;
    MySceneManager sceneManager;


    //references to script
    private KunaiAttack ThrowKunai;
    private PlayerHealth playerHealth;
    private FireballAttack ThrowFireball;

    //references to gameObjects
    public GameObject sword;


    private Rigidbody2D playerBody;
    private BoxCollider2D swordCollider;
    private BoxCollider2D boxCollider;


    [SerializeField]
    private LayerMask groundLayer;
    [SerializeField]
    public float playerSpeed = 7.0f;

    private float playerScale = 0.4f;
    [SerializeField]
    private float jumpForce = 5.0f;

    //private bool grounded;
    private bool isMeeleAttacking;
    private bool isKunaiAttacking;
    private bool isTakingDamage;
    private bool isTakingTrapDamage = false;
    private bool isPlayerDead = false;
    private bool isCrouching = false;
```

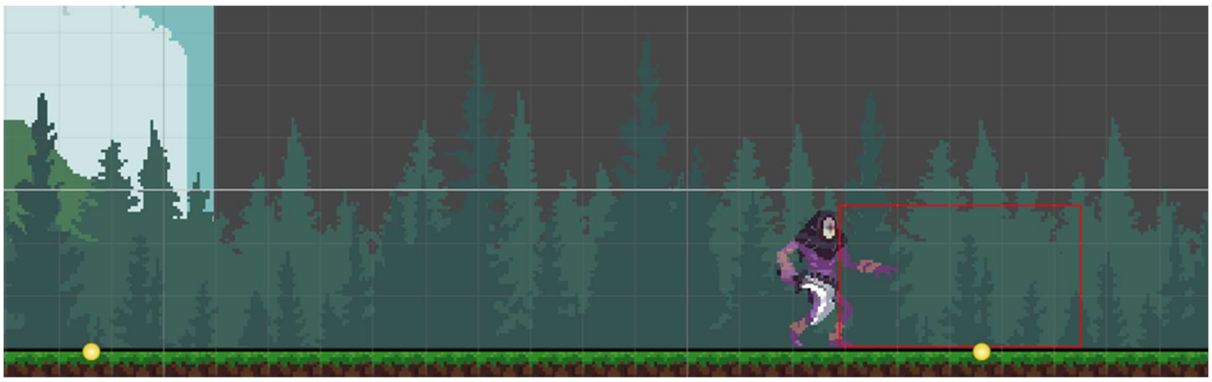
We implemented two types of enemy scripts: Patrolling Chase and Patrolling.

**Patrolling Chase**: This script allows enemies to chase the player when they come within a certain distance. For example, if a player approaches a boss or a zombie enemy, that enemy will start pursuing the player, making the gameplay more challenging.

**Patrolling**: This script is used for enemies like samurai and mages. These enemies move back and forth between two points in a designated area. They will attack the player if the player enters their attack range. However, if the player sneaks up from behind, these enemies won't attack, encouraging players to use strategy to approach them.

Overall, the Patrolling Chase script is mainly assigned to  enemies, like bosses and zombies, while the simpler Patrolling script is used for basic enemies like samurai and mages. This differentiation helps us create various gameplay dynamics, making encounters with different enemies feel unique and engaging.





In developing our game, we paid special attention to creating diverse backgrounds and scenes that enhance the gameplay experience. Each scene is meticulously crafted to immerse players in a vibrant and dynamic world, ranging from serene natural landscapes to the fiery depths of hell.

For the **Nature Scene**, we designed a lush environment filled with vibrant greenery and colorful flora. The background features tall trees swaying in the wind and distant mountains, creating a sense of depth and adventure. As players progress through this scene, they encounter various obstacles

Transitioning from the beauty of nature, players enter the **Cavern Scene**. This setting is darker and more mysterious, with glowing crystals illuminating the paths and eerie sounds echoing throughout. The design includes secret passages hidden behind rocks encouraging exploration and rewarding players with power-ups.

In each scene, we integrated smooth transitions and dynamic elements to keep the gameplay engaging. Background animations, such as moving clouds or flickering flames, add to the overall immersion. Our level design not only focuses on aesthetics but also incorporates gameplay mechanics, ensuring that players are continually challenged while exploring these diverse environments.

Overall, the backgrounds and scenes in our game serve as more than just visual elements; they create an engaging narrative and enhance the player experience by providing an ever-changing and interactive world to explore.


## Game Play Features:

**Movement Controls:**

- **Arrow Keys:** The primary movement of the player character is controlled using the arrow keys.
  - **Left Arrow Key:** Pressing this key moves the ninja to the left, allowing players to retreat or navigate back through the level.
  - **Right Arrow Key:** This key moves the ninja to the right, enabling players to advance through the level and engage enemies.
  - **Up Arrow Key:** Players can use this key to jump, allowing the ninja to leap over obstacles, reach higher platforms, or evade enemy attacks.

- **Down Arrow Key:** This key is used for crouching, enabling the ninja to avoid enemy attacks or navigate through tight spaces.

**Attack Controls:**

- **Q Key (Kunai Attack):** Pressing the Q key triggers the kunai attack.
- **F Key (Melee Attack):** The F key activates a melee attack. This is a close-range attack where the ninja swings a weapon, dealing damage to any enemies in proximity.
- **Z Key (Fireball Attack):** Pressing the Z key allows the player to unleash a fireball attack. This special move sends a powerful fireball towards enemies, causing significant damage. The fireball can travel a longer distance than the kunai and can also pass through the enemies.
- **Animation:** Each attack has a unique animation that provides visual cues to the player, indicating that the action has been successfully executed.
- **Sound Effects:** We incorporated distinct sound effects for each action, such as the sound of the kunai slicing through the air, the impact of the melee attack, and the explosive sound of the fireball. This auditory feedback helps to reinforce the impact of each action and adds excitement to the gameplay.

In addition to the control scheme we designed, we have incorporated various collectibles throughout the levels to enhance gameplay and provide players with additional abilities.

**Collectibles:**

- **Fireball Power-Up:** Throughout the levels, players can find fireball power-ups. When the player collects a fireball, it adds to their stock of fireball attacks, allowing them to use this powerful ability more frequently.

- **Apple:** Players can also find apples scattered throughout the game levels. Collecting an apple restores a portion of the player's health, allowing them to recover after taking damage from enemies.

# Testing

During the testing phase of our game, we encountered several issues that needed to be resolved to ensure a smooth gaming experience.

First, we noticed that the enemies could not be flipped correctly. This meant that when an enemy was supposed to face the player, it wouldn't turn around properly. We realized that this was essential for making the game feel more realistic and engaging, so we worked on fixing this issue. After some adjustments, we were able to implement a solution that allowed enemies to flip and face the direction of the player, enhancing their behavior and making them more challenging to defeat.

Next, we faced another problem where the enemies were not chasing the player effectively. Initially, the enemy AI didn't react properly when the player came within range. This made it easy for players to avoid encounters, which wasn't the intended gameplay experience. We revised the enemy scripts to improve their detection and chasing mechanics. By adjusting the parameters and refining the logic, we ensured that enemies would actively pursue the player, adding to the excitement and challenge of the game.

Additionally, we discovered an error in the user interface (UI) related to the score not carrying over correctly between levels. The score is an important aspect of the game, as it reflects the player's performance. We found that when transitioning from one stage to another, the score was resetting or not updating properly, which was frustrating for players. To address this, we reviewed the code responsible for managing the score and made sure it was designed to persist across levels. After implementing these changes, we tested the transitions again, and the score now carries over smoothly from one stage to the next.

By identifying and resolving these issues during testing, we significantly improved the gameplay experience. Each fix contributed to making the game more enjoyable and ensuring that players have a fair and challenging experience as they navigate through the levels.

# Conclusion

In conclusion, the development of our 2D Unity game, Ninja-Run, inspired by Metal Slug, has been an enriching and challenging experience. We successfully implemented a range of features that enhance gameplay, including responsive player controls, engaging animations, and dynamic enemy behaviors. Through rigorous testing and problem-solving, we addressed critical issues such as enemy flipping, chasing mechanics, and score persistence across levels, all of which contributed to a smoother and more enjoyable gaming experience.

The diverse environments we created, from lush natural landscapes to the depths of hell, not only add visual appeal but also introduce various challenges, such as traps and secret passages. Our design choices and coding practices emphasized a balance between functionality and user experience, allowing for easy adjustments within Unity.

Overall, the project has strengthened our skills in game design, coding, and problem-solving. We believe that Ninja-Run offers an engaging and enjoyable experience for players, with the potential for further enhancements and expansions in the future. As we continue to refine our game, we look forward to gathering player feedback and making improvements that will enhance the overall enjoyment of Ninja-Run.