ΔΙΕΘΝΕΣ
ΠΑΝΕΠΙΣΤΗΜΙΟ
ΤΗΣ ΕΛΛΑΔΟΣ

ΣΧΟΛΗ ΜΗΧΑΝΙΚΩΝ
ΤΜΗΜΑ ΜΗΧΑΝΙΚΩΝ ΠΛΗΡΟΦΟΡΙΚΗΣ ΚΑΙ ΗΛΕΚΤΡΟΝΙΚΩΝ ΣΥΣΤΗΜΑΤΩΝ

ΔΙΠΛΩΜΑΤΙΚΗ ΕΡΓΑΣΙΑ
«Παιχνίδι Δωματίου Απόδρασης με χρήση τεχνολογιών Επαυξημένης Πραγματικότητας»

Της φοιτήτριας
Σταυρουλάκη Π. Στυλιανής
Αρ. Μητρώου: 134105

Επιβλέπων
Ευκλείδης Κεραμόπουλος
Αναπληρωτής Καθηγητής

Ημερομηνία …13/01/2021……

Τίτλος Δ.Ε. «Παιχνίδι Δωματίου Απόδρασης με χρήση τεχνολογιών Επαυξημένης Πραγματικότητας»
Κωδικός Δ.Ε. 20204
Ονοματεπώνυμο φοιτήτριας Σταυρουλάκη Π. Στυλιανή
Ονοματεπώνυμο εισηγητή Ευκλείδης Κεραμόπουλος
Ημερομηνία ανάληψης Δ.Ε. 16/10/2020
Ημερομηνία περάτωσης Δ.Ε. 13/1/2021

Βεβαιώνω ότι είμαι ο συγγραφέας αυτής της εργασίας και ότι κάθε βοήθεια την οποία είχα για την προετοιμασία της είναι πλήρως αναγνωρισμένη και αναφέρεται στην εργασία. Επίσης, έχω καταγράψει τις όποιες πηγές από τις οποίες έκανα χρήση δεδομένων, ιδεών, εικόνων και κειμένου, είτε αυτές αναφέρονται ακριβώς είτε παραφρασμένες. Επιπλέον, βεβαιώνω ότι αυτή η εργασία προετοιμάστηκε από εμένα προσωπικά, ειδικά ως διπλωματική εργασία, στο Τμήμα Μηχανικών Πληροφορικής και Ηλεκτρονικών Συστημάτων του ΔΙ.ΠΑ.Ε.

«Αφιέρωση»

Πρόλογος

Στα τελευταία εξάμηνα της φοίτησής μου άρχισε να με ενδιαφέρει αρκετά η ανάπτυξη ηλεκτρονικών παιχνιδιών. Με αφορμή κάποια μαθήματα της σχολής ξεκίνησα να μελετώ την πλατφόρμα Unity και να δημιουργώ σε αυτήν κάποια μικρά παιχνίδια. Εξίσου ενδιαφέρθηκα και για τις νέες τεχνολογίες VR και AR και για την χρήση τους στους τομείς της ψυχαγωγίας. Έχοντας παίξει παιχνίδια απόδρασης αλλά και παιχνίδια εικονικής πραγματικότητας, εφόσον έχει ήδη υλοποιηθεί ο συνδυασμός των δύο, αποφάσισα να προσπαθήσω να συνδυάσω την επαυξημένη πραγματικότητα με τα παιχνίδια απόδρασης. Μέσα από αυτή τη διαδικασία, βελτίωσα τις γνώσεις μου στον προγραμματισμό και στις νέες τεχνολογίες, γνώρισα νέες τεχνολογίες, ανακάλυψα τις πιέσεις και την ανάγκη για πειθαρχία που απαιτεί η δημιουργία κάποιου οποιουδήποτε εμπορικού λογισμικού, και εξασκήθηκα στη συνεργασία με εξωτερικούς επαγγελματίες.

Περίληψη

Δημιουργία εφαρμογής επαυξημένης πραγματικότητας για χρήση σε παιχνίδι Δωματίου Απόδρασης. Η εφαρμογή εμφανίζει στον χρήστη/παίκτη 3D αντικείμενα/μοντέλα στο χώρο, μέσω της κάμερας και οθόνης της συσκευής του (Android λογισμικό) δίνοντάς του στοιχεία για να προχωρήσει στην επίλυση του παιχνιδιού και την έξοδό του από το δωμάτιο. Χρησιμοποιήθηκε η πλατφόρμα του Unity, και η εργαλειοθήκη Vuforia για την υλοποίηση της εφαρμογής. Αρχικά πραγματοποιήθηκε μια έρευνα σύγκρισης των εργαλειοθηκών ARCore και Vuforia ώστε να διευκρινιστεί η καλύτερη για την παρούσα εφαρμογή. Στη συνέχεια καταγράφηκε μια σχηματική ιδέα των γρίφων που θα αποτελούσαν το παιχνίδι. Έπειτα ξεκίνησε η δημιουργία των 2D και 3D μοντέλων στο Unity και η συγγραφή των απαραίτητων αρχείων εντολών σε C# για τη λειτουργία τους στο παιχνίδι. Στη συνέχεια δημιουργήθηκαν και προστέθηκαν κατάλληλα γραφικά και εκτυπώθηκαν κατάλληλες εικόνες για την σωστή εμφάνιση των μοντέλων και αναγνώριση εικόνων στο περιβάλλον. Τέλος, έγιναν πρακτικοί έλεγχοι λειτουργίας και διορθώθηκαν κωλύματα στον κώδικα και τα γραφικά μοντέλα.

Thesis Title

«Escape Room game using AR technologies»

Student name

«Styliani P. Stavroulaki»

Abstract

Creation of Augmented Reality application for Escape Room game. The application shows 3D or 2D objects/models to the user/player through the camera and screen of their Android device, giving them clues to proceed to solve the game and escape from the room. The tools that were used to realize the project was Unity engine and Vuforia kit. At first, there was a research comparing two AR kits, ARCore and Vuforia, to determine which would be best for this particular application. Then there was a schematic representation of the puzzles that would constitute the game. Afterwards began the creation of the 2D and 3D models in Unity, along with writing the necessary scripts in C# for the models' function in the game. Later, the appropriate graphics were added and adjusted in the project, as well as images were printed for the correct projection of the models and image recognition in the real world. Lastly, several test-runs took place , before correcting any bugs in the code and in the graphic models.

Ευχαριστίες

Θεωρώ υποχρέωσή μου να ευχαριστήσω τον επιβλέποντα καθηγητή μου Ευκλείδη Κεραμόπουλο για την πολύτιμη καθοδήγηση και ενθάρρυνσή του. Επιπλέον θα ήθελα να ευχαριστήσω τη δεσποινίς Ανδριάννα Κουμεντάκου για τη γενναιόδωρη συνεργασία της στη δημιουργία των γραφικών υλικών που χρησιμοποιήθηκαν στην υλοποίηση αυτής της εφαρμογής. Τέλος, θα ήθελα να ευχαριστήσω τις ανοιχτές κοινότητες του Unity και Stack Overflow για την ατέρμονη αρωγή τους σε όλους τους νέους προγραμματιστές και δημιουργούς.

Περιεχόμενα

Συντομογραφίες

AR	Augmented Reality (Επαυξημένη Πραγματικότητα)
VR	Virtual Reality (Εικονική Πραγματικότητα)
UI	User Interface (Διεπαφή Χρήστη)
3D	Three Dimensional (Τρισδιάστατο)

Κεφάλαιο 1ο Γενική εισαγωγή

1.1 Εισαγωγή

Η παρούσα εργασία είναι ένα εγχείρημα ενσάρκωσης ενός πραγματικού δωματίου απόδρασης με χρήση τεχνολογιών επαυξημένης πραγματικότητας σε συσκευές Android. Οι παίκτες θα εισέρχονται σε ένα πραγματικό δωμάτιο στο οποίο θα υπάρχουν εκτυπωμένες εικόνες σε συγκεκριμένες θέσεις. Χρησιμοποιώντας την εφαρμογή, θα κοιτούν τις εικόνες μέσω της κάμερας της συσκευής τους και ανάλογα την εικόνα, η εφαρμογή θα επαυξάνει την εικόνα με εικονικά 3D γραφικά. Μέσω της οθόνης αφής της συσκευής τους, οι παίκτες θα μπορούν να αλληλεπιδρούν με τα γραφικά, ώστε να λύσουν τον εκάστοτε γρίφο, ή να συλλέξουν κάποιο στοιχείο για τον επόμενο γρίφο.

Η ιδέα για την εργασία αυτή γεννήθηκε από την εξέλιξη των τεχνολογιών επαυξημένης πραγματικότητας και από το πού χωρούν στον τομέα της ψυχαγωγίας και της ανάπτυξης ηλεκτρονικών παιχνιδιών. Είναι μία απόπειρα συνδυασμού πραγματικών και εικονικών στοιχείων για την ολοκλήρωση μιας ατομικής ή και ομαδικής ψυχαγωγικής δραστηριότητας.

Το κλίμα και το ύφος του παιχνιδιού είναι εμπνευσμένο από διάφορα παλαιότερα ηλεκτρονικά παιχνίδια, που κοινά τους στοιχεία είναι η επίλυση γρίφων, η συλλογή στοιχείων μέσω παρατηρητικότητας, και η παγκόσμια μορφή επικοινωνίας και αλληλεπίδρασης – η εικόνα. Αυτά τα παιχνίδια, όπως και το παρόν, μπορούν να παιχτούν από οποιονδήποτε στον κόσμο, χωρίς να παίζει ρόλο η γλώσσα την οποία μιλά, διότι δεν υπάρχει κείμενο ή ομιλία μέσα σε αυτά, παρά μόνο αριθμητικά σύμβολα.

1.2 Δωμάτια απόδρασης

Τα δωμάτια απόδρασης (escape rooms) είναι ζωντανά παιχνίδια γρίφων, τα οποία παίζονται από δυάδες ή ομάδες παικτών, όπου συνήθως οι παίκτες βρίσκονται πραγματικά κλειδωμένοι σε ένα ή και περισσότερα δωμάτια. Στόχος είναι η ομάδα να αποδράσει μέσα σε σύντομο χρονικό διάστημα, λύνοντας μία σειρά από γρίφους ή αποστολές. Το χρονικό διάστημα ποικίλλει και κυμαίνεται από 60 λεπτά μέχρι και 3 ώρες. Τα θέματα στα δωμάτια απόδρασης συνήθως είναι σε φανταστικές τοποθεσίες, όπως κελιά φυλακής, διαστημόπλοια, στοιχειωμένα σπίτια ή ακόμα και μαγεμένα δάση. [1]

Επί του παρόντος, υπάρχουν παραλλαγές δωματίων απόδρασης με τεχνολογίες VR (Εικονική Πραγματικότητα), όπου οι παίκτες εισέρχονται σε ένα εικονικό δωμάτιο και μέσω του ειδικού

εξοπλισμού για VR κινούνται μέσα σε αυτό και λύνουν τους γρίφους για να βγουν από το δωμάτιο και να τερματίσουν το παιχνίδι.

1.3 Περιβάλλον ανάπτυξης της εφαρμογής

Για την ανάπτυξη οποιασδήποτε εφαρμογής που εμπεριέχει γραφικά χρειάζεται κάποια πλατφόρμα ανάπτυξης λογισμικού και γραφικής σχεδίασης [2]. Για αυτήν την εφαρμογή επιλέχθηκε το πρόγραμμα Unity Real-Time Development Platform [3] λόγω εμπειρίας με το συγκεκριμένο εργαλείο και συμβατότητας με τις απαιτήσεις της εφαρμογής.

Το Unity υποστηρίζει τη δημιουργία έργων για τα περισσότερα λογισμικά: Windows, Mac, Linux, Android, iOS, tvOS, Lumin, PS4, Xbox One αλλά και ιστού μέσω του WebGL API [4]. Επιπλέον, υποστηρίζει τη χρήση AR εργαλειοθηκών όπως το Vuforia και το ARCore.

1.4 AR και εργαλειοθήκες

Επαυξημένη πραγματικότητα (αγγλικά: augmented reality) είναι η σε πραγματικό χρόνο άμεση ή έμμεση θέαση ενός φυσικού, πραγματικού περιβάλλοντος, του οποίου τα στοιχεία επαυξάνονται από στοιχεία αναπαραγόμενα από συσκευές υπολογιστών, όπως ήχος, βίντεο, γραφικά ή δεδομένα τοποθεσίας [5] . Ο όρος εισήχθη το 1992 από τον Τομ Κάουντελ.

Στο ξεκίνημα αυτής της εργασίας πραγματοποιήθηκε μια έρευνα προς την εύρεση της πιο κατάλληλης εργαλειοθήκης για αυτήν. Υπήρχε ήδη μια εμπειρία με το Vuforia kit οπότε έγινε σύγκριση με το ARCore. Το ARCore είναι μια εργαλειοθήκη της Google για Android συσκευές. Αντίστοιχα, για iOS συσκευές, υπάρχει το ARKit. Το ARCore σε σχέση με το Vuforia έχει συμβατότητα με λιγότερες συσκευές, και το Vuforia έχει καλύτερο Image Tracking (Ανίχνευση Εικόνων) που αφορά άμεσα την εφαρμογή αυτή.

Αφού δημιουργήθηκε ένα project με ARCore και AR Foundation στο Unity και δουλεύτηκε ο πρώτος γρίφος του παιχνιδιού, βγήκε το συμπέρασμα ότι αυτή η εργαλειοθήκη δεν είναι κατάλληλη για αυτό το project. Ένας σημαντικός ανασταλτικός παράγοντας ήταν η αδυναμία προσομοιώσεων των δοκιμών στην συσκευή Android που χρησιμοποιήθηκε. Οι δοκιμές προβλεπόταν να είναι πολλές και η δημιουργία της εφαρμογής θα καθυστερούσε υπερβολικά αν δεν γίνονταν προσομοιώσεις, αλλά για κάθε δοκιμή να έπρεπε να γινόταν εγκατάσταση της μέχρι τότε εφαρμογής στη συσκευή για δοκιμή.

1.4.1 Image Targets

Η εφαρμογή βασίζεται στο Image Tracking (Ανίχνευση Εικόνων). Η κάμερα "ψάχνει" κάποιες συγκεκριμένες εικόνες στο χώρο, και όταν ανιχνεύσει κάποια από αυτές, την επαυξάνει, δηλαδή εμφανίζει πάνω σε αυτήν κάποια ψηφιακά αντικείμενα. Οι εικόνες αυτές (Image Targets (Εικόνες - στόχοι)) είναι αποθηκευμένες σε μία βάση δεδομένων μέσα στο αρχείο της εφαρμογής. Χρειάστηκε να ανέβουν αυτές οι εικόνες στην ιστοσελίδα του Vuforia, να "κατέβουν" σε μορφή βάσης δεδομένων, να εισαχθεί στο project Unity αυτή η βάση, και να εκτυπωθούν οι εικόνες. Η ιστοσελίδα αναλύει την κάθε εικόνα και εμφανίζει το κατά πόσο είναι κατάλληλη για Image Tracking, με ένα σύστημα πέντε αστεριών. Μια εικόνα θεωρείται κατάλληλη για Image Tracking αν έχει αρκετές διαβαθμίσεις στα χρώματα και τη φωτεινότητά της.

1.5 Γραφικά

Για την ανάπτυξη αυτής της εφαρμογής χρειάστηκαν κάποιες συγκεκριμένες εικόνες. Η ανάπτυξη ξεκίνησε με κάποιες εικόνες που δεν σχετίζονταν με το θέμα, για πειραματισμό και ελέγχους, μέχρι να σχεδιαστούν οι γρίφοι και να δημιουργηθούν οι κατάλληλες εικόνες. Σχέδια χρειάστηκαν τόσο για τα ψηφιακά αντικείμενα της εφαρμογής, αλλά και για τις εκτυπωμένες εικόνες που χρησιμοποιήθηκαν ως Image Targets. Τις εικόνες σχεδίασε μία εξωτερική συνεργάτης που ειδικεύεται στην σχεδίαση γραφικών, ειδάλλως θα χρησιμοποιούταν δωρεάν εικόνες από το διαδίκτυο που δε θα ταίριαζαν όμως στο παιχνίδι. Η δημιουργία ψηφιακών σχεδίων είναι ένας ξεχωριστός κλάδος της ανάπτυξης ηλεκτρονικών παιχνιδιών και συνήθως αναλαμβάνεται από επαγγελματίες γραφίστες σε απόλυτη συνεργασία με τους δημιουργούς.

Η εφαρμογή αποτελείται κυρίως από 3D (τρισδιάστατα) μοντέλα, όπου πάνω τους προστέθηκαν sprites (στοιχεία εικόνων), τα οποία δημιουργήθηκαν κατά κύριο λόγο μέσα στην πλατφόρμα του Unity. Εξαίρεση αποτέλεσαν η διάτρητη σφαίρα (4ος γρίφος) και τα δαχτυλίδια (3ος γρίφος) που δημιουργήθηκαν σε ένα πρόγραμμα σχεδίασης 3D γραφικών (Blender [6]) για τις ανάγκες της εφαρμογής.

1.6 Επίλογος

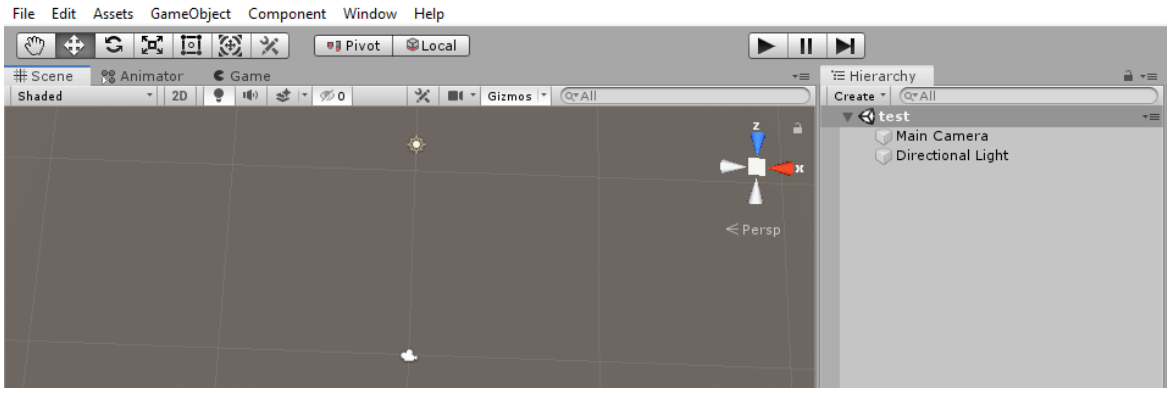
Στην ανάπτυξη παιχνιδιών χρειάζονται εξειδικεύσεις σε διάφορους τομείς πέραν του προγραμματισμού. Αυτή η εργασία έδωσε αφορμή για τη γνωριμία με επιπλέον τομείς στην δημιουργική διαδικασία, αλλά και με νέα εργαλεία, όπως το σχεδιασμό γραφικών (Photoshop) και τον σχεδιασμό 3D μοντέλων (Blender). Επιπλέον, έγινε αναγκαία η συνεργασία με κάποιον άλλο δημιουργό, και αυτό, μπορεί μόνο να βελτιώσει το ομαδικό πνεύμα, την ικανότητα επικοινωνίας και συνεργασίας, αλλά και να εμπλουτίσει το όραμα της δημιουργίας.

Κεφάλαιο 2ο Απαραίτητα συστατικά

2.1 Εισαγωγή

Υπάρχουν κάποια απαραίτητα συστατικά για να λειτουργήσει οποιοδήποτε project στο Unity, και περισσότερα για μία εφαρμογή επαυξημένης πραγματικότητας. Το Unity υποστηρίζει το Vuforia kit το οποίο χρησιμοποιήθηκε για την εφαρμογή, και οι πιο πρόσφατες εκδόσεις του Unity έρχονται μαζί με το Vuforia plug-in οπότε δεν χρειάστηκε να εγκατασταθεί κάτι πέραν από το Unity. Οι ανάγκες της εφαρμογής δεν απαίτησαν την ενημέρωση του λογισμικού στην ακόμα πιο πρόσφατη έκδοση.

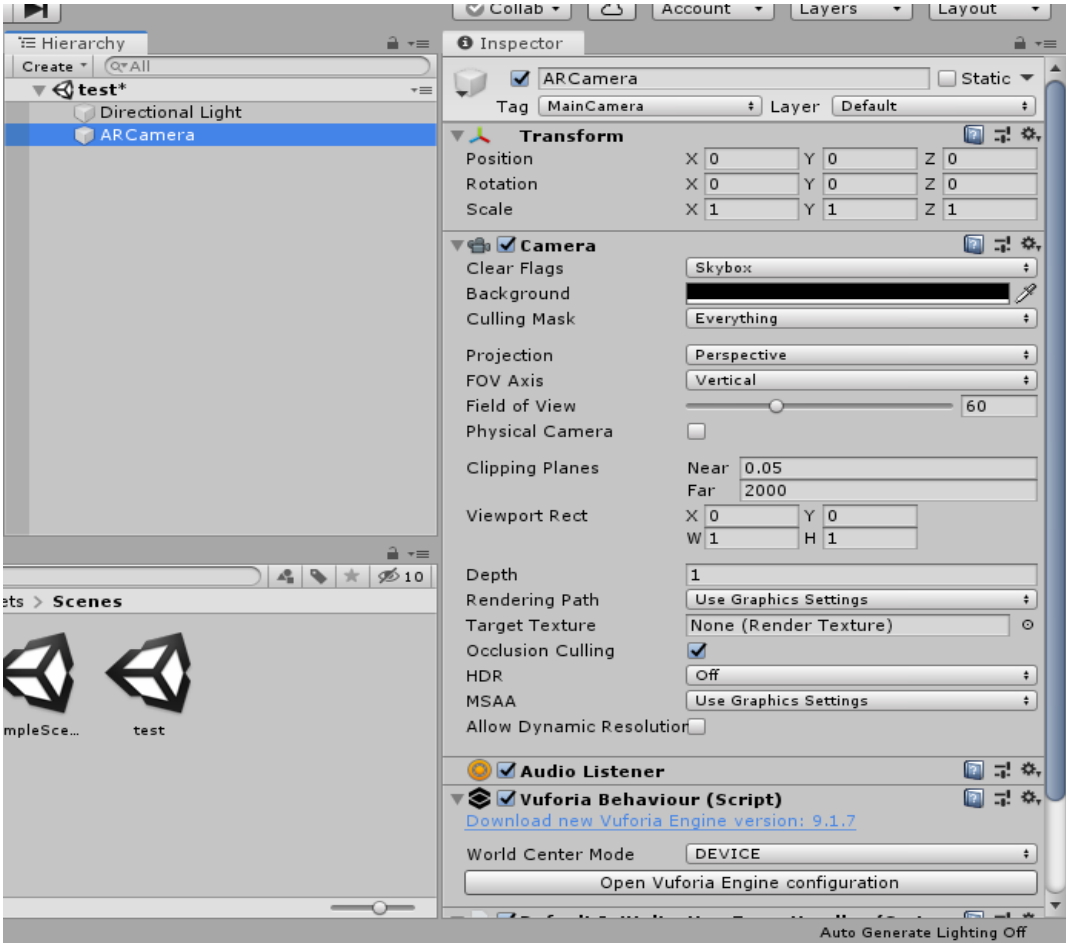
Πλέον χρειάζεται κανείς να έχει εγκαταστήσει το Unity Hub, με το οποίο μπορεί να διαχειριστεί τις διάφορες εκδόσεις Unity μαζί με τα plugins τους, και όλα τα projects του. Δημιουργήθηκε ένα νέο 3D project με την έκδοση 2019.2.13f1. Όταν ανοίγει ένα νέο project δημιουργείται αυτόματα μία νέα σκηνή (scene) και μέσα της μία main camera (κυρίως κάμερα) και ένα directional light (κατευθυνόμενο φως). Αυτά φαίνονται στην καρτέλα hierarchy (ιεραρχία) μέσα στην πλατφόρμα.



Εικόνα 2.1: Καρτέλες Scene και Hierarchy στον Editor

2.2 Vuforia Components

Κάθε project στο Unity χρειάζεται μία κάμερα. Σε ένα project Vuforia όμως χρειάζεται να αντικατασταθεί η Main camera που υπάρχει από τη δημιουργία κάθε νέου Unity project, από μία AR Camera. Με δεξί κλικ στην Main Camera στο hierarchy επιλέγοντας “Διαγραφή” αφαιρείται από τη σκηνή αυτό το component (συστατικό). Μετά από το μενού Game Object → Vuforia Engine → AR Camera προστίθεται μία AR Camera που είναι κατάλληλη για αυτού του είδους την εφαρμογή. Επιλέγοντας το συστατικό αυτό, δεξιά εμφανίζεται η καρτέλα Inspector (επιθεωρητής) που εμφανίζει όλα τα στοιχεία του συστατικού, και μέσω αυτού είναι επεξεργάσιμα (Εικόνα 2.2). Το πεδίο Projection πρέπει να παραμείνει στο Perspective για να φαίνεται η προοπτική στα αντικείμενα. Το Directional light παραμένει μέσα στη σκηνή διότι χρησιμεύει στο φωτισμό των επαυξημένων αντικειμένων.

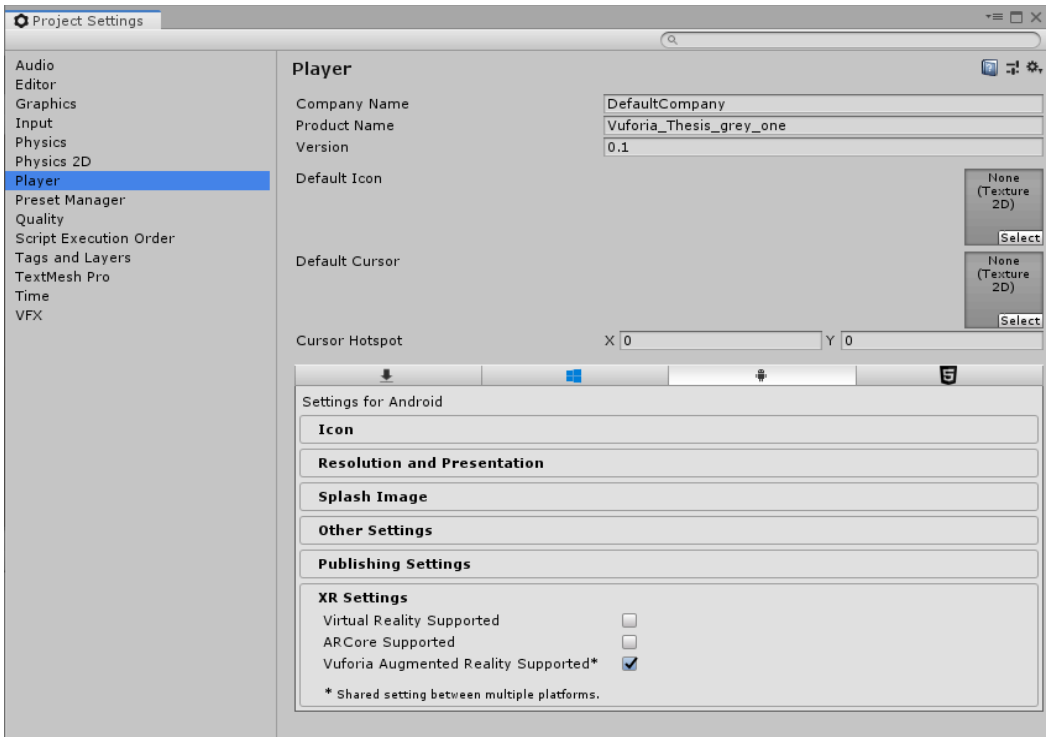


Εικόνα 2.2: Καρτέλα Inspector του συστατικού Ar Camera

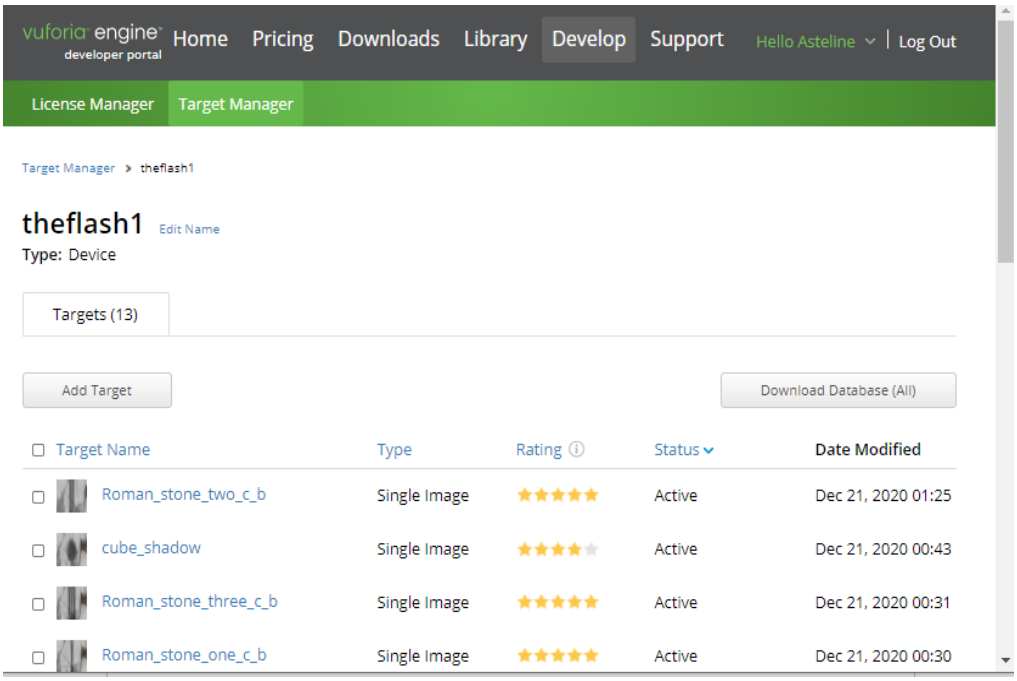
Εξίσου απαραίτητο είναι να ρυθμιστούν και κάποια άλλα στοιχεία του project για να είναι εύρυθμη και σωστή η λειτουργία των AR συστατικών στην εφαρμογή. Από το μενού Edit → Project settings στην καρτέλα Player στο υπομενού XR Settings πρέπει να τσεκαριστεί το πεδίο Vuforia Augmented Reality supported (Εικόνα 2.3).

Τέλος, από το μενού Window → Vuforia Configuration πρέπει να προστεθεί η άδεια (license key) για το Vuforia, που μπορεί να βγάλει κανείς από την επίσημη σελίδα του Vuforia, ώστε να μπορεί να προσθέσει τη βάση δεδομένων με τις εικόνες που θα χρησιμοποιηθούν στο project. Η άδεια αυτή είναι απαραίτητη για την εμποροποίηση του Vuforia project, ώστε εάν βγει σημαντικό κέρδος από την εφαρμογή, η χρήση του Vuforia kit δεν θα παραμείνει δωρεάν για τον δημιουργό..

Στην επίσημη ιστοσελίδα του Vuforia δημιουργείται ένας προσωπικός λογαριασμός, λαμβάνεται ένα license key, δημιουργείται μία βάση δεδομένων, και εκεί μπορούν να ανέβουν εικόνες ώστε να αναλυθούν και να εγκριθούν ως Image targets, και να κατέβει και να εισαχθεί αυτή η βάση στο project (Εικόνα 2.4).



Εικόνα 2.3: Project settings στην καρτέλα Player



Εικόνα 2.4: Vuforia site – Image target database

Πατώντας το κουμπί Download Database και επιλέγοντας το κατέβασμα για Unity editor κατεβαίνει ένα αρχείο τύπου .unitypackage, το οποίο μπορεί να εισαχθεί με ένα απλό drag and drop μέσα στο Unity project.

2.3 Χειριστές

Το project θα αποτελείται από μία σκηνή που θα περιέχει όλους τους γρίφους ως Image targets. Κάθε φορά που η κάμερα θα εντοπίζει μία από τις εικόνες της βάσης δεδομένων θα ενεργοποιείται και το αντίστοιχο Image target και θα επαυξάνονται τα αντικείμενα του εκάστοτε γρίφου επάνω στην εικόνα. Για να επικοινωνούν μεταξύ τους τα Image targets και να μοιράζονται δεδομένα προστέθηκαν κάποια εξωτερικά συστατικά.

2.3.1 Event System

Αρχικά προστέθηκε ένα Event System (Game Object → UI → Event System). Το Event System [7] είναι υπεύθυνο για διάφορες λειτουργίες που έχουν να κάνουν με το UI (διεπαφή του χρήστη) της εφαρμογής. Οι κύριες λειτουργίες του είναι:

- Η διαχείριση του ποιο Game Object θεωρείται επιλεγμένο
- Η διαχείριση του ποιο Input Module (ενότητα εισόδου) χρησιμοποιείται την κάθε στιγμή
- Η διαχείριση του Raycasting (παραγωγή ακτίνων) όπου είναι απαραίτητο
- Η ενημέρωση όλων των Input Modules όταν απαιτείται

2.3.2 UI Manager

Το UI Manager είναι ένα αντικείμενο που δημιουργήθηκε για να δρα ως ένας χειριστής της σκηνής. Σε αυτό το αντικείμενο είναι προσαρτημένα δύο scripts και ένα Audio Source (πηγή ήχου). Το Audio Source διαχειρίζεται ένα αρχείο ήχου που ενεργοποιείται για τον 2ο γρίφο (Κύβος).

Το πρώτο script είναι το Target Lister που διαχειρίζεται και εμφανίζει το ανιχνευμένο Image Target. Αυτό συμβαίνει ούτως ή άλλως εσωτερικά του Vuforia, αλλά χρειάστηκε να γραφτεί το script για να ελέγχουν άλλα scripts το ποιο Image Target είναι ενεργό κάθε στιγμή.

Το δεύτερο script είναι το UI Manager που περιέχει τρεις λειτουργίες που πρέπει να είναι global για να έχουν πρόσβαση όλα τα αντικείμενα και όλα τα scripts ανά πάσα στιγμή σε αυτές. Αρχικά κρατάει το Canvas, που θα αναλυθεί παρακάτω, από το να εξαφανιστεί από τη σκηνή ό,τι και να γίνει. Περιέχει τη μέθοδο rotate() που περιστρέφει κάποια αντικείμενα πατώντας το κουμπί του Canvas. Και τέλος, ελέγχει για το αν περιστράφηκε κάποιο συγκεκριμένο είδος αντικειμένου (Slice) για τον 2ο γρίφο (Κύβος), ώστε να αναπαραχθεί ο ήχος του Audio Source.

2.3.3 Canvas

Το Canvas είναι η περιοχή στην οποία πρέπει να περιέχονται όλα τα στοιχεία UI. Εφόσον προστεθεί κάποιο UI στοιχείο στο project, προστίθεται αυτόματα ένα Canvas και μέσα του, ως "παιδί" μπαίνει το νέο UI στοιχείο [8]. Φαίνεται ως ένα τετράγωνο, με την πρόσοψή του προς την κάμερα, ώστε να είναι εύκολο να τοποθετηθούν τα UI στοιχεία. Το Canvas χρησιμοποιεί το Event System για να περνάει μηνύματα εσωτερικά της εφαρμογής. Στο πεδίο Render mode επιλέγεται το Screen space – Overlay και στο πεδίο UI Scale Mode επιλέγεται το Scale with Screen Size, ώστε το Canvas να προσαρμόζεται στην οθόνη της εκάστοτε συσκευής που τρέχει το παιχνίδι.

Για αυτή την εφαρμογή χρειάστηκαν δύο UI στοιχεία. Το πρώτο ήταν το κουμπί Rotate. Τοποθετήθηκε κάτω δεξιά στο Panel από το μενού GameObject → UI → Button. Αφαιρέθηκε το συστατικό Text που δημιουργήθηκε αυτόματα μέσα στο κουμπί και άλλαξε το εικονίδιο του κουμπιού σε ένα κόκκινο βελάκι που δείχνει την κίνηση της περιστροφής. Το κύριο συστατικό του κουμπιού είναι το Button που δημιουργείται από τον Editor και περιέχει τη μέθοδο OnClick() που εκτελείται όταν πατηθεί το κουμπί. Σε εκείνο το πεδίο προστέθηκε το αντικείμενο UI Manager και από εκείνο, το script UI Manager, και από αυτό καλείται η μέθοδος rotate(). (Εικόνα 2.6). Στο Unity, σε τέτοιου είδους πεδία, δεν μπορούν να ανατεθούν απευθείας scripts, αλλά συστατικά και αντικείμενα που περιέχουν τα scripts αυτά. Θα μπορούσε να προστεθεί το script απευθείας στο συστατικό του κουμπιού και να καλεί τον εαυτό του για να προσπελάσει το script, όπως συμβαίνει σε κάποια συστατικά αυτού του project.

Η μέθοδος *void rotate()*:

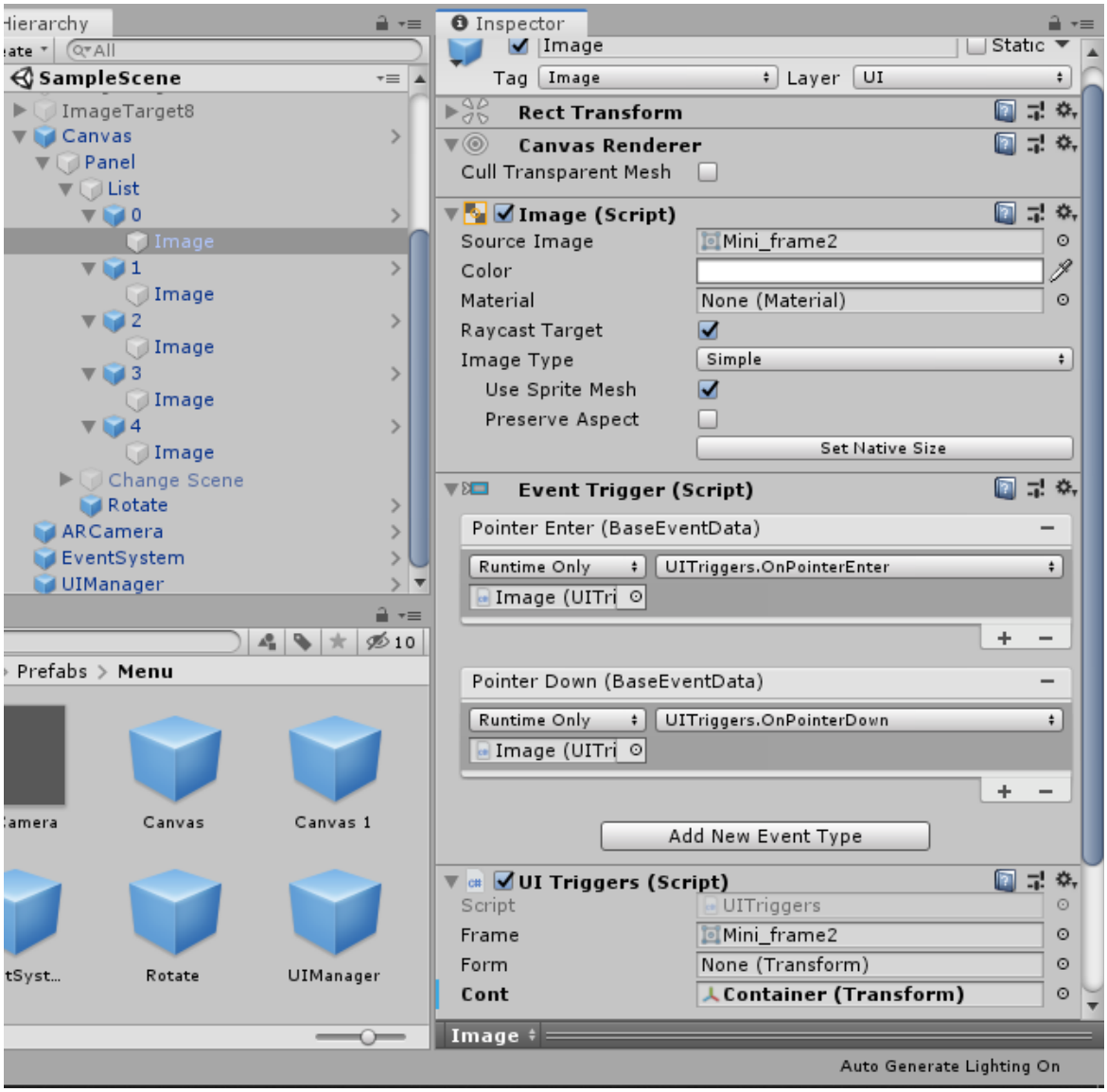
```
//το pplane3 είναι το ενεργό αντικείμενο που αγγίζει ο παίκτης και τίθεται στο script MovePlane
if (pplane3 != null){
    pplane3.transform.Rotate(0f, 90f, 0f, Space.Self);
    //γυρνάει το αντικείμενο κατά 90 μοίρες στον άξονα ψ σε σχέση με το ίδιο
    if (pplane3.CompareTag("Slice")){       //αν το αντικείμενο είναι τύπου Slice
            rusty.Play();     }}       //αναπαράγει τον ήχο rusty που τέθηκε στον inspector
```

Το δεύτερο ήταν το Inventory (λίστα αποθηκευμένων αντικειμένων), το οποίο δίνει τη δυνατότητα στον παίκτη να αποθηκεύει αντικείμενα που παίρνει από ένα Image Target και να τα μεταφέρει σε κάποιο άλλο. Εφόσον φύγει η κάμερα από κάποιο Image Target, εξαφανίζονται όλα τα αντικείμενα που υπήρχαν πάνω του, οπότε χρειαζόταν κάτι που μένει σταθερό καθ'όλη τη διάρκεια του παιχνιδιού ώστε να μπορούν να μεταφερθούν αντικείμενα από το ένα Image target στο άλλο.

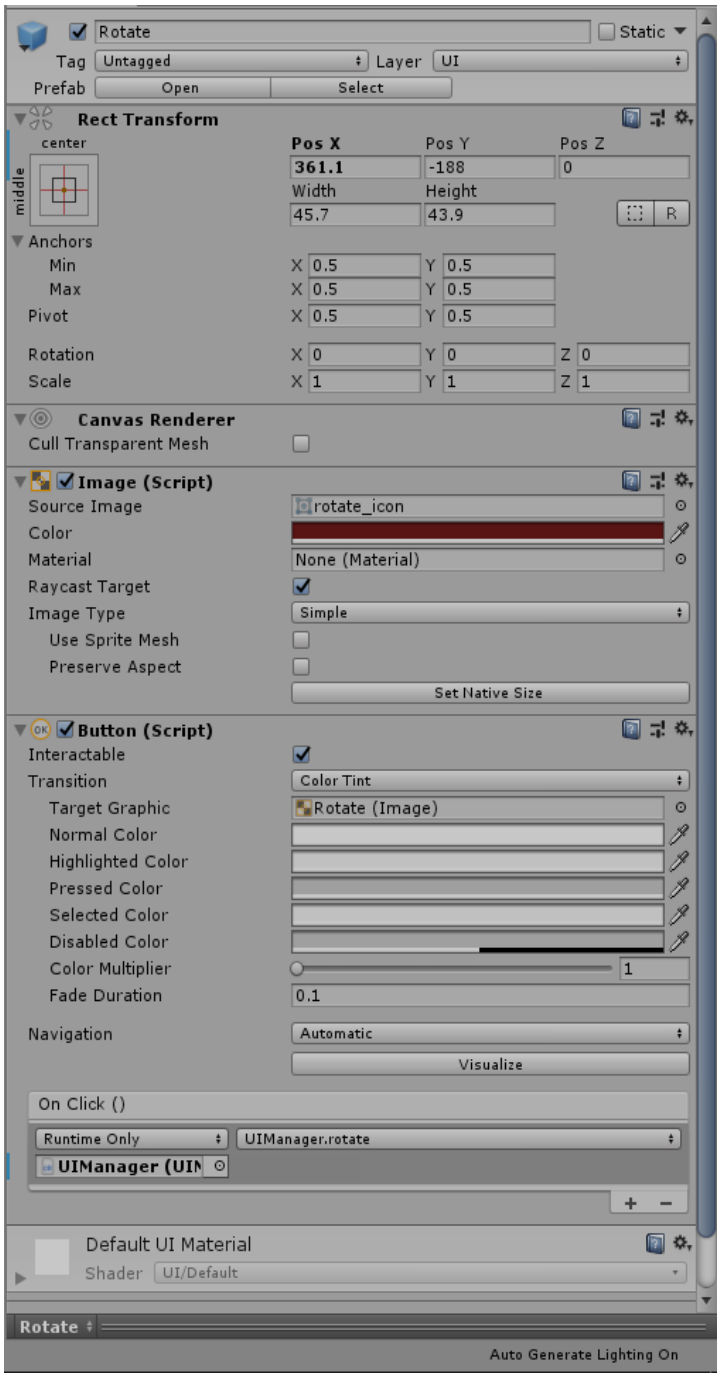
Μέσα στο Canvas μπαίνει ένα GameObject → UI → Panel για την ομαδοποίηση των UI στοιχείων που θα περιέχονται. Ένα Canvas μπορεί να περιέχει και περισσότερα Panels. Για το Inventory προστέθηκε ένα ακόμα αντικείμενο στην ιεραρχία, ένα κενό GameObject με το όνομα

List. Το Inventory θα αποτελείται από πέντε κενά τετράγωνα τα οποία θα υπάρχουν στο αριστερό μέρος της οθόνης για όσο "τρέχει" η εφαρμογή. Όταν επιλέγει ο παίκτης ένα αντικείμενο το οποίο μπορεί να μεταφέρει αγγίζοντας το στην οθόνη και σέρνοντας το δάκτυλό του, θα μπορεί να το σύρει σε ένα από αυτά τα τετράγωνα και το αντικείμενο θα εξαφανίζεται από τη σκηνή και θα μεταφέρεται μέσα στο τετράγωνο του Inventory.

Κάθε τετράγωνο ονομάστηκε με τον αριθμό της θέσης του για ευκολία του κώδικα, και μέσα στο αριθμημένο αντικείμενο μπήκε ένα αντικείμενο εικόνας (Εικόνα 2.5). Τα τετράγωνα μπήκαν στο αντικείμενο List για την ομαδοποίησή τους.



Εικόνα 2.5: Hierarchy και Inspector του Inventory



Εικόνα 2.6: Το κουμπί Rotate στον Inspector

Το Rect Transform, Canvas Renderer και το Image (Script) είναι τα βασικά στοιχεία αυτού του συστατικού που δημιουργούνται αυτόματα από τον editor. Το Event Trigger και UI Triggers προστέθηκαν στο συστατικό. Το UI Triggers είναι το αρχείο κώδικα που γράφτηκε για να εκτελεί κάποιες από τις λειτουργίες του UI. Το Unity εμφανίζει στον Inspector το αρχείο κώδικα, και σαν μεταβλητά πεδία τις public μεταβλητές που ορίζονται μέσα σε αυτό. Με ένα απλό drag and drop μπορεί κανείς να αποδώσει τιμές από την ιεραρχία ή από τα αρχεία του project στα πεδία αυτά. Για παράδειγμα το πεδίο Frame που φαίνεται στην εικόνα είναι το εικονίδιο που έχουν τα τετράγωνα του Inventory.

Το Event Trigger περιέχει δύο κλασσικά triggers, το Pointer Enter και Pointer Down, στα οποία έχουν αποδοθεί δύο συναρτήσεις του script UI Triggers (Εικόνα 2.7 & 2.8).

```
24      void Start()
25      {
26          thescript = GameObject.Find("UIManager").GetComponent<TargetLister> ();
27          //παίρνει το script TargetLister του component UIManager
28          img = GetComponent<Image>();
29          //παίρνει την εικόνα του παρόντος στοιχείου που τίθεται από τον inspector
30          sc = new Vector3(0.25f, 0.085f, 0.25f);
31          //τίθεται το transform για κάποια συγκεκριμένα αντικείμενα
32          myarrayhere = new GameObject[5];
33          //δημιουργεί έναν πίνακα με 5 θέσεις
34      }
35
36      public void OnPointerEnter(){ //ενεργοποιείται όταν μπει ο δείκτης στην περιοχή του τετραγώνου
37          imgt = thescript.img; //παίρνει την εικόνα του ανιχνευμένου Image Target από το UIManager
38          MovePlane plsc = GameObject.Find(imgt).GetComponent<MovePlane> ();
39          //παίρνει το script MovePlane από το ανιχνευμένο Image Target
40          pplane2 = plsc.pplane; //παίρνει το αντικείμενο που σέρνει ο παίκτης αυτή τη στιγμή
41          if (pplane2!=null){
42              newimage = pplane2.GetComponent<Image>().sprite;
43              //παίρνει την εσωτερική εικόνα που κουβαλάει το αντικείμενο
44              if (pplane2!= null){
45                  img.sprite = newimage;
46                  //αντικαθιστά την εικόνα του τετραγώνου με αυτή του αντικειμένου
47                  cell = transform.parent;    //παίρνει τον γονέα του τετραγώνου που είναι αριθμός
48                  namee = cell.name;          //το όνομα του γονέα
49                  x = Convert.ToInt32(namee); //κάνει το όνομα του γονέα τύπου integer
50                  //Debug.Log(x);
51                  myarrayhere[x] = pplane2;   //βάζει στον global πίνακα στην αντίστοιχη θέση το αντικείμενο
52                  tr = GameObject.Find(imgt); //βρίσκει το Image Target ως Game Object
53                  form = tr.transform;        //παίρνει το transform του Image Target
54                  pplane2.SetActive(false);   //απενεργοποιεί το αντικείμενο και το εξαφανίζει
55              }
```

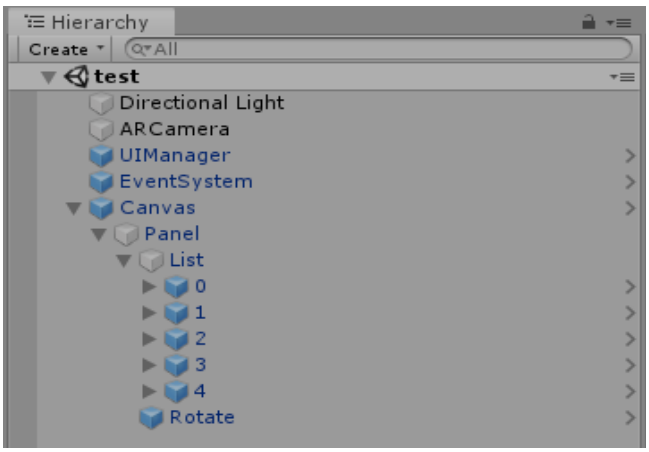
Εικόνα 2.7: Το πρώτο μέρος του script UI Triggers

```
public void OnPointerDown(){  //ενεργοποιείται όταν πατήσει ο παίκτης πάνω σε κάποιο τετράγωνο
    namee = transform.parent.gameObject.name; //παίρνει το όνομα του τετραγώνου
    x = Convert.ToInt32(namee);         //κάνει το όνομα του τετραγώνου τύπου integer
    if (myarrayhere[x]!=null)           //ελεγχος για καταλάθος κενή καταχώρηση στον πίνακα
    {
        pplane2 = myarrayhere[x];
        //θέτει το επιλεγμένο αντικείμενο αυτό που βρίσκεται στην συγκεκριμένη θέση του πίνακα myarrayhere
        imgt = thescript.img;           //παίρνει την ανιχνευμένη εικόνα
        tr = GameObject.Find(imgt);     //παίρνει την ανιχνευμένη εικόνα ως Game Object
        form = tr.transform;            //παίρνει το transform του Image Target
        pplane2.transform.SetParent(form);  //κάνει το επιλεγμένο αντικείμενο παιδί του ανιχνευμένου Image Target
        if (tr.name == "ImageTarget3"){     //αν το ανιχνευμένο Image Target είναι το ImageTarget3
            pplane2.GetComponent<SpinSlice>().enabled = false; //σταματάει το script περιστροφής του αντικειμένου
            pplane2.transform.SetParent(cont);  //θέτει ως γονέα του αντικειμένου το container που τέθηκε στον inspector
            pplane2.transform.localEulerAngles = new Vector3(0f,0f,0f);
            //θέτει τον προσανατολισμό του αντικειμένου στο 0 σε σχέση με το container
            pplane2.transform.localScale = sc;  //θέτει το μέγεθος του αντικειμένου όπως έχει οριστεί στο Start()
        }

        dist = Vector3.Distance(Camera.main.transform.position, tr.transform.position)- 0.3f;
        //η απόσταση της κάμερας από το Image Target μείον 0.3 βαθμούς
        Touch touch = Input.GetTouch(0);    //το άγγιγμα της οθόνης
        touchedPos = Camera.main.ScreenToWorldPoint(new Vector3(touch.position.x, touch.position.y, dist));
        //η θέση του δακτύλου στην οθόνη μεταφρασμένη στο περιβάλλον της σκηνής, αλλά με την τρίτη διάσταση
        //ορισμένη ως το dist
        pplane2.transform.position = touchedPos; //θέτει τη θέση του αντικειμένου με βάση τη θέση του δακτύλου στην οθόνη

        pplane2.SetActive(true);        //ενεργοποιεί και εμφανίζει το αντικείμενο
        pplane2.GetComponent<Renderer>().enabled = true;    //ενεργοποιεί τον Renderer του αντικειμένου
        pplane2.GetComponent<Collider>().enabled = true;    //ενεργοποιεί τον Collider του αντικειμένου
        //οι δύο παραπάνω εντολές είναι απαραίτητες διότι κανονικά αυτά τα συστατικά ενεργοποιούνται
        //αυτόματα όταν ανιχνευτεί ένα Image Target, όπου εδώ θα είναι ήδη ανιχνευμένο
        img.sprite = frame;         //επαναφέρεται η εικόνα του τετραγώνου του inventory
    }
```

Εικόνα 2.8: Το δεύτερο μέρος του script UI Triggers

2.4 Επίλογος

Σε αυτό το σημείο είναι έτοιμα τα στοιχεία που χρειάζονται για να λειτουργεί το περιβάλλον της εφαρμογής, τα στοιχεία δηλαδή που θα παραμένουν σταθερά όποια εικόνα και να ανιχνεύεται, και τα scripts που ανταλλάσσουν πληροφορία μεταξύ των συστατικών της εφαρμογής (Εικόνα 2.9).

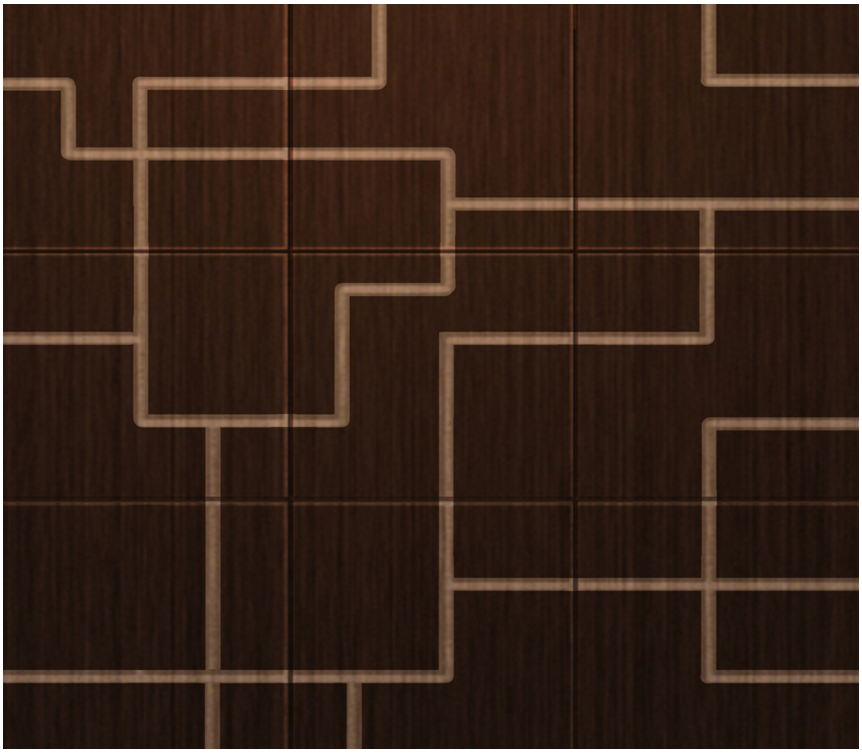


Εικόνα 2.9: Τα συστατικά περιβάλλοντος της εφαρμογής (essentials) στο Hierarchy

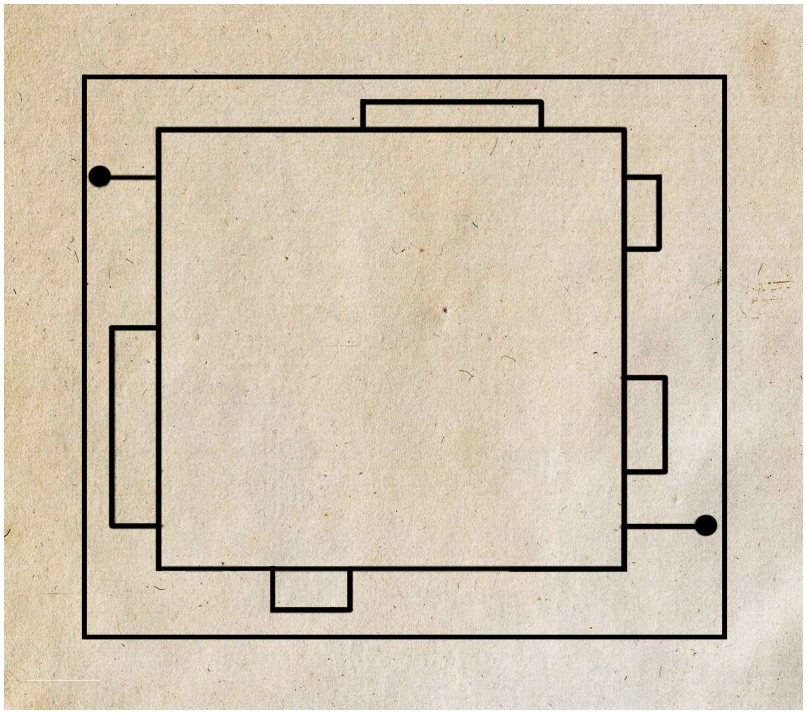
Κεφάλαιο 3ο 1ος γρίφος - Λαβύρινθος

3.1 Εισαγωγή

Διάφορα παιχνίδια του είδους puzzle (γρίφος) περιέχουν κάποιον γρίφο στον οποίο ο παίκτης καλείται να βρει το σωστό μοτίβο κάποιων κομματιών, ώστε όταν τα βάλει στις σωστές θέσεις να σχηματίζεται μια συγκεκριμένη εικόνα. Για να είναι όσο το δυνατόν πιο μινιμαλιστικό το παιχνίδι, για αυτόν τον γρίφο σχεδιάστηκε ένας λαβύρινθος, ο οποίος είναι χωρισμένος σε εννέα τετράγωνα κομμάτια που πρέπει να μπουν στη σειρά για να σχηματιστεί ολόκληρη η εικόνα. Τα κομμάτια εμφανίζονται σε σκόρπιες θέσεις, πάνω από την εικόνα του πλαισίου στο οποίο πρέπει να ταιριάξουν. Ο παίκτης δεν καλείται μόνο να τα βάλει στη σωστή θέση, αλλά και στο σωστό προσανατολισμό. Οι γραμμές πρέπει να ενώνονται, τόσο μεταξύ των κομματιών, αλλά και με το εξωτερικό πλαίσιο. Όταν ο παίκτης βάλει τα κομμάτια στις σωστές θέσεις και με το σωστό προσανατολισμό, θα εμφανιστεί ολοκληρωμένη η εικόνα του λαβυρίνθου.



Εικόνα 3.1: Ο Λαβύρινθος – Τα κομμάτια του λαβυρίνθου ενωμένα



Εικόνα 3.2: Το πλαίσιο του λαβυρίνθου στο οποίο πρέπει να ταιριάξει

3.2 Αντικείμενα του λαβυρίνθου

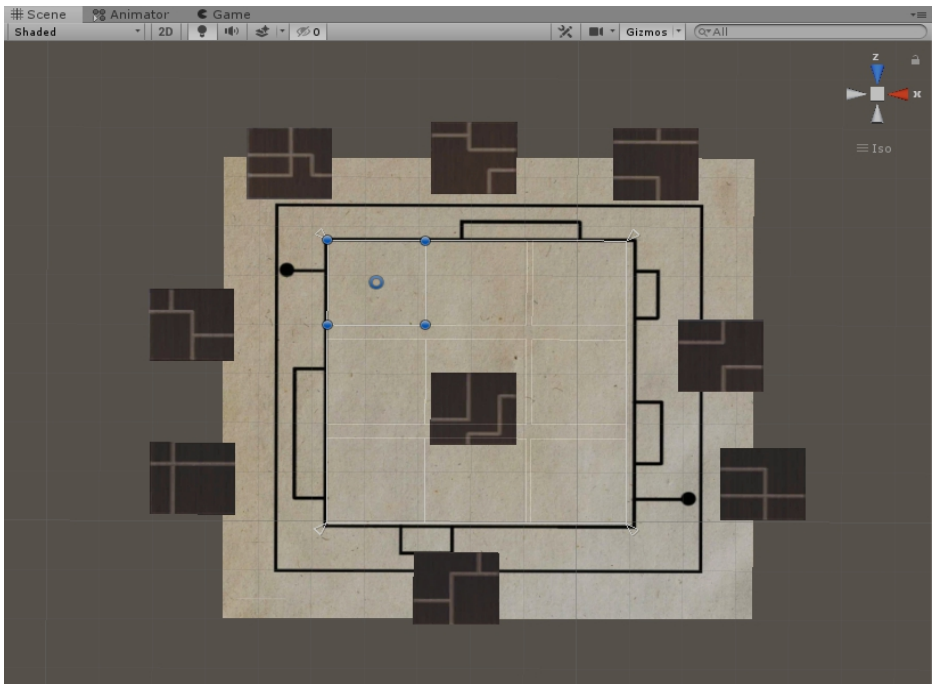
Για κάθε γρίφο χρειάζεται μία ή και παραπάνω εικόνες που θα ανιχνεύονται και θα επαυξάνονται αντικείμενα επάνω τους. Αυτές εισάγονται ως Image Targets από το μενού GameObject → Vuforia Engine → Image. Οπότε προστέθηκε ένα Image Target για τον πρώτο γρίφο, και στον Inspector στα πεδία του script Image Target Behavior καταχωρήθηκε η βάση με τις εικόνες που εισήχθη από το site του Vuforia, και επιλέχθηκε η εικόνα frame (Εικόνα 3.2). Οπότε η βάση του κάθε γρίφου θα είναι ένα Image Target, δηλαδή μία επίπεδη εικόνα που θα πρέπει να εκτυπωθεί για να ανιχνεύεται από την εφαρμογή, και που πάνω της θα επαυξάνονται τα εκάστοτε αντικείμενα. Όποια αντικείμενα προστεθούν για τον γρίφο, θα εισαχθούν μέσα στο Image Target.

Δοκιμάστηκαν αρκετοί τρόποι με τους οποίους θα ελεγχόταν αν ο παίκτης έλυσε τον γρίφο. Η τεχνική που επικράτησε και που έδινε τα πιο ορθά αποτελέσματα ήταν η εξής. Προστέθηκε ένα πλέγμα από αόρατα τετράγωνα πάνω από το Image Target, εννιά τετράγωνα που όλα μαζί σχηματίζουν ένα τετράγωνο, όπως πρέπει να πραγματοποιηθεί και με τον λαβύρινθο. Σε αυτά τα αόρατα τετράγωνα ανατέθηκε να εκπέμπουν από μία αόρατη ακτίνα προς τα πάνω. Σε κάθε frame του παιχνιδιού, όσο ανιχνεύεται αυτό το Image Target φυσικά, θα ελέγχεται αν η κάθε ακτίνα "χτυπάει" το σωστό κομμάτι του λαβυρίνθου, για να διαβεβαιώνεται ότι το κάθε κομμάτι είναι στη σωστή θέση, όπως επίσης θα ελέγχεται αν το κομμάτι έχει τον σωστό προσανατολισμό. Τα

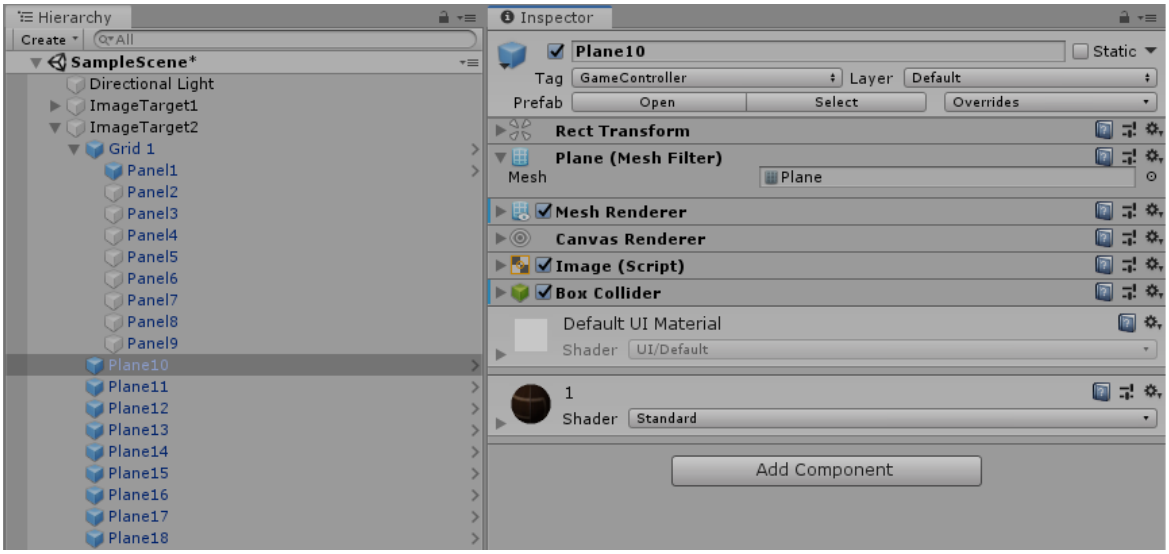
κομμάτια εμφανίζονται διασκορπισμένα λίγο πιο ψηλά από το Image Target και το αόρατο πλέγμα και σε διάφορους προσανατολισμούς.

Για το πλέγμα, προστέθηκε ένα νέο GameObject → UI → Canvas που ονομάστηκε Grid 1. Μέσα του προστέθηκαν εννέα GameObject → UI → Panel που ονομάστηκαν Panel1 – 9. Χάριν αυτοματισμού, πρώτα δημιουργήθηκε ένα Panel, το οποίο τροποποιήθηκε για να καλύπτει τις ανάγκες του γρίφου, κι έπειτα αποθηκεύτηκε ως Prefab στο φάκελο Prefabs → Maze του project , και μετά προστέθηκε και μετονομάστηκε άλλες οκτώ φορές μέσα στο Grid 1. Prefab είναι ένα αντικείμενο στο Unity το οποίο έχει δημιουργηθεί μέσα σε μια σκηνή και έχει αποθηκευτεί στους φακέλους του project ώστε να μπορεί να επαναχρησιμοποιηθεί με τις ίδιες επεξεργασίες που του έχουν υποβληθεί, ώστε να μην χρειαστεί να τα ξαναδημιουργήσει ο χρήστης από την αρχή [9]. Απενεργοποιήθηκε το συστατικό Image που έχουν αυτόματα τα Panels μιας και θεωρήθηκε ότι δεν θα ήταν καλαίσθητο να φαίνονται στο παιχνίδι. Στο prefab Plane και στο Grid 1 προστέθηκε από ένα script που θα αναλυθούν στο επόμενο υποκεφάλαιο.

Με τον ίδιο τρόπο δημιουργήθηκαν και τα κομμάτια του λαβυρίνθου, ως GameObject → 3D Object → Plane. Δημιουργήθηκε πρώτα ένα Plane (επίπεδο) με συστατικά Mesh Filter [10], Mesh Renderer, το οποίο είναι υπεύθυνο για την εμφάνιση του αντικειμένου [11], Image, και ένα Box Collider, που είναι υπεύθυνο για τη μάζα του αντικειμένου και τις συγκρούσεις με αυτό [12]. Αυτά τα συστατικά δημιουργούνται αυτόματα με κάθε 3D αντικείμενο. Το Plane αποθηκεύτηκε στα Prefabs και προστέθηκε άλλες οκτώ φορές στη σκηνή σε διάφορα σημεία (Plane10 - 18).



Εικόνα 3.3: Η σκηνή του Image Target λαβύρινθος



Εικόνα 3.4: Hierarchy του Image Target λαβύρινθος και Inspector του αντικειμένου Plane10

Στο αντικείμενο Plane ανατέθηκε το Tag GameController για να ξεχωρίζει από τα άλλα αντικείμενα και να μπορεί να γίνει αναφορά σε αυτό το είδος αντικειμένου μέσα στον κώδικα.

3.3 Αρχεία κώδικα για τον λαβύρινθο

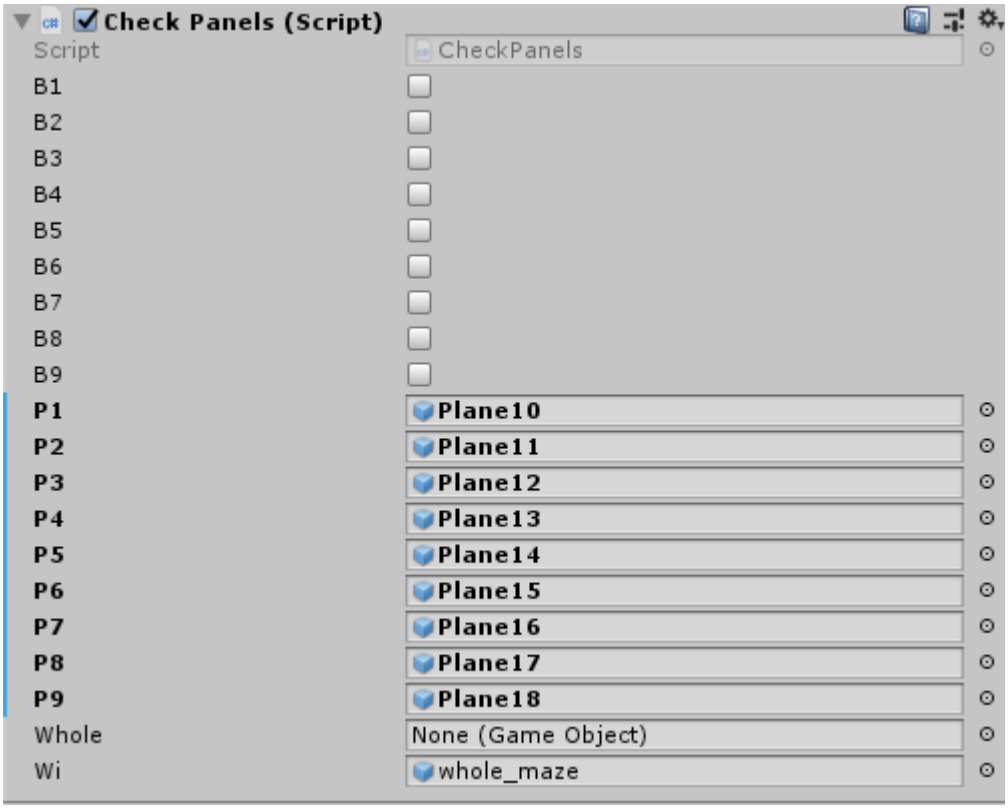
Η μέθοδος Update καλείται αυτόματα από το Unity σε κάθε frame του παιχνιδιού [13], αν είναι ενεργοποιημένο το MonoBehavior. Το MonoBehavior είναι η βασική κλάση από όπου κληρονομεί κάθε script στο Unity. Αν ένα script είναι γραμμένο σε γλώσσα προγραμματισμού C#, όπως όλα τα scripts αυτού του project, πρέπει οπωσδήποτε να δηλώνεται ότι κληρονομεί από την κλάση MonoBehavior [14].

3.3.1 Check Panels

Το τελευταίο και πιο απλό αρχείο κώδικα που γράφτηκε για αυτόν τον γρίφο ήταν το Check Panels (Εικόνα 3.5). Αυτό το αρχείο κώδικα προστέθηκε ως συστατικό στο αντικείμενο Grid 1 (Εικόνα 3.6) και ελέγχει στην μέθοδο Update αν έχουν μπει όλα τα τετράγωνα στις σωστές θέσεις και με το σωστό προσανατολισμό. Αν ισχύει αυτό, απενεργοποιεί τα εννιά κομμάτια, άρα εξαφανίζονται από τη σκηνή, και δημιουργεί ένα νέο αντικείμενο, την ολοκληρωμένη εικόνα του λαβυρίνθου (Εικόνα 3.1).

```
using System.Collections;
using System.Collections.Generic;
using UnityEngine;

public class CheckPanels : MonoBehaviour
{
    public bool b1, b2, b3, b4, b5, b6, b7, b8, b9;
    //οι boolean μεταβλητές που τίθενται από το script Panel
    public GameObject p1, p2, p3, p4, p5, p6, p7, p8, p9, whole, wi;
    //τα Panels που τίθενται από την ιεραρχία στον Inspector
    //το wi είναι ο ολόκληρος λαβύρινθος που τέθηκε στον Inspector

    void Update()
    {
        if (b1 && b2 && b3 && b4 && b5 && b6 && b7 && b8 && b9){
        //αν είναι όλα τα κομάτια στις σωστές θέσεις και στροφές
            p1.SetActive(false); //απενεργοποιούνται
            p2.SetActive(false); //τα panels στην σκηνή
            p3.SetActive(false); //και εξαφανίζονται
            p4.SetActive(false);
            p5.SetActive(false);
            p6.SetActive(false);
            p7.SetActive(false);
            p8.SetActive(false);
            p9.SetActive(false);

            whole = Instantiate (wi, this.transform);
            //δημιουργείται ένα νέο αντικείμενο (ο ολόκληρος λαβύρινθος)
            //με parent (γονέα) το ίδιο το Grid 1
        }
    }
}
```

Εικόνα 3.5: Το script Check Panels



Εικόνα 3.6: Το script Check Panels στον Inspector του Grid 1

3.3.2 Panel

Το δεύτερο script που γράφτηκε για τον λαβύρινθο ήταν το Panel που μπήκε σαν συστατικό στο prefab Panel, άρα και στα εννιά panels που υπάρχουν στη σκηνή. Σε αυτό, στη μέθοδο Update(), άρα σε κάθε frame, εκπέμπεται μία ακτίνα από το plane και, ανάλογα με το ποιο plane είναι, ελέγχεται το αν η ακτίνα του χτυπάει το σωστό plane. Αν είναι το σωστό, ελέγχεται αν βρίσκεται στο σωστό προσανατολισμό. Αν ισχύουν αυτές οι δύο συνθήκες, η αντίστοιχη Boolean μεταβλητή του script Check Panels γίνεται αληθής (true). Αν πάλι, το σωστό plane φύγει από τη σωστή θέση, αυτή η μεταβλητή επανέρχεται σε ψευδή (false) κατάσταση.

Κομμάτι κώδικα του script Panel:

```
void Start() {
    chscr = transform.parent.GetComponent<CheckPanels>();  }
    //με το που φορτωθεί το script, σε αυτή τη μεταβλητή μπαίνει το script Check Panels

void Update() {
```

```
.....
if (Physics.Raycast (pan.transform.position, fwd, out objectHit, 50)) {
//αν χτυπήσει κάτι η ακτίνα
    hit  = objectHit.collider.gameObject; //το αντικείμενο που χτυπήθηκε
    switch (pan.name){                    //ανάλογα το panel στο οποίο έχει προστεθεί το script
    case "Panel1":                        //αν το παρόν panel είναι το Panel1
        if (hit.name == "Plane10"){       //αν το plane που χτυπήθηκε είναι το Plane10
            if (hit.transform.localEulerAngles.y == 180f){
    //αν η περιστροφή του στον άξονα ψ σε σχέση με τον άμεσο γονέα του είναι 180 μοίρες
                chscr.b1 = true;}
    //η αντίστοιχη Boolean μεταβλητή του script Check Panels γίνεται true
            else {chscr.b1 = false;}   }        //αλλιώς γίνεται false
       else {chscr.b1 = false;}                 //αλλιώς γίνεται false
    break;
    .....
```

Στη συνέχεια επαναλαμβάνονται οκτώ cases για τα υπόλοιπα panels και άλλη μια μέθοδος switch για να απενεργοποιεί τις Boolean μεταβλητές αν φύγει το σωστό panel από τη θέση του.

3.3.3 Move Plane

Το τρίτο και πιο σημαντικό script που γράφτηκε για αυτόν τον γρίφο αλλά χρησιμοποιήθηκε και στον δεύτερο γρίφο, είναι το Move Plane (Εικόνα 3.7). Αυτό προστέθηκε στο ίδιο το Image Target ώστε να μπορεί ο παίκτης να κινεί αντικείμενα μέσα στο χώρο όσο είναι ενεργό αυτό το Image Target. Το script αυτό χρησιμοποιεί τη βιβλιοθήκη UnityEngine [15], όπως συμβαίνει με όλα τα scripts Unity σε C#. Μέσω αυτής της βιβλιοθήκης μπορεί κανείς να καλέσει μεθόδους και δομές όπως τις δομές Touch, Ray, RaycastHit, τον τύπο TouchPhase, ή την κλάση Physics.

Αυτό το script διαχειρίζεται τα αγγίγματα της οθόνης όσον αφορά τη μετακίνηση κάποιων αντικειμένων από τον παίκτη μέσα στη σκηνή. Στο παιχνίδι, αυτό χρειάζεται και είναι εφικτό σε δύο γρίφους, στον λαβύρινθο και στον κύβο. Τα αντικείμενα που έχουν τη δυνατότητα να μετακινηθούν είναι αυτά των τύπων GameController (τα κομμάτια του λαβυρίνθου) και Slice (τα κομμάτια του κύβου). Ειδάλλως ο παίκτης θα μπορούσε να αλλάξει τη θέση όλων των αντικειμένων στο παιχνίδι, όπως αυτά του περιβάλλοντος. Έτσι το Move Plane ελέγχει αν αγγίζεται ένα τέτοιο αντικείμενο, και το μετακινεί με την κατεύθυνση που κινείται και το άγγιγμα στην οθόνη. Επίσης θέτει το ενεργό αντικείμενο για άλλα scripts που μπορεί να το χρησιμοποιούν, όπως η μέθοδος rotate() του script UI Manager. Ο κώδικας είναι γραμμένος για οθόνες αφής. Υπάρχουν διαφορετικές μέθοδοι και δομές για τις ίδιες λειτουργίες με κέρσορα ποντικιού, όμως αυτό το παιχνίδι δεν είναι δυνατόν να παιχτεί δίχως οθόνη αφής, οπότε δεν έχουν γραφτεί παραλλαγές του κώδικα για άλλες συνθήκες.

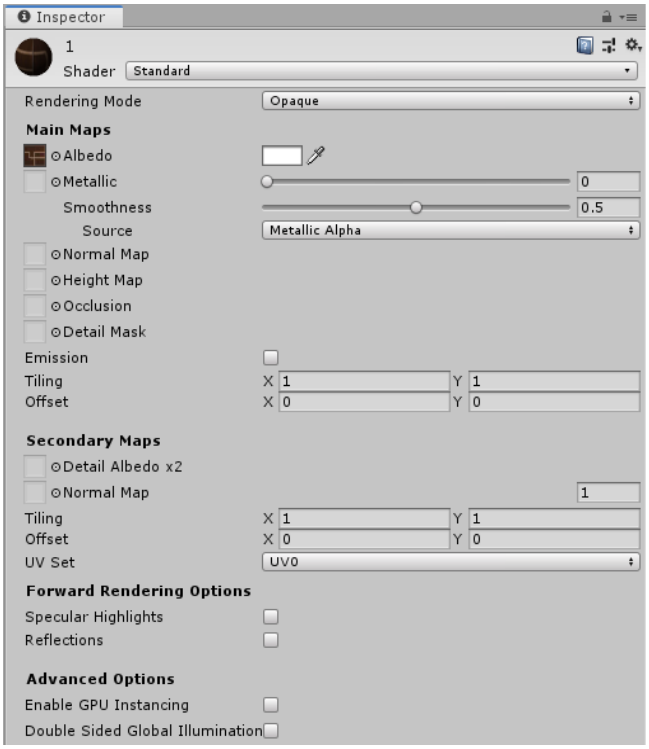
```
using System.Collections;
using System.Collections.Generic;
using UnityEngine;
using UnityEngine.UI;
using UnityEngine.EventSystems;
using Vuforia;

public class MovePlane : MonoBehaviour
{
    public GameObject pplane;
    private GameObject img;
    public Vector3 touchedPos;
    float depth, dist;
    private TargetLister thescript;
    private UIManager uiman;
    public string tracked;

void Start()
{
    thescript = GameObject.Find("UIManager").GetComponent<TargetLister> ();
    //παίρνει το script Target Lister του αντικειμένου UI Manager
    uiman = GameObject.Find("UIManager").GetComponent<UIManager> ();
    //παίρνει το script UI Manager του αντικειμένου UI Manager
}

// Update is called once per frame
void Update()
{
    if (Input.touchCount > 0)   //αν υπάρχει άγγιγμα στην οθόνη
    {
        Touch touch = Input.GetTouch(0);    //παίρνει το άγγιγμα σαν μεταβλητή

        if (touch.phase == TouchPhase.Began){   //αν μόλις ξεκίνησε το άγγιγμα
            Ray raycast = Camera.main.ScreenPointToRay(Input.GetTouch(0).position);
            //μεταφράζει το άγγιγμα σε ακτίνα
            RaycastHit raycastHit;
            if (Physics.Raycast(raycast, out raycastHit))
            {
                if ((raycastHit.collider.CompareTag("GameController")) || (raycastHit.collider.CompareTag("Slice"))){
                    //αν αυτό που άγγιξε η ακτίνα με το άγγιγμα της οθόνης είναι τύπου GameController ή Slice
                    pplane = GameObject.Find(raycastHit.collider.name);
                    //τίθεται το ενεργό αντικείμενο ως αυτό που επιλέχθηκε
                    //αν δεν είναι τέτοιου τύπου δεν είναι θεμητό να μπορεί ο παίκτης να το κουνήσει
                    uiman.pplane3 = pplane;         //τίθεται το ίδιο και στο script UI Manager για τη μέθοδο rotate
                    tracked = thescript.img;        //παίρνει το ενεργό Image Target
                    img = GameObject.Find(tracked); //το ενεργό Image Target ως GameObject
                }
            }
        }

        if (touch.phase == TouchPhase.Moved)    //αν το άγγιγμα συνεχίστηκε με κίνηση
        {
            if(pplane!=null){                   //αν υπάρχει ενεργό αντικείμενο
                dist = Vector3.Distance(Camera.main.transform.position, img.transform.position)- 0.3f;
                //η απόσταση της εικόνας Image Target από την κάμερα μείον 0.3 μονάδες
                touchedPos = Camera.main.ScreenToWorldPoint(new Vector3(touch.position.x, touch.position.y, dist));
                //το σημείο που αγγίχτηκε η οθόνη μεταφρασμένο στο περιβάλλον της σκηνής
                //επειδή η οθόνη έχει δύο διαστάσεις και η σκηνή τρεις, ως ύψος, τίθεται η απόσταση dist
                pplane.transform.position = Vector3.Lerp(pplane.transform.position, touchedPos, Time.deltaTime * 10f);
                //μεταφορά του αντικειμένου μαζί με την κίνηση του αγγίγματος αλλά ομαλά με συγκεκριμένη ταχύτητα
            }
        }
        if (touch.phase == TouchPhase.Ended)    //αν τελείωσε το άγγιγμα
        {
            pplane = null;                      //αδειάζει το ενεργό αντικείμενο
        }
    }
}
```

Εικόνα 3.7: Το Script Move Planes

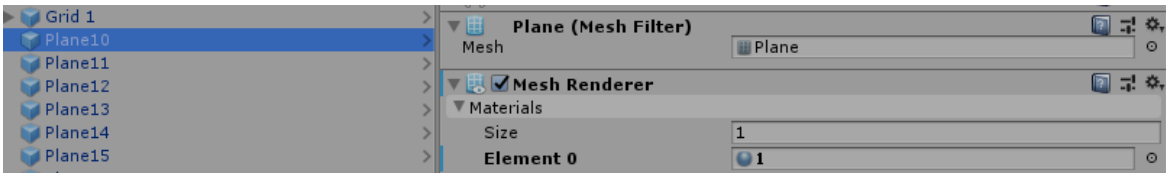
3.4 Εικόνες και υλικά για το λαβύρινθο

Η εικόνα του πλαισίου του λαβυρίνθου, όπως και των κομματιών του λαβυρίνθου σχεδιάστηκαν στο χαρτί και το προσχέδιο δόθηκε σε μία επαγγελματία για την τελική υλοποίησή τους. Υπήρχε διαρκής επικοινωνία με την συνεργάτη για την καλύτερη απεικόνιση των γρίφων, εφόσον η ανάπτυξη της εφαρμογής είχε σχεδόν τελειώσει και έλειπε να προστεθεί το αισθητικό κομμάτι. Αρχικά είχαν προστεθεί εικόνες από το διαδίκτυο, για τα πρώτα στάδια της ανάπτυξης, όμως θέλοντας όλα τα στοιχεία της εφαρμογής να είναι πρωτότυπα, δεν θα παρέμεναν έτσι.

Οι εικόνες για τον γρίφο είναι η Εικόνα 3.1 και η Εικόνα 3.2, και η πρώτη κόπηκε σε εννιά κομμάτια για να διαχωριστούν τα τετράγωνα. Για να ανατεθούν οι εικόνες στο κάθε αντικείμενο, έπρεπε πρώτα να δημιουργηθεί ένα Material (Υλικό) για το κάθε ένα. Μέσα από τον Editor, με δεξί κλικ μέσα στον φάκελο Prefabs → Maze επιλέγεται το Create → Material. Από αυτό το υλικό θα αποτελείται ουσιαστικά το κάθε κομμάτι και θα έχει την ανάλογη εμφάνιση. Επιλέγοντας το νέο υλικό, με drag-and-drop ανατέθηκε η κάθε εικόνα στο κάθε υλικό μέσω του Inspector. Επιπλέον εκεί έγιναν κάποιες ρυθμίσεις ώστε να φαίνεται όσο το δυνατόν πιο καλαίσθητο το κάθε υλικό, όπως για παράδειγμα απενεργοποιήθηκαν τα πεδία Specular Highlights και Reflections ώστε να μην γυαλίζουν τόσο οι εικόνες (Εικόνα 3.8). Στο κάθε τετράγωνο Plane που υπάρχει στη σκηνή, ανατέθηκε στον Inspector του στο συστατικό Mesh Renderer στο πεδίο Element το εκάστοτε Material (Εικόνα 3.9).



Εικόνα 3.8: Ο Inspector του Material



Εικόνα 3.9: Ο Mesh Renderer του Plane10

Σαν επιπρόσθετο στοιχείο, στο κάθε Plane ανατέθηκε η ίδια εικόνα και στο συστατικό Image, σαν εικόνα αυτή τη φορά και όχι σαν υλικό. Αυτό το συστατικό δεν είναι ορατό, αλλά χρησιμεύει για το script UI Triggers (Εικόνα 2.7), για την αποθήκευση αντικειμένων στο Inventory. Αν αποθηκεύσει ο παίκτης κάποιο τετράγωνο στο Inventory, το εικονίδιο της εκάστοτε θέσης του Inventory θα αντικατασταθεί με αυτό του εκάστοτε τετραγώνου. Αυτή η λειτουργία δεν έχει κάποια χρησιμότητα στην εφαρμογή, αλλά εφόσον δημιουργήθηκε για τον γρίφο Κύβο, προστέθηκε και σε αυτόν για μελλοντική χρήση.

3.5 Επίλογος

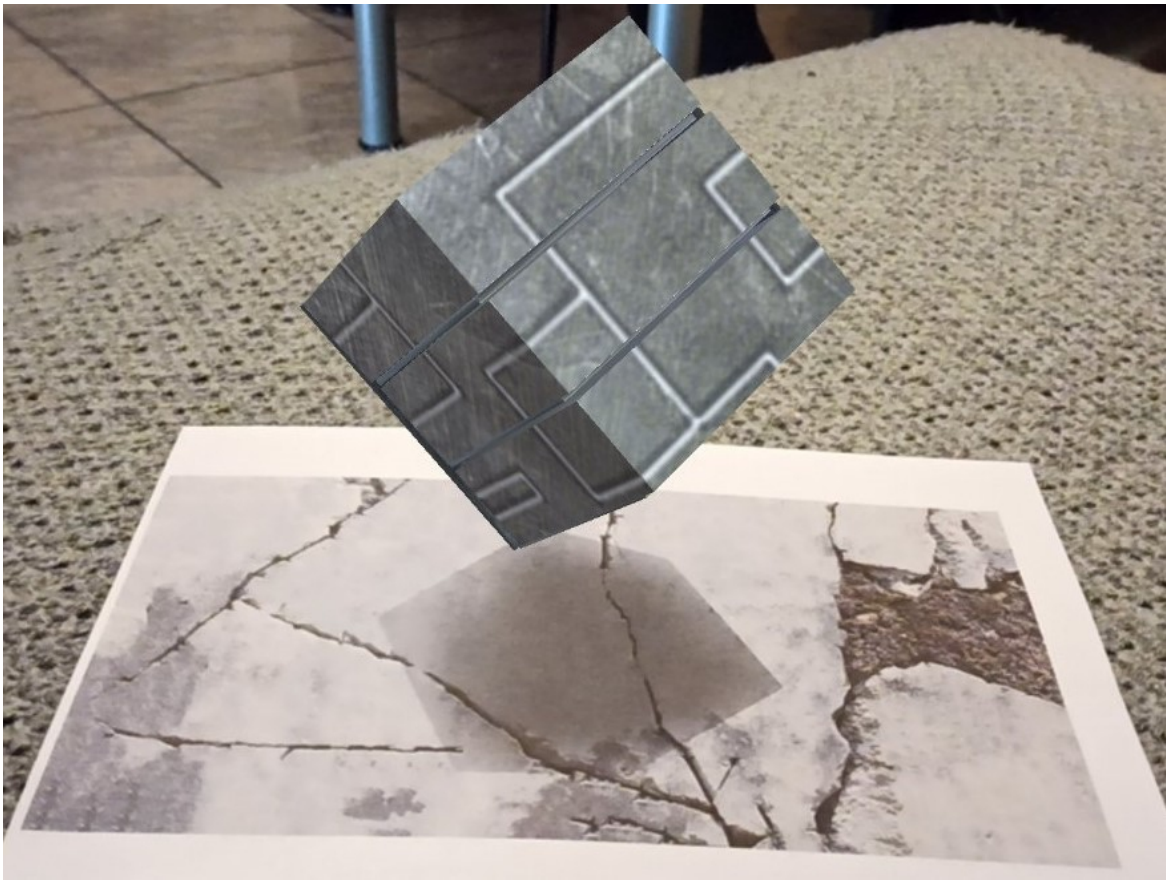
Σε αυτό το σημείο έχει ολοκληρωθεί ο πρώτος γρίφος και ένα αρχείο κειμένου που θα χρησιμοποιηθεί και στον επόμενο. Χρειάζεται να ειπωθεί πως τα αρχεία κειμένου αλλά και μεγάλο κομμάτι του σχεδιασμού του παιχνιδιού δεν συνέβη γραμμικά, δηλαδή όσο προστίθεντο νέα στοιχεία και γίνονταν δοκιμές, φάνηκαν απαραίτητες πολλές διορθώσεις, προσθήκες αλλά και αλλαγές χάριν αισθητικής και ευκολίας από την πλευρά του παίκτη.

Στο παιχνίδι οι γρίφοι δεν θα πηγαίνουν με τη σειρά, δηλαδή ο παίκτης δεν θα έρχεται αντιμέτωπος με τον γρίφο λαβύρινθο πρώτο απαραίτητα, αλλά έτσι πως θα είναι τοποθετημένες οι εικόνες μέσα στο δωμάτιο, ο παίκτης θα μπορεί να ανιχνεύσει όποια θέλει πρώτη. Βέβαια, παρακάτω, είναι σαφές πως για κάποιο γρίφο θα χρειαστεί να ανιχνευτούν παραπάνω από μία εικόνες, οπότε αυτές θα χρειαστεί να σκεφτεί ότι για να λύσει την μία εικόνα, θα πρέπει πρώτα να βρει μία άλλη.

Κεφάλαιο 4ο 2ος γρίφος – Κύβος

4.1 Εισαγωγή

Ο δεύτερος γρίφος περιέχει κι αυτός την σχεδίαση ενός λαβυρίνθου, αλλά η επίλυσή του έγκειται σε διαφορετικό σκεπτικό. Σε αυτόν τον γρίφο ο παίκτης καλείται να συμπληρώσει τα κομμάτια ενός κύβου, να τα βάλει στη σειρά ώστε να σχηματίζεται ένας ολοκληρωμένος λαβύρινθος, και να βρει το σωστό προσανατολισμό του κύβου ώστε να ταιριάζει με το εξωτερικό πλαίσιο. Για να συλλέξει τα κομμάτια του κύβου πρέπει να ψάξει στο δωμάτιο για τέσσερις εικόνες παρόμοιες μεταξύ τους, εφόσον αφορούν τον ίδιο γρίφο. Από τις τρεις θα συλλέξει από ένα κομμάτι του κύβου, βάζοντάς το στο Inventory, και στην τελευταία καλείται να τα τοποθετήσει σε ένα κουτί με τη σωστή σειρά και τον σωστό προσανατολισμό.



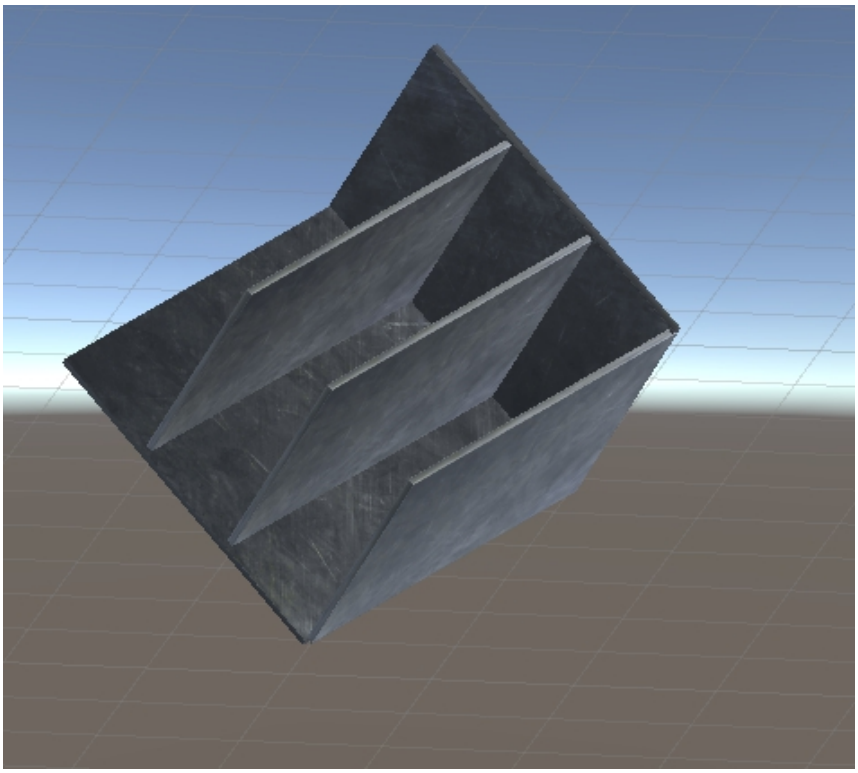
Εικόνα 4.1: Ο 2ος γρίφος λυμένος

4.2 Αντικείμενα του κύβου

Για αυτόν τον γρίφο προστέθηκαν τέσσερα Image Target στη σκηνή. Το βασικό είναι αυτό στο οποίο θα συλλεχθούν, θα τοποθετηθούν και θα περιστραφούν τα κομμάτια του κύβου. Τα άλλα τρία είναι αυτά στα οποία θα υπάρχουν τα τρία κομμάτια του κύβου μέχρι να τα βρει ο παίκτης και να τα συλλέξει. Μέσα στο βασικό Image Target δημιουργήθηκε ένα κουτί (Container) με τρία επίπεδα (ή αλλιώς ράφια) στο οποίο θα πρέπει να μπουν τα κομμάτια.

4.2.1 Container

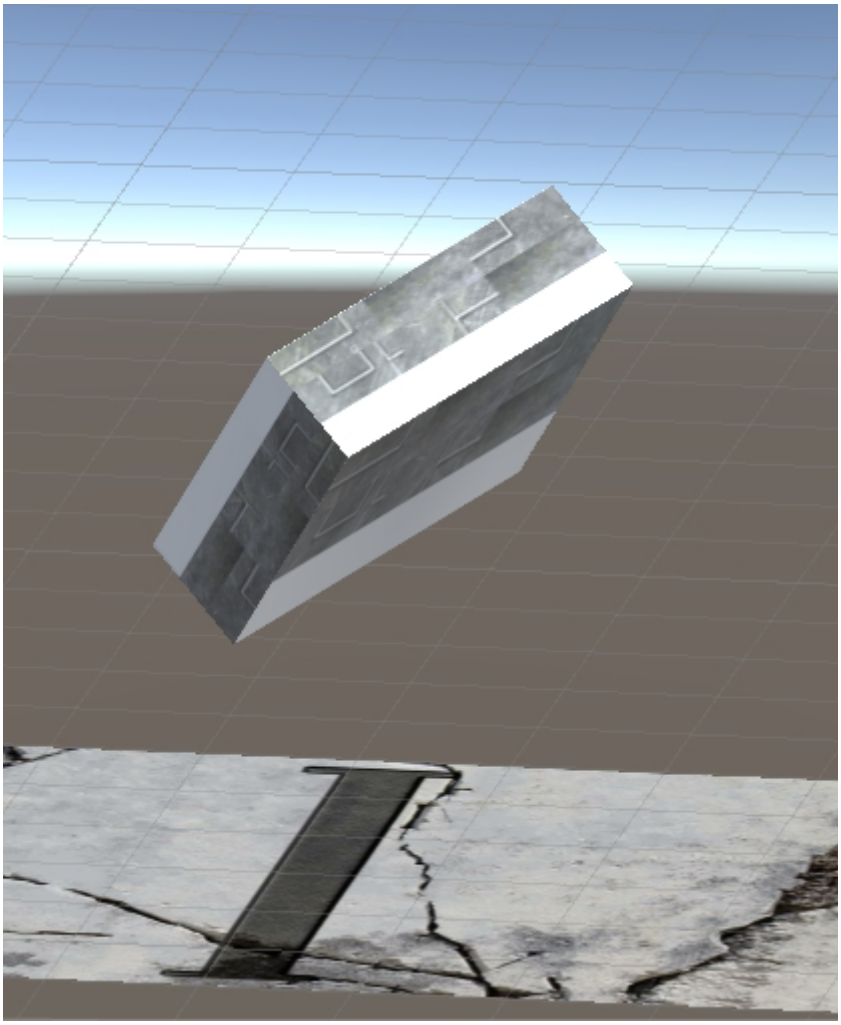
Για να υπάρχει ένα συγκεκριμένο σημείο μέσα στο Image Target που θα ενωθούν τα κομμάτια του κύβου, δημιουργήθηκε ένα κουτί με ράφια. Η ιδέα ήταν να φαίνεται ένα κουτί από το οποίο λείπουν κομμάτια και τα περιεχόμενά του φαίνονται μόνο από συγκεκριμένη σκοπιά. Προστέθηκαν στη σκηνή πέντε κύβοι (GameObject → 3d object → Cube) στους οποίους τροποποιήθηκε, μίκρυνε, η μία τους διάσταση ώστε να μεταμορφωθούν σε λεπτά επίπεδα. Τοποθετήθηκαν στο κέντρο του Image Target με μία κλίση, έτσι ώστε να φαίνεται ότι το κουτί – κύβος αιωρείται. Οι δύο κύβοι – ράφια τοποθετήθηκαν κατακόρυφα ως προς το πάτωμα ώστε να εξυπηρετούν ως τα τοιχώματα του κουτιού. Τα άλλα τρία ράφια τοποθετήθηκαν κάθετα στα άλλα, ισαπέχοντας μεταξύ τους, ώστε να προσομοιάζουν θήκες για τα κομμάτια που λείπουν (Εικόνα 4.2).



Εικόνα 4.2: Το κουτί με τα ράφια (Container)

4.2.2 Κομμάτια – Slices του κύβου

Στα άλλα τρία Image Targets προστέθηκε από ένας κύβος (GameObject → 3d object → Cube), στράφηκαν κατά την ίδια κλίση που έχει και το Container, λίγο ανυψωμένα κι αυτά για να φαίνονται σαν να αιωρούνται, και μίκρυνε η μία τους διάσταση ώστε να χωρούν στις θήκες του Container και όλα μαζί να φτιάχνουν έναν κύβο (Εικόνα 4.3). Για να υπάρχει μία κινητικότητα στο παιχνίδι, στα κομμάτια προστέθηκε και ένα script που τα κάνει να περιστρέφονται γύρω από τον εαυτό τους σε αργή ταχύτητα, μέχρι να τοποθετηθούν στο Container, όπου η περιστροφή τους θα σταματάει (Κεφάλαιο 4.3).



Εικόνα 4.3: Το πρώτο κομμάτι (Slice) του κύβου

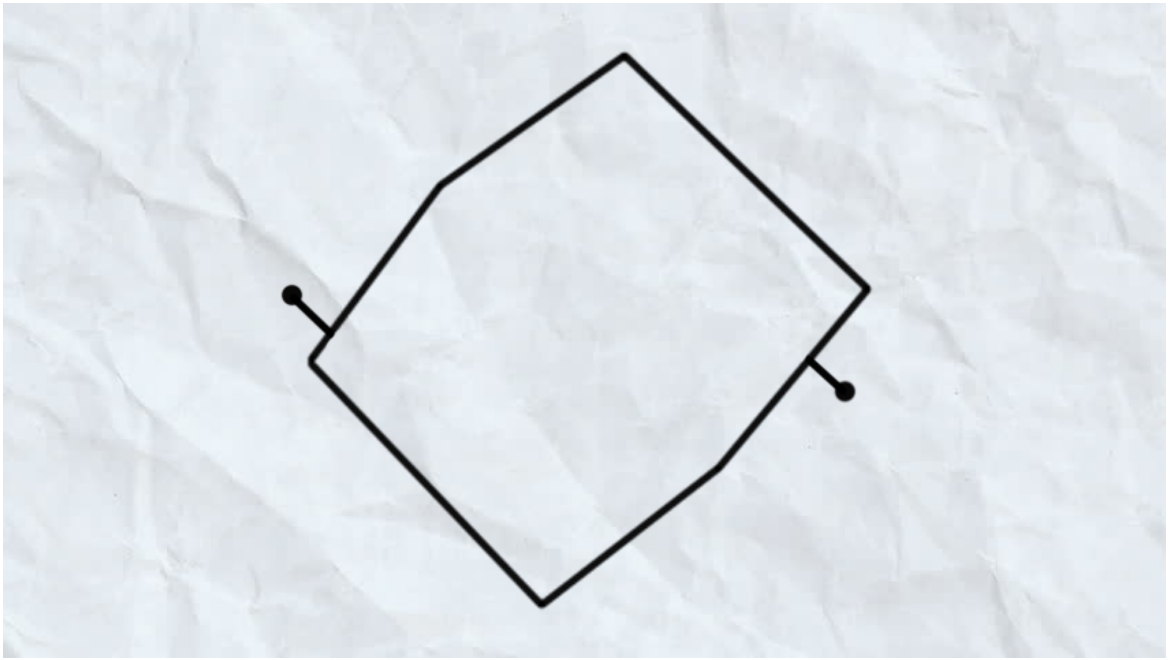
4.2.3 Shell

Στο κύριο Image Target προστέθηκε και ένα άδειο Game Object που ονομάστηκε Shell (κέλυφος) για χρησιμεύσει αφού λυθεί ο γρίφος. Όταν ο παίκτης λύσει τον γρίφο, ο κύβος θα εμφανιστεί ολοκληρωμένος και θα αρχίσει να γυρνάει γύρο από τον εαυτό του στον άξονα ψ. Επειδή όμως το Container είναι κεκλιμένο, αυτού του είδους η περιστροφή θα έδινε μια παράταιρη εικόνα. Για αυτόν το λόγο το αντικείμενο του ολοκληρωμένου κύβου θα εμφανίζεται μέσα στο Shell και θα περιστρέφεται αυτό αντί του κύβου, δίνοντας την επιθυμητή εικόνα. Σαν συστατικό, προστέθηκε στο Shell ένα πολύ απλό script, το Spin, που θα αναλυθεί παρακάτω.

Τέλος, δημιουργήθηκε ένας κύβος (GameObject → 3d object → Cube) και αποθηκεύτηκε σαν prefab, για να εμφανιστεί όταν λυθεί ο γρίφος. Σαν Material μπήκε πάνω του η Εικόνα 4.13 και προστέθηκε σαν συστατικό ξανά το script Spin απενεργοποιημένο προς το παρόν.

4.3 Αρχεία κώδικα για τον κύβο

Ο παίκτης θα συλλέγει το κάθε κομμάτι από την εικόνα του βάζοντάς το στο Inventory. Μετά θα πρέπει να τα συγκεντρώσει στο βασικό Image Target, να τα τοποθετήσει με τη σωστή σειρά στα ράφια του Container, και να τα περιστρέψει καταλλήλως ώστε να βρει τη σωστή διάταξη και κατ' επέκταση τον σωστό λαβύρινθο. Κάθε πλευρά του κύβου δημιουργεί έναν λαβύρινθο, όμως μόνο ένας ταιριάζει με την εικόνα αναφοράς στον πίσω τοίχο του δωματίου (Εικόνα 4.4).



Εικόνα 4.4: Εικόνα αναφοράς για το πλαίσιο του σωστού κύβου

Εννιά αρχεία κειμένου χρειάστηκαν για αυτόν τον γρίφο, όμως δύο από αυτά είναι αναπαραγωγές ενός άλλου, και τρία από αυτά έχουν γραφτεί ήδη για τον προηγούμενο γρίφο και το μενού. Απλώς χρειάστηκε να γίνουν κάποιες αλλαγές για να συμπεριλάβουν και τις ανάγκες του κύβου. Άρα σε αυτό το κεφάλαιο θα παρουσιαστούν τέσσερα αρχεία κώδικα και θα γίνουν αναφορές στη χρήση των προαναφερθέντων, ήδη αναλυμένων scripts.

Αρχικά και στα τέσσερα νέα Image Targets προστέθηκε σαν συστατικό το script Move Plane (Εικόνα 3.7) για να μπορεί ο παίκτης να μετακινεί τα slices, να τα αποθηκεύσει στο Inventory και να τα τοποθετήσει στο Container του βασικού Image Target. Φυσικά καλείται και το UI Triggers (Εικόνες 2.7 & 2.8) για να ενεργοποιούνται οι λειτουργίες του Inventory. Το script UI Manager (Κεφάλαιο 2.3.3) ενεργοποιείται όταν θα πατάει ο παίκτης το κουμπί rotate κάτω δεξιά στην οθόνη, για να περιστρέψει τα slices ώστε να βρει το σωστό λαβύρινθο. Όταν τα περιστρέφει, θα ενεργοποιείται το αρχείο ήχου και θα ακούγεται ένας ήχος σαν να γυρνάει κάτι σκουριασμένο (μέθοδος *void rotate()*, Κεφάλαια 2.3.2. & 2.3.3).

4.3.1 Spin

Αυτό το αρχείο κώδικα χρησιμοποιήθηκε σαν συστατικό στο αντικείμενο Shell του βασικού Image Target αλλά και στα Slices για αισθητικούς λόγους. Με μια μικρή παραλλαγή χρησιμοποιήθηκε και στον επόμενο γρίφο. Το script αποτελείται από μία εντολή και το μόνο που κάνει είναι να περιστρέφει το αντικείμενο γύρω από τον εαυτό του στον ψ άξονα με συγκεκριμένη ταχύτητα.

Η μέθοδος Update του script Spin:

```
void Update()
  {          transform.Rotate(Vector3.down * Time.deltaTime * 20f);          }
```

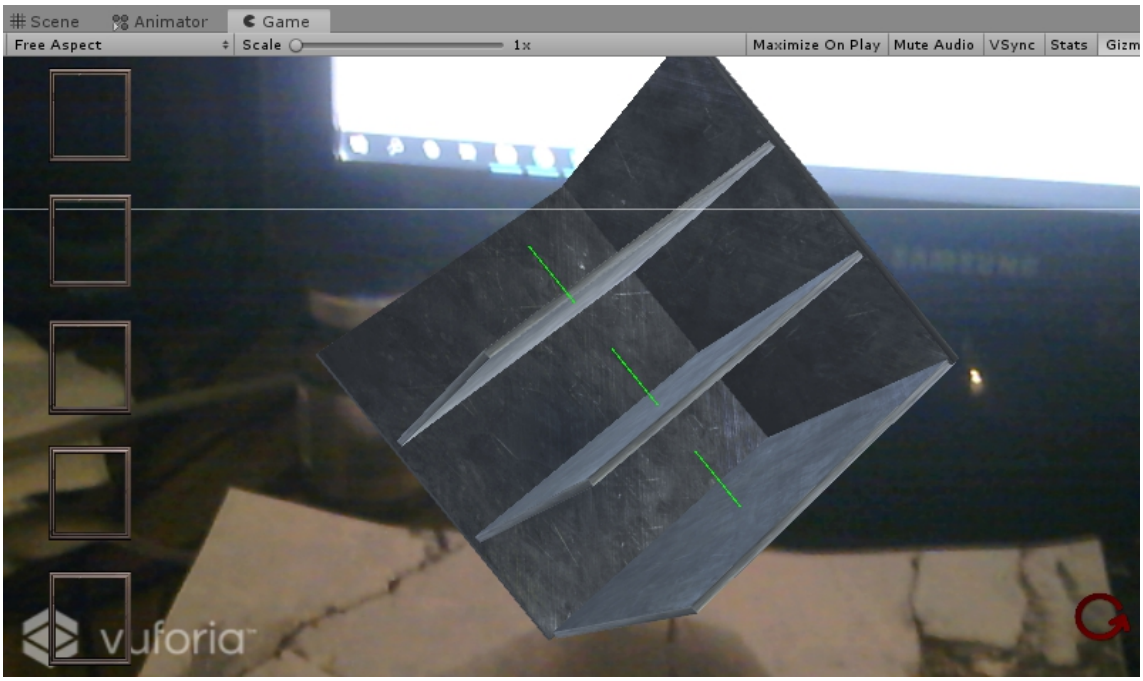
4.3.2 Shelf 1,2,3

Στα τρία οριζόντια ράφια του Container προστέθηκε το script Shelf. Θα μπορούσε κάλλιστα να είναι ένα script και να διαχωρίζει τις διαφορές των ραφιών (κυρίως την τοποθεσία τους) με μία μέθοδο switch case όπως το script Panel (Κεφάλαιο 3.3.2). Αλλά με αυτόν τον τρόπο ήταν πιο εύκολο το troubleshooting. Σε μία εφαρμογή που οι ταχύτητες προσπέλασης των αρχείων κειμένων θα ήταν κρίσιμης σημασίας, θα χρειαζόταν να μελετηθεί η συγκρινόμενη απόδοση των δύο script.

Το script Shelf εκπέμπει μία ακτίνα που ξεκινάει από το κέντρο του ραφιού προς τα πάνω για 0.05 μονάδες χώρου (Εικόνα 4.5). Αν χτυπήσει κάτι (κάποιον Collider, οτιδήποτε άλλο δεν ενεργοποιεί το raycast [16]), ελέγχει αν αυτό που χτυπήθηκε είναι το σωστό αντικείμενο ελέγχοντας το όνομά

του, και αν είναι το σωστό, ελέγχει την σχετική περιστροφή του. Αν και οι δύο αυτές συνθήκες είναι αληθείς, το script μετατρέπει την αντίστοιχη Boolean μεταβλητή του script Shelves σε αληθή. Αν το σωστό κομμάτι φύγει από την ακτίνα, επαναφέρει τη μεταβλητή σε ψευδή κατάσταση (Εικόνα 4.6).

Τα scripts Shelf2 & Shelf3 περιέχουν ακριβώς τον ίδιο κώδικα, με διαφορά το όνομα του Slice (Slice2 & Slice3 αντίστοιχα) και τις συντεταγμένες σχετικής τοποθεσίας στη μετακίνηση του αντικειμένου.



Εικόνα 4.5: Οι ακτίνες που εκπέμπουν τα ράφια του Container

```
using System.Collections;
using System.Collections.Generic;
using UnityEngine;

public class Shelf1 : MonoBehaviour
{
    public GameObject pan, hit;
    private Shelves shscr;
    public float eu;

    void Start()
    {
        shscr = transform.parent.GetComponent<Shelves>();
        //παίρνει το script Shelves από τον γονέα - Container
    }

    void Update()
    {
        RaycastHit objectHit; //το αντικείμενο που θα χτυπηθεί
        Vector3 fwd = pan.transform.TransformDirection(0, 0.05f, 0);
        //δημιουργία της ακτίνας σε μήκος 0.05 μονάδες
        Debug.DrawRay(pan.transform.position, fwd, Color.green);
        //η απεικόνιση της ακτίνας για troubleshooting
        if (Physics.Raycast(pan.transform.position, fwd, out objectHit, 0.05f))
            //αν χτυπήσει κάτι η ακτίνα
        {
            hit  = objectHit.collider.gameObject;
            //το αντικείμενο που χτυπήθηκε ως GameObject
            hit.transform.localPosition = new Vector3(0.033f, 0.428f, -0.024f);
            //μετακίνηση του αντικειμένου ώστε να εφαρμόζει στο συγκεκριμένο ράφι
            if (hit.name == "Slice1"){      //αν το αντικείμενο είναι το Slice1
                eu = hit.transform.localEulerAngles.y;
                //η σχετική περιστροφή του αντικειμένου στον άξονα ψ
                if (eu == 180f) //αν η περιστροφή είναι 180 μοίρες σε μονάδες Euler
                {
                    shscr.fill1 = true;
                    //η αντίστοιχη Boolean μεταβλητή του script Shelves γίνεται true
                }
                else { shscr.fill1 = false;}
                //αλλιώς η αντίστοιχη Boolean μεταβλητή του script Shelves γίνεται false
            }
            else { shscr.fill1 = false;}
            fwd = pan.transform.TransformDirection(0, 0, 0);
            //η ακτίνα απενεργοποιείται ώστε αν αλλάξουν οι σωστές συνθήκες να μην
            //παραμένει true το shscr.fill1
        }
        else{
            shscr.fill1 = false;
        }
    }
}
```

Εικόνα 4.6: Το script Shelf1

4.3.3 Shelves

Παρόμοια με τον πρώτο γρίφο, γράφτηκε ένα script για να ελέγχει πότε τηρούνται όλες οι προδιαγραφές για να έχει λυθεί ο κύβος. Όπως φάνηκε στο script Shelf, ενεργοποιούνται ή απενεργοποιούνται κάποιες μεταβλητές του script Shelves. Όταν όλες από αυτές είναι αληθείς, ο γρίφος θα έχει λυθεί, θα απενεργοποιηθούν και θα εξαφανιστούν από τη σκηνή τα υπάρχοντα αντικείμενα, και στη θέση τους (μέσα στο Shell) θα εμφανιστεί ολοκληρωμένος ο κύβος να περιστρέφεται (Εικόνα 4.7). Τα αντικείμενα cu (cube) και sh (shell) δηλώνονται ως public μεταβλητές έτσι ώστε να είναι ορατά και μεταβλητά στον Inspector του αντικειμένου στο οποίο προσκολλάται το script. Από εκεί, ανατίθενται μέσα στον Editor, και μπαίνει στο cu ένα prefab με ολοκληρωμένο το μοντέλο του κύβου, και στο sh το αντικείμενο Shell που υπάρχει μέσα στη σκηνή στο Image Target.

```
1  using System.Collections;
2  using System.Collections.Generic;
3  using UnityEngine;
4
5  public class Shelves : MonoBehaviour
6  {
7      public bool fill1, fill2, fill3;
8      //οι μεταβλητές που τίθενται από τα scripts Shelf
9      public GameObject s1, s2, s3, cont, cube, cu, sh;
10     //τα slices, το container,ο ολοκληρωμένος κύβος,
11     //το μοντέλο του νέου κύβου και το shell
12     // που τίθενται στον Inspector
13
14     void Update() //τσεκάρει σε κάθε frame
15     {
16         if (fill1 && fill2 && fill3){
17         //αν είναι αληθείς όλες οι μεταβλητές των ραφιών
18             s1.SetActive(false); //απενεργοποιούνται
19             s2.SetActive(false); // τα τρία slices
20             s3.SetActive(false); // και εξαφανίζονται
21             cont.SetActive(false); //και το container
22             cube = Instantiate (cu, sh.transform);
23             //δημιουργείται ο ολοκληρωμένος κύβος ως παιδί του shell
24             sh.GetComponent<Spin>().enabled = true;
25             //ενεργοποιείται το script Spin του κύβου για να γυρίζει
26         }
27     }
28 }
```

Εικόνα 4.7: Το script Shelves

4.3.4 Cube Sides

Το τελευταίο script που γράφτηκε για αυτόν τον γρίφο αφορά την εμφάνισή των μοντέλων του. Όπως περιγράφηκε στο Κεφάλαιο 3.4, για την εμφάνιση των 3D αντικειμένων δημιουργήθηκαν διάφορα αντικείμενα Materials (Υλικά) και ενσωματώθηκαν στα αντικείμενα ως Elements στον Mesh Renderer τους. Για αυτόν τον γρίφο όμως δεν λειτούργησε κάτι τέτοιο. Αντ' αυτού, όταν προστέθηκε ένα Material στα slices ή στον ολοκληρωμένο κύβο, η εικόνα προβλήθηκε σε κάθε πλευρά των κυβοειδών. Αυτό το γεγονός αποτέλεσε πρόβλημα στην ανάπτυξη του γρίφου, μιας κι αυτός βασίζεται στις διαφορές των πλευρών των κυβοειδών. Έτσι έγινε μια έρευνα για τους τρόπους που μπορούν να κατανεμηθούν διαφορετικές εικόνες στις πλευρές ενός 3D αντικειμένου στο Unity. Μία σκέψη ήταν οι κύβοι να αποτελούνταν από έξι Planes σε σχηματισμό κύβου, και πάνω στο κάθε Plane να υπάρχει και η εικόνα μίας πλευράς. Αυτό όμως θα ήταν μια πολύ πρόχειρη λύση και θα παρήγαγε μπερδεμένο προγραμματισμό στον κώδικα και στον Editor, χωρίς τη δυνατότητα αυτοματισμού και αναπαραγωγής. Έτσι βρέθηκε μια λύση που επηρέασε τόσο τον κώδικα όσο και τον σχεδιασμό των εικόνων για τους κύβους. Η εικόνα για κάθε κύβο σχεδιάστηκε έτσι ώστε να περιέχει και τις έξι πλευρές του κύβου αλλά σε συγκεκριμένα σημεία της εικόνας, και αντίστοιχα το script Cube Sides τις αναθέτει στις κατάλληλες πλευρές. Οι εικόνες θα παρατεθούν στο επόμενο κεφάλαιο.

Κομμάτι του script Cube Sides:

```
void Start()
  {
    Mesh mesh = GetComponent<MeshFilter>().mesh;
   //παίρνει την εμφάνιση του κύβου
    Vector2[] UVs = new Vector2[mesh.vertices.Length];
   //δημιουργεί έναν πίνακα με τα μήκη των κορυφών των πλευρών του κύβου
    // Front – η μπροστινή πλευρά
    UVs[0] = new Vector2(0.0f, 0.0f);       //το πάνω αριστερά σημείο
    UVs[1] = new Vector2(0.333f, 0.0f);     //το πάνω δεξιά σημείο
    UVs[2] = new Vector2(0.0f, 0.333f);     //το κάτω αριστερά σημείο
    UVs[3] = new Vector2(0.333f, 0.333f); //το κάτω δεξιά σημείο
    // Top – η πάνω πλευρά
    UVs[4] = new Vector2(0.334f, 0.333f);
    UVs[5] = new Vector2(0.666f, 0.333f);
    UVs[8] = new Vector2(0.334f, 0.0f);
    UVs[9] = new Vector2(0.666f, 0.0f);
    .
    .
    .
```

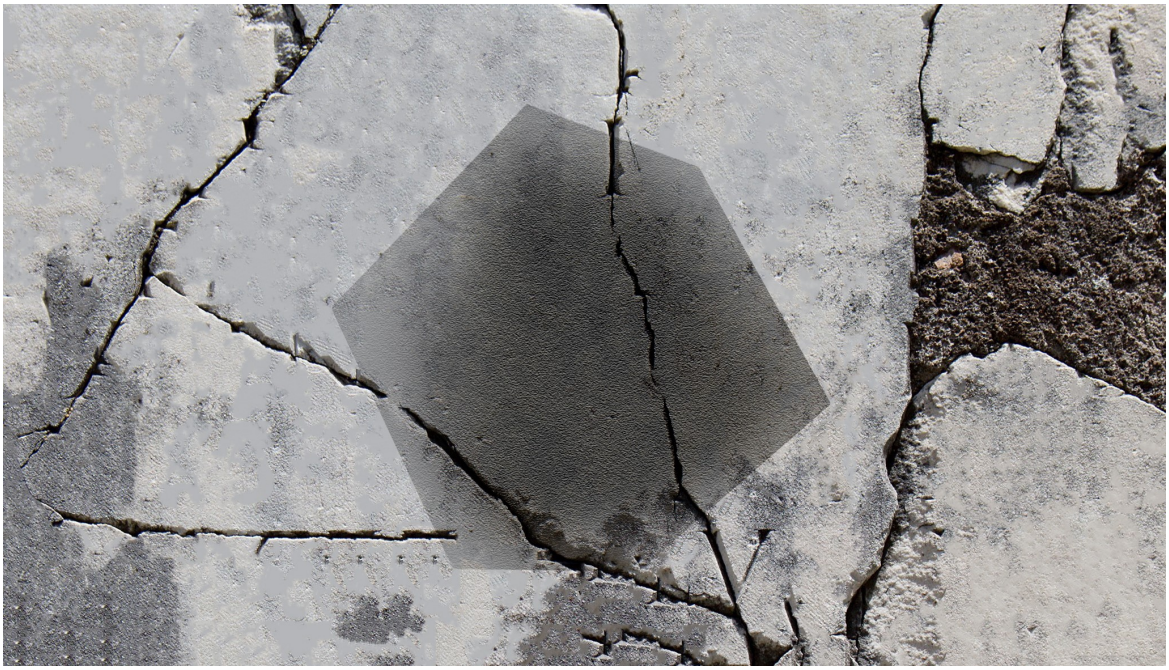
Και το script συνεχίζει με τις υπόλοιπες τέσσερις πλευρές του κύβου και τελειώνει με την εντολή mesh.uv = UVs; όπου ανατίθεται η εικόνα στα συγκεκριμένα σημεία του κύβου [17].

Το script προστέθηκε ως συστατικό στα τρία Slices και στον ολοκληρωμένο κύβο, ώστε η κάθε πλευρά τους να δείχνει ένα κομμάτι του λαβυρίνθου. Το script ξεκινά να τρέχει με το που φορτωθούν στο παιχνίδι τα αντικείμενα αυτά και δεν παρατηρήθηκε να μειώνει την ταχύτητα λειτουργίας του παιχνιδιού.

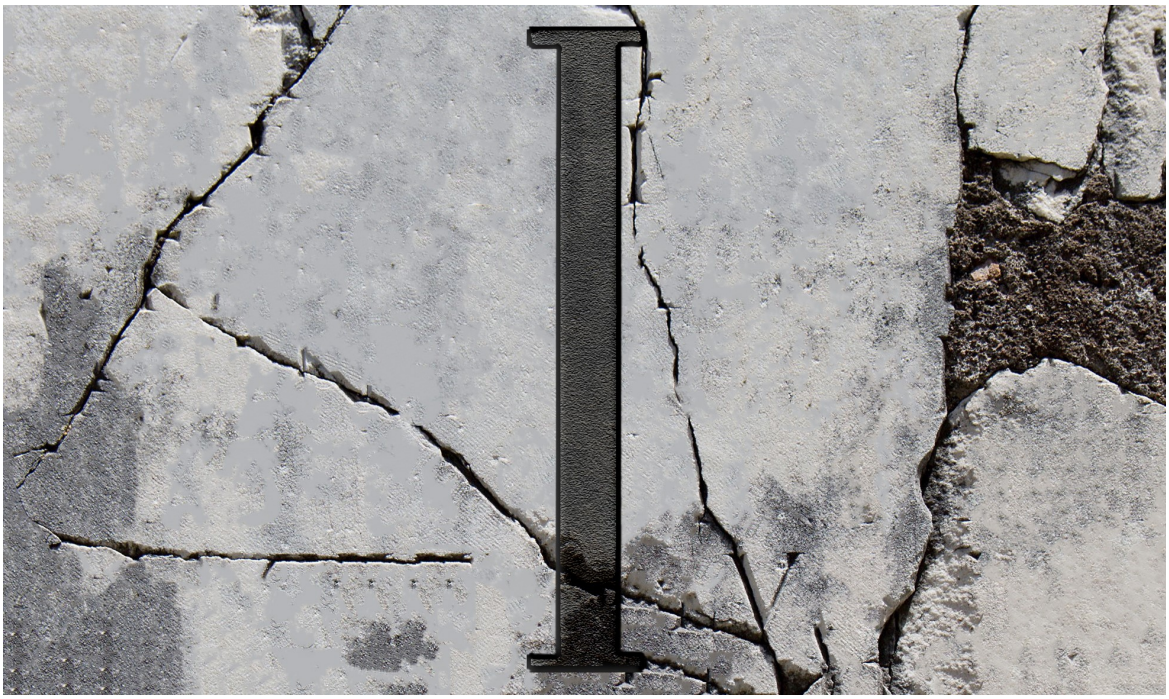
4.4 Εικόνες και υλικά για τον κύβο

Ήταν επιθυμητό να συνδέονται με κάποιο τρόπο οι εικόνες αυτού του γρίφου μεταξύ τους για να δίνουν την αίσθηση ότι αποτελούν έναν ενιαίο γρίφο και δεν είναι τέσσερις ξεχωριστές εικόνες. Γι αυτόν το λόγο επιλέχθηκε μια ταιριαστή με το ύφος του παιχνιδιού εικόνα για background και πάνω της κάποιο ξεχωριστό σχέδιο για το κάθε Image Target. Οι τρεις εικόνες των Slices απεικονίζουν τους αριθμούς 1, 2 και 3 με λατινικούς χαρακτήρες για να δοθεί στον παίκτη ένα στοιχείο ως προς το με ποια σειρά μπαίνουν τα Slices στο Container. Στη βασική εικόνα σχεδιάστηκε το περίγραμμα του κύβου με γέμισμα που να φαίνεται σαν τη σκιά του κύβου.

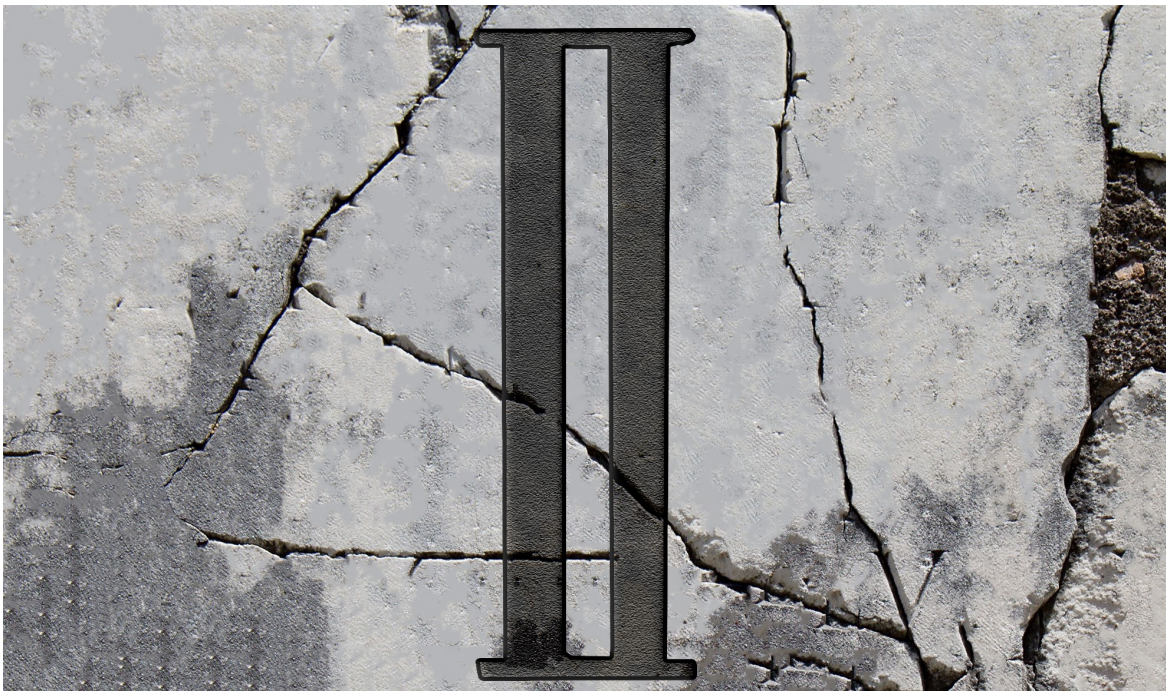
Οι εικόνες που χρησιμοποιήθηκαν για τα Image Target αυτού του γρίφου ήταν οι εξής:



Εικόνα 4.8: Η εικόνα του βασικού Image Target



Εικόνα 4.9: Η εικόνα του Image Target για το πρώτο Slice

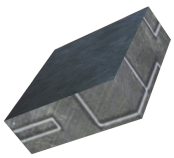


Εικόνα 4.10: Η εικόνα του Image Target για το δεύτερο Slice

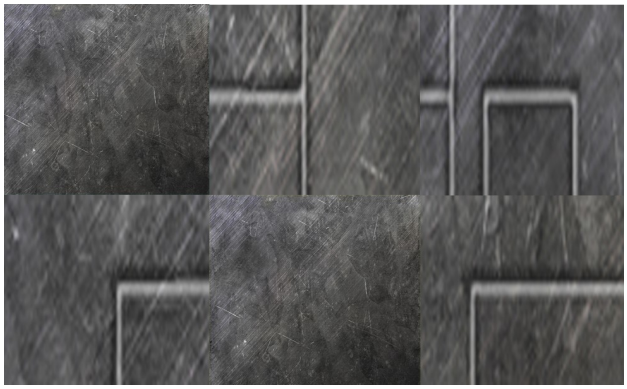
Εικόνα 4.11: Η εικόνα του Image Target για το τρίτο Slice

Για το Container χρησιμοποιήθηκε ένα απλό texture από το διαδίκτυο που να δίνει την αίσθηση της σιδερένιας επιφάνειας, με μια πιο σκούρη απόχρωση στα planes των τοιχωμάτων σε σχέση με τα ράφια για καλύτερο φωτισμό της σκηνής. Τα materials των slices και του κύβου σχεδιάστηκαν πάνω στο ίδιο texture με ασημένιες γραμμές που σχηματίζουν τους λαβυρίνθους.

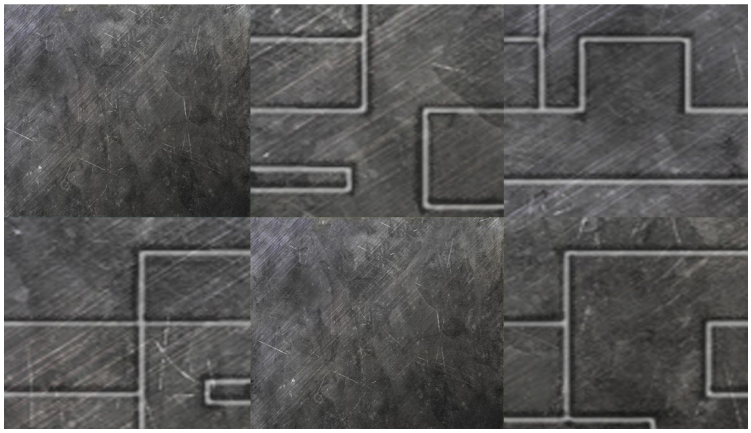
Στο κάθε Slice, στο συστατικό Image, προστέθηκε μία εικόνα του ίδιου του slice από screenshot του παιχνιδιού, ώστε να φαίνεται σαν εικονίδιο όταν ο παίκτης θα το προσθέσει στο Inventory (Εικόνα 4.12). Οι εικόνες 4.13 και 4.14 έχουν εισαχθεί ακριβώς έτσι, παραμορφωμένες, και με το script Cube Sides εφαρμόζουν σωστά πάνω στον κύβο και τα Slices.



Εικόνα 4.12: Τα εικονίδια των Slices για το Inventory



Εικόνα 4.13: Η εικόνα για το Material του Slice1

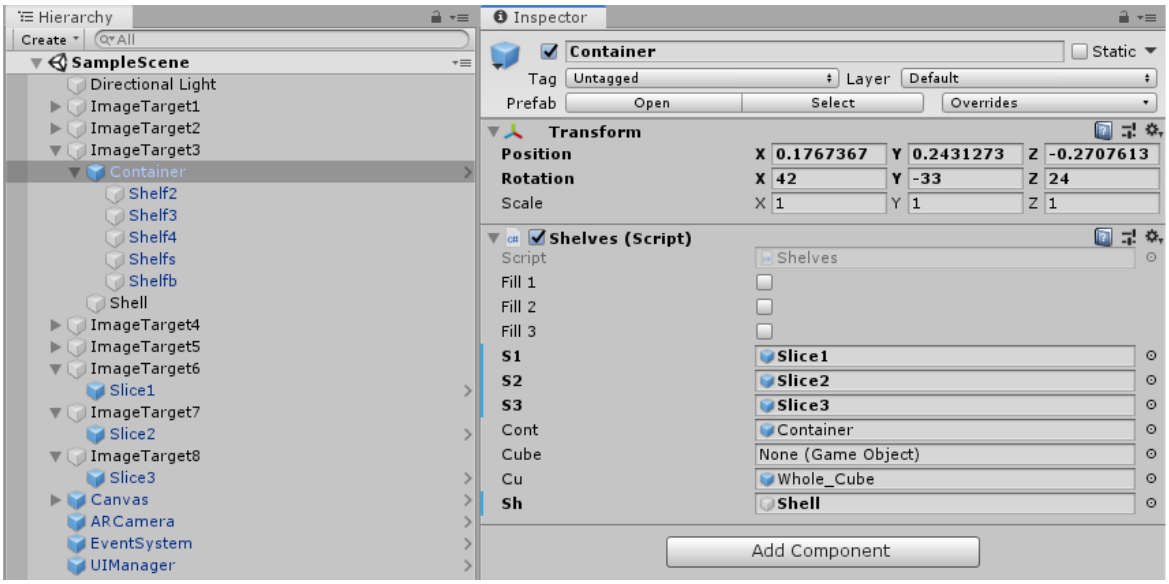


Εικόνα 4.14: Η εικόνα για το Material του ολοκληρωμένου κύβου

4.5 Επίλογος

Σε αυτό το σημείο έχει ολοκληρωθεί και ο δεύτερος γρίφος του παιχνιδιού. Σε αυτόν τον γρίφο χρειάστηκε να γίνουν οι περισσότερες δοκιμές κατά την ανάπτυξή του. Δοκιμές χρειάστηκαν τόσο για το προγραμματιστικό κομμάτι, όσο και για το αισθητικό. Από το πως θα μπαίνουν τα slices στο Inventory μέχρι το πως θα ανιχνεύεται το αν μπήκαν τα slices πάνω στο σωστό ράφι. Επίσης, αρχικά ο γρίφος είχε σχεδιαστεί μόνο με ένα Image Target στο οποίο υπήρχαν όλα τα συστατικά του, και αργότερα ενσωματώθηκε η ιδέα ο παίκτης να πρέπει να συλλέξει τα κομμάτια και να τα συγκεντρώσει στο Container. Το Inventory ήταν το πρώτο πράγμα που αναπτύχθηκε για την εφαρμογή, επειδή σαν γνώση και εμπειρία κρίθηκε πολύτιμη για την ανάπτυξη οποιουδήποτε παιχνιδιού, οπότε μπήκε σε χρήση σε αυτόν τον γρίφο.

Ο γρίφος αποτελείται από τέσσερα Image Targets όπου μέσα στα τρία υπάρχει από ένα Slice του κύβου, και στο κύριο Image Target υπάρχει το Shell στο οποίο θα εμφανίζεται ο ολοκληρωμένος κύβος, και το Container με τα πέντε ράφια, τρία εκ των οποίων είναι λειτουργικά ως θήκες για τα Slices (Εικόνα 4.15).



Εικόνα 4.15: Το Hierarchy του κύβου και ο Inspector του Container

Κεφάλαιο 5ο 3ος γρίφος – Δίσκος

5.1 Εισαγωγή

Ο τρίτος γρίφος αφορά ένα σχέδιο σε ένα δίσκο, ο οποίος χωρίζεται σε τρία δαχτυλίδια με την εικόνα πάνω του μπερδεμένη. Ο παίκτης καλείται να φέρει τα δαχτυλίδια στη σωστή θέση ώστε να ολοκληρωθεί το σχέδιο. Στην Εικόνα 5.1 φαίνεται ο δίσκος όπως ξεκινάει στο παιχνίδι. Το τέταρτο εξωτερικό δαχτυλίδι δεν είναι μέρος των γραφικών αντικειμένων μέσα στη σκηνή, αλλά η εικόνα του Image Target στο οποίο επαυξάνονται τα τρία δαχτυλίδια. Στην εικόνα διαφέρουν τα δαχτυλίδια με το περίγραμμα στο χρώμα διότι δεν φωτίζονται όταν δεν τρέχει το παιχνίδι.

Εικόνα 5.1: Ο δίσκος – γρίφος προς επίλυση

5.2 Εικόνες για τον δίσκο

Για αυτόν τον γρίφο σχεδιάστηκε ένας κύκλος με ένα περίτεχνο γέμισμα (Εικόνα 5.2), ο οποίος κόπηκε σε τέσσερα κομμάτια, τρία δαχτυλίδια (Εικόνες 5.3, 5.4, 5.5) και έναν μικρότερο κύκλο (Εικόνα 5.6), το κέντρο του μεγαλύτερου κύκλου, ο οποίος χάριν ευκολίας αναφέρεται επίσης ως δαχτυλίδι. Το εξωτερικό δαχτυλίδι (Εικόνα 5.3) δεν ενσωματώθηκε σε κάποιο αντικείμενο μέσα στον Editor αλλά χρησιμοποιήθηκε ως η εικόνα για το Image Target του γρίφου. Αυτό επιλέχθηκε για να δώσει μία κατευθυντήρια γραμμή στον παίκτη, ως προς το με ποιο σκεπτικό πρέπει να συνδέονται τα δαχτυλίδια μεταξύ τους. Το texture σχεδιάστηκε έτσι ώστε να δίνει την αίσθηση ξύλινης υφής με σκαλισμένα σχέδια.

Εικόνα 5.3: Εξωτερικό δαχτυλίδι

Εικόνα 5.4: Το τρίτο δαχτυλίδι

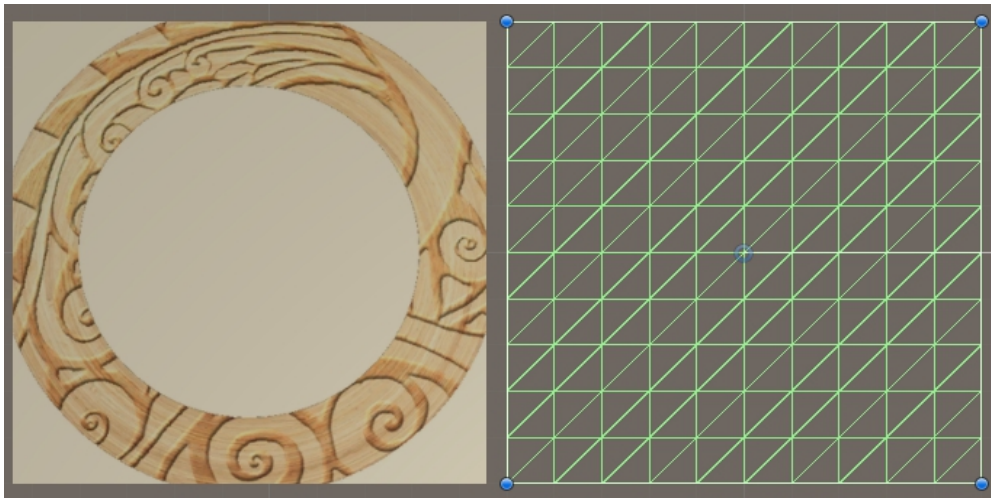
Εικόνα 5.5: Το δεύτερο δαχτυλίδι

Εικόνα 5.6: Το πρώτο δαχτυλίδι

Εικόνα 5.2: Ολόκληρος ο κύκλος

5.3 Αντικείμενα του δίσκου

Για αυτόν το γρίφο προστέθηκε ακόμη ένα Image Target και του ανατέθηκε η εικόνα circle frame (Εικόνα 5.3). Σε αυτό προστέθηκαν τρία Planes. Όταν τελείωσε η ανάπτυξη του γρίφου και δοκιμάστηκε η λειτουργία του, ήρθε στην επιφάνεια το γεγονός ότι τα Planes δεν ήταν τα κατάλληλα αντικείμενα για τη λειτουργία αυτού του γρίφου οπότε έπρεπε να αντικατασταθούν. Ο παίκτης θα πρέπει να μπορεί να αγγίξει ένα από τα δαχτυλίδια και, σέρνοντας το δάχτυλό του στην οθόνη, να γυρίζει το δαχτυλίδι προς τα πάνω ή κάτω ώστε να αλλάξει τον προσανατολισμό του, και να φέρει τον δίσκο στη σωστή του μορφή. Τα αντικείμενα Planes είναι τετράγωνα 3D αντικείμενα, των οποίων ο Collider έχει τετράγωνο σχήμα και γέμισμα, οπότε βάζοντας το ένα δαχτυλίδι μέσα στο άλλο, ακόμα και αν ο χρήστης άγγιζε το δεύτερο δαχτυλίδι, μπορεί να ενεργοποιούσε το τρίτο δαχτυλίδι, μιας και ο Collider του δεν είχε τρύπα, εκεί που είχε το σχέδιο του δαχτυλιδιού (Εικόνα 5.7).

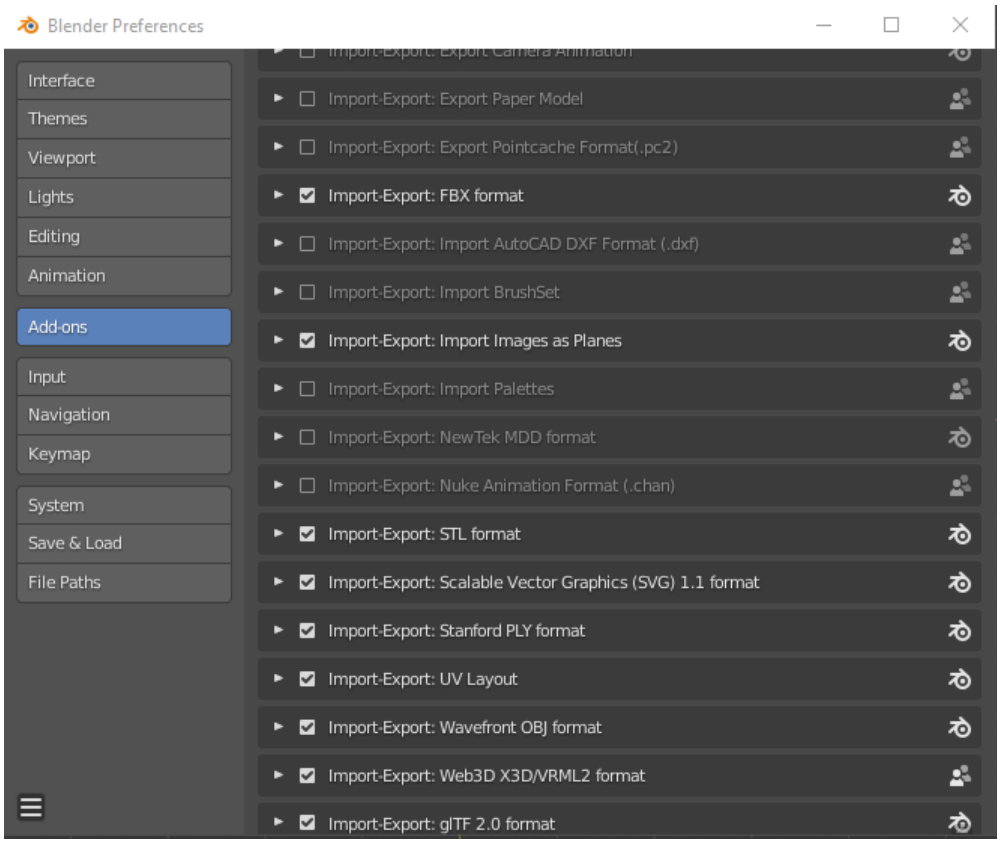


Εικόνα 5.7: Το plane δαχτυλίδι και ο Mesh Collider του

Το Unity δεν είναι μία πλατφόρμα σχεδίασης 3D γραφικών οπότε περιέχει μόνο κάποια βασικά 3D μοντέλα που μπορεί να προσθέσει και να επεξεργαστεί κανείς στο project του. Για παράδειγμα σφαίρα, κύβο, κύλινδρο, κάψουλα και επίπεδο (Plane). Για αυτόν το λόγο χρειάστηκε η χρήση ενός τρίτου προγράμματος, για τη σχεδίαση των 3D δαχτυλιδιών ώστε να έχουν τον σωστό, δικό τους collider. Επιλέχθηκε το πρόγραμμα Blender [6] λόγω μιας μικρής οικειότητας και φυσικά επειδή είναι ένα ανοιχτού κώδικα εργαλείο για ερασιτέχνες και επαγγελματίες δημιουργούς.

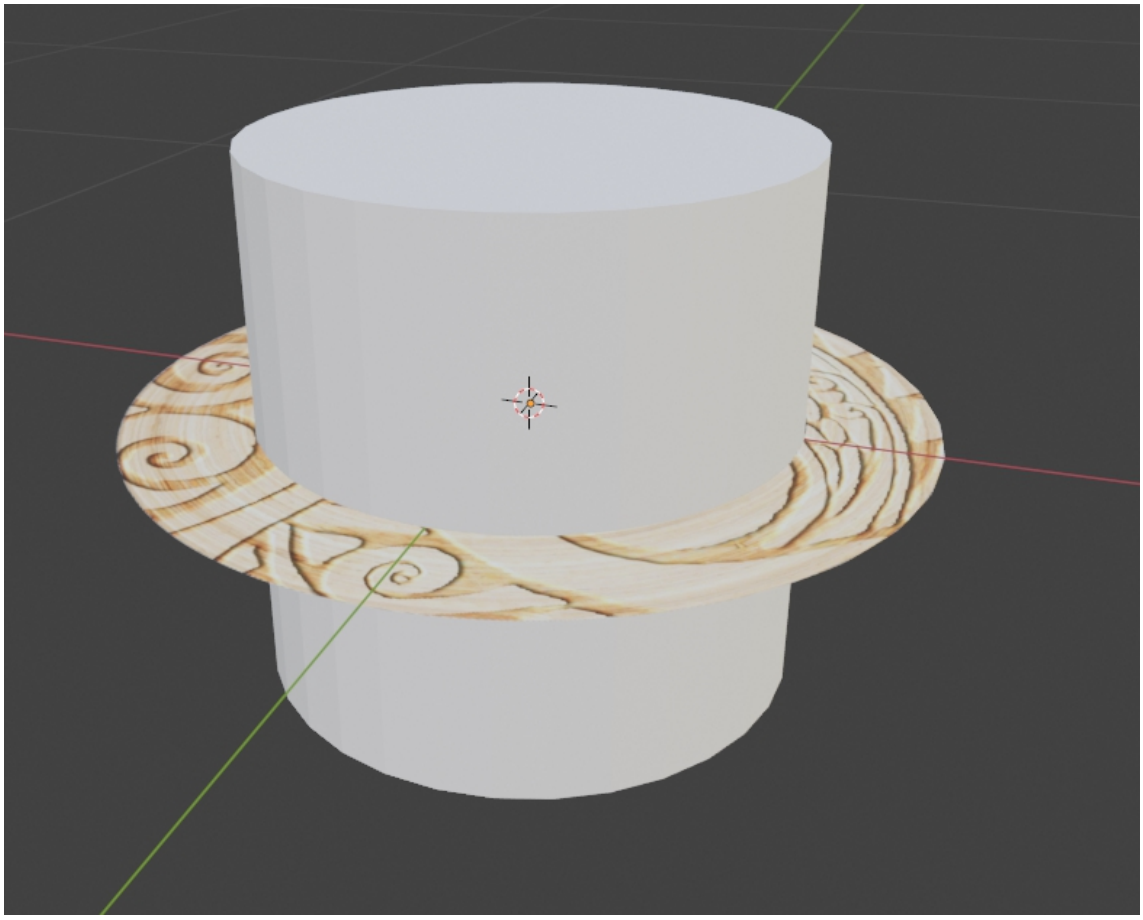
Αυτό το κομμάτι ανάπτυξης των παιχνιδιών αναλαμβάνεται συνήθως από επαγγελματίες 3D γραφίστες. Σε αυτόν και τον επόμενο γρίφο ήταν αναγκαία η ανάπτυξη νέων 3D μοντέλων, και δεν θα ήταν εφικτή χωρίς κάποιο εσπευσμένο tutorial για τις απλές ενέργειες που χρειάστηκαν [18]. Αυτό που χρειάστηκε ουσιαστικά ήταν να περαστούν οι ήδη σχεδιασμένες εικόνες των δαχτυλιδιών στο πρόγραμμα, να αποκοπούν τα κενά σημεία των εικόνων ώστε να μείνουν μόνο τα δαχτυλίδια που να έχουν collider, και να περαστούν τα νέα μοντέλα στη σκηνή του παιχνιδιού.

Ανοίγοντας το Blender, χρειάστηκε να ενεργοποιηθεί ένα add-on για να γίνει δυνατή η εισαγωγή εικόνων αυτόματα ως Planes. Αυτό έγινε από το μενού Edit → Preferences και στην ανοιχτή καρτέλα Add-ons ενεργοποιήθηκε η επιλογή Import-Export: Import images as Planes (Εικόνα 5.8)



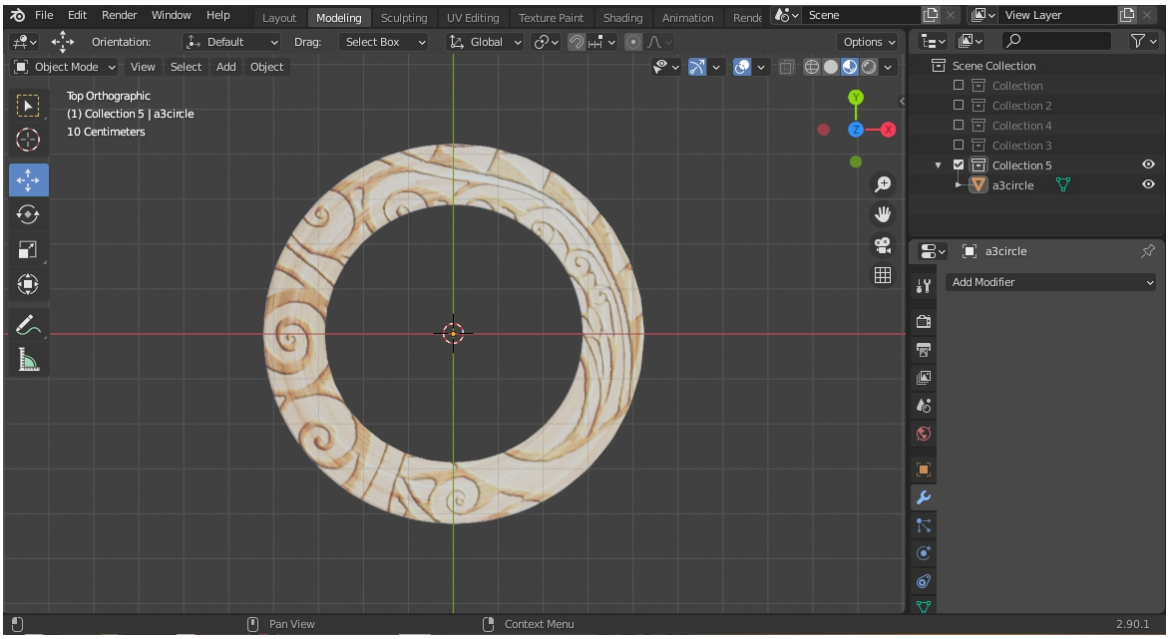
Εικόνα 5.8: Η καρτέλα Add-ons του Preferences στο Blender

Αφού ενεργοποιήθηκε αυτή η λειτουργία, από το μενού File → Import → Images as Planes εισήχθη η εικόνα του τρίτου δαχτυλιδιού. Το Blender δουλεύει κυρίως με shortcuts (συντομεύσεις πληκτρολογίου) οπότε το tutorial φάνηκε εξαιρετικά χρήσιμο σε αυτό το κομμάτι. Το Blender μπορεί να γίνει πολύ δύσχρηστο για κάποιον αρχάριο επειδή απευθύνεται σε ειδικούς του τομέα εκείνου. Αφού εισήχθη η εικόνα, χρησιμοποιήθηκαν δύο σχήματα για να λειτουργήσουν ως σημεία αναφοράς για το ποια μέρη της εικόνας θα κοπούν. Αρχικά δημιουργήθηκε ένας κύκλος με collider και βάσει αυτού κόπηκε το εξωτερικό κομμάτι της εικόνας, δηλαδή παρέμεινε ο κύκλος του δαχτυλιδιού αλλά με γέμισμα στο κέντρο. Το εξωτερικό περίβλημα διαγράφηκε. Έπειτα, δημιουργήθηκε ένας κύλινδρος που μπήκε στο κέντρο της εικόνας και μεταβλήθηκε το μέγεθός του ώστε να καλύπτει το γέμισμα του δαχτυλιδιού (Εικόνα 5.9), κι έτσι αποκόπηκε εκείνο το σημείο από την εικόνα και διαγράφηκε. Χρησιμοποιήθηκε το σχήμα του κυλίνδρου για να είναι πιο εύκολος ο μετασχηματισμός του. Έτσι έμεινε μόνο το σημαντικό σχήμα της εικόνας που περιείχε μόνο το δαχτυλίδι. Οι αποκοπές έγιναν εφαρμόζοντας έναν Boolean modifier στην εικόνα, με αντικείμενο το σχήμα που έπρεπε να αποκοπεί.



Εικόνα 5.9: Ο κύλινδρος μέσα στο δαχτυλίδι που πρόκειται να αποκοπεί

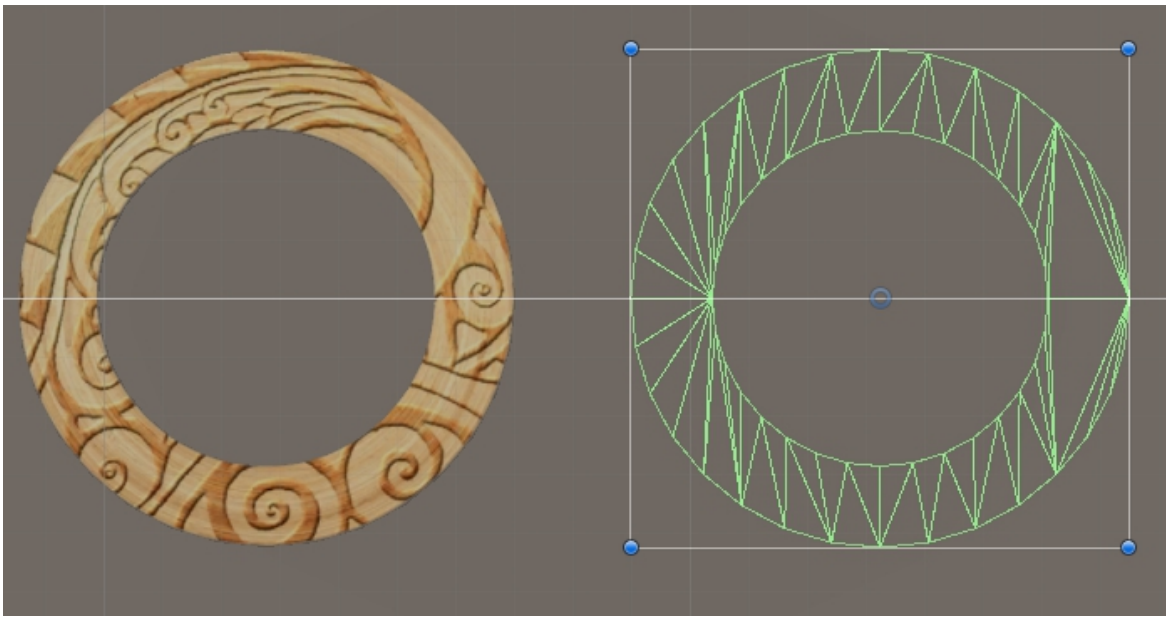
Το τελικό αποτέλεσμα φαίνεται στην εικόνα 5.10, όπου έχει μείνει μόνο το δαχτυλίδι. Απομονώθηκε σε ένα ξεχωριστό Collection, πάνω δεξιά στην εικόνα, και απενεργοποιήθηκαν τα υπόλοιπα Collections, ώστε να γίνει εξαγωγή του μοντέλου μόνο του, μιας και αυτόματα είχαν δημιουργηθεί ένα Directional light και μια κάμερα όπως με τη δημιουργία κάθε νέου project. Η εξαγωγή έγινε από το μενού File → Export → FBX(.fbx). Το αρχείο που παράχθηκε είναι ένα είδος αρχείου γραφικών μοντέλων που είναι συμβατό με την πλατφόρμα του Unity.



Εικόνα 5.10: Το τελικό μοντέλο για το τρίτο δαχτυλίδι στο περιβάλλον του Blender

Το αρχείο .fbx προστέθηκε στα prefabs του project με ένα drag-and-drop, και από τον inspector του χρειάστηκε να γίνει Extract Materials ώστε να διαχωριστεί το υλικό του από το ίδιο, για να μπορεί να είναι επεξεργάσιμο. Ήταν θεμιτό να απενεργοποιηθούν οι επιλογές Specular highlights και Reflections από το υλικό όπως συνέβη με όλα τα υλικά σε αυτό το project για να φωτίζονται σωστά στη σκηνή. Πλέον τα δαχτυλίδια έχουν τον κατάλληλο collider (Εικόνα 5.11).

Όλη η παραπάνω διαδικασία επαναλήφθηκε με τα άλλα δύο δαχτυλίδια, με διαφορά ότι στο πρώτο δαχτυλίδι, που είναι κύκλος, δεν χρειάστηκε η αποκοπή με τον κύλινδρο, μιας και δεν έχει τρύπα στο γέμισμά του. Τα αντικείμενα προστέθηκαν στη σκηνή στο κέντρο του Image Target και με κοινό κέντρο μεταξύ τους ώστε να δίνουν την εικόνα ενός ενιαίου κύκλου. Επίσης άλλαξε η περιστροφή τους στον άξονα ζ ώστε να μην φαίνεται σωστό άρα και λυμένο το σχήμα. Ο παίκτης θα χρειαστεί να τα γυρίσει ώστε να φέρει τον ζ άξονα στη σωστή θέση για το καθένα.



Εικόνα 5.11: Ο τρίτος κύκλος με τον σωστό Collider

5.4 Αρχεία κώδικα για τον δίσκο

Μόνο τρία αρχεία κώδικα γράφτηκαν για αυτόν το γρίφο, όπου το ένα είναι παραλλαγή ενός ήδη γραμμένου script. Χρησιμοποιήθηκε το script Spin Circle που, παρόμοια με το Spin (Κεφάλαιο 4.3.1) περιστρέφει το αντικείμενο γύρω από τον εαυτό του σε μια συγκεκριμένη ταχύτητα.

To script Spin Circle:

```
void Update()
   {
            transform.Rotate(Vector3.back * Time.deltaTime * 20f);
   }
```

Με την εντολή Vector3.back το αντικείμενο γυρνάει προς τα δεξιά στον άξονα ζ, και η ταχύτητα του τίθεται με τον πολλαπλασιαστή Time.deltaTime επί είκοσι μονάδες χρόνου. Το script αυτό ενσωματώθηκε στον ολοκληρωμένο κύκλο που θα εμφανιστεί όταν ο παίκτης λύσει το γρίφο, όπως έγινε και με τον δεύτερο γρίφο. Η διαφορά των scripts έγκειται στον άξονα περιστροφής, μιας και στον κύβο χρειαζόταν να γυρνάει στον άξονα ψ, άρα χρησιμοποιήθηκε η εντολή Vector3.down.

5.4.1 Rotate Circle

Στις εικόνες 5.12 & 5.13 φαίνεται το script Rotate Circle και αναλύεται παρακάτω.

```
using System.Collections;
using System.Collections.Generic;
using UnityEngine;

public class RotateCircle : MonoBehaviour
{
    private float spee;
    public GameObject pplane, r1, r2, r3;
    public float f1, f2, f3;
    public bool b1, b2, b3;
    private CheckCircles circl;

    void Start()
    {
        spee = 20f;
        //η ταχύτητα με την οποία θα περιστρέφονται τα δαχτυλίδια
        circl = GetComponent<CheckCircles>();
        // το script που θα ελέγχει αν λύθηκε ο γρίφος
    }

    void Update()
    {

        if (Input.touchCount > 0) //αν υπάρχει άγγιγμα
        {
            Touch touch = Input.GetTouch(0);    //το άγγιγμα

            if (touch.phase == TouchPhase.Began){ //οταν ξεκινήσει το άγγιγμα
                Ray raycast = Camera.main.ScreenPointToRay(Input.GetTouch(0).position);
                //το άγγιγμα μεταφρασμένο στη σκηνή
                RaycastHit raycastHit;
                //το αντικείμενο που θα αγγιχθεί με βάση τον collider του
                if (Physics.Raycast(raycast, out raycastHit))
                    //αν αγγίχθηκε κάποιο αντικείμενο
                {
                    if (raycastHit.collider.CompareTag("Circle")){
                        //αν το αντικείμενο είναι τύπου Circle
                        pplane = GameObject.Find(raycastHit.collider.name);
                        //το αντικείμενο που αγγίχθηκε ως GameObject
                    }
                }
            }
```

Εικόνα 5.12: Το πρώτο μέρος του script Rotate Circle

```
44              if ((touch.phase == TouchPhase.Moved) & (pplane!=null))
45                  //αν κινήθηκε το άγγιγμα και υπάρχει επιλεγμένο αντικείμενο
46              {
47                  Vector2 touchDeltaPosition = Input.GetTouch(0).deltaPosition;
48                  //η απόσταση που διένυσε το άγγιγμα
49                  pplane.transform.Rotate(0f, 0f, -touchDeltaPosition.y  *Time.deltaTime * spee);
50                  //περιστρέφει το αντικείμενο στον ζ άξονα κατά την κατακόρυφη κατεύθυνση του
51                  //αγγίγματος με κάποια ταχύτητα επί την ορισμένη ταχύτητα
52
53              }
54              if (touch.phase == TouchPhase.Ended) //αν τελείωσε το άγγιγμα
55              {
56                  pplane = null; //το επιλεγμένο αντικείμενο είναι κενό
57              }
58          }
59
60          //οι τοπικές περιστροφές στον άξονα ψ των δαχτυλιδιών
61          f1 = r1.transform.localEulerAngles.y;
62          f2 = r2.transform.localEulerAngles.y;
63          f3 = r3.transform.localEulerAngles.y;
64
65          //αν η κάθε μία περιστροφή είναι περίπου 180 μοίρες
66          //η αντίστοιχη Boolean μεταβλητή γίνεται αληθής αλλιώς ψευδής
67          if ((f1 > 174f) & (f1 < 186f)) { b1 = true;} else {b1 = false;}
68          if ((f2 > 174f) & (f2 < 186f)) { b2 = true;} else {b2 = false;}
69          if ((f3 > 174f) & (f3 < 186f)) { b3 = true;} else {b3 = false;}
70
71          if (b1 & b2 &b3)  //αν όλες οι boolean μεταβλητές είναι αληθείς
72          {
73              circl.b1 = true;
74              //η boolean μεταβλητή του script Check Circles γίνεται αληθής
75
76          }
77
78      }
79  }
```

Εικόνα 5.13: Το δεύτερο μέρος του script Rotate Circle

Το άγγιγμα της οθόνης συμβαίνει σε δύο άξονες, κι έτσι μεταφράζεται και στην περιστροφή των δαχτυλιδιών. Αν ο παίκτης σύρει το δάκτυλό του προς τα κάτω, το δαχτυλίδι θα περιστραφεί προς την αρνητική πλευρά του ζ άξονά του. Αν το σύρει προς τα επάνω, θα περιστραφεί προς την θετική πλευρά του ζ άξονά του. Έχει τεθεί το σημείο των 180 μοιρών ως η σωστή απεικόνιση των δαχτυλιδιών, αλλά επειδή αυτό θα ήταν πολύ δύσκολο να επιτευχθεί με το μάτι, έχει αφεθεί ένα περιθώριο των δέκα μοιρών, ώστε αν ο παίκτης βάλει τα κομμάτια περίπου στις 180 μοίρες, να θεωρηθεί σωστή η τοποθέτησή τους. Αυτό το script προστέθηκε στο ίδιο το Image Target.

5.4.2 Check Circles

Καθ' όλη τη διάρκεια που είναι ενεργό αυτό το Image Target, τρέχει και το script Check Circles (Εικόνα 5.14) που, όπως και στους προηγούμενους δύο γρίφους, ελέγχει για το αν πληρούνται οι προϋποθέσεις για την επίλυση του γρίφου και εμφανίζει το κατάλληλο αντικείμενο.

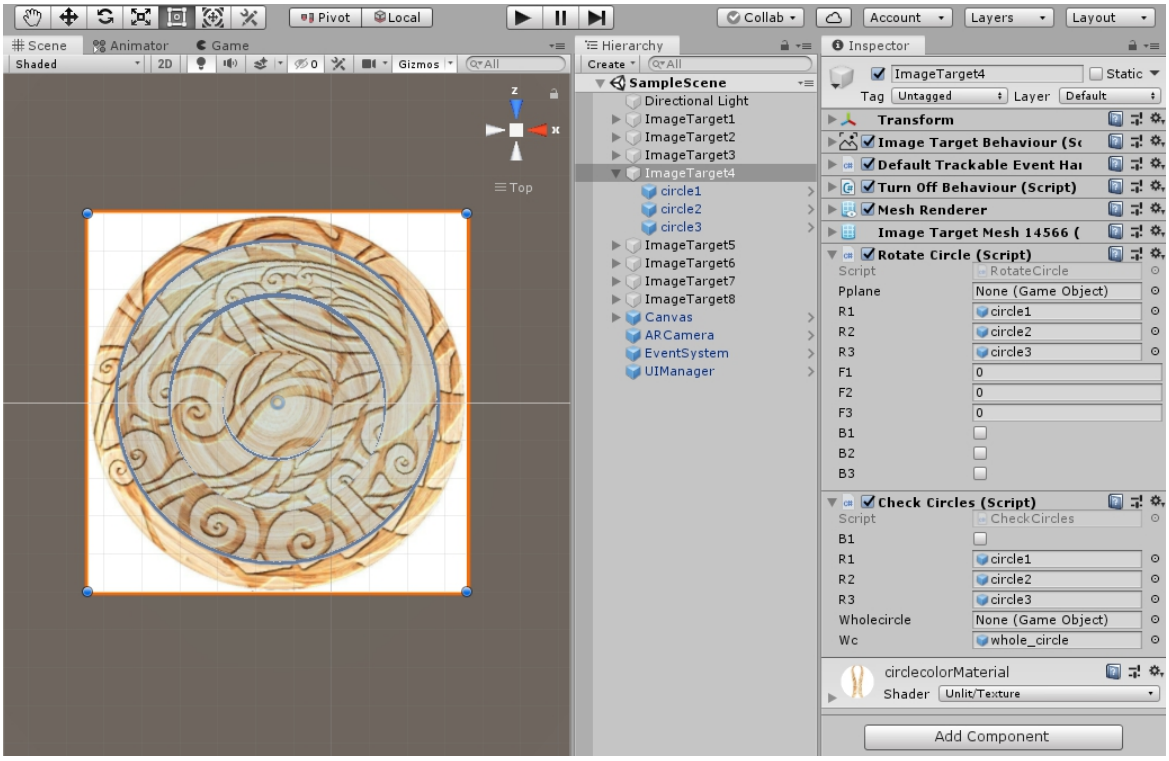
```
1 using System.Collections;
2 using System.Collections.Generic;
3 using UnityEngine;
4
5 public class CheckCircles : MonoBehaviour
6 {
7     public bool b1; //η μεταβλητή που τίθεται από το Rotate Circle
8     public GameObject r1, r2, r3, wholecircle, wc;
9     //τα τρία δαχτυλίδια, ο ολοκληρωμένος κύκλος που θα
10    //δημιουργηθεί, και το μοντέλο του που τίθεται από τον inspector
11
12    void Update()
13    {
14        if (b1){    //αν όλα τα δαχτυλίδια είναι στη σωστή θέση
15            GetComponent<RotateCircle>().enabled = false;
16            //απενεργοποιείται το script Rotate Circle
17            r1.SetActive(false);    //και τα δαχτυλίδια
18            r2.SetActive(false);    //και εξαφανίζονται
19            r3.SetActive(false);    //από την σκηνή
20
21            wholecircle = Instantiate (wc, this.transform);
22            //δημιουργείται ένα αντικείμενο που απεικονίζει
23            //ολοκληρωμένο τον δίσκο με γονέα το Image Target
24            b1 = false; //γίνεται ψευδής η μεταβλητή του
25            //ελέγχου για να μην επαναλαμβάνεται όλο το script
26        }
```

Εικόνα 5.14: Το script Check Circles

Το αντικείμενο που εμφανίζεται μετά την επίλυση του γρίφου είναι αυτό της εικόνας 5.2 και γυρνάει με μια αργή ταχύτητα με το script Spin Circle (Κεφάλαιο 5.4) που είναι προσκολλημένο στο ίδιο το prefab του ολοκληρωμένου δίσκου. Η συνθήκη σε αυτό το script με το που τεθεί αληθής από το προηγούμενο script, είναι για πάντα αληθής. Οπότε επιβλήθηκε το να απενεργοποιηθεί αφού κάνει τις ενέργειές της η μέθοδος Update(). Το script αυτό προστέθηκε ως συστατικό στο Image Target του γρίφου.

5.5 Επίλογος

Σε αυτό το σημείο έχει ολοκληρωθεί και ο τρίτος γρίφος του παιχνιδιού. Παρότι είναι ένας πιο εύκολος γρίφος στην ανάπτυξή του αλλά και στην επίλυσή του, παρουσίασε ενδιαφέρουσες δυσκολίες όσον αφορά τη δημιουργία των μοντέλων. Η παραγωγή τους έδωσε την ευκαιρία για την εξοικείωση με ένα νέο εργαλείο ανάπτυξης λογισμικού, έξω από την ακτίνα του κλάδου του προγραμματισμού, σε μία πιο δημιουργική κλίμακα. Στην εικόνα 5.15 παρουσιάζεται ολοκληρωμένος ο τρίτος γρίφος με ανοιχτό τον Inspector του Image Target.



Εικόνα 5.15: Ο τρίτος γρίφος ολοκληρωμένος στον Editor

Κεφάλαιο 6ο 4ος γρίφος – Σφαίρα

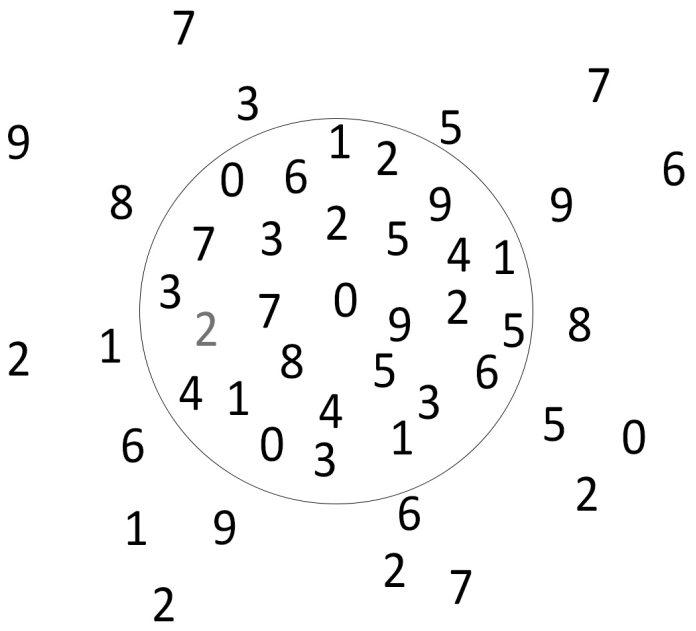
6.1 Εισαγωγή

Αυτός ο γρίφος ήταν από τις πρώτες ιδέες για αυτό το παιχνίδι, αλλά ο τελευταίος που υλοποιήθηκε λόγω των δυσκολιών που παρουσιάστηκαν στην δημιουργία του μοντέλου του. Στον γρίφο αυτό εμφανίζεται μια σφαίρα μπροστά από μία εικόνα με διάσπαρτους αριθμούς. Η σφαίρα είναι διάτρητη σε πολλά σημεία ώστε ο παίκτης μπορεί να δει από μέσα της τους αριθμούς πίσω από το μοντέλο. Σκοπός του είναι, ευθυγραμμίζοντας και αυξομειώνοντας το μέγεθος της σφαίρας, να καταφέρει να δει καθαρά τέσσερις αριθμούς. Αφού τους βρει, καλείται να τους εισάγει σε ένα άλλο Image Target που εμφανίζει τέσσερις θέσεις για αριθμούς. Αυτό είναι και ένα στοιχείο που έχει ο παίκτης για να ξέρει πόσους αριθμούς ψάχνει μέσω της σφαίρας. Δεν φαίνεται ξεκάθαρα κανένας άλλος συνδυασμός τεσσάρων αριθμών μέσω της σφαίρας.

6.2 Εικόνες για τη σφαίρα

Δύο Image Target δημιουργήθηκαν για αυτόν το γρίφο. Το πρώτο είναι εικόνα με διάσπαρτους αριθμούς και έναν κύκλο στο κέντρο όπου βρίσκονται αριθμοί σε μεγαλύτερη πυκνότητα (Εικόνα 6.1). Το δεύτερο είναι η εικόνα μιας κλειδαριάς (Εικόνα 6.2). Αυτή η εικόνα κατέβηκε από το διαδίκτυο και δεν σχεδιάστηκε αποκλειστικά για το παιχνίδι. Για το υλικό της σφαίρας χρησιμοποιήθηκε ένα απλό texture που δίνει την αίσθηση του ξύλου σε ανοιχτή απόχρωση. Για το υλικό των 3D αριθμών χρησιμοποιήθηκαν δύο απλά texture που δίνουν την αίσθηση της matte υφής, ένα σκουρόχρωμο κι ένα ανοιχτόχρωμο.

Εικόνα 6.2: Η εικόνα του δεύτερου Image Target για την εισαγωγή των αριθμών



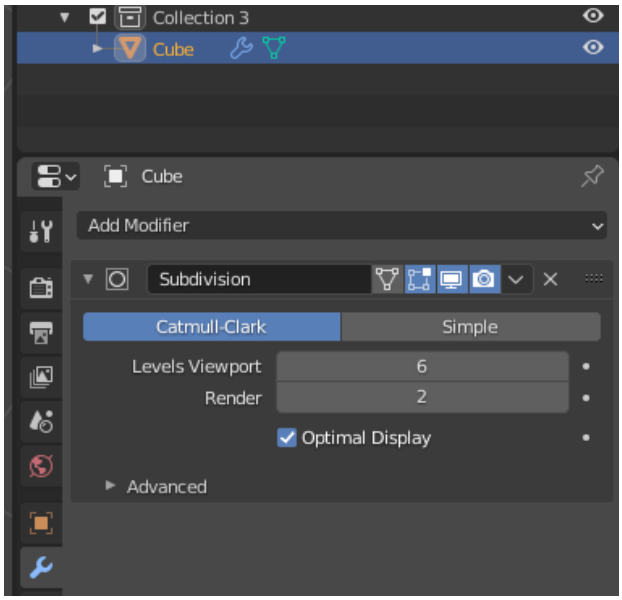
Εικόνα 6.1: Η εικόνα του πρώτου Image Target για την σφαίρα

6.3 Αντικείμενα για τον 4ο γρίφο

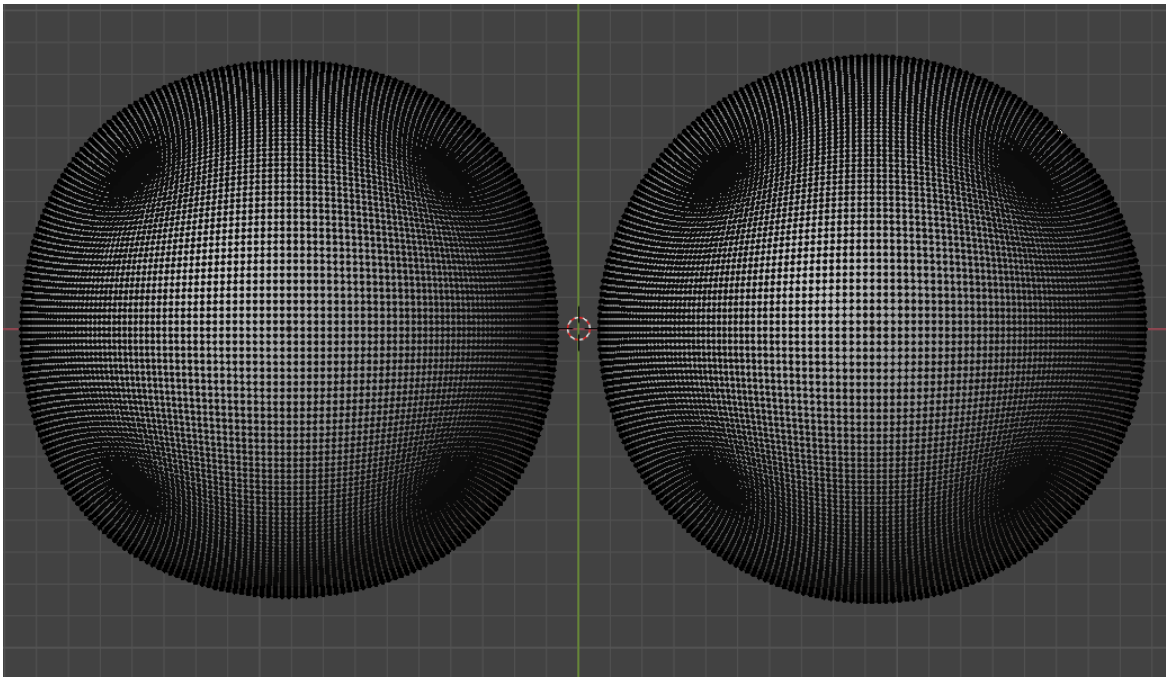
6.3.1 Σφαίρα

Για το μοντέλο της σφαίρας ήταν σίγουρο ότι θα χρειαζόταν κάποιο εργαλείο ανάπτυξης γραφικών. Όπως και στον προηγούμενο γρίφο, δημιουργήθηκε ένα νέο project με το εργαλείο Blender. Λόγω έλλειψης εμπειρίας πάνω σε αυτό το αντικείμενο, πάλι ήταν εξαιρετικά χρήσιμο ένα tutorial video για το πως φτιάχνεται μία λεία σφαίρα στο Blender [19].

Με τη δημιουργία νέου project στο Blender δημιουργείται αυτόματα ένας κύβος στο κέντρο της σκηνής. Σε αυτόν προστέθηκε ένας modifier για να πάρει το σχήμα της σφαίρας: Add modifier → Subdivision Surface και με τον αλγόριθμο Catmull - Clark ανέβηκε το Level Viewport στο 6 (Εικόνα 6.3). Έπειτα, πατώντας Tab για την εισαγωγή στο Edit mode, με τη συντόμευση Alt + Shift + S και πατώντας το 1 το σχήμα πήρε μία πιο σφαιρική μορφή (Εικόνα 6.4). Τέλος, επιλέχθηκε από το μενού Object το Shade Smooth για να γίνει ακόμη πιο λεία η επιφάνεια.

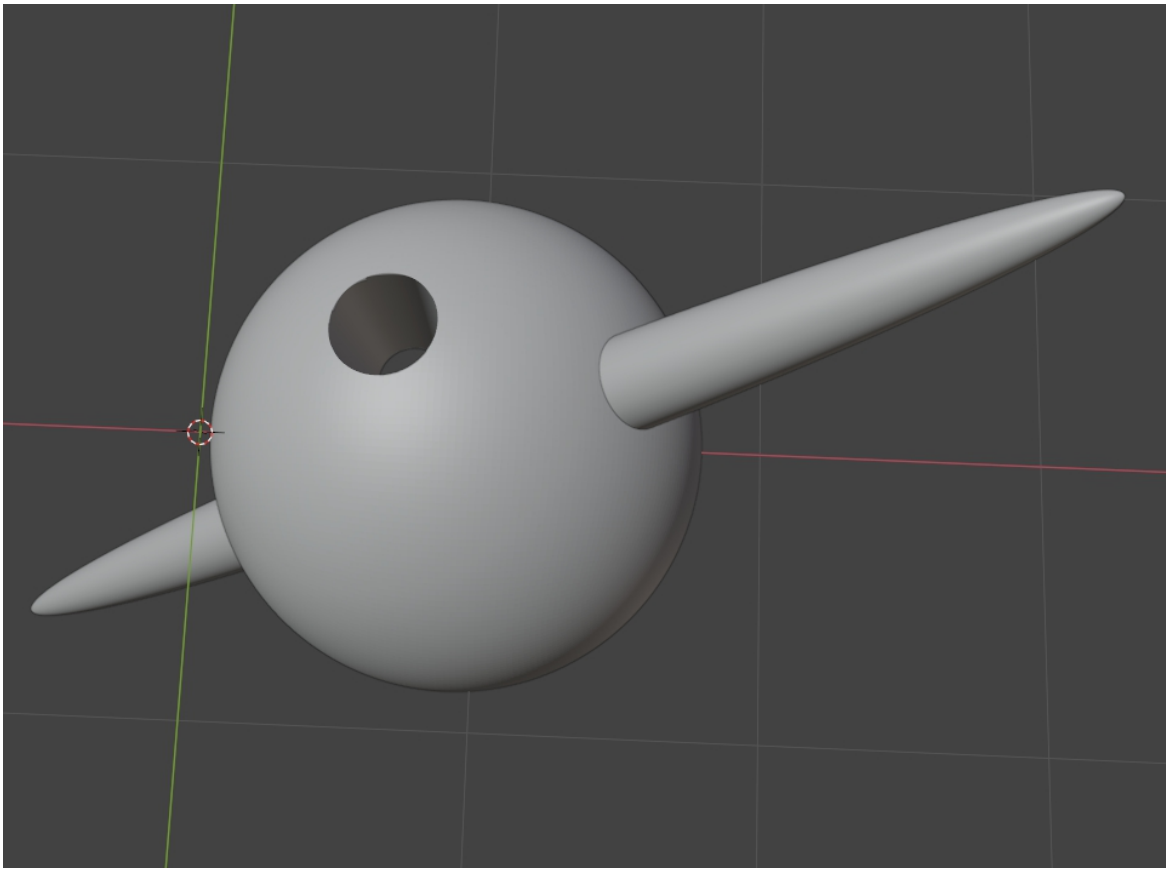


Εικόνα 6.3: Το Subdivision Surface modifier



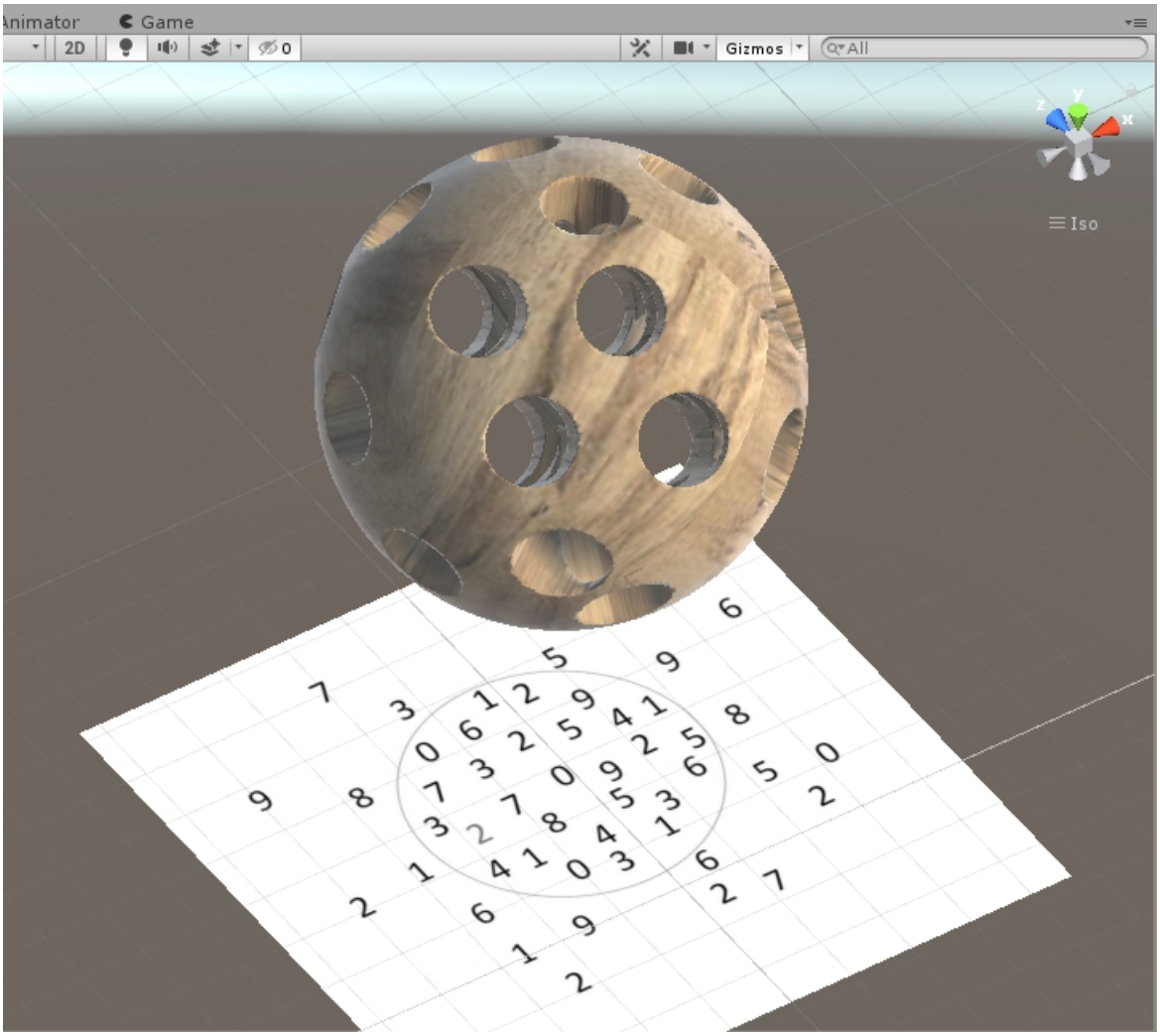
Εικόνα 6.4: Το σχήμα από κύβος σε σφαίρα

Μετά δημιουργήθηκε ένα άλλο αντικείμενο με το οποίο γίνονταν οι τρύπες στη σφαίρα. Αρχικά επιλέχθηκε το σχήμα ενός μακρύ κυλίνδρου, όμως όταν η επιφάνειά του έγινε λεία, επειδή αυτό συμβαίνει με διαφορετικό τρόπο απ' ότι στη σφαίρα, δεν κατάφερε να γίνει σωστό render όταν δημιουργούσε τις τρύπες. Αν πάλι δεν γινόταν λεία η επιφάνειά του, οι τρύπες δεν θα ήταν ολοστρόγγυλες αλλά θα είχαν μικρές γωνίες στην περίμετρό τους. Αυτό δεν ήταν θεμιτό για αισθητικούς λόγους. Οπότε δημιουργήθηκε άλλη μία σφαίρα με τον παραπάνω τρόπο, όμως στη δεύτερη μεταβλήθηκε το σχήμα της ώστε να εξυπηρετήσει ως κύλινδρος (Εικόνα 6.5). Με την ίδια τεχνική όπως και με τα δαχτυλίδια του τρίτου γρίφου, έγινε αποκοπή στη σφαίρα στα σημεία που τοποθετήθηκε ο κύλινδρος (Add modifier → Boolean με αντικείμενο τον κύλινδρο). Τέσσερα σημεία μετρήθηκαν με ακρίβεια για να συμπίπτουν με τα σημεία της εικόνας με τους αριθμούς, ώστε να φαίνονται ξεκάθαρα τέσσερις αριθμοί. Έγιναν πολλές ακόμη τρύπες πάνω στη σφαίρα, σε τυχαία σημεία, όμως από καμία άλλη όψη της σφαίρας δεν έχουν ανοίξει τέσσερις τρύπες που να δείχνουν από την άλλη πλευρά, έτσι ώστε να υπάρχει μόνο μία πιθανή λύση για τον γρίφο.



Εικόνα 6.5: Ο κύλινδρος που τρυπάει τη σφαίρα

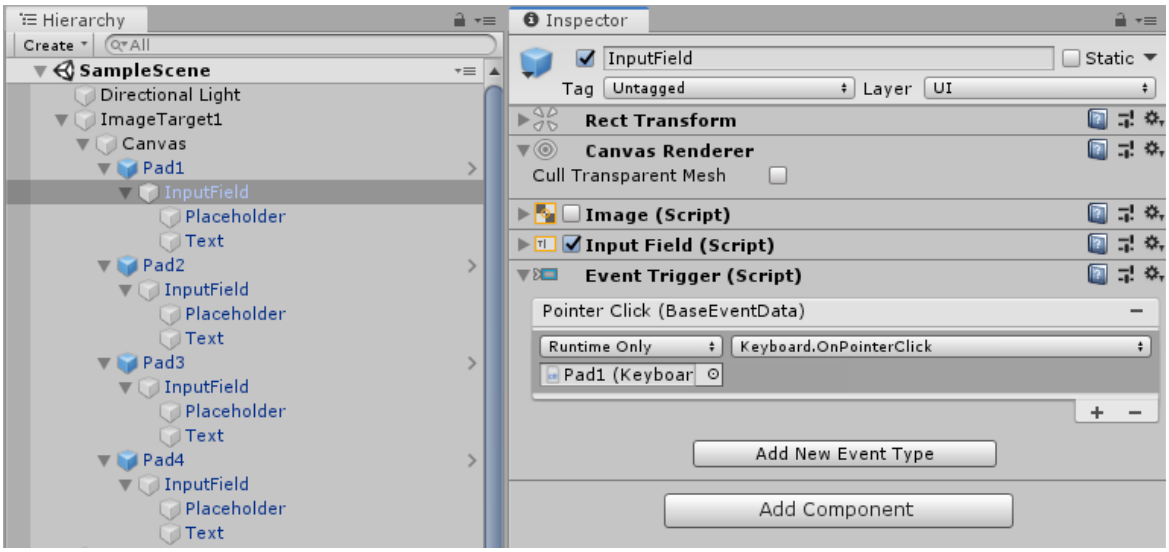
Απομονώθηκε η σφαίρα σαν αντικείμενο του project και έγινε File → Export → FBX(.fbx) του αντικειμένου. Το αρχείο προστέθηκε στο project του Unity με drag-and-drop στον φάκελο Prefabs → Sphere και από εκεί προστέθηκε στη σκηνή μέσα στο Image Target του γρίφου. Χωρίστηκε το material του ξανά από τον Inspector , και ανατέθηκε το απλό ξύλινο material στον mesh renderer του. Ανυψώθηκε από το Image Target και στοιχήθηκε ώστε να ευθυγραμμίζονται οι τρύπες με τέσσερις αριθμούς, και μετά άλλαξε η περιστροφή του για να μην είναι στη σωστή θέση όταν ξεκινάει το παιχνίδι (Εικόνα 6.6). Στο αντικείμενο προστέθηκε ως συστατικό ένα Rigidbody [20]. Το Rigidbody είναι απαραίτητο για το είδος των κινήσεων που είναι απαραίτητες για την επίλυση του γρίφου. Η σφαίρα θα γυρίζει γύρω από τον εαυτό της προς όλες τις κατευθύνσεις και θα αυξομειώνεται το μέγεθός της μέσω ενός script που θα αναλυθεί στο επόμενο κεφάλαιο.



Εικόνα 6.6: Η σφαίρα μέσα στη σκηνή του project στο Unity

6.3.2 Νούμερα

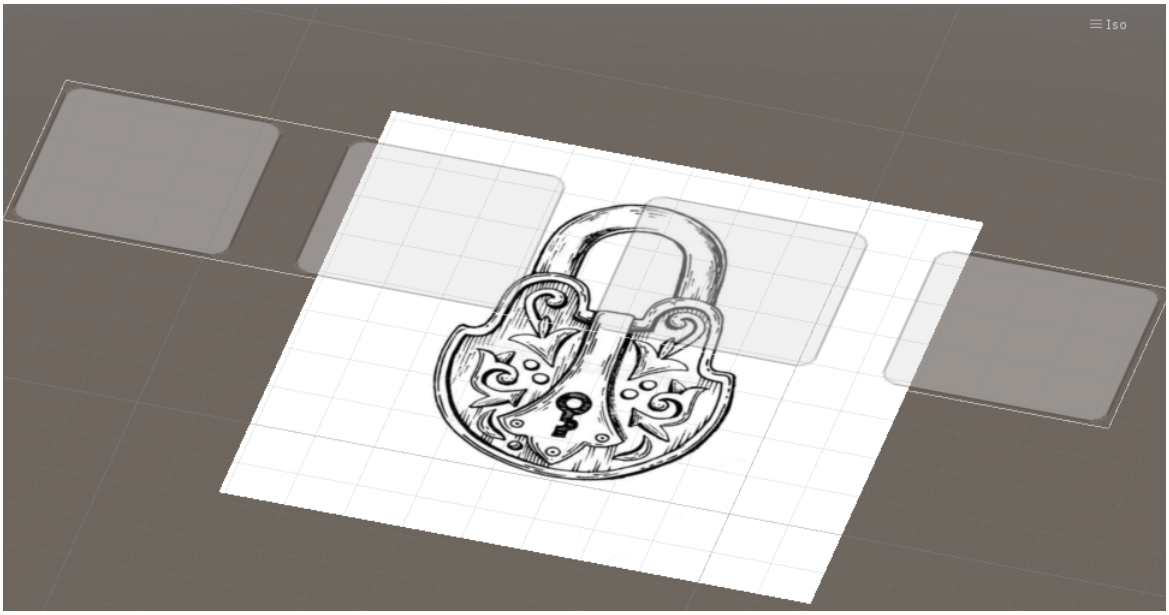
Για τα νούμερα προστέθηκε ακόμη ένα Image Target στο project, και του ανατέθηκε η εικόνα 6.2. Μέσα του προστέθηκε ένα Canvas, και μέσα στο Canvas τέσσερα Panels. Για την ακρίβεια πρώτα δημιουργήθηκε ένα Panel με όλα τα συστατικά του και μετά αποθηκεύτηκε ως prefab και αναπαράχθηκε άλλες τρεις φορές. Μέσα στο Panel προστέθηκε ένα GameObject → UI → Input Field [21]. Το Input Field δημιουργείται μαζί με τα συστατικά Image και Input Field (script). Διαγράφηκε το συστατικό Image, είναι θεμιτό να φαίνεται μόνο η εικόνα του panel, δηλαδή ένα ημιδιαφανές τετράγωνο (pad) όπου πάνω του θα επαυξάνεται το νούμερο. Μέσα στο Input Field υπάρχουν αυτόματα τα στοιχεία Placeholder και Text. Από το placeholder διαγράφηκε το κείμενο "Enter text..." που υπάρχει για να δείχνει ότι σε αυτό το σημείο μπορεί να γράψει ο χρήστης κάτι. Στο Text άλλαξε το χρώμα του κειμένου που θα εισέρχεται και μίκρυνε η γραμματοσειρά του, ώστε να μη φαίνεται ο αριθμός που θα γράψει ο παίκτης, μιας και θα εμφανίζεται η τρισδιάστατη μορφή του αριθμού πάνω στο pad. Στο Input Field προστέθηκε ως συστατικό ένα Event Trigger με τη μέθοδο Pointer Click ενός script που θα αναλυθεί παρακάτω (Εικόνα 6.7). Πατώντας πάνω στο Input Field θα ενεργοποιείται η μέθοδος. Τα pads που αναπαράχθηκαν στοιχήθηκαν το ένα δίπλα στο άλλο για να γράφονται οι αριθμοί με τη σειρά και όλο το Canvas ανυψώθηκε από το Image Target για να φαίνεται σαν να αιωρείται πάνω από την εικόνα (Εικόνα 6.8).



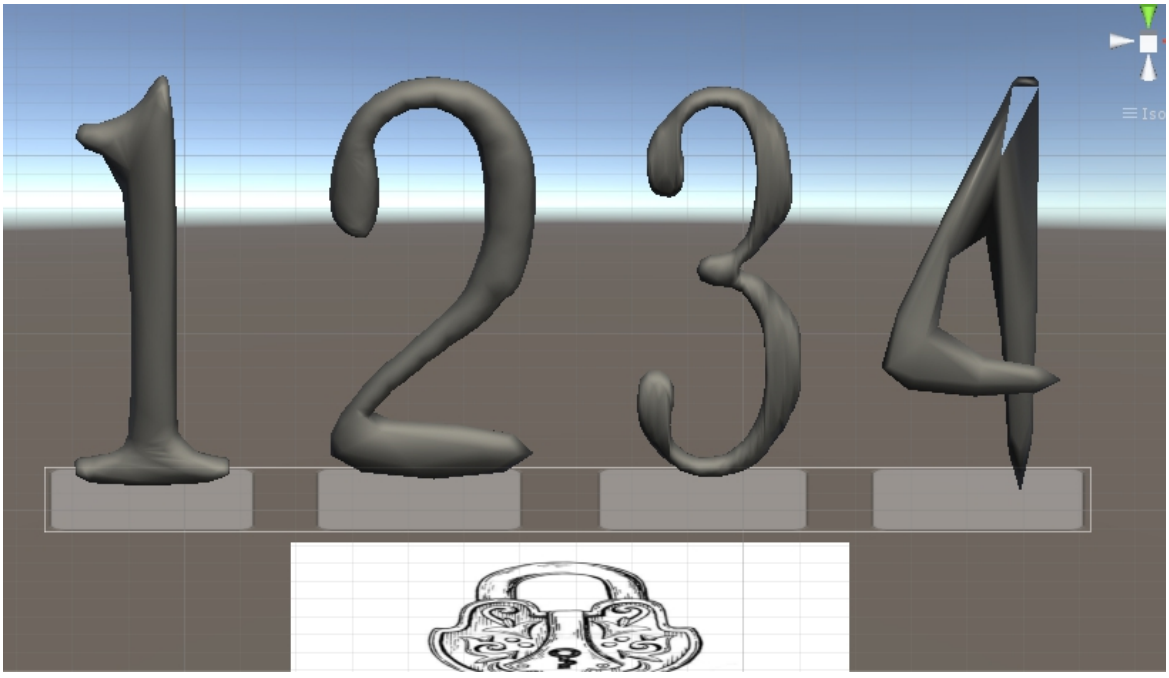
Εικόνα 6.7: Hierarchy και Inspector του Image Target των αριθμών

Στη συνέχεια έγινε μια αναζήτηση για δωρεάν 3D μοντέλα αριθμών στο διαδίκτυο που να ταιριάζουν στην αισθητική του παιχνιδιού. Από μία ιστοσελίδα μόνο με τη δημιουργία δωρεάν λογαριασμού [22] κατέβηκε το αρχείο με τους αριθμούς. Για ακόμη μια φορά χρειάστηκε η χρήση του προγράμματος Blender για να διαχωριστούν οι αριθμοί σαν μοντέλα μεταξύ τους.

Προστέθηκαν κι αυτοί ως prefabs και μεταβλήθηκε το μέγεθος και το material τους μέσω του Editor. Ανατέθηκε σε αυτούς ένα σκούρο απλό texture (Εικόνα 6.9).



Εικόνα 6.8: Το Image Target για την εισαγωγή των αριθμών



Εικόνα 6.9: Κάποιοι από τους αριθμούς μέσα στη σκηνή του Unity

6.4 Αρχεία κώδικα για τη σφαίρα

Για αυτόν τον γρίφο γράφτηκαν τρία scripts. Δύο για τους αριθμούς και ένα για τη σφαίρα.

6.4.1 Spin with Touch

Το script Spin with Touch προστέθηκε ως συστατικό στο μοντέλο της σφαίρας και του ανατέθηκαν στον inspector το Rigidbody της σφαίρας στη μεταβλητή rb, η ίδια η σφαίρα στη μεταβλητή m_objecttorotate, ο αριθμός 10 στο m_minScale και ο αριθμός 100 στο m_maxScale. Η σφαίρα έχει μέγεθος 40, 40, 40 στη σκηνή και θα φτάνει από 10, 10, 10 μέχρι 100, 100, 100. Το script ανιχνεύει αν αγγίχτηκε το αντικείμενο με ένα ή δύο δάχτυλα. Αν αγγίχτηκε με ένα, περιστρέφει στη σφαίρα στην κατεύθυνση που σύρθηκε το δάχτυλο. Αν αγγίχτηκε με δύο, αν σύρθηκε μόνο το ένα κάνει την ίδια διαδικασία, και αν σύρθηκαν και τα δύο ("τσίμπημα"), αλλάζει το μέγεθος της σφαίρας με βάση την απόσταση του συρσίματος (Εικόνες 6.10 & 6.11).

```
using UnityEngine;
using System.Collections;

public class SpinWithTouch : MonoBehaviour
{
    public Rigidbody rb;
    public GameObject m_objecttorotate;
    public int m_minScale;
    public int m_maxScale;
    private float initialFingersDistance;
    private Vector3 initialScale;

    void Update()
    {
        if (Input.touchCount == 1 && Input.GetTouch(0).phase == TouchPhase.Moved)
        //αν αγγίχτηκε η οθόνη σε ένα σημείο (δηλαδή με ένα δάχτυλο) και σύρθηκε το άγγιγμα
        {
            Vector2 touchDeltaPosition = Input.GetTouch(0).deltaPosition;
            //παίρνει την απόσταση που έκανε το άγγιγμα
            rb.AddTorque(Camera.main.transform.up * -touchDeltaPosition.x);
            //δίνει περιστροφική δύναμη στο Rigidbody της σφαίρας στον
            //άξονα ψ, όσο κινήθηκε το άγγιγμα στον άξονα χ της οθόνης
            rb.AddTorque(Camera.main.transform.right * touchDeltaPosition.y);
            //δίνει περιστροφική δύναμη στο Rigidbody της σφαίρας στον
            //άξονα χ, όσο κινήθηκε το άγγιγμα στον άξονα ψ της οθόνης
        }

        if (Input.touches.Length == 2)  //αν το άγγιγμα είναι με δύο δάχτυλα
        {
            _Scaling(); //καλεί τη μέθοδο _Scaling()
            return;     //και επιστρέφει
        }
    }
```

Εικόνα 6.10: Το πρώτο κομμάτι του script Spin with Touch

```
void _Scaling()
{
    if (Input.touches.Length == 2)
    {
        Touch t1 = Input.touches[0];    //το πρώτο άγγιγμα
        Touch t2 = Input.touches[1];    //το δεύτερο άγγιγμα

        if (t1.phase == TouchPhase.Began || t2.phase == TouchPhase.Began)
            //αν ξεκίνησε ένα από τα δύο αγγίγματα
        {
            initialFingersDistance = Vector2.Distance(t1.position, t2.position);
            //παίρνει την αρχική απόσταση των αγγιγμάτων
            initialScale = m_objecttorotate.transform.localScale;
            //παίρνει το αρχικό μέγεθος της σφαίρας (τέθηκε από τον inspector η σφαίρα)
        }
        else if (t1.phase == TouchPhase.Stationary && t2.phase == TouchPhase.Moved)
            //αν μόνο ένα από τα αγγίγματα κινηθεί επαναλαμβάνεται η
            //διαδικασία της περιστροφής της σφαίρας
            {
                Vector2 touchDeltaPosition = Input.GetTouch(0).deltaPosition;
                rb.AddTorque(Camera.main.transform.up * -touchDeltaPosition.x);
                rb.AddTorque(Camera.main.transform.right * touchDeltaPosition.y);
            }

        else if (t1.phase == TouchPhase.Moved && t2.phase == TouchPhase.Moved)
            //αν κουνηθούν και τα δύο αγγίγματα
        {
            float currentFingersDistance = Vector2.Distance(t1.position, t2.position);
            //παίρνει την δεύτερη απόσταση των αγγιγμάτων
            var scaleFactor = currentFingersDistance / initialFingersDistance;
            //βρίσκει την απόσταση που έκαναν τα αγγίγματα
            Vector3 m_scale= initialScale * scaleFactor;
            //πολλαπλασιάζει το μέγεθος της σφαίρας με την απόσταση
            //και ελέγχει τον κάθε άξονα να μην ξεπερνάει τα όρια
            //μέγιστου και ελάχιστου που τέθηκαν στον Inspector
            m_scale.x = Mathf.Clamp(m_scale.x, m_minScale,m_maxScale);
            m_scale.y = Mathf.Clamp(m_scale.y, m_minScale, m_maxScale);
            m_scale.z = Mathf.Clamp(m_scale.z, m_minScale, m_maxScale);
            m_objecttorotate.transform.localScale = m_scale;
            //αλλάζει το μέγεθος της σφαίρας
        }
    }
}
```

Εικόνα 6.11: Η μέθοδος *Scaling()* του script Spin with Touch

6.4.2 Keyboard

Για τους αριθμούς, σε κάθε number pad από τα τέσσερα της σκηνής, προστέθηκε το script Keyboard. Η μέθοδος *OnPointerClick()* αυτού, καλείται όποτε ο χρήστης πατήσει σε κάποιο Pad, γι' αυτό και ανατέθηκε στο Event Trigger κάθε Input Field. Το script προστέθηκε στο Pad και όχι στο Input Field, ώστε όταν δημιουργούνται μοντέλα αριθμών, να μπαίνουν ως παιδιά πιο εύκολα στο Pad.

```
using System;
using System.Collections;
using System.Collections.Generic;
using UnityEngine;
using UnityEngine.UI;

public class Keyboard : MonoBehaviour
{
    private TouchScreenKeyboard keyboard;
    public string text = "";    //το κείμενο που θα εισαχθεί από τον παίκτη
    public GameObject zero, one, two, three, four, five, six, seven, eight, nine, number;
    //τα μοντέλα των αριθμών που τέθηκαν στο prefab από τον inspector
    public InputField inp;

    public void OnPointerClick(){
        keyboard = TouchScreenKeyboard.Open("", TouchScreenKeyboardType.NumberPad);
        //όταν αγγίξει ο παίκτης το input field ανοίγει ένα πληκτρολόγιο στην οθόνη
        //της συσκευής τύπου numberpad για να μπορεί να πατήσει μόνο νούμερα
        inp.onEndEdit.AddListener(delegate {Submit(inp.text);});
        //όταν τελειώσει η εισαγωγή και πατηθεί το κουμπί καταχώρησης
        //καλείται η μέθοδος Submit με attribute το κείμενο που εισήχθη
    }
    public void Submit(string t){
        text = t;   //το κείμενο - αριθμός
        int n = int.Parse(text);    //μετατράπηκε σε integer
        if ((n>=0) & (n<=9)){   //αν είναι μεταξύ 0-9
            Destroy(number);    //καταστρέφει το τυχόν ήδη υπάρχον μοντέλο

        switch (text){  //κάνει ενέργειες ανάλογα με το ποιος αριθμός εισήχθη
        case "0":       //αν είναι το 0
            number = Instantiate (zero, this.transform);
            //δημιουργείται το μοντέλο του αριθμού 0 με γονέα το Pad
            break;
        case "1":
            number = Instantiate (one, this.transform);
            break;
```

Εικόνα 6.12: Το script Keyboard

Το script συνεχίζει με τις υπόλοιπες οκτώ περιπτώσεις αριθμών και αναλόγως εμφανίζει το αντίστοιχο μοντέλο στο επιλεγμένο Pad. Χρησιμοποιήθηκε η κλάση TouchScreenKeyboard [23] για να ανοίξει και να ελεγχθεί το πληκτρολόγιο της συσκευής.

6.4.3 Check Numbers

Το τρίτο script που γράφτηκε γι αυτόν τον γρίφο προστέθηκε στο Image Target των αριθμών και ελέγχει αν εισήχθηκαν οι σωστοί αριθμοί (Εικόνα 6.13). Για αυτήν την έκδοση του παιχνιδιού, μέσω της σφαίρας φαίνονται οι αριθμοί 2794 με σειρά ύψους. Οπότε το script ελέγχει αν έχει μπει ο αριθμός 2 στο 1ο pad, ο αριθμός 7 στο 2ο pad, ο αριθμός 9 στο 3ο pad και ο αριθμός 4 στο 4ο pad. Αν ισχύει αυτή η συνθήκη, αλλάζει το material των μοντέλων των αριθμών σε ένα πιο ανοιχτό χρώμα. Αν αλλάξουν ξανά οι αριθμοί τους επαναφέρει στο προηγούμενο χρώμα. Στον inspector του script έχουν ανατεθεί τα δύο material ως brown και beige.

```
public Material beige, brown;
private Keyboard key1, key2, key3, key4;

void Start()
{
    //παίρνει τα 4 scripts από τα pads
    key1 = GameObject.Find("Pad1").GetComponent<Keyboard> ();
    key2 = GameObject.Find("Pad2").GetComponent<Keyboard> ();
    key3 = GameObject.Find("Pad3").GetComponent<Keyboard> ();
    key4 = GameObject.Find("Pad4").GetComponent<Keyboard> ();
}

void Update()
{
    if((key1.text=="2") & (key2.text=="7") & (key3.text=="9") & (key4.text=="4"))
    //αν έχουν και τα 4 τους σωστούς αριθμούς μετατρέπει τα materials
    //των μοντέλων στο beige που ορίστηκε στον inspector
    {
        if (key1.text != null)
        {
            key1.number.GetComponent<Renderer>().material = beige;
        }
        if (key2.text != null)
        {
            key2.number.GetComponent<Renderer>().material = beige;
        }
        if (key3.text != null)
        {
            key3.number.GetComponent<Renderer>().material = beige;
        }
        if (key4.text != null)
        {
            key4.number.GetComponent<Renderer>().material = beige;
        }
    }
    else
        //αν δεν είναι οι σωστοί αριθμοί επαναφέρει το προηγούμενο material
    {
        if (key1.number != null)
        {
            key1.number.GetComponent<Renderer>().material = brown;
        }
```

Εικόνα 6.13: Το script Check Numbers

6.5 Επίλογος

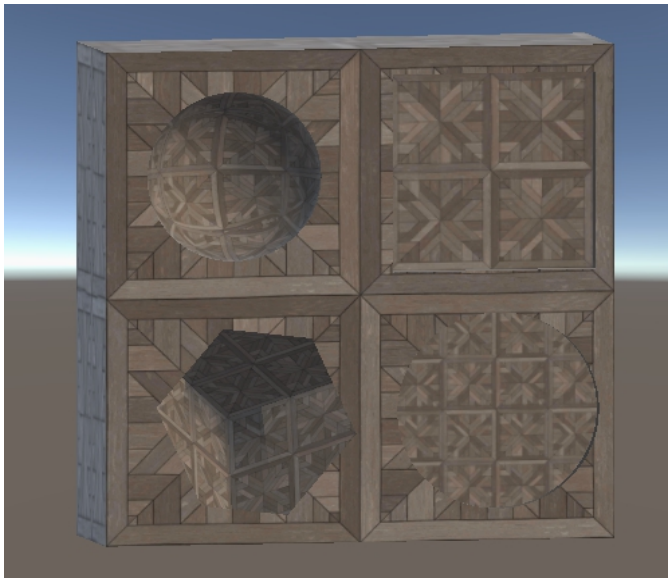
Σε αυτό το σημείο έχει ολοκληρωθεί και ο τέταρτος γρίφος του παιχνιδιού. Η δυσκολία που παρουσίασε η ανάπτυξή του βρισκόταν στη δημιουργία του μοντέλου της σφαίρας, μιας και αφορούσε ένα σχετικά άγνωστο αντικείμενο. Σαν κώδικας σίγουρα επιδέχεται βελτιώσεων, μιας και η αρχική ιδέα, που ήταν να γυρίζει με χειρονομία δύο δακτύλων η σφαίρα, δεν κατάφερε να υλοποιηθεί. Όσον αφορά τα νούμερα, χρειάστηκε να μελετηθούν νέες μέθοδοι, όπως η χειραγώγηση του πληκτρολογίου μιας εξωτερικής συσκευής και η αποκόμιση των πληροφοριών του. Με αυτόν το γρίφο ολοκληρώνεται το σετ γρίφων που είχε σχεδιαστεί για το project.

Δοκιμάζοντας σε τελικό επίπεδο την εφαρμογή στη συσκευή Android παρατηρήθηκε ότι η ανάλυση της εικόνας από την κάμερα της συσκευής δεν ήταν η καλύτερη δυνατή. Μετά από μία σύντομη έρευνα, διαπιστώθηκε πως το Vuforia δεν κάνει αυτόματη εστίαση (auto-focus) μέσω της κάμερας. Οπότε γράφτηκε ένα πολύ απλό script που ενεργοποιεί το auto-focus όταν ξεκινήσει η εφαρμογή.

Κεφάλαιο 7ο Τελικό επίπεδο

7.1 Εισαγωγή

Μετά το πέρας της δημιουργίας των τεσσάρων γρίφων, το παιχνίδι δεν έδινε την αίσθηση πως τελειώνει με κάποιον τρόπο. Παρόλο που προστέθηκαν μοντέλα των ολοκληρωμένων γρίφων, ακόμα και με κίνηση, σε όλους τους γρίφους με τη λύση τους, κρίθηκε απαραίτητη η προσθήκη ενός επιπλέον επιπέδου. Στο τελικό επίπεδο ο παίκτης θα καλείται να συγκεντρώσει όλα τα ολοκληρωμένα μοντέλα σε ένα κουτί, στο οποίο φαίνεται ότι λείπουν τέσσερα σχήματα, μία σφαίρα, ένας κύβος, ένας κύκλος και ένα τετράγωνο (Εικόνα 7.1). Αυτό το επίπεδο δεν έχει κάποια ιδιαίτερη δυσκολία για τον παίκτη, προσφέρει όμως ένα κλείσιμο στο παιχνίδι.

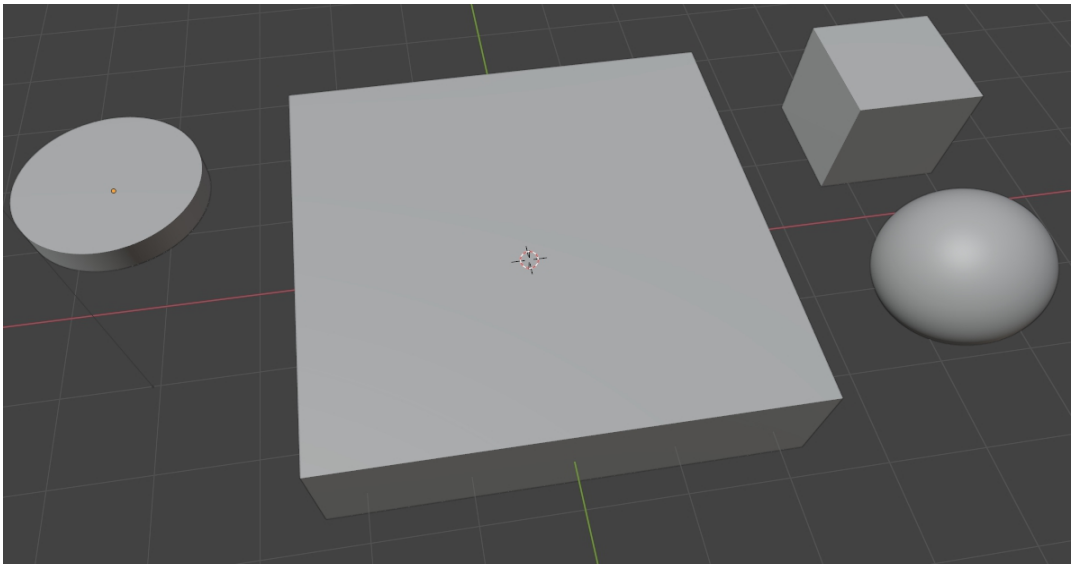


Εικόνα 7.1: Το κουτί/δοχείο στο οποίο πρέπει να τοποθετηθούν τα ολοκληρωμένα μοντέλα

7.2 Αντικείμενα για το δοχείο

Το μόνο αντικείμενο που χρειάστηκε να δημιουργηθεί για αυτό το επίπεδο ήταν ένα δοχείο και τέσσερα υπό-δοχεία για να τοποθετηθούν τα ολοκληρωμένα μοντέλα. Για άλλη μία φορά έγινε χρήση του προγράμματος Blender. Δημιουργήθηκε ένα νέο project και μέσα του ένας κύβος. Διαμορφώθηκε το σχήμα του ώστε το βάθος του να είναι μικρότερο από τις άλλες πλευρές, για να φαίνεται σαν τροπαιοθήκη. Σε αυτό έπρεπε να κοπούν τέσσερα σχήματα που να φαίνεται σαν να λείπουν από το σχήμα τέσσερα αντικείμενα διαφορετικού σχήματος. Έτσι προστέθηκαν δύο κύβοι και ένας κύλινδρος στη σκηνή. Ο ένας κύβος σμιλεύτηκε με τη μέθοδο που περιγράφεται στο

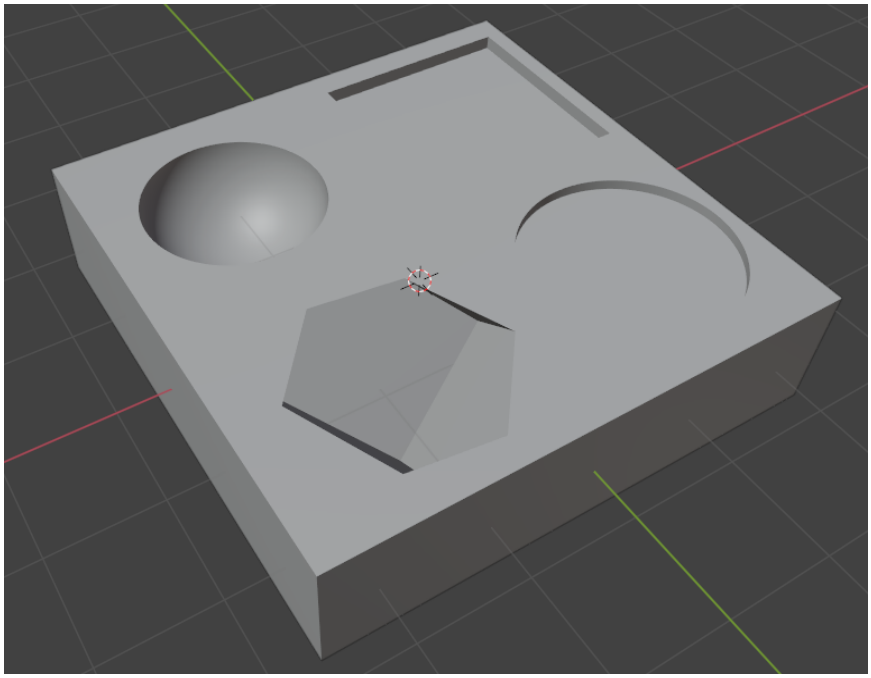
κεφάλαιο 6.3.1 ώστε να πάρει το σχήμα μιας σφαίρας, και ο κύλινδρος σμιλεύτηκε ώστε να είναι λείος (Εικόνα 7.2).



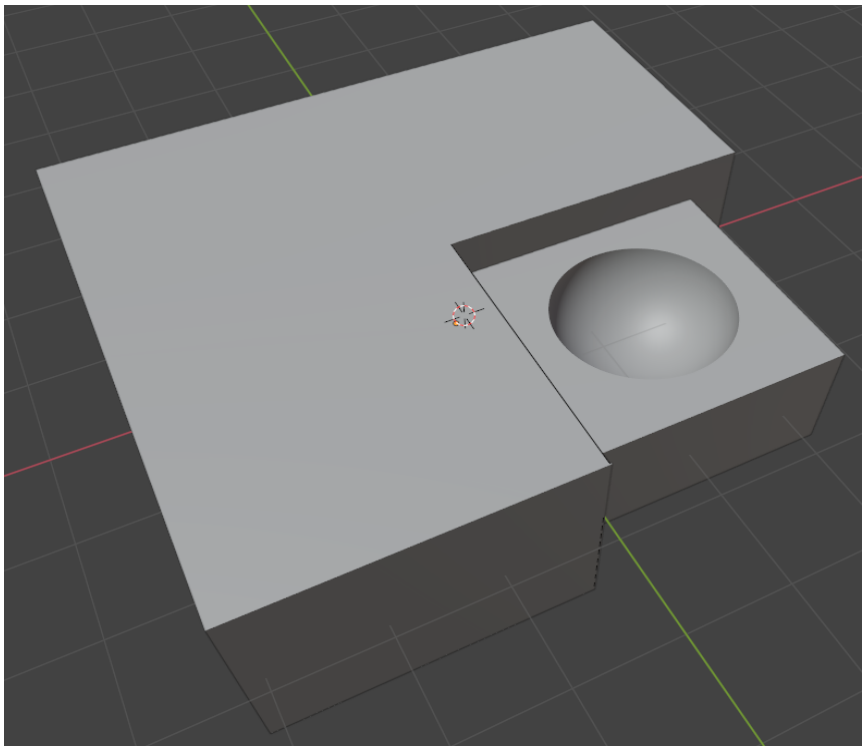
Εικόνα 7.2: Τα αντικείμενα που χρειάστηκαν για τις κοπές

Στο κάθε τεταρτημόριο του αρχικού σχήματος τοποθετήθηκε ένα σχήμα ώστε να γίνει Boolean τομή στο πρώτο. Πρώτα μεταβλήθηκαν τα σχήματα ώστε να χωρούν ομοιόμορφα στο αρχικό σχήμα. Πάνω αριστερά τοποθετήθηκε η σφαίρα μέχρι τη μέση της βαθιά στο σχήμα και κόπηκε από εκεί ώστε να φαίνεται ότι λείπει μία σφαίρα. Πάνω δεξιά τοποθετήθηκε ο κύβος ελάχιστα βαθιά στο σχήμα ώστε να φαίνεται ότι λείπει ένα τετράγωνο. Κάτω δεξιά τοποθετήθηκε ο κύλινδρος, ξανά ελάχιστα βαθιά στο σχήμα για να φαίνεται ότι λείπει ένας κύκλος. Τέλος, ο κύβος πήρε κλίση ίδια με την κλίση που έχει το μοντέλο του ολοκληρωμένου κύβου στο Unity και τοποθετήθηκε κάτω δεξιά του αρχικού σχήματος, μέχρι τη μέση ώστε να φαίνεται ότι λείπει ένας στραβός κύβος. Έγινε μετά από κάθε βήμα μία τομή Boolean στο αρχικό σχήμα στο κάθε τεταρτημόριο (Εικόνα 7.3).

Το τελικό σχήμα αντιγράφτηκε άλλες τέσσερις φορές σε ένα νέο Collection μέσα στη σκηνή. Δημιουργήθηκε ένα ίδιο σχήμα αλλά χωρίς τις τομές, και κόπηκε το ένα τεταρτημόριό του με έναν κύβο (Εικόνα 7.4). Το δεύτερο σχήμα χρησιμοποιήθηκε για να κοπούν τα τέσσερα τεταρτημόρια από το τελικό σχήμα, ώστε να υπάρχουν τέσσερα ξεχωριστά meshes που θα προστεθούν στο Unity project για καλύτερη διαχείριση. Τα τελικά μοντέλα που έγιναν export ήταν το ολοκληρωμένο δοχείο, και τα τέσσερα υπό-δοχεία, ως .fbx αρχεία και εισήλθαν στο Unity project με ένα drag-and-drop σε έναν νέο φάκελο στα prefabs. Δημιουργήθηκε ένα νέο Image Target στη σκηνή και τοποθετήθηκαν τα νέα μοντέλα. Εφαρμόστηκε στο screen (δοχείο) ένα σκούρο ξύλινο material με μοτίβο (Εικόνα 7.1).



Εικόνα 7.3: Το ολοκληρωμένο σχήμα του δοχείου



Εικόνα 7.4: Ο τρόπος που κόπηκαν τα τέσσερα υπό-δοχεία

7.3 Προσθήκες στον κώδικα και στη σκηνή

Μόνο ένα script γράφτηκε για αυτό το Image Target αλλά προστέθηκε κώδικας και σε άλλα τρία ήδη υπάρχοντα scripts: Move Plane, UI Manager, UI Triggers. Επίσης προστέθηκε ένα νέο είδος αντικειμένου (ταμπέλα - tag) στο project, το “Win”.

7.3.1 Προσθήκες σε scripts

Προστέθηκε μία γραμμή κώδικα και στα τέσσερα scripts που ελέγχουν αν λύθηκε ο κάθε γρίφος: Shelves, Check Numbers, Check Panels, Check Circles. Αν έχει λυθεί ο γρίφος, ενεργοποιείται μία Boolean μεταβλητή του script UI Manager που ελέγχει όλο το παιχνίδι. Ανάλογα το γρίφο ενεργοποιείται η μεταβλητή sp, ci, ma, ή cu:

GameObject.Find("UIManager").GetComponent<UIManager>().sp = true;

Στο script UI Manager στη μέθοδο Update που τρέχει σε κάθε frame προστέθηκε ο κώδικας που φαίνεται στην εικόνα 7.5. Αν έχουν λυθεί όλοι οι γρίφοι, τα ολοκληρωμένα αντικείμενα που υπάρχουν πλέον στη σκηνή (ο λαβύρινθος, ο κύβος, ο δίσκος και η σφαίρα που προϋπήρχε) παίρνουν τη νέα ταμπέλα “Win”. Με αυτόν τον τρόπο θα μπορεί ο παίκτης να τα κινήσει, μιας και στο script Move Plane όταν ελέγχεται η ταμπέλα των αντικειμένων προς μετακίνηση, προστέθηκε και το είδος “Win”. Το script Move Plane προστέθηκε πλέον σε όλα τα Image Targets, ώστε να μπορούν να μετακινηθούν παντού αντικείμενα με το κατάλληλο tag. Παρακάτω, σταματάει η περιστροφή όσων αντικειμένων περιστρέφονταν μετά τη λύση των γρίφων, και το αντικείμενο sphere2 του γρίφου της σφαίρας δεν μπορεί πλέον να περιστραφεί από τον παίκτη. Τέλος, απενεργοποιούνται οι Boolean μεταβλητές για να μην τρέχει συνέχεια η μέθοδος.

```
void Update()
{
    imgt = thescript.img;   //παίρνει το όνομα του ανιχνευμένου Image Target
    if (cu && ci && ma && sp){  //αν έχουν ενεργοποιηθεί όλες οι boolean μεταβλητές
        //βάζει σε όλα τα ολοκληρωμένα αντικείμενα το tag Win
        GameObject.Find("whole_maze(Clone)").tag = "Win";
        GameObject.Find("Whole_Cube(Clone)").tag = "Win";
        GameObject.Find("whole_circle(Clone)").tag = "Win";
        GameObject.Find("sphere2").tag = "Win";
        sh.GetComponent<Spin>().enabled = false;
        //σταματά να γυρνάν ο κύβος και ο δίσκος
        GameObject.Find("whole_circle(Clone)").GetComponent<Spin_Circle>().enabled = false;
        //ο δίσκος μπαίνει σε ορθή περιστροφή
        GameObject.Find("whole_circle(Clone)").transform.localEulerAngles = new Vector3(90f, 0f, 0f);
        //η σφαίρα δεν μπορεί πια να περιστραφεί
        GameObject.Find("sphere2").GetComponent<SpinWithTouch>().enabled = false;
        cu = ci = ma = sp = false;
        //απενεργοποιούνται οι boolean μεταβλητές για να μην τρέχει συνέχεια η μέθοδος
    }
}
```

Εικόνα 7.5: Η νέα μέθοδος Update του script UI Manager

Στο script UI Triggers προστέθηκαν τέσσερις έλεγχοι για την τοποθέτηση των αντικειμένων εντός και εκτός του Inventory. Όταν θα βγαίνουν από το Inventory τα νέα αντικείμενα, θα παίρνουν νέο μέγεθος και περιστροφή, ώστε να χωρούν στο δοχείο (Εικόνα 7.6). Οι έλεγχοι προστέθηκαν στη μέθοδο *OnPointerDown()* του script. Επιπλέον, προστέθηκε το συστατικό Image στα τέσσερα prefabs των αντικειμένων, με ένα εικονίδιο το οποίο θα εμφανίζεται στο Inventory όταν αυτά θα μπαίνουν εκεί.

```
//ανάλογα το αντικείμενο που εισέρχεται στη σκηνή, μετατρέπεται το μέγεθος
//και η κλίση του για να ταιριάζουν στο δοχείο
if (pplane2.name == "whole_circle(Clone)"){
    pplane2.transform.localScale = new Vector3(0.2f, 0.2f, 0.2f);
    pplane2.transform.localEulerAngles = new Vector3(0f,0f,0f);
}

if (pplane2.name == "sphere2"){
    pplane2.transform.localScale = new Vector3(9.8f, 9.8f, 9.8f);
    pplane2.transform.localEulerAngles = new Vector3(0f,0f,0f);
}

if (pplane2.name == "whole_maze(Clone)"){
    pplane2.transform.localScale = new Vector3(0.02f, 0.02f, 0.02f);
    pplane2.transform.localEulerAngles = new Vector3(90f,-90f,90f);
}

if (pplane2.name == "Whole_Cube(Clone)"){
    pplane2.transform.localScale = new Vector3(0.144f, 0.147f, 0.144f);
    pplane2.transform.localEulerAngles = new Vector3(214.5f, 130.2f, 297.7f);
}
```

Εικόνα 7.6: Οι έλεγχοι που προστέθηκαν στη μέθοδο *OnPointerDown()* του script UI Triggers

7.3.2 To script Screen

Μέσα στο νέο Image Target προστέθηκε ένα κενό αντικείμενο, το Shell2, και μέσα σε αυτό, το δοχείο (screen) με τα τέσσερα υπό-δοχεία. Στο αντικείμενο screen προστέθηκε το script Screen, το οποίο ελέγχει το αν έχουν μπει όλα τα αντικείμενα στη σωστή θέση (Εικόνα 7.7). Αυτό συμβαίνει όπως συνέβη με τον γρίφο του λαβυρίνθου και του κύβου, δηλαδή παράγονται κάποιες ακτίνες που περιμένουν να χτυπήσουν τον σωστό collider. Σε όλα τα αντικείμενα τροποποιήθηκε το συστατικό του collider τους ώστε να γίνει Convex και Is Trigger. Αυτό έγινε για να ανιχνεύονται σωστά από τις ακτίνες. Αφού χτυπηθεί το σωστό αντικείμενο ενεργοποιείται η αντίστοιχη Boolean μεταβλητή, και δεν μπορεί πια ο παίκτης να το μετακινήσει. Τέλος, ελέγχεται αν ενεργοποιήθηκαν και οι τέσσερις μεταβλητές, και τότε τα αντικείμενα παίρνουν την ίδια εμφάνιση με το δοχείο, τοποθετούνται μέσα στο Shell2 για να ομαδοποιηθούν, και ξεκινούν να γυρνάν όλα τα στοιχεία μαζί σε μία αργή ταχύτητα με το script Spin που έχει προστεθεί στο Shell2 (Εικόνα 7.8).

```
public class Screen : MonoBehaviour
{
    public GameObject hsp, hcu, hma, hci, hitsp, hitcu, hitma, hitci;
    public Vector3 rsp, rcu, rma, rci;
    public bool bsp, bcu, bma, bci;
    private UIManager uiman;
    private Transform tran;

    void Start()
    {
        uiman = GameObject.Find("UIManager").GetComponent<UIManager> ();
        //παίρνει το script UI Manager
        tran = GameObject.Find("Shell2").transform;
        //παίρνει το transform του Shell στο Image Target
    }

    void Update()
    {
        RaycastHit objectHit;
        //παράγεται μία ακτίνα από το κάθε υπό-δοχείο με μικρό μήκος
        Vector3 rsp = hsp.transform.TransformDirection(0f, 0f, 0.12f);
        Vector3 rcu = hcu.transform.TransformDirection(0f, 0f, 0.12f);
        Vector3 rma = hma.transform.TransformDirection(0f, 0f, 0.05f);
        Vector3 rci = hci.transform.TransformDirection(0f, 0f, 0.05f);

        //εμφάνιση των ακτινών για debugging
        Debug.DrawRay(hsp.transform.position, rsp, Color.green);
        Debug.DrawRay(hcu.transform.position, rcu, Color.green);
        Debug.DrawRay(hma.transform.position, rma, Color.green);
        Debug.DrawRay(hci.transform.position, rci, Color.green);

        //αν χτυπήσει κάτι η ακτίνα της σφαίρας
        if (Physics.Raycast(hsp.transform.position, rsp, out objectHit, 0.12f))
        {
            hitsp  = objectHit.collider.gameObject; //παίρνει το αντικείμενο
            if (hitsp.name == "sphere2"){       //αν είναι η σφαίρα
                hitsp.transform.localPosition = new Vector3(-0.126f, 0.7572f, 0.09f);
                //την τοποθετεί σε θέση που γεμίζει το κενό του μοντέλου
                bsp = true;         //ενεργοποιεί τη boolean μεταβλητή νίκης
                hitsp.tag = "Untagged";
                //θέτει το αντικείμενο να είναι τύπου "Untagged"
                //για να μην μπορεί να το ξανα μετακινήσει ο παίκτης
            }
            rsp = hsp.transform.TransformDirection(0, 0, 0);
            //απενεργοποιείται η ακτίνα για να ενεργοποιηθεί ξανά
        }
```

Εικόνα 7.7: Το πρώτο μέρος του script Screen

Ακολουθούν τρεις ίδιοι έλεγχοι και για τα άλλα τρία υπό-δοχεία, με τις μεταβλητές hitcu, hitma, hitci, rcu, rma, rci, bcu, bma, bci.

```
    if (bsp && bcu && bma && bci){  //αν έχουν ενεργοποιηθεί όλες οι μεταβλητές νίκης
        //το material των αντικειμένων αλλάζει ώστε να είναι ίδιο με του δοχείου
        hitsp.GetComponent<Renderer>().material = this.GetComponent<Renderer>().material;
        hitcu.GetComponent<Renderer>().material = this.GetComponent<Renderer>().material;
        hitma.GetComponent<Renderer>().material = this.GetComponent<Renderer>().material;
        hitci.GetComponent<Renderer>().material = this.GetComponent<Renderer>().material;
        //τα αντικείμενα μπαίνουν μέσα στο shell για να ομαδοποιηθούν
        hitsp.transform.SetParent(tran);
        hitcu.transform.SetParent(tran);
        hitma.transform.SetParent(tran);
        hitci.transform.SetParent(tran);
        //το shell ξεκινά να περιστρέφεται με το script Spin
        GameObject.Find("Shell2").GetComponent<Spin>().enabled = true;
        //απενεργοποιούνται οι boolean μεταβλητές για να μην τρέχει συνέχεια η μέθοδος
        bsp = bcu = bma = bci = false;
    }
}
```

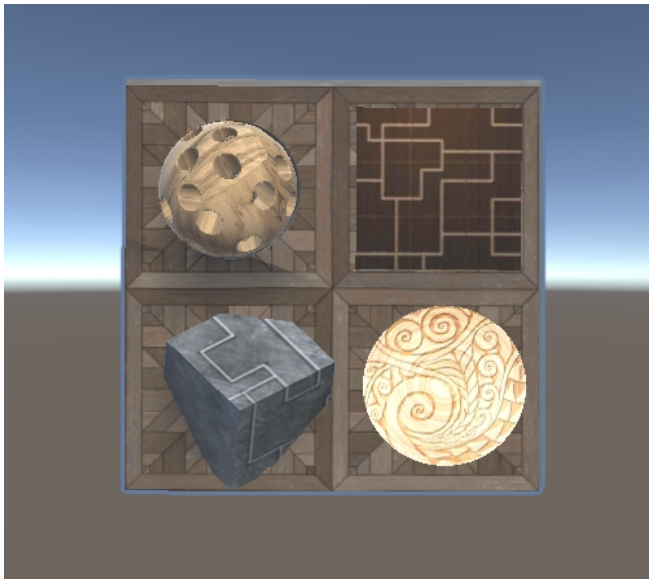
Εικόνα 7.8: Το δεύτερο μέρος του script Screen

Επιλέχθηκε μια εικόνα για το Image Target. Κατέβηκε από το διαδίκτυο, εκτυπώθηκε, ανέβηκε στη σελίδα, κατέβηκε σαν βάση δεδομένων, εισήχθη στο project με drag-and-drop και ανατέθηκε στο Image Target (Εικόνα 7.9).

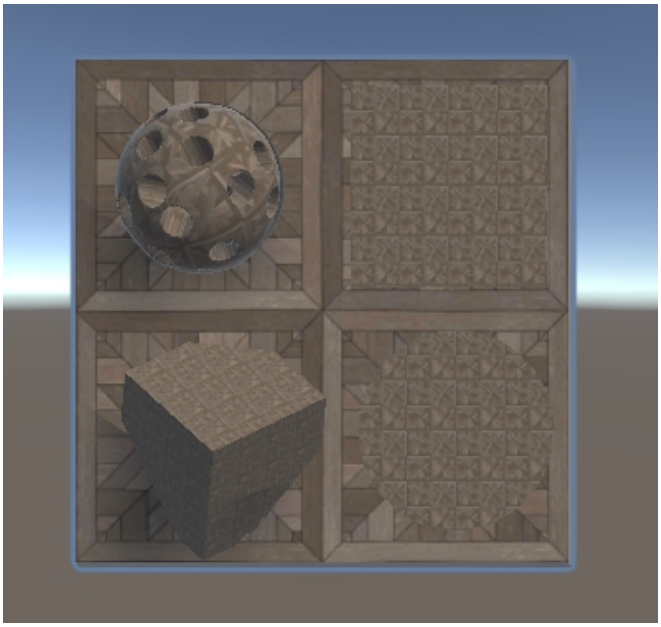
Εικόνα 7.9: Η εικόνα του Image Target για το τελικό επίπεδο

7.4 Επίλογος

Σε αυτό το σημείο έχει ολοκληρωθεί και το τελικό επίπεδο του παιχνιδιού. Σε αυτό το κεφάλαιο δεν εισήχθησαν νέες έννοιες ή μέθοδοι, αλλά προστέθηκαν επιπλέον λειτουργίες στο παιχνίδι για την εύρυθμη και καλαίσθητη ολοκλήρωσή του. Στις εικόνες 7.10 και 7.11 φαίνεται το τελικό αποτέλεσμα του τελικού επιπέδου.



Εικόνα 7.10: Πως ταιριάζουν τα αντικείμενα μέσα στο screen



Εικόνα 7.11: Το τελικό σχήμα του ολοκληρωμένου μοντέλου

Κεφάλαιο 8ο Συμπεράσματα και μελλοντικοί στόχοι

8.1 Συμπεράσματα

Το πρώτο συμπέρασμα που βγήκε από την ανάπτυξη αυτής της εφαρμογής είναι ότι χρειάζεται ενδελεχής μελέτη για την επιλογή της τεχνολογίας που θα χρησιμοποιηθεί σε μία εφαρμογή με AR τεχνολογίες. Ειδικά στην περιοχή των δωρεάν AR kits υπάρχουν αρκετές επιλογές, και αναλόγως των δυνατοτήτων που επιθυμούνται από τον δημιουργό για την εφαρμογή, διαφορετικά πακέτα καλύπτουν τις εκάστοτε ανάγκες.

Η ανάπτυξη παιχνιδιών με AR διαφέρει από αυτή των συμβατικών παιχνιδιών. Παρουσιάζει διαφορετικές δυσκολίες αλλά και ανάγκες. Ο παράγοντας του εξωτερικού περιβάλλοντος πρέπει να ληφθεί σοβαρά υπ' όψιν, μιας και αποτελεί το μισό παιχνίδι, σε αντίθεση με τα συμβατικά παιχνίδια που αποτελούνται πλήρως από γραφικά στοιχεία. Αυτό το παιχνίδι είναι φτιαγμένο για να παιχτεί σε ένα δωμάτιο με συγκεκριμένα χαρακτηριστικά, δηλαδή τον σωστό φωτισμό και τις εκτυπωμένες εικόνες στα σωστά σημεία.

Αυτό το project προσέφερε μια ανεπτυγμένη εμπειρία στην ανάπτυξη παιχνιδιών με την πλατφόρμα Unity. Συνηθίζεται σε τέτοιου είδους projects η συνεργασία μιας ολόκληρης ομάδας, όπου κάθε μέλος της είναι υπεύθυνο για ένα συγκεκριμένο κομμάτι της ανάπτυξης, ή ακόμα και για έναν από τους γρίφους. Το γεγονός ότι φτιάχτηκε ατομικά, παρουσίασε αρκετούς περιορισμούς, στο χρόνο ανάπτυξης, στις προγραμματιστικές δυνατότητες, αλλά και στην εξειδίκευση σε κομμάτια εκτός του προγραμματισμού. Επιπλέον, απαιτήθηκε σε μικρό χρονικό διάστημα η γέννηση δημιουργικής έμπνευσης, όσον αφορά τις ιδέες των γρίφων. Σε αυτό βοήθησε εν μέρει η ενασχόληση με παρόμοια παιχνίδια puzzle στο παρελθόν, αλλά και μια σύντομη επανεπίσκεψη σε κάποια από αυτά (The Room [24], Myst Riven [25], Myst Exile [25], etc)

Ο σκοπός ήταν να δημιουργηθεί ένα νέο παιχνίδι που να θυμίζει μεν παλαιότερα, μεγάλα παιχνίδια του είδους puzzle, αλλά να είναι αυτοδημιούργητο και πρωτότυπο. Γι αυτόν το λόγο έπρεπε αν όχι ολόκληρο, το μεγαλύτερο κομμάτι του να είναι καινούριο και να μην χρησιμοποιεί ξένα αντικείμενα και στοιχεία. Έτσι δεν θα μπορούσε παρά να χρειαστεί η δημιουργία νέων γραφικών. Η χρήση κάποιου προγράμματος σχεδίασης 3D γραφικών ήταν απαραίτητη, και αυτό προσέφερε μία κάποια εμπειρία και σε αυτόν τον τομέα. Όντας ένα ξένο αντικείμενο, τα προϊόντα αυτής της προσπάθειας αναγκαστικά παρουσιάζουν ένα πρώιμο στάδιο αυτής της γνώσης. Παρ' όλα αυτά, κρίθηκαν επαρκή για αυτό το στάδιο του project, αν όχι και για την έκθεσή του σε άτομα για play-testing. Play-testing είναι το στάδιο όπου ένα παιχνίδι διανέμεται σε ένα μικρό δείγμα ατόμων για μαζικό έλεγχο του παιχνιδιού και για να δημιουργηθούν εντυπώσεις προς βελτίωσή του (feedback).

Τέλος, το παιχνίδι έφερε σε επαφή δύο δημιουργούς για τον σχεδιασμό των εικόνων και υλικών και έδωσε την εμπειρία της συνεργασίας σε κάποιο project ανάπτυξης εφαρμογής με γραφικά. Χωρίς τον συνεργάτη γραφίστα αυτό το παιχνίδι θα χρησιμοποιούσε κατά προσέγγιση κάποιες εικόνες από το διαδίκτυο και αυτοσχέδια υλικά, πάλι με νέα εξοικείωση στον τομέα του σχεδίου, με αποτέλεσμα να μην ήταν τόσο καλαίσθητο αλλά και ίσως την μη ολοκλήρωση του παιχνιδιού σε αυτό το στάδιο, ίσως με λιγότερους γρίφους.

Καθολικά, ήταν μία πολύ έντονη, δημιουργική, πιεστική αλλά και πολύ ευχάριστη εμπειρία.

8.2 Μελλοντικοί στόχοι

Οι μελλοντικοί στόχοι για αυτό το παιχνίδι είναι σίγουρα η ανάπτυξη και προσθήκη επιπλέον γρίφων, η τελειοποίηση των γραφικών 3D μοντέλων, ίσως από κάποιον επαγγελματία, η έκθεσή του σε play-testing, και αργότερα η παράδοσή του σε κάποια εταιρία με escape rooms για τη διάθεσή του στο κοινό. Δεν είναι μέσα στα σχέδια να ανέβει η εφαρμογή στο Play store της Google για Android, διότι απαιτεί τη χρήση εξωτερικών αντικειμένων και δεν μπορεί να παιχτεί αυτόνομα σε κάποια συσκευή.

Η εφαρμογή πρέπει να περάσει από πολλά ακόμη στάδια για να θεωρηθεί μια επαγγελματική δουλειά, όπως να δοκιμαστούν τα στατιστικά της σε θέματα ταχύτητας (optimization) του κώδικα. Είναι σίγουρο πως η προσθήκη νέων γρίφων θα είναι μια πιο χρονοβόρα διαδικασία για να δοθεί εκτενέστερη προσοχή, μιας και το βασικό παιχνίδι έχει ολοκληρωθεί. Μετά το play-testing είναι σχεδόν σίγουρο πως θα έρθουν στην επιφάνεια νέα σημεία βελτίωσης του παιχνιδιού. Προσδοκάται αυτό το παιχνίδι να είναι το πρώτο πολλών άλλων που ίσως γίνουν στο μέλλον.

ΒΙΒΛΙΟΓΡΑΦΙΑ

[1] Nicholson, Simon (2015). Peking behind the Locked Door: A survey of Escape Room Facilities(PDF). Canada: Wilfrid Laurier University. [Προσπελάστηκε 16/12/20] http://scottnicholson.com/pubs/erfacwhite.pdf

[2] The Top 10 Video Game Engines Until December 2020 [Προσπελάστηκε 16/12/20] https://www.gamedesigning.org/career/video-game-engines/

[3] Unity Real-Time Development Platform https://unity.com/ [Προσπελάστηκε 16/12/20]

[4] Unity Build Settings https://docs.unity3d.com/Manual/BuildSettings.html [Προσπελάστηκε 16/12/20]

[5] Augmented Reality definition https://www.dictionary.com/browse/augmented-reality [Προσπελάστηκε 16/12/20]

[6] Blender 3D creation suite https://www.blender.org/ [Προσπελάστηκε 17/12/20]

[7] Λειτουργίες του EventSystem [Προσπελάστηκε 21/12/20] https://docs.unity3d.com/Packages/com.unity.ugui@1.0/manual/EventSystem.html

[8] Unity Canvas [Προσπελάστηκε 22/12/20] https://docs.unity3d.com/Packages/com.unity.ugui@1.0/manual/UICanvas.html

[9] Unity Prefab https://docs.unity3d.com/Manual/Prefabs.html [Προσπελάστηκε 30/12/20]

[10] Unity Mesh Filter https://docs.unity3d.com/Manual/class-MeshFilter.html [Προσπελάστηκε 30/12/20]

[11] Unity Mesh Renderer https://docs.unity3d.com/Manual/class-MeshRenderer.html [Προσπελάστηκε 30/12/20]

[12] Unity Box Collider https://docs.unity3d.com/Manual/class-BoxCollider.html [Προσπελάστηκε 30/12/20]

[13] Μέθοδος Update() [Προσπελάστηκε 31/12/20] https://docs.unity3d.com/2017.3/Documentation/ScriptReference/MonoBehaviour.Update.html

[14] Κλάση MonoBehavior [Προσπελάστηκε 31/12/20] https://docs.unity3d.com/2017.3/Documentation/ScriptReference/MonoBehaviour.html

[15] Βιβλιοθήκες - Using Namespaces https://docs.unity3d.com/Manual/Namespaces.html [Προσπελάστηκε 31/12/20]

[16] Η μέθοδος Physics.Raycast https://docs.unity3d.com/ScriptReference/Physics.Raycast.html [Προσπελάστηκε 02/01/21]

[17] Η μέθοδος Mesh.uv https://docs.unity3d.com/ScriptReference/Mesh-uv.html [Προσπελάστηκε 02/01/21]

[18] Blender Beginners Tutorial [Προσπελάστηκε 04/01/21] https://www.youtube.com/watch?v=ucs4E2VBUo4&lc=z234f3crcnabcbotracdp431gpcwgsmsz24butbyfplw03c010c

[19] Blender Tutorial: How to Make A Perfect Sphere Without Lines [Προσπελάστηκε 05/01/21] https://www.youtube.com/watch?v=CamNlztBhUk

[20] Rigidbody https://docs.unity3d.com/ScriptReference/Rigidbody.html [Προσπελάστηκε 05/01/21]

[21] Συστατικό Input Field [Προσπελάστηκε 05/01/21] https://docs.unity3d.com/Packages/com.unity.ugui@1.0/manual/script-InputField.html

[22] Το μοντέλο των 3D αριθμών που χρησιμοποιήθηκε https://www.turbosquid.com/3d-models/free-numbers-1-2-3d-model/266953 [Προσπελάστηκε 07/01/21]

[23] Η κλάση TouchScreenKeyboard [Προσπελάστηκε 07/01/21] https://docs.unity3d.com/ScriptReference/TouchScreenKeyboard.html

[24] The Room games https://en.wikipedia.org/wiki/The_Room_(video_game) [Προσπελάστηκε 07/01/21]

[25] Myst (series) games https://en.wikipedia.org/wiki/Myst_(series) [Προσπελάστηκε 07/01/21]

ΠΑΡΑΡΤΗΜΑ Α : ΒΙΝΤΕΟ

Στον παρακάτω σύνδεσμο είναι αναρτημένα κάποια βίντεο που παρουσιάζουν τον κάθε γρίφο του παιχνιδιού και το πως λύνεται.

https://www.dropbox.com/sh/o5mkn2r0vakzhjg/AADOMWFWDVKAG7M7B1BnotF_a?dl=0