Πανεπιστήμιο Ιωαννίνων
Τμήμα Μηχ. Η/Υ και Πληροφορικής Χειμερινό Εξάμηνο 2024

ΜΥΥ702

Γραφικά Υπολογιστών & Συστήματα Αλληλεπίδρασης

Διδάσκων: Ιωάννης Φούντος

Υπεύθυνη εργαστηριακού μέρους μαθήματος: Βασιλική Σταμάτη

Προγραμματιστική Άσκηση 2

Unity 3D 2024-2025

Αναφορά

Παναγιώτης Παρασκευόπουλος

ΑΜ:2905

Περιεχόμενα

Περιγραφή της εργασίας 3
Ερώτημα (i) 4
Ερώτημα (ii) 5
Ερώτημα (iii) 6
Ερώτημα (iv) 9
Ερώτημα (v) 11
Bonus ερωτήματα 12
Bonus (a) 12
Bonus (b) 13
Bonus (c) 14
Bonus (d) 15
Extra δικές μας προσθήκες 16
1) 16
2) 16
3) 18
Τέλος 19
Περιγραφή δυσκολιών υλοποίησης -προβλήματα που συναντήθηκαν 20
Αναφορές – Πηγές που χρησιμοποιήθηκαν κατά την εκπόνηση της εργασίας. 21

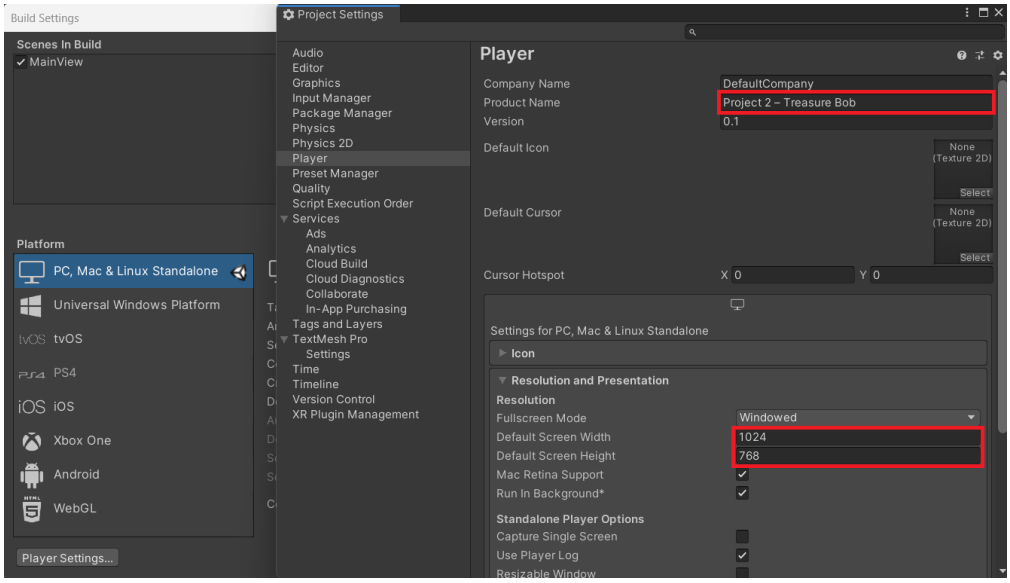
Περιγραφή της εργασίας

Η συγκεκριμένη αναφορά βασίζεται στην προγραμματιστική άσκηση Unity 3D. Σκοπός αυτής της άσκησης είναι να εξοικειωθούμε με την χρήση πλατφορμών γραφικών όπως η Unity3D. Στην άσκηση αυτή θα κατασκευάσουμε μία εφαρμογή - παιχνίδι σε Unity3D όπου ένας χαρακτήρας, ο Treasure Bob, μετακινείται μέσα σε έναν λαβύρινθο και μαζεύει θησαυρούς.

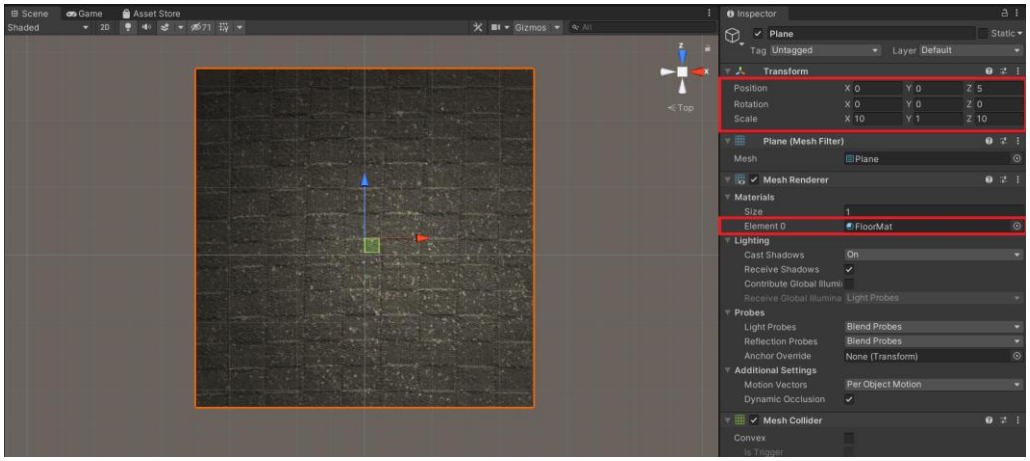
Η άσκηση αυτή έγινε από ένα άτομο. Στην αναφορά χρησιμοποιώ α’ πληθυντικό για καλύτερη ανάγνωση.

Ερώτημα (i)

Φτιάχνουμε μια εφαρμογή Unity 3D που θα τρέχει με ανάλυση 1024x768 και θα έχει τίτλο «Project 2 – Treasure Bob».

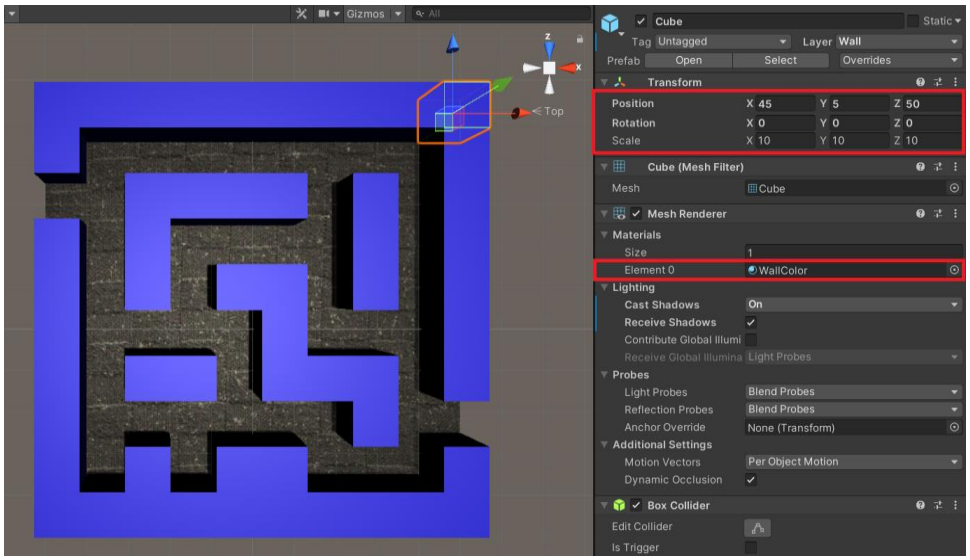


Όταν θα ξεκινάει η εφαρμογή θα φορτώνεται κατ' αρχάς ο λαβύρινθος: Φτιάχνουμε για αρχή ένα plane. Του ορίζουμε τη θέση του κέντρου σε (0, 0, 5). Στον Inspector αλλάζουμε το scale του σε (10, 1, 10) ώστε να καλύπτει περιοχή 100x100 τετραγωνάκια, σχεδιασμένο στο xz επίπεδο. Εισάγουμε το αρχείο floor.jpg στον φάκελο μας στην εφαρμογή μας. Δημιουργούμε ένα material (FloorMat), του δίνουμε την υφή floor και το εφαρμόζουμε στο plane.



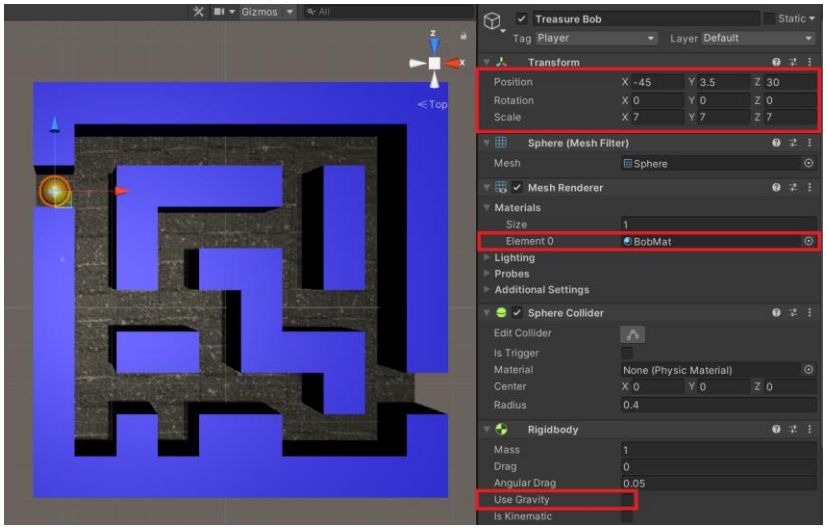
Τώρα θέλουμε να δημιουργήσουμε τον λαβύρινθο που θα ακουμπάει πάνω σε αυτό. Δημιουργούμε ένα 3D object > Cube. Ορίζουμε τις διαστάσεις του σε (10, 10, 10). Δημιουργούμε ένα material (WallColor) στο οποίο δίνουμε το χρώμα μπλε (RGB:0, 0, 255) και το εφαρμόζουμε στον κύβο. Δημιουργούμε ένα Prefab του κύβου για να τον χρησιμοποιήσουμε πολλές φορές και να σχηματίσουμε τον τοίχο του λαβύρινθου. Τοποθετούμε τον κάθε κύβο στις σωστές συντεταγμένες του. Π.χ. το κέντρο του κύβου

στην πάνω δεξιά γωνία έχει συντεταγμένες (45, 5, 50), επειδή οι διαστάσεις του είναι (10,10,10), έτσι ώστε να πατάει πάνω στον επίπεδο και επειδή το κέντρο του λαβύρινθου είναι(0, 0, 5).



Ερώτημα (ii)

Στο ερώτημα αυτό υλοποιούμε τον Treasure Bob και την κίνηση του μέσα στον λαβύρινθο με τα πλήκτρα <j, l, i, k>.
Δημιουργούμε ένα 3D object > Sphere την οποία ονομάζουμε “Treasure Bob”. Ορίζουμε τις διαστάσεις του σε (7, 7, 7) και τον τοποθετούμε στη θέση (-45, 3.5, 30) ώστε να βρίσκεται στη είσοδο του λαβύρινθου και να πατάει πάνω στο επίπεδο. Του δίνουμε την υφή bob με τον ίδιο τρόπο που δώσαμε την υφή floor στο plane. Προσθέτουμε Rigidbody στη σφαίρα και καταργούμε την βαρύτητα.



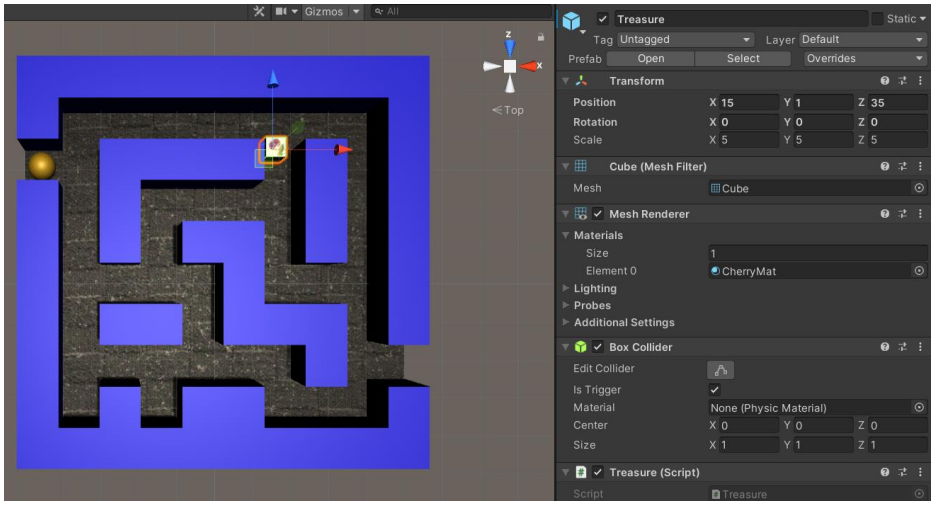
Τώρα δημιουργούμε ένα script BobController για την κίνηση του Bob.
Φτιάχνουμε την κλάση `public class BobController : MonoBehaviour`
Ορίζουμε τα όρια του plane
```
    private float minX = -50f;
    private float maxX = 50f;
```
Στην Update function ορίζουμε την κίνηση του Bob στους x και z άξονες με τα πλήκτρα <j, l ,i ,k> και υπολογίζουμε την νέα θέση του. Τα speedLevels είναι η ταχύτητα με την οποία κινείται ο Bob τα οποία θα τα αναφέρουμε στη συνέχεια που υλοποιούμε το bonus ερώτημα (b). Περιορίζουμε και την θέση του, να μην μπορεί να κινηθεί έξω από το plane

```
// Λήψη εισόδου από τον χρήστη
float moveX = 0f;
float moveZ = 0f;

if (Input.GetKey(KeyCode.J)) moveX = -1f;
if (Input.GetKey(KeyCode.L)) moveX = 1f;
if (Input.GetKey(KeyCode.I)) moveZ = 1f;
if (Input.GetKey(KeyCode.K)) moveZ = -1f;

// Υπολογισμός νέας θέσης
Vector3 moveDirection = new Vector3(moveX, 0, moveZ).normalized;
Vector3 newPosition = transform.position + moveDirection * speedLevels[currentSpeedIndex] * Time.deltaTime;

// Περιορισμός της θέσης εντός του plane
newPosition.x = Mathf.Clamp(newPosition.x, minX, maxX);

// Έλεγχος για τοίχους
if (!Physics.CheckSphere(newPosition, 3.5f, wallLayer))
{
    transform.position = newPosition;
}
```

Επίσης, έχουμε αντιστοιχίσει στην εφαρμογή όλους τους κύβους που σχηματίζουν τον τοίχο του λαβύρινθου το Layer ”Wall” και ελέγχουμε μέσα στην BobController να μην περνάει μέσα από τοίχους. `public LayerMask wallLayer; // Για ανίχνευση τοίχων`

Ερώτημα (iii)

Τώρα θέλουμε μέσα στον λαβύρινθο εμφανίζονται «θησαυροί» που ο χαρακτήρας προσπαθεί να «πιάσει».
Δημιουργούμε ένα 3D Object > Cube και το ονομάζουμε "Treasure". Ορίζουμε τις διαστάσεις του κύβου σε (5, 5, 5) και τον τοποθετούμε σε μια τυχαία θέση.
Εφαρμόζουμε ένα από τα τρία materials (CherryMat, OrangeMat, LemonMat) τα οποία δημιουργήσαμε με τον γνωστό τρόπο. Δημιουργούμε ένα Prefab του θησαυρού και τον σβήνουμε από τη σκηνή.



Φτιάχνουμε το script TreasureManager για την λειτουργία του θησαυρού και το τοποθετούμε σε ένα άδειο GameObject. Μέσα στο script:

Καλούμε την SpawnTreasure για να δημιουργήσει τον πρώτο θησαυρό

```
28      void Start()
29      {
30          SpawnTreasure();
31      }
```

Λειτουργία της SpawnTreasure:

Αν υπάρχει ήδη θησαυρός, τον καταστρέφει.

```
35      if (currentTreasure != null)
36      {
37          Destroy(currentTreasure);
38      }
```

Μέσα στο loop επιλέγει μια τυχαία θέση μέσα στα όρια του λαβύρινθου και ελέγχει να μην είναι μέσα σε τοίχο και να είναι μακριά από τη θέση του Bob.

```
45      do
46      {
47          // Επιλογή τυχαίας θέσης
48          randomPosition = new Vector3(
49              Random.Range(-mazeBounds.x, mazeBounds.x),
50              2.5f,
51              Random.Range(-mazeBounds.z, mazeBounds.z)
52          );
53
54          // Έλεγχος αν η θέση βρίσκεται μέσα σε τοίχο
55          bool notInWall = !Physics.CheckSphere(randomPosition, 2.5f, LayerMask.GetMask("Wall"));
56
57          // Έλεγχος απόστασης από τον Bob
58          bool farFromBob = Vector3.Distance(randomPosition, treasureBob.transform.position) > minDistanceFromBob;
59
60          isValidPosition = notInWall && farFromBob;
61          attempts++;
62      }
63      while (!isValidPosition && attempts < 100); // Περιορισμός προσπαθειών για αποφυγή infinite loop
```

Αν η θέση είναι κατάλληλη, δημιουργεί τον θησαυρό με ένα τυχαίο material από τα τρία και θέτει χρονικό όριο με χρήση Invoke.

```
65      if (isValidPosition)
66      {
67          // Δημιουργία νέου θησαυρού
68          currentTreasure = Instantiate(treasurePrefab, randomPosition, Quaternion.identity);
69          currentTreasure.GetComponent<Renderer>().material = treasureMaterials[Random.Range(0, treasureMaterials.Length)];
70
71          // Καταστροφή θησαυρού μετά από συγκεκριμένο χρόνο
72          Invoke(nameof(CheckAndRespawnTreasure), treasureLifetime);
73      }
```

Η μέθοδος TreasureCollected καλείται όταν ο θησαυρός συλλεχθεί.

```
80      public void TreasureCollected()
81      {
82          CancelInvoke(nameof(CheckAndRespawnTreasure));
83          SpawnTreasure();
84      }
```

Ακυρώνει το χρονοδιακόπτη για καταστροφή του θησαυρού και δημιουργεί έναν νέο. Η μέθοδος CheckAndrespawnTreasure ελέγχει αν ο θησαυρός δεν συλλέχθηκε, τον καταστρέφει και δημιουργεί νέο.

```
86      private void CheckAndRespawnTreasure()
87      {
88          if (currentTreasure != null) Destroy(currentTreasure);
89          SpawnTreasure();
90      }
```

Με αυτές τις υλοποιήσεις εξασφαλίζουμε ότι ο θησαυρός εξακολουθεί να εμφανίζεται περιοδικά με τη χρήση του Invoke στη SpawnTreasure αλλά και ότι όταν συλλέγεται ένας θησαυρός, ο TreasureManager καλείται να δημιουργήσει νέο θησαυρό αμέσως.

Στο gameObject "TreasureManager", προσθέτουμε το prefab του θησαυρού στο πεδίο Treasure Prefab και τα τρία Materials στο πεδίο Treasure Materials.

Στη συνέχεια πρέπει να υλοποιήσουμε την λειτουργία του θησαυρού σε αλληλεπίδραση με τον Bob. Δημιουργούμε το script "Treasure" και το προσθέτουμε στο prefab του θησαυρού.

Έχουμε ορίσει την μεταβλητή `5  private bool isBeingCollected = false;` για να αποτρέψουμε την συνεχόμενη πολλαπλή συλλογή του ίδιου θησαυρού. Έχουμε δώσει στον Treasure Bob το tag "Player" και δημιουργούμε την μέθοδο OnTriggerEnter.

```
15      void OnTriggerEnter(Collider other)
16      {
17          if (!isBeingCollected && other.CompareTag("Player"))
18          {
19              isBeingCollected = true;
20              StartCoroutine(CollectTreasure());
21          }
22      }
```

Ελέγχει αν το αντικείμενο που συγκρούεται είναι ο Treasure Bob και αν ο θησαυρός δεν έχει συλλεχθεί και ενεργοποιεί την διαδικασία CollectTreasure.

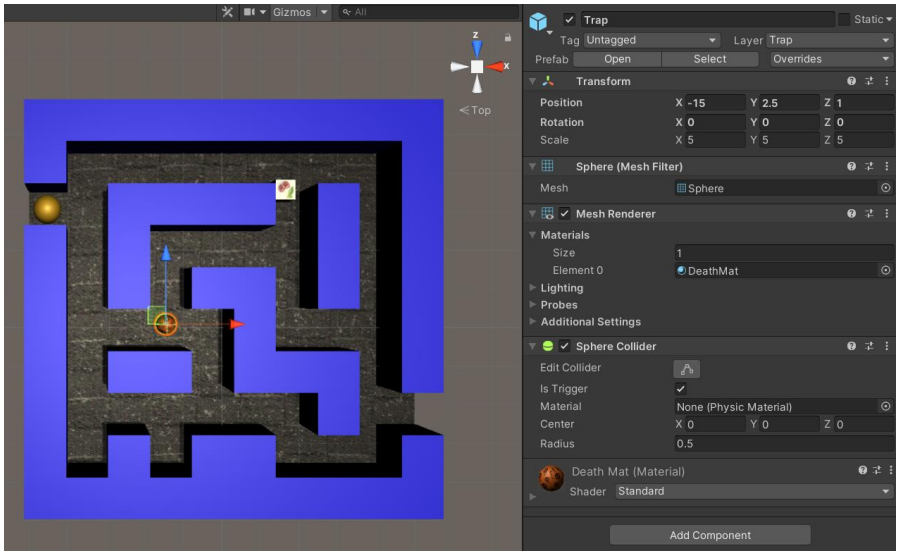
Στην CollectTreasure υλοποιούμε την συρρίκνωση.

```
65          float shrinkSpeed = 5f;
66          while (transform.localScale.x > 0.1f)
67          {
68              transform.localScale -= Vector3.one * shrinkSpeed * Time.deltaTime;
69              yield return null;
70          }
71          Destroy(gameObject); // Καταστροφή θησαυρού
72          TreasureManager.Instance.TreasureCollected();
73      }
```

Ορίζουμε την ταχύτητα συρρίκνωσης σε 5. Μειώνουμε το μέγεθος σταδιακά μέχρι να φτάσει 0.1 και σταματάμε τη διαδικασία. Μετά καταστρέφουμε τον θησαυρό και καλούμε την TreasureCollected για να δημιουργήσει έναν νέο.

Ερώτημα (iv)

Θέλουμε επίσης εκτός από τους θησαυρούς, μέσα στον λαβύρινθο να εμφανίζονται σε τυχαία χρονική στιγμή και για τυχαία διάρκεια αντικείμενα «παγίδες» που όταν τα ακουμπήσει ο Bob, «πεθαίνει» και τερματίζει το παιχνίδι.
Δημιουργούμε ένα 3D Object > Sphere με το όνομα "Trap". Ορίζουμε τις διαστάσεις σε (5, 5, 5) και εφαρμόζουμε το material death. Μετατρέπουμε το αντικείμενο σε Prefab.



Δημιουργούμε το script TrapManager για την διαχείριση των παγίδων.
Ορίζουμε τα όρια του λαβύρινθου που θα μπορεί να εμφανιστεί η παγίδα, τους χρόνους για την εμφάνιση της παγίδας και την διάρκεια ζωής.

```
6       public Vector3 mazeBounds; // Op
7       public float minSpawnTime = 5f;
8       public float maxSpawnTime = 15f;
9       public float minDuration = 5f; /
10      public float maxDuration = 10f;
```

Δημιουργούμε την πρώτη παγίδα

```
14      void Start()
15      {
16          ScheduleNextTrap();
17      }
```

Στην μέθοδο ScheduleNextTrap υπολογίζεται τυχαίος χρόνος ανάμεσα στον min και max SpawnTime που ορίσαμε, και καλούμε την SpawnTrap.

```
19      void ScheduleNextTrap()
20      {
21          float spawnTime = Random.Range(minSpawnTime, maxSpawnTime);
22          Invoke(nameof(SpawnTrap), spawnTime);
23      }
```

Η υλοποίηση της SpawnTrap ξεκινάει με τον έλεγχο αν υπάρχει ήδη παγίδα στον λαβύρινθο. Αν υπάρχει, την καταστρέφει.

```
25      void SpawnTrap()
26      {
27          if (currentTrap != null)
28          {
29              Destroy(currentTrap);
30          }
```

Το loop είναι ίδιας λογικής με του θησαυρού, επιλέγει μια τυχαία θέση μέσα στα όρια του λαβύρινθου και ελέγχει να μην είναι μέσα σε τοίχο και να είναι μακριά από τη θέση του Bob.

```
do
{
    // Επιλέγουμε τυχαία θέση
    randomPosition = new Vector3(
        Random.Range(-mazeBounds.x, mazeBounds.x),
        2.5f, // Ύψος της παγίδας
        Random.Range(-mazeBounds.z, mazeBounds.z)
    );

    // Έλεγχος αν η θέση βρίσκεται μέσα σε τοίχο
    bool notInWall = !Physics.CheckSphere(randomPosition, 2.5f, LayerMask.GetMask("Wall"));

    // Έλεγχος απόστασης από τον Bob
    bool farFromBob = Vector3.Distance(randomPosition, treasureBob.transform.position) > minDistanceFromBob;

    isValidPosition = notInWall && farFromBob;
    attempts++;
}
while (!isValidPosition && attempts < 100);
```

Αν η βρεθεί κατάλληλη θέση, δημιουργεί μια παγίδα με έναν τυχαίο χρόνο διάρκειας ζωής μέσα στα όρια του min και max Duration.

```
if (isValidPosition)
{
    // Δημιουργούμε την παγίδα
    currentTrap = Instantiate(trapPrefab, randomPosition, Quaternion.identity);

    // Προγραμματίζουμε την καταστροφή της μετά από τυχαία διάρκεια
    float duration = Random.Range(minDuration, maxDuration);
    Destroy(currentTrap, duration);

    // Προγραμματίζουμε την επόμενη παγίδα
    ScheduleNextTrap();
}
```

Μετά, καταστρέφουμε την τρέχουσα παγίδα και καλούμε ξανά το ScheduleNextTrap για να ξεκινήσει η διαδικασία από την αρχή.

Υλοποιούμε και εφαρμόζουμε στον Treasure Bob το script TreasureBob, για τον θάνατο του και τον τερματισμό του παιχνιδιού όταν αυτός πέσει σε μια παγίδα.
Ορίζουμε την κατάσταση του Bob `private bool isDead = false;`. Έχουμε ορίσει στην εφαρμογή το layer "Trap" στο Prefab της παγίδας. Υλοποιούμε την μέθοδο OnTriggerEnter, η οποία εκτελείται όταν ο Treasure Bob πέσει πάνω στην παγίδα και όταν η κατάσταση του Bob είναι ”ζωντανος”

```
private void OnTriggerEnter(Collider other)
{
    if (other.gameObject.layer == LayerMask.NameToLayer("Trap") && !isDead)
    {
```

και εκτελεί την μέθοδο Die. `Die();` Στην μέθοδο Die δηλώνουμε τον Treasure Bob ως ”νεκρό”

```
void Die()
{
    isDead = true;
    Debug.Log("Treasure Bob has died!");
```

και στο τέλος τερματίζουμε το παιχνίδι.

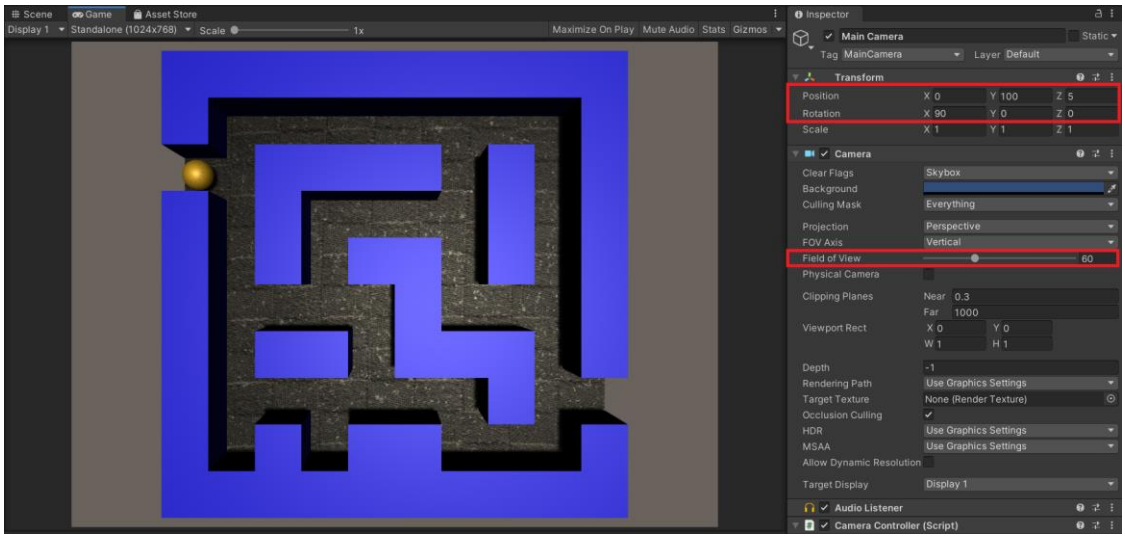
```
#if UNITY_EDITOR
        UnityEditor.EditorApplication.isPlaying = false; //
#else
        Application.Quit(); // Πλήρης τερματισμός στο Build
#endif
    }
}
```

(Επειδή έγιναν προσθήκες στη συνέχεια, το παιχνίδι δεν τερματίζει κατευθείαν μόλις εκτελεστεί η μέθοδος Die.)

Ερώτημα (v)

Στο ερώτημα αυτό θα υλοποιήσουμε μια απλή κάμερα ώστε ο χρήστης να βλέπει τη σκηνή από οποιαδήποτε γωνία και θέση και από οποιοδήποτε ύψος.

Έχουμε ορίσει αρχικά την Main Camera στην θέση (0, 100, 5) και με περιστροφή (90, 0, 0), ώστε να βρίσκεται στον y άξονα και να κοιτάει στο κέντρο του λαβύρινθου από ύψος 100. Έχουμε ορίσει το FOV σε 60 ώστε να κοιτάει τη σκηνή με ολόκληρο τον λαβύρινθο (όπως στην Εικόνα 2 της εκφώνησης).



Δημιουργούμε το script CameraController και το προσθέτουμε στην Main Camera.
Αρχικά, ορίζουμε την ταχύτητα κίνησης και ταχύτητα περιστροφής της κάμερας

```
5       public float moveSpeed = 10f; //
6       public float rotateSpeed = 100f;
```

, τα οποία αλλάζουν και μέσα από την εφαρμογή
Μέσα στην Update μέθοδο, διαβάζουμε την είσοδο από τον χρήστη

```
11          float moveX = Input.GetAxis("Horizontal") * moveSpeed * Time.deltaTime;
12          float moveZ = Input.GetAxis("Vertical") * moveSpeed * Time.deltaTime;
```

Χρησιμοποιούμε το Input.GetAxis για να λαμβάνονται οι τιμές από τα βελάκια ή από τα πλήκτρα A/D (για τον άξονα x) και W/S (για τον άξονα z).
Αν ο χρήστης πατήσει το πλήκτρο '+' ή '-' αυξάνεται ή μειώνεται το ύψος της κάμερας στον y άξονα.

```
15          float moveY = 0f;
16          if (Input.GetKey(KeyCode.KeypadPlus) || Input.GetKey(KeyCode.PageUp))
17          {
18              moveY = moveSpeed * Time.deltaTime;
19          }
20          else if (Input.GetKey(KeyCode.KeypadMinus) || Input.GetKey(KeyCode.PageDown))
21          {
22              moveY = -moveSpeed * Time.deltaTime;
23          }
```

(Μπορεί να γίνει και με τα πλήκτρα PageUp/PageDown)

Εφαρμόζουμε τη συνολική κίνηση στο transform.position μέσω προσθήκης των διανυσμάτων.
```
26          transform.position += new Vector3(moveX, moveY, moveZ);
```

Με το πλήκτρο R περιστρέφουμε (clockwise) την κάμερα γύρω από τον άξονα y της κάμερας με κέντρο τον εαυτό της.

```
28          // Περιστροφή (clockwise) γύρω από τον άξονα y με το πλήκτρο R
29          if (Input.GetKey(KeyCode.R))
30          {
31              float rotationY = rotateSpeed * Time.deltaTime;
32              transform.Rotate(0, rotationY, 0, Space.Self);
33          }
34          // Περιστροφή (counterclockwise) γύρω από τον άξονα y με το πλήκτρο E
35          if (Input.GetKey(KeyCode.E))
36          {
37              float rotationY = rotateSpeed * Time.deltaTime;
38              transform.Rotate(0, -rotationY, 0, Space.Self);
39          }
```

Με το πλήκτρο E την περιστρέφουμε (counterclockwise).

Bonus ερωτήματα

Bonus (a)

Εδώ θα προσθέσουμε εφέ και ήχο όταν ακουμπάει ο παίχτης τους θησαυρούς.
Για αρχή, στην Unity φτιάχνουμε ένα Particle System (CollectEffect) με εφέ σύννεφα σκόνης και το αποθηκεύουμε ως Prefab. Μέσα στο script Treasure κάνουμε αναφορά στο prefab του Particle System. `8  public GameObject collectEffectPrefab;` Στην μέθοδο CollectTreasure δημιουργούμε το particle effect στη θέση του θησαυρού, το οποίο καταστρέφουμε μετά από δύο δευτερόλεπτα.

```
51          if (collectEffectPrefab != null)
52          {
53              // Δημιουργία του Particle System στην θέση του θησαυρού
54              GameObject particleEffect = Instantiate(collectEffectPrefab, transform.position, Quaternion.identity);
55
56              // Περιμένουμε μέχρι να ολοκληρωθεί το particle effect
57              float effectDuration = 2.0f; // Διάρκεια
58              Destroy(particleEffect, effectDuration); // Καταστρέφει το particle effect μετά την διάρκεια
59          }
```

Για τον ήχο, περνάμε στην εφαρμογή μας ένα αρχείο ήχου (coinSound) και στο GameObject Treasure προσθέτουμε ένα AudioSource component στο οποίο απενεργοποιούμε την επιλογή Play On Awake καθώς θέλουμε να παίζεται μόνο όταν συλλέγεται ο θησαυρός. Στο script Treasure ορίζουμε τον ήχο που θα παίξει κατά τη

συλλογή του θησαυρού και το AudioSource.
```
6       public AudioClip collectSound; /
7       private AudioSource audioSource;
```
Αποθηκεύουμε το AudioSource component που πρέπει να υπάρχει στο αντικείμενο.
```
10      void Start()
11      {
12          audioSource = GetComponent<AudioSource>();
13      }
```
και μέσα στην CollectTreasure μέθοδο βάζουμε να παίζει ο ήχος όταν συλλεχθεί ο θησαυρός.
```
45      if (audioSource != null && collectSound != null)
46      {
47          audioSource.PlayOneShot(collectSound);
48      }
```
Στο πεδίο Collect Sound βάζουμε το αρχείο coinSound.

Bonus (b)

Για να προσθέσουμε πλήκτρα για την αυξομείωση της ταχύτητας με την οποία κινείται ο Bob, θα κάνουμε την εξής τροποποίηση στο script BobController που ελέγχει την κίνηση του Bob.
Ορίζουμε τις 5 διαβαθμίσεις ταχύτητας και ένα πεδίο για την τρέχουσα ταχύτητα του Bob με την οποία ξεκινάει κιόλας.
```
5       public float[] speedLevels = { 4f, 8f, 16f, 32f, 64f };
6       private int currentSpeedIndex = 2; // Ξεκινάμε από τη μ
```
Μέσα στην Update θα ορίσουμε τα πλήκτρα ’O’ και ’U’ επειδή είναι τα πιο προσβάσιμα στα πλήκτρα κίνησης του πληκτρολογίου.
```
16      if (Input.GetKeyDown(KeyCode.O)) // Αύξηση ταχύτητας
17      {
18          if (currentSpeedIndex < speedLevels.Length - 1)
19              currentSpeedIndex++;
20      }
21      if (Input.GetKeyDown(KeyCode.U)) // Μείωση ταχύτητας
22      {
23          if (currentSpeedIndex > 0)
24              currentSpeedIndex--;
25      }
```
Αν πατηθεί το πλήκτρο ’O’ αυξάνεται η τρέχουσα ταχύτητα, ενώ με το ’U’ μειώνεται.
Προσθέτουμε και την συνάρτηση OnGUI στο τέλος για να εμφανίζεται στην οθόνη και να βλέπει ο χρήστης την τρέχουσα ταχύτητα του Bob.
```
51      void OnGUI()
52      {
53          GUI.Label(new Rect(10, 10, 200, 20), "Speed: " + speedLevels[currentSpeedIndex]);
54      }
```

Bonus (c)

Εδώ, θα φτιάξουμε σκορ σύστημα, ώστε ο παίχτης να μαζεύει βαθμούς ανάλογα με τον θησαυρό που συλλέγει.
Υλοποιούμε το script ScoreManager και δημιουργούμε ένα Text αντικείμενο (ScoreBoard), τα οποία προσθέτουμε σε ένα GameObject. Μέσα στο script ορίζουμε το Text για το UI `public Text scoreText;` και το τρέχον σκορ του παίκτη `private int score = 0;`. Εξασφαλίζουμε ότι υπάρχει μόνο ένα ScoreManager στο παιχνίδι.

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

Στην μέθοδο Start καλούμε την UpdateScoreText

```
void Start()
{
    UpdateScoreText();
```

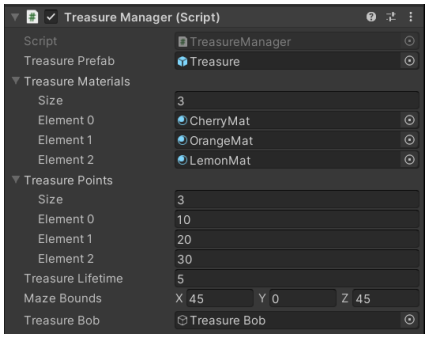
η οποία είναι υπεύθυνη για να ενημερώνει το UI Text που δείχνει το σκορ.

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

Στην συνάρτηση AddScore προσθέτουμε τους πόντους στο συνολικό σκορ και καλούμε την UpdateScoreText.

```
public void AddScore(int points)
{
    if (gameEnded) return; // Av
    score += points;
    UpdateScoreText();
```

Θα προσθέσουμε ένα πεδίο στο TreasureManager για να αντιστοιχίσουμε τις βαθμολογίες στα υλικά των θησαυρών. `public int[] treasurePoints;`



10 πόντους για τον θησαυρό κεράσι, 20 για τον πορτοκάλι και 30 για τον λεμόνι.

Πρέπει να τροποποιήσουμε και το Treasure script για να ενημερώνουμε το σκορ ανάλογα με το υλικό του θησαυρού. Υλοποιούμε την μέθοδο GetTreasurePoints

```
private int GetTreasurePoints()
{
    Renderer renderer = GetComponent<Renderer>();
    if (renderer != null && TreasureManager.Instance.treasureMaterials != null)
    {
        for (int i = 0; i < TreasureManager.Instance.treasureMaterials.Length; i++)
        {
            // Χρησιμοποιούμε ReferenceEquals για να συγκρίνουμε τα υλικά
            if (ReferenceEquals(renderer.sharedMaterial, TreasureManager.Instance.treasureMaterials[i]))
            {
                return TreasureManager.Instance.treasurePoints[i];
            }
        }
    }
    return 0; // Default βαθμολογία αν δεν βρεθεί αντιστοιχία
}
```

στην οποία εντοπίζουμε ποιο υλικό έχει ο θησαυρός χρησιμοποιώντας το Renderer, συγκρίνουμε το υλικό του θησαυρού με τα υλικά του TreasureManager μέσω ReferenceEquals και επιστρέφουμε την αντίστοιχη βαθμολογία. Μέσα στην CollectTreasure υπολογίζουμε τη βαθμολογία του θησαυρού με τη GetTreasurePoints και καλούμε το ScoreManager για να προσθέσει τη βαθμολογία του παίκτη.

```
int points = GetTreasurePoints();
ScoreManager.Instance.AddScore(points);
```

Bonus (d)

Δημιουργούμε ένα UI Text (GameOverText), το τοποθετούμε στο κέντρο της οθόνης, ρυθμίζουμε το μέγεθος, χρώμα, και το μήνυμα να είναι "Game Over" και στη συνέχεια το απενεργοποιούμε.

Στο script TreasureBob προσθέτουμε την αναφορά `public GameObject gameOverText;` και μέσα στη συνάρτηση Die το ενεργοποιούμε

```
if (gameOverText != null)
{
    gameOverText.SetActive(true);
}
```

οπότε μόλις πεθάνει ο Bob εμφανίζεται το μήνυμα

Extra δικές μας προσθήκες

1)

Μία προσθήκη που κάναμε είναι να φτιάξουμε την εφαρμογή μας όταν πεθαίνει ο Bob, να μην τερματίζει το παιχνίδι κατευθείαν, αλλά να παγώνει το παιχνίδι για δύο δευτερόλεπτα με το μήνυμα "Game Over" στην οθόνη και να παίζει ένας ήχος. Οπότε, ορίζουμε το delayBeforeQuit στα δύο δευτερόλεπτα

```
10      public float delayBeforeQuit = 2.0f;
```

Στην μέθοδο Start αρχικοποιούμε το AudioSource και ενεργοποιούμε την επιλογή ignoreListenerPause, επιτρέποντας την αναπαραγωγή ήχων ακόμα και αν το παιχνίδι έχει παγώσει.

```
12      void Start()
13      {
14          audioSource = GetComponent<AudioSource>();
15          if (audioSource != null)
16          {
17              audioSource.ignoreListenerPause = true;
18          }
19      }
```

Μέσα στην μέθοδο OnTriggerEnter που ελέγχουμε αν ο Bob πέσει πάνω σε trap και πεθάνει, παίζουμε τον ήχο

```
35              if (audioSource != null && deathSound != null)
36              {
37                  audioSource.PlayOneShot(deathSound);
38              }
```

Στην συνάρτηση Die παγώνουμε το παιχνίδι

```
55              Time.timeScale = 0;
```

και ξεκινάμε την κορουτίνα WaitAndQuit.

```
58              StartCoroutine(WaitAndQuit());
```

Η κορουτίνα περιμένει τα δύο δευτερόλεπτα και στη συνέχεια καλεί την QuitGame η οποία τερματίζει το παιχνίδι

```
61      IEnumerator WaitAndQuit()
62      {
63          yield return new WaitForSecondsRealtime(delayBeforeQuit);
64
65          QuitGame();
66      }
```

2)

Άλλη μια προσθήκη είναι παρόμοια υλοποίηση με αυτή του "GameOver", η οποία είναι όταν φτάσει ο παίκτης 500 score να εμφανίζει μήνυμα Congratulations, You Won! παίζοντας έναν ήχο.

Στο script ScoreManager ορίζουμε τις μεταβλητές για το Text, τον ήχο και το σκορ για νίκη να είναι 500

```
8       public Text scoreText; // UI γι
9       public GameObject winText; // A
10      public AudioClip winSound; // H
11      private AudioSource audioSource
12      private int score = 0;
13      private bool gameEnded = false;
14      public int winningScore = 500;
```

Στην AddScore ελέγχουμε αν το σκορ φτάσει ή ξεπεράσει το winningScore και καλούμε τη μέθοδο WinGame

```
45      if (score >= winningScore)
46      {
47          WinGame();
48      }
```

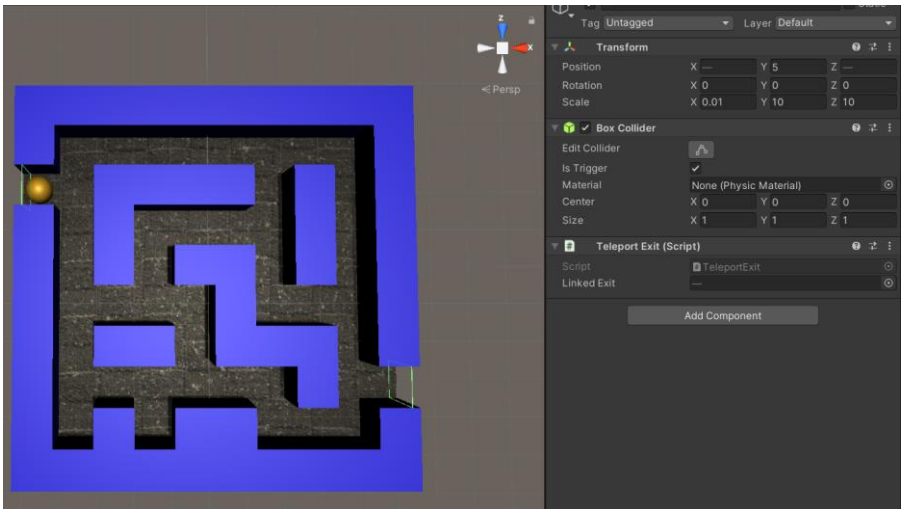
Η WinGame παγώνει το παιχνίδι, παίζει τον ήχο και εμφανίζει το μήνυμα στην οθόνη

```
59      private void WinGame()
60      {
61          gameEnded = true;
62          Debug.Log("Congratulations, You Won!");
63
64          // Παίξιμο ήχου νίκης
65          if (winSound != null && audioSource != null)
66          {
67              audioSource.PlayOneShot(winSound);
68          }
69
70          // Εμφάνιση του μηνύματος νίκης
71          if (winText != null)
72          {
73              winText.SetActive(true);
74          }
75
76          // Πάγωμα του παιχνιδιού
77          Time.timeScale = 0;
78
79          // Τερματισμός μετά από λίγη ώρα
80          StartCoroutine(WaitAndQuit());
81      }
82
83      private System.Collections.IEnumerator WaitAndQuit()
84      {
85          yield return new WaitForSecondsRealtime(4.0f); // Αναμονή για 2 δευτερόλεπτα
86
87          // Τερματισμός παιχνιδιού
88  #if UNITY_EDITOR
89          UnityEditor.EditorApplication.isPlaying = false; // Τερματισμός στο Unity Editor
90  #else
91          Application.Quit(); // Τερματισμός στο Build
92  #endif
93      }
```

Έπειτα καλεί την κορουτίνα WaitAndQuit η οποία τερματίζει το παιχνίδι μετά από 4 δευτερόλεπτα.

3)

Προσθέσαμε επίσης την υλοποίηση όταν ο Bob βγαίνει από την μία έξοδο του λαβύρινθου να εμφανίζεται στην άλλη.
Δημιουργήσαμε δύο GameObjects: Exit1 και Exit2. Προσθέσαμε σε κάθε μία από αυτές ένα Box Collider, ενεργοποιήσαμε το Is Trigger και τα τοποθετήσαμε στις εξόδους του λαβύρινθου.



Φτιάχνουμε ένα script TeleportExit το οποίο περνάμε σε κάθε Exit και στο πεδίο Linked Exit περνάμε την αντίθετη έξοδο.
Ορίζουμε τη μεταβλητή linkedExit που αναφέρεται στην έξοδο τηλεμεταφοράς του Bob και το flag isTeleporting

```
5       public Transform linkedExit; // Ανα
6       private bool isTeleporting = false;
```

Φτιάχνουμε την μέθοδο OnTriggerEnter, η οποία ενεργοποιείται μόνο αν ο Bob εισέλθει στο trigger και το flag είναι false.

```
8   private void OnTriggerEnter(Collider other)
9   {
10      if (other.CompareTag("Player") && !isTeleporting) // Αν ο Bob περάσει και δεν
11      {
12          isTeleporting = true; // Ενεργοποίηση του flag για να αποτραπεί το διπλό
13
14          // Μεταφορά του Bob στη θέση της άλλης εξόδου
15          Vector3 offset = other.transform.position - transform.position; // Για να
16          other.transform.position = linkedExit.position + offset;
17
18          // Επαναφορά του flag στην άλλη έξοδο
19          TeleportExit linkedExitScript = linkedExit.GetComponent<TeleportExit>();
20          if (linkedExitScript != null)
21          {
22              linkedExitScript.isTeleporting = true;
23          }
24      }
25  }
```

Το flag isTeleporting ενεργοποιείται για να αποτρέψουμε την συνεχόμενη κυκλική τηλεμεταφορά. Μετά Ο Bob μεταφέρεται στη θέση της άλλης εξόδου. Η μεταφορά ενημερώνει και την έξοδο προορισμού, ρυθμίζοντας το isTeleporting σε true στο script της άλλης εξόδου.

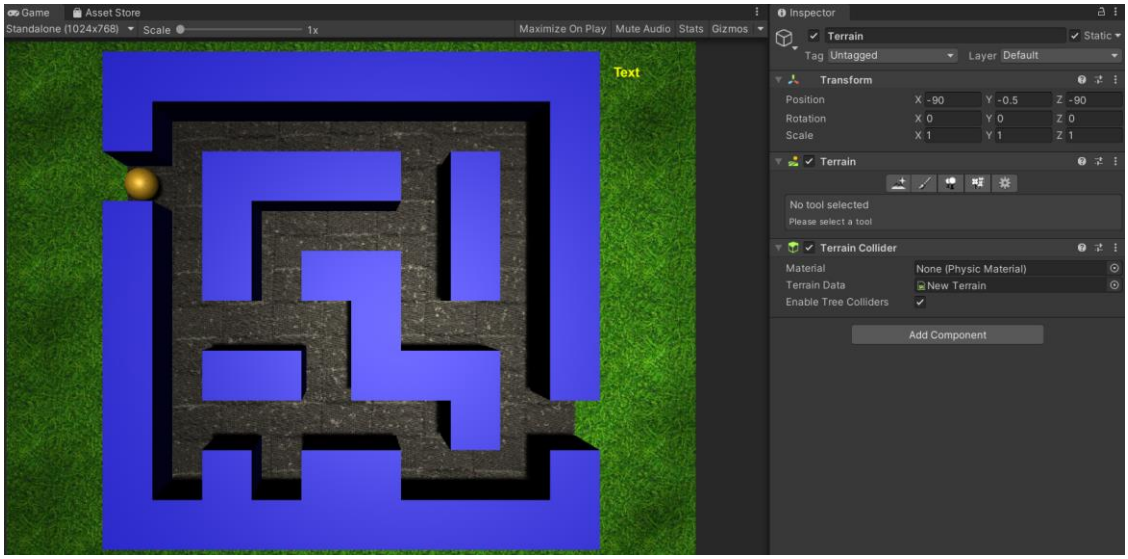
Προσθέτουμε και την μέθοδο OnTriggerExit η οποία όταν ο Bob φύγει από το trigger, το isTeleporting επαναφέρεται σε false.

```
27    private void OnTriggerExit(Collider other)
28    {
29        if (other.CompareTag("Player")) // Όταν
30        {
31            isTeleporting = false; // Επαναφορά
32        }
33    }
```

Τέλος

-

Εφαρμόζουμε ένα Background στον λαβύρινθο. Φτιάχνουμε ένα 3D Object > Terrain, ορίζουμε τις διαστάσεις του να είναι γύρω από τον λαβύρινθο και του δίνουμε την υφή grass.



-

Επίσης εφαρμόζουμε στο παιχνίδι να τερματίζει και με το πλήκτρο Escape το οποίο υλοποιούμε στο script TreasureBob

```
21    void Update()
22    {
23        // Έλεγχος αν πατήθηκε το πλήκτρο ESC
24        if (Input.GetKeyDown(KeyCode.Escape))
25        {
26            QuitGame();
27        }
28    }
```

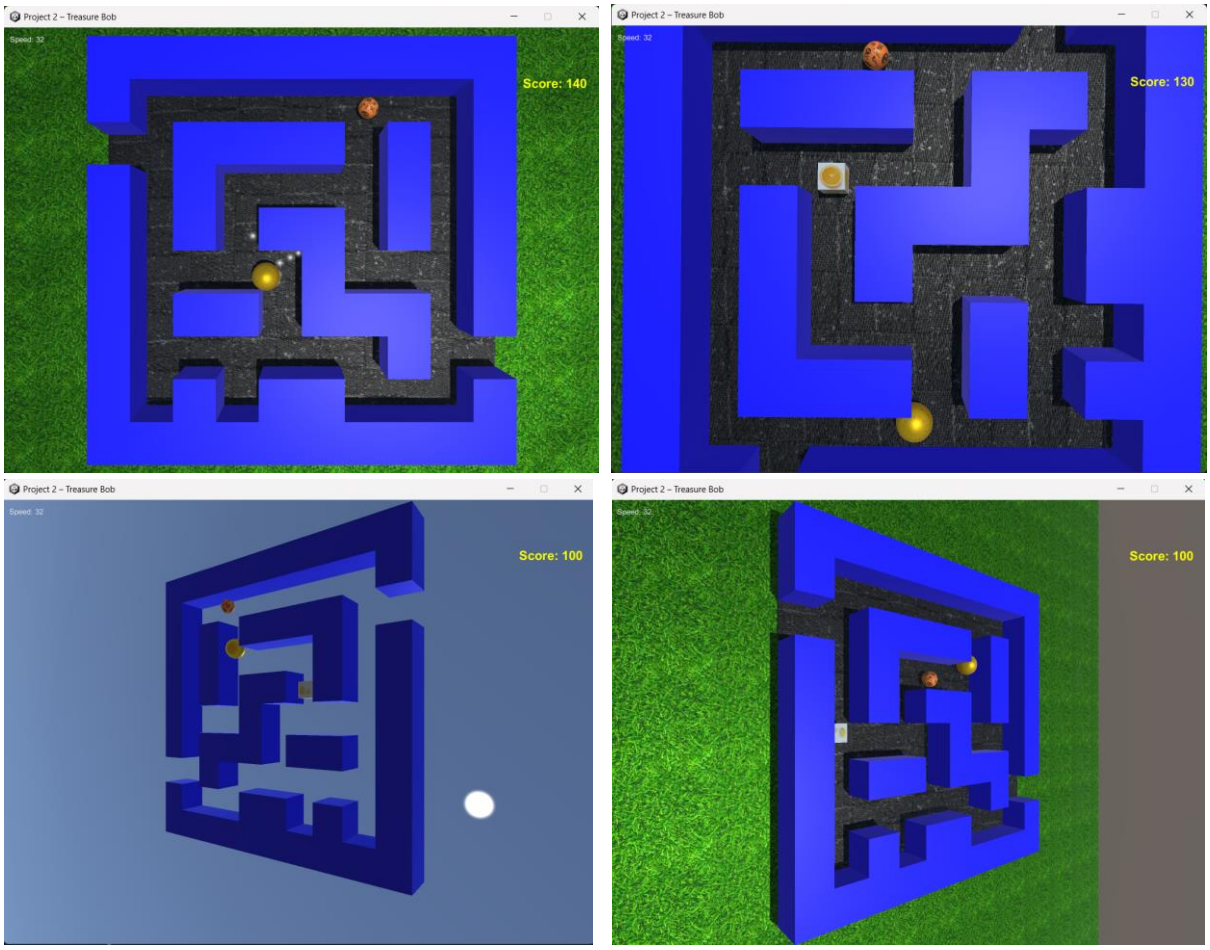
Περιγραφή δυσκολιών υλοποίησης -προβλήματα που συναντήθηκαν

Δεν συναντήθηκαν ιδιαίτερες δυσκολίες μέσα στην υλοποίηση της άσκησης μόνο μικρές σε κάποια σημεία, όπως στα collisions μεταξύ αντικειμένων, στις οποίες ανταπεξήλθαμε άμεσα. Η μεγαλύτερη δυσκολία ήταν στο teleport από την μία έξοδο στην άλλη, το οποίο δεν μας είχε ζητηθεί στην άσκηση αλλά υλοποιήσαμε ως extra.

Πληροφορίες σχετικά με την υλοποίηση

Λειτουργικό σύστημα: Windows 11 Pro Version 23H2
Περιβάλλον: Unity 3D Version 2021.3.14f1

Στιγμιότυπα



Αναφορές – Πηγές που χρησιμοποιήθηκαν κατά την εκπόνηση της εργασίας.

https://docs.unity3d.com/6000.0/Documentation/Manual/GameObjects.html

https://docs.unity3d.com/ScriptReference/MonoBehaviour.Update.html

https://docs.unity3d.com/ScriptReference/Object.Instantiate.html

https://gamedevbeginner.com/how-to-quit-the-game-in-unity/

https://docs.unity3d.com/6000.0/Documentation/Manual/coroutines.html

https://docs.unity3d.com/2021.3/Documentation/Manual/rigidbody-physics-section.html

https://docs.unity3d.com/2021.3/Documentation/Manual/collision-section.html

https://docs.unity3d.com/2021.3/Documentation/Manual/class-Quaternion.html

https://docs.unity3d.com/2021.3/Documentation/ScriptReference/GUI.Label.html

Ευχαριστώ για την ανάγνωση.

Παναγιώτης Παρασκευόπουλος
ΑΜ: 2905