



ΠΑΝΕΠΙΣΤΗΜΙΟ ΙΩΑΝΝΙΝΩΝ

ΤΜΗΜΑ ΜΗΧ. Η/Υ & ΠΛΗΡΟΦΟΡΙΚΗΣ

Γραφικά Υπολογιστών και Συστημάτων Αλληλεπίδρασης

Ημερομηνία παράδοσης: 22/12/2024

ΠΡΟΓΡΑΜΜΑΤΙΣΤΙΚΗ ΑΣΚΗΣΗ 2-Unity 3D

Ανδρέας Καραγεώργος
Βασίλης Κωτόπουλος

ΠΕΡΙΕΧΟΜΕΝΑ

1. Περιγραφή της εργασίας

(i)(15%)

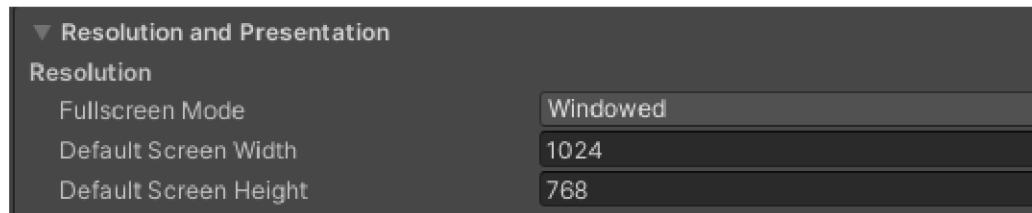
Για την υλοποίηση του πρώτου ερωτήματος θέλουμε να φτιάξουμε μια εφαρμογή Unity 3D που θα τρέχει με ανάλυση 1024x768 και θα έχει για τίτλο «Project 2 – Treasure Bob».

Οπότε πηγαίνουμε Edit->Project Settings->Player και αλλάζουμε

το product name:

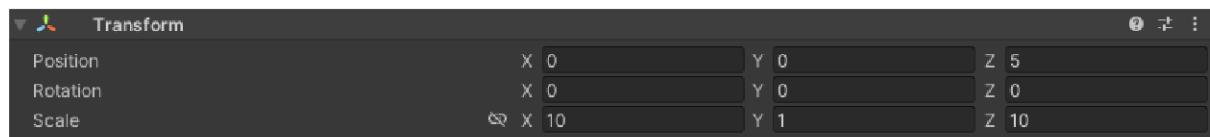


τις αντίστοιχες διαστάσεις:



Έπειτα θέλουμε όταν θα ξεκινάει η εφαρμογή θα φορτώνεται ο λαβύρινθος, ο οποίος ακουμπάει σε ένα πάτωμα, το οποίο σχηματίζεται από ένα επίπεδο (plane) από 100 x 100 τετραγωνάκια.

Ουσιαστικά κάνουμε δεξί κλικ στο Hierarchy , 3D object -> plane . Για να έχει τις διαστάσεις 100x100 το plane αλλάζουμε το scale του x,z άξονα σε 10 :



σχεδιασμένο στο xz επίπεδο (δηλαδή y=0).

Στο έδαφος εφαρμόζουμε την υφή που σας δίνεται (floor.jpg) με απλό drag and drop αφού μας το επιτρέπει η Unity .

Τον λαβύρινθο τον σχεδιάζουμε με κύβους διαστάσεων 10x10x10 δεξί κλικ στο Hierarchy , 3D object -> cube και αλλάζοντας το scale σε 10 για κάθε διάσταση με χρήση drag and drop του material blue για εφαρμογή στον κύβο .



Η “ένωση” των κύβων έγινε με το shortcut κρατημένο ν για να τοποθετηθούν στις ακριβώς επιθυμητές θέσεις κολλητά τον έναν από τον άλλο, επίσης το κέντρο του λαβύρινθου είναι το σημείο (0,0,5) που φαίνεται πάνω με τις συντεταγμένες του plane(χρησιμοποιήθηκε ctrl+d για duplicate των κύβων για να μην χρειάζεται κάθε φορά να αλλάζουμε όλες τις παραμέτρους).

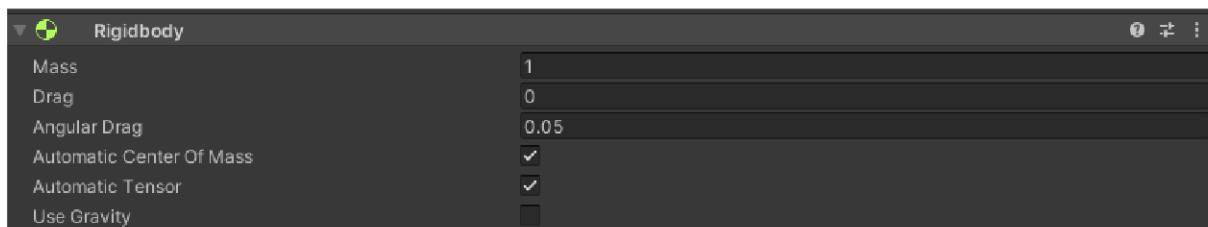
(ii)(20%)

Στο δεύτερο τμήμα ζωγραφίζουμε τον Treasure Bob ο οποίος είναι μια σφαίρα με διάμετρο 7, με drag and drop εφαρμόζουμε την υφή bob.jpg. Του δίνουμε τρόπο να κινείται όπως θα φανεί παρακάτω και υλοποιούμε το **bonus b) (15)** που μας δίνει πλήκτρα για αυξομείωση της ταχύτητας του Bob(πέντε διαβαθμίσεις).
3D object -> sphere



Ο Bob δεν μετακινείται στον γάξονα:

Για να το πετύχουμε αυτό κάνουμε add component στον Treasure Bob ->Rigidbody και κάνουμε uncheck την Use Gravity ιδιότητα.



Ο Bob δεν μπορεί να βγει έξω από τον λαβύρινθο , δεν μπορεί να περάσει μέσα από τοίχους και κινείται από τον χρήστη, με σταθερό βήμα, με τα εξής πλήκτρα:

- κίνηση παράλληλα στον άξονα x του συστήματος παγκόσμιων συντεταγμένων
- κίνηση παράλληλα στον άξονα z του συστήματος παγκόσμιων συντεταγμένων

Για το παραπάνω φτιάξαμε ένα script το οποίο ονομάσαμε TreasureBobMovements και το εφαρμόσαμε στον Bob με drag and drop.

Αρχικά ορίζουμε τις μεταβλητές :

```
public float speed = 5f;
5 references
private Transform transform;
```

Αποθηκεύουμε μια αναφορά στο **Transform** component του αντικειμένου, ώστε να μπορούμε να το χρησιμοποιούμε για να μετακινήσουμε τον Bob και την ταχύτητα του .

Μέσα στην Update ,η οποία ανανεώνεται σε κάθε frame, βάζουμε τον ακόλουθο έλεγχο:

```
if(Input.GetKey(KeyCode.L)){transform.Translate(0,0, speed * Time.deltaTime);} // Δεξιά κίνηση.
if(Input.GetKey(KeyCode.J)){transform.Translate(0,0, -speed * Time.deltaTime);} // Αριστερή κίνηση.
if(Input.GetKey(KeyCode.I)){transform.Translate(-speed * Time.deltaTime,0,0);} // Πάνω κίνηση.
if(Input.GetKey(KeyCode.K)){transform.Translate(speed * Time.deltaTime,0,0);} // Κάτω κίνηση.
```

Ο έλεγχος αυτός διαβάζει την είσοδο από το πληκτρολόγιο και μετακινεί τον Bob στις αντίστοιχες κατευθύνσεις, ως εξής:

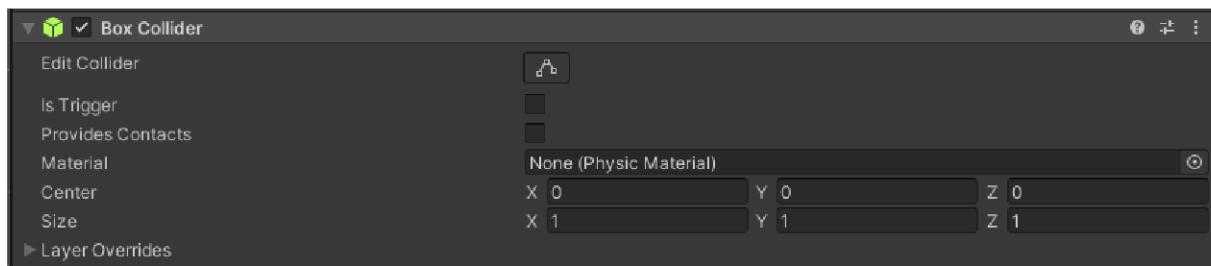
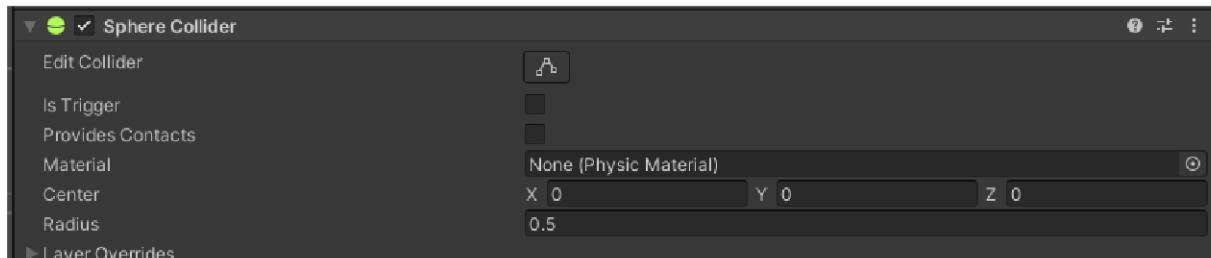
1. KeyCode.L: Μετακίνηση δεξιά
2. KeyCode.J: Μετακίνηση αριστερά
3. KeyCode.I: Μετακίνηση πάνω
4. KeyCode.K: Μετακίνηση κάτω

Επίσης για να γίνει σωστά η κίνηση του Bob πρέπει να κάνουμε check τις ιδιότητες Freeze rotation στο Rigidbody αλλιώς ο Bob περιστρέφεται γύρω από τον εαυτό του και δημιουργεί bug στις κινήσεις του.

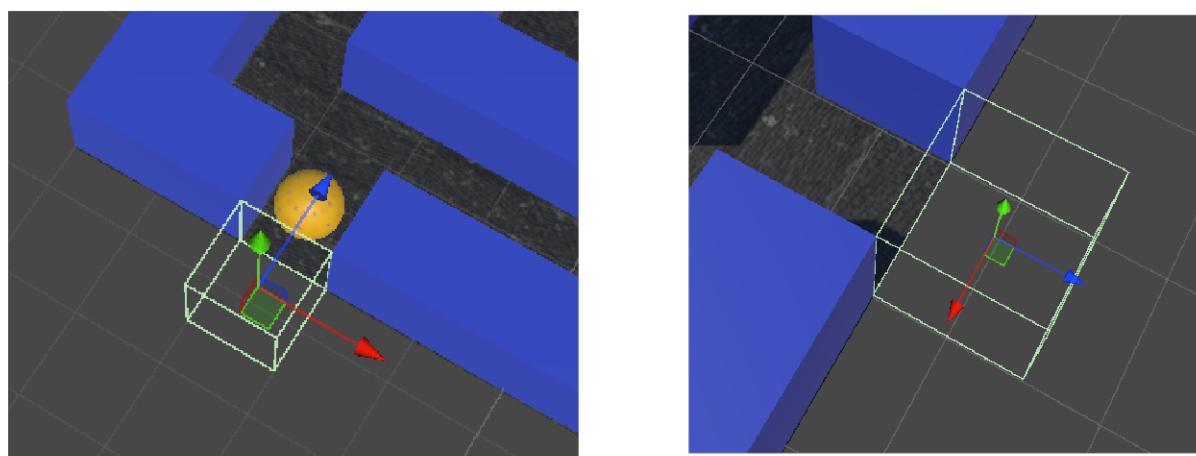


Ο περιορισμός που δεν αφήνει τον Bob να βγει έξω από τον λαβύρινθο και δεν μπορεί να περάσει μέσα από τους τοίχους γίνεται με την χρήση των colliders.

Έχουμε ένα collider για κάθε κύβο και έναν στην σφαίρα , όταν ο Bob κινείται προς κάποιος τοίχο εμποδίζεται από το να περάσει μέσα από αυτόν.



Για να μην μπορεί ο Bob να βγει εκτός του λαβυρίνθου φτιάχνουμε δύο κύβους ακόμα τους οποίους ονομάζουμε Barriers .Τους κάνουμε αόρατους, απενεργοποιούμε στον καθένα το mesh αλλά τους αφήνουμε το collider. Έτσι όταν ο Bob κατευθύνεται προς εκείνα τα σημεία εμποδίζεται ακόμα και δεν βλέπουμε ορατά το πως.



Τέλος με αυτόν τον έλεγχο στο TreasureBobMovements υλοποιήσαμε το **bonus b) (15)** που μας δίνει δυνατότα αυξομείωσης της ταχύτας του Bob(πέντε διαβαθμίσεις). Αύξηση με το πλήκτρο Z και μείωση με το πλήκτρο X.

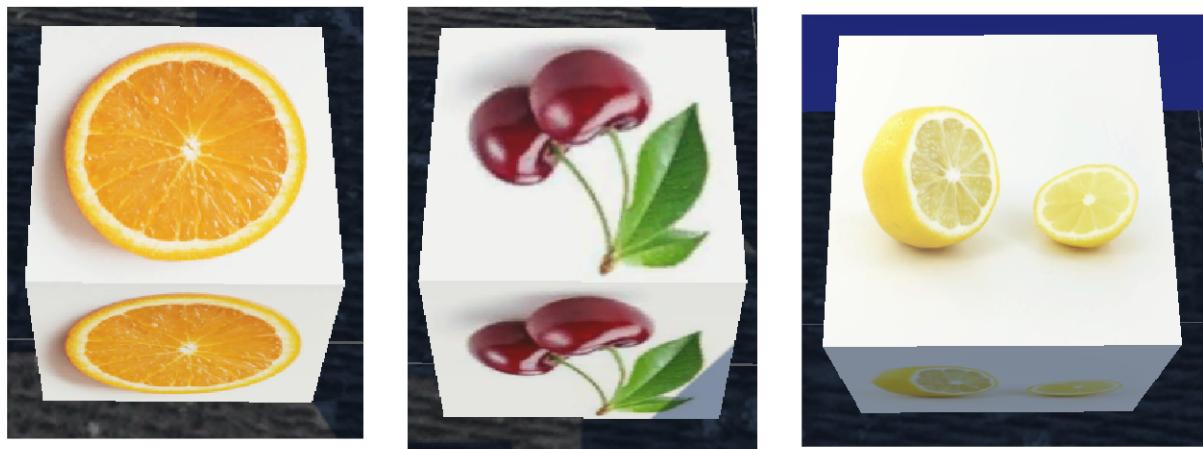
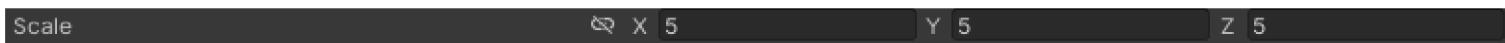
```
if(Input.GetKeyDown(KeyCode.Z)){ // Αυξάνουμε την ταχύτητα  
    if(speed<10){speed+=1f;}  
if(Input.GetKeyDown(KeyCode.X)){ // Μειώνουμε την ταχύτητα  
    if(speed>5){speed-=1f;}  
}
```

(iii)(20%)

Στο τρίτο τμήμα φτιάχνουμε τους θησαυρούς, την τυχαία εμφάνισή τους μέσα στο λαβύρινθο και την τυχαία επανεμφάνισή τους, αφότου τους ακουμπήσει ο Bob ή αφότου περάσει ένα ορισμένο χρονικό διάστημα. Επίσης υλοποιούμε το **bonus a) (15)** προσθέτουμε ήχο δηλαδή όταν ο Bob ακουμπήσει τους θησαυρούς και το **bonus c) (15)**, να μαζεύει βαθμούς (να έχει score) ο Bob και κάθε θησαυρός να αντιστοιχεί σε διαφορετική βαθμολογία.

Αρχικά φτιάχνουμε τρεις κύβους με διαστάσεις 5x5x5 :

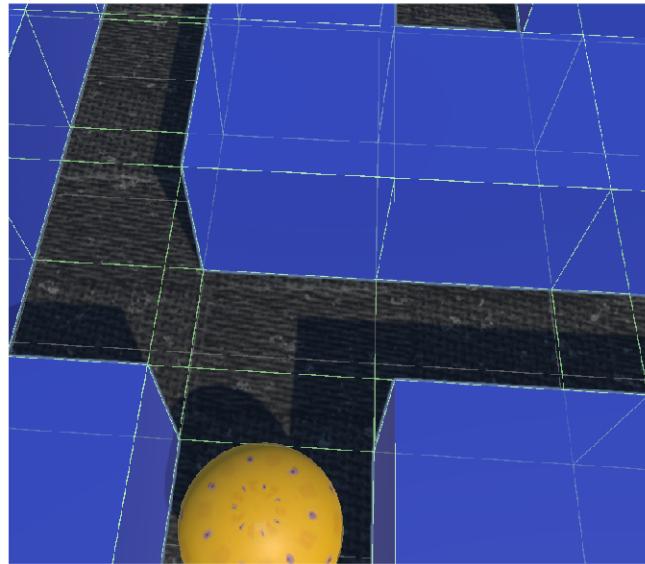
δεξί κλικ στο Hierarchy , 3D object -> cube ,αλλάζουμε το scale σε 5 για κάθε διάσταση, με drag and drop εφαρμόζουμε τις τρεις υφές μία για κάθε κύβο και τους τοποθετούμε μέσα στο λαβύρινθο.



Για να υλοποιήσουμε την τυχαία εμφάνιση τους μέσα στο λαβύρινθο :

Αρχικά δημιουργούμε στις εσωτερικές θέσεις του λαβυρίνθου ένα κύβο για κάθε θέση . Έπειτα στον κάθε κύβο απενεργοποιούμε τα Mesh Collider έτσι ώστε ο Bob να μπορεί να

κινείται κανονικά μέσα από αυτά .Ουσιαστικά αυτοί οι κύβοι ομαδοποιούνται χρησιμοποιούνται για τις πιθανές θέσεις στις οποίες θα εμφανίζονται οι θησαυροί(και για το επόμενο ερώτημα οι εχθροί).



Στο TreasureAndEnemies έχουμε ομαδοποιήσει τους θησαυρούς και τους εχθρούς ,το οποίο είναι ένα parent object που χρησιμοποιούν και οι δύο .

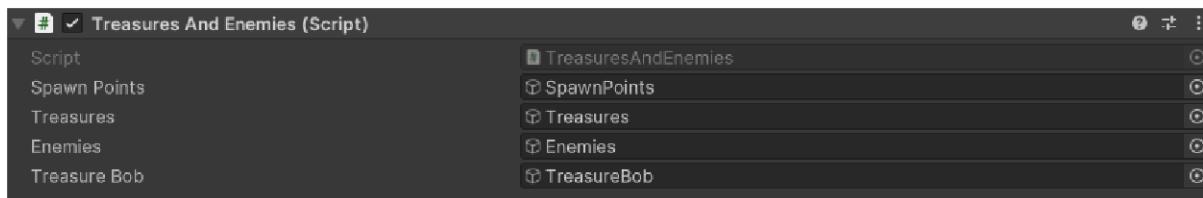
Πρώτα ορίζουμε τα πεδία μας :

```
1 reference
public GameObject SpawnPoints; // Ο γονέας των spawn points
1 reference
public GameObject Treasures; // Ο γονέας των treasures
1 reference
public GameObject Enemies; // Ο γονέας των enemies
3 references
public GameObject TreasureBob; // Ο Bobs

3 references
private GameObject[] spawnPoints; // Πίνακας με τα spawn points
7 references
private GameObject[] treasures; // Πίνακας με τους θησαυρούς
5 references
private GameObject[] enemies; // Πίνακας με τους εχθρούς
8 references
private float[,] occupiedPositions; // Οι θέσεις με τους θησαυρούς και τους εχθρούς.
```

(Τα οποία εξηγούνται μέσα από τα σχόλια του κώδικα, όπου τα spawnPoints είναι οι πιθανές κενές θέσεις για τις οποίες μιλήσαμε παραπάνω)

Αρχικοποιούμε τα public πεδία :



Στην μέθοδο Start() κάνουμε την αρχικοποίηση των private πεδίων:

```
void Start()
{
    treasures = returnChildren(Treasures).ToArray();
    enemies = returnChildren(Enemies).ToArray();
    spawnPoints = returnChildren(SpawnPoints).ToArray();
    occupiedPositions = new float[6,2];
}
```

Η μέθοδος returnChildren() παίρνει σαν όρισμα ένα GameObject γονέα και επιστρέφει σε μία λίστα (List) όλα τα παιδιά του (όχι τα παιδιά των παιδιών αν υπάρχουν). (Δεν βάλαμε την μέθοδο να επιστρέφει κατευθείαν πίνακα για την περιπτωση που μπορεί να χρειαζόμασταν στην αναπτυξή του κώδικα λίστα και όχι array) :

```
// Βοηθητική μέθοδος που επιστρέφει όλα τα παιδιά του γονέα.
3 references
private List<GameObject> returnChildren(GameObject parent){
    List<GameObject> childObjects = new List<GameObject>();
    foreach(Transform child in parent.transform){
        childObjects.Add(child.gameObject);
    }
    return childObjects;
}
```

Η μέθοδος updateOccupiedPositions() μας βοηθάει στο να βάζουμε σε έναν δισδιάστατο πίνακα τις θέσεις των θησαυρών και των εχθρών , έτσι ώστε να μπορούμε να ξέρουμε ποιες θεσεις ειναι ελευθερες και να μην τοποθετήσουμε θησαυρούς και εχθρούς στην ίδια θέση κατά λάθος.

```
public void updateOccupiedPositions(){ // Ενημέρωση του πίνακα occupiedPositions
    int i,c;
    for(i=0; i<treasures.Length; i++){
        occupiedPositions[i,0] = treasures[i].transform.position.x;
        occupiedPositions[i,1] = treasures[i].transform.position.z;
    }
    for(c=0, i=treasures.Length; i<treasures.Length+enemies.Length; i++,c++){
        occupiedPositions[i,0] = enemies[c].transform.position.x;
        occupiedPositions[i,1] = enemies[c].transform.position.z;
    }
}
```

Η μέθοδος `isPositionFree()` μας βοηθάει στο να ξέρουμε αν μία θέση είναι κενή ή όχι, χρησιμοποιώντας τον πίνακα `occupiedPositions`.

```
2 references
public bool isPositionFree(Vector3 position){ // Έλεγχος αν μια θέση είναι ελεύθερη
    for(int i=0; i<occupiedPositions.GetLength(0); i++){
        if(position.x==occupiedPositions[i,0] && position.z==occupiedPositions[i,1]){
            return false;
        }
    }
    return true;
}
```

Η μέθοδος `isPlayerNotNear()` μας βοηθάει στο να ξέρουμε αν ο παίκτης είναι κοντά στο όρισμα `Vector2`, ώστε να μπορούμε να αποφύγουμε να τοποθετήσουμε θησαυρό ή εχθρό πάνω στον παίκτη. Η δήλωση των μεταβλητών γίνεται έξω από την μέθοδο ώστε ο κώδικας να είναι περισσότερο `optimized`:

```
private float bx;
2 references
private float bz;
2 references
private float px;
2 references
private float pz;
2 references
public bool isPlayerNotNear(Vector3 p){ // Έλεγχος αν ο παίκτης είναι κοντά σε μια θέση
    bx = abs(TreasureBob.transform.position.x);
    bz = abs(TreasureBob.transform.position.z);
    px = abs(p.x);
    pz = abs(p.z);
    return abs(px-bx)>=4f || abs(bz-pz)>=4f;
}
```

Η μέθοδος `abs()` φτιάχτηκε διότι είχε πρόβλημα με την `Math.abs` όταν χρησιμοποιήσαμε την βιβλιοθήκη `System` με την `Random`:

```
// Βοηθητική μέθοδο διότι υπάρχει πρόβλημα με την Math.Abs στο 'using System' με την Random
6 references
public float abs(float number){
    return number < 0 ? -number : number;
}
```

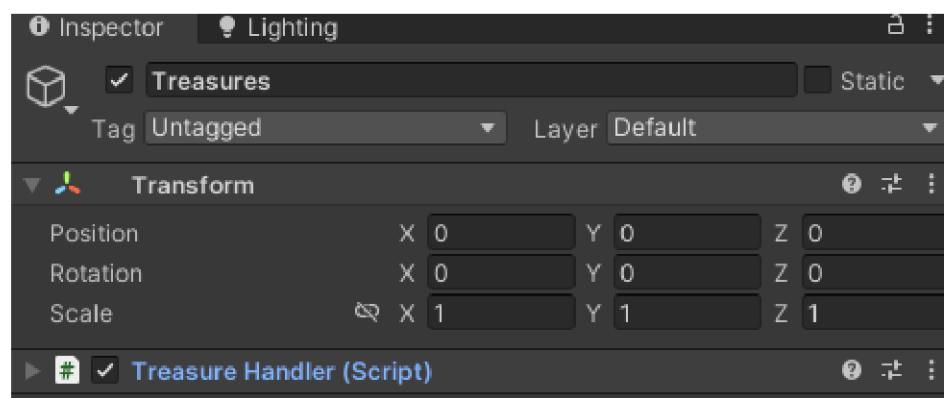
Η μέθοδος `isBobDead()` μας βοηθάει να ξέρουμε αν ο παίκτης έχει πεθάνει ή όχι:

```
public bool isBobDead(){ // Έλεγχος αν ο Bob είναι νεκρός
    return TreasureBob.GetComponent<DeathHandler>().isDead();
```

Και τέλος έχουμε βάλει μεθόδους πρόσβασης (Getters) για την πρόσβαση τους στις άλλες κλάσεις:

```
// Getters
2 references
public GameObject getRandomSpawnPoint(){return spawnPoints[Random.Range(0,spawnPoints.Length)];}
1 reference
public GameObject[] getTreasures(){return treasures;}
1 reference
public GameObject[] getEnemies(){return enemies;}
```

Αφότου εξηγήσαμε αυτήν την ομαδοποίηση , περνάμε στο script TreasureHandler το οποίο βρίσκεται στην ομάδα Treasures. To script αυτό διαχειρίζεται τις ιδιότητες των θησαυρών:

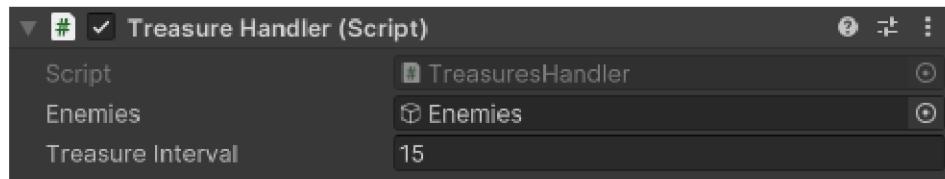


Πρώτα ορίζουμε τα πεδία μας :

```
2 references
public GameObject Enemies; //Η ομάδα με τους εχθρούς.
2 references
public float treasureInterval; // Διάστημα σε δευτερόλεπτα για να καλεστεί η μέθοδος
4 references
private float treasureTimer; // Χρονοδιακόπτης για την παρακολούθηση του χρόνου που έχει παρέλθει
2 references
private GameObject[] treasures; // Οι θησαυροί.

7 references
private TreasuresAndEnemies treasuresAndEnemies;
```

Αρχικοποίηση public πεδίων:



Αρχικοποίηση private πεδίων:

```
void Start()
{
    treasureTimer = treasureInterval;
    treasuresAndEnemies = transform.parent.GetComponent<TreasuresAndEnemies>();
    treasures = treasuresAndEnemies.getTreasures();

}
```

Στην Update γίνεται η κύρια υλοποίηση

```
void Update()
{
    //=====
    if(!treasuresAndEnemies.isBobDead()){
        treasureTimer += Time.deltaTime;
        if (treasureTimer >= treasureInterval)
        {
            // Reset τον χρόνο
            treasureTimer = 0f;
            // Παύση της κίνησης των εχθρών.
            Enemies.GetComponent<EnemyHandler>().setCanMove(false);
            placeTreasuresInRandomPosition();
            // Εκκίνηση της κίνησης των εχθρών.
            Enemies.GetComponent<EnemyHandler>().setCanMove(true);
        }
    }
    //=====
}
```

Εξηγούμε γραμμή γραμμή την Update:

Ελέγχουμε αν ο Bob είναι νεκρός. Αν ο Bob δεν είναι νεκρός (το isBobDead() επιστρέφει false), τότε εκτελείται ο υπόλοιπος κώδικας.

```
if(!treasuresAndEnemies.isBobDead()){
```

To treasureTimer αυξάνεται κατά το Time.deltaTime, δηλαδή τον χρόνο που μεσολάβησε από το προηγούμενο frame. Χρησιμοποιείται για τη μέτρηση του χρόνου.

```
treasureTimer += Time.deltaTime;
```

Αν το treasureTimer ξεπεράσει το treasureInterval, δηλαδή το διάστημα που έχουμε θέσει:

```
if (treasureTimer >= treasureInterval)
```

Το χρονόμετρο επανεκκινείται.

```
treasureTimer = 0f;
```

Σταματάμε την κίνηση των εχθρών(για να μην υπάρχει περίπτωση να πέσει ο εχθρός(στον οποίο θα αναφερθούμε αργότερα) στο ίδιο σημείο με τον θησαυρό).

```
Enemies.GetComponent<EnemyHandler>().setCanMove(false);
```

Τοποθετούμε τους θησαυρούς σε τυχαία σημεία.

```
placeTreasuresInRandomPosition();
```

Τέλος επιτρέπουμε ξανά την κίνηση των εχθρών.

```
Enemies.GetComponent<EnemyHandler>().setCanMove(true);
```

Η placeTreasuresInRandomPosition() οφείλεται για την τοποθέτηση των θησαυρών στις τυχαίες θέσεις :

```
// Μέθοδος για την τοποθέτηση των θησαυρών σε τυχαίες θέσεις
4 references
private GameObject spawnPoint;
3 references
private bool placed;
1 reference
private void placeTreasuresInRandomPosition(){
    foreach(GameObject treasure in treasures){
        placed = false;
        while(!placed){
            treasuresAndEnemies.updateOccupiedPositions();
            spawnPoint = treasuresAndEnemies.getRandomSpawnPoint();
            // Εάν η θέση είναι ελεύθερη και ο παίκτης δεν είναι κοντά, τοποθετούμε τον θησαυρό
            if(treasuresAndEnemies.isPositionFree(spawnPoint.transform.position) && treasuresAndEnemies.isPlayerNotNear(spawnPoint.transform.position)){
                treasure.transform.position = spawnPoint.transform.position;
                // Ενεργοποίηση των Collider και το Renderer του θησαυρού
                if(!treasure.GetComponent<BoxCollider>().enabled){
                    treasure.GetComponent<BoxCollider>().enabled = true;
                    treasure.GetComponent<MeshRenderer>().enabled = true;
                    treasure.transform.localScale *= 2f;
                }
                placed = true;
            }
        }
    }
}
```

Δήλωση μεταβλητών εκτός της μεθόδου για optimization :

```
private GameObject spawnPoint;
3 references
private bool placed;
```

To spawnPoint είναι ένα αντικείμενο που θα χρησιμοποιηθεί ως σημείο εμφάνισης του θησαυρού και η place είναι μια boolean μεταβλητή που δείχνει να έχει τοποθετηθεί ο θησαυρός.

Εξετάζουμε κάθε θησαυρό μέσα στη λίστα treasures .

```
foreach(GameObject treasure in treasures){
```

Αρχικά ορίζουμε τον θησαυρό να μην έχει τοποθετηθεί .

```
placed = false;
```

Όσο ο θησαυρός δεν έχει τοποθετηθεί, συνεχίζεται η αναζήτηση κατάλληλης θέσης.

```
while(!placed){
```

Ενημερώνουμε οι θέσεις που είναι ήδη κατειλημμένες, ώστε να μην τοποθετήσουμε θησαυρό εκεί.

```
treasuresAndEnemies.updateOccupiedPositions();
```

Επιλέγουμε ένα τυχαίο σημείο για την πιθανή τοποθέτηση του θησαυρού.

```
spawnPoint = treasuresAndEnemies.getRandomSpawnPoint();
```

Ελέγχουμε αν η θέση είναι ελεύθερη και ο παίκτης δεν είναι κοντά σε αυτήν την θέση.

```
if(treasuresAndEnemies.isPositionFree(spawnPoint.transform.position) && treasuresAndEnemies.isPlayerNotNear(spawnPoint.transform.position)){
```

Αν η θέση είναι κατάλληλη, μετακινούμε τον θησαυρό στην επιλεγμένη θέση .

```
treasure.transform.position = spawnPoint.transform.position;
```

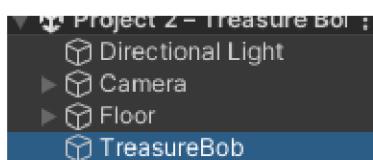
Ελέγχουμε αν το BoxCollider του θησαυρού είναι απενεργοποιημένο . Αν είναι το ενεργοποιούμε (για να μπορεί να “ξαναπιάσει” ο bob τον θησαυρό) , ενεργοποιούμε τον MeshRenderer για να είναι ορατός ο θησαυρός και επαναφέρουμε το scale του θησαυρού στο αρχικό , διπλασιάζοντας το (διότι όταν ο παίκτης πιάνει τον θησαυρό , ο θησαυρός υποδιπλασιάζεται, αυτό συμβαίνει μέσα στο script TreasureCollector).

```
if(!treasure.GetComponent<BoxCollider>().enabled){
    treasure.GetComponent<BoxCollider>().enabled = true;
    treasure.GetComponent<MeshRenderer>().enabled = true;
    treasure.transform.localScale *= 2f;
}
```

Τέλος η μεταβλητή placed γίνεται true και ο βρόχος σταματάει .

```
placed = true;
```

Έπειτα για να συλλέξει ο Bob τον θησαυρό χρησιμοποιείται το script CollisionHandler(το οποίο ανήκει στον Bob):



Πρώτα ορίζουμε τα πεδία μας :

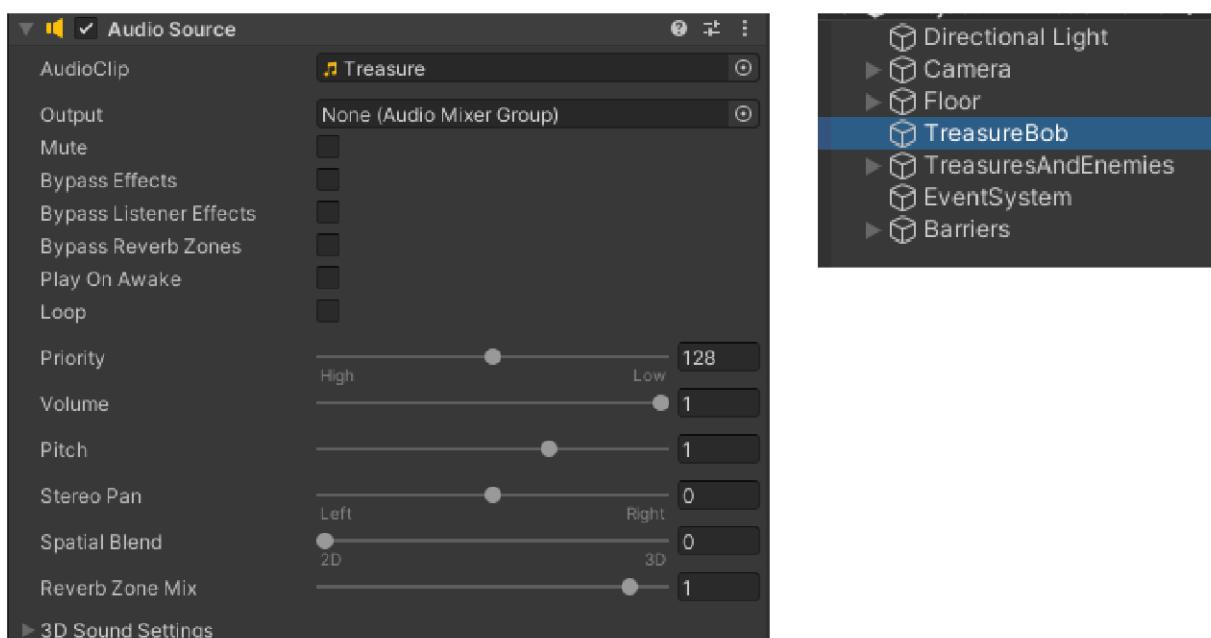
```
private ScoreHandler scoreHandler;
2 references
private DeathHandler deathHandler;
2 references
private AudioSource sound;
4 references
private float timer; // Χρονομετρητής για την εξαφάνιση του θησαυρού
2 references
private float timeInterval; // Χρονικό διάστημα για την εξαφάνιση του θησαυρού

4 references
private bool treasureFound; // Αναγνωρίζει αν βρέθηκε θησαυρός
2 references
private GameObject collidedTreasure; // Αναφορά στο αντικείμενο του συγκρουόμενου θησαυρού
```

Τα αρχικοποιούμε στην start:

```
void Start()
{
    timer = 0f;
    treasureFound = false;
    timeInterval = 1f;
    scoreHandler = this.GetComponent<ScoreHandler>();
    deathHandler = this.GetComponent<DeathHandler>();
    sound = this.GetComponent<AudioSource>();
}
```

Εδώ αναφέρουμε ότι έχουμε υλοποιήσει και το [bonus α\)\(15\)](#) τού ήχου για το οποίο έχουμε προσθέσει ένα Audio Source component στον Bob



```
void Update(){
    if(treasureFound){ // Αν βρέθηκε θησαυρός, ξεκινάει ο χρονομετρητής για την εξαφάνισή του
        timer += Time.deltaTime;
        if(timer>=timeInterval){
            collidedTreasure.gameObject.GetComponent<MeshRenderer>().enabled = false;
        }
    }else{
        timer = 0; // Επαναφορά χρονομετρητή
    }
}
```

Ελέγχουμε αν έχει βρεθεί ο θησαυρός, αν έχει βρεθεί αρχίζει ο μετρητής

```
if(treasureFound){
```

Αυξάνουμε την τιμή της μεταβλητής timer κατά το Time.deltaTime. Το Time.deltaTime είναι ο χρόνος που πέρασε από το προηγούμενο frame και χρησιμοποιείται για να υπολογίζουμε το πέρασμα του χρόνου ανεξάρτητα από τα FPS του υπολογιστή .

```
    timer += Time.deltaTime;
```

Αν ο μετρητής έχει περάσει το όριο που έχουμε δώσει:

```
if(timer>=timeInterval){
```

Εξαφανίζουμε τον θησαυρό (απενεργοποιώντας τον το MeshRenderer).

```
    collidedTreasure.gameObject.GetComponent<MeshRenderer>().enabled = false;
```

Επαναφέρουμε το treasureFound σε false, αφού ο θησαυρός δεν είναι πλέον ενεργός ή "βρέθηκε".

```
    treasureFound = false;
```

Τέλος να δεν έχει βρεθεί ο θησαυρός , ο μετρητής μηδενίζεται.

```
}else{
    timer = 0; // Επαναφορά χρονομετρητή
}
```

Έπειτα έχουμε την μέθοδο OnCollisionEnter() της Unity η οποία χειρίζεται τα collisions. Αυτή η μέθοδος ενεργοποιείται όταν ο Bob συγκρουστεί με κάποιο άλλο αντικείμενο. Η πληροφορία της σύγκρουσης αποθηκεύεται στο αντικείμενο collision.

```
private void OnCollisionEnter(Collision collision){
    // Συγκρουση με θησαυρό
    if(collision.transform.tag.Contains("treasure")){
        collidedTreasure = collision.gameObject;
        collision.collider.enabled = false;
        sound.Play();
        collision.transform.localScale *= 0.5f;
        treasureFound = true;
    }
    // Προσδιορισμός τύπου θησαυρού και αύξηση σκορ ανάλογα
    if(collision.transform.tag == "treasureLemon"){
        Debug.Log("A treasure lemon has been found !");
        scoreHandler.setScore(scoreHandler.getScore()+1); // Αυξάνουμε το scor κατά ένα.
    }
    else if(collision.transform.tag == "treasureOrange"){
        Debug.Log("A treasure orange has been found !");
        scoreHandler.setScore(scoreHandler.getScore()+2); // Αυξάνουμε το scor κατά δύο.
    }
    else if(collision.transform.tag == "treasureCherry"){
        Debug.Log("A treasure cherry has been found !");
        scoreHandler.setScore(scoreHandler.getScore()+3); // Αυξάνουμε το scor κατά τρία.
    }else if(collision.transform.tag == "enemy"){ // Συγκρουση με εχθρό
        deathHandler.kill();
        Debug.Log("You died.");
    }
}
```

```
// Συγκρουση με θησαυρό
if(collision.transform.tag.Contains("treasure")){
    collidedTreasure = collision.gameObject;
    collision.collider.enabled = false;
    sound.Play();
    collision.transform.localScale *= 0.5f;
    treasureFound = true;
}
```

```
if(collision.transform.tag.Contains("treasure")){
```

Ελέγχουμε αν το tag του αντικειμένου με το οποίο έγινε σύγκρουση περιέχει τη λέξη "treasure". Αν ναι, σημαίνει ότι το αντικείμενο είναι θησαυρός.

```
    collidedTreasure = collision.gameObject;
```

Αποθηκεύουμε το αντικείμενο θησαυρού που συγκρούστηκε στο collidedTreasure.

```
collision.collider.enabled = false;
```

Απενεργοποιούμε το collider του θησαυρού ώστε να μην μπορεί να συγκρουστεί ξανά με τον παίκτη.

```
sound.Play();
```

Παίζουμε τον ήχο που υποδηλώνει ότι συλλέξαμε τον θησαυρό.

```
collision.transform.localScale *= 0.5f;
```

Μειώνουμε το scale του θησαυρού στο μισό όπως μας έχει ζητηθεί.

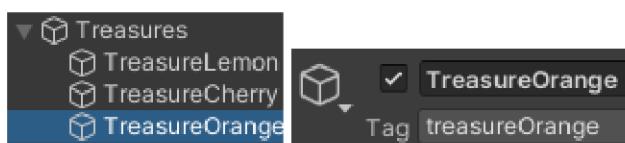
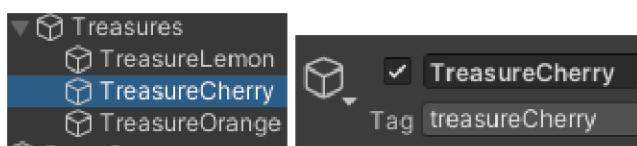
```
treasureFound = true;|
```

Ορίζουμε την μεταβλητή treasureFound σε true, υποδηλώνοντας ότι βρέθηκε ο θησαυρός. Ελέγχουμε το tag του κάθε θησαυρού ώστε να υλοποιήσουμε το bonus c)(15) με τα ξεχωριστά score ανα θησαυρό .

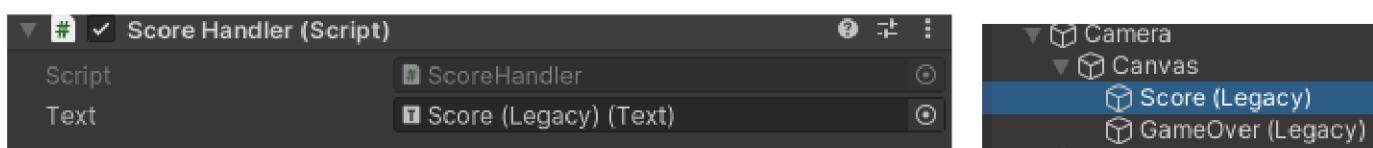
Στο treasureLemon δίνουμε ένα πόντο , στο treasureOrange δίνουμε δύο και στο treasureCherry δίνουμε 3. Τα Debug.Log χρησιμοποιήθηκαν για έλεγχο του κώδικα όταν τον αναπτύσσαμε. Οι αυξήσει γίνονται χρησιμοποιώντας το αντικείμενο scoreHandler.

```
// Προσδιορισμός τύπου θησαυρού και αύξηση σκορ ανάλογα
if(collision.transform.tag == "treasureLemon"){
    Debug.Log("A treasure lemon has been found !");
    scoreHandler.setScore(scoreHandler.getScore()+1); // Αυξάνουμε το score κατά ένα.
}
else if(collision.transform.tag == "treasureOrange"){
    Debug.Log("A treasure orange has been found !");
    scoreHandler.setScore(scoreHandler.getScore()+2); // Αυξάνουμε το score κατά δύο.
}
else if(collision.transform.tag == "treasureCherry"){
    Debug.Log("A treasure cherry has been found !");
    scoreHandler.setScore(scoreHandler.getScore()+3); // Αυξάνουμε το score κατά τρία.
} else if(collision.transform.tag == "enemy"){ // Συγκρουση με εχθρό
    deathHandler.kill();
    Debug.Log("You died.");
}
```

Τα tags τα οποία δόθηκαν σε κάθε θησαυρό ξεχωριστά φαίνονται εδώ:



To script ScoreHandler διαχειρίζεται το score του παίκτη :



Ορίζουμε τα πεδία του :

```
4 references
private int score; // Το σκορ του παίκτη

1 reference
public Text text; // Το αντικείμενο Text για την εμφάνιση του σκορ
```

Αρχικοποιούμε το score στην Start να είναι 0 :

```
void Start()
{
    score = 0;
}
```

Ενημερώνουμε το score στο text σε κάθε frame :

```
void Update(){
    text.text = "Score: " + score; // Ενημέρωση του κειμένου του Text με το τρέχον σκορ
}
```

Τέλος έχουμε μεθόδους πρόσβασης και ανάθεσης έτσι ώστε να μπορούν άλλα αντικείμενα να αλληλεπιδρούν με το scoreHandler:

```
// Getter για το σκορ  
3 references  
public int getScore() { return score; }  
// Setter για το σκορ  
3 references  
public void setScore(int score) { this.score = score; }
```

(iv) (20%)

Στο τέταρτο τμήμα φτιάχνουμε τις παγίδες (η ομαδοποίηση των οποίων έχει ήδη γίνει όπως αναφερθήκαμε παραπάνω στο TreasureAndEnemies), την τυχαία εμφάνισή τους μέσα στο λαβύρινθο και την τυχαία επανεμφάνισή τους , αφότου περάσει ένα ορισμένο χρονικό διάστημα. Επίσης την λήξη του παιχνιδιού όταν ο Bob “πεθάνει” , όταν δηλαδή ακουμπήσει κάποια παγίδα .

Εδώ επίσης υλοποιούμε το **bonus d) (10)** , να εμφανίζεται δηλαδή το μήνυμα <>**Game Over**>> αν το παιχνίδι τερματήσει.

Αρχικά φτιάχνουμε τρεις σφαίρες με διαστάσεις 5x5x5 :

δεξί κλικ στο Hierarchy , 3D object -> sphere ,αλλάζουμε το scale σε 5 για κάθε διάσταση, με drag and drop εφαρμόζουμε το “death” material σε κάθε σφαίρα και τις τοποθετούμε μέσα στο λαβύρινθο.

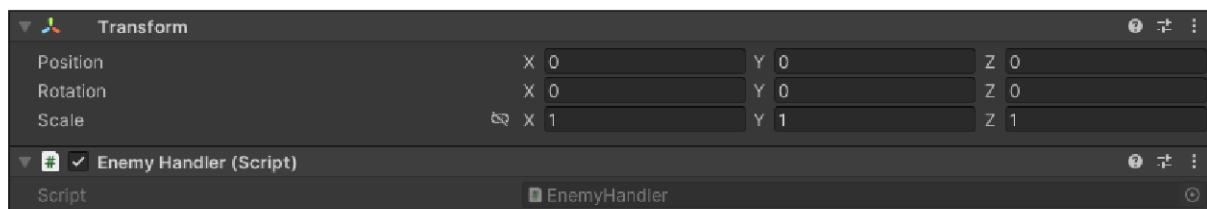
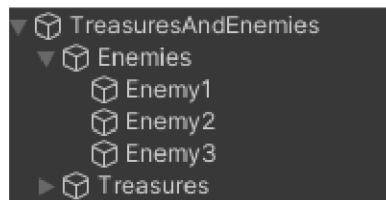


Στους τρεις εχθρούς δώσαμε το tag enemy



Για να υλοποιήσουμε τις λειτουργίες των παγίδων χρησιμοποιούμε το script EnemyHandler το οποίο βρίσκεται στην ομάδα Enemies:

:



Πρώτα ορίζουμε τα πεδία μας :

```
private GameObject[] enemies; // Πίνακας με όλους τους εχθρούς
4 references
private float enemyInterval;
4 references
private float enemyTimer; // Χρονομετρητής για τον υπολογισμό του χρόνου που έχει περάσει
3 references
private bool canMove; // Έλεγχος αν μπορούν να τοποθετηθούν οι εχθροί
7 references
private TreasuresAndEnemies treasuresAndEnemies; // Αντικείμενο γονέας
```

Αρχικοποιούμε τα private πεδία :

```
void Start()
{
    // Αρχικοποίηση πεδίων
    treasuresAndEnemies = transform.parent.GetComponent<TreasuresAndEnemies>();
    enemyInterval = (float) Random.Range(0,11);
    enemyTimer = enemyInterval;
    enemies = treasuresAndEnemies.getEnemies();
    canMove = false;
}
```

Δήλωση μεταβλητών εκτός της update για optimization :

```
private int enemyPosition;
3 references
private bool placed;
4 references
private GameObject spawnPoint;
```

Στην Update γίνεται η κύρια υλοποίηση η οποία εξηγείται λεπτομερώς με τα υπάρχον σχόλια:

```
void Update()
{
    if(!treasuresAndEnemies.isBobDead()){ // Όσο ο bob είναι ζωντανός το παιχνίδι συνεχίζει.
        placed = false;
        enemyTimer += Time.deltaTime;
        if(enemyTimer>=enemyInterval && canMove){
            // Εύρεση τυχαίου εχθρού
            enemyPosition = Random.Range(0,enemies.Length);
            // Επαναλαμβάνουμε μέχρι να βρούμε μια ελεύθερη θέση
            while(!placed){
                // Ενημέρωση των κατειλημμένων θέσεων
                treasuresAndEnemies.updateOccupiedPositions();
                // Εύρεση τυχαίας θέσης για να μπει ο εχθρός
                spawnPoint = treasuresAndEnemies.getRandomSpawnPoint();
                // Αν η θέση είναι ελεύθερη και ο παίκτης δεν είναι κοντά
                if(treasuresAndEnemies.isPositionFree(spawnPoint.transform.position) && treasuresAndEnemies.isPlayerNotNear(spawnPoint.transform.position)){
                    // Τοποθέτηση του εχθρού στη νέα θέση
                    enemies[enemyPosition].transform.position = spawnPoint.transform.position;
                    placed = true;
                    enemyTimer = 0f;
                    // Εύρεση τυχαίας χρονικής στοιχημής για τη νέα επανατοποθέτηση
                    enemyInterval = (float) Random.Range(0,11);
                }
            }
        }
    }
}
```

Όταν συμβεί κάποιο collision ανάμεσα σε κάποια παγίδα και τον Bob όπως προαναφέρθηκε ,ο Bob πρέπει να “πεθαίνει” και το παιχνίδι να τερματίζει (εδώ επιλέξαμε όταν ο Bob αγγίζει την παγίδα , για να γίνεται εμφανής η σύγκρουση να αλλάζουμε την υφή του σε αυτήν των παγίδων). Αυτό το υλοποιούμε μέσω του script DeathHandler το οποίο εφαρμόζουμε στον Bob:



Ορίζουμε τα πεδία του :

```
private bool dead; // Δεικτης θανάτου του παίκτη
1 reference
public Texture deathTexture; // Υφή για την εμφάνιση του θανάτου
1 reference
public Text GameOverText; // Text για την εμφάνιση του μηνύματος "GAME OVER"
```

Τον αρχικοποιούμε σε ζωντανό :

```
void Start()
{
    dead = false;
}
```

Την μέθοδο που μας επιστρέφει το αν είναι νεκρός :

```
// Επιστρέφει αν ο παίκτης είναι νεκρός
3 references
public bool isDead() { return dead; }
```

Τέλος την μέθοδο που τον σκοτώνει :

```
public void kill(){// Μέθοδος για τον θάνατο του παίκτη
    dead = true;
    this.GetComponent<Renderer>().material.mainTexture = deathTexture;
    GameOverText.text = "GAME OVER !";
}
```

Ανιχνεύουμε πότε ο Bob έχει συγκρουστεί με κάποιον εχθρό χρησιμοποιώντας το script CollisionHandler(έχουμε εξηγήσει τι κάνει στο ερώτημα των θησαυρών):

```
else if(collision.transform.tag == "enemy"){ // Συγκρουση με εχθρό
    deathHandler.kill();
    Debug.Log("You died.");
}
```

Το **bonus d) (10)** υλοποιήθηκε αφού δημιουργήσαμε ένα πεδίο GamerOverText τύπου Text το οποίο εμφανίζεται στην οθόνη μόλις σκοτωθεί ο Bob, δηλαδή μόλις τερματήσει το παιχνίδι.

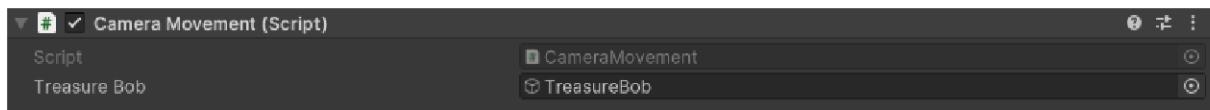


(V) (20%)

Στο πέμπτο ερώτημα υλοποιούμε την κίνηση της κάμερας .Με τα βελάκια του πληκτρολογίου η κάμερα κινείται στους άξονες x και z του συστήματος παγκόσμιων συντεταγμένων και τα πλήκτρα <+>/<-> κινείται κατά μήκος του άξονα y. Τέλος με το κουμπί <r> θα περιστρέφεται η κάμερα γύρω από τον εαυτό της .

(Το rotate υλοποιήθηκε στον άξονα τον x αντί για τον y λόγω της αρχικής τοποθέτησης της κάμερας , η λειτουργία του παρ' όλα αυτά είναι η ίδια)

Για να υλοποιήσουμε τις λειτουργίες της κάμερας υλοποιήσαμε το script CameraMovement το οποίο είναι τοποθετημένο στο camera:



Ορίσαμε τα πεδία :

```
private Transform transform;
7 references
private float speed = 10f;
1 reference
public GameObject treasureBob; // O Bob
```

Αρχικοποιούμε το transform πεδίο:

```
void Start()
{
    transform = this.GetComponent<Transform>();
```

Αν ο Bob είναι ζωντανός έχουμε την κίνηση της κάμερας:

```
void Update()
{
    // Αν ο Bob είναι ζωντανός, μπορούμε να κουνίσουμε την κάμερα
    if(!treasureBob.GetComponent<DeathHandler>().isDead()){
        if(Input.GetKey(KeyCode.RightArrow)){transform.Translate(speed * Time.deltaTime,0,0);} // Δεξιά
        if(Input.GetKey(KeyCode.LeftArrow)){transform.Translate(-speed * Time.deltaTime,0,0);} // Αριστερά
        if(Input.GetKey(KeyCode.UpArrow)){transform.Translate(0,speed * Time.deltaTime,0);} // Πάνω
        if(Input.GetKey(KeyCode.DownArrow)){transform.Translate(0,-speed * Time.deltaTime,0);} // Κάτω
        if(Input.GetKey(KeyCode.KeypadPlus)){transform.Translate(0,0,speed * Time.deltaTime);} // Zoom in
        if(Input.GetKey(KeyCode.KeypadMinus) || Input.GetKey(KeyCode.Minus)){transform.Translate(0,0,-speed * Time.deltaTime);} // Zoom out
        if(Input.GetKey(KeyCode.R)){transform.Rotate(speed * Time.deltaTime,0,0);} // Rotate
    }
}
```

φωτογραφίες από rotations του camera angle :

Η αρχική θέση της κάμερας :

Score: 0



προς τα κάτω :



προς τα πάνω:



φωτογραφίες από zoom in/zoom out:



2. Πληροφορίες σχετικά με την υλοποίηση

Λειτουργικό Σύστημα: Για την υλοποίηση του έργου χρησιμοποιήθηκε Linux.

Unity-Version: Unity 2022.3.49f1

Ιδιαιτερότητες στον τρόπο εκτέλεσης - χρήσης: Η ιδιαιτερότητα που αντιμετωπίσαμε είναι ότι όταν βάλαμε ειδικούς χαρακτήρες στον τίτλο(παύλα) δεν εμφανίζοταν σωστά . Κατά τα άλλα κατεβάσαμε το περιβάλλον της Unity με την έτοιμη εντολή του Ανδρέα Καραγεώργου και δεν αντιμετωπίσαμε θέμα στην υπόλοιπη υλοποίηση .

Φάκελος που βρίσκεται το εκτελέσιμο :
TreasureBob/TreasureBob.x86_64

md5 checksum: 81ee20b650b6575fcc592d54a674f5a1 TreasureBob.zip

Project link- Drive:

<https://drive.google.com/file/d/1q1A59yGmM8zB6kTRPyQcwzxQwCpTElV2/view?usp=sharing>

Δεν χρησιμοποιήθηκε κάποια άλλη βιβλιοθήκη και οι μόνες ιδιαιτερότητες είναι το rotation της κάμερας που αναφέρθηκε όπως και η αλλαγή υφής του παίχτη όταν αυτός “πεθαίνει” . Τα bonus έχουν υλοποιηθεί όλα και τοποθετηθεί με μπλε γράμματα μέσα στην υλοποίηση. Τελος οι “εχθροί” που αναφέρουμε μέσα στον κώδικα ταυτίζονται με τις παγίδες απλά χρησιμοποιήθηκε άλλο όνομα.

3. Αξιολόγηση λειτουργίας ομάδας

Υπήρχε αμοιβαία συνεισφορά τόσο στον τομέα της υλοποίησης του κώδικα όσο και στο report/readme και από τα δύο μέλη της ομάδας . Δουλεύαμε παράλληλα εξ αποστάσεως λύνοντας την άσκηση σε ξεχωριστό αρχείο το κάθε μέλος και αφού φτιάχναμε την υλοποίηση κρατούσαμε σημειώσεις για το report/readme pdf και συγχωνεύσαμε τις απαραίτητες αλλαγές-τροποποιήσεις στο τελικό παραδοτέο(έγινε χρήση του github για γρήγορη ανανέωση αλλαγών στην υλοποίηση).

Διαμορφώσαμε ένα γενικό πλάνο προσέγγισης της άσκησης, φτιάξαμε με βάση τα βήματα τμηματικά τον κώδικα κάνοντας προσθήκες στο δοσμένο αρχείο και αφού το υλοποιήσαμε φτιάξαμε το αρχείο report/readme pdf.

4. Αναφορές-Πηγές

- Διαφάνειες εργαστηρίου (+forum ερωτήσεων)
- Unity tutorials
- Να αναφερθεί ότι δεν χρησιμοποιήθηκε κώδικας από την προηγούμενη άσκηση
- <https://pixabay.com/sound-effects/> (για την επιλογή του sound-effect)