A blue and red shield with a white and blue logo

Description automatically generated

**ΠΑΝΕΠΙΣΤΗΜΙΟ ΠΕΙΡΑΙΩΣ**

**ΤΜΗΜΑ ΠΛΗΡΟΦΟΡΙΚΗΣ**

**Τελική Εργασία Μαθήματος**

**ΕΙΚΟΝΙΚΗ ΠΡΑΓΜΑΤΙΚΟΤΗΤΑ**

**Τεχνικό Εγχειρίδιο Εφαρμογής**

**Ουρανία Αργύρα P20023**

**Γεώργιος Χριστόπουλος P20206**

**Επιβλέποντες Καθηγητές : Παναγιωτόπουλος Θεμιστοκλής**

**Τηλέφωνο επικοινωνίας : 6981059764**

**ΠΕΙΡΑΙΑΣ**

**ΙΑΝΟΥΑΡΙΟΣ 2024**

Περιεχόμενα

**Εισαγωγή**

* Δημιουργία Εικονικού Κόσμου ......................................................................................................
* Unity Engine ……………………………….. ..............................................................................................
* Δημιουργία Project ………………......................................................................................................
* Συνοπτική επεξήγηση της εφαρμογής .........................................................................................

**Πηγές των Assets**

* Unity Asset Store …………...............................................................................................................
* Mixamo …………………………………………………………………………………………………………………...................

**Περιγραφή του προβλήματος**

**Αναλυτική Παρουσίαση όλων των φάσεων Ανάπτυξης και Αντιμετώπιση των κριτηρίων που θέτονται**

**Πίνακας με C# scripts και περιγραφή τους**

* Ανάλυση .......................................................................................................................................
* Σχεδιασμός ...................................................................................................................................
* Υλοποίηση ...................................................................................................................................

**Αναλυτική παρουσίαση για χρήστες**

**Εισαγωγή**

Το αντικείμενο της εργασίας μας με βάση την εκφώνηση του μαθήματος Εικονική Πραγματικότητα του 7ου εξαμήνου είναι η δημιουργία ενός 3D χώρου με ένα χωριό της δικής μας έμπνευσης. Αποφασίσαμε να αναπαραστήσουμε ένα δασικό χωριό τρισδιάστατης αναπαράστασης, όπου θα υπάρχει η δυνατότητα διαβίωσης στην απομακρυσμένη δασική περιοχή. Μέσα από αυτόν τον χώρο, αναπαρίσταται η ζωή σε ένα χωριό δικής μας φαντασίας, με πλούσια δάση και βλάστηση, λίμνες ανάμεσα από τα βουνά, απόκρυφα μονοπάτια και εκπληκτικά βουνά που απλώνονται σχεδόν σε κάθε άκρη του αρχικού μας terrain. Το δάσος προσπαθεί να μεταφέρει στον παίκτη ένα συναίσθημα γαλήνης και ηρεμίας, εξερευνώντας κάθε πτυχή του terrain, ενώ παράλληλα μπορεί να μετατραπεί και σε παιχνίδι τρόμου, καθώς, σβήνοντας τα φώτα, ο παίκτης θα νιώθει χαμένος σε ένα εικονικό δάσος. Σκοπός της εργασίας είναι η δημιουργία κάποιων σπιτιών με αρχαίο στυλ αρχιτεκτονικής, καθώς ενσωματώνεται πιο ομαλά στις ανάγκες της εργασίας. Στο δάσος, μπορούν να υπάρχουν, επίσης, στοχευμένες εγκαταστάσεις που θα παράγουν έργο με σεβασμό στο περιβάλλον, διατηρώντας την οικολογική ισορροπία του δάσους. Για παράδειγμα, μπορούν να ενταχθούν εργοστάσια που δραστηριοποιούνται στον τομέα της δασοκομίας και της διαχείρισης των δασών, προσφέροντας την απαραίτητη συντήρηση για την υγιή ανάπτυξη του περιβάλλοντος.

Να σημειωθεί ότι για τη δημιουργία του word και για τη δημιουργία του 3D περιβάλλοντος χρησιμοποιήθηκε η ιστοσελίδα <https://sites.google.com/view/unipivr2023/%CE%BC%CE%B1%CE%B8%CE%AE%CE%BC%CE%B1%CF%84%CE%B1/%CE%BC%CE%AC%CE%B8%CE%B7%CE%BC%CE%B1-1> και οι σημειώσεις του μαθήματος.

**Δημιουργία Εικονικού Κόσμου**

Οι εικονικοί κόσμοι, όπως το Metaverse, είναι περιβάλλοντα προσομοίωσης που βασίζονται σε υπολογιστή, όπου οι χρήστες μπορούν να αλληλεπιδρούν με αυτόν χρησιμοποιώντας κάποιο avatar ή έναν χαρακτήρα με δυνατότητα αναπαραγωγής. Η δημιουργία εικονικών κόσμων γίνονται όλο και πιο εξελιγμένοι με καλύτερες και σαφέστερες εικόνες και πιο ζωντανούς avatar. Αυτό μπορεί εύκολα να διαπιστωθεί συγκρίνοντας παλιές εκδόσεις παιχνιδιών με πιο νέες, όπου τα γραφικά και η εμπιερία είναι σαφώς καλύτερα και πιο κοντά στην πραγματικότητα. (Για παράδειγμα, σε αυτό το βίντεο βλέπουμε την εξέλιξη των γραφικών στα βιντεοπαιχνίδια μεταξύ του 1958 και του 2020 <https://www.youtube.com/watch?v=IsPPWWlV-T8&ab_channel=Cussan> ) Για τη δημιουργία ενός εικονικού κόσμου, μπορούμε να εγγραφούμε σε πλατφόρμα εικονικού κόσμου και να δημιουργήσουμε τον δικό μας χώρο, ακόμη και εάν δεν έχουμε εμπειρία κωδικοποίησης.

**Unity Engine**

Στα μαθήματα που πραγματοποιήθηκαν στο πλαίσιο της διδασκαλίας «Εικονική Πραγματικότητα» χρησιμοποιήθηκε η εφαρμογή Unity, η οποία δίνει τη δυνατότητα στους χρήστες να δημιουργούν παιχνίδια και εμπειρίες τόσο σε 2D όσο και σε 3D, προσφέροντας ένα κύριο API με σύνολο εντολών γραμμένων σε C#, βοηθώντας τον επεξεργαστή Unity με τη μορφή πρόσθετων, τα ίδια τα παιχνίδια, ενώ παράλληλα προσφέρει λειτουργικότητα. Η μηχανή ανάπτυξης 3D σε πραγματικό χρόνο της Unity επιτρέπει σε καλλιτέχνες, σχεδιαστές και προγραμματιστές να συνεργαστούν για να δημιουργήσουν εκπληκτικές και καθηλωτικές εμεπιρίες.

**Δημιουργία Project**

To Unity Hub είναι μία αυτόνομη εφαρμογή που βελτιστοποιεί τον τρόπο πλοήγησης, λήψης και διαχείρισης των έργων και εγκαταστάσεων του Unity. Μπορούμε να χρησιμοποιήσουμε υο Unity Hub για διαχείριση, λήψη και εγκατάσταση λειτουργικών μονάδων και εκδόσεων του Unity Editor. Το πρόγραμμα επεξεργασίας Unity Editor είναι το οπτικό στοιχείο που μας δίνει τη δυνατότητα να δημιουργήσουμε παιχνίδια. Επειδή το μεγαλύτερο μέρος τη αλληλεπίδρασης που έχουμε είναι στην πραγματικότητα με τον επεξεργαστή, πολλοι άνθρωποι το αναφέρουν και ως Unity.

Για το Unity Hub ζητείται να δημιουργηθεί προσωπικός λογαριασμός και personal license (άδεια).

Για τη δημιουργία ενός project, πατάμε στο New Project τoυ Unity Hub, και επιλέγουμε το προτζεκτ 3D URP (University Render Pipeline), και έπειτα download template. Τέλος δίνουμε ένα όνομα στο προτζεκτ, επιλέγουμε σε ποιον φάκελο θα αποθηκευτεί και πατάμε Create Project.

**Συνοπτική επεξήγηση της εφαρμογής**

Όταν ανοίξει το πρότζεκτ μας, μπορούμε πάνω αριστερά να παρατηρήσουμε την ιεραρχία της εφαρμογής **(hierarchy)**, όπου βρίσκονται τα αντικείμενα του χώρου σε μορφή λίστας (πιθανόν να έχουν και σχέση γονέα – παιδιού). Οι σχέσεις μεταξύ των αντικειμένων είναι ένα προς ένα (1-1), δηλαδή ό,τι υπάρχει στη λίστα, υπάρχει και στον τρισδιάστατο χώρο. Μπορούμε κάποια από αυτά να τα ορίσουμε και ως αόρατα, οπότε στο χώρο δε θα φαίνονται, αλλά στο hierarchy θα υπάρχουν κανονικά. Με δεξί κλικ πάνω σε ένα αντικείμενο, μπορούμε να δούμε τα properties (ιδιότητες του κάθε αντικειμένου), τα οποία εμφανίζονται στον inspector.

* Πώς μπορούμε να αλλάξουμε ιδιότητες σε ένα αντικείμενο και να αλλάξουν αυτόματα οι ίδιες ιδιότητες σε παρόμοια αντικείμενα;

Με τη διαδικασία **prefabricated** (πρωτότυπο), όπου σε έναν ειδικό τύπο αρχείου κρατάμε το πρωτότυπο αντικείμενο και όλα τα υπόλοιπα που πηγάζουν από αυτό, στην ουσία είναι σαν αντίγραφα. Αν αλλάξει η ιδιότητα στα αντίγραφα, δεν αλλάζει στο πρωτότυπο.

Το default στον inspector μας επιτρέπει να επιστρέψουμε στις αρχικές ιδιότητες του αντικειμένου.

Αντικείμενο **Game Object**: Είναι θεμελιώδη αντικείμενα που αντιπροσωπεύουν χαρακτήρες και σκηνικά. Συνήθως, λειτουργούν ως δοχεία για components, τα οποία υλοποιούν πραγματική λειτουργικότητα. Ο **inspector** διαθέτει **components**. Ένα από αυτά είναι το **transform**. Έχει διανύσματα-συντεταγμένες (οι default τιμές είναι 0) και την κλίμακα 1. Επίσης, το scale έχει lock που αλλάζουν όλα τα x,y και γίνονται ίδια. Με το **add component** μπορούμε να προσθέσουμε λογική στο αντικείμενο (ήδη φτιαγμένος κώδικας-κλάσεις). Κάτω στην εφαρμογή, υπάρχει το **Project**, το οποίο λειτουργεί σαν file explorer. Δίπλα, το παράθυρο **console** εμφανίζει τα errors της εφαρμογής.

A screenshot of a computer program

Description automatically generatedA screenshot of a computer

Description automatically generated

Εργαλεία **querty**: Το πρώτο είναι απλά για να κοιτάμε το χώρο, το δεύτερο είναι το move (εργαλείο μετακίνησης - position), το τρίτο είναι το rotation (περιστροφή αντικειμένου), το τέταρτο είναι για την κλίμακα-scale, δηλαδή καθορίζει το μέγεθος το αντικειμένου, το πέμπτο χρησιμεύει κυρίως σε αντικείμενα δύο διαστάσεων, ενώ το τελευταίο τα διαμορφώνει όλα μαζί.

Κάνοντας δεξί κλικ στην ιεραρχία, βλέπουμε κάποια ήδη φτιαγμένα αντικείμενα, τα οποία ονομάζονται **primitive**. Στα components έχουμε τη φυσική υπόσταση του αντικειμένου (**capsule collider**) και το **mesh renderer – mesh filter**, το οποίο είναι η εμφάνιση του αντικειμένου.

Για να τρέξουμε την εφαρμογή πατάμε πάνω το βελάκι με το σήμα **play**.

**Freeze position**: Με αυτή την επιλογή, μπορούμε να «ακινητοποιούμε» τις συντεταγμένες της θέσης ενός αντικειμένου στο χώρο 3D, δηλαδή παραμένουν σταθερές και δε μπορούν να τροποποιηθούν κατά τη διάρκεια της εκτέλεσης του παιχνιδιού. Αυτό είναι χρήσιμο σε περιπτώσεις που θέλουμε να διατηρήσουμε ένα αντικείμενο σε μία συγκεκριμένη θέση στον κόσμο του παιχνιδιού, χωρίς να μπορεί να μετακινηθεί από τον παίκτη ή από άλλες δυνάμεις.

**Rigidbody**: Συστατικό που χρησιμοποιείται για την προσωμοίωση της φυσικής συμπεριφοράς αντικειμένων στο παιχνίδι. Το αντικείμενο αποκτά τη δυνατότητα να αλληλεπιδρά με την κίνηση και τις φυσικές δυνάμεις στον κόσμο του παιχνιδιού. Κύρις λειτουργίες του μπορεί να είναι η δυναμική κίνηση, η ανίχνευση συγκρούσεων και η προσομοίωση φυσικής.

**Object Parenting** : Τα γονικά αντικείμενα είναι ο τρόπος με τον οποίο μπορούμε να συσχετίσουμε προσωρινά αντικείμενα με άλλα αντικείμενα. Ένα αντικείμενο μπορεί να έχει ένα γονικό αντικείμενο, δηλαδή του έχει εκχωρηθεί ένας γονέας και μπορεί να μοιραστεί με αυτόν κώδικα, ενέργειες και συμβάντα (σύστημα ιεραρχίας). Αυτή η κοινή χρήση ονομάζεται «κληρονομι;a» και ένα αντικείμενο που έχει γονέα ονομάζεται αντικείμενο «παιδί».

**Nesting objects** : Το ένα αντικείμενο βρίσκεται μέσα στο άλλο.

**Πηγές των assets**

**Unity Asset Store**

Τα αντικείμενα που χρησιμοποιήθηκαν στην εργασία υπάρχουν στην ιστοσελίδα Unity Asset Store <https://assetstore.unity.com/> . Στην ιστοσελίδα, μπορούμε να βρούμε αρκετά δωρεάν assets και να τα προσθέσουμε στο λογαριασμό μας στο Unity. Κάποια από αυτά ίσως να περιέχουν και ExampleScenes, δηλαδή σκηνές που έχει φτιάξει ο δημιουργός με τα asset.

Ένας render pipeline εκτελεί μία σειρά λειτουργιών που λαμβάνουν το περιεχόμενο μιας σκηνής. Το render pipeline που έχει χρησιμοποιηθηκε για την εργασία είναι το URP (Universal Render Pipeline), το οποίο χρησιμοποιείται για τη βελτιστοποίηση της απόδοσης των γραφικών σε όλες τις πλατφόρμες. Κάποια από τα αντικείμενα που προσθέτουμε μπορεί να μη χρησιμοποιούν URP (εμφανίζονται με ροζ χρώμα στη σκηνή). Για να δούμε αν μπορούμε να το μετατρέψουμε σε URP πατάμε πάνω στο αντικείμενο και έπειτα στο Window -> Rendering -> Render Pipeline Converter, κάνουμε material update και Built-i to URP και μετά Convert.

A screenshot of a computer

Description automatically generated

Επίσης, στην εφαρμογή μας από το Unity Asset Store προσθέτουμε έναν προφτιαγμένο χαρακτήρα ThirdPersonController, με τον οποίο θα μπορούμε να δούμε τη σκηνή, μέσω της κάμερας που θα τον ακολουθεί.

**Mixamo**

Το mixamo δίνει τη δυνατότητα στους χρήστες να ζωντανέψουν τρισδιάστατους χαρακτήρες για παιχνίδια, ταινίες και άλλα (<https://www.mixamo.com/#/> ). Ενισχύει τη δημιουργηκότητα με κινούμενους τρισδιάστατους χαρακτήρες έτοιμους για χρήση.

A screenshot of a computer

Description automatically generated

Μέσω του mixamo μπορούμε να επιλέξουμε χαρακτήρες και κινήσεις που θέλουμε να κάνουν, καθώς επίσης και επιλογές για το πόσο θέλουμε να έχουν ανοιχτά τα χέρια τους (character arm space), αν θέλουμε την κίνηση ολόκληρη ή από κάποιο χρονικό διάστημα (trim) και μετά και πόσο γρήγροα θέλουμε να κινείται ο χαρακτήρας (overdrive). Κατεβάζουμε τους χαρακτήρες σε fbx for unity, with skin, 30 frames per second and none keyframe reduction. Κατεβαίνει αρχείο με κατάληξη .fbx.

A screenshot of a computer

Description automatically generated

Οι χαρακτήρες λειτουργούν με controllers. Ανάλογα με τον controller που θα βάλουμε σε κάθε φιγούρα θα κάνει και την αντίστοιχη κίνηση. Αν θέλουμε μία φιγούρα να κάνει διαφορετικές κινήσεις, χρησιμοποιούμε διαφορετικούς controllers.

**Περιγραφή του προβλήματος**

Για την περιγραφή του προβλήματος, αρχικά θα πρέπει να περιγράψουμε την έννοια της εικονικής πραγματικότητας. Η εικονική πραγματικότητα (Virtual Reality) είναι ένα περιβάλλον που δημιουργείται από τον υπολογιστή με σκηνές και αντικείμενα που φαίνονται αληθινά, παροτρύνοντας το χρήστη να ενταχθεί σε αυτό το περιβάλλον.

Η περιγραφή του προβλήματος είναι συχνά το πρώτο βήμα για την έναρξη ενός έργου. Στη δική μας περίπτωση, η περιγραφή μπορεί να μας βοηθήσει να κατανοήσουμε καλύτερα τη σημασία του προβλήματος.

Στο πλαίδιο της εργασίας, έχουμε το πρόβλημα της δημιουργίας ενός σύγχρονου χωριού μέσω της εφαρμογής Unity που θα προσφέρει στο χρήστη μοναδική, ρεαλιστική και απολαυστική εμπειρία εξερεύνησης και αλληλεπίδρασης. Αυτό απαιτεί αρκετές διαδικασίες, όπως η δημιουργία μοντέλων και αντικειμένων που θα συνθέτουν το χωριό, η εφαρμογή του φωτισμού, των υλικών και των κινήσεων που θα προσδώσουν στο παιχνίδι ρεαλισμό, αληθοφάνεια και λειτουργικότητα. Υπάρχει, δηλαδή, ανάγκη για ρεαλιστική απεικόνιση των αντικειμένων και του περιβάλλοντος, για τη διαχείριση του φωτισμού (για παράδειγμα μπορούμε να ορίσουμε τη διάρκεια της ημέρας 20 λεπτά και να διαχειριστούμε ανάλογα το φωτισμό μες σκοπό να δημιουργηθεί μία πιο φυσική αίσθηση) καθώς και για την αποδοτική αντιμετώπιση της αλληλεπίδρασης του χρήστη με το περιβάλλον (στο οποίο πιθανόν να απαιτείται κώδικας). Στο σχεδιασμό, θα πρέπει να υπάρχει ακρίβεια, δημιουργικότητα και λεπτομέρεια, όπως επίσης και καλή αισθητική και δομή.

Για το σχεδιασμό της εφαρμογής και εφόσον θα υλοποιηθεί σύγχρονο χωριό, θα πρέπει να χρησιμοποιηθούν προϋπάρχοντα ή να δημιουργηθούν μοντέλα και αντικείμενα, όπως ξύλινα σπιτάκια, λουλούδια, δένδρα, λιμνούλες, ζωάκια κτλ. Αυτά τα αντικείμενα, θα πρέπει να τοποθετηθούν στο χώρο με σωστή δομή και προστεθούν τα απαραίτητα υλικά και εφέ, καθώς και ο απαραίτητος φωτισμός. Θα πρέπει να ληφθεί υπόψη η αλληλεπίδραση του χρήστη με το χωριό, όπως για παράδειγμα η δυνατότητα περιηγήσεώς του σε αυτό, η δυνατότητα αλληλεπίδρασής του με αντικείμενα (όπως για παράδειγμα το άνοιγμα της πόρτας στα σπιτάκια) και η αλληλεπίδρασή του με άλλους χαρακτήρες (όπως για παράδειγμα χαιρετισμός όταν φτάνουμε σε συγκεκριμένη απόσταση από αυτόν). Είναι λογικό ότι η περιγραφή του προβλήματος δεν αναφέρεται μόνο στην υλοποίηση ενός περιβάλλοντος, αλλά και στις προκλήσεις που προκύπτουν κατά τη διαδικασία δημιουργίας του.

Παραπάνω αναφέρθηκαν και αναλύθηκαν κάποια από τα εργαλεία που χρησιμοποιούνται στην εφαρμογή Unity για τη δημιουργία ενός εικονικού περιβάλλοντος, καθώς και κάποιες τεχνικές οι οποίες εφαρμόζονται για την επίλυση συγκεκριμένων προβλημάτων. Για την δημιουργία του χώρου μας, θα πρέπει να αναητάμε συνεχώς πιθανές τεχνικές λύσεις για τις προκλήσεις που προκύπτουν (erros, render pipline converter κτλ.)

**Αναλυτική παρουσίαση όλων των φάσεων ανάπτυξης και αντιμετώπιση των κριτηρίων**

Στο συγκεκριμένο βήμα της εργασίας, ζητείται να κάνουμε μία αναλυτική παρουσίαση όλων των φάσεων ανάπτυξης του κύκλου ζωής της εφαρμογής. Η σωστή αντιμετώπιση ενός έργου οφείλεται σε μια σειρά βημάτων τα οποία πρέπει να ακολουθηθούν για να επιτευχθούν τα επιθυμητά αποτελέσματα. Για τον κύκλο ζωής ανάπτυξης ενός παιχνιδιού διατίθεται ένα σύνολο 3 φάσεων για την αποτελεσματική ανάπτυξή του.

Αυτό περιλαμβάνει την **ανάλυση** των απαιτήσεων του έργου, το **σχεδιασμό** της αρχιτεκτονικής της εφαρμογής, των δοκιμών στοιχείων, των διεπαφών και της λειτουργικότητας της εφαρμογής και τέλος την **υλοποίηση**, δηλαδή την πραγματική κατασκευή της εφαρμογής.

**Ανάλυση**

Κατά την ανάλυση του παιχνιδιού, προσπαθούμε να απαντήσουμε στις εξής ερωτήσεις: Ποιος θα χρησιμοποιήσει το παιχνίδι, για ποιο σκοπό δημιουργείται και πώς θα χρησιμοποιηθεί.

Αναφέραμε και παραπάνω, ότι το παιχνίδι αναπτύσσεται με σκοπό τη διασκέδαση και για να δώσει μία μοναδική εμπειρία στο χρήστη. Διατίθεται, κυρίως για προσωπική χρήση και διασκέδαση του χρήστη. Το χωριό που θα δημιουργηθεί μπορεί να εξυπηρετήσει πολλούς και διαφορετικούς σκοπούς. Θα μπορούσε να χρησιμοποιηθεί για περιβαλλοντικούς σκοπούς, αφού στόχος μας είναι να προσθέσουμε αρκετή βλάστηση, βουνά και πεδιάδες. Αυτά μπορούν να ανακαλυφθούν από το χρήστη, το οποίο προσδίδει μία ανυπομονησία στο χρήστη και περιέργεια. Στο χωριό, επίσης, θα μπορούσαμε να προσθέσουμε αλληλεπίδραση μεταξύ των χαρακτήρων.

Αρχικά, θα πρέπει να προσδιοριστεί η στρατηγική της ανάλυσης του παιχνιδιού που θα πρέπει να ακολουθηθεί, καθώς και κάποιοι τρόποι σχεδιασμού του παιχνιδιού. Για την υλοποίηση αυτού του βήματος, συγκεντρώσαμε κάποιες πληροφορίες από τις διαλέξεις και από το διαδίκτυο (οι οποίες αναφέρονται παρακάτω), με σκοπό να δημιουργηθεί η κεντρική ιδεά του παιχνιδιού από την οποία παράγεται και το μοντέλο ανάλυσης. Το μοντέλο ανάλυσης περιγράφει τον τρόπο λειτουργίας του παιχνιδιού. Τέλος, συνδέουμε την κεντρική ιδέα και το μοντέλο ανάλυσης για να προσωρήσουμε στο σχεδιασμό.

Οι απαιτήσεις ενός έργου αναφέρονται στις λειτουργικές και μη λειτουργικές ανάγκες και προδιαγραφές που πρέπει να πληροί η τελική εφαρμογή. Αυτές οι απαιτήσεις καθορίζουν το πεδίο του έργου και ορίζουν τις ανάγκες και τις προσδοκίες για την τελική εφαρμογή. Κατά τη διαδικασία ανάπτυξης ενός έργου, η κατανόηση και η καταγραφή αυτών των απαιτήσεων είναι κρίσιμη για την επιτυχή ολοκλήρωση του έργου.

Οι απαιτήσεις αυτές μπορούν να χωριστούν σε δύο βασικές κατηγορίες:

* Λειτουργικές απαιτήσεις : Σε αυτή την κατηγορία συμπεριλαμβάνονται οι λεπτομερείς περιγραφές των λειτουργιών και των δυνατοτήτων της εφαρμογής. Στο παιχνίδι που υλοποιούμε οι λειτουργικές απαιτήσεις περιλαμβάνουν τον τρόπο κίνησης των χαρακτήρων, τον τρόπο επικοινωνίας μεταξύ τους, τα επίπεδα και τους στόχους του παιχνιδιού, δυνατότητα αλληλεπίδρασης του χρήστη με το περιβάλλον, τη προβολή ρεαλιστικών γραφικών για τη δημιουργία ενός αληθοφανούς κόσμου κτλ.
* Μη λειτουργικές απαιτήσεις : Σε αυτή την κατηγορία αναφέρονται οι περιορισμοί ή οι προδιαγραφές που πρέπει να πληροί η εφαρμογή, αλλά δεν αφορούν απαραίτητα τις λειτουργίες της. Περιλαμβάνονται, δηλαδή, θέματα ασφαλείας, απόδοσης, χρηστικότητας, συμβατότητας με συγκεκριμένες πλατφόρμες, επεκτασιμότητας κτλ.

Για τις λειτουργικές απαιτήσεις, μπορούμε να δημιουργήσουμε διάφορους **shaders**, δηλαδή κώδικες που πάνω σε αυτούς μπορούμε να φτιάξουμε τα materials. Μπορεί να ορίζει το πώς θα απεικονίζεται ή θα επεξεργάζεται το material, καθώς επιτρέπει τη δημιουργία ποικίλων εφέ, όπως σκιές, φωτισμό, ανακλάσεις κτλ προσθέτοντας ρεαλισμό. Επίσης, μπορεί να περιέχει λειτουργίες και εντολές που καθορίζουν το πώς θα αλλάξει η εικόνα ή θα γίνει επεξεργασία της γραφικής πληροφορίας κάθε pixel στην οθόνη. Στη δική μας περίπτωση, μπορεί να χρησιμέψει για να κινηθούν τα αντικείμενά μας ή οι χαρακτήρες μας.

Στις απαιτήσεις έχουμε και το έδαφος, το οποίο είναι πολύ σημαντικό για την εφαρμογή, καθώς προσδίδει αληθοφάνεια και πιο ρεαλιστική εμπειρία στο χρήστη.

**Σχεδιασμός**

Ο σχεδιασμός αναφέρεται στη διαδικασία που προηγείται της υλοποίησης του έργου, κατά την οποία καθορίζονται οι λεπτομερείς προδιαγραφές και οι προσεγγίσεις για την επίλυση του προβλήματος. Ο σχεδιασμός περιλαμβάνει τη δημιουργία μοντέλων, αλγορίθμων και προγραμμάτων που θα χρησιμοποιηθούν στην υλοποίηση του έργου, καθώς και το σχεδιασμό του παιχνιδιού.

Σε αυτό το βήμα, ακολουθείται η στρατηγική ανάπτυξης που αναφέρθηκε στην ανάλυση, δηλαδή αρχίζουμε να υλοποιούμε τις λειτουργικές και μη λειτουργικές απαιτήσεις της εφαρμογής. Στην ουσία, γίνεται ένα σχέδιο προγράμματος κατά το οποίο καθορίζονται τα προγράμματα που πρέπει να γραφτούν, το πώς θα διαμορφωθεί ο χώρος κτλ.

Ο σχεδιασμός αποτελεί κρίσιμο στάδιο στη διαδικασία ανάπτυξης ενός έργου, καθώς καθορίζει τη βάση για την επόμενη φάση της υλοποίησης.

Οι βασικές φάσεις του σχεδιασμού είναι:

* Ανάλυση των απαιτήσεων : Σε αυτή τη φάση του σχεδιασμού, γίνεται μία λεπτομερής ανάλυση των απαιτήσεων του έργου, προκειμένου να κατανοηθεί πλήρως το πρόβλημα που πρέπει να επιλυθεί και οι λειτουργικότητες που πρέπει να παρέχονται. Στην ανάλυση των απαιτήσεων συμπεριλαμβάνονται και οι λειτουργικές και μη λειτουργικές απαιτήσεις που αναφέρθηκαν στην ανάλυση, η απαιτήσεις απόδοσης του παιχνιδιού, οι οποίες περιλαμβάνουν την ταχύτητα φόρτωσής του και την απόδοση των γραφικών του και τέλος οι απαιτήσεις ασφαλείας, οι οποίες αφορούν την προστασία των δεδομένων και την αποτροπή των απειλών.
* Σχεδιασμός αρχιτεκτονικής : Σε αυτή τη φάση, καθορίζεται η γενική δομή του συστήματος και ο τρόπος λειτουργίας των διάφορων συνιστωσών του. Η αρχιτεκτονική περιγράφει το υλικό, το λογισμικό και δικτυακή υποδομή που θα χρησιμοποιηθούν για το παιχνίδι.
* Σχεδιασμός διεπαφών χρήστη : Εάν το έργο περιλαμβάνει διεπαφή χρήστη, καθορίζονται οι λειτουργίες και ο σχεδιασμός των διάφορων ενεργει΄ψν που απαιτούνται.
* Σχεδίαση αλγορίθμων και δομών δεδομένων : Καθορίζονται οι αλγόριθμοι που θα χρησιμοποιηθούν για την επίλυση των προβλημάτων και οι δομές δεδομένων που θα χρησιμοποιηθούν για την οργάνωση και την αποθήκευση των δεδομένων.
* Σχεδιασμός ασφάλειας και απόδοσης : Λαμβάνονται υπόψη θέματα ασφαλείας και απόδοσης του συστήματος, προκειμένου να εξασφαλιστεί η ασφαλής και αποτελεσματική λειτουργία του.

Αρχικά, για το σχεδιασμό του παιχνιδιού, θα πρέπει να δημιουργήσουμε ένα terrain (έδαφος), το οποίο είναι ένα από τα βασικότερα βήματα της εφαρμογής (3D object -> Terrain). Για την εμφάνιση και το σχεδιασμό του παιχνιδιού, θα πρέπει να λάβουμε υπόψη τη γεωμορφία του χώρου. Στο παιχνίδι, αφού αποφασίστηκε να κάνουμε ένα χωριό, προστέθηκε στο έδαφος αρκετή βλάστηση και διάφορα κτίρια και σπιτάκια, καθώς και διάφορες λιμνούλες, πεδιάδες και κοιλάδες. Το έδαφος συμβάλλει αρκετά στην εμπειρία του παίκτη καθώς δημιουργεί το μεγαλύτερο μέρος της ατμόσφαιρας και του περιβάλλοντος του παιχνιδιού και στη δική μας περίπτωση, δημιουργείται μία φιλική και φιλόξενη ατμόσφαιρα. Επίσης, δίνει κίνητρο στο χρήστη για ποικιλία και εξερεύνης του περιβάλλοντος, καθώς στο παιχνίδι υπάρχει ποικιλία στο έδαφος, η οποία προσθέτει ενδιαφέρον και περιπέτεια στο χρήστη.

Στο έδαφος, πρέπει να προστεθούν όρια για να μη μπορεί ο παίκτης να βγει εκτός ορίων και να πέσει στο κενό. Επίσης, στο οπτικό πεδίο του χρήστη πρέπει να προστεθεί κάτι πιο αληθοφανές, όπως για παράδειγμα ομίχλη ή μία εικόνα.

Στο έδαφος, θα προσθέσουμε δένδρα με το paint trees στον inspector. Μπορούμε να κάνουμε add ενός prefab δένδρου και να ζωγραφίσουμε δένδρα μαζικά εκεί που θέλουμε. Στα δένδρα εμπεριέχεται και το level of detail, δηλαδή το πόση λεπτομέρεια θέλουμε να έχει το πλέγμα του αντικειμένου. Εάν θέλουμε δένδρα σε όλο το έδαφος, τότε πατάμε το number of trees και επιλέγουμε πόσα δένδρα θέλουμε να εμφανιστούν μαζικά σε όλο το έδαφος. Το ίδιο μπορούμε να κάνουμε και με το γρασίδι, τα μανιτάρια, τους βράχους κτλ. Να σημειωθεί ότι εάν το δένδρο δεν είναι αντικείμενο και το ζωγραφίζουμε με αυτόν τον τρόπο, τότε είναι δύσκολο να το επεξεργαστούμε. Μπορούμε να τα βάζουμε 1 προς 1 ή να κάνουμε duplicate, το οποίο προφανώς και είναι πιο χρονοβόρο.

A screenshot of a computer

Description automatically generated

Στο σχεδιασμό, θα πρέπει να ληφθεί υπόψη ότι το έδαφος θα πρέπει να γεμίσει με χαρακτήρες και αντικείμενα, τα οποία αναλύονται και παρακάτω στην αναφορά. Τα αντικείμενα και οι χαρακτήρες θα χρησιμοποιηθούν από το unity asset store και το mixamo, τα οποία έχουν αναλυθεί στις πηγές των assets.

Να σημειωθεί ότι μπορούμε και εμείς οι ίδιοι να φτιάξουμε animation, το δυσκολότερο όμως κομμάτι είναι να φτιάξουμε χαρακτήρες (δηλαδή ότι διαθέτει humanoid ή για ζωντανούς οργανισμούς κυρίως).

Το «Humanoid» αναφέρεται σε έναν τύπο ανθρωποειδούς χαρακτήρα που χρησιμοποιείται συχνά στα παιχνίδια, όπου απαιτείται κίνηση και αλληλεπίδραση με το χρήστη. Το humanoid είναι ένα τύπος avatar που έχει ανθρωποειδή χαρακτηριστικά και ανατροφοδοτείται από ένα συγκεκριμένο σκελετό κίνησης. Ένας σημαντικός λόγος για τη χρήση του humanoid είναι η δυνατότητα ανταλλαγής και χρήσης διάφορων animations (κινήσεων) που έχουν δημιουργηθεί για το humanoid σκελετό. Αυτό επιτρέπει στους προγραμματιστές να χρησιμοποιούν animation που δημιουργήθηκαν για διαφορετικούς χαρακτήρες χωρίς την ανάγκη προσαρμογής κάθε animation για κάθε χαρακτήρα ξεχωριστά. Ο σκελετός humanoid συνήθως περιλαμβάνει βασικές κινήσεις, όπως τρέξιμο, περπάτημα και άλλες κοινές ενέργειες. Αυτός ο τύπος σκελετού χρησιμοποιείται επίσης για να διευκολύνει την αλληλεπίδραση με διάφορα συστήματα που υποστηρίζουν τον Humanoid τύπο χαρακτήρα.

Για να ενεργοποιήσουμε το Humanoid, πάμε στο χαρακτήρα, έπειτα στο Rig και μετατρέπουμε το Animation Type σε Humanoid.

A screenshot of a computer

Description automatically generated

Θα πρέπει να σημειωθεί σε αυτό το σημείο ότι οι περισσότερες ενέργειες έχουν γίνει μέσω ρυθμίσεων animation και όχι μέσω κώδικα.

Θα μπορούσαμε να προσθέσουμε και προσβασιμότητα-accessibility (όπως πολυγλωσσικότητα η οποία δε ζητείται από την εργασία, ή γενικότερα υπότιτλοι, μετατροπή κειμένου σε ομιλία, συντομεύσεις πληκτρολογίου κτλ.). Είναι σημαντικό να έχουμε accessibility options στις εφαρμογές μας, με σκοπό να μπορεί να τις χρησιμοποιήσει και να επηρεαστεί από την εφαρμογή μας ένα μεγαλύτερο εύρος χρηστών. Τα accessibility options έχουν σκοπό να κάνουν την εφαρμογή μας πιο εύχρηστη (πχ autosave). Πιο συγκεκριμένες τεχνολογίες που χρειάζονται πρόσθετο υλικό μπορούν να αναφερθούν και ως υποστηρικτική τεχνολογία. Γενικότερα, θα μας εξυπηρετούσε σε αυτή τη περίπτωση το localization (τοπική προσαρμογή), δηλαδή η διαδικασία προσαρμογής της εφαρμογής στις ανάγκες του κοινού (καλύπτουμε όσες περισσότερες μπρούμε) μιας συγκεκριμένης αγοράς. Για το Localization θα μπορούσαμε να αλλάξουμε τη γλώσσα της εφαρμογής όπως αναφέρθηκε, τις νομικές απαιτήσεις, τον ήχο, λειτουργίες για την εξυπηρέτηση ατόμων με κάποια αναπηρία, κτλ.

Χρειαζόμαστε κώδικα για την αλληλεπίδραση μεταξύ των animation και του χαρακτήρα, ο οποίος θα γραφτεί παρακάτω. Επίσης, θα χρειαστούμε κώδικα και για την αλληλεπίδραση του χρήστη με αντικείμενα.

**Υλοποίηση**

Η φάση της υλοποίηση κρίνεται απαραίτητη στη διαδικασία ανάπτυξης ενός έργου. Κατά τη διάρκεια αυτής της φάσης μεταφέρονται οι απαιτήσεις και ο σχεδιασμός σε λειτουργική και εκτελέσιμη μορφή. Κατά την υλοποίηση, λαμβάνονται υπόψη οι ακόλουθες αρχές και πρακτικές, οι οποίες συνεισφέρουν στην αποτελεσματική και αξιόπιστη υλοποίηση του παιχνιδιού μας, ενισχύοντας την απόδοση, την ασφάλεια και τη σταθερότητά του:

* Καλή πρακτική στον κώδικα : Ο κώδικας πρέπει να είναι κατανοητός, οργανωμένος και ευανάγνωστος για τους άλλους προγραμματιστές. Πρέπει να εμπεριέχει σωστά σχόλια, κατάλληλη ονοματοδοσία μεταβλητών και συναρτήσεων και βέλτιστες πρακτικές προγραμματισμού.
* Δοκιμές κώδικα : Πρέπει να γίνονται συχνές δοκιμές στον κώδικα με σκοπό να επιβεβαιωθεί η λειτουργικότητα και η ακρίβεια του κώδικα.
* Συνεχής ενημέρωση : Ενημέρωση για προβλήματα, σφάλματα, για την αντιμετώπισή τους, για ελλείψεις κτλ.
* Υλοποίηση του περιβάλλοντος με διάφορα αντικείμενα και χαρακτήρες
* Αλληλεπίδραση με χαρακτήρων με αντικείμενα ή χαρακτήρων με άλλους χαρακτήρες

Το ύψος του εδάφους μπορεί να ρυθμιστεί μέσω του inspector (πλέγμα από ακμές που δε

μπορεί να λάβει αρνητικές τιμές) με το raise or lower terrain. Με τα brushes μας δίνεται η δυνατότητα να σηκώσουμε βουνά. Μπορούμε να ορίσουμε στο transform x=-500, y=-200, z=-500 για να ανεβάσουμε το επίπεδο και να μπορούν οι ακμές να κατέβουν πιο κάτω. Μπορούμε, επίσης, να αλλάξουμε το χρώμα του με το paint texture, με το οποίο μπορούμε να δημιουργήσουμε και διαφόρων ειδών μονοπάτια στο χωριό. Για παραπάνω πολυπλοκότητα, πατάμε στο Windows -> Package Manager και αναζητάμε terrain tools (Πχ terrain sample assett pack), λειτουργία που μπορεί να κάνει τα εργαλεία που θα χρησιμοποιήσουμε πιο εξελιγμένα (advanced).

Για να μη μπορεί ο παίκτης να βγει εκτός ορίων, δημιουργούμε έναν αόρατο τοίχο με όνομα InvisibleWall στα αντικείμενα, ο οποίος αποτελείται από 4 κύβους διαμορφωμένους, έτσι ώστε να καλύπτουν τις 4 πλευρές του εδάφους, και να αποτρέπουν το χρήστη να πέσει στο κενό και να δημιουργηθεί λογικό σφάλμα.

A screenshot of a computer program

Description automatically generated

Ένα σημαντικό μέρος της υλοποίησης της εργασίας και της τρισδιάστατης εμπειρίας που στοχεύει να προσφέρει, είναι η αλληλεπίδραση του χρήστη με άλλα αντικείμενα στο χώρο. Για να επιτευχθεί αυτό, χρειαζόμαστε διάφορα scripts για τις λειτουργίες που θέλουμε να κάνουν οι χαρακτήρες. Για παράδειγμα, θα μπορούσε να υπάρξει χαιρετισμός μεταξύ του main character με κάποιον μίξαμο χαρακτήρα, ή να ανοίγει κάποια πόρτα αυτόματα όταν φτάνουμε κοντά, ή να ρυθμίζεται ο ήλιος ανάλογα με το που βρίσκεται ο main character.

Για το φωτισμό του παιχνιδιού δημιουργήθηκε ένα δικό μας animation, με τον τρόπο που παρουσιάστηκε στις διαλέξεις. Πάμε στο Directional Light (σε ένα αντικείμενο δηλαδή που δεν έχει animation ή animator) και έπειτα πάμε στο Windows -> Animation -> Animation.

A screenshot of a computer program

Description automatically generated

Εκεί, εμφανίζεται ένα ειδικό παράθυρο όπου εκεί μπορούμε να φτιάξουμε το animation μας.

A screenshot of a computer

Description automatically generated

Για να γίνεται συνέχεια, θα πρέπει το loop time να είναι ενεργοποιημένο.

Στο animation της Έρικας, μπορούμε να δούμε τα keyframes, δηλαδή τα στιγμιότυπα. Το KeyFraming είναι η διαδικασία που γίνεται μέσω κάποιου προγράμματος όπου εμείς ορίζουμετην αρχική θέση ενός απλού ή σύνθετου αντικειμένου και την τελική του θέση, επίσης, ενώ στη μέση παράλληλα καταγράφονται όλες οι ενέργεις που γίνονται για να πάει το αντικείμενο από την αρχική του θέση στην τελική. Αριστερά, παρατηρούμε όλα τα κόκκαλα του χαρακτήρα μας, και το animation μας δείχνει σε ποια σημεία κατά τη διάρκεια του animation θα βρίσκεται σε κίνηση. Μπορούμε, επίσης, να τροποποιήσουμε την περιστροφή, την κλίμακα κτλ.

A black and white keyboard

Description automatically generated

Στα Assets, δημιουργούμε φάκελο MyAnimations για να προσθέσουμε το Animation.

A screenshot of a video game

Description automatically generated

Μπορούμε να προςθέσουμε keyframes. Στη δική μας περίπτωση, θα γίνεται μία περιστροφή μία μόνο φορά. Πατάμε το record και κινούμε κατά τη διάρκεια βιντεοσκόπησης το φως, κάνοντας του μια περιστροφή (αλλάζουμε θέση αντικειμένου και inspector). Όταν τελειώσουμε με την περιστροφή, σταματάμε το record. Το transform και το rotation το αλλάζουμε μέσω animation και όχι μέσω κώδικα. Μπορούμε, έπειτα, να αλλάξουμε την ταχύτητα (οι μεταβλητές που αλλάζουμε είναι ανεξάρτητες μεταξύ τους). Η ταχύτητα για να γίνει η αλλαγή της ημέρας ορίστηκε στο 0.05, δηλαδή το animation θα τρέχει στο 5% της αρχικής του ταχύτητας. Η τιμή 1.0 αντιπροσωπεύει την κανονική ταχύτητα, ενώ τιμές μικρότερες από 1.0 θα επιβραδύνουν την κινούμενη εικόνα και οι τιμές μεγαλύτερες από 1.0 θα την επιταχύνουν. Η αλλαγή της ημέρας στο παιχνίδι γίνεται κάθε 20 λεπτά περίπου.

A screenshot of a computer

Description automatically generated

Για την αλληλεπίδραση μεταξύ των χαρακτήρων, δηλαδή το interact, πρέπει να γίνεται εκεί που βλέπει ο χαρακτήρας και όχι η κάμερα. Χρησιμοποιώ το trigger και την απόσταση για τον έλεγχο των αντικειμένων που είναι πιο κοντά στο χρήστη (αυτό που είναι πιο κοντά από τα υπόλοιπα υπερισχύει) και κάνω συγκρίσεις με γωνία. Για να γίνει αυτό διαμορφώνουμε το σχήμα σε κώνο, κόβουμε αν θέλουμε τις γωνίες ή τους βάζουμε βάρη και αποστάσεις. Από τον παίκτη κάνουμε κίνηση για αλληλεπίδραση σε ένα αντικείμενο, που ίσως να είναι αλληλεπιδράσιμο, ίσως και όχι.

Raycast : το αντικείμενο βρίσκεται ακριβώς μπροστά στον παίκτη και το πρώτο επιλέγεται. Το αντικείμενο αλληλεπιδράει με το χρήστη όταν βρίσκεται στην περιοχή του.

Trigger + Raycast : Τα αντικείμενα βρίσκονται στην περιοχή Trigger, δεν εμποδίζονται και το κοντινότερο επιλέγεται.

Trigger + Raycast + Camera : Επιλογή κοντινότερου αντικειμένου, ενέργεια η οποία μπορεί να διαφέρει στην υλοποίησή της με βάση το χαρακτήρα που χρησιμοποιούμε.

Interactors : Ξέρουμε ποιες μεθόδους καλούμε όταν ένα αντικείμενο ταιριάζει στα χαρακτηριστικά που έχουμε ορίσει.

**Σύστημα αλληλεπίδρασης** : Ο χρήστης επιλέγει και αλληλεπιδρά με αντικείμενα στο χώρο όπως το πότε και με ποιον. Τα αντικείμενα ακολουθούν ένα συμβόλαιο και λειτουργεί για αντικείμενα κάθε πολυπλοκότητας. Για την αλληλεπίδραση, δημιουργούμε φάκελο Myscripts και προσθέτουμε μέσω τον κώδικα.

A screenshot of a computer

Description automatically generated

Στη maincamera του χαρακτήρα μας κάνουμε add component τον κώδικα Interactor :

A screenshot of a computer

Description automatically generated

Παρατηρούμε ότι υπάρχουνε κάποια attributes στο script interactor, στο setup και στα buttons. Με την πρώτη αρχικοποίηση εισάγονται κάποιες default τιμές. Στη maincamera υπάρχουνε layers (επίπεδα, που πρόκειται για μία σειρά από 0 και 1), όπου αντικείμενα επιλέγουνε σε ποιο layer θα είναι. Φτιάχνουμε Layer Player και για τον παίκτη και layer για το interactable.

Το canvas είναι ένα δημιουργημένο GameObject (UI -> canvas) και το δημιουργούμε για να βάλουμε κουμπί και να γίνεται το interact με το αντικείμενο. Με αυτό το αντικείμενο, υπολογίζεται που βρίσκονται τα στοιχεία με pixels. Με διπλό αριστερό κλικ στον καμβά παρατηρούμε ότι είναι ένα γιγαντιαίο αντικείμενο. Κάνουμε δεξί κλικ στον καμβά -> UI -> Image (έχει διαφορετικό transform, width, height, κτλ). To image μπορεί να πάρει ένα sprite (UI γραφικό) και είναι ένα ακίνητο τετράγωνο που ακολουθεί την κάμερα σαν δισδιάστατο αντικείμενο. Θέτουμε το canvas scaler -> scale with screen size και το reference resolution σε x=1920 και y=1080 me Match -> 1. Με αυτές τις ρυθμίσεις, το UI στοιχείο παραμένει ακίνητο στην οθόνη του χρήστη. Μπορούμε, επίσης, να αλλάξουμε και τις διαστάσεις του παιχνιδιού, καθώς στο export του παιχνιδιού επιλέγουμε ποιες αναλύσεις θέλουμε. Βάζω την εικόνα στη μέση με x-y-z = 0, width-height = 16 pixels και έπειτα επιλέγω sprite (συνηθίζεται να βάζουμε κύκλο) και χρώμα από τον πολλαπλασιαστή χρωμάτων. Με τις επιλογές μας, θα πρέπει ο χρήστης να ξέρει που θα πατήσει για να γίνει αλληλεπίδραση. Στο Canvas κάνω προσθήκη το script InteractorUI και πατάω δεξί κλικ στον καμβά για να δημιουργήσω ένα UI TextMeshPro αντικείμενο, το οποίο θα εισαχθεί στο MessageText. Σε αυτό το αντικείμενο, βάζουμε width = 600 και Height = 80, βάζουμε το wrapping -> Disable (για να μη πηγαίνει το κείμενο στην επόμενη γραμμή). Μπορούμε να προσθέσουμε δικό μας fontsize και γενικότερα είδος γραμματοσειράς. Με \n δημιουργούμε μία νέα γραμμή

Οι χαρακτήρες και τα αντικείμενα που έχουν προστεθεί, καθώς και κώδικας που έχει γραφτεί για αυτά, αναλύεται παρακάτω. Το RectTransform το βάζω button. Φτιάχνω τα UI σε HD οθόνη, δηλαδή width = 1800 και height = 120. Ο canvas στο Interactor UI ζητάει Text Mesh, άρα βάζω αυτό που έχω δημιουργήσει. Ο interactor στην maincamera ζητάει έναν InteractorUI, οπότε βάζουμε τον καμβά που δημιουργήσαμε. Κάνω το chosshair duplicate και το ονομάζω InteractHint.

Να σημειωθεί, ότι στην ιεραρχία όποιο αντικείμενο εμφανίζεται πρώτο, ζωγραφίζεται και πρώτο. Το Hint έχει δημιουργηθεί για αλληλεπίδραση, το chosshair και καλά το κουμπί και Text το μήνυμα. Για να υλοποιηθούν αυτά, προσθέτουμε ένα κύβο και μέσα σαν παιδί έναν άλλον κύβο που τον ονομάζουμε indicator. Στον αρχικό κύβο κάνουμε add component το Text Sign Script και στον indicator βάζουμε το παιδί του κύβου indicator.

Μπορούμε να βάλουμε τώρα ένα αντικείμενο παιδί σαν indicator. Ο indicator θα εμφανίζεται όταν ο interactor το έχει κάνει select.

Για τη διευκόλυνση του χρήστη, δημιουργήθηκε ένα σκριπτ με όνομα MainMenu, με το οποίο με συγκεκριμένα κουμπιά θα μπορεί να διαχειρίζεται την εφαρμογή. Δημιουργούμε μία μέθοδο QuitApp(), έτσι ώστε ο χρήστης να μη κάνει Alt+F4 και μπορούμε να το καλέσουμε απ’ όπου θέλουμε. Φτάχνουμε επίσης μία μέθοδο StartApp(), η οποία θα εκκινεί την εφαρμογή. Το MainMenu θέλουμε να βρίσκεται στην ίδια σκηνή, στο ίδιο επίπεδο όπου βρίσκεται και η κανονική εφαρμογή. Ένας δεύτερος τρόπος είναι να φτιάξουμε και μία δεύτερη σκηνή με απλά γραφικά και κάποια κουμπιά, και με το start να μας φορτώνεται η δεύτερη σκηνή. Εμείς υλοποιήσαμε τη δεύτερη επιλογή, οπότε πάνω αριστερά πατάμε File -> New Scene -> StandarURP -> Create -> και την ονομάζουμε MainMenu. Πατάμε δεξί κλικ UI -> Button – TextMeshPro. Όταν δημιουργήσουμε το κουμπί, εμφανίζονται 3 αντικείμενα μαζί : Canvas που περιέχει όλα τα UI στοιχεία, το κουμπί Button που φτιάξαμε και περιέχει μέσα Text (είναι 2ω διαφορετικά αντικείμενα) και έχουμε και το EventSystem αντικείμενο, όπου πρακτικά είναι αυτό που υλοποιεί τα κλικ μας. Αν δεν υπάρχει το EventSystem δεν ανταποκρίνεται η οθόνη. Το script MainMenu προστίθεται στον canva.

A close-up of a logo

Description automatically generated

Για να βλέπουμε το τελικό αποτέλεσμα, πάμε στο τελικό παράθυρο του game και όχι στη σκηνή. Το Button λαμβάνει τη θέση του στο χώρο με βάση τα pixels, οπότε στον καμβά, στο canvas scaler βάζουμε στο UI Scale Mode -> Scale With Screen Size, δίνουμε στο reference resolution τις τιμες x=1920 και y=1080 και κάνουμε Match height στο 1.

A screenshot of a computer program

Description automatically generated

Ορίζω στο κουμπί width = 500, height = 100, x=y=z=0 για να μπει στο κέντρο. Κάνω duplicate για να έχω ένα κουμπί για το start της εφαρμογής και ένα για το quit.

Με δεξί κλικ στον καμβά -> UI -> Image δημιουργούμε μία εικόνα. Αυτή η εικόνα θα οριστεί ως το backgraound μας. Στο stretch βάζω να καλύπτει η εικόνα όλη την οθόνη και θέτω x=y=z=right=bottom=0. Μπορούμε να προσθέσουμε κάποια εικόνα ή να βάλουμε κάποιο χρώμα ως background.

Στα κουμπιά, υπάρχει ένα Unity Event που λέγεται OnClick() και στον inspector εμφανίζεται ως μία λίστα στην οποία μπορούμε να προσθέσουμε μεθόδους. Χωρίς να γράψουμε κώδικα, όταν πατάμε το κουμπί Start, με το OnClick, αν θέσουμε το SetActive για το κουμπι Quit στο false, το κουμπί εξαφανίζεται πατώντας το Start. Αυτό γίνεται γιατί το κουμπί quit είναι public.

A screenshot of a computer

Description automatically generated

A screenshot of a computer

Description automatically generated

**A screen shot of a computer

Description automatically generated**

Με αυτόν τον τρόπο, στο onclick του start βάζουμε τον καμβά και τη μέθοδο StartApp() που έχουμε δημιουργήσει στο πρόγραμμά μας (πρέπει να είναι Public οι μέθοδοι για να μπορούν να χρησιμοποιηθούν).

**A screenshot of a computer

Description automatically generated**

Κάνω το ίδιο και στο Quit Button με την QuitApp μέθοδο. Για να κάνουμε quit την εφαρμογή πρέπει να γίνει build της εφαρμογής. Εμείς στην ουσία όμως θέλουμε να καλέσουμε και να φορτώσουμε μία άλλη σκηνή. Για να φορτώσουμε μία άλλη σκηνή πρέπει να κάνουμε using UnityEngine.SceneManagement, δηλαδή η κλάση MainMenu, με την προσθήκη αυτής της βιβλιοθήκης θα μπορεί να διαχειρίζεται σκηνές. Να σημειωθεί ότι για να ανοίξουμε την Samplescene, η διαδικασία γίνεται ασύγχρονα και είναι πιθανό για κάποια λεπτά η εφαρμογή μας να «παγώσει», μέχρι να ανοίξει την άλλη σκηνή.

Στον καμβά, φτιάχνουμε ένα slider (δεξί κλικ στον καμβά -> UI -> Slider), τον σβήνω από interactable και φτιάχνω το transform. Στο canvas, στο mainmenu προσθέτω τον slider. Στην ουσία, όταν φορτώσει ο slider, εμφανίζεται και η άλλη σκηνή, δηλαδή η SampleScene.

A screenshot of a computer

Description automatically generated

Να σημειωθεί ότι μπορούμε να έχουμε και δύο σκηνές active ταυτόχρονα. Αν σύρουμε τη μία σκηνή μέσα στην ιεραρχία του χάρτη, τότε εμφανίζονται και οι δύο σκηνές στην οθόνη. Υπάρχει επίσης τρόπος ένα αντικείμενο να μη διαγραφεί. Για να γίνει αυτό, θα πρέπει στο MainMenu, στη μέθοδο Start() να προσθέσουμε τον κώδικα DontDestroyOnLoad(gameObject);, όπου πρακτικά και θα τρέχει η εφαρμογή και θα έχουμε και το MainMenu. Τώρα, όταν πατήσουμε start δημιουγείται η σκηνή DontDestroyOnLoad και όταν φορτώσω το καινούριο MainMenu, ο καμβάς εμφανίζεται για δεύτερη φορά στο OnDestroy. Εάν θέλουμε, μπορούμε να υλοποιήσουμε το singleton pattern, όπου θα ξέρουμε ότι το αντικείμενο θα υπάρχει μόνο μία φορά (single) και θέλουμε άλλα αντικείμενα να έχουνε πρόσβαση σε αυτό.

Το Singleton pattern είναι ένας από τα πιο διαδεδομένα σχεδιαστικά μοτίβα στον αντικειμενοστραφή προγραμματισμό. Σκοπός του είναι να διασφαλίσει ότι μια κλάση θα έχει μόνο ένα μοναδικό αντίγραφο (instance) και να παρέχει ένα γενικά προσβάσιμο σημείο πρόσβασης προς αυτό το αντίγραφο. Ουσιαστικά, περιορίζει τη δημιουργία νέων αντιγράφων μιας κλάσης σε μία μόνο κλήση, παρέχοντας ένα public σημείο πρόσβασης σε αυτό το αντίγραφο. Ο τρόπος υλοποίησης του μοτίβου περιλαμβάνει συνήθως μία στατική μέθοδο ή μεταβλητή που επιστρέφει το μοναδικό αντίγραφο της κλάσης, ελέγχοντας πρώτα αν το αντίγραφο έχει ήδη δημιουργηθεί. Αυτό γίνεται συνήθως, με έναν ιδιωτικό constructor που αποτρέπει τη δημιουργία νέων αντιγράφων από άλλες μεθόδους.

Στον canvas προσθέτω UI -> Image -> Blocker για να μπλοκάρω το χρήστη απ’ το να πατάει άλλα κουμπιά κατά τη διάρκεια της φόρτωσης. Οπότε για να το ενεργοποιήσω, πάω στο κουμπί Start και προσθέτω την εικόνα.

A screenshot of a computer program

Description automatically generated

A screenshot of a computer

Description automatically generated

Για να χτίσουμε την εφαρμογή πάμε στο File -> Build Settings -> Συμπεριλαμβάνουμε όλες τις σκηνές (προαιρετικά) -> Ανοίγουμε όλες τις σκηνές και τις φορτώνουμε στο Scenes in Build (όποια έχει το Index 0 θα φορτωθεί πρώτη)

**Πίνακα με C# scripts και περιγραφή τους**

Για τη συγγραφή κώδικα, πατάμε αριστερό κλικ στο φάκελο που θέλουμε να αποθηκευτεί και έπειτα δεξί κλικ -> Create -> C# script. Με το enter γίνεται compile και ανοίγει η εφαρμογή Visual Studio. Στον κώδικα, χρησιμοποιούμε κάποια libraries (κλάσεις που φτιάχνονται αυτόματα, μπορούμε και εμείς να κάνουμε αναφορά σε κάποια βιβλιοθήκη) και όταν ανοίγουμε το script έχουμε τις μεθόδους Start() και Update(). H Start() καλείται μία φορά στο πρώτο καρέ που το GameObject είναι ενεργό και πριν την πρώτη κλήση της Update. Η μέθοδος Update() καλείται μία φορά κάθε καρέ εάν το GameObject και το script είναι ενεργά. Τα scripts προσθέτονται σε ένα χαρακτήρα ή αντικείμενο με το add component.

Στα scripts μπορούμε να υλοποιήσουμε τη λειτουργικότητα των χαρακτήρων, τις κινήσεις τους, τη συμπεριφορά τους και τις δράσεις τους εντός του παιχνιδιου, με σκοπό να γίνει πιο ρεαλιστικό και ενδιαφέρον. Επίσης, υλοποιούμε τη λειτουργικότητα του παιχνιδιού γενικότερα, ορίζοντας τους κανόνες και τους μηχανισμούς του (ελέγχους, επικοινωνία μεταξύ χαρακτήρων κτλ.). Σημαντικά είναι τα scripts για την αλληλεπίδραση του παίκτη με το περιβάλλον γενικότερα που έχουμε δημιουργήσει. Τέλος, μπορούμε να διαχειριστούμε τους ήχους και τα εφέ, καθώς και τη διαχείριση των δεδομένων του παιχνιδιού.

Αναφέρθηκε και παραπάνω, ότι για να προστεθεί ένα πρόγραμμα σε ένα αντικείμενο, πρέπει να κάνουμε add component το script. Στα scripts χρησιμοποιήθηκε το OnCollisionEnter και το OnTriggerEnter. Το OnTriggerEnter καλείται 50 φορές το δευτερόλεπτο για κάθε αντικείμενο που βρίσκεται στο χώρο (κάθε fixedUpdate), δηλαδή μπορεί σε μία επανάληψη να κληθεί 3 φορές αν εκείνη τη στιγμή εισαχθούν δύο αντικείμενα στην περιοχή, αλλά μπορεί να κληθεί και 0. Στα Project Settings υπάρχουνε μεταβλητές πχ. Time με fixed timestep, όπου οι κινήσεις σχετίζονται με φυσική και γίνονται κάθε 50 φορές το δευτερόλεπτο. Μπορούμε, επίσης, να ορίσουμε time scale, δηλαδή το πόσο γρήγορα θα τρέχει η εφαρμογή μας (0.3 -> 30% ταχύτητα ή 10 για μέγιστη ή 0 για παύση του παιχνιδιού).

Στο πλαίσιο των παιχνιδιών και της φυσικής μηχανικ’ης, η ιδιότητα «kinematic» αφορά την κίνηση ενός αντικειμένου. Όταν ένα αντικείμενο ορίζεται ως kinematic, σημαίνει ΄οτι η κίνησή του δεν επηρεάζεται από δυνάμεις που δρουν σε άλλα αντικείμενα όπως η βαρύτητα, οι συγκρούσεις κτλ. Επίσης, ένα αντικείμενο kinematic ελέγχεται και κινείται από τον προγραμματιστή άμεσα, χωρίς να επηρεάζεται από τις φυσικές δυνάμεις του περιβάλλονοτος. Αυτό χρησιμοποιείται για να ελέγχεται η κίνηση του χαρακτήρα ή άλλων αντικειμένων από τον κώδικα του παιχνιδιού.

|  |
| --- |
| Στους χαρακτήρες mixamo χρησιμοποιήθηκε το Set IsTrigger to true. Το other είναι το αντικείμενο collider και με το gameObject παίρνω όλο το αντικείμενο. Όταν ένα άλλο αντικείμενο μπαίνει μέσα στον collider, η μέθοδος OnTriggerEnter ενεργοποιείται. Με το SetActive(false) απενεργοποιείται το gameObject που εισήλθε μέσα στον collider. Για να γίνει η σύγκρουση πρέπει και τα δύο αντικείμενα να έχουν collider. Επίσης, πρέπει τουλάχιστον ένα απ’ τα δύο αντικείμενα να έχει component rigidBody (component για βαρύτητα, kinematic όταν δε θέλω σύγκρουση με αντικείμενο).  Script onTrigger  private void OnTriggerEnter (Collider other)  {  other.gameObject.SetActive(false);  } |
| Με τον συγκεκριμένο κώδικα ελέγχουμε πότε ο παίκτης και μόνο (κάνουμε φιλτράρισμα) μπαίνει στην περιοχή που έχουμε ορίσει και άμα μπει γίνεται reference στον animator με σκοπο να αλλάξει η μετανλητή που λέγεται wave και σε αυτή την περίπτωση είναι trigger. Στα μιξαμο, υπάρχει συνήθως ήδη ένας collider trigger sphere (ο trigger καλεί οποιοδήποτε αντικείμενο βρίσκεται στην περιοχή). Ο Animator είναι public και σειριοποιήσιμος, άρα εκτεθειμένη μεταβλητή null, στην οποία κάνουμε drag and drop με αριστερό κλικ τον animator που θέλουμε να έχει ο μίξαμο χαρακτήρας.  Script npc (non playable character)  public class npc : MonoBehaviour  {  public Animator npcAnimator;  // Start is called before the first frame update  void Start()  {    }  // Update is called once per frame  void Update()  {    }  private void OnTriggerEnter(Collider other)  {  if (other.tag == "Player") //the player  {  Wave();  }  }  void Wave()  {  npcAnimator.SetTrigger("Wave");  }  } |
| Για την αλληλεπίδραση, μπορούμε να χρησιμοποιήσουμε το SetTrigger και μέσω προγράμματος να το θέσουμε true (έχει αναλυθεί και στο πρώτο script το setTrigger).  Script messageDisplay  public GameObject messageDisplay; //can be text or 3d object  private void OnTriggerEnter(Collider other)  {  //filter only player  if (other.tag == "Player")  {  messageDisplay.gameObject.SetActive(true);  }  }  } |
| Ο συγκεκριμένος κώδικας παρουσιάστηκε στις διαλέξεις και έχει χρησιμοποιηθεί στη συγκεκριμένη εργασία για το σύστημα αλληλεπίδρασης. Αυτός ο κώδικας ορίζει το σύστημα φιλτραρίσματος για την επιλογή των αντικειμένων και Interactable ορίζονται τα αλληλεπιδράσιμα αντικείμενα. Ο interactor θα ψάχνει για interactables. Ο interactor κάνει την αναζήτηση παίρνουμε member variables. Στον κώδικα έχουμε 4 μεθόδους παρακάτω : την ReadyInteract για το Select, την AbortInteract για το Deselection, την Interact και την EndInteraction. Χρησιμοποιούμε δηλαδή receive μεθόδους, με τις οποίες καλούμε τα αντικείμενα σε εμάς. Επίσης, έχουμε κάποιες overload μεθόδους, δηλαδή έχουν ίδιο όνομα αλλά είναι διαφορετικές μέθοδοι. Οι μεταβλητές που έχουμε ορίσει στην αρχή του προγράμματος, ορίζουν με ποιο αντικείμενο αλληλεπιδράμε (currentInteractable). Στην αρχή το ορίζουμε ως Null και μετά με το φιλτράρισμα ενδέχεται να αλλάξει. Το lastinteractable χρησιμοποιείται για ελέγχους από το ένα καρέ στο επόμενο. Το Targettag είναι η ετικέτα, το rayMaxDistance η ακτίνα που ορίσαμε και το layerMask είναι μία μεταβλητή 32bit, όπου το κάθε bit (0 και 1) αναπαριστά το αν θα περιλαμβάνουμε το layer ή όχι. Γίνεται, έπειτα, αναφορά στο InteractorUI και το hint και με το KeyCode interactkey επιλέγω με ποιο κουμπί θα γίνεται interact. Με το KeyCode[] cancelInteractionKeys κάνω απόρριψη του interaction. Η αρχικοποίηση γίνεται όταν προσθέσω για πρώτη φορά το component. Στο Update χτυπάω μια ακτίνα μπροστά στον παίκτη με απόσταση 20 και αν είναι interactable, βάζω στο currentInteractable αυτό που βρέθηκε. Στο if (!interacting) γίνεται το interact, με το φιλτράρισμα να έχει γίνει πιο πάνω. Το πιο απλό φιλτράρισμα που μπορεί να γίνει είναι απ’ το κέντρο της κάμεραςνα πετάμε ακτίνα στο κέντρο και να πετυχαίνουμε αντικείμενα. Ο interactor στέλενι το μήνυμα και την κλάση, συνεπώς προστίθεται πάνω στην κάμερα. Στο layerMask επιλέγω σε ποια layers θα χτυπάω.  Script Interactor  using System.Collections;  using System.Collections.Generic;  using UnityEngine;  //Version 1.2  public class Interactor : MonoBehaviour  {  [Header("Output")] //--------------------------------------------------    public IInteractable currentInteractable; //interactable object that player is currently looking at  IInteractable lastInteractable; //saved last object for calculations between checks/frames    [Header("Setup")] //--------------------------------------------------  [SerializeField] private string targetTag = "Interactable"; //the tag name of the interactables  [SerializeField] private float rayMaxDistance = 5; //interact max distance  //layer mask determines which layers the layer have to search  //for example layermash should be set to everything except maybe the player adn ingoreRaycast layer  //Edit/Project Setting/Physics/Layer Collision Mask to see layer interaction  [SerializeField] private LayerMask layerMask = ~0; //negative binary of 0 -> 2^32-1  [SerializeField] private InteractorUI interactorUI;  [SerializeField] private GameObject hint; //UI or 3D object hint (like "press E to interact")  [Header("Buttons")] //--------------------------------------------------  [SerializeField] private KeyCode interactKey = KeyCode.E;  [SerializeField] private KeyCode[] cancelInteractionKeys; //list of all cancel buttons  //--------------------------------------------------  //state booleans  bool readyInteract;  bool interacting;  // Start is called before the first frame update  void Start()  {  //reset variables  currentInteractable = null;  interacting = false;  //console warning if variables doesn't have a reference  //we may put if statements on methods but this will fill up the code without practical use  //...other that we forgot to assign these. We rimind it on the Start method.  if (interactorUI==null) Debug.LogWarning("Interactor: InteractorUI component was not set.");  if (hint==null) Debug.LogWarning("Interactor: Hint GameObject was not set.");  //call abort method that we know it resets variables instead of doing it here  AbortInteract();  }  // Update is called once per frame  void Update()  {  currentInteractable = null;  RaycastHit hit; //a ray with useful properties  //Physics.Raycast(start position, direction, output, distance"lenght" of raycast)  if (Physics.Raycast(transform.position, transform.forward, out hit, rayMaxDistance, layerMask)) //raycast check  {  //output object's collider => then get object's transform and others  if (hit.collider.CompareTag(targetTag)){  IInteractable interactable = hit.collider.GetComponent<IInteractable>();  if (interactable != null)  {  currentInteractable = interactable;  }  }  }  //if we are looking at an interactable object and we did NOT last frame/check  if (currentInteractable != null && !readyInteract){  ReadyInteract();  }  //if we are NOT looking at an interactable object and we did last frame/check  else if (currentInteractable == null && readyInteract){  AbortInteract();  }  if(!interacting){ //if you aren't currently interacting  if (readyInteract && Input.GetKeyDown(interactKey)){ //...but you are ready to interact AND pressing interact button  Interact();  }  }  else { //if interacting  if (IsCancelButtonPressed()){ //is pressing any cancel button (movement buttons for example)  EndInteract();  }  //if this frame the interactable object that we are looking changed  //if current = null or if this frame/check the object is different from last frame/check  //this is an extra cancelation check  else if (currentInteractable == null || currentInteractable != lastInteractable){  EndInteract();  }  }  //if current interactable not null then match current and last  //with lastInteractable we can check if current and last are different objects above this next line  if (currentInteractable != null) lastInteractable = currentInteractable;  }  //ready interaction for player (maybe show a text on a canvas)  void ReadyInteract(){  readyInteract = true;  hint.SetActive(true); //show interact hint  currentInteractable.OnReadyInteract(); //call interface method on selected/current object    //Debug.Log("Ready");  }  //cancel/abort interaction input for player (maybe hide a text on a canvas)  void AbortInteract(){  readyInteract = false;  hint.SetActive(false); //hide interact hint  if (lastInteractable != null)  lastInteractable.OnAbortInteract(); //call interface method on previously selected object (as current is null)  //Debug.Log("Abort");  }  void Interact(){  interacting = true;  readyInteract = true; //consistancy with EndInteract (not so useful)  hint.SetActive(false); //hide interact hint  //call interface method on selected/current object  //send this interactor as parameter so object has a reference  currentInteractable.OnInteract(this);  //Debug.Log("Interact");  }  void EndInteract(){  interacting = false;  //setting readyInteract to false we allow for a clean check next frame/check  //when an interacting is cancelled, a next suitable object could be ready to interact (already looking at it)  //...but without setting readyInteract to false, it thinks we aren't allowed to search for anything new  //(will not call ReadyInteract method properly)  readyInteract = false;  if (lastInteractable != null)  lastInteractable.OnEndInteract();  interactorUI.HideTextMessage();  }  //pubic override method that calls EndInteract if the GameObject that requested matches lastInteractable  public void EndInteract(IInteractable requester){  if (requester == lastInteractable)  EndInteract();  }  //we call OnInteract method on our interactables and we expect to receive something  //this can be a string or the object itself  //a string can be returned from the text of a sign for example  //example of oveload methods  public void ReceiveInteract(string message){  //Debug.Log(message);  //call interactorUI (a separete script that handles UI elements) and show text on UI  interactorUI.ShowTextMessage(message);  }  //overload for more abstract unimplemented received call  public void ReceiveInteract(IInteractable interactable){  //Debug.Log(interactable);  }  //returns true if any of the "cancel" keys is pressed, else return false  bool IsCancelButtonPressed(){  for (int i = 0; i < cancelInteractionKeys.Length; i++)  {  //if button was pressed down this frame  if (Input.GetKeyDown(cancelInteractionKeys[i])){  return true;  }  /\* if button is pressed this frame  if (Input.GetKey(cancelInteractionKeys[i]))  {  return true;  }  \*/  }  return false;  }  } |
| Κλάση που ορίζει το αποτέλεσμα. Αρχικά, έχουμε χρησιμοποιήσει τη βιβλιοθήκη TMPro για να χρησιμοποιήσουμε πιο εξελιγμένα UI text αντικείμενα. Στον κώδικα, παρατηρούμε ότι υπάρχει η μέθοδος ShowText που αλλάζει και εμφανίζει το string και η HideText που το εξαφανίζει.  Script InteractorUI  using System.Collections;  using System.Collections.Generic;  using UnityEngine;  using TMPro; //we will use advanced UI text elements  using UnityEngine.UI; //default UI text elements  //Version 1.2  //interactor helper that return an output to the User using UI elements  //this supports only showing a message on a text UI element  public class InteractorUI : MonoBehaviour  {  [SerializeField] private TextMeshProUGUI messageText; //text that displayes a message  //comment out above line and use the one below if you are NOT using TextMeshPro;  //[SerializeField] private Text messageText;  void Start(){  HideTextMessage(); //reset and hide  //console warning if variables doesn't have a reference  //we may put if statements on methods but this will fill up the code without practical use  //...other that we forgot to assign these. We rimind it on the Start method.  if (messageText == null) Debug.LogWarning("InteractorUI: messageText was not set.");  }    //public method that shows a given message on a UI text element  public void ShowTextMessage(string message){  if (messageText==null){ //check if not null  Debug.LogWarning("ShowTextMessage (method): messageText was not found.");  return;  }  messageText.text = message; //set text of ui text element  messageText.gameObject.SetActive(true); //enable/show text  }  //public method that hides UI text element  public void HideTextMessage(){  if (messageText == null){ //check if not null  Debug.LogWarning("HideTextMessage (method): messageText was not found.");  return;  }  messageText.text = ""; //empty text for safety  messageText.gameObject.SetActive(false); //disable/hide text  }  } |
| Γίνεται inherit του behaviour και υλοποιεί το Interface. Το IInteraactable είναι συμβόλαιο και υλοποιεί τις 4 μεθόδους που αναφέρθηκαν παραπάνω. Μπορούμε να υλοποιήσουμε το συγκεκριμένο συμβόλαιο ή όσα συμβόλαια θέλουμε. Στο OnInteract (Interactor interactor) γίνεται αναφορά του χρήστη, όπου του επιστρέφεται κάτι.  Script Interactable  using System.Collections;  using System.Collections.Generic;  using UnityEngine;  //Version 1.2  //interactor calls and receives from an IInteractable  //anything we want to be interactable has to implement IInteractable  public interface IInteractable  {  //abstract OnInteract. It gets an Interactor so that it may return to it.  void OnInteract(Interactor interactor);  //abstract. Called when interaction had started and now ends (Forced or not).  void OnEndInteract();  //abstract. Called when object gets selected and ready to interact.  //Can be used to indicate selection.  void OnReadyInteract();  //abstract. Called when object gets deselected from interacting.  //Can be used to remove select indication.  void OnAbortInteract();  } |
| Αυτός ο κώδικας αναλαμβάνει το ρόλο μιας πινακίδας sign ή ενός αντικειμένου γενικότερα, στο οποίο θα εμφανίζεται το μήνυμα. Εισάγουμε αρχικά, τις βιβλιοθήκες και τα πακέτα που είναι απαραίτητα για τον κώδικα και έπειτα δημιουργούμε την κλάση TextSign, όπου υλοποιεί 2 interfaces, το MonoBehaviour και το IInteractable, τα οποία περιέχουν μεθόδους που σχετίζονται με τη διαδραστικότητα του παιχνιδιού, όπως είναι η εκκίνηση, η ολοκλήρωση και η ακύρωση μιας αλληλεπίδρασης. Έχουμε επίσης τη μεταβλητή indicator που χρησιμοποιείται ως δείκτης για την πινακίδα και το Text, το οποίο περιέχει το μήνυμα που θα εμφανίζεται στην πινακίδα. Η μέθοδος OnInteract καλείται όταν ο παίκτης αλληλεπιδρά με το στοιείο και η μέθοδος ReceiveInteract καλείται από τον interactor, μεταφέροντας το μήνυμα, ενώ παράλληλα απενεργοποιείται και ο indicator. Τέλος, έχουμε και τις μεθόδους OnAbortInteract, OnReadyInteract κτλ, οι οποίες χρησιμοποιήθηκαν και αναλύθηκαν και στο script Interactor.  Script TextSign  using System.Collections;  using System.Collections.Generic;  using UnityEngine;  //Version 1.2  //A class of a sign, it contains a message for display  public class TextSign : MonoBehaviour, IInteractable  {  [SerializeField] private GameObject indicator;  [TextArea(3,10)]  public string text;  public void OnInteract(Interactor interactor){  indicator.SetActive(false); //hide  //call interactor's public method ReceiveInteract  //...with override method that gets a string as a parameter  interactor.ReceiveInteract(text);  }    //unimplemented Methods  public void OnEndInteract()  {  }  public void OnAbortInteract()  {  indicator.SetActive(false); //hide  }  public void OnReadyInteract()  {  indicator.SetActive(true); //show  }  } |
| Ο κώδικας αυτός γράφτηκε για να μπορεί να σπρώχνει κάποια άλλα αντικείμενα στο περιβάλλον του παιχνιδιου. Το pushlayers (LayerMask) καθορίζει ποια στρώματα θα επηρεάζονται από αυτό το σπρώξιμο, το canpush καθορίζει αν ένα αντικείμενο μπορεί να εκτελεί σπρώξιμο ή όχι και το strength είναι η δύναμη με την οποία γίνεται το σπρώξιμο. Με τη μέθοδο PushRigidBodies εκτελείται η λειτουργία του σπρώξιμου. Παίρνει ως παράμετρο τη σύγκρουση μεταξύ του αντικειμένου και του αντικειμένου με Collider και ελέγχει το αντικείμενο με το οποίο έχει γίνει η σύγκρουση για να ελέγξει αν είναι RigidBody. Ελέγχει επίσης, αν το αντικείμενο ανήκει σε κάποιο από τα layers, δηλαδή στη μάσκα στρωμάτων. Τέλος, εφαρμόζεται η κατεύθυνση και η δύναμη για το σπρώξιμο στο αντικείμενο που συγρκούεται χρησιμοποιώντας τη μέθοδο AddForce.  Script BasicRigidBodyPush  using UnityEngine;  public class BasicRigidBodyPush : MonoBehaviour  {  public LayerMask pushLayers;  public bool canPush;  [Range(0.5f, 5f)] public float strength = 1.1f;  private void OnControllerColliderHit(ControllerColliderHit hit)  {  if (canPush) PushRigidBodies(hit);  }  private void PushRigidBodies(ControllerColliderHit hit)  {  // https://docs.unity3d.com/ScriptReference/CharacterController.OnControllerColliderHit.html  // make sure we hit a non kinematic rigidbody  Rigidbody body = hit.collider.attachedRigidbody;  if (body == null || body.isKinematic) return;  // make sure we only push desired layer(s)  var bodyLayerMask = 1 << body.gameObject.layer;  if ((bodyLayerMask & pushLayers.value) == 0) return;  // We dont want to push objects below us  if (hit.moveDirection.y < -0.3f) return;  // Calculate push direction from move direction, horizontal motion only  Vector3 pushDir = new Vector3(hit.moveDirection.x, 0.0f, hit.moveDirection.z);  // Apply the push and take strength into account  body.AddForce(pushDir \* strength, ForceMode.Impulse);  }  } |
| Η μεταβλητή \_camera αναφέρεται σε αντικείμενο τύπου Camera και με το start αρχικοποιείται η βασική κάμερα του παιχνιδιού (MainCamera), ενώ με το Update, δηλαδή κάθε φορά που ανανεώνεται ένα frame στο παιχνίδι, ορίζεται η κατεύθυνη της κάμερας να είναι ίδια με την κατεύθυνση του αντικειμένου που εφαρμόζεται το σκριπτ.  Script camera  using System.Collections;  using System.Collections.Generic;  using UnityEngine;  public class NewBehaviourScript : MonoBehaviour  {  private Camera \_camera;  // Start is called before the first frame update  void Start()  {  \_camera = Camera.main;  }  // Update is called once per frame  void Update()  {  transform.forward = \_camera.transform.forward;  }  } |
| Ο κώδικας αυτός χρησιμοποιείται για να ανοίξουν οι πόρτες των σπιτιών στο παιχνίδι αυτόματα αν ο παίκτης βρίσκεται εντός των επιτρεπόμενων ορίων, δηλαδή στην ουσία το χειρισμό της πόρτας. Το door είναι η πόρτα, το doorOpern δείχνει εάν η πόρτα είναι ανοιχτή ή κλειστή, το doorOpenposition η τελική θέση της πόρτας σε σχέση με την αρχική της θέση στον άξονα χ και το palyertag είναι η ετικέτα που ορίζει τον παίκτη. Η μέθοδος ToogleDoorState() ελέγχει εάν η πόρτα είναι ανοικτή και εάν δεν είναι καλεί τη μέθοδο OpenDoor() για να την ανοίξει (περιστρέφει την πόρτα κατά 90 μοίρες γύρω από τον άξονα Y). Η μέθοδος IsPlayerAtDoorPosition() ελέγχει εάν ο παίκτης βρίσκεται κοντά στη θέση της πόρτας και ελέγχει εάν η απόσταση μεταξύ του χαρακτήρα και της πόρτας είναι εντός των επιτρεπόμενων ορίων. Στο Update() καλείται η ToogleDoorState, αν βρίσκεται ο παίκτης εντός των ορίων.  Script DoorOpener  using System.Collections;  using System.Collections.Generic;  using UnityEngine;  public class DoorOpener : MonoBehaviour  {  public GameObject door; // Reference to the door object  private bool doorOpen = false;  public float doorOpenPosition = 1f; // Adjust the desired x-position for opening the door  public string playerTag = "Player"; // Set the tag for the player  private void ToggleDoorState()  {  if (!doorOpen)  {  OpenDoor();  }  }  private void OpenDoor()  {  door.transform.Rotate(Vector3.up, 90f); // Rotate around the Y-axis by 90 degrees  doorOpen = true;  }    private bool IsPlayerAtDoorPosition()  {  GameObject player = GameObject.FindGameObjectWithTag(playerTag);  float playerXPosition = player.transform.position.x;  float doorXPosition = door.transform.position.x;  float allowedRange = 2f; // Set the desired range  return Mathf.Abs(playerXPosition - doorXPosition) <= allowedRange;  return false;  }  private void Update()  {  if (IsPlayerAtDoorPosition())  {  ToggleDoorState();  }  }  void Start()  {    }  } |
| Έχουμε τη μέθοδο QuitApp για να κάνουμε κάνουμε quit στην εφαρμογή και StartApp για να την εκκινήσουμε. Για να φορτώσουμε μία άλλη σκηνή πρέπει να κάνουμε using UnityEngine.SceneManagement, δηλαδή η κλάση MainMenu, με την προσθήκη αυτής της βιβλιοθήκης θα μπορεί να διαχειρίζεται σκηνές. Να σημειωθεί ότι για να ανοίξουμε την Samplescene, η διαδικασία γίνεται ασύγχρονα και είναι πιθανό για κάποια λεπτά η εφαρμογή μας να «παγώσει», μέχρι να ανοίξει την άλλη σκηνή. Το allowsceneactivation = false, ενώ φορτώνει η σκηνή, λίγο πριν να είναι έτοιμη για εκκίνηση (πχ στο 90%) βλέπω ότι υπάρχει η σκηνή αλλά δεν εμφανίζεται ποτέ. Θέλω, όταν πηγαίνει στο 90% να το κάνω true με το κουμπί Ε και να επιτρέψω στην εφαρμογή να μεταπηδήσει στην επόμενη σκηνή. Ο κώδικας αυτός χρησιμεύει στο να μεταφορτώνουμε διαφορετικές σκηνές και στο να μην επιβαρύνουμε το σύστημα. Ο κώδικας χρησιμοποιεί επίσης και το μοτίβο Singleton για να αποτρέψει τη δημιουργία περισσότερων από ένα αντιγράφων για την κλάση. Έτσι, δημιουργήθηκε το public static MainMenu Instance το οποίο δηλώνει ένα δημόσιο στατικό μέλος τύπου MainMenu για να μπορέσουμε να αναφερθούμε στο μοναδικό αντίγραφο της κλάσης MainMenu που θα δημιουργηθεί. Όπως και σε προηγούμενα σκριπτ, η μέθοδος Start() καλείται κατά την αρχικοποίηση του αντικειμένου και ελέγχει εάν η μεταβλητή Instance είναι κενή, και εάν είναι η τρέχουσα στιγμή της κλάσης ανατίθεται στην Instance. Αν δεν είναι, τότε σημαίνει ότι έχει δημιουργηθεί ήδη μία κλάση προηγουμένως, οπότε το τρέχον αντικείμενο καταστρέφεται με τη χρήση του Destroy(gameObject) για να αποφευχθεί η δημιουργία περισσότερων αντιγράφων. Στη συνέχεια, η μέθοδος DontDestroyOnLoad(gameObject); χρησιμοποιείται για να διατηρηθεί το αντικείμενο της κλάσης MainMenu στη μνήμη, ανεξάρτητα με το αν θα αλλάξει η σκηνή στο Unity ή όχι.    Script MainMenu  using System.Collections;  using System.Collections.Generic;  using UnityEngine;  using UnityEngine.SceneManagement;  using UnityEngine.UI;  public class MainMenu : MonoBehaviour  {  public static MainMenu Instance; //singleton  public Slider progressBar;  // Start is called before the first frame update  void Start()  {  if (Instance == null)  Instance = this;  else  {  Destroy(gameObject);  return;  }  DontDestroyOnLoad(gameObject);  }  // Update is called once per frame  void Update()  {  }  public void StartApp()  {  //TODO    Debug.Log("Start");  StartCoroutine(LoadYourAsyncScene());  }  IEnumerator LoadYourAsyncScene()  {  yield return new WaitForSeconds(5);  AsyncOperation asyncLoad = SceneManager.LoadSceneAsync("MainMenu");  asyncLoad.allowSceneActivation = false;  while (!asyncLoad.isDone)  {  progressBar.value = asyncLoad.progress;  yield return null;  if (Input.GetKeyDown(KeyCode.E))  {  //allow scene to be activated  asyncLoad.allowSceneActivation = true;  }  }  }  public void QuitApp()  {  Debug.Log("Quit");  Application.Quit();  }  } |

**Αναλυτική παρουσίαση για χρήστες**