Scripting σε Unity

Μέρος 2ο

Lerp

- Ο όρος lerp προκύπτει από την έννοια του linear interpolation (γραμμική παρεμβολή), και αφορά στη λειτουργίας ομαλής μετάβασης από μια τιμή σε μια άλλη.
- Εντός του API της Unity συναντάται σε διάφορες μορφές οι οποίες έχουν γενικά την εξής μορφή:
  - Lerp(τιμή1, τιμή2, τιμή παρεμβολής)
- Η τιμή παρεμβολής είναι ένας πραγματικός αριθμός από το 0 εώς το 1. Όταν η τιμή αυτή είναι στο 0, η συνάρτηση Lerp θα επιστρέψει την πρώτη τιμή ενώ όταν η τιμή αυτή είναι στο 1 θα επιστρέψει τη δεύτερη τιμή. Αν η τιμή είναι στο 0.5 θα επιστρέψει τη μεσαία τιμή ανάμεσα στις δύο τιμές. Αντίστοιχα λειτουργεί και για οποιαδήποτε άλλη τιμή παρεμβολής.

Lerp

- Οι συχνές μορφές στις οποίες θα συναντάται η συνάρτηση Lerp είναι οι εξής:
  - Mathf.Lerp(float a, float b, float t);
  - Vector2/3.Lerp(Vector2/3 a, Vector2/3 b, float t);
  - Quaternion.Slerp(Quaternion a, Quaternion b, float t);

- Video tutorial για χρήση των συναρτήσεων Mathf.Lerp και Vector3.Lerp: Modulating values with Lerp - Unity Official Tutorials

Coroutines

- Τα coroutines αποτελούν ειδικές περιπτώσεις συναρτήσεων που έχουν την ιδιότητα να εκτελούνται ανά διαστήματα.
- Ενώ η κλήση μιας απλής συνάρτησης από την update θα προϋποθέτει πως όλη η συνάρτηση θα πρέπει να εκτελεστεί σε ένα frame, ένα coroutine μπορεί σε οποιοδήποτε σημείο του σώματός του να επιστρέψει τις αλλαγές που έχουν γίνει και να συνεχίσει την εκτέλεσή του στο επόμενο frame.
- Λόγω της ιδιότητας αυτής, ένα coroutine μπορεί να θεωρηθεί πως εκτελείται ταυτόχρονα με την συνάρτηση Update().

Coroutines

- Ο τρόπος με τον οποίο σταματάει προσωρινά την εκτέλεσή του ένα coroutine είναι χρησιμοποιώντας την εξής γραμμή:
  - yield return null;
- Εφόσον εκτελεστεί αυτή η γραμμή, ό,τι αλλαγή έχει γίνει στο σώμα του coroutine μέχρι εκείνη τη γραμμή θα καταχωρηθούν και ο υπόλοιπος κώδικας θα εκτελεστεί στο επόμενο frame.
- Για να σταματήσει η εκτέλεση ενός coroutine χρησιμοποιείται η εξής γραμμή:
  - yield break;

Coroutines

- Μέσα από coroutines δίνεται επίσης η δυνατότητα εκτέλεσης κώδικα μετά το πέρας κάποιου συγκεκριμένου χρονικού διαστήματος.
- Αυτό γίνεται μέσω της εξής γραμμής:
  - yield return new WaitForSeconds(seconds);

όπου seconds ένα float όρισμα το οποίο προσδιορίζει μετά από πόσα δευτερόλεπτα θα συνεχιστεί η εκτέλεση του coroutine.

Coroutines

- Ο τύπος επιστροφής ενός coroutine είναι IEnumerator, που αποτελεί τη κλάση που επιτρέπει τη χρήση της λειτουργίας yield.
- Αν και είναι συναρτήσεις, τα coroutines δεν καλούνται με τον παραδοσιακό τρόπο αλλά μέσω της συνάρτησης: StartCoroutine(<όνομα_coroutine>(<ορίσματα>));
- Αντίστοιχα, υπάρχει η StopCoroutine που παίρνει σαν όρισμα ένα string με το όνομα ενός coroutine και σταματάει την εκτέλεσή του.
- Τέλος, υπάρχει η συνάρτηση StopAllCoroutines() που σταματάει την εκτέλεση όλων των coroutines της τρέχουσας κλάσης.

Coroutines

- Παράδειγμα χρήσης:

```
public Vector3 newPos;
void Start() {
        StartCoroutine(LerpCoroutine(newPos));
}
```

Coroutines

```
IEnumerator LerpCoroutine(Vector3 pos) {
        float tParam = 0;
        Vector3 orPos = transform.position;
        while(tParam < 1) {
                tParam += Time.deltaTime;
                transform.position = Vector3.Lerp(orPos, pos, tParam);
                yield return null;
        }
        yield return new WaitForSeconds(3.5f);
        print("Lerping is done.");
}
```

Coroutines

- Video tutorial πάνω στη χρήση coroutines: Coroutines - Unity Official Tutorials

Συντεταγμένες οθόνης & κόσμου

- Είναι συχνό να θέλουμε να ταυτίσουμε κάποιο σημείο της οθόνης με κάποιο σημείο του κόσμου του παιχνιδιού. Για παράδειγμα, ένα κλικ σε κάποιο σημείο της οθόνης να εμφανίζει κάποιο αντικείμενο μέσα στο παιχνίδι.
- Προφανώς, ωστόσο, η οθόνη έχει διαφορετικό σύστημα συντεταγμένων σε σχέση με τον κόσμο του παιχνιδιού, και μάλιστα μπορεί να αλλάζει διαρκώς αυτό εφόσον αλλάζει η ανάλυση του παραθύρου.

Συντεταγμένες οθόνης & κόσμου

- Συγκεκριμένα, σε συντεταγμένες οθόνης, η κάτω αριστερή γωνία αντιστοιχεί στο σημείο (0,0) ενώ η πάνω δεξιά γωνία αντιστοιχεί στο σημείο (Screen.width, Screen.height), όπου Screen.width/height πεδία που περιγράφουν τον αριθμό pixel στο πλάτος και ύψος της οθόνης αντίστοιχα.
- Για παράδειγμα, σε μια 1080p ανάλυση, το Screen.width θα είναι 1920 ενώ το Screen.height θα είναι 1080.

Συντεταγμένες οθόνης & κόσμου

- Μέσα από το API της Unity δίνεται η δυνατότητα να ανακτήσουμε τη θέση του ποντικιού σε συντεταγμένες οθόνης. Αυτό γίνεται μέσα από το Vector3 πεδίο:

  Input.mousePosition

- Επειδή οι συντεταγμένες οθόνης και κόσμου συνδέονται μέσα από τη κύρια κάμερα προβολής, στην αντίστοιχη κλάση της κάμερας βρίσκονται οι συναρτήσεις που μετατρέπουν ένα σημείο στην οθόνη ως σημείο στο κόσμο.

Συντεταγμένες οθόνης & κόσμου

- Η συναρτήσεις που επιτρέπουν την αντιστοίχηση των διαφορετικών συστημάτων συντεταγμένων είναι οι εξής:
  - ScreenToWorldPoint(Vector3 position)
  - WorldToScreenPoint(Vector3 position)
- Οι συναρτήσεις αυτές ανήκουν στην κλάση Camera και δεν είναι static, καθώς χρειάζονται κάποιο αντικείμενο με component Camera βάσει του οποίου θα κάνουν την αντιστοίχηση.
- Επειδή συχνά υπάρχει μία MainCamera στη σκηνή, χρησιμοποιείται το static πεδίο Camera.main το οποίο απευθύνεται σε αυτήν.

Συντεταγμένες οθόνης & κόσμου

- Video tutorial πάνω στη χρήση των διαφορετικών συστημάτων συντεταγμένων: Unity Mobile Dev From Scratch: Understanding Screen and World Coordinates for Raycasting

Raycasting

- Οι ακτίνες (rays) έχουν μεγάλη χρησιμότητα στα παιχνίδια καθώς λειτουργούν ως νοητές ευθείες ανάμεσα σε 2 σημεία που αναγνωρίζουν τυχόν εμπόδια στη πορεία τους.
- Χρησιμοποιούνται κυρίως για ανίχνευση αντικειμένων και μπορούν να βρουν πολλές εφαρμογές σε κάθε τύπο παιχνιδιού.

Raycasting

- Η διαδικασία της εκπομπής τέτοιων ακτινών εμπλέκει γενικά τα εξής στοιχεία:
  - Τη κλάση Ray, της οποίας τα αντικείμενα αναπαριστούν μια ακτίνα με προέλευση, κατεύθυνση και, ενδεχομένως, μήκος.
  - Τη κλάση RaycastHit, της οποίας τα αντικείμενα περιέχουν πληροφορίες ως προς τη συνάντηση αντικειμένων με μια ακτίνα.
  - Τη συνάρτηση Physics.Raycast η οποία επιστρέφει εφόσον κάποια ακτίνα συναντήθηκε με κάποιο αντικείμενο.

Raycasting

- Η διαδικασία του raycasting μπορεί να γίνει ως εξής:

```
void Update() {
        Ray ray = new Ray(transform.position,
        transform.forward);
        RaycastHit hit;
        if (Physics.Raycast(ray, out hit)) {
                if (hit.collider != null) { //Hit! }
        }
}
```

Raycasting

- Για 2D περιβάλλον η διαδικασία διαφοροποιείται ως εξής:
  - Χρησιμοποιείται η κλάση RaycastHit2D αντί για την κλάση RaycastHit.
  - Ο έλεγχος γίνεται μέσω της συνάρτησης Physics2D.Raycast αντί της Physics.Raycast. Η Physics2D.Raycast επιστρέφει ένα RaycastHit2D αντί για bool.

Raycasting

- Ένα παράδειγμα της χρήσης raycasting σε 2D περιβάλλον θα ήταν ως εξής:

```
void Update() {
    RaycastHit2D hit = Physics2D.Raycast(transform.position,
    transform.right);
    if (hit.collider != null) { //Hit! }
}
```

- Video tutorial πάνω στη χρήση του 3D raycasting: Raycasting - Unity Official Tutorials

Επικοινωνία μεταξύ scripts

- Πολλές φορές είναι χρήσιμο να υπάρχει η δυνατότητα να κληθεί μια συνάρτηση ενός script μέσα από κάποιο άλλο script. Για να γίνει αυτό υπάρχουν δύο δημοφιλείς τρόποι:
  - Μέσω του component του άλλου script
  - Μέσω ενός public static instance του script

Επικοινωνία μεταξύ scripts

- Για να κληθεί μια συνάρτηση ενός script από άλλο, αρκεί η συνάρτηση να είναι public και το δεύτερο script να μπορεί να αναφερθεί στο αντίστοιχο component.
- Για να μπορεί να αναφερθεί στο component του άλλου script μπορεί να χρησιμοποιήσει κάποιο public πεδίο ή, πιο συχνά, τη συνάρτηση GetComponent.

Επικοινωνία μεταξύ scripts

- Εφόσον το component στο οποίο θέλουμε να αναφερθούμε υπάρχει μόνο μία φορά (π.χ. Ένας game manager) μπορούμε να χρησιμοποιήσουμε ένα public static instance του script.
- Αν το script περιέχει μια κλάση με το όνομα MyClass, το πεδίο αυτό θα έχει την εξής μορφή:

```
public static MyClass instance;
void Awake() {
    instance = this;
}
```

Επικοινωνία μεταξύ scripts

- Αν υπάρχει και έχει αρχικοποιηθεί το πεδίο instance, μπορεί να προσπελαστεί από οποιοδήποτε άλλο script ως εξής:

  MyClass.instance.<πεδίο/μέθοδος>

εφόσον το πεδίο ή η μέθοδος είναι public.

- Επειδή η κλάση MyClass είναι public, τα public static πεδία της είναι ορατά από οποιαδήποτε άλλη κλάση.

Delegates

- Το delegate αποτελεί ειδικό τύπο ο οποίος αποτελεί αναφορά σε μεθόδους με συγκεκριμένο τύπο επιστροφής, πλήθος και τύπο ορισμάτων.
- Η χρήση των delegates μπορεί να κάνει τον κώδικα πολύ δυναμικό και ευέλικτο και μπορούν να βοηθήσουν πάρα πολύ στην οργάνωσή του.
- Εφόσον φτιαχτεί ένα delegate, μπορούν να προστεθούν ή να αφαιρεθούν συναρτήσεις σε αυτό. Αν υπάρχει έστω και μία συνάρτηση στην οποία αναφέρεται το delegate, μπορεί να κληθεί με ίδιο τρόπο όπως μια απλή συνάρτηση.

Delegates

- Παράδειγμα ορισμού delegate:

  delegate void MyDelegate();

- Οποιοδήποτε delegate τύπου MyDelegate παίρνει μόνο αναφορές σε συναρτήσεις με τύπο επιστροφής void και χωρίς καμία παράμετρο.
- Η κατασκευή ενός delegate αυτού του τύπου γίνεται απλά ως εξής:

  MyDelegate myDel;

Delegates

- Έστω οι εξής συναρτήσεις:

```
void foo() {
    print("5");
}
```

```
void bar() {
    print("10");
}
```

- Αφού οι συναρτήσεις foo και bar ταιριάζουν στον τύπο του delegate που έχει οριστεί μπορούν να προστεθούν στο myDel ως εξής:

```
myDel += foo;
myDel += bar;
```

- Με αντίστοιχο τρόπο μπορεί να αφαιρεθεί μια αναφορά από το delegate.

Delegates

- Όλες οι συναρτήσεις που έχουν προστεθεί στο delegate θα εκτελεστούν ως εξής:

```
//If myDel is referencing any functions
if (myDel != null) {
      myDel();
}
```

- Video tutorial σχετικό με τη χρήση delegates: Delegates - Unity Official Tutorials

Lambda expressions

- Ένα lambda expression είναι πρακτικά μία ανώνυμη συνάρτηση που ορίζεται εντός μιας άλλης συνάρτησης.
- Χρησιμοποιείται συχνά για να αποφευχθεί η δημιουργία ξεχωριστής συνάρτησης αν αυτή έχει μικρό σώμα ή πρόκειται να χρησιμοποιηθεί μόνο μια φορά.
- Πιο πολλές φορές τα lambda expressions χρησιμοποιούνται σε συνεργασία με delegates.

Lambda expressions

- Ο ορισμός ενός lambda expression γίνεται με τον εξής τρόπο:

  (<ορίσματα>) => { <σώμα_συνάρτησης}

- Για παράδειγμα, χρησιμοποιώντας το delegate myDel από τα προηγούμενα παραδείγματα θα μπορούσαμε να προσθέσουμε τη λειτουργικότητα της συνάρτησης foo ως εξής:

  myDel += () => { print(“5”); };

Enumerations

- Ένα enumeration αποτελεί τύπο που αντιστοιχεί σε μια συλλογή από σταθερές που συμβολίζονται με κάποια λέξη.
- Χρησιμοποιούνται για οργάνωση και για εύκολο και γρήγορο έλεγχο μεταξύ των διαφορετικών τιμών.
- Παράδειγμα ορισμού enumeration:

```
enum Direction {North, East, South, West};
```

Enumerations

- Αφού έχει οριστεί ο τύπος του enumeration, μπορεί να φτιαχτεί μία αναφορά σε αυτό ως εξής:

```
Direction dir;
```

- Και μπορεί να δοθεί τιμή σε αυτήν την αναφορά έτσι:

```
dir = Direction.North;
```

- Video tutorial πάνω στα enumerations: Enumerations - Unity Official Tutorials

Scene management

- Για εναλλαγή σκηνών εντός του παιχνιδιού χρησιμοποιούνται τα εξής στοιχεία του API της Unity:
  - SceneManager.LoadScene(int/string scene): Παίρνει σαν όρισμα είτε το build index μιας σκηνής είτε το όνομά της και τη φορτώνει.
  - SceneManager.GetActiveScene(): Επιστρέφει ένα αντικείμενο τύπου Scene που αντιστοιχεί στη τρέχουσα σκηνή. Το πεδίο SceneManager.GetActiveScene().buildIndex αντιστοιχεί στο build index της τρέχουσας σκηνής.

Scene management

- Σημειώνεται πως για τη χρήση των παραπάνω συναρτήσεων θα πρέπει να εισαχθεί και το αντίστοιχο namespace μέσω της εξής γραμμής:

  using UnityEngine.SceneManagement;

Animator

- Προκειμένου να τροποποιηθούν οι παράμετροι του animator component χρησιμοποιούνται οι εξής συναρτήσεις της κλάσης Animator πάνω σε κάποιο αντικείμενο (έστω Animator anim):
  - anim.SetFloat(<όνομα_παραμέτρου>, <τιμή>)
  - anim.SetBool(<όνομα_παραμέτρου>, <τιμή>)
  - anim.SetTrigger(<όνομα_παραμέτρου>)
- Ανάλογα με το πώς έχει ρυθμιστεί ο animator του αντικειμένου θα προκύψει διαφορετικό αποτέλεσμα από τις συναρτήσεις αυτές.