ΕΘΝΙΚΟ ΜΕΤΣΟΒΙΟ ΠΟΛΥΤΕΧΝΕΙΟ

ΣΧΟΛΗ

ΗΛΕΚΤΡΟΛΟΓΩΝ ΜΗΧΑΝΙΚΩΝ ΚΑΙ ΜΗΧΑΝΙΚΩΝ Η/Υ

ΠΑΝΕΠΙΣΤΗΜΙΟ ΠΕΙΡΑΙΑ

ΤΜΗΜΑ

ΒΙΟΜΗΧΑΝΙΚΗΣ ΔΙΟΙΚΗΣΗΣ ΚΑΙ ΤΕΧΝΟΛΟΓΙΑΣ

ΔΠΜΣ

ΤΕΧΝΟ-ΟΙΚΟΝΟΜΙΚΑ ΣΥΣΤΗΜΑΤΑ

ΔΙΠΛΩΜΑΤΙΚΗ ΕΡΓΑΣΙΑ

Εφαρμογή εικονικής πραγματικότητας για διαδραστική φωτορεαλιστική απεικόνιση ενός μνημείου πολιτιστικής κληρονομιάς - Εκκλησία Άγιου Παύλου

Μεταπτυχιακός φοιτητής

Παναγιώτης Μυσίρης

Επιβλέπων καθηγητής

Αναστάσιος Δουλάμης

Αθήνα,

Φεβρουάριος 2018

ΠΡΟΛΟΓΟΣ

Η παρούσα διπλωματική εργασία εκπονήθηκε στα πλαίσια του Διαπανεπιστημιακού Προγράμματος Μεταπτυχιακών Σπουδών «Τεχνο-Οικονομικά Συστήματα» που προσφέρεται από τη Σχολή Ηλεκτρολόγων Μηχανικών και Μηχανικών Ηλεκτρονικών Υπολογιστών του Εθνικού Μετσόβιου Πολυτεχνείου και από το Τμήμα Βιομηχανικής Διοίκησης και Τεχνολογίας του Πανεπιστημίου Πειραιώς υπό την εποπτεία του καθηγητή κ. Αναστάσιου Δουλάμη, ο οποίος διατύπωσε το αντικείμενο σύμφωνα με τα ειδικά μου ενδιαφέροντα. Σε αυτό το σημείο, θα ήθελα να εκφράσω τις θερμές ευχαριστίες μου για την υπομονή του και τη συνεχή στήριξη και καθοδήγηση σε όλα τα στάδια της εργασίας. Θα ήθελα επίσης να ευχαριστήσω την οικογένεια μου για την υποστήριξη της.

ΠΕΡΙΕΧΟΜΕΝΑ

ΠΙΝΑΚΑΣ ΕΙΚΟΝΩΝ

ΠΙΝΑΚΑΣ ΛΙΣΤΩΝ

ΠΕΡΙΛΗΨΗ

Αυτή η εργασία αποσκοπεί στην ανάπτυξη μιας εφαρμογής μέσω της οποίας ο χρήστης θα μπορεί να πλοηγηθεί εικονικά σε ένα τρισδιάστατο (3D) μοντέλο χρησιμοποιώντας εισόδους (ερεθίσματα) από το πληκτρολόγιο και το ποντίκι. Επιπλέον, θα είναι δυνατό μέσω της διεπαφής χρήστη να παρατηρεί το μοντέλο σε διαφορετικές προβολές (οπτικές γωνίες), να το περιστρέφει και να εμφανίζει πληροφορίες σχετικά με αυτό. Η εφαρμογή αναπτύχθηκε χρησιμοποιώντας το πρόγραμμα UNITY, ένα από τα πιο δημοφιλή εργαλεία για την ανάπτυξη τρισδιάστατων ή δισδιάστατων εφαρμογών, κινούμενων σχεδίων και βίντεο.

Στο πρώτο κεφάλαιο συζητούμε για την πολιτιστική κληρονομιά, τη σημασία της, τους τύπους της και παρουσιάζουμε τη παγκόσμια δραστηριοποίηση για τη διατήρηση της. Τέλος, παραθέτουμε κάποιες γενικές πληροφορίες για το χρησιμοποιούμενο 3D μοντέλο.

Στο δεύτερο κεφάλαιο παρουσιάζουμε το Unity, ένα ολοκληρωμένο περιβάλλον ανάπτυξης εφαρμογών, ικανό να χρησιμοποιηθεί σε διαφορετικά υπολογιστικά συστήματα ή με διαφορετικά πακέτα λογισμικού (cross platform integrated development environment). Το κεφάλαιο αυτό πραγματεύεται γιατί το Unity είναι μια εξαιρετική επιλογή, τη λειτουργία του επεξεργαστή, το βασικό σύστημα συνιστωσών (component system) και τον προγραμματισμό στο Unity.

Στο τρίτο κεφάλαιο, εξετάζουμε θέματα όπως οι είσοδοι-ερεθίσματα μέσω του ποντικιού και του πληκτρολογίου (mouse and keyboard inputs). Επιπλέον, εξηγούνται με μεγάλη λεπτομέρεια ο καθορισμός και ο χειρισμός τρισδιάστατων θέσεων και περιστροφών (3D positions and rotations).

Στο τέταρτο κεφάλαιο παρουσιάζουμε τη λειτουργία της Γραφικής Διεπαφής Χρήστη (Graphical User Interface) στο Unity. Κάθε εφαρμογή χρειάζεται ένα περιβάλλον διεπαφής χρήστη και οι τελευταίες εκδόσεις του Unity διαθέτουν ένα βελτιωμένο σύστημα για τη δημιουργία διεπαφών.

Στο πέμπτο κεφάλαιο παρουσιάζονται δύο βοηθητικά στοιχεία, ένα παράθυρο εκκίνησης (start window) και ένα παράθυρο καθοδήγησης (instruction window). Το πρώτο έχει ένα κύριο μενού και το δεύτερο, όπως υποδηλώνει και το όνομα του, εξηγεί στο χρήστη πώς μπορεί να χειριστεί την εφαρμογή.

ABSTRACT

This work aims to develop an application via the user can virtually navigate to a three-dimensional (3D) model using inputs from the keyboard and the mouse. In addition, it is possible through the user interface to observe the model from different views, rotate and look information about this. The application was developed using the UNITY program, one of the most popular tool for developing 3D or 2D applications, animations and videos.

In the first chapter we discuss for the cultural heritage, its importance, the type of heritage and we present the world heritage movement. Then, we refer to the used 3D model.

In the second chapter we present Unity, the cross-platform integrated development environment. This chapter covers what makes Unity a great choice, operating the Unity editor, the fundamental component system and programming in Unity.

In the third chapter we cover topics like mouse and keyboard inputs. Defining and manipulating both 3D positions and rotations are thoroughly explained.

In the fourth chapter we present the latest Graphical User Interface functionality in Unity. Every application needs a UI, and the latest versions of Unity feature an improved system for creating user interfaces.

In the fifth chapter we present two auxiliary elements, a start window and an instruction window. The first have a main menu and the second, as its name implies, explains to user how to handle the model's functionality.

ΚΕΦΑΛΑΙΟ 1

Η ΠΟΛΙΤΙΣΤΙΚΗ ΚΛΗΡΟΝΟΜΙΑ

ΚΕΦΑΛΑΙΟ 1

Η ΠΟΛΙΤΙΣΤΙΚΗ ΚΛΗΡΟΝΟΜΙΑ

1.1 Ορισμός

Πολιτιστική κληρονομιά (Εθνική κληρονομιά ή απλά Κληρονομιά) είναι το κληροδότημα από φυσικά αντικείμενα (πολιτιστική ιδιοκτησία-περιουσία) και από άυλα χαρακτηριστικά μιας ομάδας ή κοινωνίας, τα οποία έχουν κληροδοτηθεί από τις παλαιότερες γενιές και διατηρούνται στο παρόν ενώ παράλληλα παραχωρούνται στο μέλλον για να επωφεληθούν οι επόμενες γενιές. Ο όρος Πολιτιστική κληρονομιά περιλαμβάνει τον απτό πολιτισμό (όπως κτίρια, μνημεία, τοπία, βιβλία, έργα τέχνης και τεκμήρια), τον άυλο πολιτισμό (όπως τη λαογραφία, τις παραδόσεις, τη γλώσσα και τη γνώση) και τη φυσική κληρονομιά, που περιλαμβάνει σημαντικά πολιτιστικά τοπία και τη βιοποικιλότητα. Η σκόπιμη ενέργεια της διατήρησης της πολιτιστικής κληρονομιάς από το παρόν για το μέλλον, είναι γνωστή ως Διατήρηση (Preservation) ή συντήρηση (Conservation), αν και αυτοί οι όροι μπορεί να έχουν πιο συγκεκριμένο ή τεχνικό νόημα στα ίδια γενικά πλαίσια μιας άλλης διαλέκτου.

Η πολιτιστική κληρονομιά είναι μοναδική και αναντικατάστατη, γεγονός το οποίο φορτίζει τον σύγχρονο πολιτισμό με την ευθύνη της διατήρησης και διαφύλαξης της. Μικρότερα αντικείμενα όπως έργα τέχνης και άλλα πολιτιστικά αριστουργήματα συλλέγονται από μουσεία και γκαλερί. Στα πλαίσια αυτής της διαφύλαξης του πολιτιστικού αποθέματος, έγιναν επιτυχείς προσπάθειες για τη διατήρηση της μελλοντικής κληρονομιάς πολλών εθνών. Οργανισμοί και πολιτιστικές ομάδες, όπως ο διεθνής οργανισμός της UNESCO συνέβαλαν με επιτυχία στο εν λόγω έργο.

1.2 Η δεοντολογία και οι αρχές της πολιτιστικής διατήρησης

Τα αντικείμενα αποτελούν κλάδο της μελέτης της ανθρώπινης ιστορίας. Ένα απλό αντικείμενο μπορεί να παρέχει μια συγκεκριμένη αρχή ιδεών και ταυτόχρονα με την ύπαρξη του να την επικυρώνει. Μέσω της διατήρησης των αντικειμένων καθίσταται ευκολότερη η κατανόηση παρελθόντων πολιτισμών και η μετάδοση της ιστορίας τους στις επόμενες γενεές. Στο βιβλίο «Το

παρελθόν είναι μια ξένη χώρα», ο David Lowenthal παρατηρεί ότι τα διαφυλαγμένα αντικείμενα επικυρώνουν αναμνήσεις. Οι ψηφιακές τεχνικές απόκτησης μπορούν να παρέχουν μια τεχνολογική λύση, στα πλαίσια της οποίας, το αντικείμενο αποκτά το αντίστοιχο σχήμα και την εμφάνιση των πρωτότυπων αριστουργημάτων με πρωτοφανή ακρίβεια στην εξέλιξη της ανθρώπινης ιστορίας, η επικαιρότητα όμως του αντικειμένου, σε αντίθεση με την αναπαραγωγή του, ενσωματώνει τον άνθρωπο μέσα σε αυτό, δίνοντας του ένα κυριολεκτικό μέσο άμεσης επαφής με το παρελθόν. Αυτή η σχέση όμως που αναπτύσσεται ανάμεσα σε άνθρωπο και παρελθόν ελλοχεύει κινδύνους, αφού μέρη και αντικείμενα φθείρονται από τα χέρια των τουριστών, από το φως που απαιτείται για την έκθεση τους, καθώς και άλλους κινδύνους οι οποίοι συνοδεύουν ένα γνωστό και εκτιθέμενο αντικείμενο. Την συνειδητοποίηση αυτού του κινδύνου ενισχύει το γεγονός, ότι όλα τα προς προβολή αντικείμενα βρίσκονται σε μία αδιάκοπη διαδικασία χημικού μετασχηματισμού, που καταλήγει σε ένα αντίθετο από το επιθυμητό αποτέλεσμα. Η πορεία της διατήρησης του έργου ανακόπτεται από μια εμφανή αλλαγή σε αυτό. Δεν είναι ποτέ αυτό που ήταν κάποτε. Αντιστοίχως η αλλαγή αυτή αποτελεί τη δυνητική αξία που μπορεί η κάθε σύγχρονη γενιά να τοποθετήσει στο παρελθόν ή στα αντικείμενα που τη συνδέουν με αυτό.

Οι κλασικοί πολιτισμοί αποδίδουν υπέρτατη αξία στη διατήρηση της παράδοσης. Στα πλαίσια αυτής της πεποίθησης, οι κοινωνικοί θεσμοί, η επιστημονική γνώση και οι τεχνολογικές εφαρμογές πρέπει να χρησιμοποιούν την κληρονομιά τους ως πόρο-μέσο.

Μιλώντας σε μία πιο σύγχρονη γλώσσα, θα μπορούσαμε να πούμε ότι θεωρούν ως κοινωνικούς πόρους, τα οικονομικά προνόμια (όπως τους φυσικούς πόρους και την ακίνητη περιουσία) αλλά και τους παράγοντες που προωθούν την κοινωνική ολοκλήρωση (όπως θεσμούς για τη διατήρηση της γνώσης και για την συντήρηση της κοινωνικής ιεραρχίας και τάξης). Βάση των κανόνων της Ηθικής, ότι έχει κληροδοτηθεί δεν πρέπει να καταναλώνεται, αντιθέτως οφείλει να κληροδοτηθεί ξανά, πιθανά εμπλουτισμένο σε διαδοχικές γενιές.

1.3 Τύποι πολιτιστικής κληρονομιάς

Πολιτιστική ιδιοκτησία

Η πολιτιστική ιδιοκτησία περιλαμβάνει τη φυσική ή απτή πολιτιστική κληρονομιά, όπως έργα τέχνης. Αυτή γενικά χωρίζεται σε δύο ομάδες τη κινητή και ακίνητη κληρονομιά. Η ακίνητη κληρονομιά περιλαμβάνει κτίρια (τα οποία μπορεί να περιλαμβάνουν εγκατεστημένη τέχνη όπως όργανα, βιτρό και τοιχογραφίες), μεγάλες βιομηχανικές εγκαταστάσεις ή άλλους ιστορικούς τόπους και μνημεία. Η κινητή κληρονομιά περιλαμβάνει βιβλία, έγγραφα, κινητά έργα τέχνης, μηχανές, ρούχα και άλλα αντικείμενα που θεωρούνται άξια συντήρησης για το μέλλον. Αυτά περιλαμβάνουν αντικείμενα σημαντικά για την αρχαιολογία, την αρχιτεκτονική, την επιστήμη και την τεχνολογία.

Άυλος πολιτισμός

Η άυλη πολιτιστική κληρονομιά αποτελείται από μη φυσικές πτυχές μιας συγκεκριμένης κουλτούρας, η οποία συντηρείται συχνότερα από τα κοινωνικά έθιμα κατά τη διάρκεια μιας συγκεκριμένης περιόδου της ιστορίας. Η έννοια περιλαμβάνει τους τρόπους και τα μέσα συμπεριφοράς σε μια κοινωνία και τους συχνά τυπικούς κανόνες για τη λειτουργία σε ένα συγκεκριμένο πολιτισμικό περιβάλλον. Αυτές περιλαμβάνουν κοινωνικές αξίες και παραδόσεις, ήθη και έθιμα, αισθητικές και πνευματικές πεποιθήσεις, καλλιτεχνική έκφραση, γλώσσα και άλλες βασικές πτυχές της ανθρώπινης δραστηριότητας. Η σημασία (αξία) των φυσικών αντικειμένων μπορεί να ερμηνευτεί με βάση τις κοινωνικοοικονομικές, πολιτικές, θρησκευτικές, εθνικές και φιλοσοφικές αξίες μιας συγκεκριμένης ομάδας ανθρώπων. Φυσικά, η άυλη πολιτιστική κληρονομιά γίνεται πιο δύσκολα αντιληπτή από τα φυσικά αντικείμενα.

Φυσική κληρονομιά

Η φυσική κληρονομιά αποτελεί επίσης σημαντικό μέρος της κληρονομιάς της κοινωνίας, που περιλαμβάνει την ύπαιθρο και το φυσικό περιβάλλον, συμπεριλαμβανομένης της χλωρίδας και της πανίδας, επιστημονικά γνωστή ως βιοποικιλότητα, καθώς και γεωλογικά στοιχεία (συμπεριλαμβανομένων

των ορυκτολογικών, γεωμορφολογικών, παλαιοντολογικών κ.λπ.). Αυτού του είδους οι τοποθεσίες πολιτιστικής κληρονομιάς χρησιμεύουν συχνά ως σημαντικό στοιχείο στην τουριστική βιομηχανία μιας χώρας, προσελκύοντας πολλούς επισκέπτες τόσο από το εξωτερικό όσο και από την περιοχή. Η κληρονομιά μπορεί επίσης να περιλαμβάνει πολιτιστικά τοπία (φυσικά χαρακτηριστικά που μπορεί να έχουν πολιτιστικές ιδιότητες).

1.4 Η παγκόσμια δραστηριοποίηση για τη διαφύλαξη της

Σημαντική ήταν η Σύμβαση για την Προστασία της Παγκόσμιας Πολιτιστικής και Φυσικής Κληρονομιάς που εγκρίθηκε από τη Γενική Διάσκεψη της UNESCO το 1972. Από το 2011, υπάρχουν 936 περιοχές παγκόσμιας πολιτιστικής κληρονομιάς: 725 πολιτιστικά, 183 φυσικά και 28 μικτά ακίνητα σε 153 χώρες . Κάθε μία από αυτές τις τοποθεσίες θεωρείται σημαντική για τη διεθνή κοινότητα.

Η υποθαλάσσια πολιτιστική κληρονομιά προστατεύεται από τη Σύμβαση της UNESCO για την προστασία της υποβρύχιας πολιτιστικής κληρονομιάς. Η σύμβαση αυτή είναι ένα νομικό μέσο που βοηθά τα συμβαλλόμενα μέρη να βελτιώσουν την προστασία της υποθαλάσσιας πολιτιστικής κληρονομιάς.

Επιπλέον, η UNESCO άρχισε να ορίζει αριστουργήματα της Προφορικής και Άυλης Κληρονομιάς. Η επιτροπή Οικονομικών, Κοινωνικών και Πολιτιστικών Δικαιωμάτων, που συστάθηκε στο πλαίσιο του Οικονομικού και Κοινωνικού Συμβουλίου των Ηνωμένων Εθνών με το άρθρο 15 του Συμφώνου της, προσπάθησε να ενσταλάξει τις αρχές βάσει των οποίων προστατεύεται η πολιτιστική κληρονομιά ως μέρος ενός βασικού ανθρώπινου δικαιώματος.

1.5 Το χρησιμοποιούμενο 3D μοντέλο

Το τρισδιάστατο μοντέλο είναι η εκκλησία του Αγίου Παύλου στο Μιραμπέλλο της Ιταλίας, που οικοδομήθηκε το 1795-1804, κατεδαφίστηκε το 1929, ανακατασκευάστηκε το 1943 και κατέρρευσε εν μέρει λόγω του σεισμού της 20ης Μαΐου 2012 που έπληξε την περιοχή Emilia-Romagna της Ιταλίας.

Για τη μοντελοποίηση χρησιμοποιήθηκαν εικόνες UAV της εκκλησίας. Το σύνολο δεδομένων εικόνων που απεικονίζουν την εκκλησία του Αγίου

Παύλου στο Μιραμπέλλο αποκτήθηκε χρησιμοποιώντας ένα UAV VTOL (κάθετη απογείωση και προσγείωση) μετά τον σεισμό του 2012 στη Βόρεια Ιταλία. Αυτό αποτελείται από 131 εικόνες χαμηλού ύψους από διάφορες οπτικές γωνίες και 79 κυρίως κατακόρυφες εναέριες εικόνες, που λήφθηκαν από ψηφιακή φωτογραφική μηχανή Olympus E-P2 με αισθητήρα 12.3 megapixel (4032 x 3024 pixels), που αντιστοιχεί σε εστιακή απόσταση 17 mm. Το γεγονός ότι ο προγραμματισμός των πτήσεων και ο σχεδιασμός της λήψης εικόνων εκτελούνται γρήγορα και υπό πίεση σε τέτοιες καταστάσεις μπορεί να οδηγήσουν σε ατελή σύνολα δεδομένων. Το συγκεκριμένο σύνολο δεδομένων στερείται πληρότητας, καθώς δεν περιλαμβάνει λοξές εικόνες για να παρέχει μια σύνδεση μεταξύ των εικόνων χαμηλού ύψους (σχεδόν οριζόντιας εικόνας) και κάθετης εικόνας. Επίσης, οι εικόνες δεν διαθέτουν πληροφορίες GPS/INS. Ως εκ τούτου, το παραγόμενο μοντέλο δεν είναι γεωαναφερμένο.

Το λογισμικό Agisoft Photoscan χρησιμοποιήθηκε για την τρισδιάστατη μοντελοποίηση της κατεστραμμένης εκκλησίας του Αγίου Παύλου. Πρόκειται για ένα αυτόνομο εμπορικό λογισμικό που παράγει τρισδιάστατα μοντέλα από εικόνες. Πρόσφατη έρευνα που επικεντρώθηκε στην αξιολόγηση του εμπορικού και ανοιχτού λογισμικού για τρισδιάστατη μοντελοποίηση καταστροφών έδειξε ότι οδηγεί σε ικανοποιητικά αποτελέσματα σε μια μικρή περίοδο. Βασίζεται στις τεχνικές SfM και DIM για να δημιουργήσει ένα πυκνό σύννεφο σημείων της σκηνής και εφαρμόζει αλγόριθμους αντιστοίχισης και σύστασης υφής για τη δημιουργία ενός πλήρους φωτορεαλιστικού 3D μοντέλου επιφάνειας.

Το μοντέλο τρισδιάστατου πλέγματος του κατεστραμμένου χώρου Πολιτιστικής Κληρονομιάς θα χρησιμοποιηθεί για τη δημιουργία εφαρμογής εικονικής πραγματικότητας. Η εύκολη πρόσβαση των χρηστών σε ένα τέτοιο μοντέλο θα μπορούσε να τους ενημερώσει με ρεαλιστικό τρόπο σχετικά με το μέγεθος και την έκταση της καταστροφής. Ως εκ τούτου, κάτι τέτοιο δεν μπορούμε να το αγνοήσουμε στη σημερινή εποχή των πολυμέσων και της απεικόνισης.

ΚΕΦΑΛΑΙΟ 2

ΕΙΣΑΓΩΓΗ ΣΤΟ UNITY

ΚΕΦΑΛΑΙΟ 2

ΕΙΣΑΓΩΓΗ ΣΤΟ UNITY

2.1 Γενικά

Ο τομέας της πληροφορικής είναι καινούργιος σε σχέση με άλλες επιστήμες και ο ρυθμός των καινοτομιών και των εξελίξεων είναι ταχύτατα αυξανόμενος. Διαρκώς ένα νέο είδος τεχνολογίας ανακαλύπτεται που επηρεάζει όχι μόνο το χώρο της πληροφορικής, αλλά επεκτείνεται στον επιχειρηματικό τομέα και στην καθημερινή ζωή. Ένα τέτοιο είδος τεχνολογίας είναι οι εφαρμογές για PC, Smartphones ή Tablets, οι οποίες αξίζουν ιδιαίτερης προσοχής. Αυτές σε συνδυασμό με την εξελισσόμενη τεχνολογία του Διαδικτύου και την ανάπτυξη των ηλεκτρονικών καταστημάτων αποτελούν ένα πολύ πρόσφορο έδαφος για έρευνα και ανάπτυξη. Σήμερα, οι προγραμματιστές διαθέτουν μια μεγάλη δεξαμενή πληροφοριών και ισχυρά εργαλεία ανάπτυξης εφαρμογών. Ένα τέτοιο εργαλείο είναι το Unity, το οποίο χρησιμοποιείται στην παρούσα εργασία.

2.2 Γιατί το Unity είναι τόσο σημαντικό;

Το Unity είναι μια μηχανή εφαρμογών επαγγελματικής ποιότητας που χρησιμοποιείται για τη δημιουργία εφαρμογών που στοχεύουν σε διάφορες πλατφόρμες. Μια τέτοια μηχανή παρέχει μια πληθώρα χαρακτηριστικών που είναι χρήσιμα σε πολλές διαφορετικές εφαρμογές, οπότε μια εφαρμογή που υλοποιείται χρησιμοποιώντας αυτή λαμβάνει όλα αυτά τα χαρακτηριστικά, και στα οποία προστίθενται προσαρμοσμένα στοιχεία τέχνης (art assets) και κώδικας εφαρμογής (application code). Το Unity έχει φυσική προσομοίωση (physics simulation), κανονικούς χάρτες (normal maps), screen space ambient occlusion (SSAO), δυναμικές σκιές (dynamic shadows) και ο κατάλογος συνεχίζεται. Πολλές μηχανές διαθέτουν τέτοιες δυνατότητες, αλλά το Unity έχει δύο βασικά πλεονεκτήματα σε σχέση με άλλα πρωτοποριακά εργαλεία ανάπτυξης: μια εξαιρετικά παραγωγική οπτική ροή εργασίας (visual workflow) και ένα υψηλό επίπεδο υποστήριξης μεταξύ πλατφόρμων (cross-platform support).

2.2.1 Πλεονεκτήματα του Unity

Η οπτική ροή εργασίας (visual workflow) είναι ένας μοναδικός σχεδιασμός, διαφορετικός από τα περισσότερα περιβάλλοντα ανάπτυξης. Ενώ τα άλλα εργαλεία ανάπτυξης είναι συχνά μια πολύπλοκη σειρά από διαφορετικά κομμάτια που πρέπει να επικοινωνήσουν ή ίσως μια βιβλιοθήκη προγραμματισμού που απαιτεί από εμάς να δημιουργήσουμε το δικό μας ολοκληρωμένο περιβάλλον ανάπτυξης (IDE), την αλυσίδα σχεδιασμού και οτιδήποτε άλλο, η ροή εργασιών ανάπτυξης στο Unity συνδέεται με ένα εκλεπτυσμένο επεξεργαστή (visual editor). Ο επεξεργαστής χρησιμοποιείται για το σχεδιασμό των σκηνών στην εφαρμογή και για να συνδέσει μαζί τα στοιχεία της τέχνης (art assets) και τον κώδικα σε διαδραστικά αντικείμενα. Η αξία του επεξεργαστή είναι ότι επιτρέπει την γρήγορη και αποτελεσματική κατασκευή εφαρμογών επαγγελματικής ποιότητας, δίνοντας στους προγραμματιστές εργαλεία που είναι απίστευτα παραγωγικά, ενώ συγχρόνως χρησιμοποιεί μια εκτεταμένη λίστα των τελευταίων τεχνολογιών στον τομέα της ψηφιακής επεξεργασίας.

Ο επεξεργαστής είναι ιδιαίτερα χρήσιμος για να κάνει ταχεία επανάληψη, με σκοπό την τελειοποίηση της εφαρμογής μέσω κύκλων πρωτοτύπων και δοκιμών. Μπορούμε να προσαρμόσουμε τα αντικείμενα στον επεξεργαστή και να μετακινήσουμε τα αντικείμενα ακόμα και όταν η εφαρμογή τρέχει. Επιπλέον, το Unity μας επιτρέπει να προσαρμόζουμε τον ίδιο τον επεξεργαστή γράφοντας scripts που προσθέτουν νέες λειτουργίες και μενού στη διεπαφή (interface).

Εκτός από τα σημαντικά πλεονεκτήματα παραγωγικότητας του επεξεργαστή, το άλλο βασικό πλεονέκτημα της εργαλειοθήκης του Unity είναι ο υψηλός βαθμός υποστήριξης μεταξύ πλατφόρμων (cross-platform support). Όχι μόνο το Unity είναι μια πολλαπλή πλατφόρμα όσον αφορά τους στόχους ανάπτυξης - deployment targets (μπορούμε να αναπτύξουμε για υπολογιστή, ιστό, κινητά ή και κονσόλες), αλλά είναι πολλαπλή πλατφόρμα από την άποψη των εργαλείων ανάπτυξης - development tools (μπορούμε να αναπτύξουμε την εφαρμογή σε Windows ή Mac OS). Αυτή η δυνατότητα της πλατφόρμας οφείλεται σε μεγάλο βαθμό στο γεγονός ότι το Unity ξεκίνησε ως λογισμικό μόνο για Mac και αργότερα μεταφέρθηκε στα Windows. Η

πρώτη έκδοση ξεκίνησε το 2005, αλλά τώρα το Unity είναι στην πέμπτη κύρια έκδοση (με πολλές μικρές ενημερώσεις που κυκλοφορούν συχνά). Αρχικά, το Unity υποστήριζε μόνο Mac για ανάπτυξη (developing και deployment), αλλά μέσα σε λίγους μήνες το Unity ενημερώθηκε για να δουλεύει εξίσου καλά και με Windows. Οι διαδοχικές εκδόσεις πρόσθεταν σταδιακά νέες πλατφόρμες ανάπτυξης (deployment platforms), όπως τη cross-platform web player το 2006, το iPhone το 2008, το Android το 2010, ακόμα και κονσόλες παιχνιδιών όπως το Xbox και το PlayStation. Πρόσφατα έχουν προσθέσει τη δυνατότητα ανάπτυξης σε WebGL, το νέο πλαίσιο για 3D γραφικά σε προγράμματα περιήγησης ιστού. Λίγες μηχανές υποστηρίζουν τόσους στόχους ανάπτυξης (deployment targets) όπως το Unity και καμιά δεν κάνει την ανάπτυξη σε πολλαπλές πλατφόρμες τόσο απλή.

Εν τω μεταξύ, εκτός από αυτά τα κύρια πλεονεκτήματα, ένα τρίτο και μικρότερο όφελος προέρχεται από το αρθρωτό σύστημα συνιστωσών (modular component system) που χρησιμοποιείται για την κατασκευή αντικειμένων (objects). Σε ένα σύστημα συνιστωσών, τα συστατικά (components) είναι πακέτα συνδυασμού και αντιστοίχισης λειτουργιών και τα αντικείμενα δημιουργούνται ως μια συλλογή συστατικών, και όχι ως μια αυστηρή ιεραρχία κλάσεων. Με άλλα λόγια, ένα σύστημα συνιστωσών είναι μια διαφορετική και πιο ευέλικτη προσέγγιση για αντικειμενοστραφή προγραμματισμό (object-oriented programming), όπου τα αντικείμενα (objects) κατασκευάζονται μέσω της σύνθεσης και όχι της κληρονομιάς (inheritance).

Σε ένα σύστημα συνιστωσών, τα αντικείμενα υπάρχουν σε μια επίπεδη ιεραρχία και διαφορετικά αντικείμενα έχουν διαφορετικές συλλογές συστατικών, αντί για μια δομή κληρονομιάς όπου διαφορετικά αντικείμενα βρίσκονται σε εντελώς διαφορετικούς κλάδους του δέντρου. Αυτή η διάταξη διευκολύνει την γρήγορη δημιουργία πρωτοτύπων, διότι μπορούμε γρήγορα να αναμειγνύουμε και να ταιριάζουμε διαφορετικά συστατικά αντί να χρειάζεται να αναδιαμορφώνουμε την αλυσίδα κληρονομίας όταν αλλάζουν τα αντικείμενα.

Παρόλο που θα μπορούσαμε να γράψουμε κώδικα για την υλοποίηση ενός προσαρμοσμένου συστήματος συνιστωσών εάν δεν υπήρχε, το Unity διαθέτει

ήδη ένα ισχυρό σύστημα συνιστωσών και αυτό το σύστημα ενσωματώνεται χωρίς προβλήματα στον επεξεργαστή (visual editor). Αντί να μπορούμε μόνο να χειριζόμαστε τα συστατικά σε κώδικα, μπορούμε να προσθέτουμε και να αφαιρούμε αυτά μέσα στον επεξεργαστή. Εν τω μεταξύ, δεν περιοριζόμαστε μόνο στην κατασκευή αντικειμένων μέσω σύνθεσης. Εξακολουθούμε να έχουμε τη δυνατότητα χρησιμοποίησης κληρονομικότητας στον κώδικα μας, συμπεριλαμβανομένων όλων των σχεδιαστικών μοτίβων βέλτιστης πρακτικής που προέρχονται μέσω αυτής.

2.2.2 Μειονεκτήματα του Unity

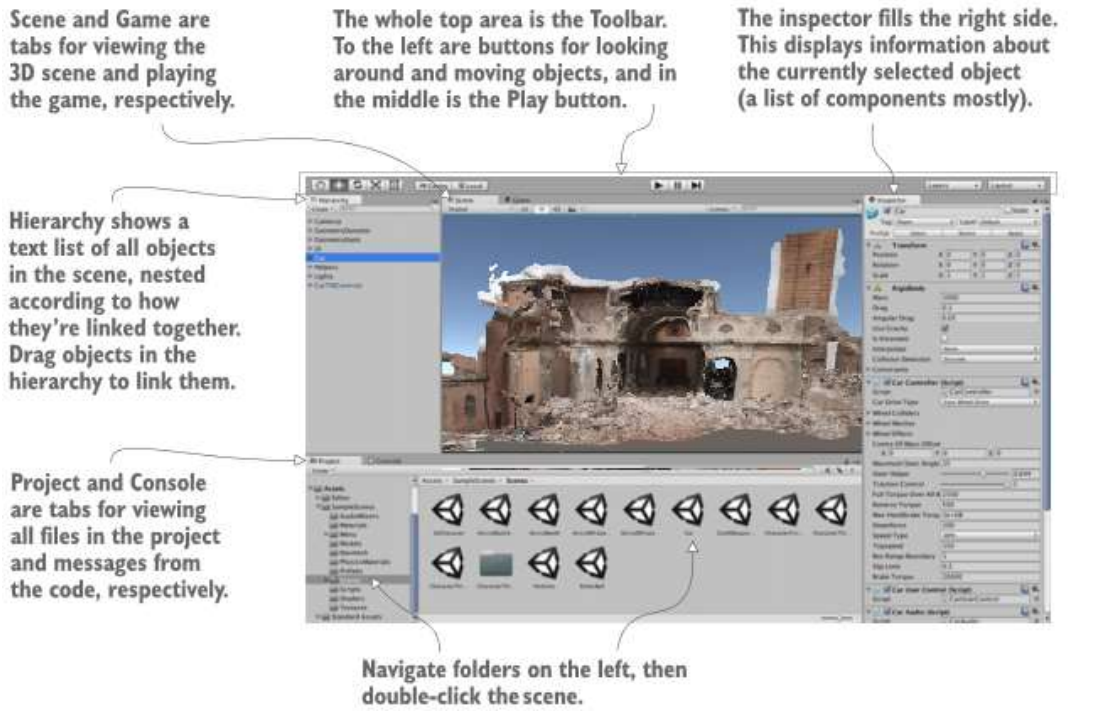
Το Unity έχει πολλά πλεονεκτήματα που το καθιστούν μια εξαιρετική επιλογή για την ανάπτυξη εφαρμογών και το συνιστώ ανεπιφύλακτα, αλλά θα ήμουν αμελής εάν δεν ανέφερα τις αδυναμίες του. Ο συνδυασμός του οπτικού επεξεργαστή και της εξελιγμένης κωδικοποίησης, αν και είναι πολύ αποτελεσματικό με το σύστημα συνιστωσών του Unity, δεν είναι συνηθισμένο και μπορεί να δημιουργήσει δυσκολίες. Σε σύνθετες σκηνές, μπορούμε να χάσουμε την εποπτεία, δηλαδή σε ποιο αντικείμενο στη σκηνή έχουν προσαρτηθεί συγκεκριμένα συστατικά (components). Το Unity παρέχει δυνατότητα αναζήτησης συνημμένων scripts (attached scripts), αλλά η αναζήτηση αυτή θα μπορούσε να είναι πιο ισχυρή. Καλούμαστε μερικές φορές να αντιμετωπίσουμε καταστάσεις όπου πρέπει να επιθεωρήσουμε χειροκίνητα τα πάντα στη σκηνή για να βρούμε συνδέσεις scripts (script linkages). Αυτό δεν συμβαίνει συχνά, αλλά όταν συμβαίνει μπορεί να είναι κουραστικό.

Ένα άλλο μειονέκτημα που μπορεί να προκαλέσει έκπληξη και απογοήτευση σε έμπειρους προγραμματιστές είναι ότι το Unity δεν υποστηρίζει τη σύνδεση με εξωτερικές βιβλιοθήκες κώδικα. Οι πολλές διαθέσιμες βιβλιοθήκες πρέπει να αντιγραφούν χειροκίνητα σε κάθε έργο όπου θα χρησιμοποιηθούν, αφού δεν υπάρχει δυνατότητα αναφοράς σε μια κεντρική κοινόχρηστη τοποθεσία. Η έλλειψη κεντρικής θέσης για τις βιβλιοθήκες μπορεί να δυσχεράνει την ανταλλαγή λειτουργικότητας μεταξύ διαφόρων έργων.

2.3 Πως χρησιμοποιείται το Unity

Η προηγούμενη ενότητα αναφέρθηκε στα οφέλη από την παραγωγικότητα του οπτικού επεξεργαστή του Unity, οπότε ας δούμε πως φαίνεται η διεπαφή και πώς λειτουργεί. Η διεπαφή στο Unity χωρίζεται σε διάφορες ενότητες: Scene view, Game view, Toolbar, Hierarchy view, Inspector, Project view και Console view. Κάθε τμήμα έχει διαφορετικό σκοπό, αλλά όλα είναι κρίσιμα για τον κύκλο δημιουργίας της εφαρμογής:

1. Μπορούμε να περιηγηθούμε σε όλα τα αρχεία στην καρτέλα Project view
2. Μπορούμε να τοποθετήσουμε αντικείμενα στη 3D σκηνή χρησιμοποιώντας την καρτέλα Scene view
3. Η γραμμή εργαλείων Toolbar διαθέτει εργαλεία για το σχεδιασμό της σκηνής
4. Μπορούμε να σύρουμε και να δημιουργήσουμε σχέσεις αντικειμένων (object relationships) στην καρτέλα Hierarchy view
5. Η καρτέλα Inspector παραθέτει όλες τις πληροφορίες σχετικά με το επιλεγμένο αντικείμενο, συμπεριλαμβανομένου του συνδεδεμένου κώδικα
6. Μπορούμε να δοκιμάσουμε να τρέξουμε την εφαρμογή ενώ βλέπουμε τα σφάλματα να εμφανίζονται στην καρτέλα Console view



Εικόνα 2.1: Τμήματα της διεπαφής στο Unity

Αυτή είναι η προεπιλεγμένη διάταξη στο Unity. Όλες οι διάφορες απεικονίσεις βρίσκονται σε καρτέλες και μπορούν να μετακινηθούν ή να τροποποιηθούν, προσαρμόζοντας τες σε διάφορα σημεία της οθόνης.

2.3.1 Οι καρτέλες Scene view, Game view και Toolbar

Το πιο σημαντικό μέρος της διεπαφής είναι η σκηνή (Scene view) στη μέση. Είναι ο χώρος που μπορούμε να δούμε με τι μοιάζει ο κόσμος της εφαρμογής και να μετακινήσουμε αντικείμενα γύρω. Ένα αντικείμενο (mesh object) είναι ένα οπτικό αντικείμενο (visual object) σε τρισδιάστατο χώρο, το οποίο κατασκευάζεται από πολλές συνδεδεμένες γραμμές και σχήματα. Μπορούμε επίσης να δούμε πολλά άλλα αντικείμενα στη σκηνή, που αντιπροσωπεύονται από διάφορα εικονίδια όπως κάμερες, φώτα, πηγές ήχου και ούτω καθεξής. Ας σημειώσουμε ότι η προβολή εδώ δεν είναι η ίδια με την προβολή όταν τρέχει η εφαρμογή.

Η καρτέλα Game view δεν αποτελεί ξεχωριστό τμήμα της οθόνης, αλλά μια άλλη καρτέλα που βρίσκεται δίπλα στη σκηνή. Κάποια μέρη στη διεπαφή έχουν πολλές καρτέλες όπως αυτό. Αν κάνουμε κλικ σε μια άλλη καρτέλα, η προβολή θα αντικατασταθεί από τη νέα ενεργή καρτέλα. Όταν η εφαρμογή τρέχει αυτό που βλέπουμε σε αυτή είναι το οπτικό αποτέλεσμα που θα βλέπει και ο χρήστης της εφαρμογής. Δεν είναι απαραίτητο να αλλάζουμε χειροκίνητα τις καρτέλες κάθε φορά που τρέχουμε την εφαρμογή, επειδή η προβολή μεταβαίνει αυτόματα στην καρτέλα Game view.

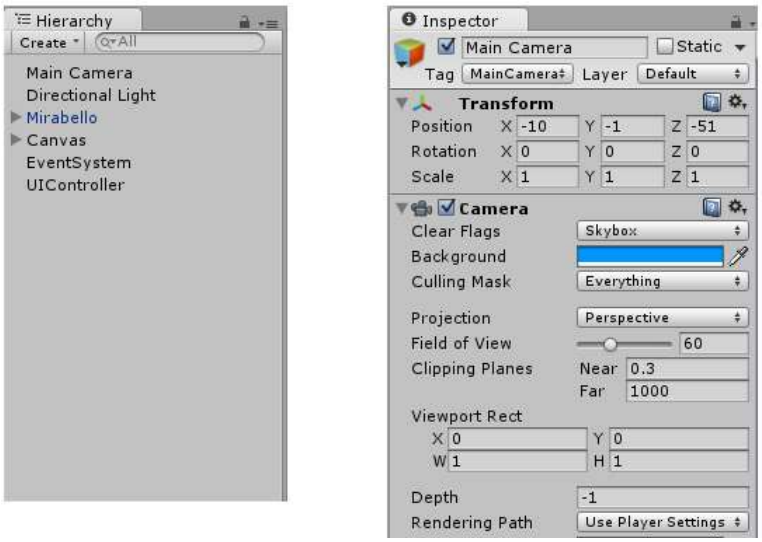
Εικόνα 2.2: Εικόνα του επεξεργαστή που δείχνει τις καρτέλες Toolbar, Scene και Game view

Μιλώντας για την εκτέλεση της εφαρμογής, πατάμε το κουμπί Play λίγο πάνω από την καρτέλα Scene view. Το άνω τμήμα της διεπαφής αναφέρεται ως γραμμή εργαλείων (Toolbar) και το Play βρίσκεται ακριβώς στη μέση. Η εικόνα 2.2 διαχωρίζει την πλήρη διεπαφή του προγράμματος επεξεργασίας για να εμφανίσει μόνο το Toolbar στην κορυφή, καθώς και τις καρτέλες Scene view και Game view ακριβώς κάτω.

Στην αριστερή πλευρά της γραμμής εργαλείων υπάρχουν κουμπιά που διευκολύνουν τη πλοήγηση στη σκηνή και το μετασχηματισμό των αντικειμένων. Δηλαδή η δυνατότητα να κοιτάμε γύρω από τη σκηνή και να μετακινούμε αντικείμενα.

2.3.2 Οι καρτέλες Hierarchy view και Inspector

Κοιτάζοντας στις πλευρές της οθόνης, βλέπουμε τις καρτέλες Hierarchy στα αριστερά και Inspector στα δεξιά (εικόνα 2.3). Η πρώτη είναι μια προβολή λίστας με το όνομα κάθε αντικειμένου στην αναφερόμενη σκηνή, με τα ονόματα να είναι ενωμένα σύμφωνα με τους ιεραρχικούς δεσμούς τους. Βασικά, είναι ένας τρόπος επιλογής αντικειμένων με το όνομα, αντί να τα αναζητούμε και να κάνουμε κλικ μέσα στη σκηνή. Η σύνδεση των αντικειμένων, τα ομαδοποιεί σε φακέλους και μας επιτρέπει να μετακινούμε ολόκληρη την ομάδα μαζί.

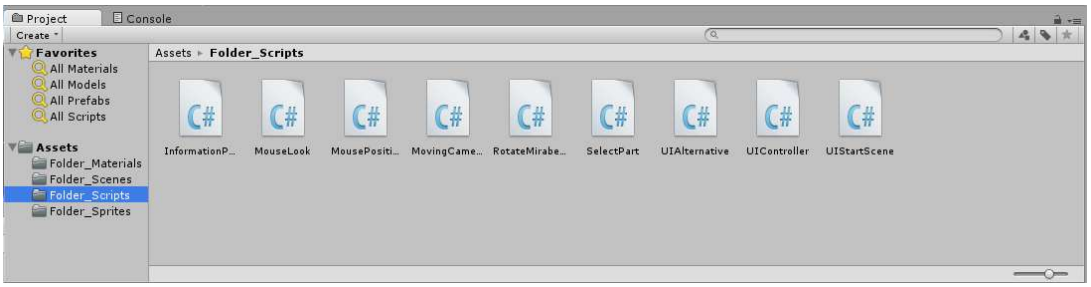


Εικόνα 2.3: Εικόνα από τον επεξεργαστή που δείχνει τις καρτέλες Hierarchy view και Inspector

Η δεύτερη εμφανίζει πληροφορίες σχετικά με το επιλεγμένο αντικείμενο, δηλαδή μπορούμε να επιλέξουμε ένα αντικείμενο και να εμφανιστούν πληροφορίες για αυτό. Οι πληροφορίες που εμφανίζονται είναι λίγο πολύ μια λίστα συνιστωσών (components) και μπορούμε να προσθέσουμε ή να αφαιρέσουμε στοιχεία από τα αντικείμενα. Όλα τα αντικείμενα έχουν τουλάχιστον ένα στοιχείο, Transform component, όπου βλέπουμε πληροφορίες σχετικά με την θέση και την περιστροφή του. Πολλές φορές, τα αντικείμενα θα έχουν πολλά στοιχεία, συμπεριλαμβανομένων και των scripts που επισυνάπτονται.

2.3.3 Οι καρτέλες Project και Console view

Στο κάτω μέρος της οθόνης βλέπουμε τις καρτέλες Project και Console view (εικόνα 2.4). Όπως και με το Scene view και το Game view, αυτές δεν είναι δύο ξεχωριστά τμήματα της οθόνης αλλά καρτέλες που μπορούμε να εναλλάξουμε. Η πρώτη παρουσιάζει όλα τα στοιχεία (τέχνη, κώδικα κλπ.) στο έργο. Συγκεκριμένα, στην αριστερή πλευρά της προβολής υπάρχει μια λίστα με τους καταλόγους του έργου. Όταν επιλέγουμε έναν κατάλογο, στη δεξιά πλευρά της προβολής εμφανίζονται τα μεμονωμένα αρχεία στον συγκεκριμένο κατάλογο. Η καταχώριση καταλόγου στο Project είναι σαν τη προβολή λίστας στην καρτέλα Hierarchy view, αλλά ενώ η πρώτη δείχνει αντικείμενα στη σκηνή, η δεύτερη εμφανίζει αρχεία που δεν περιέχονται σε κάποια συγκεκριμένη σκηνή (συμπεριλαμβανομένων των αρχείων σκηνών). Η κονσόλα είναι ο τόπος όπου εμφανίζονται τα μηνύματα από τον κώδικα. Ορισμένα από αυτά τα μηνύματα τα εισάγουμε σκόπιμα για αποσφαλμάτωση, ενώ αλλά τα εκπέμπει το Unity αν έχει πρόβλημα το script που γράψαμε.



Εικόνα 2.4: Εικόνα από τον επεξεργαστή που δείχνει τις καρτέλες Project και Console view

2.4 Προγραμματισμός στο Unity

Ας δούμε τώρα πώς λειτουργεί η διαδικασία προγραμματισμού στο Unity. Παρόλο που τα στοιχεία τέχνης (art assets) μπορούν να τοποθετηθούν στον επεξεργαστή, πρέπει να γράψουμε κώδικα για να τα ελέγξουμε και να κάνουμε την εφαρμογή πιο διαδραστική. Το Unity υποστηρίζει τις εξής γλώσσες προγραμματισμού, JavaScript και C#. Στα πλαίσια της εργασίας θα χρησιμοποιήσουμε τη δεύτερη.

Όλη η εκτέλεση κώδικα στο Unity ξεκινά από τα αρχεία κώδικα που συνδέονται με ένα αντικείμενο της σκηνής. Τελικά, όλα αποτελούν μέρος του συστήματος συνιστωσών που περιγράφηκε προηγουμένως. Τα αντικείμενα δημιουργούνται ως μια συλλογή στοιχείων και αυτή η συλλογή μπορεί να περιλαμβάνει ένα σύνολο εντολών για εκτέλεση.

Στο Unity, τα scripts είναι συστατικά. Όχι όμως όλα τα scripts, αλλά μόνο εκείνα που κληρονομούν από το MonoBehavior την βασική κλάση για τα στοιχεία script. Το MonoBehavior καθορίζει το αόρατο υπόβαθρο με το οποίο τα συστατικά επισυνάπτονται σε αντικείμενα και η κληρονομιά από αυτό παρέχει μερικές μεθόδους αυτόματης εκτέλεσης που μπορούμε να παρακάμψουμε. Αυτές οι μέθοδοι περιλαμβάνουν τη μέθοδο Start(), η οποία καλείται μία φορά όταν το αντικείμενο γίνεται ενεργό (το οποίο λαμβάνει χώρα μόλις φορτωθεί η σκηνή με αυτό το αντικείμενο) και τη μέθοδο Update(), η οποία καλείται σε κάθε πλαίσιο.

```
using UnityEngine;
using System.Collections;

public class Example : MonoBehaviour
{
    void Start()
    {
        // do something once
    }

    void Update()
    {
        // do something every frame
```

```
    }
}
```

Λίστα 2.1: Πρότυπο κώδικα για ένα βασικό script

Αυτό είναι το περιεχόμενο του αρχείου όταν δημιουργούμε ένα νέο C # script, τον ελάχιστο κώδικα που ορίζει ένα έγκυρο Unity στοιχείο. Το Unity διαθέτει ένα πρότυπο σενάριο στο πυρήνα του και όταν δημιουργούμε ένα νέο script αντιγράφει αυτό το πρότυπο και μετονομάζει την κλάση ώστε να ταιριάζει με το όνομα του αρχείου. Υπάρχουν επίσης κενά κελύφη για τις μεθόδους Start() και Update(), επειδή αυτές είναι οι δύο πιο συνηθισμένες μέθοδοι για να καλούμε τον τρέχον κώδικα μας.

Ο προγραμματισμός δεν γίνεται ακριβώς μέσα στο Unity, αλλά ο κώδικας υπάρχει ως ξεχωριστά αρχεία που δείχνουμε στο Unity. Τα αρχεία λίστας εντολών μπορούν να δημιουργηθούν μέσα στο Unity, αλλά πρέπει να χρησιμοποιήσουμε κάποιο πρόγραμμα επεξεργασίας κειμένου ή IDE για να γράψουμε όλο τον κώδικα μέσα σε αυτά τα αρχικά κενά αρχεία. Το Unity συνοδεύεται από το MonoDevelop, ένα ανοικτού κώδικα IDE για τη γλώσσα προγραμματισμού C#. Πάντα πρέπει να έχουμε κατά νου ότι αν και ο κώδικας συντάσσεται στο MonoDevelop, ο κώδικας δεν τρέχει στην πραγματικότητα εκεί. Το IDE είναι λίγο πολύ ένα πρόγραμμα επεξεργασίας κειμένου και ο κώδικας εκτελείται όταν πατάμε Play στο πλαίσιο του Unity.

ΚΕΦΑΛΑΙΟ 3

ΔΗΜΙΟΥΡΓΩΝΤΑΣ ΤΟ ΕΡΓΟ

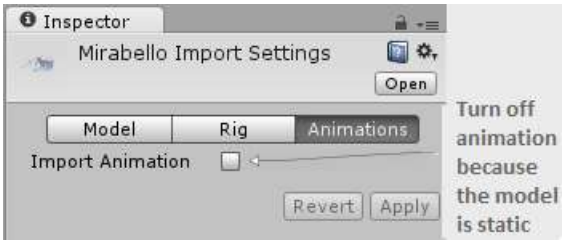
ΚΕΦΑΛΑΙΟ 3

ΔΗΜΙΟΥΡΓΩΝΤΑΣ ΤΟ ΕΡΓΟ

3.1 Πως εισάγουμε το 3D μοντέλο

Ας ξεκινήσουμε να εξηγούμε πώς δημιουργήσαμε την εφαρμογή. Το πρώτο βήμα είναι να εκκινήσουμε το Unity και να δημιουργήσουμε ένα νέο έργο. Επιλέγουμε File → New Project για να ανοίξει το παράθυρο New Project. Πληκτρολογούμε ένα όνομα για το έργο και στη συνέχεια επιλέγουμε πού θέλουμε να το αποθηκεύσουμε. Ένα έργο Unity είναι απλά ένας κατάλογος γεμάτος από διάφορα αρχεία (assets και ρυθμίσεις), οπότε μπορούμε να σώσουμε το έργο οπουδήποτε στον υπολογιστή μας. Κάνουμε κλικ στην επιλογή Create Project και στη συνέχεια το Unity θα εξαφανιστεί σύντομα για να δημιουργήσει τον κατάλογο έργου. Όταν επανεμφανιστεί, θα δούμε ένα κενό έργο. Μετά τη δημιουργία του νέου έργου, αποθηκεύουμε αμέσως την τρέχουσα κενή προεπιλεγμένη σκηνή, επειδή το έργο δεν έχει αρχικά κανένα αρχείο σκηνής. Η σκηνή ξεκινάει κενή και τα πρώτα αντικείμενα που δημιουργούνται είναι τα πιο προφανή (Main Camera και Directional Light).

Τώρα, ήρθε η ώρα να εισάγουμε το 3D μοντέλο στο Unity. Κάνουμε δεξί κλικ στη καρτέλα Project View και στη συνέχεια επιλέγουμε Import New Asset. Το 3D μοντέλο θα αντιγραφεί στην καρτέλα Project View στο Unity και θα εμφανιστεί έτοιμο να τοποθετηθεί στη σκηνή. Υπάρχει μια προεπιλεγμένη ρύθμιση που χρησιμοποιείται για την εισαγωγή του μοντέλου που χρειάζεται να αλλάξουμε αμέσως. Εναλλάσσουμε στην καρτέλα Animation view τις ρυθμίσεις εισαγωγής και καταργούμε την επιλογή Import Animation, επειδή το μοντέλο μας είναι στατικό.

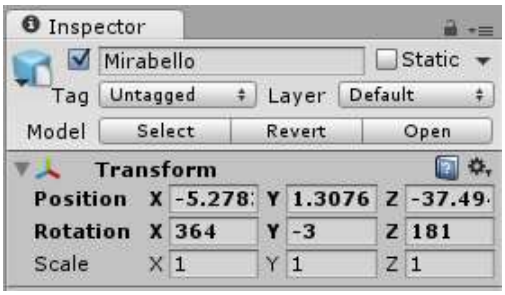


Εικόνα 3.1: Ορίζουμε τη ρύθμιση εισαγωγής για το 3D μοντέλο

Τώρα για την υφή, όταν κάναμε εισαγωγή του αρχείο OBJ, το Unity δημιούργησε ένα υλικό για το μοντέλο. Αυτό το υλικό είναι προεπιλεγμένο ως κενό (ακριβώς όπως και κάθε νέο υλικό), γι' αυτό επισυνάπτουμε την υφή του μοντέλου (εικόνα 3.2). Σέρνουμε την εικόνα υφής στην καρτέλα Project view για να την εισαγάγουμε στο Unity και στη συνέχεια σέρνουμε την υφή στην υποδοχή υφής του υλικού του μοντέλου. Η εικόνα φαίνεται κάπως περίεργη, με διαφορετικά τμήματα της εικόνας να εμφανίζονται σε διαφορετικά μέρη του μοντέλου. Οι συντεταγμένες υφής του μοντέλου έχουν υποστεί επεξεργασία για να οριστεί αυτή η χαρτογράφηση εικόνας-πλέγμα (image-to-mesh).

Εικόνα 3.2: Η 2D εικόνα για την υφή του μοντέλου

Μεταφέρουμε το μοντέλο από την καρτέλα Project view και το τοποθετούμε στην καρτέλα Hierarchy view. Το μοντέλο θα εμφανιστεί στιγμιαία στη σκηνή. Ρυθμίζουμε τη θέση και την περιστροφή αυτού, προκειμένου να αποκτήσει το κατάλληλο σχήμα, καθώς και το όνομά του. Η εικόνα 3.3 δείχνει ποιες τιμές θα πρέπει οριστούν στην καρτέλα Inspector.



Εικόνα 3.3: Ρυθμίσεις του 3D μοντέλου στην καρτέλα Inspector

3.2 Πως εφαρμόζουμε Μετασχηματισμούς

Για να έχει ο χρήστης τη δυνατότητα να περιηγηθεί στη σκηνή και να εξερευνήσει το μοντέλο, θα γράψουμε scripts μετακίνησης (movement scripts) και θα τα επισυνάψουμε στην κάμερα. Τα στοιχεία (components) είναι αρθρωτά τμήματα λειτουργικότητας που προσθέτουμε σε αντικείμενα και τα scripts είναι ένα τέτοιο είδος. Τελικά αυτά θα ανταποκρίνονται σε ερεθίσματα-εισόδους από το πληκτρολόγιο και το ποντίκι.

3.2.1 Script component για να εξετάζουμε περιστροφικά το χώρο

Σε αυτή την ενότητα θα κάνουμε την περιστροφή να ανταποκρίνεται σε εισόδους-ερεθίσματα από το ποντίκι (δηλαδή, περιστροφή του αντικειμένου που επισυνάπτεται αυτό το script και το οποίο είναι η κάμερα). Θα το αναπτύξουμε σε διάφορα βήματα, προσθέτοντας προοδευτικά νέες δυνατότητες κίνησης στην κάμερα. Πρώτα η κάμερα θα περιστρέφεται μόνο από τη μια πλευρά στην άλλη και στη συνέχεια η κάμερα θα περιστρέφεται μόνο προς τα πάνω και προς τα κάτω. Τελικά η κάμερα θα μπορεί να κοιτάζει προς όλες τις κατευθύνσεις (θα περιστρέφεται οριζόντια και κατακόρυφα ταυτόχρονα).

Δεδομένου ότι θα υπάρχουν τρία διαφορετικά είδη περιστροφής (οριζόντια, κατακόρυφη και αμφότερες), θα ξεκινήσουμε γράφοντας το πλαίσιο για την υποστήριξη και των τριών. Δημιουργούμε ένα νέο C# script, το ονομάζουμε MouseLook και γράφουμε τον ακόλουθο κώδικα.

```
using UnityEngine;
using System.Collections;

public class MouseLook : MonoBehaviour
{
    public enum RotationAxes
    {
        MouseXAndY = 0,
        MouseX = 1,
        MouseY = 2
    }
```

```
    public RotationAxes axes = RotationAxes.MouseXAndY;

    void Update()
    {
        if (axes == RotationAxes.MouseX)
        {
            // horizontal rotation here
        {

        else if (axes == RotationAxes.MouseY)
        {
            // vertical rotation here
        }

        else
        {
            // both horizontal and vertical rotation here
        }
    }
}
```

Λίστα 3.1: Πλαίσιο MouseLook script με απαριθμητή για τη ρύθμιση της περιστροφής

Ο απαριθμητής χρησιμοποιείται για την επιλογή οριζόντιας ή κατακόρυφης περιστροφής για το MouseLook script. Ο ορισμός μιας δομής δεδομένων μας επιτρέπει να ορίσουμε τιμές με το όνομα, αντί να πληκτρολογούμε αριθμούς και να προσπαθούμε να θυμηθούμε τι σημαίνει ο κάθε αριθμός. Αν στη συνέχεια δηλώσουμε μια δημόσια μεταβλητή σε αυτό τον απαριθμητή, θα εμφανιστεί στην καρτέλα Inspector ως αναπτυσσόμενο (drop-down) μενού (εικόνα 3.4), το οποίο είναι χρήσιμο για την επιλογή των ρυθμίσεων.

Εικόνα 3.4: Στην καρτέλα Inspector παρουσιάζονται δημόσιες μεταβλητές απαριθμητή ως αναπτυσσόμενο μενού

Οριζόντια περιστροφή που ακολουθεί την κίνηση του ποντικιού

Ο πρώτος και απλούστερος κλάδος είναι η οριζόντια περιστροφή. Ξεκινάμε γράφοντας την εντολή περιστροφής για να κάνουμε το αντικείμενο να γυρίζει, όπως βλέπουμε στην παρακάτω λίστα. Πρέπει να δηλώσουμε μια δημόσια μεταβλητή για την ταχύτητα περιστροφής. Δηλώνουμε τη νέα μεταβλητή μετά τους άξονες, αλλά πριν από τη μέθοδο Update() και στη συνέχεια καλούμε τη μεταβλητή sensitivityHor επειδή έχουμε πολλαπλές περιστροφές. Ορίζουμε την τιμή της μεταβλητής ίση με 5. Ο προσαρμοσμένος κώδικας πρέπει να μοιάζει με την ακόλουθη λίστα.

```
public float sensitivityHor = 5.0f;

void Update()
{
    if (axes == RotationAxes.MouseX)
    {
        Transform.Rotate (0, sensitivityHor, 0);
    }
```

Λίστα 3.2: Οριζόντια περιστροφή που δεν ανταποκρίνεται ακόμα στο ποντίκι

Αν τρέξουμε τώρα το script, η απεικόνιση θα περιστραφεί. Το επόμενο βήμα είναι να κάνουμε την περιστροφή να αντιδρά στην κίνηση του ποντικιού, οπότε ας εισαγάγουμε μια νέα μέθοδο Input.GetAxis(). Η κλάση Input έχει μια δέσμη μεθόδων για τη διαχείριση συσκευών εισόδου (όπως το ποντίκι) και η μέθοδος GetAxis() επιστρέφει αριθμούς που σχετίζονται με την κίνηση του ποντικιού (θετικούς ή αρνητικούς, ανάλογα με την κατεύθυνση κίνησης). Η μέθοδος GetAxis() παίρνει το όνομα του επιθυμητού άξονα ως παράμετρο και ο οριζόντιος άξονας καλείται Mouse X. Αν πολλαπλασιάσουμε την ταχύτητα περιστροφής με την τιμή του άξονα, η περιστροφή θα ανταποκρίνεται στην κίνηση του ποντικιού. Η ταχύτητα θα κλιμακωθεί ανάλογα με την κίνηση του ποντικιού. Η εντολή Περιστροφής μοιάζει με την επόμενη λίστα.

```
transform.Rotate (0, Input.GetAxis ("Mouse X") * sensitivityHor, 0);
```

Λίστα 3.3: Ρυθμισμένη εντολή περιστροφής ώστε να ανταποκρίνεται στο ποντίκι

Εάν πατήσουμε το Play και στη συνέχεια μετακινήσουμε το ποντίκι, θα δούμε καθώς μετακινούμε το ποντίκι από τη μία πλευρά στην άλλη, η απεικόνιση να περιστρέφεται από τη μία πλευρά στην άλλη. Το επόμενο βήμα είναι να περιστρέψουμε κατακόρυφα αντί για οριζόντια.

Κατακόρυφη περιστροφή με όρια

Για την οριζόντια περιστροφή χρησιμοποιούμε τη μέθοδο Rotate(), αλλά θα εφαρμόσουμε μια διαφορετική προσέγγιση για την κατακόρυφη περιστροφή. Αν και αυτή η μέθοδος είναι βολική για την εφαρμογή μετασχηματισμών, είναι επίσης κάπως περιοριστική. Είναι χρήσιμη μόνο για την αύξηση της περιστροφής χωρίς περιορισμούς, η οποία είναι κατάλληλη για οριζόντια περιστροφή, αλλά η κατακόρυφη περιστροφή απαιτεί όρια για το πόσο μπορεί να γίνει κλίση προς τα πάνω ή προς τα κάτω. Η παρακάτω λίστα εμφανίζει τον κώδικα της κατακόρυφης περιστροφής.

```
public float sensitivityHor = 9.0f;
public float sensitivityVert = 9.0f;
public float minimumVert = -45.0f;
public float maximumVert = 45.0f;
private float _rotationX = 0;

void Update ()
{
    if (Input.GetKey("r"))
    {
        if (axes == RotationAxes.MouseX)
        {
            transform.Rotate (0, Input.GetAxis ("Mouse X") * sensitivityHor, 0);
        }
        else if (axes == RotationAxes.MouseY)
```

```
        {
            _rotationX -= Input.GetAxis ("Mouse Y") * sensitivityVert;
            _rotationX = Mathf.Clamp(_rotationX,minimumVert,maximumVert);
            float rotationY = transform.localEulerAngles.y;
            transform.localEulerAngles = new Vector3(_rotationX,rotationY,0);
        }
    }
```

Λίστα 3.4: Κατακόρυφη περιστροφή για το MouseLook script

Υπάρχουν πολλές νέες έννοιες σε αυτόν τον κώδικα που πρέπει να εξηγηθούν. Πρώτα από όλα, δεν χρησιμοποιούμε αυτή τη φορά τη μέθοδο Rotate(), γι' αυτό χρειαζόμαστε μια μεταβλητή (που ονομάζεται _rotationX επειδή η κατακόρυφη περιστροφή είναι γύρω από τον άξονα X) στην οποία αποθηκεύεται η γωνία περιστροφής. Η μέθοδος Rotate() αυξάνει την τρέχουσα περιστροφή, ενώ αυτός ο κώδικας ρυθμίζει απευθείας τη γωνία περιστροφής. Πρέπει ακόμα να αυξήσουμε τη γωνία περιστροφής, για αυτό ο κώδικας έχει τον χειριστή -=, για να αφαιρέσει μια τιμή από τη γωνία περιστροφής αντί να ρυθμίσει τη γωνία σε αυτή την τιμή. Αλλά μη χρησιμοποιώντας τη μέθοδο Rotate(), μπορούμε να χειριστούμε τη γωνία περιστροφής με διάφορους τρόπους εκτός από την αύξηση της. Η τιμή περιστροφής πολλαπλασιάζεται με την εντολή Input.GetAxis() ακριβώς όπως και στον κώδικα για την οριζόντια περιστροφή, με εξαίρεση ότι τώρα χρησιμοποιούμε το Mouse Y επειδή αυτός είναι ο κάθετος άξονας του ποντικιού.

Η γωνία περιστροφής ρυθμίζεται περαιτέρω στην επόμενη γραμμή. Χρησιμοποιούμε τη εντολή Mathf.Clamp() για να διατηρήσουμε τη γωνία περιστροφής μεταξύ των ελάχιστων και των μέγιστων ορίων. Αυτά τα όρια είναι δημόσιες μεταβλητές που δηλώθηκαν προηγουμένως στον κώδικα και διασφαλίζουν ότι η απεικόνιση μπορεί να στραφεί μόνο 45 μοίρες προς τα πάνω ή προς τα κάτω. Η μέθοδος Clamp() δεν είναι αποκλειστικά μόνο για την περιστροφή, αλλά είναι γενικά χρήσιμη για τη διατήρηση της τιμής μιας μεταβλητής μεταξύ ορίων.

Επειδή η ιδιότητα των γωνιών μετασχηματισμού είναι ένα Vector 3, πρέπει να δημιουργήσουμε ένα νέο Vector 3 με τη γωνία περιστροφής να περνά στον κατασκευαστή (constructor). Η μέθοδος Rotate() εκτελούσε αυτόματα αυτή τη διαδικασία, αυξάνοντας τη γωνία περιστροφής και στη συνέχεια δημιουργώντας ένα νέο διάνυσμα.

Υπάρχει μια ακόμη ρύθμιση της περιστροφής για το MouseLook script που χρειάζεται να γράψουμε κώδικα, οριζόντια και κατακόρυφη περιστροφή ταυτόχρονα.

Οριζόντια και κατακόρυφη περιστροφή ταυτόχρονα

Σε αυτό το τελευταίο κομμάτι κώδικα δεν θα χρησιμοποιήσουμε τη μέθοδο Rotate() για τον ίδιο λόγο που δεν την χρησιμοποιήσαμε παραπάνω, ότι η κατακόρυφη γωνία περιστροφής περιορίζεται μεταξύ των ορίων αφού αυξηθεί. Αυτό σημαίνει ότι η οριζόντια περιστροφή πρέπει να υπολογιστεί άμεσα τώρα.

```
else
{
    _rotationX -= Input.GetAxis ("Mouse Y") * sensitivityVert;
    _rotationX = Mathf.Clamp (_rotationX, minimumVert, maximumVert);

    float delta = Input.GetAxis ("Mouse X") * sensitivityHor;
    float rotationY = transform.localEulerAngles.y + delta;
    transform.localEulerAngles = new Vector3 (_rotationX, rotationY, 0);
}
```

Λίστα 3.5: Οριζόντια και κατακόρυφη περιστροφή για το MouseLook script

Το πρώτο ζευγάρι γραμμών, που ασχολούνται με το _rotationX, είναι ακριβώς το ίδιο όπως προηγουμένως. Επειδή η οριζόντια περιστροφή δεν γίνεται πλέον με τη χρήση της μεθόδου Rotate(), αυτό είναι που κάνουν οι γραμμές delta και rotationY. Το delta είναι η ποσότητα που πρέπει να αλλάξει η περιστροφή. Αυτή η ποσότητα αλλαγής στη συνέχεια προστίθεται στην τρέχουσα γωνία περιστροφής για να επιτευχθεί η επιθυμητή νέα γωνία

περιστροφής. Τέλος και οι δύο γωνίες, δηλαδή κατακόρυφη και οριζόντια, χρησιμοποιούνται για να δημιουργήσουν ένα νέο διάνυσμα που εκχωρείται στην ιδιότητα γωνίας μετασχηματισμού. Στην επόμενη λίστα βλέπουμε το ολοκληρωμένο MouseLook script.

```
using UnityEngine;
using System.Collections;

public class MouseLook : MonoBehaviour
{
   public enum RotationAxes
   {
      MouseXAndY = 0,
      MouseX = 1,
      MouseY = 2
   }

   public RotationAxes axes = RotationAxes.MouseXAndY;
   public float sensitivityHor = 9.0f;
   public float sensitivityVert = 9.0f;
   public float minimumVert = -45.0f;
   public float maximumVert = 45.0f;
   private float _rotationX = 0;

   void Update ()
   {
       if (Input.GetKey("r"))
       {
           if (axes == RotationAxes.MouseX)
           {
                transform.Rotate(0, Input.GetAxis ("Mouse X") * sensitivityHor, 0);
            }

           else if (axes == RotationAxes.MouseY)
           {
              _rotationX -= Input.GetAxis ("Mouse Y") * sensitivityVert;
              _rotationX = Mathf.Clamp (_rotationX,minimumVert,maximumVert);
              float rotationY = transform.localEulerAngles.y;
              transform.localEulerAngles = new Vector3 (_rotationX, rotationY, 0);
            }
```

```
            else
            {
                _rotationX -= Input.GetAxis ("Mouse Y") * sensitivityVert;
                _rotationX = Mathf.Clamp (_rotationX,minimumVert,maximumVert);
                float delta = Input.GetAxis ("Mouse X") * sensitivityHor;
                float rotationY = transform.localEulerAngles.y + delta;
                transform.localEulerAngles = new Vector3 (_rotationX, rotationY, 0);
            }
        }
    }
}
```

Λίστα 3.6: Το ολοκληρωμένο MouseLook script

Αν εκτελέσουμε το νέο script, μπορούμε να κοιτάξουμε προς όλες τις κατευθύνσεις, ενώ μετακινούμε το ποντίκι. Αλλά είμαστε ακόμα κολλημένοι σε ένα μέρος, κοιτώντας γύρω σαν να βρισκόμαστε πάνω σε ένα πυργίσκο. Το επόμενο βήμα είναι η περιήγηση μέσα στη σκηνή.

3.2.2 Script component για να μετακινούμαστε

Το να κοιτάμε τριγύρω σε απόκριση της κίνησης του ποντικιού είναι ένα σημαντικό κομμάτι, αλλά υπάρχει ακόμα μια ανάγκη. Ο χρήστης να μπορεί να μετακινήσει την κάμερα σε απόκριση εισόδων-ερεθισμάτων από το πληκτρολόγιο. Ας γράψουμε μια συνιστώσα ελέγχου από το πληκτρολόγιο για να συμπληρώσουμε τη συνιστώσα ελέγχου από το ποντίκι. Αρχικά, δημιουργούμε ένα νέο C# script, το ονομάζουμε Moving Camera και το επισυνάπτουμε στη κάμερα μαζί με το MouseLook script.

Ο κώδικας που γράψαμε στην προηγούμενη ενότητα επηρέαζε μόνο την περιστροφή, αλλά τώρα θα αλλάξουμε τη θέση του αντικειμένου. Ο κώδικας για τη μετακίνηση με βάση τα πλήκτρα του πληκτρολογίου που πιέζονται εμφανίζεται παρακάτω.

```
void Update ()
{
    float deltaX = Input.GetAxis ("Horizontal") * speed;
```

```
    float deltaZ = Input.GetAxis ("Vertical") * speed;
    transform.Translate (deltaX, 0, deltaZ);
}
```

Λίστα 3.7: Μετακίνηση που ανταποκρίνεται στα πλήκτρα πίεσης

Όπως και πριν, οι τιμές GetAxis() πολλαπλασιάζονται με την ταχύτητα προκειμένου να καθοριστεί η ποσότητα κίνησης. Ενώ πριν ο απαιτούμενος άξονας ήταν πάντα " Mouse something", τώρα χρησιμοποιούμε Horizontal ή Vertical. Αυτά τα ονόματα είναι απλοποιήσεις για τις ρυθμίσεις εισόδου στο Unity. Αν κοιτάξουμε στο Edit menu κάτω από το Project Settings και μετά κοιτάξουμε κάτω από την ένδειξη Input, θα βρούμε μια λίστα αφηρημένων ονομάτων εισόδου και τα ακριβή στοιχεία ελέγχου που αντιστοιχούν σε αυτά τα ονόματα. Τόσο τα πλήκτρα βέλους αριστερά/δεξιά όσο και τα γράμματα A /D αντιστοιχίζονται σε Horizontal, ενώ τα δύο πάνω/κάτω βέλη και τα γράμματα W/S αντιστοιχίζονται σε Vertical.

Η κίνηση εφαρμόζεται στις συντεταγμένες X και Z. Η πρώτη (συντεταγμένη X) μετακινεί από τη μια πλευρά στην άλλη πλευρά και η δεύτερη (συντεταγμένη Z) μετακινεί προς τα εμπρός και προς τα πίσω.

Βάζουμε αυτόν τον νέο κώδικα κίνησης και μπορούμε να κινηθούμε πατώντας είτε τα πλήκτρα βέλους είτε τα πλήκτρα WASD. Ο κώδικας κίνησης είναι σχεδόν πλήρης, αλλά έχουμε κάποιες ακόμα προσαρμογές να κάνουμε.

Ορισμένοι υπολογιστές μπορούν να επεξεργάζονται τον κώδικα και τα γραφικά γρηγορότερα από άλλους. Αυτή τη στιγμή, η κάμερα θα κινηθεί με διαφορετικές ταχύτητες σε διαφορετικούς υπολογιστές, επειδή ο κώδικας της κίνησης συνδέεται με την ταχύτητα του υπολογιστή. Αυτό αναφέρεται ως εξαρτώμενος ρυθμός καρέ (frame rate dependent), επειδή ο κώδικας της κίνησης εξαρτάται από το ρυθμό των καρέ.

Η λύση είναι να ρυθμίσουμε τον κώδικα της κίνησης ώστε να καταστεί ανεξάρτητος από το ρυθμό των καρέ. Για να επιτευχθεί αυτό δεν πρέπει να εφαρμόσουμε την ίδια τιμή ταχύτητας σε κάθε ρυθμό καρέ. Αντί αυτού, κλιμακώνουμε την τιμή της ταχύτητας υψηλότερα ή χαμηλότερα ανάλογα με

τη ταχύτητα του υπολογιστή. Αυτό επιτυγχάνεται πολλαπλασιάζοντας την τιμή της ταχύτητας με μια άλλη τιμή που ονομάζεται deltaTime, όπως φαίνεται στην επόμενη λίστα.

```
void Update ()
{
    float deltaX = Input.GetAxis ("Horizontal") * speed;
    float deltaZ = Input.GetAxis ("Vertical") * speed;
    transform.Translate (deltaX * Time.deltaTime, 0, deltaZ * Time.deltaTime);
}
```

Λίστα 3.8: Κίνηση ανεξάρτητη από το ρυθμό των καρέ χρησιμοποιώντας το deltaTime

Η κλάση Time έχει πολλές μεταβλητές και μεθόδους χρήσιμες για χρονισμό και μία από αυτές τις μεταβλητές είναι το deltaTime. Επειδή γνωρίζουμε ότι το delta εκφράζει το ποσό της αλλαγής, αυτό σημαίνει ότι το deltaTime είναι το ποσό της αλλαγής στο χρόνο. Συγκεκριμένα, το deltaTime είναι το χρονικό διάστημα μεταξύ των πλαισίων. Ο χρόνος μεταξύ των πλαισίων ποικίλλει σε διαφορετικούς ρυθμούς καρέ (για παράδειγμα, 30 fps είναι ένα deltaTime του 1/30 του δευτερολέπτου), οπότε πολλαπλασιάζοντας την τιμή της ταχύτητας με το deltaTime θα κλιμακώσει την τιμή της ταχύτητας σε διαφορετικούς υπολογιστές. Τώρα η ταχύτητα κίνησης θα είναι η ίδια σε όλους τους υπολογιστές.

Η ακόλουθη λίστα έχει ολόκληρο τον κώδικα.

```
using UnityEngine;
using System.Collections;

public class MovingCamera : MonoBehaviour
{
   public float speed = 6.0f;
   void Update ()
   {
      float deltaX = Input.GetAxis ("Horizontal") * speed;
```

```
        float deltaZ = Input.GetAxis ("Vertical") * speed;

        transform.Translate (deltaX * Time.deltaTime, 0, deltaZ * Time.deltaTime);
    }
}
```

Λίστα 3.9: Το ολοκληρωμένο Moving Camera script

ΚΕΦΑΛΑΙΟ 4

ΔΗΜΙΟΥΡΓΙΑ ΓΡΑΦΙΚΗΣ ΔΙΕΠΑΦΗΣ ΧΡΗΣΤΗ

ΚΕΦΑΛΑΙΟ 4

ΔΗΜΙΟΥΡΓΙΑ ΓΡΑΦΙΚΗΣ ΔΙΕΠΑΦΗΣ ΧΡΗΣΤΗ

4.1 Γενικά

Στο κεφάλαιο αυτό θα δημιουργήσουμε μια δυσδιάστατη απεικόνιση διεπαφής για το έργο μας. Κάθε εφαρμογή χρειάζεται αλληλεπίδραση και παρουσίαση πληροφοριών πρόσθετα από την εικονική σκηνή. Αυτές οι αφαιρετικές ενδείξεις αλληλεπίδρασης αναφέρονται ως περιβάλλον χρήστη (UI), ή πιο συγκεκριμένα γραφικό περιβάλλον χρήστη (GUI). Το GUI αναφέρεται στο οπτικό τμήμα της διεπαφής, όπως το κείμενο και τα κουμπιά. Οι τελευταίες εκδόσεις του Unity διαθέτουν ένα νέο σύστημα διεπαφής βασισμένο σε δυσδιάστατα γραφικά που ορίζονται απευθείας στον επεξεργαστή.

Το νέο σύστημα UI λειτουργεί σε κατάσταση παραμονής, έτσι ώστε τα γραφικά να τοποθετούνται μία φορά και στη συνέχεια σχεδιάζονται σε κάθε πλαίσιο χωρίς να χρειάζεται να επαναπροσδιορίζονται συνεχώς. Σε αυτό το σύστημα, τα γραφικά για το UI τοποθετούνται στον επεξεργαστή της Unity. Αυτό παρέχει δύο πλεονεκτήματα σε σχέση με τον άμεσο τρόπο δημιουργίας UI: 1) μπορούμε να δούμε πως φαίνεται η διεπαφή χρήστη κατά την τοποθέτηση στοιχείων της διεπαφής και 2) αυτό το σύστημα καθιστά εύκολη την προσαρμογή της διεπαφής με τις δικές μας εικόνες (sprites).

4.2 Σχεδιάζοντας τη διάταξη

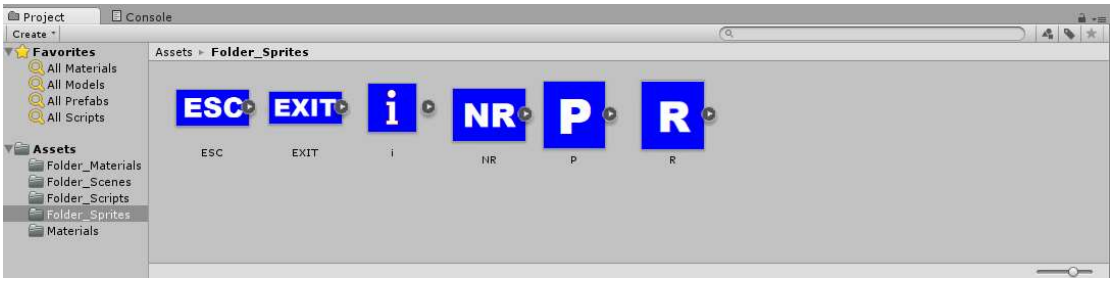
Θα τοποθετήσουμε πολλά κουμπιά στις γωνίες της οθόνης πάνω από την κύρια προβολή της εφαρμογής. Τα κουμπιά στην πάνω αριστερή γωνία καθορίζουν διαφορετικές προβολές του μοντέλου. Ένας γρήγορος τρόπος για να ορίσουμε μια προβολή είναι να επιλέξουμε μία από τις προκαθορισμένες 3D προβολές. Αυτές οι προβολές αντιπροσωπεύουν τις συνήθεις επιλογές: Top, Front, Left, Right και Back. Τα κουμπιά στην πάνω δεξιά γωνία εμφανίζουν και εξαφανίζουν το παράθυρο που παρουσιάζονται οι πληροφορίες για το μοντέλο. Τα κουμπιά στην κάτω αριστερή γωνία μας επιτρέπουν να περιστρέφουμε, να παγώνουμε και να επαναφέρουμε το μοντέλο στην αρχική του θέση. Το κουμπί στην κάτω δεξιά γωνία μας οδηγεί

στο μενού έναρξης. Το επόμενο βήμα είναι να εισάγουμε τις εικόνες που χρειαζόμαστε.

Εικόνα 4.1: Διάταξη της διεπαφής χρήστη

4.3 Ορίζοντας τη παρουσίαση της γραφικής διεπαφής χρήστη

Αυτή η διεπαφή χρήστη απαιτεί την εμφάνιση ορισμένων εικόνων που θα λειτουργούν ως κουμπιά. Το περιβάλλον διεπαφής χρήστη δημιουργείται από δυσδιάστατες εικόνες. Πρώτον, πρέπει να μεταφέρουμε τις εικόνες στην καρτέλα Project view (όπως φαίνεται στην εικόνα 4.2) για να τις εισαγάγουμε και στη συνέχεια στη καρτέλα Inspector αλλάζουμε τη ρύθμιση Type Texture σε Sprite (2D και UI). Το Unity παρέχει ειδικά εργαλεία για να κάνουμε τις 2D εικόνες να εμφανίζονται ως HUD (heads-up display) πάνω στη 3D σκηνή, αντί να εμφανίζει τις εικόνες ως μέρος της σκηνής. Η τοποθέτηση UI στοιχείων έχει επίσης μερικές ειδικές τεχνικές, λόγω της ανάγκης που υπάρχει η παρουσίαση να μπορεί να αλλάζει σε διαφορετικές οθόνες.



Εικόνα 4.2: Εικόνες που εισάγουμε στην καρτέλα Project view

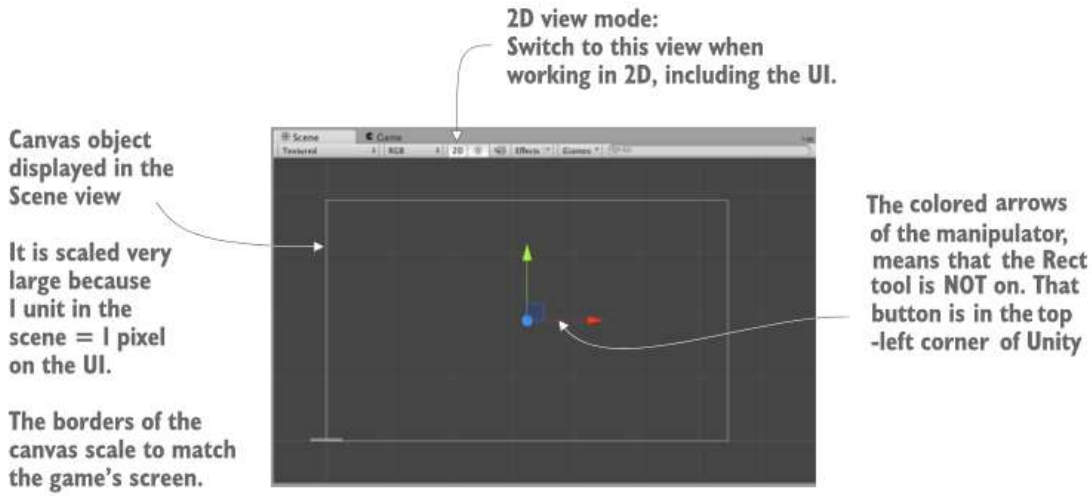
4.3.1 Δημιουργία καμβά για τη διεπαφή

Μία από τις πιο θεμελιώδεις και μη προφανείς πτυχές του τρόπου λειτουργίας του συστήματος UI είναι ότι όλες οι εικόνες πρέπει να συνδέονται με ένα

αντικείμενο καμβά. Ο καμβάς είναι ένα ειδικό είδος αντικειμένου που το Unity χρησιμοποιεί για τη δημιουργία της διεπαφής.

Αν ανοίξουμε το GameObject μενού, θα δούμε τα διάφορα είδη αντικειμένων που μπορούμε να δημιουργήσουμε. Στην κατηγορία UI, επιλέγουμε Canvas. Ένα αντικείμενο καμβά θα εμφανιστεί στη σκηνή. Αυτό το αντικείμενο αντιπροσωπεύει ολόκληρη την έκταση της οθόνης και είναι τεράστιο σε σχέση με την τρισδιάστατη σκηνή επειδή κλιμακώνει ένα pixel της οθόνης σε ένα unit στη σκηνή. Όταν δημιουργούμε ένα καμβά, δημιουργείται αυτόματα και ένα αντικείμενο EventSystem. Αυτό το αντικείμενο απαιτείται για την αλληλεπίδραση με τη διεπαφή.

Πρέπει να μεταβούμε στη λειτουργία 2D προβολής (εικόνα 4.3) και κάνουμε διπλό κλικ στον καμβά στην καρτέλα Hierarchy view για να μικρύνει (zoom out) και να το δούμε πλήρως. Η λειτουργία 2D προβολής είναι αυτόματη όταν ολόκληρο το έργο είναι 2D, αλλά σε μια 3D εφαρμογή πρέπει να γίνει κλικ για εναλλαγή μεταξύ της διεπαφής χρήστη και της κύριας σκηνής. Για να επιστρέψουμε στην προβολή της τρισδιάστατης σκηνής, απενεργοποιούμε τη λειτουργία 2D προβολής και στη συνέχεια κάνουμε διπλό κλικ στο μοντέλο για να κάνουμε ζουμ σε αυτό το αντικείμενο.



Εικόνα 4.3: Ένα κενό αντικείμενο καμβά στην καρτέλα Scene view

Ο καμβάς έχει πολλές ρυθμίσεις που μπορούμε να προσαρμόσουμε. Πρώτον, είναι η επιλογή Render Mode. Αφήνουμε την προεπιλεγμένη ρύθμιση, αλλά είναι καλό να γνωρίζουμε ποιες είναι οι τρεις πιθανές ρυθμίσεις:
1. Screen Space - Overlay: Εμφανίζει τη διεπαφή χρήστη ως δισδιάστατα γραφικά πάνω στη προβολής της κάμερας (αυτή είναι η προεπιλεγμένη ρύθμιση).
2. Screen Space - Camera: Επίσης εμφανίζει τη διεπαφή χρήστη πάνω στη προβολή της κάμερας, αλλά τα στοιχεία της διεπαφής μπορούν να περιστραφούν δημιουργώντας εφέ.
3. World Space: Τοποθετεί το αντικείμενο καμβά μέσα στη σκηνή, σαν να είναι η διεπαφή μέρος της 3D σκηνής.
Οι δύο τελευταίοι μπορούν να είναι χρήσιμοι για συγκεκριμένα εφέ, αλλά είναι λίγο πιο περίπλοκοι.

Η άλλη σημαντική ρύθμιση είναι το Pixel Perfect. Αυτή η ρύθμιση επιτρέπει τη προσαρμοσμένη απεικόνιση με ακρίβεια στις θέσεις των εικόνων έτσι ώστε να είναι πάντα τέλεια ευκρινείς και αιχμηρές (σε αντίθεση με τη θολότητα που παρουσιάζουν όταν τοποθετούνται μεταξύ εικονοστοιχείων). Επιλέγουμε αυτό το πλαίσιο ελέγχου. Τώρα ο καμβάς HUD έχει ρυθμιστεί, αλλά είναι ακόμα κενός και χρειάζεται εικόνες (sprites).

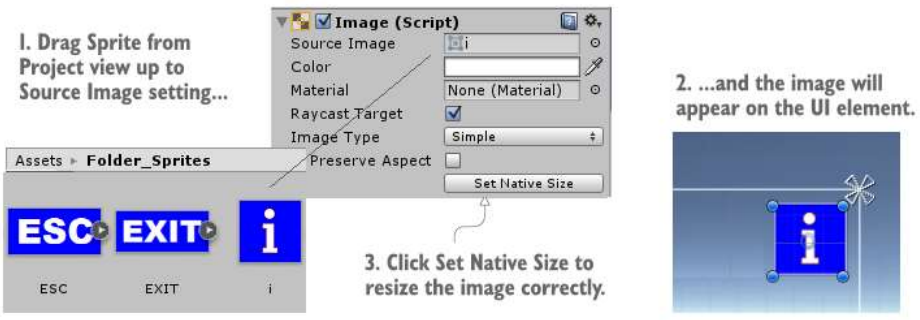
Ο καμβάς καθορίζει τη περιοχή που θα εμφανίζεται ως διεπαφή χρήστη, αλλά υπάρχει ανάγκη ακόμη να τοποθετήσουμε εικόνες. Αν ανατρέξουμε πίσω στο σχεδιάγραμμα της διεπαφής στο σχήμα 4.1, υπάρχουν διάφορα κουμπιά με διαφορετικές λειτουργίες. Στην επιλογή UI του GameObject μενού υπάρχουν επιλογές για τη δημιουργία αντικειμένων εικόνας, κειμένου ή κουμπιού. Πρέπει να δημιουργήσουμε έντεκα διαφορετικά κουμπιά.

Τα στοιχεία της διεπαφής πρέπει να είναι ένα παιδιά του αντικειμένου καμβά έτσι ώστε να εμφανίζονται σωστά. Το Unity το κάνει αυτόματα, αλλά μπορούμε να το ορίσουμε και μόνοι μας. Μεταφέρουμε αντικείμενα στην καρτέλα Hierarchy view (εικόνα 4.4) για να δημιουργήσουμε δεσμούς γονέα – παιδιού (parent-child relationship).

Εικόνα 4.4: Ο καμβάς με τα συνδεδεμένα κουμπιά στην καρτέλα Hierarchy view

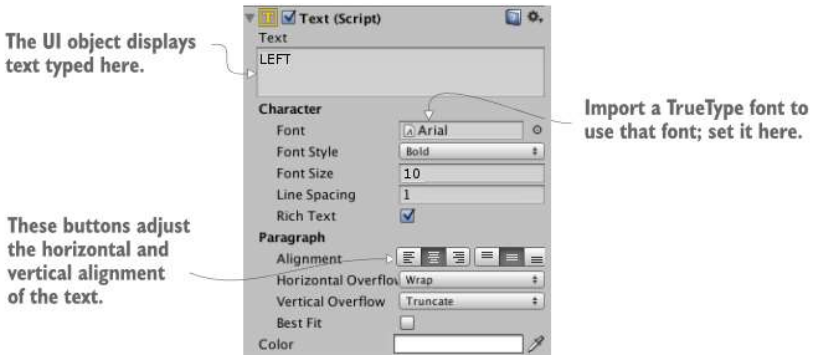
Τα αντικείμενα μέσα στον καμβά μπορούν να συνδέονται μεταξύ τους για λόγους τοποθέτησης, όπως και κάθε άλλο αντικείμενο της σκηνής. Για παράδειγμα, μπορεί να θέλουμε να σύρουμε το αντικείμενο κειμένου (text object) στην εικόνα, έτσι ώστε το κείμενο να μετακινείται μαζί με την εικόνα. Ομοίως, το προεπιλεγμένο αντικείμενο κουμπιού έχει ένα αντικείμενο κειμένου ως παιδί του. Τα κουμπιά στην πάνω αριστερή γωνία χρειάζονται μια ετικέτα κειμένου. Τα άλλα δεν χρειάζονται, γι' αυτό τις διαγράφουμε από αυτά. Τοποθετούμε στο περίπου τα στοιχεία της διεπαφής στις γωνίες τους. Στη συνέχεια θα κάνουμε τις θέσεις ακριβείς.

Αυτή τη στιγμή τα κουμπιά είναι κενά. Αν επιλέξουμε ένα αντικείμενο UI και κοιτάξουμε στην καρτέλα Inspector, θα δούμε μια υποδοχή Source Image κοντά στο πάνω μέρος της συνιστώσας της εικόνας (image component). Όπως φαίνεται στην εικόνα 4.5, σέρνουμε τις εικόνες από την καρτέλα Project view και τις αντιστοιχούμε στα αντικείμενα. Επισυνάπτουμε τη σωστή εικόνα στο σωστό κουμπί (για παράδειγμα, την εικόνα για τις πληροφορίες στο κουμπί που εμφανίζει πληροφορίες). Τσεκάρουμε την επιλογή Set Native Size αφού έχουμε αναθέσει τις εικόνες για να αποκτήσουν το σωστό μέγεθος.



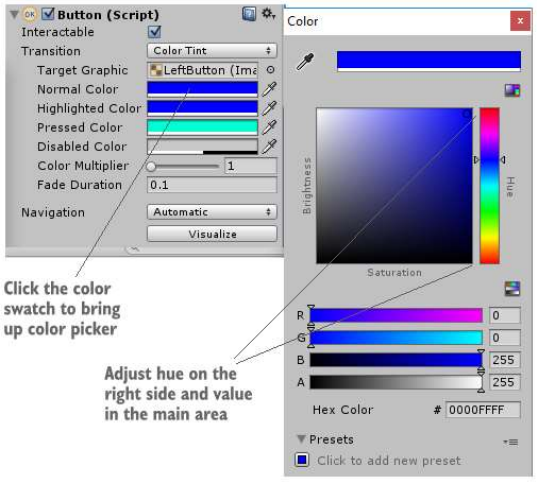
Εικόνα 4.5: Ανάθεση εικόνων (sprites) στην υποδοχή των στοιχείων της διεπαφής

Αυτό φρόντισε για την εμφάνιση των κουμπιών χωρίς κείμενο. Όσον αφορά τα κουμπιά με κείμενο, υπάρχει μια σειρά ρυθμίσεων στην καρτέλα Inspector. Αρχικά, πληκτρολογούμε ένα όνομα στο πλαίσιο κειμένου. Ορίζουμε το μέγεθος γραμματοσειράς σε 10 και κάνουμε το στυλ Bold. Θέλουμε επίσης να ορίσουμε αυτήν την ετικέτα σε κεντρική οριζόντια ευθυγράμμιση και μεσαία κάθετη ευθυγράμμιση (εικόνα 4.6).



Εικόνα 4.6: Ρυθμίσεις για ένα κουμπί διεπαφής με κείμενο

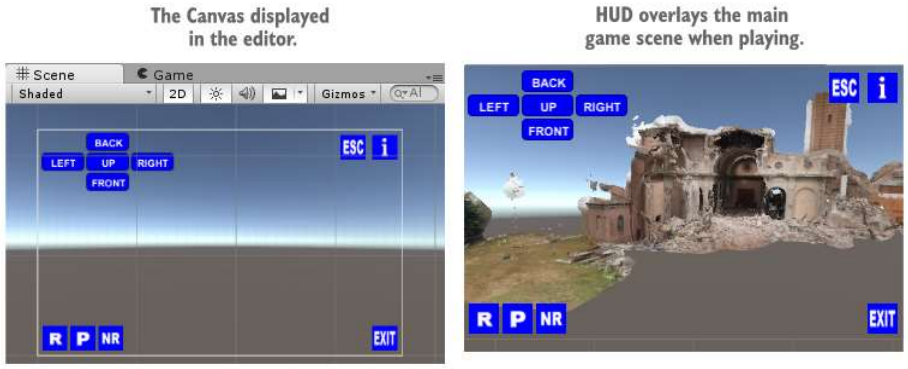
Αν επιλέξουμε ένα αντικείμενο κουμπί της διεπαφής και κοιτάξουμε στην καρτέλα Inspector, θα δούμε τις επιλογές Normal Color, Highlighted Color και Pressed Color κοντά στο επάνω μέρος. Επιλέγουμε το δείγμα χρώματος. Κάνοντας κλικ σε αυτό θα εμφανιστεί ένα νέο παράθυρο επιλογής χρώματος. Σπρώχνουμε τη μπάρα χρώματος στη δεξιά πλευρά και στην κύρια περιοχή επιλογής ρυθμίζουμε το χρώμα σε μπλε.



Εικόνα 4.7: Ρυθμίζοντας το χρώμα ενός κουμπιού

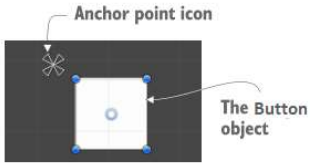
Εκτός από το πλαίσιο κειμένου και την ευθυγράμμιση, η πιο κοινή ιδιότητα που προσαρμόζεται είναι η γραμματοσειρά. Μπορούμε να εισαγάγουμε μια γραμματοσειρά στο Unity και έπειτα να βάλουμε αυτή τη γραμματοσειρά στην καρτέλα Inspector.

Τώρα που οι εικόνες (sprites) έχουν ανατεθεί στα κουμπιά της διεπαφής και το κείμενο έχει ρυθμιστεί, μπορούμε να πατήσουμε το Play για να δούμε τη HUD διεπαφή στη 3D σκηνή. Όπως φαίνεται στην εικόνα 4.8, ο καμβάς που εμφανίζεται στον επεξεργαστή του Unity δείχνει τα όρια της οθόνης και τα στοιχεία της διεπαφής σχεδιάζονται στην οθόνη σε αυτές τις θέσεις. Μία πιο σύνθετη οπτική ρύθμιση παραμένει, η ακριβής τοποθέτηση των στοιχείων της διεπαφής σε σχέση με τον καμβά.



Εικόνα 4.8: Η διεπαφή όπως φαίνεται στον επεξεργαστή (αριστερά) και όταν επιλέγουμε Play (δεξιά)

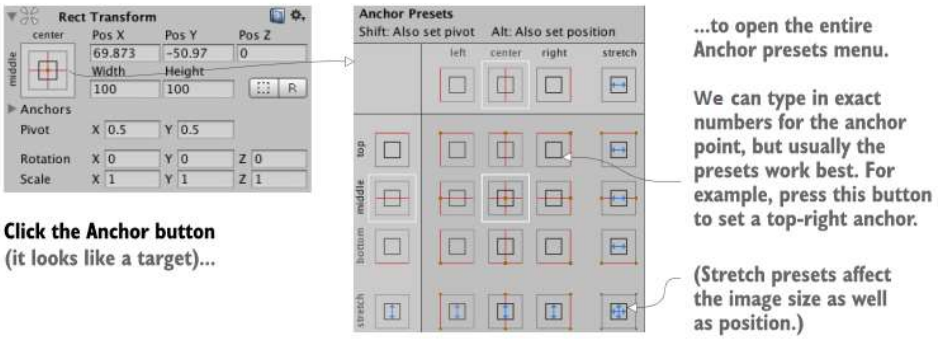
Όλα τα αντικείμενα της διεπαφής έχουν μια άγκυρα, που εμφανίζεται στον επεξεργαστή ως στόχος X (εικόνα 4.9). Μια άγκυρα είναι ένας ευέλικτος τρόπος τοποθέτησης αντικειμένων στη διεπαφή. Η άγκυρα ενός αντικειμένου είναι το σημείο όπου ένα αντικείμενο συνδέεται με τον καμβά ή την οθόνη και καθορίζει τη σχετική θέση του αντικειμένου.



Εικόνα 4.9: Το σημείο αγκύρωσης σε ένα κουμπί της διεπαφής

Ο σκοπός της άγκυρας είναι ότι ενώ το αντικείμενο παραμένει στη θέση του σε σχέση με το σημείο αγκύρωσης, η άγκυρα κινείται με βάση τον καμβά. Η άγκυρα ορίζεται σαν το "κέντρο της οθόνης" και στη συνέχεια η άγκυρα θα παραμείνει κεντραρισμένη ενώ η οθόνη αλλάζει μέγεθος. Ομοίως, η ρύθμιση της άγκυρας στη δεξιά πλευρά της οθόνης θα διατηρήσει το αντικείμενο ριζωμένο στη δεξιά πλευρά ακόμα και αν αλλάξει το μέγεθος της οθόνης.

Από προεπιλογή, τα στοιχεία της διεπαφής έχουν την άγκυρα ρυθμισμένη στο κέντρο, αλλά εμείς θέλουμε να ρυθμίσουμε την αγκύρωση πάνω αριστερά, πάνω δεξιά, κάτω αριστερά και κάτω δεξιά, ακριβώς όπως στο σχεδιάγραμμα της διεπαφής στην εικόνα 4.1. Το Unity μας επιτρέπει να αλλάξουμε την άγκυρα του κουμπιού. Αν θέλουμε να την ορίσουμε Top Right, πρέπει να κάνουμε κλικ στην επάνω δεξιά γωνία πρόσδεσης. Τώρα αν προσπαθήσουμε να κλιμακώσουμε το παράθυρο αριστερά και δεξιά, χάρη στις άγκυρες, τα αντικείμενα της διεπαφής θα παραμείνουν στις γωνίες τους ενώ ο καμβάς αλλάζει μέγεθος. Οι ρυθμίσεις αγκύρωσης (εικόνα 4.10) εμφανίζονται ακριβώς κάτω από το στοιχείο μετασχηματισμού (transform component).



Εικόνα 4.10: Πως ρυθμίζουμε την αγκύρωση

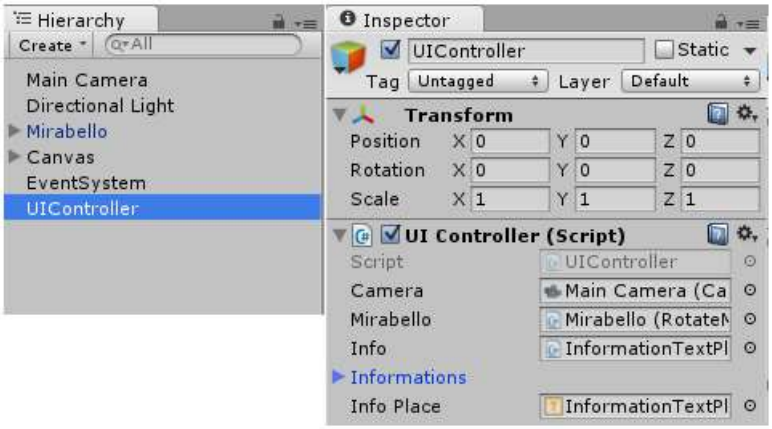
Τώρα όλες οι οπτικές ρυθμίσεις έχουν ολοκληρωθεί, οπότε είναι η ώρα να προγραμματίσουμε έτσι ώστε τα κουμπιά να γίνουν λειτουργικά.

4.3.2 Προγραμματισμός για τη λειτουργία της διεπαφής χρήστη

Σε γενικές γραμμές, η αλληλεπίδραση της διεπαφής προγραμματίζεται με μια τυπική σειρά βημάτων που είναι ίδια για όλα τα στοιχεία της:

1. Δημιουργούμε ένα αντικείμενο διεπαφής (UI object) στη σκηνή (τα κουμπιά δημιουργήθηκαν στην προηγούμενη ενότητα)
2. Γράφουμε τον κώδικα (script) για να καλούμε όταν η διεπαφή λειτουργεί
3. Επισυνάπτουμε τον κώδικα σε ένα αντικείμενο της σκηνής
4. Συνδέουμε στοιχεία της διεπαφής (όπως τα κουμπιά) στο αντικείμενο με το συγκεκριμένο κώδικα

Για να ακολουθήσουμε αυτά τα βήματα, πρώτα πρέπει να δημιουργήσουμε ένα αντικείμενο (controller object) για να το συνδέσουμε με τα κουμπιά. Δημιουργούμε ένα script, το ονομάζουμε UIController και το σέρνουμε στο αντικείμενο της σκηνής που δημιουργήσαμε προηγουμένως (όπως φαίνεται στη παρακάτω εικόνα).



Εικόνα 4.11: UIController αντικείμενο και script

Η διεπαφή χρήστη έχει πολλά κουμπιά με διαφορετικές λειτουργίες. Θα ξεκινήσουμε να εξηγούμε πώς να προγραμματίσουμε καθένα από αυτά.

Κουμπιά 3D προβολής

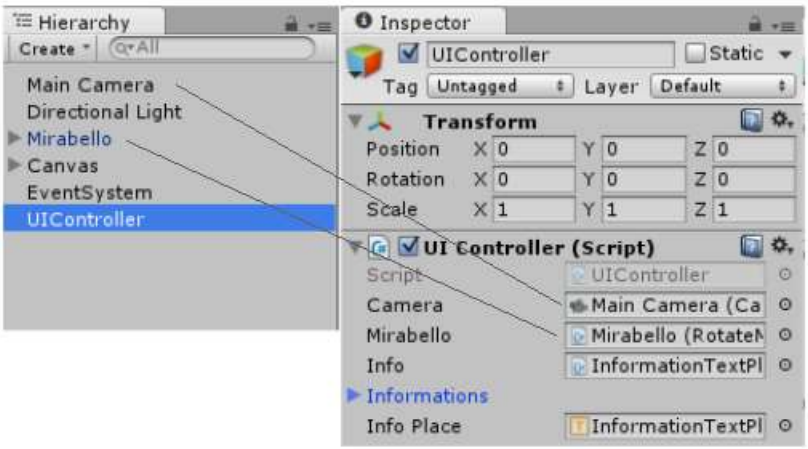
Στην πάνω αριστερή γωνία υπάρχουν πέντε κουμπιά προκαθορισμένης 3D προβολής. Έτσι, ανοίγουμε το UIController script και γράφουμε τον κώδικα, όπως φαίνεται στην παρακάτω λίστα.

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

public class UIController : MonoBehaviour
{
    [SerializeField]
    private Camera _camera;

    [SerializeField]
    private RotateMirabello _mirabello;

    public void SelectLeftButton()
    {
        _mirabello.transform.position = new Vector3 (-5, 1, -37);
        _mirabello.transform.localEulerAngles = new Vector3 (364, -3, 181);
        _camera.transform.position = new Vector3 (-27, 3, -12);
        _camera.transform.localEulerAngles = new Vector3 (0, 105, 0);
        Debug.Log ("select left button");
    }

    public void SelectRighttButton()
    {
        _mirabello.transform.position = new Vector3 (-5, 1, -37);
        _mirabello.transform.localEulerAngles = new Vector3 (364, -3, 181);
        _camera.transform.position = new Vector3 (28, 4, -19);
        _camera.transform.localEulerAngles = new Vector3 (0, 292, 0);
        Debug.Log ("select right button");
    }

    public void SelectFronttButton()
    {
        _camera.transform.position = new Vector3 (-10, -1, -51);
        _camera.transform.localEulerAngles = new Vector3 (0, 0, 0);
        Debug.Log ("select front button");
    }

    public void SelectBacktButton()
    {
        _mirabello.transform.position = new Vector3 (-5, 1, -37);
        _mirabello.transform.localEulerAngles = new Vector3 (364, -3, 181);
```

```
        _camera.transform.position = new Vector3 (12, 7, 20);
        _camera.transform.localEulerAngles = new Vector3 (0, 190, 0);
        Debug.Log ("select back button");
    }

    public void SelectUptButton()
    {
        _mirabello.transform.position = new Vector3 (-5, 1, -37);
        _mirabello.transform.localEulerAngles = new Vector3 (364, -3, 181);
        _camera.transform.position = new Vector3 (-8, 12, -41);
        _camera.transform.localEulerAngles = new Vector3 (0, 35, 0);
        Debug.Log ("select up button");
    }
```

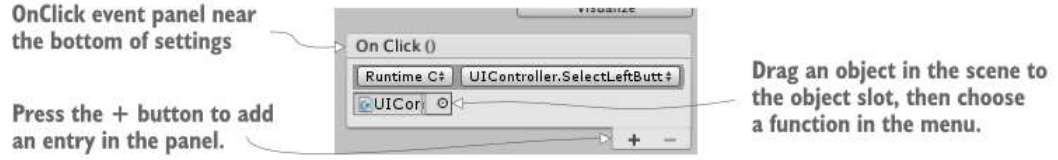
Λίστα 4.1: UIController script που χρησιμοποιείται για το προγραμματισμό των κουμπιών της 3D προβολής

Αυτό το απόσπασμα κώδικα εισάγει δύο νέες έννοιες. Η πρώτη έννοια που επισημαίνουμε είναι οι μεταβλητές για την αναφορά στη κάμερα και στο μοντέλο. Κάθε μεταβλητή απλά δημιουργεί μια τοπική αναφορά στο αντικείμενο. Ορίζουμε τη μεταβλητή ως serialized και στη συνέχεια σέρνουμε το αντικείμενο από την καρτέλα Hierarchy view στην υποδοχή της καρτέλας Inspector. Ορίζοντας την αναφορά του αντικειμένου, ο κώδικας θα επιδρά τώρα στο αντικείμενο της σκηνή (εικόνα 4.12).



Εικόνα 4.12: Ορίζουμε τις αναφορές των αντικειμένων

Η δεύτερη έννοια είναι ένας αριθμός μεθόδων (SelectRightButton, SelectLeftButton, SelectFrontButton, SelectBackButton, SelectUpButton). Η καθεμία ορίζει τη θέση και την περιστροφή τόσο της κάμερας όσο και του μοντέλου. Για να γίνουν λειτουργικά τα κουμπιά, πρέπει να προσθέσουμε μια καταχώρηση OnClick στο κουμπί και να σύρουμε το αντικείμενο (controller object) πάνω σε αυτό. Επιλέγουμε το κουμπί για να δούμε τις ρυθμίσεις του στην καρτέλα Inspector. Προς το τέλος βλέπουμε ένα πάνελ OnClick. Αρχικά ο πίνακας είναι κενός, αλλά (όπως φαίνεται στην εικόνα 4.13) μπορούμε να πατήσουμε το κουμπί + για να προσθέσουμε μια καταχώρηση σε αυτό το πάνελ. Κάθε καταχώρηση καθορίζει μια μόνο συνάρτηση που καλείται όταν πατηθεί το κουμπί. Η καταχώριση έχει μια υποδοχή για ένα αντικείμενο και ένα μενού για τη κλήση της μεθόδου. Μεταφέρουμε το αντικείμενο controller στην υποδοχή του αντικειμένου και, στη συνέχεια, αναζητούμε το UIController στο μενού. Επιλέγουμε τη σωστή μέθοδο για κάθε κουμπί (για παράδειγμα, στο δεξί κουμπί επιλέγουμε SelectRightButton()). Η διαδικασία θα επαναληφθεί για κάθε κουμπί 3D προβολής. Εάν επιλέξουμε Play και κάνουμε κλικ στο κουμπί θα δούμε την αντίστοιχη 3D απεικόνιση.



Εικόνα 4.13: Το OnClick πάνελ στις ρυθμίσεις του κουμπιού

Κουμπιά περιστροφής, παύσης και επαναφοράς

Στην κάτω αριστερή γωνία υπάρχουν τρία κουμπιά. Το πρώτο περιστρέφει το μοντέλο, το δεύτερο σταματάει την περιστροφή και το τρίτο επαναφέρει το μοντέλο στην αρχική του θέση. Τώρα ας δούμε πώς μπορούμε να προγραμματίσουμε καθένα από αυτά. Πρώτον, δημιουργούμε ένα νέο C# script, το ονομάζουμε RoateMirabello και γράφουμε τον κώδικα από την ακόλουθη λίστα.

```
using UnityEngine;
using System.Collections;

public class RotateMirabello : MonoBehaviour
{
    [SerializeField]
    private Camera _camera;

    public float speed = 0f;

    void Update()
    {
       transform.Rotate (0, speed, 0);
    }

    public void RotateAroundYAxis()
    {
       _camera.transform.position = new Vector3 (-10, -1, -100);
       _camera.transform.localEulerAngles = new Vector3 (0, 0, 0);
       speed = 0.2f;
    }

    public void NotRotateAroundYAxis()
    {
       speed = 0f;
       _camera.transform.position = new Vector3 (-10, -1, -51);
       _camera.transform.localEulerAngles = new Vector3 (0, 0, 0);
       transform.localEulerAngles = new Vector3 (364, -3, 181);
    }

    public void PauseRotation()
    {
       speed = 0f;
    }
}
```

Λίστα 4.2: Κάνοντας το μοντέλο να περιστρέφεται, να παγώνει και να επανέρχεται στην αρχική του θέση

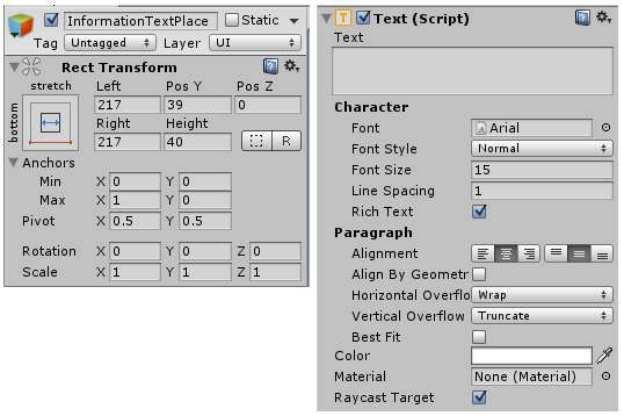
Για να προσθέσουμε το κώδικα (script) στο αντικείμενο Mirabello, το σέρνουμε προς τα πάνω από την καρτέλα Project view και το τοποθετούμε στο μοντέλο στην καρτέλα Hierarchy view. Αυτός ο κώδικας έχει μια αναφορά στη κάμερα, μια δημόσια μεταβλητή για την ταχύτητα περιστροφής και τέσσερις μεθόδους. Πρώτον, η αναφορά στη κάμερα χρησιμοποιείται με τον ίδιο τρόπο που περιγράψαμε προηγουμένως. Δεύτερον, υπάρχει η μεταβλητή για την ταχύτητα που εισάγεται προς την κορυφή του ορισμού κλάσης. Υπάρχουν δύο λόγοι για τον ορισμό της ταχύτητας περιστροφής ως μεταβλητή, ο πρώτος είναι ένας τυπικός κανόνας προγραμματισμού και ο δεύτερος σχετίζεται με το πώς το Unity εμφανίζει δημόσιες μεταβλητές. Η πρώτη μέθοδος είναι το Rotate(). Αυτή είναι μέσα στη μέθοδο Update() έτσι ώστε η εντολή να εκτελείται σε κάθε καρέ. Το Rotate() είναι μια μέθοδος της κλάσης μετασχηματισμού (Transform class). Η ταχύτητα περιστροφής έχει οριστεί κατάλληλα με αποτέλεσμα να επιτυγχάνεται μια ομαλή κίνηση περιστροφής. Η δεύτερη μέθοδος είναι το RotateAroundYAxis(). Αυτό καλείται όταν πιέζουμε το κουμπί περιστροφής. Οι εντολές στο εσωτερικό της καθορίζουν τη θέση και την περιστροφή της κάμερας, καθώς και την τιμή της ταχύτητας. Η τρίτη μέθοδος είναι το NotRotateAroundYAxis(). Αυτό καλείται όταν πατάμε το κουμπί μη περιστροφής (pause). Οι εσωτερικές εντολές ρυθμίζουν την τιμή της ταχύτητας, τη θέση και την περιστροφή της κάμερας, καθώς και την περιστροφή του μοντέλου. Η τελευταία μέθοδος είναι το PauseRotation(). Αυτή απλά ορίζει την τιμή ταχύτητας ίση με 0.

Για να γίνουν λειτουργικά τα κουμπιά, πρέπει να προσθέσουμε μια καταχώρηση OnClick στο κουμπί και να σύρουμε το αντικείμενο (controller object) πάνω σε αυτό. Επιλέγουμε το κουμπί για να δούμε τις ρυθμίσεις του στην καρτέλα Inspector. Προς το τέλος βλέπουμε ένα πάνελ OnClick. Αρχικά ο πίνακας είναι κενός, αλλά μπορούμε να πατήσουμε το κουμπί + για να προσθέσουμε μια καταχώρηση σε αυτό. Κάθε καταχώρηση καθορίζει μια μόνο συνάρτηση που καλείται όταν πατηθεί το κουμπί. Η καταχώριση έχει μια υποδοχή για ένα αντικείμενο και ένα μενού για τη κλήση της μεθόδου. Σέρνουμε το αντικείμενο mirabello στην υποδοχή του αντικειμένου και στη συνέχεια, αναζητούμε το RotateMirabello στο μενού. Επιλέγουμε τη σωστή μέθοδο για κάθε κουμπί (για παράδειγμα, στο κουμπί περιστροφής επιλέγουμε το RotateAroundYAxis()). Η διαδικασία θα επαναληφθεί για κάθε

κουμπί. Εάν επιλέξουμε Play μπορούμε να περιστρέψουμε, να παγώσουμε και να επαναφέρουμε το 3D μοντέλο στην αρχική του θέση.

Κουμπιά εμφάνισης - απόκρυψης πληροφοριών για το μοντέλο

Στην επάνω δεξιά γωνία υπάρχουν δύο κουμπιά. Επιλέγοντας ένα από αυτά εμφανίζονται πληροφορίες σχετικά με το 3D μοντέλο (Mirabello). Το άλλο κλείνει το παράθυρο πληροφοριών. Αρχικά, δημιουργούμε ένα αντικείμενο κειμένου. Έτσι, επιλέγουμε Create → UI → Text. Ένα αντικείμενο κειμένου θα εμφανιστεί στη σκηνή. Το τοποθετούμε στο κάτω μέρος της οθόνης. Στη συνέχεια, καθορίζουμε τις ακόλουθες ρυθμίσεις.



Εικόνα 4.14: Πως ρυθμίζουμε το αντικείμενο κειμένου (text object)

Στη συνέχεια δημιουργούμε ένα νέο script και το επισυνάπτουμε σε αυτό το αντικείμενο. Επιλέγουμε Create → C# script, το ονομάζουμε Information PanelSettings και γράφουμε τον κώδικα από την παρακάτω λίστα.

```
using UnityEngine;
using System.Collections;

public class InformationPanelSettings : MonoBehaviour
{
   public void Open()
   {
      gameObject.SetActive (true);
   }
```

```
    public void Close ()
    {
        gameObject.SetActive (false);
    }
}
```

Λίστα 4.3: InformationPanelSettings script για το αντικείμενο κειμένου (text object)

Ο παραπάνω κώδικας έχει δύο μεθόδους. Η πρώτη ανοίγει το παράθυρο κειμένου και η δεύτερη το κλείνει. Στη συνέχεια, ανακαλούμε το UIController script για να κάνουμε μερικές προσαρμογές, όπως φαίνεται στην παρακάτω λίστα.

```
[SerializeField]
private InformationPanelSettings _info;

[SerializeField]
private string[] informations;

float speed = 0.03f;
int stringIndex = 0;
int characterIndex = 0;

IEnumerator DisplayTimer ()
{
    while (true)
    {
        yield return new WaitForSeconds (speed);
        if (characterIndex > informations[stringIndex].Length)
        {
            continue;
        }
        _infoPlace.text = informations [stringIndex].Substring (0,characterIndex);
        characterIndex++;
    }
}

void Start ()
{
```

```
    _info.Close ();
}

void Update()
{
    if (Input.GetKeyDown (KeyCode.Space))
    {
        if (characterIndex < informations [stringIndex].Length)
        {
            characterIndex = informations [stringIndex].Length;
        }
        else if (stringIndex < informations.Length)
        {
            stringIndex++;
            characterIndex = 0;
        }
    }

         if (stringIndex == informations.Length)
         {
             stringIndex = 0;
             characterIndex = 0;
             _info.Close ();
         }
}

public void OnOpenInformationPanel()
{
   _info.Open ();
   StartCoroutine (DisplayTimer ());
   Debug.Log ("information panel open");
}

public void OnCloseInformationPanel()
{
   _info.Close ();
   Debug.Log ("information panel close");
}
```

Λίστα 4.4: Ρυθμίσεις στο UIController script

Η πρώτη γραμμή είναι μια αναφορά στο αντικείμενο κειμένου, το οποίο δημιουργήσαμε προηγουμένως. Στην καρτέλα Hierarchy view επιλέγουμε το αντικείμενο UI Controller και σέρνουμε το αντικείμενο κειμένου (text object) στην υποδοχή στην καρτέλα Inspector. Στη συνέχεια, δηλώνουμε ένα πίνακα συμβολοσειρών για να αποθηκεύσουμε μια σειρά δεδομένων (πληροφορίες για το μοντέλο). Η μεταβλητή της ταχύτητας καθορίζει την ακολουθία που τυπώνονται τα γράμματα στο παράθυρο κειμένου. Τα stringIndex και characterIndex είναι δείκτες σε έναν πίνακα και μια συμβολοσειρά αντίστοιχα. Στη μέθοδο Start() ορίσαμε το παράθυρο κειμένου να είναι κλειστό. Στη μέθοδο Update(), ορίσαμε το τρόπο που θα εμφανίζονται οι πληροφορίες στο παράθυρο κειμένου. Στο τέλος έχουμε ορίσει δύο μεθόδους, οι οποίες θα καλούνται όταν ο χρήστης κάνει κλικ στα αντίστοιχα κουμπιά. Μόνο ένα πράγμα μένει να εξηγήσουμε. Αυτή είναι η μέθοδος DisplayTimer(). Αυτή η συνάρτηση ορίζεται με το IEnumerator και αυτός ο τύπος συνδέεται με την έννοια των κορουτίνων (coroutines).

Από τεχνική άποψη, οι κορουτινές δεν είναι ασύγχρονες (οι ασύγχρονες μέθοδοι δεν εμποδίζουν την εκτέλεση του υπόλοιπου κώδικα), αλλά με την έξυπνη χρήση απαριθμητών (IEnumerator), το Unity κάνει τις κορουτίνες να συμπεριφέρονται παρόμοια με ασύγχρονες μεθόδους. Το μυστικό κρύβεται στη λέξη-κλειδί yield. Αυτή η λέξη επιβάλει την προσωρινή παύση της κορουτίνας, επιστροφή στη ροή του προγράμματος και επανάληψη από το ίδιο σημείο στο επόμενο πλαίσιο. Έτσι, οι κορουτίνες τρέχουν φαινομενικά πίσω από το προγράμμα, μέσω ενός επαναλαμβανόμενου κύκλου εκτελέσιμων εντολών και στη συνέχεια επιστρέφουν στο υπόλοιπο πρόγραμμα. Όπως υποδηλώνει και το όνομα, το StartCoroutine() θέτει σε λειτουργία μια κορουτίνα (coroutine).

Μόλις ξεκινήσει μια κορουτίνα, συνεχίζει να τρέχει μέχρι να ολοκληρωθεί η λειτουργία της. Ένα σημαντικό σημείο είναι ότι η μέθοδος που πέρασε στο StartCoroutine() έχει ένα σύνολο παρενθέσεων που ακολουθούν το όνομα. Αυτή η σύνταξη σημαίνει ότι καλούμε αυτή τη λειτουργία, και δεν έχουμε πέρασμα του ονόματος της. Η καλούμενη μέθοδος τρέχει μέχρι να συναντήσει μια εντολή yield, οπότε η λειτουργία παύει. Η μέθοδος

DisplayTimer() προκαλεί καθυστέρηση στην ταχύτητα που τυπώνονται τα γράμματα στην οθόνη.

Η επόμενη λίστα έχει το πλήρες UIController script.

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

public class UIController : MonoBehaviour
{

    [SerializeField]
    private Camera _camera;

    [SerializeField]
    private RotateMirabello _mirabello;

    [SerializeField]
    private InformationPanelSettings _info;

    [SerializeField]
    private string[] informations;

    [SerializeField]
    private Text _infoPlace;

    float speed = 0.03f;
    int stringIndex = 0;
    int characterIndex = 0;

    IEnumerator DisplayTimer ()
    {
        while (true)
        {
            yield return new WaitForSeconds (speed);
            if (characterIndex > informations[stringIndex].Length)
            {
                continue;
            }
            _infoPlace.text = informations [stringIndex].Substring (0,characterIndex);
```

```
            characterIndex++;
        }
    }

    void Start ()
    {
        _info.Close ();
    }

    void Update()
    {
        if (Input.GetKeyDown (KeyCode.Space))
        {
            if (characterIndex < informations [stringIndex].Length) {
                characterIndex = informations [stringIndex].Length;
            } else if (stringIndex < informations.Length) {
                stringIndex++;
                characterIndex = 0;
            }
        }

        if (stringIndex == informations.Length)
        {
            stringIndex = 0;
            characterIndex = 0;
            _info.Close ();
        }
    }

    public void SelectLeftButton()
    {
        _mirabello.transform.position = new Vector3 (-5, 1, -37);
        _mirabello.transform.localEulerAngles = new Vector3 (364, -3, 181);
        _camera.transform.position = new Vector3 (-27, 3, -12);
        _camera.transform.localEulerAngles = new Vector3 (0, 105, 0);
        Debug.Log ("select left button");
    }

    public void SelectRighttButton()
    {
        _mirabello.transform.position = new Vector3 (-5, 1, -37);
        _mirabello.transform.localEulerAngles = new Vector3 (364, -3, 181);
        _camera.transform.position = new Vector3 (28, 4, -19);
        _camera.transform.localEulerAngles = new Vector3 (0, 292, 0);
```

```
        Debug.Log ("select right button");
    }

    public void SelectFronttButton()
    {
        _camera.transform.position = new Vector3 (-10, -1, -51);
        _camera.transform.localEulerAngles = new Vector3 (0, 0, 0);
        Debug.Log ("select front button");
    }

    public void SelectBacktButton()
    {
        _mirabello.transform.position = new Vector3 (-5, 1, -37);
        _mirabello.transform.localEulerAngles = new Vector3 (364, -3, 181);
        _camera.transform.position = new Vector3 (12, 7, 20);
        _camera.transform.localEulerAngles = new Vector3 (0, 190, 0);
        Debug.Log ("select back button");
    }

    public void SelectUptButton()
    {
        _mirabello.transform.position = new Vector3 (-5, 1, -37);
        _mirabello.transform.localEulerAngles = new Vector3 (364, -3, 181);
        _camera.transform.position = new Vector3 (-8, 12, -41);
        _camera.transform.localEulerAngles = new Vector3 (0, 35, 0);
        Debug.Log ("select up button");
    }

    public void OnOpenInformationPanel()
    {
        _info.Open ();
        StartCoroutine (DisplayTimer ());
        Debug.Log ("information panel open");
    }

    public void OnCloseInformationPanel()
    {
        _info.Close ();
        Debug.Log ("information panel close");
    }
}
```

Λίστα 4.5: Ολόκληρο το UIController script

Κουμπί εξόδου

Στην κάτω δεξιά γωνία υπάρχει ένα κουμπί εξόδου. Όπως εξηγήσαμε σε προηγούμενη ενότητα, ένα αντικείμενο στο Unity αποτελείται από συστατικά (components). Έτσι, δημιουργούμε ένα νέο C# script, το ονομάζουμε UIStartScene και γράφουμε τον κώδικα από την ακόλουθη λίστα.

```
using UnityEngine;
using UnityEngine.SceneManagement;
using System.Collections;

public class UIStartScene : MonoBehaviour
{
   public void exit()
   {
      SceneManager.LoadScene ("start_scene");
   }
}
```

Λίστα 4.6: Ένα μέρος του UIStartScene script

Η εντολή αυτού του κώδικα είναι μια μέθοδος που ονομάζεται Exit(). Μέσα σε αυτή καλούμε μια προκαθορισμένη μέθοδο του Unity, το LoadScene(). Όπως υπονοεί το όνομα, αυτή η μέθοδος μπορεί να φορτώσει διαφορετικές σκηνές. Αλλά τι ακριβώς συμβαίνει όταν φορτώνεται μια σκηνή. Αυτό που συμβαίνει είναι ότι τα πάντα από την αντίστοιχη σκηνή (όλα τα αντικείμενα της σκηνής και συνεπώς όλα τα scripts που συνδέονται με αυτά) καταλαμβάνουν τη μνήμη και στη συνέχεια τα πάντα από τη νέα σκηνή φορτώνονται. Το Unity μας επιτρέπει να καλούμε τη σκηνή μόνο με το όνομά της.

4.4 Μια εναλλακτική διεπαφή χρήστη

Όταν εισαγάγαμε το μοντέλο, επειδή ήταν μεγάλο, το πρόγραμμα το διαίρεσε σε επιμέρους τμήματα. Επισυνάπτουμε ένα mesh collider σε κάθε τμήμα (στην καρτέλα Inspector επιλέγουμε Add Component → Physics → Mesh

Collider). Με αυτό τον τρόπο μπορούμε να κάνουμε κλικ σε οποιοδήποτε σημείο του μοντέλου.

Στη συνέχεια, δημιουργούμε δύο αντικείμενα διεπαφής χρήστη, μια εικόνα και ένα κείμενο. Δημιουργούμε ένα νέο C# script, το ονομάζουμε UIAlternative, γράφουμε τον ακόλουθο κώδικα και το επισυνάπτουμε και στα δύο αντικείμενα.

```
using UnityEngine;
using System.Collections;

public class UIAlternative : MonoBehaviour
{
   void Start ()
   {
      gameObject.SetActive (false);
   }

   public void Hide()
   {
      gameObject.SetActive (false);
   }

   public void Reveal()
   {
      gameObject.SetActive (true);
   }
}
```

Λίστα 4.7: Ολόκληρο το UIAlternative script

Το κύριο μέρος αυτού του script είναι οι δύο μέθοδοι, Hide() και Reveal(). Η πρώτη ενεργοποιεί το αντικείμενο με το οποίο είναι συνδεδεμένο. Η δεύτερη απενεργοποιεί το αντικείμενο με το οποίο είναι συνδεδεμένο.

Στη συνέχεια δημιουργούμε ένα νέο script. Το ονομάζουμε SelectPart και γράφουμε τον κώδικα από την επόμενη λίστα.

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

public class SelectPart : MonoBehaviour
{
    [SerializeField]
    private UIAlternative _uialternative;

    [SerializeField]
    private Text _alternativeText;

    [SerializeField]
    private UIAlternative _alternativeImage;

    void OnMouseDown()
    {
        if (gameObject.tag == "part_0")
        {
            _alternativeText.text = "Mesh_Part_0";
            StartCoroutine (Display ());
            Debug.Log ("tag:" + tag);
        }

        else if (gameObject.tag == "part_1")
        {
            _alternativeText.text = "Mesh_Part_1";
            StartCoroutine (Display ());
            Debug.Log ("tag:" + tag);
        }

        else if (gameObject.tag == "part_2")
        {
            _alternativeText.text = "Mesh_Part_2";
            StartCoroutine (Display ());
            Debug.Log ("tag:" + tag);
        }

        else if (gameObject.tag == "part_3")
        {
            _alternativeText.text = "Mesh_Part_3";
```

```
    StartCoroutine (Display ());
    Debug.Log ("tag:" + tag);
}

else if (gameObject.tag == "part_4")
{
    _alternativeText.text = "Mesh_Part_4";
    StartCoroutine (Display ());
    Debug.Log ("tag:" + tag);
}

else if (gameObject.tag == "part_5")
{
    _alternativeText.text = "Mesh_Part_5";
    StartCoroutine (Display ());
    Debug.Log ("tag:" + tag);
}

else if (gameObject.tag == "part_6")
{
    _alternativeText.text = "Mesh_Part_6";
    StartCoroutine (Display ());
    Debug.Log ("tag:" + tag);
}

else if (gameObject.tag == "part_7")
{
    _alternativeText.text = "Mesh_Part_7";
    StartCoroutine (Display ());
    Debug.Log ("tag:" + tag);
}

else if (gameObject.tag == "part_8")
{
    _alternativeText.text = "Mesh_Part_8";
    StartCoroutine (Display ());
    Debug.Log ("tag:" + tag);
}

else if (gameObject.tag == "part_9")
{
    _alternativeText.text = "Mesh_Part_9";
    StartCoroutine (Display ());
    Debug.Log ("tag:" + tag);
```

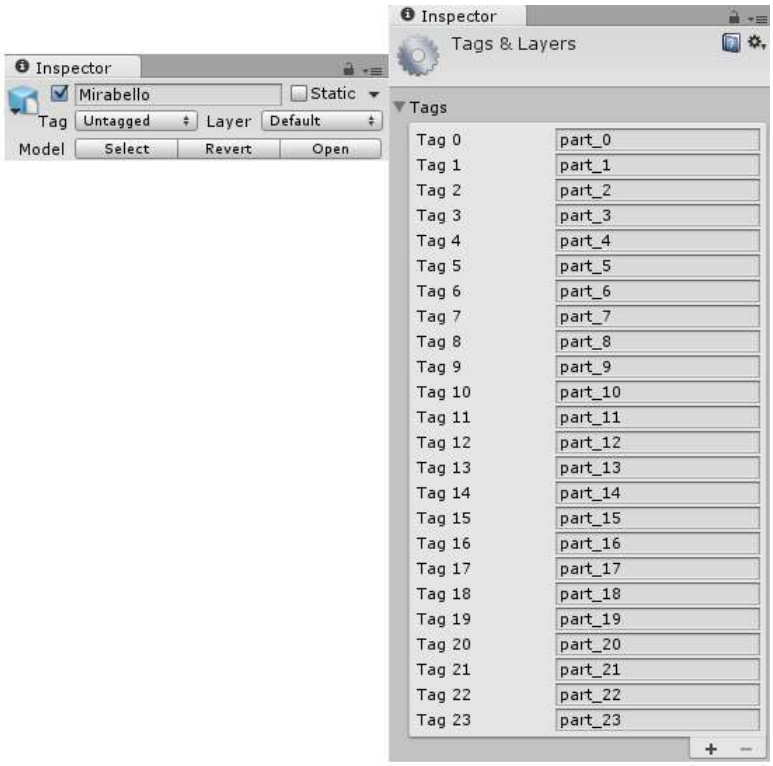
```
}

else if (gameObject.tag == "part_10")
{
   _alternativeText.text = "Mesh_Part_10";
   StartCoroutine (Display ());
   Debug.Log ("tag:" + tag);
}

else if (gameObject.tag == "part_11")
{
   _alternativeText.text = "Mesh_Part_11";
   StartCoroutine (Display ());
   Debug.Log ("tag:" + tag);
}

else if (gameObject.tag == "part_12")
{
   _alternativeText.text = "Mesh_Part_12";
   StartCoroutine (Display ());
   Debug.Log ("tag:" + tag);
}

else if (gameObject.tag == "part_13")
{
   _alternativeText.text = "Mesh_Part_13";
   StartCoroutine (Display ());
   Debug.Log ("tag:" + tag);
}

else if (gameObject.tag == "part_14")
{
   _alternativeText.text = "Mesh_Part_14";
   StartCoroutine (Display ());
   Debug.Log ("tag:" + tag);
}

else if (gameObject.tag == "part_15")
{
   _alternativeText.text = "Mesh_Part_15";
   StartCoroutine (Display ());
   Debug.Log ("tag:" + tag);
}
```

```
else if (gameObject.tag == "part_16")
{
   _alternativeText.text = "Mesh_Part_16";
   StartCoroutine (Display ());
   Debug.Log ("tag:" + tag);
}

else if (gameObject.tag == "part_17")
{
   _alternativeText.text = "Mesh_Part_17";
   StartCoroutine (Display ());
   Debug.Log ("tag:" + tag);
}

else if (gameObject.tag == "part_18")
{
   _alternativeText.text = "Mesh_Part_18";
   StartCoroutine (Display ());
   Debug.Log ("tag:" + tag);
}

else if (gameObject.tag == "part_19")
{
   _alternativeText.text = "Mesh_Part_19";
   StartCoroutine (Display ());
   Debug.Log ("tag:" + tag);
}

else if (gameObject.tag == "part_20")
{
   _alternativeText.text = "Mesh_Part_20";
   StartCoroutine (Display ());
   Debug.Log ("tag:" + tag);
}

else if (gameObject.tag == "part_21")
{
   _alternativeText.text = "Mesh_Part_21";
   StartCoroutine (Display ());
   Debug.Log ("tag:" + tag);
}

else if (gameObject.tag == "part_22")
```

```
        {
            _alternativeText.text = "Mesh_Part_22";
            StartCoroutine (Display ());
            Debug.Log ("tag:" + tag);
        }

        else
        {
            _alternativeText.text = "Mesh_Part_23";
            StartCoroutine (Display ());
            Debug.Log ("tag:" + tag);
        }
    }

    IEnumerator Display()
    {
        Cursor.visible = false;
        _alternativeImage.Reveal ();
        yield return new WaitForSeconds (0.4f);
        _uialternative.Reveal ();
        yield return new WaitForSeconds (3f);
        _uialternative.Hide ();
        yield return new WaitForSeconds (0.25f);
        _alternativeImage.Hide ();
        Cursor.visible = true;
    }
}
```

Λίστα 4.8: Το ολοκληρωμένο SelectPart script

Τώρα, ας εξηγήσουμε τις γραμμές εντολών του κώδικα. Αρχικά, δηλώνουμε τρεις αναφορές. Όπως και στο προηγούμενο τμήμα, πρέπει να σύρουμε και να τοποθετήσουμε τα αντίστοιχα αντικείμενα στην υποδοχή στην καρτέλα Inspector. Στη συνέχεια ορίζουμε μια νέα μέθοδο OnMouseDown(). Αυτή καλείται όταν ο χρήστης κάνει κλικ σε οποιοδήποτε σημείο του 3D μοντέλου. Μέσα σε αυτή, έχουμε μια σειρά από προϋποθέσεις. Προκειμένου να είναι εκτελέσιμος ο κώδικας, πρέπει να ορίσουμε ετικέτες για κάθε τμήμα του μοντέλου. Επιλέγοντας στην καρτέλα Inspector Tag → Add Tag, εμφανίζεται μια κενή λίστα, στην οποία μπορούμε να προσθέσουμε μια νέα ετικέτα (εικόνα 4.15).



Εικόνα 4.15: Πως δημιουργούμε και προσθέτουμε μια ετικέτα (tag)

Τέλος, πρέπει να αναφερθούμε στη μέθοδο Display(). Ο κώδικας μέσα σε αυτή κάνει τον κέρσορα του ποντικιού αόρατο, εμφανίζει το παράθυρο που παρουσιάζονται οι πληροφορίες, το παγώνει για κάποιο χρονικό διάστημα, στη συνέχεια εξαφανίζει το παράθυρο που εμφανίζονται οι πληροφορίες και κάνει τον κέρσορα του ποντικιού ορατό. Σε αυτό το σημείο ολοκληρώθηκε η δυσδιάστατη διεπαφή χρήστη για τη 3D εφαρμογή μας.

ΚΕΦΑΛΑΙΟ 5

ΔΗΜΙΟΥΡΓΩΝΤΑΣ ΒΟΗΘΗΤΙΚΑ ΣΤΟΙΧΕΙΑ

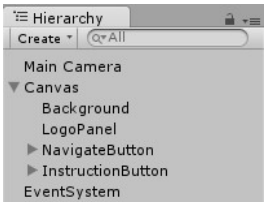
ΚΕΦΑΛΑΙΟ 5

ΔΗΜΙΟΥΡΓΩΝΤΑΣ ΒΟΗΘΗΤΙΚΑ ΣΤΟΙΧΕΙΑ

5.1 Παράθυρο εκκίνησης

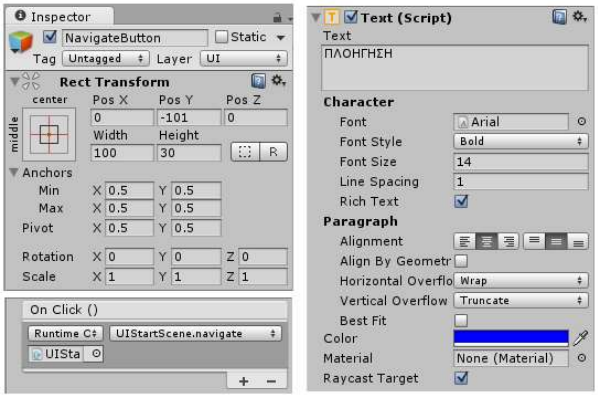
Σε αυτό το κεφάλαιο θα ασχοληθούμε με δύο βοηθητικά στοιχεία, ένα παράθυρο εκκίνησης και ένα παράθυρο οδηγιών. Το πρώτο θα έχει ένα κύριο μενού που θα επιτρέπει στο χρήστη να πλοηγηθεί στο 3D μοντέλο ή θα τον κατευθύνει σε ένα παράθυρο οδηγιών. Το δεύτερο θα εξηγεί στο χρήστη πώς να χειριστεί τη λειτουργικότητα της εφαρμογής. Ας δούμε τώρα πώς τα δημιουργούμε.

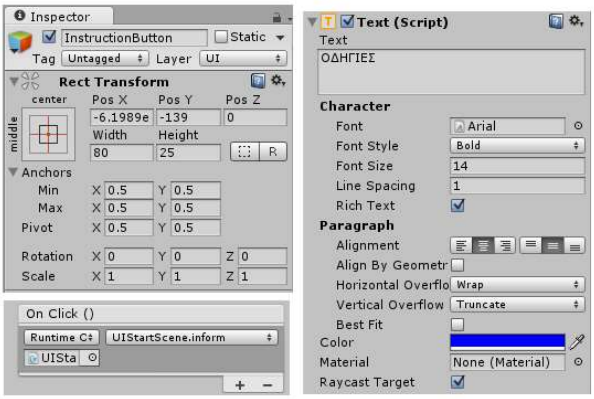
Το κύριο μενού είναι μια γραφική διεπαφή χρήστη (GUI), η οποία αποτελείται από δύο αντικείμενα εικόνας και δύο αντικείμενα κουμπιών. Για να τα δημιουργήσουμε, επιλέγουμε Create → UI → Image ή Button. Η καρτέλα Hierarchy view μοιάζει με την ακόλουθη εικόνα.



Εικόνα 5.1: Η καρτέλα Hierarchy view δείχνει τα αντικείμενα εικόνας και κουμπιού

Οι ρυθμίσεις των δύο κουμπιών (Navigate και Instruction Button) παρουσιάζονται στη παρακάτω εικόνα.





Εικόνα 5.2: Ρυθμίσεις για τα αντικείμενα κουμπιά (button objects)

Πρέπει να κάνουμε μερικές ρυθμίσεις στα αντικείμενα εικόνας (Background και LogoPanel) όπως φαίνεται στην επόμενη εικόνα.

Εικόνα 5.3: Ρυθμίσεις για τα αντικείμενα εικόνας

Για να γίνουν λειτουργικά τα κουμπιά, πρέπει να κάνουμε μερικές ρυθμίσεις στο UIStartScene script που δημιουργήσαμε σε προηγούμενη ενότητα.

```
public void navigate()
{
    SceneManager.LoadScene ("scene_1");
}
```

```
public void inform()
{
    SceneManager.LoadScene ("instruction_scene");
}
```

Λίστα 5.1: Ρυθμίσεις του UIStartScene script

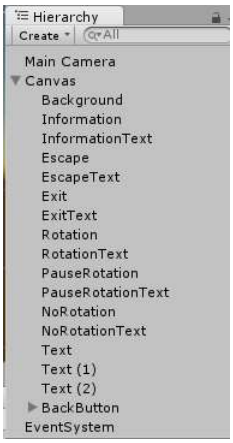
Το απόσπασμα του κώδικα είναι δύο μέθοδοι που καλούνται Navigate() και Inform(). Μέσα σε αυτές καλούμε μια προκαθορισμένη μέθοδο από το Unity, τη μέθοδο LoadScene(). Αλλά ας δούμε τι γίνεται όταν φορτώνεται μια σκηνή. Αυτό που συμβαίνει είναι ότι τα πάντα από την αντίστοιχη σκηνή (όλα τα αντικείμενα στη σκηνή και συνεπώς όλα τα scripts που συνδέονται με αυτά) καταλαμβάνουν τη μνήμη και στη συνέχεια τα πάντα από τη νέα σκηνή φορτώνονται. Το Unity μας επιτρέπει να καλέσουμε μια σκηνή μόνο με το όνομά της. Τώρα, αν επιλέξουμε Play κάθε κουμπί φορτώνει τη σωστή σκηνή.

Το κύριο μενού παρουσιάζεται στην ακόλουθη εικόνα.

Εικόνα 5.4: Το κύριο μενού της εφαρμογής

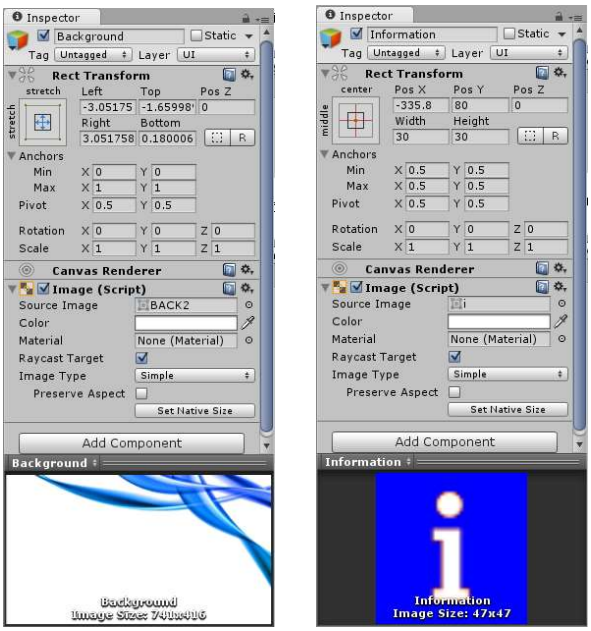
5.2 Παράθυρο οδηγιών

Το παράθυρο οδηγιών είναι μια γραφική διεπαφή χρήστη που αποτελείται από αντικείμενα εικόνας, κειμένου και κουμπιά. Για να τα δημιουργήσουμε, επιλέγουμε Create → UI → Image ή Text ή Button. Η καρτέλα Hierarchy view μοιάζει με την ακόλουθη εικόνα.



Εικόνα 5.5: Η καρτέλα Hierarchy view δείχνει τα αντικείμενα εικόνας, κειμένου και κουμπιού

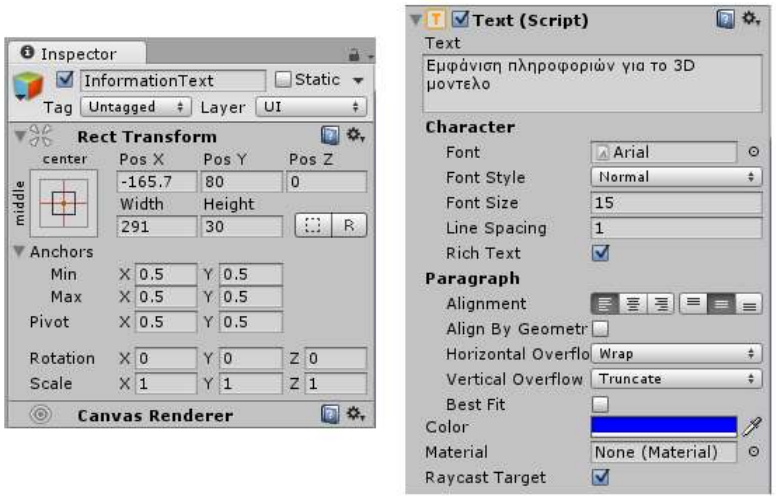
Υπάρχει μια σειρά ρυθμίσεων που πρέπει να κάνουμε. Αυτές εμφανίζονται στην επόμενη εικόνα.



Εικόνα 5.6: Ρυθμίσεις στα αντικείμενα εικόνας

Οι εικόνες (Information, Escape, Exit, Rotation, PauseRotation, NoRotation όπως εμφανίζονται στην προηγούμενη εικόνα) έχουν τις ίδιες ρυθμίσεις. Τα αντικείμενα κειμένου έχουν την ίδια λογική με αυτή που παρουσιάστηκε στο

προηγούμενο κεφάλαιο. Χρειάζεται μόνο να εφαρμόσουμε τις αλλαγές που παρουσιάζονται στην παρακάτω εικόνα.



Εικόνα 5.7: Ρυθμίσεις στα αντικείμενα κειμένου

Λίγες ρυθμίσεις παραμένουν για να ολοκληρωθεί η λειτουργικότητα του παράθυρου οδηγιών. Πρέπει να προσθέσουμε στο UIStartScene script μια μέθοδο για να γίνει το κουμπί Πίσω λειτουργικό.

```
public void back()
{
    SceneManager.LoadScene ("start_scene");
}
```

Λίστα 5.2: Ρυθμίσεις για το UIStartScene script

Ολόκληρος ο κώδικας για το UIStartScene script παρουσιάζεται παρακάτω.

```
using UnityEngine;
using UnityEngine.SceneManagement;
using System.Collections;

public class UIStartScene : MonoBehaviour
```

```
{
    public void navigate()
    {
        SceneManager.LoadScene ("scene_1");
    }

    public void inform()
    {
        SceneManager.LoadScene ("instruction_scene");
    }

    public void back()
    {
        SceneManager.LoadScene ("start_scene");
    }

    public void exit()
    {
        SceneManager.LoadScene ("start_scene");
    }
}
```

Λίστα 5.3: Το ολοκληρωμένο UIStartScene script

Το παράθυρο οδηγιών παρουσιάζεται στην ακόλουθη εικόνα.

Εικόνα 5.8: Το παράθυρο οδηγιών της εφαρμογής

ΒΙΒΛΙΟΓΡΑΦΙΑ

https://www.introprogramming.info/wp-content/uploads/2013/07/Books/CSharpEn/Fundamentals-of-Computer-Programming-with-CSharp-Nakov-eBook-v2013.pdf

https://docs.unity3d.com/Manual/index.html

https://books.google.gr/books?id=ufFnDQAAQBAJ&pg=PA55&lpg=PA55&dq=Mirabello+Church&source=bl&ots=aLkj94wurt&sig=d2r5tm8p9fCJ3nE2IY6mptrOt3I&hl=el&sa=X&ved=0ahUKEwiG5onm4cDYAhXJ1ywKHaHXCWkQ6AEIfjAH#v=onepage&q=Mirabello%20Church&f=false

https://el.wikipedia.org/wiki/%CE%A0%CE%BF%CE%BB%CE%B9%CF%84%CE%B9%CF%83%CF%84%CE%B9%CE%BA%CE%AE_%CE%9A%CE%BB%CE%B7%CF%81%CE%BF%CE%BD%CE%BF%CE%BC%CE%B9%CE%AC