ΓΡΑΦΙΚΑ ΥΠΟΛΟΓΙΣΤΩΝ ΚΑΙ ΣΥΣΤΗΜΑΤΑ ΑΛΛΗΛΕΠΙΔΡΑΣΗΣ

Προγραμματιστική Άσκηση 2.

Team

Γκόβαρης Χρήστος-Γρηγόριος
Σπανού Μαρία

Χρονοδιάγραμμα Εργαστηριακής Άσκησης 2

Έκδοση	Ημερομηνία	Progress
1.0	2/12/2024	Ανακοίνωση εργαστηριακής άσκησης 2
2.0	8/12/2024	Υλοποίηση του ερωτήματος (i)
2.1	8/12/2024	Υλοποίηση του ερωτήματος (ii) και του (β)
2.2	8/12/2024	Υλοποίηση του ερωτήματος (iii) και των (α) και (γ)
2.3	9/12/2024	Υλοποίηση του ερωτήματος (iv) και του (δ)
2.4	9/12/2024	Υλοποίηση του ερωτήματος (v)
3.0	10/12/2024	Υλοποίηση Report εργαστηριακής άσκησης

Ανάλυση Χαρακτηριστικών

Η εργασία υλοποιήθηκε σε ένα μηχάνημα (Lenovo Ideapad 3), με τα εξής χαρακτηριστικά πυρήνα:

- AMD Ryzen 7 (7730U)
- 8 CPU cores
- 16 Threads
- Boost Clock up to 4.5GHz
- Base Clock 2.0GHz
- CPU Socket FP6
- Intergraded Graphics (Radeon Graphics)

Επιπλέον, η **εργασία 2** υλοποιήθηκε σε **Windows 11 Home**, χρησιμοποιώντας το **Visual Studio Code** και το **UnityHub 3.10.0**.

Λειτουργία Ομάδας / Ανάλυση Ερωτημάτων

Η συνεργασία της ομάδας ήταν υποδειγματική, εξασφαλίζοντας την επιτυχή υλοποίηση της Άσκησης 2. Οι προκλήσεις που αντιμετωπίσαμε περιλάμβαναν την ευθυγράμμιση των θησαυρών με τα μη κεντραρισμένα τοιχώματα του λαβυρίνθου, την αρχική τοποθέτηση αντικειμένων στο επίπεδο και την διασφάλιση ότι ο χαρακτήρας **Bob** δεν μπορεί να διεισδύσει στα τοιχώματα του λαβυρίνθου. Επιπλέον, στα ερωτήματα bonus, δυσκολευτήκαμε με την εμφάνιση του κειμένου **"Game Over"** και του **"Score"**.

2.0 Υλοποίηση του ερωτήματος (i)

Στο πρώτο ερώτημα ζητήθηκε η δημιουργία μιας **Unity 3D** εφαρμογής με ανάλυση **1024x768** και τίτλο «**Project 2 – Treasure Bob**». Ο λαβύρινθος έπρεπε να τοποθετηθεί σε ένα επίπεδο (**plane**) με διαστάσεις **100x100** στο **xz** επίπεδο, ενώ τα τοιχώματα σχηματίζονται από κύβους με διαστάσεις **10x10x10**. Η υφή του δαπέδου έπρεπε να είναι η εικόνα **floor.jpg**, ενώ τα τοιχώματα του λαβύρινθου έχουν μπλε χρώμα. Πιο συγκεκριμένα, έγιναν οι εξής αλλαγές:

Στην εφαρμογή **UnityHub**, ρυθμίστηκε η ανάλυση και ο τίτλος της εφαρμογής. Μέσα από τα **File > Build Settings > Player Settings**, ορίστηκε ο τίτλος ως «**Project 2 – Treasure Bob**». Στο ίδιο μενού, στο τμήμα **Resolution and Presentation**, η ανάλυση ορίστηκε σε **1024x768**, και η επιλογή **Fullscreen Window** ενεργοποιήθηκε για παρακολούθηση του παραθύρου σε πλήρη οθόνη. Δημιουργήθηκε το δάπεδο, μέσω **GameObject > 3D Object > Plane**, προσθέσαμε ένα επίπεδο (**plane**) στη σκηνή. Στον **Inspector**, προσαρμόσαμε το μέγεθος του **Plane** στις διαστάσεις **100x100** (scale: X=100, Y=1, Z=100). Εφαρμόσαμε την υφή **floor.jpg**: Δημιουργήσαμε ένα **Material**. Στο πεδίο **Albedo**, προσθέσαμε την εικόνα **floor.jpg**. Αντιστοιχίσαμε το **Material** στο **Plane** μέσω **drag-and-drop**. Δημιουργήθηκαν τα τοιχώματα (**MazeN**), χρησιμοποιώντας **GameObject > 3D Object > Cube**, προσθέσαμε κύβους (**cubes**) που λειτουργούν ως τοιχώματα. Στον **Inspector**, προσαρμόσαμε τις διαστάσεις κάθε κύβου στις τιμές **10x10x10** (**scale: X=10, Y=10, Z=10**). Τοποθετήσαμε τους κύβους στις σωστές θέσεις σύμφωνα με την εκφώνηση και την εικόνα του λαβύρινθου και μετακινήσαμε κάθε κύβο μέσω του **Transform > Position** για να σχηματίσουμε τη δομή του λαβύρινθου. Για να δώσουμε μπλε χρώμα στα τοιχώματα δημιουργήσαμε ένα νέο **Material**. Στο πεδίο **Albedo**, επιλέξαμε μπλε χρώμα. Αντιστοιχίσαμε το Material στους κύβους μέσω **drag-and-drop**. Ως **layer**, δημιουργήσαμε και προσθέσαμε το **Walls**. Τοποθετήσαμε τον λαβύρινθο μετακινώντας ολόκληρη τη δομή του έτσι ώστε το κέντρο του να βρίσκεται στο σημείο **(0,0,5)**. Αυτό έγινε με τη μετακίνηση ενός **Parent GameObject** που περιείχε όλους τους κύβους.

2.1 Υλοποίηση του ερωτήματος (ii) και του (β)

Για το ερώτημα (ii):

Στο δεύτερο ερώτημα, ζητήθηκε η δημιουργία του χαρακτήρα **Treasure Bob**, ο οποίος είναι μια σφαίρα με διάμετρο 7 και υφή **bob.jpg**. Ο Bob ξεκινά από την είσοδο του λαβύρινθου και πλοηγείται μέσα σε αυτό με τα πλήκτρα **<i>, <k>, <j>, <l>**. Ο **Bob** κινείται με σταθερό βήμα στους άξονες **x** και **z**, δεν επηρεάζεται από τη βαρύτητα, και δεν μπορεί να περάσει μέσα από τοίχους ή να βγει έξω από τον λαβύρινθο. Πιο συγκεκριμένα, έγιναν οι εξής αλλαγές:

Στην εφαρμογή **UnityHub**, δημιουργήθηκε το Treasure Bob. Προστέθηκε μια **Sphere** στη σκηνή μέσω του **GameObject > 3D Object > Sphere**. Η διάμετρος της σφαίρας ρυθμίστηκε σε 7 μέσω του **Inspector > Transform > Scale**, με τιμές X=7, Y=7, Z=7.Η υφή **bob.jpg** προστέθηκε με τη δημιουργία ενός νέου **Material**. Επίσης, για την αρχική θέση και κίνηση του χαρακτήρα **Bob**, ορίστηκε στην είσοδο του λαβύρινθου (π.χ., (-45, 3.5, -45)), ώστε να είναι πάνω στο δάπεδο. Προστέθηκε ένα **Rigidbody** στη σφαίρα και ενεργοποιήθηκε η επιλογή **Is Kinematic** για να διασφαλιστεί ότι δεν επηρεάζεται από τη βαρύτητα. Ένα **Box Collider** προστέθηκε στους τοίχους και στη σφαίρα, ώστε να μην μπορεί ο **Bob** να περάσει μέσα από αυτούς. Στο πεδίο **Tag** προστέθηκε το **Player**.

Υλοποιήθηκε ο παρακάτω κώδικας, στο αρχείο **moveBob.cs**

```
float moveX = 0, moveZ = 0;


if (Input.GetKey(KeyCode.I)) moveZ = 3;
if (Input.GetKey(KeyCode.K)) moveZ = -3;
if (Input.GetKey(KeyCode.J)) moveX = -3;
if (Input.GetKey(KeyCode.L)) moveX = 3;


Vector3 movement = new Vector3(moveX, 0, moveZ) * speed * Time.deltaTime;
Vector3 newPosition = transform.position + movement;
```

σύμφωνα με τον οποίο, τα πλήκτρα **<i> και <k>** ελέγχουν την κίνηση στον άξονα **Z** (προς τα μπροστά/πίσω), τα πλήκτρα **<j> και <l>** ελέγχουν την κίνηση στον άξονα **X** (αριστερά/δεξιά). Η μεταβλητή movement καθορίζει την κατεύθυνση και το μέγεθος της μετατόπισης, πολλαπλασιασμένη με την ταχύτητα (**speed**) και τον χρόνο (**Time.deltaTime**) για ομαλή κίνηση.

Υλοποιήθηκε ο παρακάτω κώδικας, στο αρχείο **moveBob.cs**

```
Vector3 checkBoxSize = new Vector3(3.5f, 3.5f, 3.5f);


if (!Physics.CheckBox(newPosition, checkBoxSize, Quaternion.identity, wallLayer)) {
   newPosition.x = Mathf.Clamp(newPosition.x, minX, maxX);
   newPosition.z = Mathf.Clamp(newPosition.z, minZ, maxZ);
   transform.position = newPosition;
}
```

σύμφωνα με τον οποίο, το **Physics.CheckBox** ελέγχει αν η νέα θέση του **Bob** επικαλύπτεται με κάποιον τοίχο. Αν η θέση είναι έγκυρη, εφαρμόζεται ο περιορισμός μέσω **Mathf.Clamp** ώστε να μην βγει ο **Bob** εκτός των ορίων του λαβύρινθου. Τέλος, η νέα θέση αποδίδεται στον **Bob**.

*Το **script moveBob.cs** έχει προστεθεί ως **component** στο **3D Object** του **Bob**. Έχουν προστεθεί τα **speed = 5, Min X = -45, Max X = 45, Min Z = -45, max Z = 45** και **Wall Layer = Walls**.*

Για το ερώτημα (β):

Στο bonus ερώτημα (β), ζητήθηκε η προσθήκη δυνατότητας αυξομείωσης της ταχύτητας με την οποία κινείται ο χαρακτήρας **Bob**. Η ρύθμιση της ταχύτητας γίνεται με τη χρήση συγκεκριμένων πλήκτρων, προσφέροντας πέντε διαβαθμίσεις ταχύτητας. Τα πλήκτρα που χρησιμοποιούνται για την αλλαγή της ταχύτητας επιτρέπουν την ομαλή μετάβαση μεταξύ των επιπέδων, εξασφαλίζοντας έτσι πιο δυναμική πλοήγηση του **Bob** μέσα στον λαβύρινθο. Οι αλλαγές αυτές υλοποιήθηκαν με την κατάλληλη προσαρμογή του κώδικα, ώστε να ενημερώνονται οι παράμετροι κίνησης σε πραγματικό χρόνο. Πιο συγκεκριμένα, έγιναν οι εξής αλλαγές:

Υλοποιήθηκε ο παρακάτω κώδικας, στο αρχείο **moveBob.cs**

```
public class moveBob : MonoBehaviour {

    public float speed = 5.0f;

    private float[] speedLevels = { 2f, 4f, 6f, 8f, 10f };

    private int currentSpeedIndex = 2;

    public float minX = -45f, maxX = 45f;

    public float minZ = -45f, maxZ = 45f;


    public LayerMask wallLayer;


    void Update() {

        if (Input.GetKeyDown(KeyCode.Alpha1)) currentSpeedIndex = 0;

        if (Input.GetKeyDown(KeyCode.Alpha2)) currentSpeedIndex = 1;

        if (Input.GetKeyDown(KeyCode.Alpha3)) currentSpeedIndex = 2;

        if (Input.GetKeyDown(KeyCode.Alpha4)) currentSpeedIndex = 3;

        if (Input.GetKeyDown(KeyCode.Alpha5)) currentSpeedIndex = 4;


        speed = speedLevels[currentSpeedIndex];
```

σύμφωνα με τον οποίο, οι ταχύτητες αποθηκεύονται στον πίνακα **speedLevels**, με πέντε επίπεδα ταχύτητας (2, 4, 6, 8, 10). Ο παίκτης μπορεί να αλλάξει την ταχύτητα του **Bob** πατώντας τα πλήκτρα **1, 2, 3, 4, 5**. Η τρέχουσα ταχύτητα (**speed**) ρυθμίζεται ανάλογα με την τιμή που επιλέγεται από το **currentSpeedIndex**.

2.2 Υλοποίηση του ερωτήματος (iii) και των (α) και (γ)

Για το ερώτημα (iii):

Στο τρίτο ερώτημα, ζητήθηκε η προσθήκη «**θησαυρών**» μέσα στον λαβύρινθο. Οι θησαυροί είναι κύβοι διαστάσεων **5x5x5**, με διαφορετικές υφές (**κεράσια**, **πορτοκάλια**, **λεμόνια**) και ορίστηκαν στα **Object** τους, με **Tag Collectibles**. Αυτοί εμφανίζονται σε **τυχαίες θέσεις** μέσα στο λαβύρινθο, παραμένουν για συγκεκριμένο χρόνο και εξαφανίζονται. Όταν ο χαρακτήρας «**ακουμπήσει**» έναν θησαυρό, αυτός **συρρικνώνεται** σταδιακά πριν εξαφανιστεί. Πιο συγκεκριμένα, έγιναν οι εξής αλλαγές:

Υλοποιήθηκε ο παρακάτω κώδικας, στο αρχείο **TreasureSpawner.cs**

```
private IEnumerator SpawnTreasure() {
    while (true) {
        GameObject randomPrefab = treasurePrefabs[Random.Range(0, treasurePrefabs.Length)];
        Vector3 spawnPosition = GetRandomPosition();
        GameObject treasure = Instantiate(randomPrefab, spawnPosition, Quaternion.Euler(0, 90, 90));
        Destroy(treasure, lifetime);
        yield return new WaitForSeconds(spawnInterval);
    }
}
```

σύμφωνα με τον οποίο, οι θησαυροί δημιουργούνται από μια λίστα **Prefabs** που περιέχει τα κεράσια, τα λεμόνια, και τα πορτοκάλια. Κάθε θησαυρός εμφανίζεται σε μια τυχαία θέση μέσω της **GetRandomPosition()** και καταστρέφεται μετά από ένα χρονικό διάστημα (**lifetime**). Η μέθοδος **SpawnTreasure** εκτελείται συνεχώς, εμφανίζοντας θησαυρούς ανά τακτά χρονικά διαστήματα (**spawnInterval**).

Υλοποιήθηκε ο παρακάτω κώδικας, στο αρχείο **CollectItems.cs**

```
private IEnumerator ShrinkAndDestroy(GameObject collectible) {
    Vector3 originalScale = collectible.transform.localScale;
    float shrinkDuration = 1f;
    float time = 0;
    while (time < shrinkDuration) {
        if (collectible != null) {
```

```
            collectible.transform.localScale = Vector3.Lerp(originalScale, Vector3.zero, time / shrinkDuration);
            time += Time.deltaTime;
        } else {
            yield break;
        }
        yield return null;
    }
    if (collectible != null) {
        Destroy(collectible);
    }
}
```

σύμφωνα με τον οποίο, ο θησαυρός **συρρικνώνεται σταδιακά** χρησιμοποιώντας τη μέθοδο **Vector3.Lerp** για να μειωθεί το μέγεθός του από την αρχική του κλίμακα στο μηδέν. Μετά τη συρρίκνωση, ο θησαυρός καταστρέφεται με την εντολή **Destroy**.

*Το **script TreasureSpawner.cs** έχει προστεθεί ως **component** σε ένα **Create Empty (TreasureManager)**, όπου έχει καθοριστεί η λίστα των **Prefabs**.*

*Το **script CollectItems.cs** έχει προστεθεί ως **component** στο **3D Object** του **Bob**. Έχουν προστεθεί τα **Collect Sound = Whoosh (Audio Source), Collect Effect = Shine, Cherry Points = 1, Lemon Points = 2** και **Orange Points = 3**.*

Για το ερώτημα (α):

Στο bonus ερώτημα (α), ζητήθηκε η προσθήκη εφέ και ήχου όταν ο χαρακτήρας **Bob** συλλέγει θησαυρούς. Για την υλοποίηση, ενσωματώθηκαν ηχητικά εφέ που ενεργοποιούνται τη στιγμή της συλλογής, καθώς και οπτικά εφέ, όπως λάμψεις ή αλλαγές χρώματος, για να ενισχυθεί η αλληλεπίδραση του παίκτη με το παιχνίδι. Πιο συγκεκριμένα, έγιναν οι εξής αλλαγές:

Υλοποιήθηκε ο παρακάτω κώδικας, στο αρχείο **CollectItems.cs**

```
if (collectEffect != null) {
    GameObject effect = Instantiate(collectEffect, other.transform.position, Quaternion.identity);
    Destroy(effect, 2f);
}
if (collectSound != null && audioSource != null) {
    audioSource.PlayOneShot(collectSound);
}
StartCoroutine(ShrinkAndDestroy(other.gameObject));
```

σύμφωνα με τον οποίο, το εφέ (**collectEffect**) εμφανίζεται στην τοποθεσία του θησαυρού και καταστρέφεται μετά από 2 δευτερόλεπτα και ο ήχος (**collectSound**) αναπαράγεται μέσω του **audioSource** όταν ο θησαυρός συλλέγεται.

Για το ερώτημα (γ):

Στο bonus ερώτημα (γ), ζητήθηκε η δημιουργία ενός συστήματος σκορ, ώστε ο **Bob** να κερδίζει πόντους καθώς συλλέγει θησαυρούς. Κάθε είδος θησαυρού αντιστοιχεί σε διαφορετική βαθμολογία, προσφέροντας έτσι ποικιλία στο **gameplay** και επιπλέον κίνητρο για εξερεύνηση. Το σκορ εμφανίζεται δυναμικά στην οθόνη και ενημερώνεται άμεσα μετά από κάθε συλλογή θησαυρού. Πιο συγκεκριμένα, έγιναν οι εξής αλλαγές:

Υλοποιήθηκε ο παρακάτω κώδικας, στο αρχείο **CollectItems.cs**

```
if (other.CompareTag("Collectible")) {
        if (other.name.Contains("cherry")) {
            scoreManager.AddScore(cherryPoints);
        } else if (other.name.Contains("lemon")) {
            scoreManager.AddScore(lemonPoints);
        } else if (other.name.Contains("orange")) {
            scoreManager.AddScore(orangePoints);
        }
```

σύμφωνα με τον οποίο, ανάλογα με το όνομα του θησαυρού (**cherry**, **lemon**, **orange**), προστίθενται οι αντίστοιχοι πόντοι μέσω της μεθόδου **AddScore**.

Υλοποιήθηκε ο παρακάτω κώδικας, στο αρχείο **ScoreManager.cs**

```
  public void AddScore(int points) {
      score += points;
      UpdateScoreUI();
  }
  private void UpdateScoreUI() {
      if (scoreText != null) {
          scoreText.text = "Score: " + score.ToString();
      }
  }
```

σύμφωνα με τον οποίο, το **ScoreManager** ενημερώνει το **UI** με τη νέα τιμή του σκορ μέσω της **UpdateScoreUI()**.

*Το **script ScoreManager.cs** έχει προστεθεί ως **component** σε ένα **Create Empty (GameManager)**, όπου έχουμε προσθέσει το **ScoreText**.*

2.3 Υλοποίηση του ερωτήματος (iv) και του (δ)

Για το ερώτημα (iv):

Μέσα στον λαβύρινθο εμφανίζονται τυχαία αντικείμενα-παγίδες σε απρόβλεπτες χρονικές στιγμές και για περιορισμένη διάρκεια. Οι παγίδες, που σχηματίζονται από σφαίρες διαμέτρου 5 με εφαρμοσμένη την υφή **death.jpg**, προκαλούν τον τερματισμό του παιχνιδιού όταν ο **Bob** έρθει σε επαφή με αυτές. Επίσης, ορίστηκαν στα **Object** τους, με **Tag DeathTrap**. Πιο συγκεκριμένα, έγιναν οι εξής αλλαγές:

Υλοποιήθηκε ο παρακάτω κώδικας, στο αρχείο **DeathTrapSpawner.cs**

```
private IEnumerator SpawnDeathTraps() {
    while (true) {
        Vector3 spawnPosition = GetRandomPosition();
        GameObject deathTrap = Instantiate(deathTrapPrefab, spawnPosition, Quaternion.identity);
        float lifetime = Random.Range(minLifetime, maxLifetime);
        Destroy(deathTrap, lifetime);
        yield return new WaitForSeconds(spawnInterval);
    }
}
```

σύμφωνα με τον οποίο, οι παγίδες εμφανίζονται σε τυχαίες θέσεις μέσω της **GetRandomPosition()**. Κάθε παγίδα έχει προκαθορισμένη διάρκεια ζωής (**trapLifetime**), μετά την οποία καταστρέφεται. Η εμφάνιση γίνεται ανά τυχαία χρονικά διαστήματα που κυμαίνονται μεταξύ **minLifetime** και **maxLifetime**.

Υλοποιήθηκε ο παρακάτω κώδικας, στο αρχείο **DeathTrapCollision.cs**

```
private void OnTriggerEnter(Collider other) {
    if (other.CompareTag("Player")) {
        FindObjectOfType<GameManager>().TriggerGameOver();
        Destroy(other.gameObject);
    }
}
```

σύμφωνα με τον οποίο, όταν ο **Bob** (**Player**) συγκρουστεί με μια παγίδα, καλείται η μέθοδος **TriggerGameOver()** από το **GameManager** για να τερματίσει το παιχνίδι. Ο

Bob καταστρέφεται με την εντολή **Destroy (other.gameObject)**, δηλώνοντας έτσι τον «θάνατό» του.

*Το **script DeathTrapSpawner.cs** έχει προστεθεί ως **component** σε ένα **Create Empty (DeathTrapSpawner)**, όπου έχουμε ορίσει ως **Prefab** το **Death** και τις τιμές **Spawn Interval = 10, Min Lifetime = 3** και **Max Lifetime = 8**.*

*Το **script DeathTrapCollision.cs** έχει προστεθεί ως **component** στο **3D Object** του **Bob**.*

Για το ερώτημα (δ):

Στο bonus ερώτημα (δ), ζητήθηκε η εμφάνιση μηνύματος «**Game Over**» όταν το παιχνίδι τερματίζει. Το μήνυμα εμφανίζεται σε κεντρική θέση στην οθόνη, με ευδιάκριτη γραμματοσειρά και κατάλληλο σχεδιασμό, ώστε να είναι άμεσα αντιληπτό από τον παίκτη. Αυτή η προσθήκη ολοκληρώνει την εμπειρία του παιχνιδιού, παρέχοντας έναν σαφή τερματισμό και ενισχύοντας την αίσθηση ολοκλήρωσης για τον παίκτη. Πιο συγκεκριμένα, έγιναν οι εξής αλλαγές:

Υλοποιήθηκε ο παρακάτω κώδικας, στο αρχείο **GameManager.cs**

```
public void TriggerGameOver() {
    if (!isGameOver) {
        isGameOver = true;
        if (gameOverText != null) {
            gameOverText.gameObject.SetActive(true);
        }
        Time.timeScale = 0;
    }
}
```

σύμφωνα με τον οποίο, όταν καλείται η μέθοδος **TriggerGameOver**, ενεργοποιείται το αντικείμενο **gameOverText** που εμφανίζει το μήνυμα "**Game Over**". Η ταχύτητα του χρόνου (**Time.timeScale**) ορίζεται σε **0**, διακόπτοντας την κίνηση και τη φυσική του παιχνιδιού.

*Το **script GameManager.cs** έχει προστεθεί ως **component** σε ένα **Create Empty (GameManager)**, όπου έχουμε προσθέσει το **GameOverText**.*

2.4 Υλοποίηση του ερωτήματος (v)

Στο πέμπτο ερώτημα, ζητήθηκε η υλοποίηση μιας απλής κάμερας που επιτρέπει στον χρήστη να παρακολουθεί τη σκηνή από οποιαδήποτε γωνία, θέση και ύψος. Η κάμερα ελέγχεται από τα βελάκια του πληκτρολογίου για την κίνηση στους άξονες **x** και **z** του συστήματος παγκόσμιων συντεταγμένων, ενώ τα πλήκτρα **<+>** και **<->** χρησιμοποιούνται για την αλλαγή ύψους στον άξονα **y**. Επιπλέον, με το πλήκτρο , η κάμερα περιστρέφεται γύρω από τον εαυτό της, πραγματοποιώντας περιστροφή γύρω από τον άξονα **y**, με κέντρο περιστροφής το κέντρο της κάμερας. Οι αλλαγές αυτές επιτρέπουν στο χρήστη να εξερευνήσει τη σκηνή από διάφορες οπτικές γωνίες, ενισχύοντας την αίσθηση ελευθερίας και ελέγχου στο περιβάλλον. Πιο συγκεκριμένα, έγιναν οι εξής αλλαγές:

Υλοποιήθηκε ο παρακάτω κώδικας, στο αρχείο **CameraController.cs** για την κίνηση της κάμερας στον άξονα **X** και **Z**, για την κίνηση της κάμερας στον άξονα **Y**, για την κίνηση της κάμερας γύρω από στον άξονα **Y**

```
public class CameraController : MonoBehaviour {
    public float moveSpeed = 10f;
    public float rotationSpeed = 85f;
    void Update() {
        float moveX = Input.GetAxis("Horizontal") * moveSpeed * Time.deltaTime;
        float moveZ = Input.GetAxis("Vertical") * moveSpeed * Time.deltaTime;
        float moveY = 0f;
        if (Input.GetKey(KeyCode.Plus) || Input.GetKey(KeyCode.KeypadPlus)) {
            moveY = moveSpeed * Time.deltaTime;
        } else if (Input.GetKey(KeyCode.Minus) || Input.GetKey(KeyCode.KeypadMinus)) {
            moveY = -moveSpeed * Time.deltaTime;
        }
        transform.Translate(new Vector3(moveX, moveY, moveZ), Space.World);
        if (Input.GetKey(KeyCode.R)) {
            transform.Rotate(Vector3.up, rotationSpeed * Time.deltaTime, Space.World);
        }
    }
}
```

σύμφωνα με τον οποίο, επιτρέπει την κίνηση και περιστροφή της κάμερας στο χώρο. Η μετακίνηση γίνεται με τα πλήκτρα **"Horizontal"** (π.χ. αριστερά/δεξιά βέλη ή A/D) και **"Vertical"** (π.χ. πάνω/κάτω βέλη ή W/S), ελέγχοντας τις συντεταγμένες **X** και **Z**, ενώ η ταχύτητα καθορίζεται από τη μεταβλητή **moveSpeed**. Η μετακίνηση στον άξονα Y (πάνω/κάτω) πραγματοποιείται με τα πλήκτρα **Plus/KeypadPlus** και **Minus/KeypadMinus**. Όλες οι κινήσεις είναι ανεξάρτητες του τοπικού χώρου και βασίζονται στον παγκόσμιο χώρο (**Space.World**). Επιπλέον, με το πλήκτρο **R**, η κάμερα περιστρέφεται γύρω από τον άξονα **Y** με ταχύτητα που ορίζεται από τη μεταβλητή **rotationSpeed**. Η μέθοδος **Update()** διασφαλίζει ότι αυτές οι ενέργειες ενημερώνονται σε κάθε καρέ του παιχνιδιού.

*Το **script CameraController.cs** έχει προστεθεί ως **component** σε ένα **Create Empty (MainCamera)**, όπου έχουμε προσθέσει το **Move Speed = 10** και **Rotation Speed = 85**.*

Αναφορές

- Ενδεικτικά video από την ιστοσελίδα του μαθήματος στο e-course (Βασιλική Σταμάτη).
- Introduction to 3D Animation Systems
- Beginner Movement Tutorial
- 3D Movement Tutorial
- Layer Mask
- Physics.CheckBox
- IEnumerator
- Random
- Math.round