

Βρέχει Μπάλες!!

Γραφικά Υπολογιστών και Συστημάτων Αλληλεπίδρασης

Σταυρούλα Κούτσικου ΑΜ:4396

16/12/2022

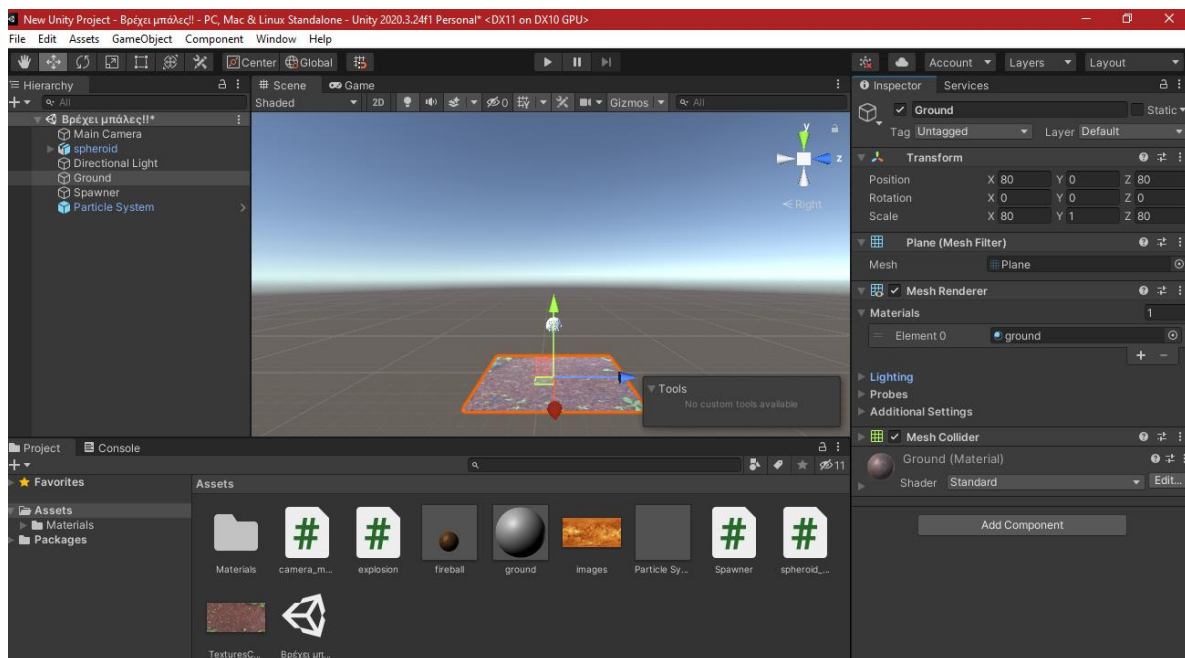
ΑΝΑΛΥΤΙΚΗ ΠΕΡΙΓΡΑΦΗ ΤΗΣ ΕΡΓΑΣΙΑΣ

Τα ερωτήματα της εργασίας τα οποία ολοκληρώσαμε είναι τα εξής:

(i)

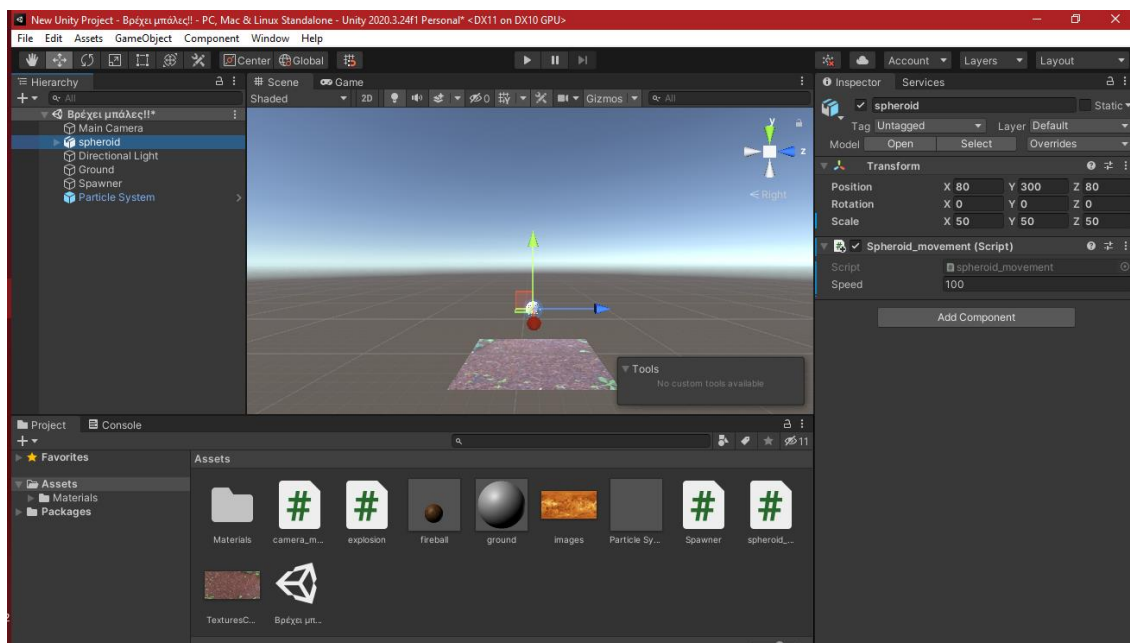
Για το ερώτημα αυτό, δημιουργήσαμε το project με το τίτλο “Βρέχει Μπάλες!!”, με ανάλυση 1024x768μπάλες!!», με μπλε background, όπως ακριβώς μας ζητήθηκε, με αρχική φόρτωση στην οθόνη το «έδαφος» και το «σκάφος».

Το έδαφος αποτελείται απο 80 x 80 τετράγωνα, σχεδιασμένο πάνω στο xz επίπεδο (δηλαδή $y=0$), στο οποίο εφαρμόσαμε υφη εδάφους, όπως φαίνεται και στην παρακάτω εικόνα.



(Εικόνα 1. Η απεικόνιση του εδάφους στην Unity3d)

Το σκάφος το φορτώσαμε απτο δοσμένο αντικείμενο spheroid.obj, το οποίο κλιμακώθηκε κατά παράγοντα 50 και εμφανίζεται στο κέντρο του εδάφους σε ύψος $y=300$.

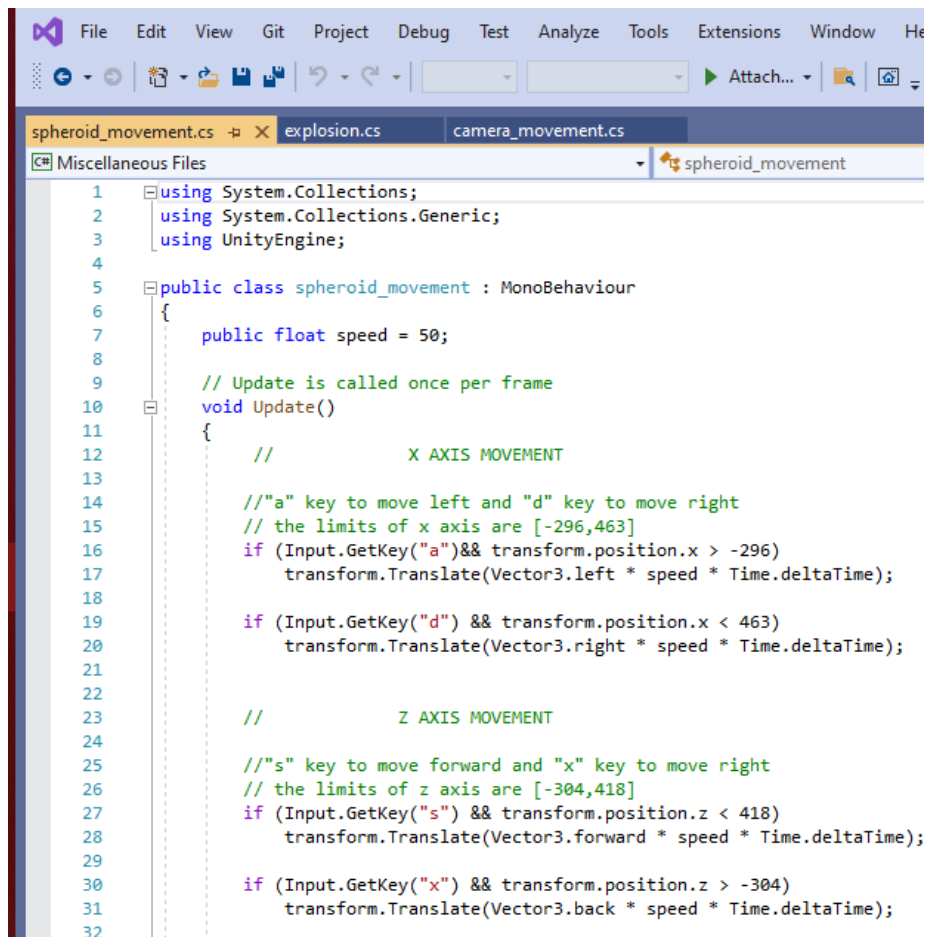


(Εικόνα 2. Η απεικόνιση του σκάφους στην Unity3d)

(ii)

Για το δεύτερο ερώτημα ρεαστήκαμε με τον εξής τροπο.

Δημιουργήσαμε το αρχείο spheroid_movement.cs, μέσα στο οποίο βάλαμε κώδικα για την σταθερή κίνηση του σκάφους καθώς και για την οριοθέτηση της κίνησης μέσα στο έδαφος.



```
1 using System.Collections;
2 using System.Collections.Generic;
3 using UnityEngine;
4
5 public class spheroid_movement : MonoBehaviour
6 {
7     public float speed = 50;
8
9     // Update is called once per frame
10    void Update()
11    {
12        // X AXIS MOVEMENT
13
14        //"a" key to move left and "d" key to move right
15        // the limits of x axis are [-296,463]
16        if (Input.GetKey("a") && transform.position.x > -296)
17            transform.Translate(Vector3.left * speed * Time.deltaTime);
18
19        if (Input.GetKey("d") && transform.position.x < 463)
20            transform.Translate(Vector3.right * speed * Time.deltaTime);
21
22        // Z AXIS MOVEMENT
23
24        //"s" key to move forward and "x" key to move right
25        // the limits of z axis are [-304,418]
26        if (Input.GetKey("s") && transform.position.z < 418)
27            transform.Translate(Vector3.forward * speed * Time.deltaTime);
28
29        if (Input.GetKey("x") && transform.position.z > -304)
30            transform.Translate(Vector3.back * speed * Time.deltaTime);
31
32    }
```

```
        // Y AXIS MOVEMENT
        //"w" key to move up and "e" key to move down
        // the limits of y axis are [38,500]
        if (Input.GetKey("w") && transform.position.y < 1000)
            transform.Translate(Vector3.up * speed * Time.deltaTime);
        if (Input.GetKey("e") && transform.position.y > 38)
            transform.Translate(Vector3.down * speed * Time.deltaTime);
    }
```

(Εικόνες 3 και 4. Ο κώδικας spheroid_movement.cs)

Όπως βλέπετε εργαστήκαμε διαφορετικά για τον κάθε άξονα.

Η κάθε if έχει σαν παράμετρο την εντολή "Input.GetKey("a")" η οποία δεσμεύει το πλήκτρο μας(στην περίπτωση εδώ το πλήκτρο a) και, στην συνέχεια, έχουμε την "transform.position.x > -296)" η οποία οριοθετεί την κίνηση στον εκάστοτε άξονα(στην περίπτωση εδώ λέει πως όταν το x γίνει μεγαλύτερο του -296 σταματάμε την κίνηση).

Μέσα στην if, τοποθετήσαμε την εντολή "transform.Translate(Vector3.left * speed * Time.deltaTime);" η οποία μετακινεί το εκάστοτε αντικείμενο στον x άξονα κατά 1 unit/second. Ο πολλαπλασιασμός με το speed έγινε για να γίνεται πιο γρήγορα η κίνηση και για να εδραιωθεί η σταθερότητά της.

Πρίν δημιουργήσουμε κώδικα, μετακινήσαμε το σκάφος στους άξονες x,y,z μέσα στα όρια του εδάφους και συλλέξαμε τα εξής στοιχεία:

Όρια για τον άξονα x:

<a>: -296

<d>: 463

Όρια για τον άξονα y:

<w>: 435

<e>: 38

Όρια για τον άξονα z:

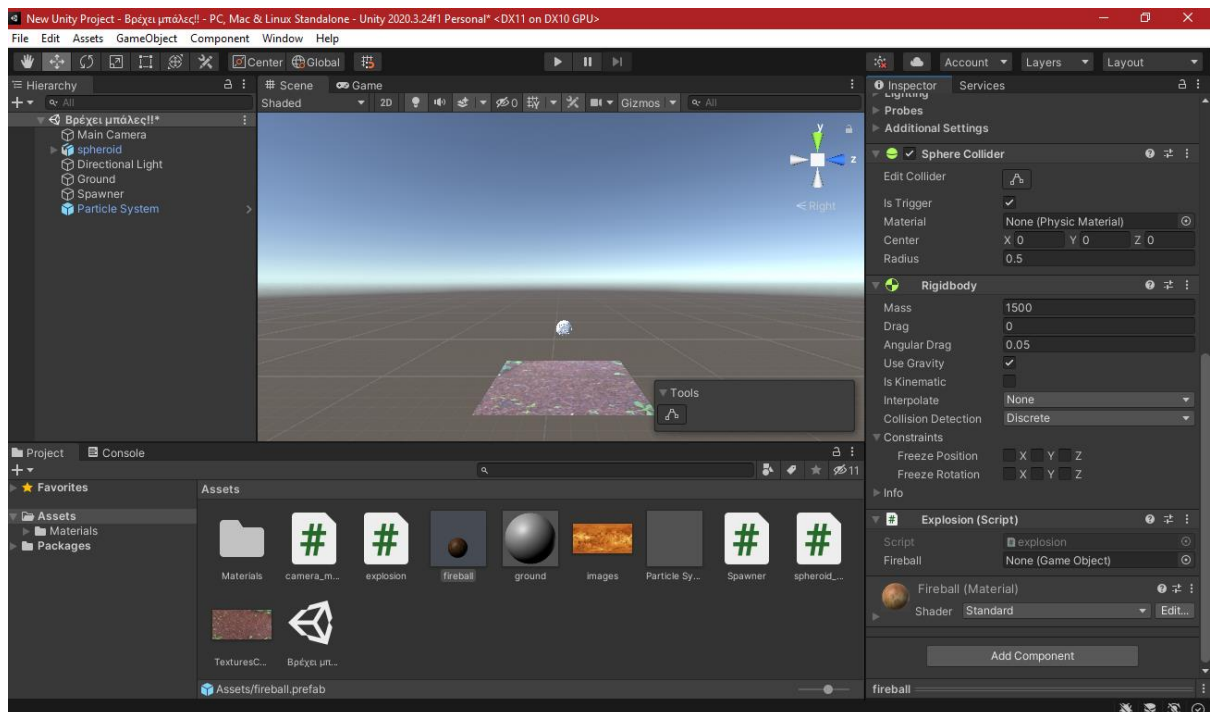
<s>: 418

<x>: -304

Πλέον, το σκάφος μας κινείται πάνω απτο έδαφος με σταθερή ταχύτητα, πατώντας τα <a,d> κινείται αριστερά, δεξιά αντίστοιχα, πατώντας τα <s,x> κινείται μπρος, πίσω και <w,e> κινείται πάνω, κάτω. Όταν δεν πατάμε κάποιο πλήκτρο το σκάφος δεν κινείται.

(iii) Το τρίτο μέρος της εργασίας περιλαμβάνει αρχικά την δημιουργία μιας μπάλας φωτιάς πατώντας το Spasebar, το οποίο και κάναμε. Η μπάλα θα εμφανίζεται σε μια τυχαία θέση στο χώρο πάνω από το έδαφος, με τυχαίο ύψος y που είναι ένας ακέραιος αριθμός στην περιοχή $600 \leq y < 1000$. Στη συνέχεια, η σφαίρα, στην οποία έχουμε εφαρμόσει υφή φωτιάς, πέφτει κάθετα προς το έδαφος λόγω βαρύτητας.

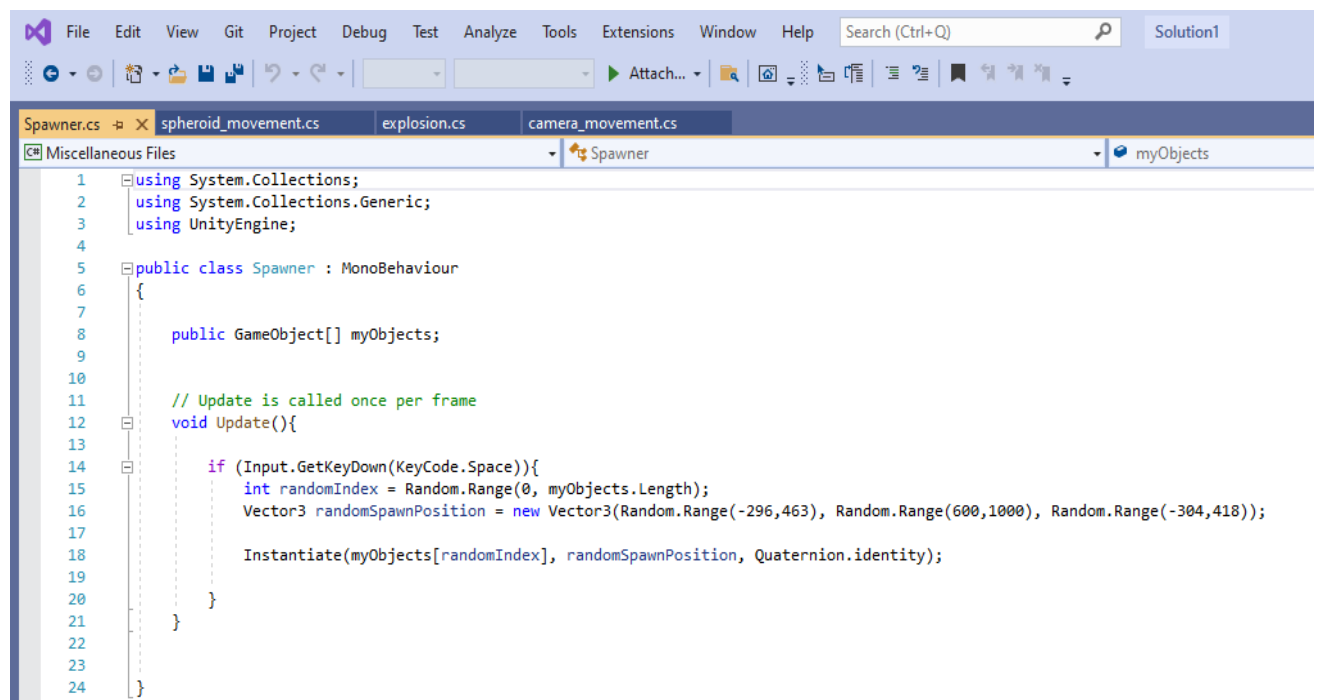
Αυτό που δεν εκπονήθηκε ακριβώς σε αυτό το ερώτημα είναι η τυχαιότητα του μεγέθους της σφαίρας. Ανταυτού, όλες οι σφαίρες που δημιουργούνται είναι ίδιου μεγέθους.



(Εικόνα 5. Η σφαίρα (Fireball))

Για να πετύχουμε την εφαρμογή βαρύτητας στην σφαίρα, την μετατρέψαμε σε Rigidbody όπως φαίνεται και στα δεξιά της Εικόνας 5. Παρόλα αυτά η κάθε σφαίρα που δημιουργείται πέφτει αρκετά αργά, πράγμα που δεν μπορέσαμε να διορθώσουμε.

Ο κώδικας για την δημιουργία των διαφόρων σφαιρών είναι ο εξής:



```
1 using System.Collections;
2 using System.Collections.Generic;
3 using UnityEngine;
4
5 public class Spawner : MonoBehaviour
6 {
7
8     public GameObject[] myObjects;
9
10
11     // Update is called once per frame
12     void Update(){
13
14         if (Input.GetKeyDown(KeyCode.Space)){
15             int randomIndex = Random.Range(0, myObjects.Length);
16             Vector3 randomSpawnPosition = new Vector3(Random.Range(-296,463), Random.Range(600,1000), Random.Range(-304,418));
17
18             Instantiate(myObjects[randomIndex], randomSpawnPosition, Quaternion.identity);
19
20         }
21     }
22 }
23
24
25
```

(Εικόνα 6. Spawner.cs)

Πιο συγκεκριμένα, σαν παράμετρο της if βάλαμε την εντολή “Input.GetKeyDown(KeyCode.Space”, η οποία διασφαλίζει πως για να γίνει μια ενέργεια, πρέπει να πατησουμε το space. Στην συνέχεια, η εντολή “int randomIndex = Random.Range(0, myObjects.Length);” και

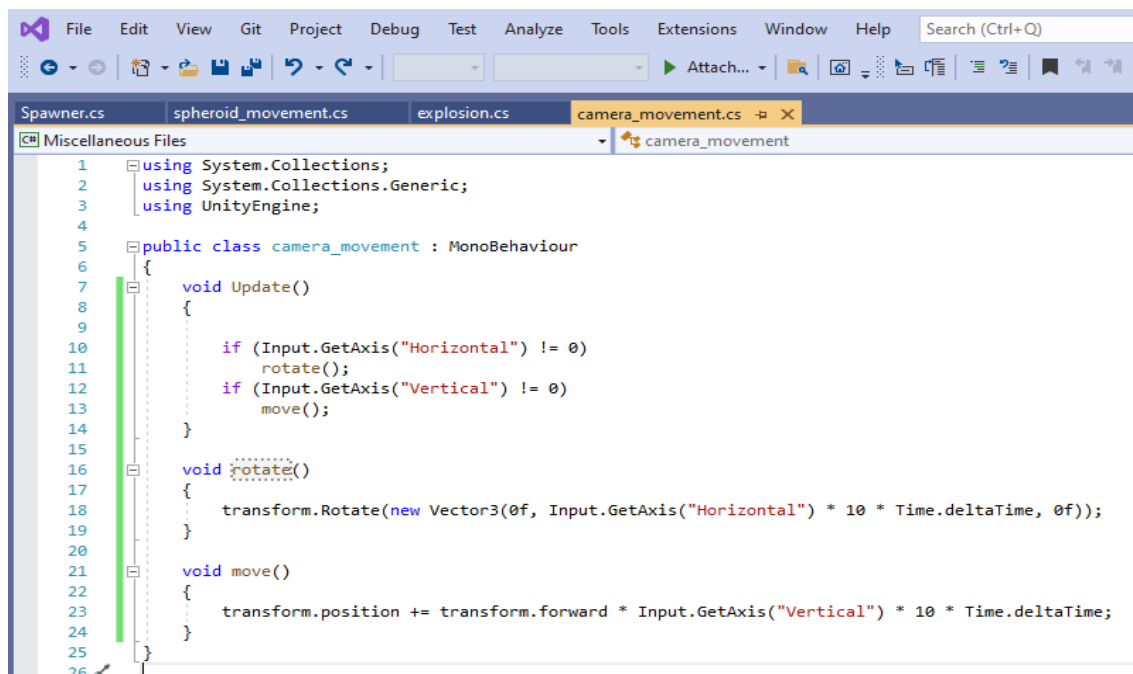
“Vector3 randomSpawnPosition = new Vector3(Random.Range(-296,463), Random.Range(600,1000), Random.Range(-304,418));” είναι υπεύθυνες για το spawn των σφαιρών. Δημιουργούμε κάθε φορά που πατιέται το space, κλώνους της αρχικής μας σφαίρας, σε τυχαίες θέσεις μαζί με την εντολή “Instantiate(myObjects[randomIndex], randomSpawnPosition, Quaternion.identity); “

Η δημιουργία των κλώνων παρόλα αυτά γίνεται σε σημεία με τυχαίο x,y,z που είναι όμως μέσα στα όρια που ορίζει η εργασία, δηλαδή:

Για τον x άξονα: [-296,463], για τον y άξονα: [600,1000] όπως ζητάει η εκφώνηση και για τον z άξονα: [-304,418].

(iv) Στο 4ο ερώτημα της εργασίας φτιάξαμε μια απλή κάμερα η οποία ελέγχεται από το χρήστη με τα βελάκια του πληκτρολογίου για κίνηση στους άξονες x και z του συστήματος παγκοσμίων συντεταγμένων. Δεν καταφέραμε παρόλα αυτά να πραγματοποιήσουμε την κίνηση στον y άξονα με τα πλήκτρα <+>/<->.

Ο κωδικός είναι ο εξής:



```
1 using System.Collections;
2 using System.Collections.Generic;
3 using UnityEngine;
4
5 public class camera_movement : MonoBehaviour
6 {
7     void Update()
8     {
9
10         if (Input.GetAxis("Horizontal") != 0)
11             rotate();
12         if (Input.GetAxis("Vertical") != 0)
13             move();
14     }
15
16     void rotate()
17     {
18         transform.Rotate(new Vector3(0f, Input.GetAxis("Horizontal") * 10 * Time.deltaTime, 0f));
19     }
20
21     void move()
22     {
23         transform.position += transform.forward * Input.GetAxis("Vertical") * 10 * Time.deltaTime;
24     }
25 }
26
```

(Εικόνα 7. camera_movement.cs)

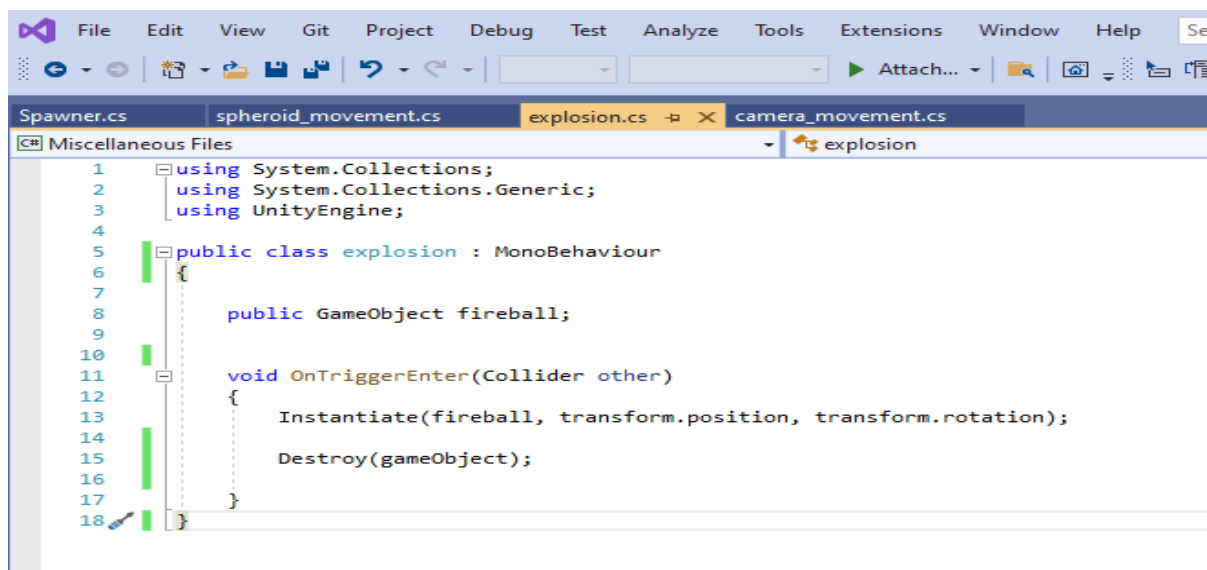
Στον εξής κώδικα, ως παράμετροι των if, μπήκαν αντίστοιχα οι δυο άξονες x και z. Στην πρώτη if καλούμε την συνάρτηση “rotate();” που σημαίνει πως στον άξονα x η κίνηση θα είναι καταμήκος του μέσω rotation, ενώ παράλληλα στην δεύτερη if, καλούμε την “move()”, η οποία θα δημιουργήσει κίνηση.

Έπειτα αναλύουμε ακριβώς τις συναρτήσεις move(), rotate(). Η πρώτη, με την χρήση της “transform.Rotate(new Vector3(0f, Input.GetAxis("Horizontal") * 10 * Time.deltaTime, 0f));” διασφαλίζει την περιστροφή της κάμερας κατά Time.deltaTime, χρόνος πολλαπλασιασμένος επι 10 για να είναι πιο εύχρηστη η εμπειρία του παίκτη. Η δεύτερη, με την χρήση της “transform.position += transform.forward * Input.GetAxis("Vertical") * 10 * Time.deltaTime;” προσθέτει κάθε φορά την ανίχνευση διαφορετικού z και έτσι συνολικά έχουμε την επιθυμητή μας κίνηση, ισχύοντας το ίδιο για το “10 * Time.deltaTime”.

(v) Τελευταίο ερώτημα της εργασίας μας είναι η δημιουργία έκρηξης όταν μια μπάλα φωτιάς συγκρουστεί με το έδαφος αλλά και με το διαστημόπλοιο. Στην πρώτη περίπτωση, έπρεπε να υπάρχει έκρηξη και έπειτα η μπάλα να εξαφανίζεται και στην δεύτερη έπρεπε μετά την έκρηξη να εξαφανίζονται και η μπάλα και το σκάφος.

Το εφε έκρηξης το χρησιμοποιήσαμε από την επιλογή εφε “particle system” το οποίο το δημιουργήσαμε και το ρυθμίσαμε.

Ο κώδικας για την έκρηξη:



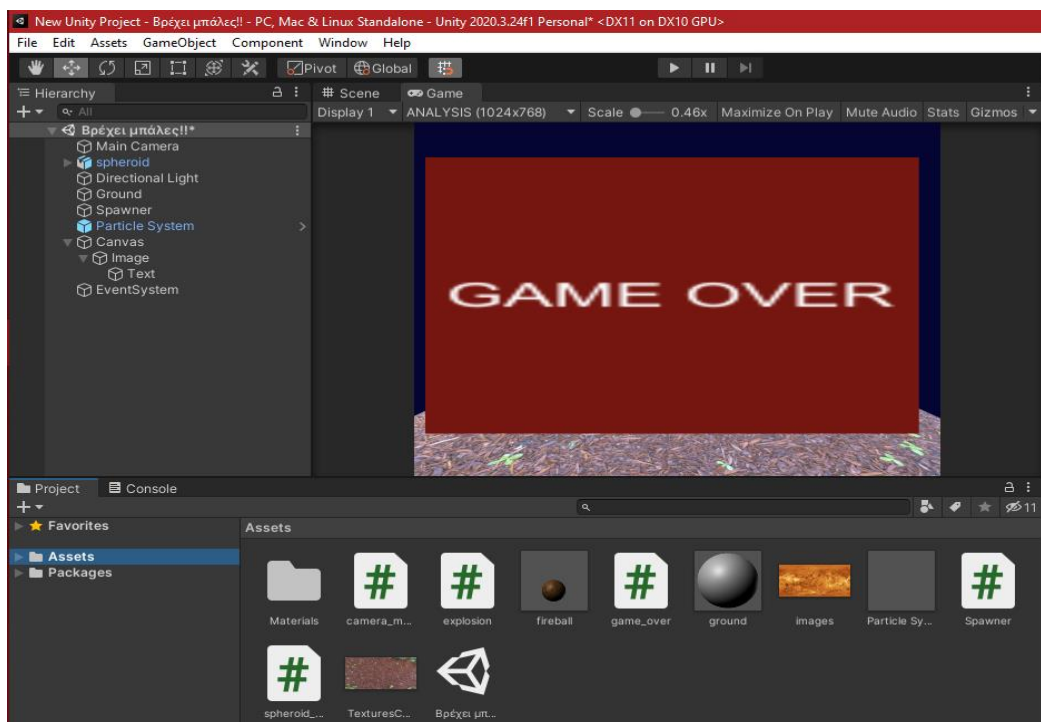
```
1 using System.Collections;
2 using System.Collections.Generic;
3 using UnityEngine;
4
5 public class explosion : MonoBehaviour
6 {
7
8     public GameObject fireball;
9
10
11     void OnTriggerEnter(Collider other)
12     {
13         Instantiate(fireball, transform.position, transform.rotation);
14
15         Destroy(gameObject);
16
17     }
18 }
```

(Εικόνα 8. explosion.cs)

Ο κωδικός μας δεν πληρεί τελείως τις προϋποθέσεις που ζητούνται. Για την ακρίβεια, αρχίζοντας το παιχνίδι, πατώντας χ φορές space, δημιουργούνται χ κλώνοι σφαιρών που πέφτουν κατακόρυφα. Μόλις όμως, η πρώτη σφαίρα ακουμπήσει το έδαφος το παιχνίδι σταματάει. Το παιχνίδι αντιλαμβάνεται ότι η στιγμή της επαφής μπάλας-εδάφους είναι trigger moment αλλά δεν αντιδράει όπως θα θέλαμε.

BONUS:

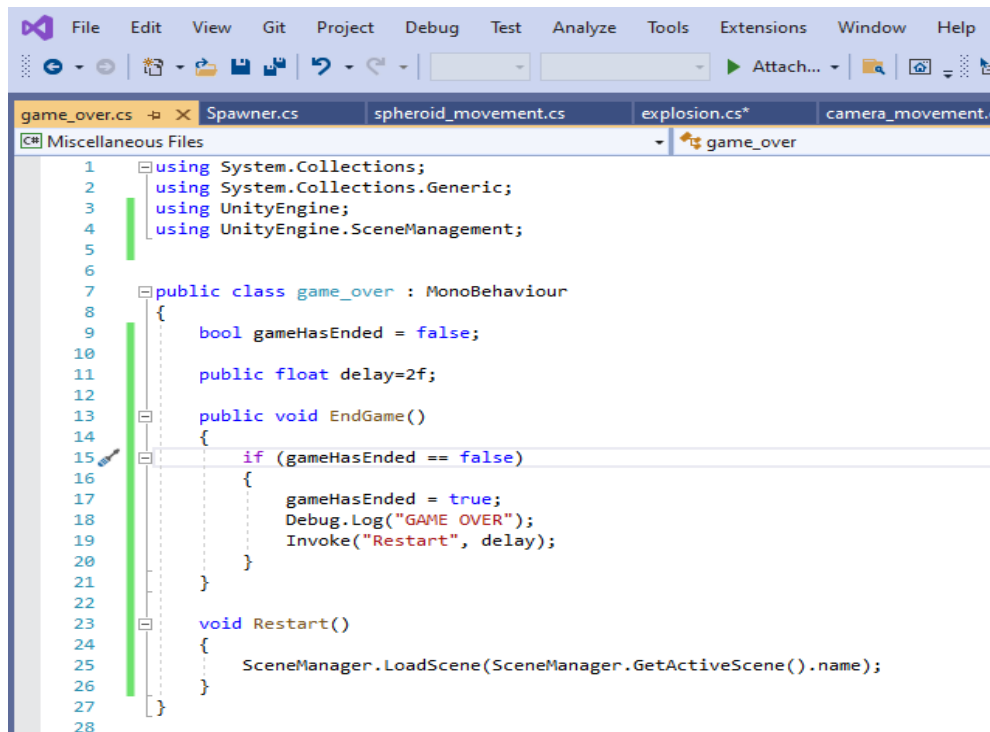
Δοκιμάσαμε να ακολουθήσουμε και ένα απτα bonus ερωτήματα και συγκεκριμένα αυτό με την ένδειξη GAME OVER όταν τελειώνει το παιχνίδι.



(Εικόνα 9. GAME OVER text)

Δημιουργήσαμε ένα Image, επιλέξαμε το χρώμα του και προσθέσαμε το text “GAME OVER” σε αυτό. Δυστυχώς εμφανίζεται συνέχεια όσο παίζεται το παιχνίδι και όχι μόνο στο τέλος όπως θα έπρεπε κανονικά. Θα το παραθέσουμε μαζί με το αρχείο του project αλλά όχι μέσα για να μην εμποδίζει το παιχνίδι.

Ο κωδικός που προσπαθήσαμε να φτιάξουμε ανεπιτυχώς είναι ο εξής:



```
1 using System.Collections;
2 using System.Collections.Generic;
3 using UnityEngine;
4 using UnityEngine.SceneManagement;
5
6
7 public class game_over : MonoBehaviour
8 {
9     bool gameHasEnded = false;
10
11     public float delay=2f;
12
13     public void EndGame()
14     {
15         if (gameHasEnded == false)
16         {
17             gameHasEnded = true;
18             Debug.Log("GAME OVER");
19             Invoke("Restart", delay);
20         }
21     }
22
23     void Restart()
24     {
25         SceneManager.LoadScene(SceneManager.GetActiveScene().name);
26     }
27 }
28
```

(Εικόνα 10. game_over.cs)

Ο κώδικας αυτός στην ουσία αντιλαμβάνεται τότε τελειώνει το παιχνίδι, βγάζει το μήνυμα “GAME OVER”, περιμένει 2 δευτερόλεπτα και ξαναξεκινάει το παιχνίδι απτην αρχη.

ΠΛΗΡΟΦΟΡΙΕΣ ΣΧΕΤΙΚΑ ΜΕ ΤΗΝ ΥΛΟΠΟΙΗΣΗ

Το λειτουργικό σύστημα στο οποίο διεκπονήθηκε αυτή η εργασία είναι τα Windows10

Η έκδοση Unity3d είναι η εξής: 2020.3.24f1

Md5 checksum: 0F29B6BA2C659CDB381871DAF154CF4B

Project link- Drive:

https://drive.google.com/file/d/16lcrV49NVgpj9ZcAl3GhFSjTfgmWmwBA/view?usp=share_link

ΣΥΝΤΟΜΗ ΑΞΙΟΛΟΓΗΣΗ ΛΕΙΤΟΥΡΓΙΑΣ ΟΜΑΔΑΣ

Και τα δύο μέλη της ομάδας δουλέψαμε μαζί και ισάξια τόσο για τους κώδικες του κάθε ερωτήματος όσο και για τις ιδέες και την συνεισφορά.

ΑΝΑΦΟΡΕΣ ΚΑΙ ΠΗΓΕΣ ΠΟΥ ΧΡΗΣΙΜΟΠΟΙΗΘΗΚΑΝ

Τεράστια βοήθεια ήταν το ίδιο το site της Unity3d απτο οποίο καταφέραμε να εντοπίσουμε πληροφορίες για όποια εντολή ή συνάρτηση θέλαμε, αλλά και εξίσου μεγάλη βοήθεια ήταν το Stackoverflow στο οποίο βρίσκαμε απαντήσεις σε διάφορα ερωτήματα που προέκυπταν Realtime κατά την προσπάθειά μας.

