Treasure Bob

Unity Project

Vasileios Skarafigkas

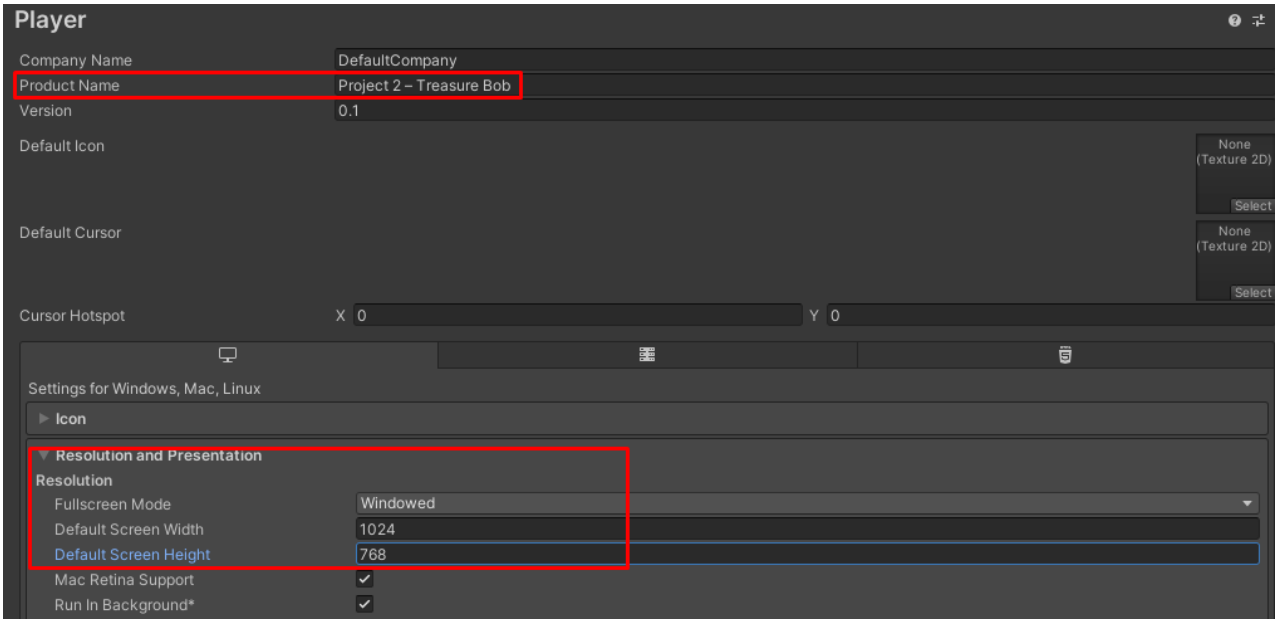
Βασίλειος Σκαραφίγκας

Περιεχόμενα

Δημιουργία Λαβύρινθου και Σκηνής στην Unity

Για να ικανοποιήσουμε τις απαιτήσεις του πρώτου ερωτήματος, ξεκινήσαμε με τις παρακάτω ρυθμίσεις στην **Unity**

Γενικές Ρυθμίσεις της Εφαρμογής



- **Ανάλυση Παραθύρου**:
    - Πήγαμε στο μενού **File > Build Settings > Player Settings**.
    - Στην ενότητα **Resolution and Presentation**, ορίσαμε:
        - **Default Is Full Screen** = **Unchecked** (Παράθυρο, όχι πλήρης οθόνη).
        - **Default Screen Width** = `1024`.
        - **Default Screen Height** = `768`.

- **Τίτλος Εφαρμογής**:
    - Στην ίδια ενότητα **Player Settings**, στη **Product Name**, ορίσαμε τον τίτλο της εφαρμογής ως **"Project 2 – Treasure Bob"**.

Δημιουργία του Εδάφους (Plane - Floor)

- **Προσθήκη Plane**:
    - Στο **Hierarchy**, κάναμε **Right Click > 3D Object > Plane**.
    - Ονομάσαμε το αντικείμενο **"Floor"**.

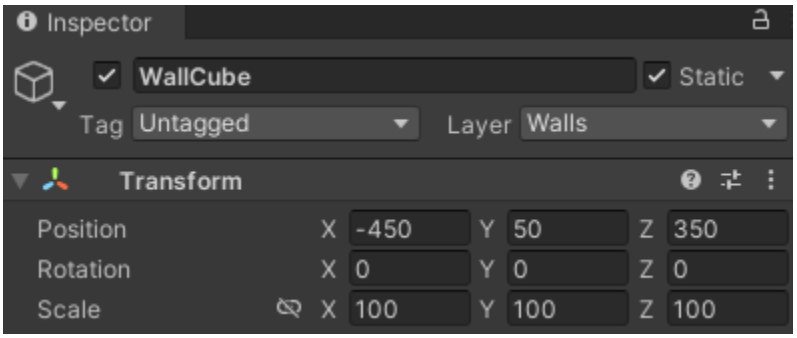
- **Διαστάσεις του Plane**:
    - Ρυθμίσαμε το **Scale** του Plane ως εξής:
        - **X = 100**, **Y = 1**, **Z = 100**.
    - Με αυτό το Scale, το Plane αποτελείται από **100 x 100 τετραγωνάκια** (10x10 = 100 κατά μήκος των x και z αξόνων).

- **Τοποθέτηση του Plane**:
    - Θέση του Plane:
        - **Position = (0, 0, 0)**.
    - Το Plane τοποθετείται στο επίπεδο `y = 0`.

- **Εφαρμογή Υφής (floor.jpg)**:
    - Στα **Assets**, εισάγαμε την υφή **floor.jpg**:
        - **Right Click > Import New Asset** και επιλέξαμε το αρχείο `floor.jpg`.
    - Δημιουργήσαμε ένα νέο **Material**:
        - **Right Click > Create > Material**.
        - Στο Material, σύραμε την υφή **floor.jpg** στο πεδίο **Albedo**.
    - Εφαρμόσαμε το **Material** στο **Floor** σύροντάς το πάνω στο Plane.

Σχεδιασμός του Λαβύρινθου (Walls)

- **Κύβοι για τα Τοιχώματα**:
    - Για να δημιουργήσουμε τα **τοίχωματά** του λαβύρινθου:
        - Πήγαμε στο **Hierarchy > Right Click > 3D Object > Cube**.
        - Ονομάσαμε το πρώτο Cube **"WallCube"**.

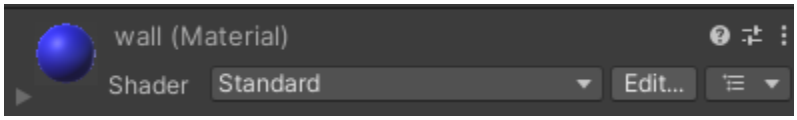


- **Διαστάσεις των Τοιχωμάτων**:
    - Ρυθμίσαμε το **Scale** του Cube ως εξής:
        - **X = 100**, **Y = 100**, **Z = 100**.
    - Αυτό δίνει στους κύβους τις διαστάσεις **10x10x10**, όπως ζητείται.

- **Τοποθέτηση των Κύβων**:
    - Με βάση το σχέδιο του λαβύρινθου (σύμφωνα με την Εικόνα 1 του pdf της εκφώνησης), δημιουργήσαμε αντίγραφα του **WallCube** και τα τοποθετήσαμε στις κατάλληλες θέσεις.
    - Χρησιμοποιήσαμε το **Duplicate**:
        - **Ctrl + D** για γρήγορη αντιγραφή των κύβων.
    - Παραδείγματα θέσεων κύβων (σε x, y, z):
        - **Περιμετρικά Τοιχώματα**:
            - Κύβοι τοποθετήθηκαν σε **x = -500 έως 500**, **z = -500 έως 500**.
            - `y = 50` (κέντρο του ύψους των κύβων).
        - **Εσωτερικά Τοιχώματα**:
            - Ορίστηκαν σύμφωνα με το σχεδιασμό από την **Εικόνα 1**.

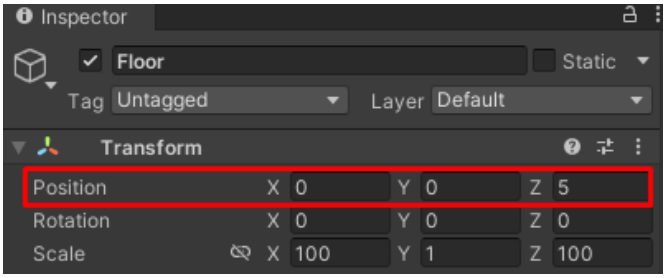
- **Χρώμα των Τοιχωμάτων**:
    - Δημιουργήσαμε ένα νέο **Material** για τα τοιχώματα:

- **Right Click > Create > Material**.
- Στο πεδίο **Albedo**, επιλέξαμε το χρώμα **μπλε**.



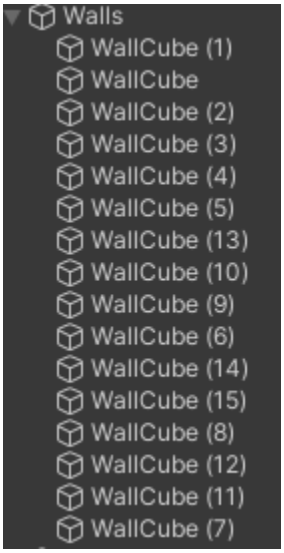
- Εφαρμόσαμε το Material στα τοιχώματα σύροντάς το πάνω στους κύβους.

Κέντρο του Λαβύρινθου



- Το **κέντρο** του λαβύρινθου τοποθετήθηκε στο σημείο:
  - **(0, 0, 5)** στο **παγκόσμιο σύστημα συντεταγμένων**.
- Αυτό εξασφαλίστηκε με προσεκτική τοποθέτηση των κύβων γύρω από αυτό το σημείο.

Οργάνωση των Αντικειμένων



Για καλύτερη οργάνωση:

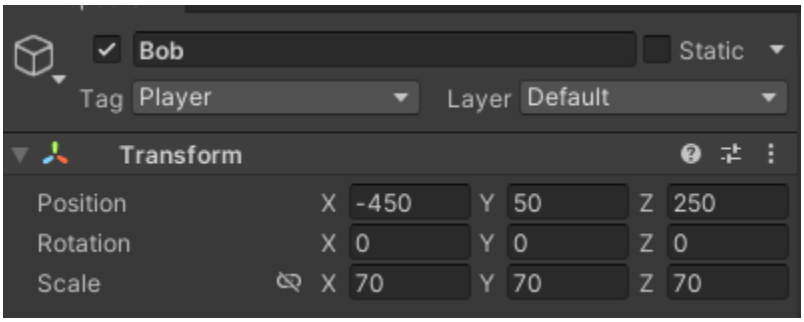
- Δημιουργήσαμε ένα **Empty GameObject** ονομάζοντάς το **"Walls"**.
- Σύραμε όλα τα **WallCubes** (τοιχώματα) **Walls** GameObject.

Δημιουργία και Πλοήγηση του Treasure Bob

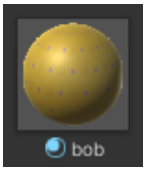
Δημιουργία του Treasure Bob

Για τον χαρακτήρα του παιχνιδιού, **Treasure Bob**, υλοποιήσαμε μια **σφαίρα** που λειτουργεί ως ο βασικός παίκτης. Ακολουθήσαμε τα παρακάτω βήματα:

1. **Δημιουργία Σφαίρας (Sphere)**:
   - Από το **Hierarchy**, δημιουργήσαμε ένα αντικείμενο σφαίρας:
     - **Right Click > 3D Object > Sphere**.
   - Ονομάσαμε το αντικείμενο **"TreasureBob"** για καλύτερη οργάνωση.
2. **Ρυθμίσεις Σφαίρας**:

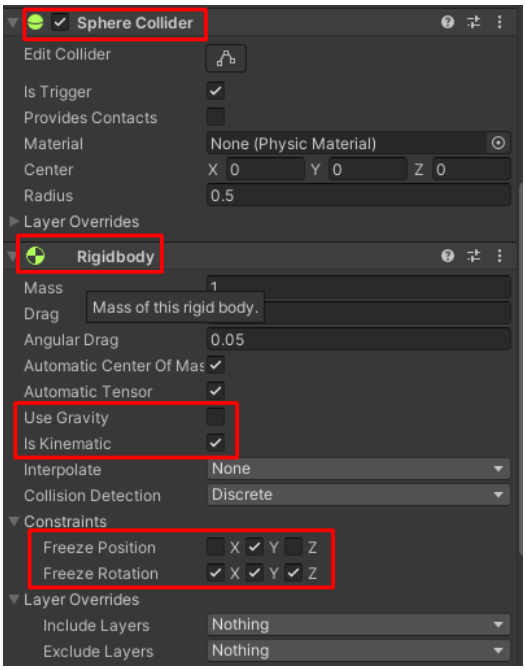


   - **Διάμετρος**: Για να ορίσουμε τη διάμετρο της σφαίρας ως **7** μονάδες, προσαρμόσαμε το **Scale**:
     - **Scale = (70, 70, 70)**.
   - **Θέση Εκκίνησης**: Τοποθετήσαμε τον **Treasure Bob** στην «είσοδο» του λαβύρινθου:
     - **Position = (-450, 50, 250)** (ύψος 50 για να μην εφάπτεται στο έδαφος με τη σφαίρα 7 διαμέτρου και να φαινεται πως αιωρείται).
3. **Εφαρμογή Υφής (bob.jpg)**:



   - Εισάγαμε την εικόνα **bob.jpg** στα **Assets**:
     - **Right Click > Import New Asset** και επιλέξαμε το αρχείο `bob.jpg`.
   - Δημιουργήσαμε ένα νέο **Material**:
     - **Right Click > Create > Material**.
     - Στο Material, σύραμε την υφή **bob.jpg** στο πεδίο **Albedo**.
   - Εφαρμόσαμε το **Material** στη σφαίρα **TreasureBob**.

4. **Ρύθμιση Collider και Gravity**:
    - **Sphere Collider**: Η σφαίρα περιλαμβάνει ήδη ένα **Sphere Collider**.
    - **Gravity**: Απενεργοποιήσαμε τη βαρύτητα:
        - Πρόσθετο **Rigidbody** με ρύθμιση **Use Gravity = false**.



Πλοήγηση του Treasure Bob

Για την πλοήγηση του Bob μέσα στον λαβύρινθο, δημιουργήσαμε το script **BobMovement** με βάση τις απαιτήσεις:

1. **Βασική Λειτουργία**:
    - Ο παίκτης μπορεί να μετακινεί τον **Bob** με τα **πλήκτρα `j`, `l`, `i`, `k`**:
        - `j` → Κίνηση αριστερά στον **x** άξονα.
        - `l` → Κίνηση δεξιά στον **x** άξονα.
        - `i` → Κίνηση μπροστά στον **z** άξονα.
        - `k` → Κίνηση πίσω στον **z** άξονα.
2. **Αλλαγή Ταχύτητας**:
    - Προσθέσαμε πέντε διαβαθμίσεις ταχύτητας με τα **πλήκτρα z (μείωση) και x (αύξηση):**
        - **Πίνακας Ταχυτήτων**: `{100.0f, 120.0f, 140.0f, 160.0f, 200.0f}`.
        - Η αρχική ταχύτητα είναι **120.0f**.
3. **Κίνηση και Έλεγχος Σύγκρουσης**:
    - Χρησιμοποιήσαμε το **`Physics.BoxCast`** για να αποτρέψουμε τον Bob από το να περάσει μέσα από τοίχους:
        - Ο **BoxCast** εκτελεί ανίχνευση σε όγκο γύρω από τον Bob.
        - Αν εντοπίσει τοίχο, ο Bob δεν κινείται προς αυτήν την κατεύθυνση.

- Προσθέσαμε έλεγχο με **LayerMask** ώστε η ανίχνευση να γίνεται **μόνο με τοίχους**.

4. **Περιορισμός στα Όρια του Λαβύρινθου**:
   - Ορίσαμε ότι ο Bob δεν μπορεί να βγει εκτός των ορίων του Plane:
     - Χρησιμοποιώντας τα **bounds** του εδάφους (**planeBounds**) ελέγχουμε κάθε νέα θέση πριν την εφαρμόσουμε.

Πλήρη Ανάλυση Script (BobMovement.cs) κίνησης

Ο παραπάνω κώδικας αποτελεί το script που είναι υπεύθυνο για τον έλεγχο της κίνησης και της συμπεριφοράς του χαρακτήρα **Treasure Bob** μέσα στον λαβύρινθο. Ας το αναλύσουμε **γραμμή-γραμμή** με περιγραφή κάθε μεθόδου και λειτουργίας.

Μεταβλητές και Δηλώσεις

```
public class BobMovement : MonoBehaviour
{
    public Transform Bob; // Αναφορά στον Bob (Sphere)
    public GameObject Plane; // Αναφορά στο Plane
    public float[] speedLevels = { 100.0f, 120.0f, 140.0f, 160.0f, 200.0f }; // Διαβαθμίσεις ταχύτητας
    public int currentSpeedLevel = 1; // Αρχικό επίπεδο ταχύτητας
    public LayerMask wallLayer; // Layer για τους τοίχους

    private float moveStep; // Ταχύτητα κίνησης
    private Bounds planeBounds;
```

- **Bob**: Αναφορά στη **σφαίρα του Bob** μέσω του **Transform** για να ενημερώνουμε τη θέση του.
- **Plane**: Το έδαφος (Floor) που περιορίζει την κίνηση του Bob.
- **speedLevels**: Ένας πίνακας που περιέχει **πέντε διαβαθμίσεις ταχύτητας** (100.0f έως 200.0f).
- **currentSpeedLevel**:Το τρέχον επίπεδο ταχύτητας. Ξεκινά από τη δεύτερη διαβάθμιση (120**f**).
- wallLayer: Χρησιμοποιείται για να ορίσουμε Layer Mask ώστε να ανιχνεύσουμε μόνο τους τοίχους.
- **moveStep**: Η ταχύτητα κίνησης του Bob. Καθορίζεται δυναμικά από τον πίνακα **speedLevels**.
- **planeBounds**: Τα όρια του **Plane**, υπολογισμένα μέσω του **MeshRenderer**.

Start Method

```
void Start()
{
    // Υπολογισμός των ορίων του Plane
    MeshRenderer planeRenderer = Plane.GetComponent<MeshRenderer>();
    planeBounds = planeRenderer.bounds;

    // Ορισμός αρχικής ταχύτητας
    moveStep = speedLevels[currentSpeedLevel];
}
```

- **Υπολογισμός των ορίων του Plane**:

Χρησιμοποιούμε το **MeshRenderer.bounds** για να πάρουμε τα όρια του εδάφους. Με αυτά τα όρια περιορίζουμε την κίνηση του Bob.

- **Ορισμός Αρχικής Ταχύτητας**:

Η ταχύτητα κίνησης ορίζεται από την αρχική διαβάθμιση του πίνακα `speedLevels`.

Update Method

```
void Update()
{
    AdjustSpeed(); // Διαχείριση διαβαθμίσεων ταχύτητας
    MoveBob();     // Κίνηση του Bob
}
```

Η **Update** καλείται σε κάθε frame και ελέγχει:

1. Την **ταχύτητα** του Bob με τη μέθοδο AdjustSpeed().
2. Την **κίνηση** του Bob με τη μέθοδο MoveBob().

MoveBob Method

```
void MoveBob()
{
    Vector3 direction = Vector3.zero;

    if (Input.GetKey(KeyCode.J)) direction = Vector3.left;  // Αριστερά
    if (Input.GetKey(KeyCode.L)) direction = Vector3.right; // Δεξιά
    if (Input.GetKey(KeyCode.I)) direction = Vector3.forward; // Μπροστά
    if (Input.GetKey(KeyCode.K)) direction = Vector3.back; // Πίσω

    if (direction != Vector3.zero)
    {
        // Νέα θέση του Bob
        Vector3 newPosition = Bob.position + direction.normalized * moveStep * Time.deltaTime;

        // Έλεγχος για σύγκρουση με τοίχους μέσω BoxCast
        if (!Physics.BoxCast(Bob.position, Vector3.one * 35.0f, direction, Quaternion.identity, moveStep * Time.deltaTime, wallLayer))
        {
            // Έλεγχος αν η θέση είναι εντός των ορίων του Plane
            if (IsWithinBounds(newPosition))
            {
                Bob.position = newPosition;
            }
        }
        else
        {
            Debug.Log("Ο Bob εμποδίζεται από τον τοίχο!");
        }
    }
}
```

1. **Καθορισμός Κατεύθυνσης**:
   Χρησιμοποιούμε τα **πλήκτρα `j`, `l`, `i`, `k`** για να ορίσουμε την κατεύθυνση της κίνησης.
2. **Υπολογισμός Νέας Θέσης**:
   Η νέα θέση του Bob υπολογίζεται προσθέτοντας την κατεύθυνση στην τρέχουσα θέση και προσαρμόζοντας με την ταχύτητα **moveStep**.
3. **Έλεγχος Σύγκρουσης με Τοιχώματα:**
   Χρησιμοποιούμε το Physics.BoxCast για να ανιχνεύσουμε αν υπάρχει εμπόδιο (τοίχος) στην κατεύθυνση που κινείται ο Bob. Αν δεν υπάρχει εμπόδιο, ελέγχουμε αν η νέα θέση είναι εντός των ορίων του εδάφους με τη μέθοδο IsWithinBounds().
4. **Ενημέρωση Θέσης:** Αν η θέση είναι έγκυρη, ενημερώνουμε το Bob.position.

AdjustSpeed Method

```
void AdjustSpeed()
{
    // Μείωση ταχύτητας με Z
    if (Input.GetKeyDown(KeyCode.Z) && currentSpeedLevel > 0)
    {
        currentSpeedLevel--;
        moveStep = speedLevels[currentSpeedLevel];
        Debug.Log("Ταχύτητα μειώθηκε σε: " + moveStep);
    }

    // Αύξηση ταχύτητας με X
    if (Input.GetKeyDown(KeyCode.X) && currentSpeedLevel < speedLevels.Length - 1)
    {
        currentSpeedLevel++;
        moveStep = speedLevels[currentSpeedLevel];
        Debug.Log("Ταχύτητα αυξήθηκε σε: " + moveStep);
    }
}
```

- **Αύξηση/Μείωση Ταχύτητας**:
    - Πλήκτρο **z**: Μειώνει το επίπεδο ταχύτητας.
    - Πλήκτρο **x**: Αυξάνει το επίπεδο ταχύτητας.

- **Έλεγχος Ορίων**:

Διασφαλίζουμε ότι το επίπεδο ταχύτητας παραμένει εντός του πίνακα `speedLevels`.

IsWithinBounds Method && OnTriggerEnter Method

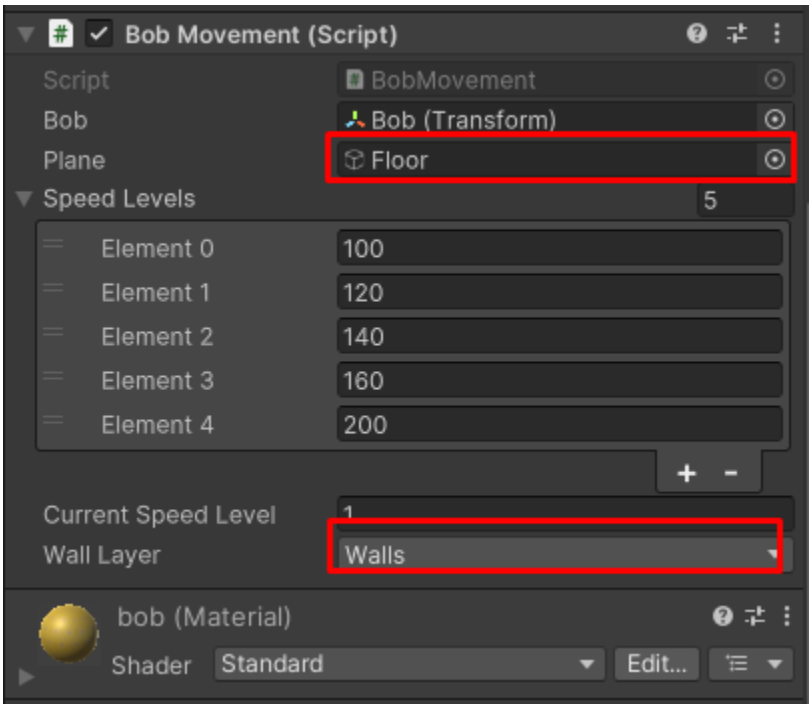
```
bool IsWithinBounds(Vector3 position)
{
    // Έλεγχος αν η νέα θέση είναι μέσα στα όρια του Plane
    return (position.x >= planeBounds.min.x && position.x <= planeBounds.max.x &&
            position.z >= planeBounds.min.z && position.z <= planeBounds.max.z);
}

void OnTriggerEnter(Collider other)
{
    if (other.CompareTag("Trap"))
    {
        FindObjectOfType<GameOverManager>().TriggerGameOver();
    }
}
```

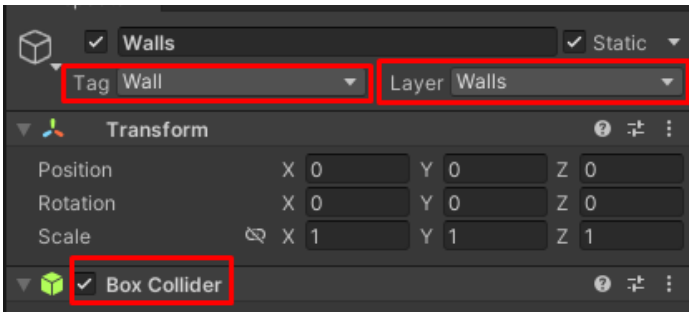
Ελέγχουμε αν η νέα θέση του Bob βρίσκεται **εντός των ορίων του εδάφους**.

Αν ο **Bob** ακουμπήσει ένα αντικείμενο με **Tag = "Trap"**, ενεργοποιούμε τη μέθοδο TriggerGameOver**()** από το script GameOverManager.

Σχέσεις Αντικειμένων και Ρυθμίσεις στο Unity



1. **Bob και Script**:
   - Προσθέσαμε το **BobMovement.cs** στο αντικείμενο **TreasureBob**.
   - Στο **Inspector** του **BobMovement** συνδέσαμε:
     - **Bob** → Αναφορά στο Transform της σφαίρας.
     - **Plane** → Το έδαφος του λαβύρινθου.
     - **WallLayer** → Layer των τοίχων (επιλέχθηκε ως "Wall").
2. **Τοίχοι και Layer**:



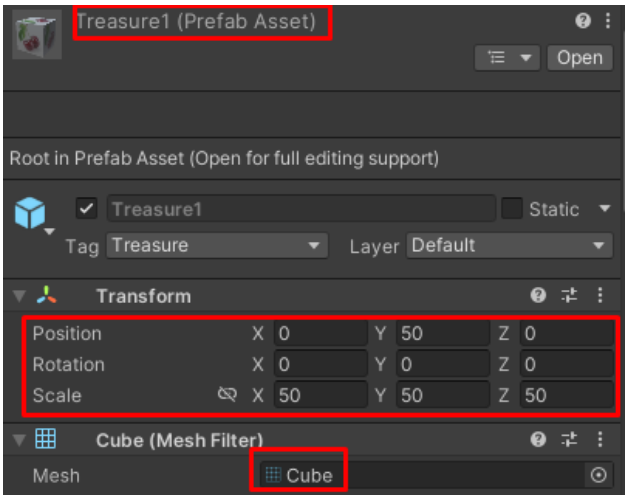
   - Όλοι οι κύβοι του λαβύρινθου (τοίχοι) έχουν:
     - **Box Collider** για ανίχνευση σύγκρουσης.
     - Αντιστοίχιση στο **Layer = "Walls"** για σωστή αναγνώριση μέσω **BoxCast**.
3. **Ρυθμίσεις Rigidbody στον** Bob:
   - Προσθέσαμε **Rigidbody** στον Bob:
     - **Use Gravity = false** για να παραμείνει στο ίδιο ύψος (y σταθερό).
     - **Is Kinematic = true** ώστε ο Bob να κινείται μέσω script και όχι φυσικής.

Δημιουργία Θησαυρών, Εφέ, Ήχου και Συστήματος Σκορ

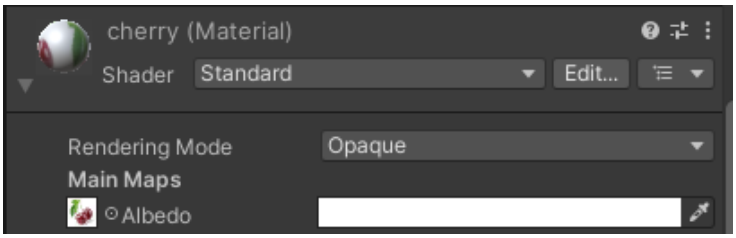
Δημιουργία των Θησαυρών

Για να δημιουργήσουμε τους θησαυρούς στο λαβύρινθο, ακολουθήσαμε τα παρακάτω βήματα:

1. **Δημιουργία Prefabs Θησαυρών**:

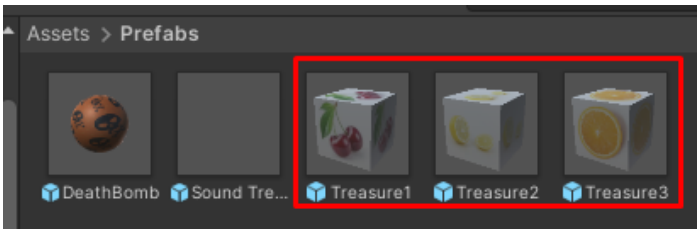


   - Χρησιμοποιήσαμε **3 κύβους** (Cubes) για να αναπαραστήσουμε τους θησαυρούς:
     - **Διαστάσεις**: `5 x 5 x 5` με Scale **(50, 50, 50)**.
     - **Position**: 0 x 50 x 0 (για να αιωρείται όπως ο Bob).
     - Εφαρμόσαμε **υφές** για κάθε θησαυρό:
       - **Κεράσια**: Υφή `cherry.jpg`.
       - **Πορτοκάλια**: Υφή `orange.jpg`.
       - **Λεμόνια**: Υφή `lemon.jpg`.



   - Στα **Assets**:

     - Δημιουργήσαμε 3 διαφορετικά **Materials** με τις υφές και τα εφαρμόσαμε στους κύβους.



- Αποθηκεύσαμε κάθε κύβο ως **Prefab**.

2. **Script: TreasureSpawner**
   Η λογική δημιουργίας των θησαυρών σε τυχαίες θέσεις υλοποιήθηκε στο **TreasureSpawner.cs**.

- Η τυχαία επιλογή θέσης γίνεται με την **Random.Range** για τις συντεταγμένες x και z εντός των ορίων του λαβύρινθου:

```
// Επαναλαμβάνουμε μέχρι να βρούμε έγκυρη θέση
do
{
    float randomX = Random.Range(mazeBounds.min.x, mazeBounds.max.x);
    float randomZ = Random.Range(mazeBounds.min.z, mazeBounds.max.z);
    spawnPosition = new Vector3(randomX, 25.0f, randomZ);
```

- Η αποφυγή εμφάνισης μέσα σε τοίχους γίνεται με **Physics.CheckBox**, το οποίο ελέγχει αν υπάρχει σύγκρουση με το Layer των τοίχων:

```
} while (Physics.CheckBox(spawnPosition, Vector3.one * 25.0f, Quaternion.identity, wallLayer));
```

- Η επιλογή τυχαίου τύπου θησαυρού γίνεται από τον πίνακα των **Prefabs**:

```
// Επιλογή τυχαίου θησαυρού
GameObject treasurePrefab = treasurePrefabs[Random.Range(0, treasurePrefabs.Length)];
```

3. **Χρονική Διαχείριση Θησαυρών**:

   Οι θησαυροί εμφανίζονται ανά **`spawnInterval`** δευτερόλεπτα και παραμένουν για **`treasureLifetime`** δευτερόλεπτα πριν καταστραφούν

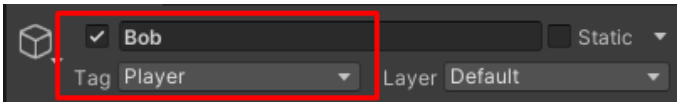
```
// Καταστροφή του θησαυρού μετά από χρόνο
Destroy(treasure, treasureLifetime);
```

Αντίδραση Όταν ο Bob Πιάσει Θησαυρό

Για να ορίσουμε τη συμπεριφορά του θησαυρού όταν τον ακουμπήσει ο **Bob**, δημιουργήσαμε το **Treasure.cs**.

Πώς το πετυχαίνουμε στον κώδικα:

1. **Ανίχνευση Σύγκρουσης**:



   - Χρησιμοποιούμε το **`OnTriggerEnter`** για να ανιχνεύσουμε όταν ο Bob (που έχει **Tag = "Player"**) ακουμπήσει τον θησαυρό:

```
void OnTriggerEnter(Collider other)
{
    if (other.CompareTag("Player")) // Ο Bob έχει tag "Player"
    {
        PlayPickupEffects();
        ScoreManager.instance.AddScore(points); // Προσθήκη στο σκορ
        StartCoroutine(ShrinkAndDestroy());
    }
}
```

2. **Εφαρμογή Εφέ και Ήχου**:
   - Εφέ **Particle System**:
     - Δημιουργούμε ένα **Prefab** με **Particle System** (π.χ. λάμψη ή έκρηξη) και το εμφανίζουμε
   - Ήχος:
     - Παίζουμε τον ήχο συλλογής με το **`AudioSource.PlayClipAtPoint`**

```
void PlayPickupEffects()
{
    Instantiate(pickupEffect, transform.position, Quaternion.identity);
    AudioSource.PlayClipAtPoint(pickupSound, transform.position);
}
```

3. **Σμίκρυνση και Εξαφάνιση Θησαυρού**:
   - Χρησιμοποιούμε **`Vector3.Lerp`** για ομαλή σμίκρυνση της κλίμακας του θησαυρού:
   - Μετά τη σμίκρυνση, ο θησαυρός διαγράφεται:

```
IEnumerator ShrinkAndDestroy()
{
    float shrinkDuration = 0.5f;
    Vector3 originalScale = transform.localScale;

    for (float t = 0; t < shrinkDuration; t += Time.deltaTime)
    {
        transform.localScale = Vector3.Lerp(originalScale, Vector3.zero, t / shrinkDuration);
        yield return null;
    }

    Destroy(gameObject);
```

Σύστημα Σκορ

Η διαχείριση του σκορ πραγματοποιείται μέσω του **ScoreManager.cs**.

Πώς το πετυχαίνουμε στον κώδικα:

1. **Singleton Pattern**:
   - Το **ScoreManager** χρησιμοποιεί Singleton για εύκολη πρόσβαση από οποιοδήποτε script
2. **Προσθήκη Πόντων**:
   - Η μέθοδος `AddScore` προσθέτει τους πόντους και ενημερώνει το UI
3. **Ενημέρωση UI**:
   - Το UI Text ενημερώνεται με την τρέχουσα τιμή του σκορ

```
using System.Collections;
using System.Collections.Generic;
using UnityEngine;
using UnityEngine.UI;

public class ScoreManager : MonoBehaviour
{
    public static ScoreManager instance; // Singleton για εύκολη πρόσβαση
    public int score = 0; // Το συνολικό σκορ
    public Text scoreText; // UI Text για εμφάνιση σκορ

    void Awake()
    {
        if (instance == null)
        {
            instance = this;
        }
        else
        {
            Destroy(gameObject);
        }
    }

    void Start()
    {
        UpdateScoreText();
    }

    public void AddScore(int points)
    {
        score += points; // Προσθήκη πόντων
        UpdateScoreText();
    }

    void UpdateScoreText()
    {
        scoreText.text = "Score: " + score;
    }
}
```

4. **Στο Script Treasure.cs για την βαθμολογία:**

- Προσθέσαμε μια δημόσια μεταβλητή `points` η οποία επιτρέπει να ορίζουμε ξεχωριστούς πόντους για κάθε θησαυρό:

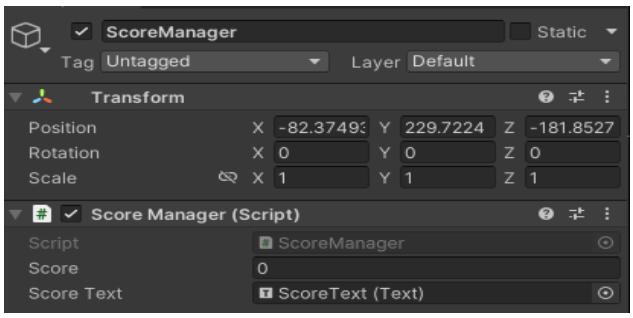
```
public int points = 10; // Πόντοι για τον συγκεκριμένο θησαυρό
```

- Όταν ο Bob ακουμπά έναν θησαυρό, ο **ScoreManager** καλείται να προσθέσει τους πόντους που αντιστοιχούν στο **specific treasure**:

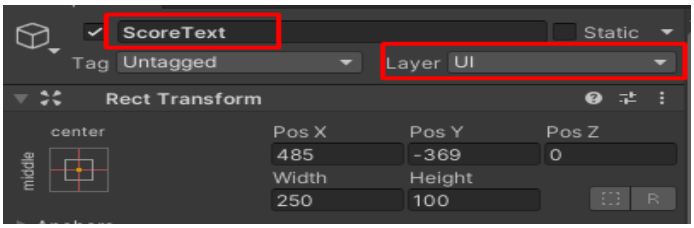
```
ScoreManager.instance.AddScore(points); // Προσθήκη στο σκορ
```

Οπτικοποίηση και Σχέσεις στο Unity

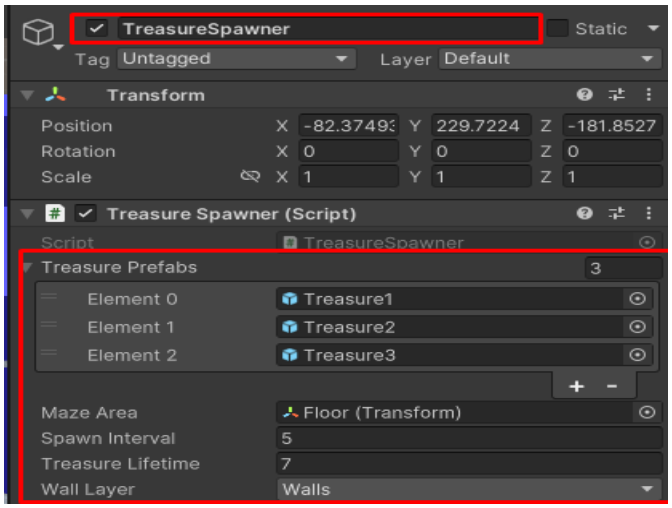
1. **Prefab Θησαυρών**:
    - Δημιουργήσαμε **3 Prefabs** θησαυρών με διαφορετικές υφές και τα προσθέσαμε στον πίνακα `treasurePrefabs`.
2. **ScoreManager**:



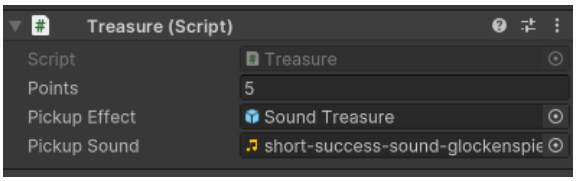
    - Δημιουργήσαμε ένα **Empty GameObject** με το όνομα **"ScoreManager"** και προσθέσαμε το script **ScoreManager.cs**.



    - Συνδέσαμε το UI Text στο πεδίο **`scoreText`**.
3. **TreasureSpawner**:
    - Δημιουργήσαμε ένα **Empty GameObject** με το όνομα **"TreasureSpawner"** και προσθέσαμε το **TreasureSpawner.cs**.
    - Συνδέσαμε τα **Prefabs** θησαυρών και το **mazeArea**.



4. **Treasure**:



   - Το script **Treasure.cs** προστέθηκε σε κάθε Prefab θησαυρού.
   - Συνδέσαμε:
     - **pickupEffect**: Το εφέ.
     - **pickupSound**: Το αρχείο ήχου.
5. **Ορισμός Σκορ για Κάθε Prefab στο Unity:**

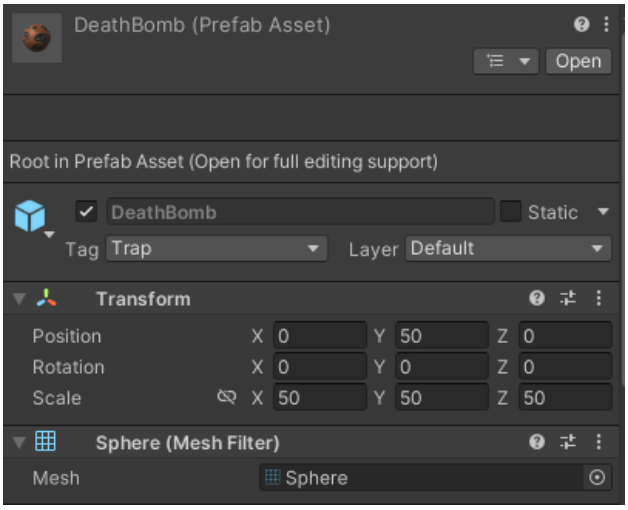
   - **Επιλογή του κάθε Prefab**:
     - Επίλεγουμε το **Prefab** του κάθε θησαυρού (π.χ. Cherry, Orange, Lemon).
   - **Ρύθμιση του `points`**:
     - Στο **Inspector** του **Prefab**, βρισκουμε το script **Treasure**.
     - Στο πεδίο **`points`**, εισάγουμε τη βαθμολογία:
       - **Κεράσια (Cherry)**: 5 πόντοι.
       - **Πορτοκάλια (Orange)**: 10 πόντοι.
       - **Λεμόνια (Lemon)**: 15 πόντοι

Δημιουργία Παγίδων και Τερματισμός Παιχνιδιού

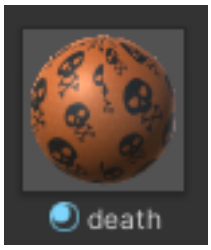
Δημιουργία Παγίδων (Traps)

Για να υλοποιήσουμε τις παγίδες θανάτου που εμφανίζονται σε τυχαίες θέσεις και χρονικές στιγμές, χρησιμοποιήσαμε το script **TrapSpawner**. Ας το αναλύσουμε:

1. **Prefab Παγίδας**:
   - Δημιουργήσαμε μια **σφαίρα**:



- **Scale = (50, 50, 50)** (λόγω διαμέτρου 5).
- Εφαρμόσαμε την υφή **death.jpg**:



- **Materials > Albedo** → Σύραμε την υφή πάνω στη σφαίρα.

- Αποθηκεύσαμε τη σφαίρα ως **Prefab**.

2. **Script: TrapSpawner.cs**
    - Το **TrapSpawner** αναλαμβάνει να εμφανίζει παγίδες σε **τυχαίες θέσεις** μέσα στο λαβύρινθο για **τυχαία χρονική διάρκεια**.

Πώς το πετυχαίνουμε στον κώδικα:

```
Vector3 GetValidPosition()
{
    Vector3 spawnPosition;
    float trapRadius = 50.0f; // Ακτίνα της παγίδας (λόγω διαμέτρου 5)

    // Επαναλαμβάνουμε μέχρι να βρούμε έγκυρη θέση
    do
    {
        float x = Random.Range(mazeBounds.min.x, mazeBounds.max.x);
        float z = Random.Range(mazeBounds.min.z, mazeBounds.max.z);
        spawnPosition = new Vector3(x, 50.0f, z);

    } while (Physics.CheckSphere(spawnPosition, trapRadius, wallLayer)); // Έλεγχος για σύγκρουση με τοίχο

    return spawnPosition;
}
```

- Η θέση επιλέγεται με **Random.Range** για τους άξονες `x` και `z`, ενώ το ύψος `y` είναι σταθερό:
- Για να **μην εμφανίζονται οι παγίδες πάνω σε τοίχους**, χρησιμοποιούμε το **Physics.CheckSphere**:
- Οι παγίδες καταστρέφονται μετά από τυχαίο χρονικό διάστημα

3. **Εκκίνηση της Ροής**:

```
IEnumerator SpawnTraps()
{
    while (true)
    {
        Vector3 spawnPosition = GetValidPosition();

        // Δημιουργία παγίδας
        GameObject trap = Instantiate(trapPrefab, spawnPosition, Quaternion.identity);

        // Καταστροφή της παγίδας μετά από τυχαία διάρκεια
        float trapLifetime = Random.Range(minLifetime, maxLifetime);
        Destroy(trap, trapLifetime);

        yield return new WaitForSeconds(spawnInterval);
    }
}
```

   - Η εμφάνιση των παγίδων ξεκινά στο **Start** μέσω του **`StartCoroutine(SpawnTraps())`**.

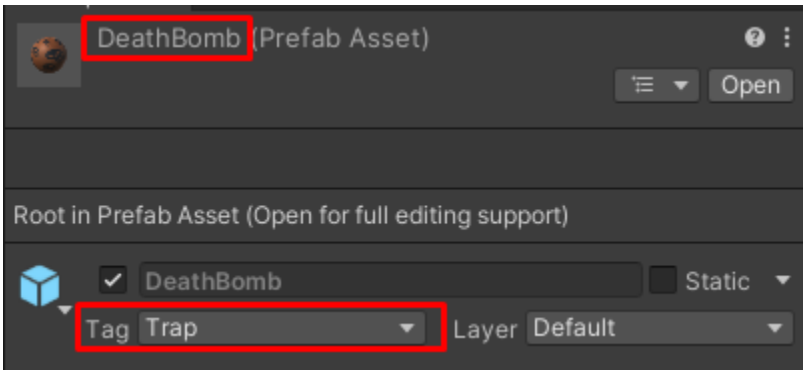
Ανίχνευση Σύγκρουσης με τον Bob

Για να τερματίζεται το παιχνίδι όταν ο Bob ακουμπήσει μια παγίδα, προσθέσαμε έλεγχο στο **BobMovement.cs**.

Πώς το πετυχαίνουμε στον κώδικα:

```
void OnTriggerEnter(Collider other)
{
    if (other.CompareTag("Trap"))
    {
        FindObjectOfType<GameOverManager>().TriggerGameOver();
    }
}
```

- Χρησιμοποιούμε το **`OnTriggerEnter`** για να ανιχνεύσουμε τη σύγκρουση:
- Το **Tag = "Trap"** έχει ανατεθεί στα **Prefabs** των παγίδων για να τα ξεχωρίζουμε.



Εμφάνιση του Game Over Panel

Όταν ο Bob ακουμπήσει μια παγίδα, ενεργοποιείται το **Game Over Panel** μέσω του script **GameOverManager**.

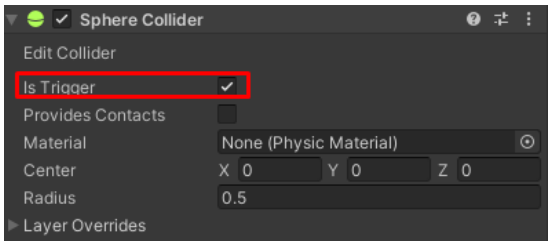
Πώς το πετυχαίνουμε στον κώδικα:

```
public void TriggerGameOver()
{
    Debug.Log("Game Over Triggered"); // Έλεγχος αν η μέθοδος καλείται
    if (gameOverPanel != null)
    {
        gameOverPanel.SetActive(true); // Ενεργοποίηση του Panel
        if (gameOverText != null)
        {
            gameOverText.text = "Game Over!";
        }
        Time.timeScale = 0; // Πάγωμα παιχνιδιού
    }
    else
    {
        Debug.LogError("Game Over Panel δεν έχει συνδεθεί στο Inspector!");
    }
}
```

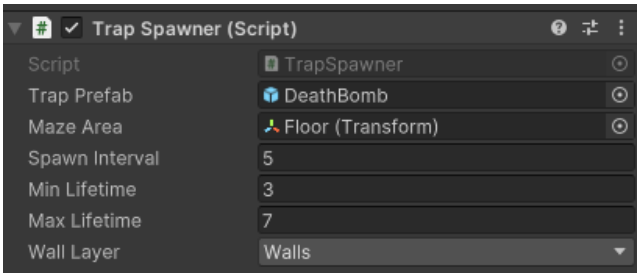
- Η μέθοδος **`TriggerGameOver`**:
    - Ενεργοποιεί το **UI Panel**:
    - Εμφανίζει το μήνυμα "Game Over" στο **Text**:
    - Παύει το παιχνίδι με **Time.timeScale = 0**.

Ρυθμίσεις στο Unity

1. **Trap Prefab**:
    - Προσθέσαμε στη σφαίρα:



        - **Sphere Collider** με **Is Trigger** ενεργοποιημένο.
        - **Tag = "Trap"**.
2. **BombTrapSpawner**:



- Δημιουργήσαμε ένα **Empty GameObject** με το όνομα **BombTrapSpawner**.
- Προσθέσαμε το script **TrapSpawner.cs**.
- Συνδέσαμε:
  - Το **trapPrefab** (Prefab παγίδας).
  - Το **mazeArea** (το Collider του λαβύρινθου).
  - Το **wallLayer** για αποφυγή τοίχων.

3. **Game Over UI**:

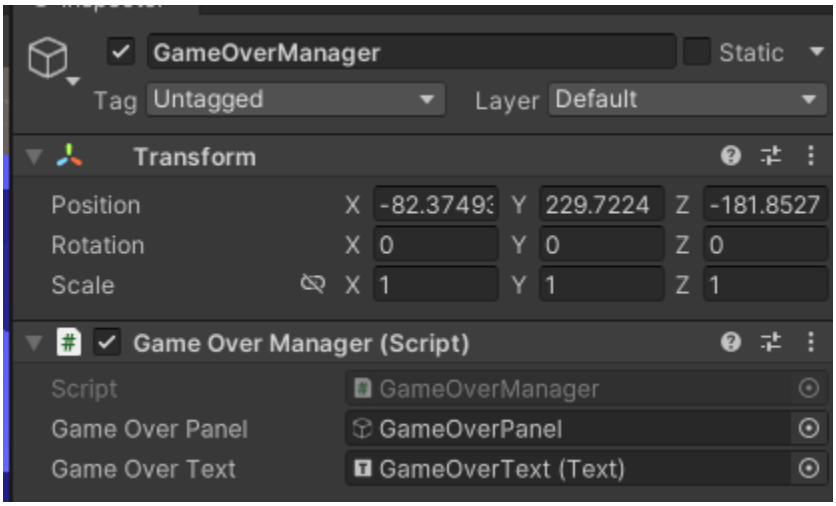
- Δημιουργήσαμε ένα **Canvas** με ένα **Panel** για το Game Over.
- Προσθέσαμε **Text** για το μήνυμα "Game Over!".
- Προσθέσαμε δύο **Buttons**:

```
public void RestartGame()
{
    Time.timeScale = 1; // Επαναφορά του χρόνου
    SceneManager.LoadScene(SceneManager.GetActiveScene().name); // Επαναφόρτωση σκηνής
}

public void ExitGame()
{
    Application.Quit(); // Έξοδος από το παιχνίδι
}
```

  - **Restart** → Καλεί τη μέθοδο `RestartGame`.
  - **Exit** → Καλεί τη μέθοδο `ExitGame`.

4. **GameOverManager**:



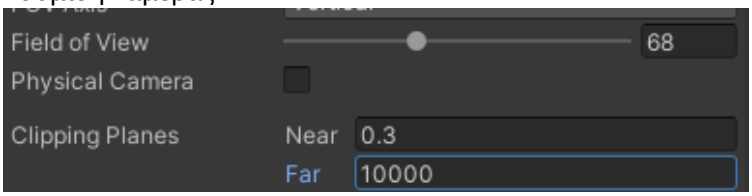
- Δημιουργήσαμε ένα **Empty GameObject** με το όνομα **GameOverManager**.
- Συνδέσαμε το **Game Over Panel** και το **Text** στο script.

Υλοποίηση Κάμερας με Ελεγχόμενη Κίνηση και Περιστροφή

Δημιουργία και Ρυθμίσεις της Κάμερας

Για να ικανοποιήσουμε την απαίτηση για κάμερα που μπορεί να μετακινηθεί και να περιστραφεί από τον χρήστη, δημιουργήσαμε το παρακάτω σύστημα:

1. **Προσθήκη Κάμερας στη Σκηνή**:
   - Από το **Hierarchy**, προσθέσαμε μια **Camera**:
     - **Right Click > Camera**.
   - Ονομάσαμε το αντικείμενο **"MainCamera"** για καλύτερη οργάνωση.
2. **Προσθήκη του Script**:
   - Δημιουργήσαμε το script **CameraController.cs** και το προσθέσαμε στην **MainCamera**.
3. Ρύθμιση κάμερας:



   - Στην MainCamera αλλάξαμε το **FoV** για καλύτερη οπτικοποίηση του συνολικού σχήματος άλλα και την τιμή του **FarClipping** Plane ώστε όταν η κάμερα απομακρύνεται από τον λαβύρινθο αυτός να μην εξαφανίζεται από το οπτικό πεδίο.

Λειτουργίες και Ρυθμίσεις της Κάμερας

Το script **CameraController** επιτρέπει στον χρήστη να ελέγχει την κάμερα με τα πλήκτρα:

- **Βελάκια**: Κίνηση στους άξονες `x` και `z`.
- **+ / -**: Κίνηση στον άξονα `y` (ύψος).
- **R**: Περιστροφή γύρω από τον εαυτό της στον `y` άξονα.

Ανάλυση Κώδικα

Μεταβλητές

```
public class CameraController : MonoBehaviour
{
    public float moveSpeed = 100.0f;      // Ταχύτητα κίνησης στους x και z άξονες
    public float heightSpeed = 50.0f;     // Ταχύτητα ανόδου/καθόδου στον y άξονα
    public float rotationSpeed = 50.0f; // Ταχύτητα περιστροφής γύρω από τον y άξονα
```

- **`moveSpeed`**: Καθορίζει την ταχύτητα κίνησης της κάμερας στους άξονες **x** και **z**.
- **`heightSpeed`**: Ελέγχει την ταχύτητα ανόδου/καθόδου στον **y** άξονα.
- **`rotationSpeed`**: Καθορίζει πόσο γρήγορα περιστρέφεται η κάμερα γύρω από τον `y` άξονα.

Update Method

```
void Update()
{
    HandleMovement(); // Κίνηση κάμερας
    HandleRotation(); // Περιστροφή κάμερας
}
```

Η **`Update()`** εκτελείται σε κάθε frame και καλεί δύο βασικές λειτουργίες:

- **HandleMovement**: Ελέγχει την κίνηση της κάμερας.
- **HandleRotation**: Ελέγχει την περιστροφή της κάμερας.

HandleMovement Method

```
void HandleMovement()
{
    // Κίνηση στους x και z άξονες με τα βελάκια
    float horizontal = Input.GetAxis("Horizontal"); // Αριστερά/Δεξιά
    float vertical = Input.GetAxis("Vertical");     // Μπροστά/Πίσω

    Vector3 moveDirection = new Vector3(horizontal, 0, vertical).normalized;
    transform.Translate(moveDirection * moveSpeed * Time.deltaTime, Space.World);

    // Κίνηση στον y άξονα με τα πλήκτρα + και -
    if (Input.GetKey(KeyCode.KeypadPlus) || Input.GetKey(KeyCode.Equals)) // Πλήκτρο +
    {
        transform.position += Vector3.up * heightSpeed * Time.deltaTime;
    }
    if (Input.GetKey(KeyCode.KeypadMinus) || Input.GetKey(KeyCode.Minus)) // Πλήκτρο -
    {
        transform.position += Vector3.down * heightSpeed * Time.deltaTime;
    }
}
```

1. **Κίνηση στους άξονες `x` και `z`**:

- Χρησιμοποιούμε τις εντολές **`Input.GetAxis`** για να διαβάσουμε τα βελάκια του πληκτρολογίου:
    - **"Horizontal"**: Ελέγχει κίνηση στον άξονα **x** (αριστερά/δεξιά).
    - **"Vertical"**: Ελέγχει κίνηση στον άξονα **z** (μπροστά/πίσω).
- Με την **`transform.Translate`**, μετακινούμε την κάμερα στην επιθυμητή κατεύθυνση

2. **Κίνηση στον `y` άξονα (αλλαγή ύψους)**:

- Με τα πλήκτρα **+** και **-**:
    - **`Vector3.up`**: Κίνηση προς τα πάνω.
    - **`Vector3.down`**: Κίνηση προς τα κάτω.
- Προσθέτουμε την κίνηση στην **`transform.position`** για να αλλάξουμε το ύψος της κάμερας.

HandleRotation Method

Περιστροφή γύρω από τον εαυτό της:

```
void HandleRotation()
{
    // Περιστροφή γύρω από τον εαυτό της στον y άξονα με το πλήκτρο R
    if (Input.GetKey(KeyCode.R))
    {
        transform.Rotate(Vector3.up, rotationSpeed * Time.deltaTime, Space.Self);
    }
}
```

- Χρησιμοποιούμε την **`transform.Rotate`** για περιστροφή γύρω από τον `y` άξονα.
- Η παράμετρος **`Space.Self`** διασφαλίζει ότι η περιστροφή γίνεται τοπικά (γύρω από το κέντρο της κάμερας).
- Το πλήκτρο **`R`** ενεργοποιεί αυτήν την περιστροφή.

Οπτικοποίηση και Σχέσεις στο Unity

- **Προσθήκη του Script**:

  - Σύραμε το script **CameraController.cs** στο αντικείμενο **MainCamera**.

- **Ρυθμίσεις στο Inspector**:

  - Προσαρμόσαμε τις ταχύτητες:
    - **Move Speed**: `100`.
    - **Height Speed**: `50`.
    - **Rotation Speed**: `50`.

- **Έλεγχος Κίνησης**:

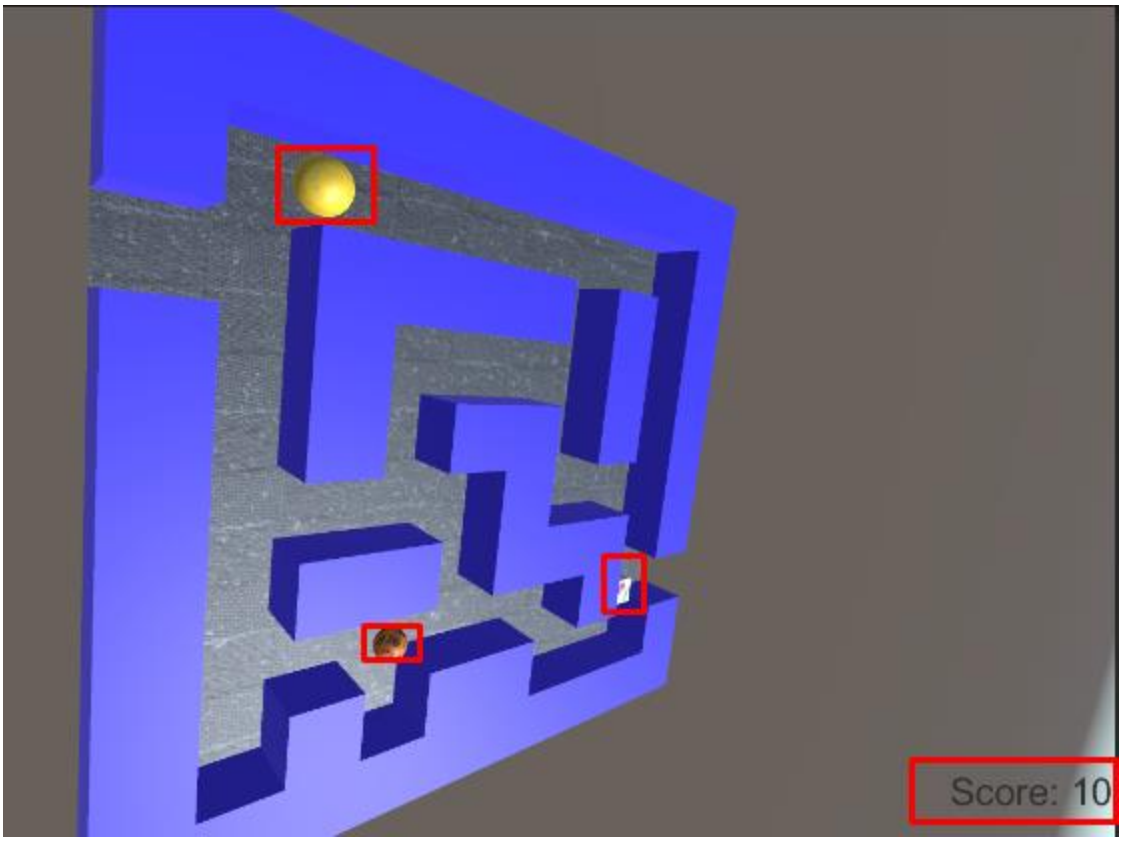
  - Πατώντας **Βελάκια**, η κάμερα κινείται στους **x** και **z** άξονες.
  - Με + / -, η κάμερα αλλάζει ύψος στον **y** άξονα.
  - Με **R**, η κάμερα περιστρέφεται γύρω από τον εαυτό της στον **y** άξονα.

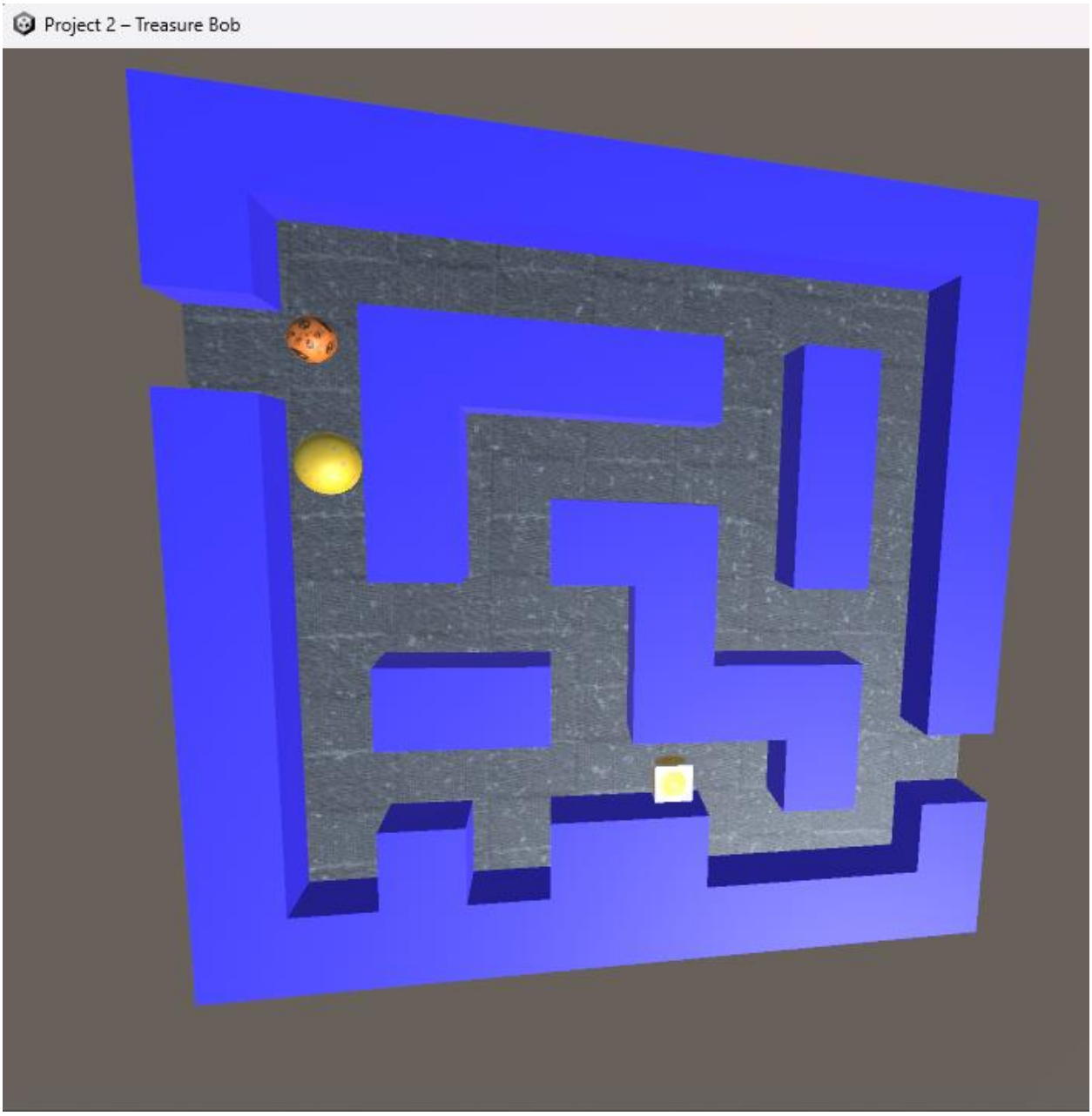
Συμπεράσματα Υλοποίησης

Στο πλαίσιο της ανάπτυξης του παιχνιδιού **"Treasure Bob"**, επιτύχαμε πλήρως όλους τους βασικούς και προαιρετικούς στόχους που τέθηκαν, συμπεριλαμβανομένων ***όλων*** των **bonus ερωτημάτων**. Η συνολική λειτουργικότητα του παιχνιδιού υλοποιήθηκε με προσεγμένο σχεδιασμό και κώδικα που επιτρέπει τη δυναμική αλληλεπίδραση του παίκτη με τον κόσμο του παιχνιδιού.

Σημείωση: Στο εκτελέσιμο αρχείο για κάποιο λόγο δεν εμφανίζεται το score κάτω δεξιά ενώ στο περιβάλλον ανάπτυξης Unity φαίνεται κανονικά όπως θα δείτε και στα παρακάτω στιγμιότυπα.

Στιγμιότυπα Εκτέλεσης Παιχνιδιού


Score: 10



Πληροφορίες σχετικά με την υλοποίηση

- **Λειτουργικό Σύστημα:**

  Windows 11 x64

- **Έκδοση Unity:**

  2022.3.15f1

- **Φάκελος εκτελέσιμου:**

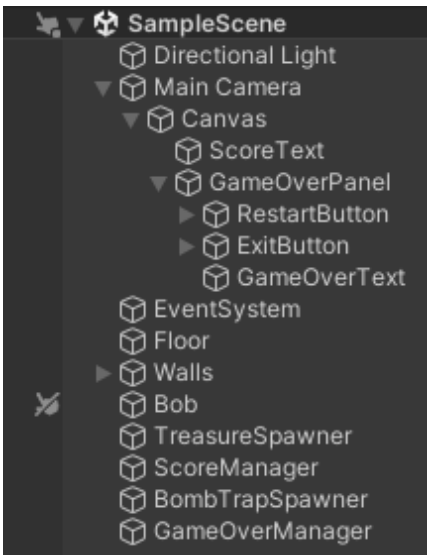
  Project 2 – Treasure Bob\Treasure Bob builder

• **Βιβλιοθήκη:**

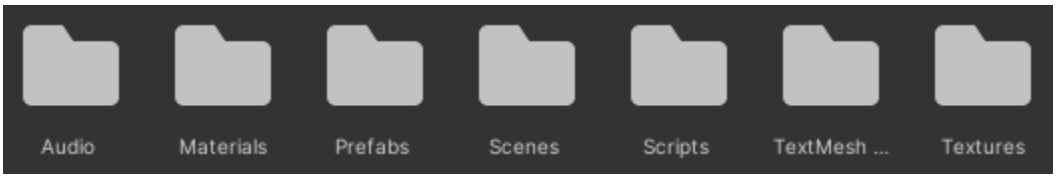
Εφέ (Particle Systems) ,

UI (π.χ. TextMesh Pro).

• **Ιεραρχία:**

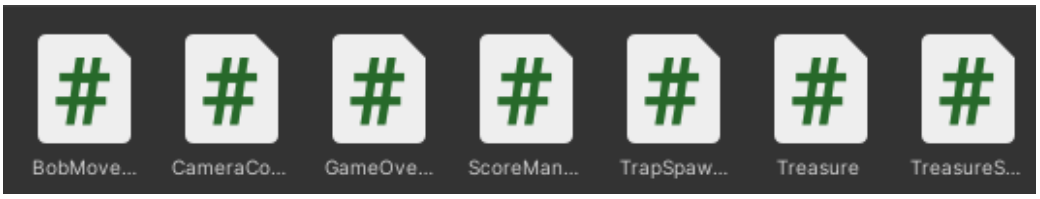


• **Συμπεριλαμβανόμενα αρχεία**:

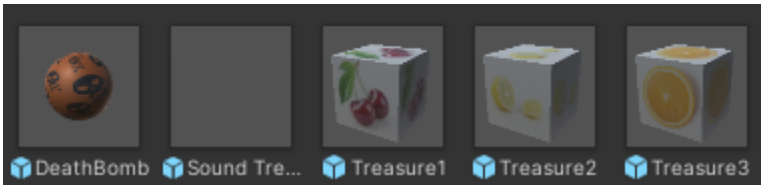


1. Textures

2. Scripts

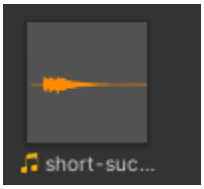


3. Prefabs



4. Materials

5. Audio



- **md5 checksum:**

00AB671C14FF95224F6C10959D5948ED

- **Project link- Drive:**

  https://drive.google.com/drive/folders/1_oF8E9_zUJnt0x7lz211M7tnzKCRPueS?usp=sharing

- **Αναφορές:**

  Όλες οι αναφορές σε πηγές βρίσκονται πάνω στην περιγραφή κάθε κεφαλαίου με bold με την χρήση υπερ-σύνδεσης με λινκ.