Virtual Worlds Assignment 2

Class : Virtual Worlds

Submission Date : 2024/05/10

Student ID : 2022742021

Affiliation : Kwangwoon University

Full name : Juho Kim

Major : Software Engineering

Abstract

This report provides a comprehensive walkthrough of the development process for the Virtual Worlds Programming Assignment 2 implemented in Unity 3D. First, the fulfillment of each requirement is discussed with descriptive code snippets. Then the report concludes by addressing the challenges encountered during development and outlining the limitations of the current code, which presents opportunities for future improvement.

The game was developed mainly applying the knowledge of vectors, transformations and dynamic object spawning. For more information, please visit my GitHub repository.

Requirements Fulfillment

Following table shows the assignment requirements from the slides along with completion status.

Easy 1	Vertical Player Movement	O
Easy 2	Animals increasing proportionally	O
Easy 3	The food no longer moves	O
Easy 4	Camera switcher	O
Medium	Animal Hunger Bar	O
Hard	Move to food	O
Expert 1	Move freely	O
Expert 2	Collision avoidance	△

Easy 1 (Vertical Player Movement)

Step1 : Initializing variables

```
private float verticalInput;
private float zRange = 10.0f;
```

To begin with, the following 2 variables at **PlayerController.cs** were made private to prevent it from being contaminated by the user :

- **verticalInput** : receive input of vertical position from user
- **zRange** : Set bound for vertical movement

Step2 : Translating player vertically with user input

```
// Enable vertical movement
verticalInput = Input.GetAxis("Vertical");
transform.Translate(Vector3.forward * verticalInput * Time.deltaTime * speed);
```

The value of total time user spent pressing the arrow key multiplied by the predefined speed was passed in the Translate function to translate the player vertically when user presses the side arrow keys.

Step3 : Setting vertical motion range of player

```
// Set bound for moving vertically
if (transform.position.z < -zRange)
{
    transform.position = new Vector3(transform.position.x, transform.position.y, -zRange);
}
if (transform.position.z > zRange + 20)
{
    transform.position = new Vector3(transform.position.x, transform.position.y, zRange+20);
}
```

Then the vertical motion range was set to prevent user from moving out of the bound.

Easy 2 (Animals increasing proportionally to current population)

Following functionalities were implemented in **SpawnManger.cs** for increasing animals proportional to the current existing animal population on the ground:

Step 1: Initializing variables

```
public GameObject[] creaturePrefabs; // Prefab for storing all creatures
public int creatureIndex; // Variable for saving creature index

// Spawn range of creatures
private float spawnRangeX = 10.0f;
private float spawnRangeZ = 10.0f;

// Spawn delay and interval
private float startDelay = 2;
private float spawnInterval = 1.5f;

// Maximum number of creatures on the ground
private int maxCreatures = 15;
private List<GameObject> creatureList = new List<GameObject>();
```

- **creaturePrefabs** : References the 3 types of animals to be spawned
- **spawnRangeX** : X range for spawning animals
- **spawnRangeZ** : Z range for spawning animals
- **startDelay** : Initial delay before starting to spawn animals
- **spawnInterval** : Delay for spawning new animals
- **maxCreatures** : Maximum number of creatures that can exist on the ground
- **creatureList** : List for tracking all animals on the ground

Step 2: Limiting spawn rate to prevent overpopulation

```
int creaturesToSpawn = Mathf.FloorToInt(1 + (maxCreatures - CurrentCreatureCount()) / 10f);
```

To prevent overpopulation, the number of creatures spawned is decreased with more animals existing on the ground. As the value returned from **CurrentCreatureCount** increases, the value saved in **creaturesToSpawn** is decreased.

Step 3: Spawning animals randomly

```
// Spawn corresponding creatures on the screen
for (int i = 0; i < creaturesToSpawn; i++)
{
    // Randomize spawn positions
    Vector3 spawnPos = new Vector3(Random.Range(-spawnRangeX, spawnRangeX), 0, Random.Range(-spawnRangeZ, spawnRangeZ));

    // Randomize direction at which the spawned creature is facing
    Quaternion spawnRotation = Quaternion.Euler(0, Random.Range(0, 360), 0);

    // Create random creature on to the ground
    creatureIndex = Random.Range(0, creaturePrefabs.Length);
    GameObject creature = Instantiate(creaturePrefabs[creatureIndex], spawnPos, spawnRotation);

    // Add spawned creature to the list
    creatureList.Add(creature);
}
```

spawnPos and **spawnRotation** are each used to randomly set the spawn position and rotation of the animal. Then random **creatureIndex** is used to spawn random animals and added to **creatureList** to track all the animals currently on the ground.

Step 4 : Untracking destroyed animal and updating spawn interval

```
void Update()
{
    // Remove any creatures in the list that's been destroyed
    creatureList.RemoveAll(item => item == null);

    // Update spawning interval
    AdjustSpawnInterval();
}
```

To untrack destroyed animals, NULL animal objects are removed from **creatureList**. Also, **AdjustSpawnInterval** is called to update spawn interval in proportion to current animal population.

```
void AdjustSpawnInterval()
{
    float newSpawnInterval = 1.5f - ((maxCreatures - CurrentCreatureCount()) * 0.05f);

    newSpawnInterval = Mathf.Max(newSpawnInterval, 1.5f);

    if (Mathf.Abs(newSpawnInterval - spawnInterval) > Mathf.Epsilon)
    {
        spawnInterval = newSpawnInterval;
        CancelInvoke("SpawnRandomCreature");
        InvokeRepeating("SpawnRandomCreature", 0, spawnInterval);
    }
}
```

As the population of animal increases, the **newSpawninterval** is decreased, hence effectively spawning animals more frequently with increased animal population.

Easy 3 (The food no longer moves)

To prevent the food from moving, Move Forward script has been disabled, but not removed from the bone object. The reason for doing so is because we might want to enable it in the future.

```
// Launch bone slgihtly ahead of the player's positio
Instantiate(
        projectilePrefab,
        transform.position + new Vector3(0f,0f,4),
        projectilePrefab.transform.rotation
    );
```

Also, inside **PlayerController.cs** the offset vector has been added to the current position of the player to spawn the bone slightly ahead of the player's current position for better user experience.

Easy 4 (Camera switcher)

Step 1 : Initializing variable for setting perspective

```
// 1=First-person view, 3=Third-person view
private int toggleView = 3;
```

A private variable toggleView is initialized to 3 for setting default view to third person perspective.

Step 2 : Changing toggleView with user input

```
// Toggle perspective when left or right shift key is pressed
if (Input.GetKeyDown(KeyCode.LeftShift) || Input.GetKeyDown(KeyCode.RightShift))
    toggleView = 4 - toggleView;
```

toggleView alternatives between 1 and 3 whenever user presses the left or right shift key.

For example, when user presses shift on third perspective, the new toggleView is 4-3 = 1.

Step 3: Setting user perspective

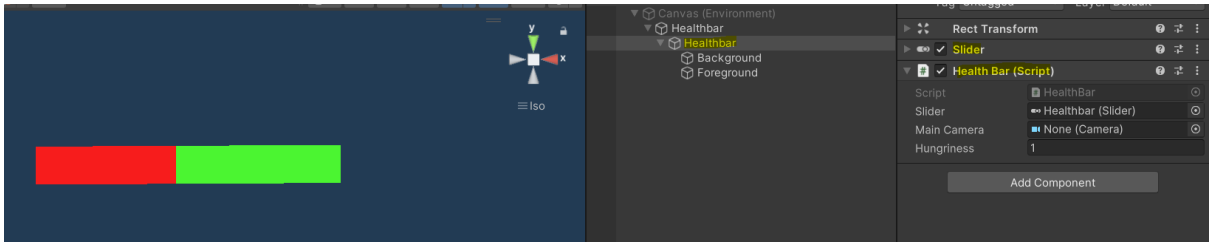
```
// Set to first-person view
if (toggleView == 1)
{
    Camera.main.orthographic = false;
    transform.rotation = Quaternion.Euler(16.6f, 0f, 0f);
    transform.position = player.transform.position + new Vector3(0f, 3f, 1f);
}

// Set to orthogonal view (Third-person view)
else if (toggleView == 3)
{
    Camera.main.orthographic = true;
    transform.rotation = Quaternion.Euler(90f, 0f, 0f);
    transform.position = new Vector3(0f, 25f, 10f);
}
```

The user perspective is set based on the value saved in toggleView, with the projection of each third and first perspective set to orthographic and perspective respectively.

Additionally, the rotation and position of the camera has been adjusted for each perspective.

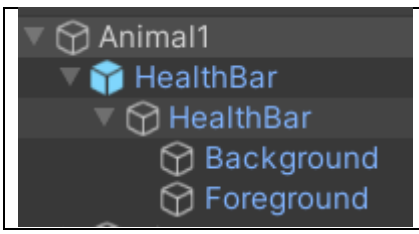
Medium (Animal Hunger Bar)

Step 1 : Creating the health bar



Health bar object was created with **Foreground** and **Background** each set as green and red color, and a slider was added to control the width of green color to give the illusion of functioning like a health bar. Finally, **HealthBar.cs** script was added to make the healthbar interactive.

Step 2: Linking healthbar with each animal

	To follow each animals, **Healthbar** has been added under each animal objects.

After setting up the heathbar object with all animals, the following were implemented in **Healthbar.cs** :

Step3 : Initializing variables

```
public Slider slider;
public Camera mainCamera;
public float hungriness = 1.0f;
```

- **Slider** : For controlling hunger bar
- **mainCamera** : Track perspective of the camera to change hunger bar rotation in the future
- **hungriness** : Rate at which hunger bar is decreasing

Step 4 : Initializing hunger bar and camera object

```
void Start()
{
    // Make health bar full by default
    slider.value = 100;
    mainCamera = Camera.main; // Declaring main camera
}
```

Hunger bar is set to max, 100 and maincamera object is initialized.

Step 5 : Destroying animal when hunger bar is zero

```
if (slider.value <= 0)
{
    Transform topParent = transform.root;
    Destroy(topParent.gameObject);
}
```

When hunger bar is zero, the parent object, its corresponding animal is accessed and destroyed.

Step 6: Decreasing hunger bar over time

```
slider.value -= 10 * Time.deltaTime * hungriness;
```

Hunger bar value decreases with change in time and hungriness.

Step 7 : Displaying hunger bar view from third perspective

```
slider.transform.LookAt(slider.transform.position +
    mainCamera.transform.rotation * Vector3.forward,
    mainCamera.transform.rotation * Vector3.up);
```

The orientation of the hunger bar changes depending on whether the game is viewed from a first-person or third-person perspective, facing either the Z-axis or the Y-axis.

Step 8 : Gaining health upon eating the bone

```
public void GainHealth(float hp)
{
    slider.value += hp;
}
```

GainHealth was defined in **HealthBar.cs** to increase health upon eating the bone.

```
public float incHealth = 3.0f;

void OnTriggerEnter(Collider other)
{
    // Increase Animal health upon eating the bone
    if (other.transform.tag == "Animal")
    {

        HealthBar healthBar = other.GetComponentInChildren<HealthBar>();
        healthBar.GainHealth(incHealth);
        Destroy(this.gameObject);
    }
}
```

Finally, **HealthPot.cs** was attached to the bone to manage the child object of the collided object (the hunger bar), utilizing the **GainHealth** method to augment the hunger bar's level. Additionally, the bone is removed from the scene when consumed by an animal.

Hard (Move to food)

Following code was implemented in **MoveForward.cs :**

Step 1 : Looking for closest bone

```
GameObject bone = FindClosestBone();
```

The animal looks for any bones nearby.

Step 2 : Calculating distance from the bone

```
if (bone != null)
{
    Vector3 direction = bone.transform.position - transform.position;
    float distance = direction.magnitude;
```

The distance of the animal from the bone is calculated.

Step 3 : Animal moving towards the bone

```
if (distance <= detectionRange)
{
    Quaternion targetRotation = Quaternion.LookRotation(direction);
    transform.rotation = Quaternion.Slerp(transform.rotation, targetRotation, rotationSpeed * Time.deltaTime);
    transform.position = Vector3.MoveTowards(transform.position, bone.transform.position, speed * Time.deltaTime);
}
```

If the animal detects the bone within the **detectionRange**, it moves toward the bone.

Expert 1 (Move freely)

Following code was implemented in **MoveForward.cs :**

```
void Wander()
{
    // 1. Wonder when time is closest
    if (Time.time >= nextWanderTime)
    {
        float wanderAngle = Random.Range(-wanderRange, wanderRange);
        targetRotation = Quaternion.Euler(0, wanderAngle, 0) * transform.rotation;
        nextWanderTime = Time.time + wanderInterval;
    }

    transform.rotation = Quaternion.Slerp(transform.rotation, targetRotation, rotationSpeed * Time.deltaTime);
    transform.Translate(Vector3.forward * Time.deltaTime * speed);
}
```

Wander rotates the animal for every **nextWanderTime** to make the animal freely.

Expert 2 (Collision avoidance)

Following code was implemented in **MoveForward.cs** :

```
void OnTriggerEnter(Collider other)
{
    // 3. If the moving animal collided with another animal, pause
    if (other.CompareTag("Animal") && !isPaused)
    {
        Quaternion targetRotation = Quaternion.Euler(0, 45, 0);
        StartCoroutine(PauseMovement(0.5f)); // Pause movement for 0.5 seconds
        transform.rotation = Quaternion.Slerp(transform.rotation, targetRotation, rotationSpeed * Time.deltaTime);
    }
}
```

Upon colliding with another animal, the animal pauses for 0.5 seconds and rotates to face other direction, effectively avoiding collisions.

Conclusion

□ Key challenges tackled

Some of the hardest parts were:

1. Creating health bar object

Initially, following the [tutorial from Kapp] the hunger bar was made using one slider, however this caused the health bar to appear to be partially filled when hunger value is zero.

To prevent partial filling of the hunger bar, two image objects were added under a slider.

2. Making health increase

I was not sure how to access the hungerbar from another script because it was a child object of an animal object. So it took some time to read the Unity documentation and understand **GetComponentInChildren.**

3. Displaying health bar at third person perspective

	Initially the hunger bar did not appear on third person perspective as shown in the picture. So I attempted to create a 3d hunger bar, but the UI for the hunger bar fell short of expectations in terms of quality and effectiveness. Therefore, in order to display hunger bar at third person perspective, code has been added to **HealthBar.cs** to rotate hunger bar towards Z axis on third person perspective

Limitations

Collision Avoidance may not occur when animal is spawned on another existing animal.

Future improvements

Prevent animal from spawning at location where there is another existing animal.

References

- Getting variable from another script in unity
- Difference between Update(), LateUpdate(), FixedUpdate()
- HungerBar in Unity2020
- How to make a HEALTHY BAR in unity
- Gain Health on Collision
- Tags – Unity Official Tutorials
- Experimenting with Obstacle Avoidance in Unity3D