ΕΘΝΙΚΟ ΜΕΤΣΟΒΙΟ ΠΟΛΥΤΕΧΝΕΙΟ
ΣΧΟΛΗ ΑΓΡΟΝΟΜΩΝ ΚΑΙ ΤΟΠΟΓΡΑΦΩΝ ΜΗΧΑΝΙΚΩΝ
ΤΟΜΕΑΣ ΤΟΠΟΓΡΑΦΙΑΣ
ΕΡΓΑΣΤΗΡΙΟ ΦΩΤΟΓΡΑΜΜΕΤΡΙΑΣ

ΔΙΠΛΩΜΑΤΙΚΗ ΕΡΓΑΣΙΑ

ΤΙΤΛΟΣ:

«ΑΝΑΠΤΥΞΗ 3D ΕΙΚΟΝΙΚΟΥ ΠΕΡΙΒΑΛΛΟΝΤΟΣ ΓΙΑ ΤΗΝ ΠΕΡΙΗΓΗΣΗ ΚΑΙ ΤΗ ΣΗΜΑΝΣΗ ΑΠΤΩΝ ΠΟΛΙΤΙΣΤΙΚΩΝ ΔΕΔΟΜΕΝΩΝ»

Δαβερώνα Άννα Χριστίνα

Επιβλέπων: Δουλάμης Αναστάσιος, Λέκτορας Ε.Μ.Π.

Αθήνα, Μάρτιος 2018

Ευχαριστίες

Θα ήθελα να ευχαριστήσω τον κύριο Αναστάσιο Δουλάμη, για το επίκαιρο και με πληθώρα εφαρμογών θέμα που μου ανέθεσε καθώς επίσης για την καθοδήγηση, τις ουσιώδεις συμβουλές και το ενδιαφέρον που επέδειξε κατά την εκπόνηση της διπλωματικής μου εργασίας.

Επίσης θα ήθελα να ευχαριστήσω τον κύριο Νικόλαο Δουλάμη, για τις πολύτιμες συμβουλές του κατά τη διάρκεια της συγγραφής της παρούσας διπλωματικής.

Θα ήθελα να ευχαριστήσω την οικογένεια και τα οικεία μου πρόσωπα που με στηρίζουν όλα αυτά τα χρόνια κατά τη διάρκεια των σπουδών μου.

Τέλος, θα ήθελα να απευθύνω τις ευχαριστίες μου στα μέλη της Εξεταστικής Επιτροπής, οι οποίοι ευγενικά δέχθηκαν να αξιολογήσουν την παρούσα διπλωματική εργασία.

Πίνακας Περιεχομένων

Πίνακας Σχημάτων

Περίληψη

Σε αυτή την εργασία παρουσιάζονται τα βήματα καθώς και η μεθοδολογία που ακολουθήθηκε για τη δημιουργία μίας εφαρμογής, η οποία δίνει τη δυνατότητα στο χρήστη να περιηγηθεί σε ένα τρισδιάστατο μοντέλο καθώς επίσης να αποθηκεύσει και να παρουσιάσει πληροφορίες για αυτό.

Αρχικά, αναλύονται τα τρισδιάστατα μοντέλα και η σημαντικότητά τους. Επίσης αναφέρονται κάποιες από τις μεθόδους δημιουργίας τους καθώς και οι τομείς εφαρμογής τους.

Στη συνέχεια, γίνεται παρουσίαση της εφαρμογής που δημιουργήθηκε καθώς και των δεδομένων που χρησιμοποιήθηκαν και αναλύονται οι δυνατότητες που δίνονται στο χρήστη κατά τη διάρκεια χρήσης της εφαρμογής.

Επιπλέον παρουσιάζονται τα λογισμικά που χρησιμοποιήθηκαν, τα οποία ήταν το Unity και το Microsoft Visual Studio.

Ακόμη περιγράφεται η διαδικασία που ακολουθήθηκε για τη δημιουργία της εφαρμογής και γίνεται επεξήγηση του κώδικα που δημιουργήθηκε για τις διάφορες λειτουργίες της.

Τέλος, παρουσιάζονται οι προοπτικές εξέλιξης της παρούσας εφαρμογής και αναφέρονται οι επιπλέον λειτουργίες που θα μπορούσαν να συμπεριληφθούν σε μία τέτοιου είδους εφαρμογή.

Abstract

This diploma thesis presents the steps and the methodology followed to create an application that enables the user to browse a 3D model as well as store and present information about it.

Initially, three-dimensional models and their significance are analyzed. Also, some of their creation methods and their application areas are mentioned.

Subsequently, a presentation of the created application and of the data used is presented, and the capabilities given to the user during the use of the application are analyzed.

In addition, there is the representation of the softwares that were used such as Unity and Microsoft Visual Studio.

It also describes the procedure followed for creating the application and explaining the code created for its various functions.

Finally, the prospects for the development of the present application are presented and the additional functions that could be included in such application.

Εισαγωγή

Ο κόσμος μέσα στον οποίο ζούμε και επομένως αυτόν αντιλαμβανόμαστε είναι τουλάχιστον τρισδιάστατος.

Η εμφάνιση και η 3D παρουσίαση ενός μοντέλου μέσα από την οθόνη ενός Η/Υ, φέρνει και το ευρύ κοινό πιο κοντά στο να μπορεί να δει και να κατανοήσει το οτιδήποτε τον ενδιαφέρει.

Μία από τις κυριότερες επιστήμες που ασχολούνται με την καταγραφή της τρισδιάστατης πληροφορίας είναι η φωτογραμμετρία η οποία στοχεύει στην ανακατασκευή της 3D γεωμετρίας ενός αντικειμένου που βασίζεται αποκλειστικά σε 2D φωτογραφίες. Αυτό έχει υπάρξει ενεργό θέμα έρευνας στον τομέα της τοπογραφίας, της πληροφορικής και της όρασης υπολογιστών για πολλά χρόνια. Σήμερα, τα αποτελέσματα της έρευνας σε συνδυασμό με μια τεράστια αύξηση στη διαθεσιμότητα των υπολογιστικών πόρων σε προσιτή τιμή, δημιούργησαν ορισμένες τεχνικές και λογισμικά που αποδεικνύουν την ωριμότητα της τεχνολογίας. Υπάρχει μεγάλο δυναμικό για εφαρμογές που χρησιμοποιούν αυτές τις τεχνικές σε ένα ευρύ φάσμα πεδίων, για παράδειγμα, μοντελοποίηση εδάφους, ανακατασκευή αρχαιολογικών μνημείων ή ολόκληρων πόλεων, κτίρια εικονικών κόσμων σε παιχνίδια υπολογιστών κ.λπ. [1].

a) Σκοπός Εργασίας

Σκοπός αυτής της Διπλωματικής Εργασίας, είναι η δημιουργία μίας εφαρμογής που θα περιέχει τρισδιάστατα μοντέλα, στα οποία ο χρήστης θα μπορεί να περιηγείται ελεύθερα και να τοποθετεί ετικέτες, με σκοπό την αποθήκευση πληροφοριών για το αντικείμενο. Επίσης ο χρήστης θα έχει τη δυνατότητα εισαγωγής δικού του τρισδιάστατου μοντέλου.

Στόχος είναι η ανάδειξη των πληροφοριών μέσω της σήμανσης του μοντέλου, καθώς επίσης και η περιήγηση σε ένα τρισδιάστατο εικονικό περιβάλλον.

b) Δομή της Εργασίας

Στο Πρώτο Κεφάλαιο, αναλύονται τα τρισδιάστατα μοντέλα. Πιο συγκεκριμένα αναφέρονται οι μέθοδοι δημιουργίας τους καθώς και οι τομείς εφαρμογής τους, ιδιαίτερα σε ότι αφορά το αντικείμενο του μηχανικού.

Στο Δεύτερο Κεφάλαιο, γίνεται μία αναλυτική παρουσίαση της εφαρμογής που δημιουργήθηκε. Αναφέρονται όλες οι λειτουργίες της καθώς και οι δυνατότητες που δίνει στο χρήστη.

Στο Τρίτο Κεφάλαιο, παρουσιάζονται τα λογισμικά που χρησιμοποιήθηκαν. Τα λογισμικά αυτά ήταν δύο. Το Unity, το οποίο είναι σχεδιασμένο για τη δημιουργία και την ανάπτυξη βιντεοπαιχνιδιών και το Microsoft Visual Studio το οποίο είναι ένα ολοκληρωμένο περιβάλλον ανάπτυξης εφαρμογών και στο οποίο έγινε η ανάπτυξη του κώδικα της εφαρμογής.

Στο Τέταρτο Κεφάλαιο της εργασίας αυτής, αναπτύσσεται η μεθοδολογία που ακολουθήθηκε και αναλύονται τα βήματα για την ανάπτυξη της εφαρμογής. Επιπλέον γίνεται αναλυτική επεξήγηση του κώδικα και των λειτουργιών της εφαρμογής.

Κεφάλαιο 1.

Τρισδιάστατα Μοντέλα

Στην επιστήμη των υπολογιστών, με τον όρο τρισδιάστατο μοντέλο, εννοούμε τη μαθηματική αναπαράσταση ενός πραγματικού ή φανταστικού τρισδιάστατου αντικειμένου [2].

Τις τελευταίες δύο δεκαετίες, χάρη στην εξέλιξη της τεχνολογίας εξελίχθηκε και η τρισδιάστατη καταγραφή. Με την καταγραφή τρισδιάστατων συντεταγμένων, δημιουργούνται τρισδιάστατα μοντέλα η χρήση των οποίων έχει εφαρμογή σε πολλούς διαφορετικούς τομείς. Η έρευνα στην φωτογραμμετρία μαζί με την ευρεία διαθεσιμότητα προσιτών υπολογιστικών πόρων, έχουν οδηγήσει στην παραγωγή ολοκληρωμένων εφαρμογών σε τομείς όπως τη μοντελοποίηση του εδάφους, ανακατασκευή αρχαιολογικών μνημείων αλλά και ολόκληρων πόλεων, καθώς και την κατασκευή εικονικών κόσμων [3].

Η καταγραφή αυτή, γίνεται με συσκευές σάρωσης, οι οποίες μπορούν να καταγράψουν εκατοντάδες ακόμη και χιλιάδες τρισδιάστατα σημεία ανά δευτερόλεπτο. Ο υπολογισμός των τρισδιάστατων συντεταγμένων ενός αντικειμένου σε πραγματικό χρόνο, μπορεί να γίνει με ποικιλία διαφορετικών μεθόδων αλλά και οργάνων [4]. Έτσι η αποτύπωση του αντικειμένου μπορεί να γίνει ακολουθώντας φωτογραμμετρικές αλλά και τοπογραφικές μεθόδους, χρησιμοποιώντας από μία απλή φωτογραφική μηχανή μέχρι και σαρωτή laser.

1.1 Δημιουργία Τρισδιάστατων Μοντέλων

a) Στάδια Δημιουργίας Τρισδιάστατων Μοντέλων

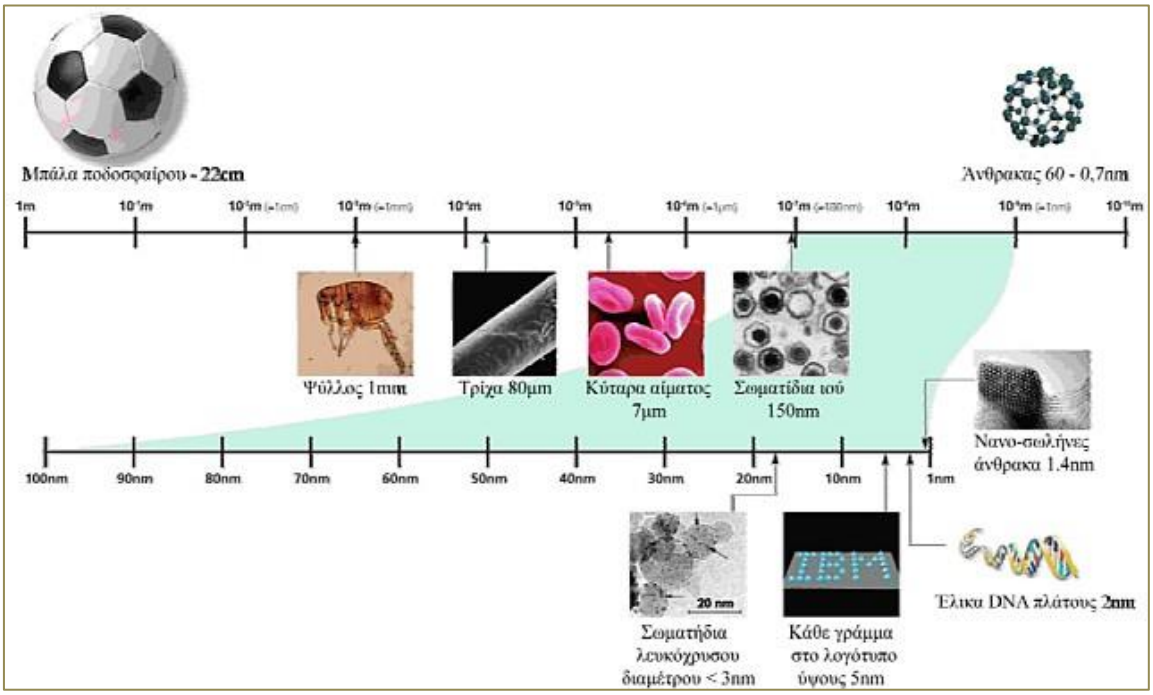
Η τρισδιάστατη ψηφιοποίηση ενός αντικειμένου, αποτελεί μία πολύπλοκη διαδικασία, η οποία αποτελείται από τρία στάδια. Το πρώτο στάδιο είναι η προετοιμασία. Στην φάση αυτή λαμβάνονται ορισμένες αποφάσεις όπως η τεχνική και η μεθοδολογία που θα ακολουθηθεί για την ψηφιοποίηση κ.α.. Στο δεύτερο στάδιο, γίνεται η ψηφιακή καταγραφή ενώ στο τρίτο η επεξεργασία των δεδομένων, η οποία περιλαμβάνει τη μοντελοποίηση του ψηφιοποιημένου αντικειμένου [5].

b) Τεχνικές Δημιουργίας Τρισδιάστατων Μοντέλων

Οι τεχνικές για τη δημιουργία ενός τρισδιάστατου μοντέλου είναι πολλές και διακρίνονται σε περιπτώσεις ανάλογα με το μέγεθος του αντικειμένου που πρόκειται να ψηφιοποιηθεί. Στον Πίνακας 1.1, αναγράφονται οι κατηγορίες αυτές. Στην Εικόνα 1.1, γίνεται μία ενδεικτική συγκριτική απεικόνιση χαρακτηριστικών αντικειμένων, γνωστών μμεγεθών από 1 μέτρο έως 1 μικρόμετρο.

Μέγεθος	Διάσταση
Μικροσκοπικό	~100(nm)
Μικρό	10x10(cm)
Μεσαίο	0.1-2(m)
Μεγάλο	>2(m)

Πίνακας 1.1: Κατηγορίες Μεγεθών Αντικειμένων προς Ψηφιοποίηση.



Εικόνα 1.1: Συγκριτικό γράφημα διαστάσεων πραγματικών αντικειμένων
Πηγή: http://www.ipet.gr/digitech2/index.php?option=com_content&task=view&id=59&Itemid=53

Οι τεχνικές για τη δημιουργία του τρισδιάστατου μοντέλου ποικίλουν ανάλογα με το μέγεθός του. Παρακάτω αναφέρονται οι μέθοδοι ανά κατηγορία αντικειμένου.

Πιο συγκεκριμένα η δημιουργία μικροσκοπικού αντικειμένου, γίνεται με τη χρήση μίας από τις παρακάτω τεχνικές:

- Atomic Force Microscopy
- Stereo Scanning Electron Microscopy
- Confocal Microscopy
- White Light Scanning Interferometry

Με τη χρήση των δύο πρώτων τεχνικών είναι δυνατή η αναγνώριση και η καταγραφή χαρακτηριστικών σε κλίμακα μερικών δεκάδων νανομέτρων(nm), ενώ οι δύο άλλες παρέχουν ανάλυση της κλίμακας μερικών εκατοντάδων νανομέτρων [6].

Σε ότι αφορά την ψηφιοποίηση μικρών αντικειμένων, χρησιμοποιούνται τεχνικές οι οποίες βασίζονται σε συστήματα τριγωνοποίησης ακτινών laser. Τα συστήματα αυτά, είναι

συνήθως φορητά, σχετικώς φθηνά και εμπορικά διαθέσιμα. Συνήθως, χρησιμοποιούνται για τη μέτρηση αντικειμένων όπως νομίσματα και μετάλλια, εγχάρακτες ή ανάγλυφες ταμπέλες και μικρά αντικείμενα χρυσοχοΐας, με ακρίβεια μέτρησης της τάξης μερικών μικρομέτρων.

Η τρίτη κατηγορία, περιέχει τις πιο πολλές τεχνικές ψηφιοποίησης μεσαίων αντικειμένων [5]:

- Σάρωση με λέιζερ
- Σχήμα από δομημένο φως
- Σχήμα από σιλουέτα
- Σχήμα από στερεοσκοπική φωτογράφιση
- Σχήμα από βίντεο
- Σχήμα από σκιά
- Σχήμα από υφή
- Σχήμα από φωτομετρική στερεοφωτογράφιση
- Σχήμα από μεταβαλλόμενη εστίαση
- Σχήμα από σκιά
- Σύστημα μέτρησης συντεταγμένων

Στην τελευταία κατηγορία, αυτή των μεγάλων αντικειμένων, περιέχονται κυρίως μνημεία αλλά και γενικότερα ανθρώπινες κατασκευές. Οι πιο ενδεδειγμένες μέθοδοι είναι οι παρακάτω [7]:

- Με γεωδαιτικούς σταθμούς, οι οποίοι παράγουν τρισδιάστατες συντεταγμένες
- Με ψηφιακή επεξεργασία εικόνων
- Με συσκευές σάρωσης, οι οποίες παράγουν σύννεφο σημείων

Η επιλογή της κατάλληλης μεθόδου εξαρτάται από το σκοπό της δημιουργίας του μοντέλου. Κάποιοι από τους παράγοντες που επηρεάζουν είναι το μέγεθος του αντικειμένου, η ακρίβεια του τελικού αποτελέσματος, το κόστος κ.α..

1.2 Τομείς Εφαρμογής Τρισδιάστατων Μοντέλων

Τα 3D μοντέλα αποτελούν ανεκτίμητο εργαλείο σε πολλούς τομείς, όπως η ιατρική, ο βιομηχανικός σχεδιασμός, η αρχιτεκτονική, ο κινηματογράφος, τα video games κ.α. [2]. Σε αυτή την παράγραφο παρουσιάζονται και περιγράφονται κάποιοι από τους σημαντικότερους τομείς στους οποίους μπορεί να χρησιμοποιηθεί ένα τρισδιάστατο μοντέλο.

Τα τρισδιάστατα μοντέλα είναι βασικό εργαλείο για την αποτύπωση, τη διατήρηση και γενικότερα την προστασία της Πολιτιστικής Κληρονομιάς. Πολιτιστική Κληρονομιά είναι μία

έννοια που περιλαμβάνει τα μνημεία, αλλά και κάθε είδους τεκμήριο πολιτισμού. Είναι φορέας ιστορικής μνήμης και κιβωτός εθνικού και παγκόσμιου πολιτισμού και μπορεί να είναι άυλη ή υλική. Στην άυλη συμπεριλαμβάνονται ιδέες, παραδόσεις, ποίηση, λογοτεχνία, τραγούδια, χοροί, εκδηλώσεις, έθιμα και χειροτεχνίες ενώ στην υλική μνημεία αλλά και καλλιτεχνήματα.

Η Πολιτιστική Κληρονομιά κινδυνεύει γιατί εύκολα καταστρέφεται, χάνεται, αλλοιώνεται και ξεχνιέται. Κάποιοι από τους λόγους είναι οι φυσικές καταστροφές, οι βίαιες ενέργειες όπως πόλεμοι, τρομοκρατικές ενέργειες κ.τ.λ., οι κλοπές και οι βανδαλισμοί, η οικοδομική δραστηριότητα, η παγκοσμιοποίησης, ο σύγχρονος τρόπος ζωής και η αδιαφορία, η αύξησης του αστικού πληθυσμού έναντι του αγροτικού κ.α..

Η προστασία της πολιτιστικής κληρονομιάς είναι υποχρέωση όλων των γενεών προς τις επόμενες. Αυτό μπορεί να συμβεί με διάφορους τρόπους όπως η συντήρηση για την αποφυγή περαιτέρω υλικής καταστροφής, η ανακατασκευή επαναφέροντας στην αρχική μορφή με προσθήκη νέων μελών, η αναστήλωση επαναφέροντας στην αρχική μορφή με αυθεντικά υλικά, η αποκατάσταση διατηρώντας τις αξιόλογες φάσεις και προσθήκες, η ανάδειξη μέσω της προβολής και της οργανικής ένταξης στο χώρο, η εξυγίανση αποφεύγοντας βλαβερές επιδράσεις, η αναβίωση αποδίδοντας λειτουργίες και χρήσεις.

Έτσι για την προστασία της, είναι αναγκαία η γεωμετρική τεκμηρίωση (Χάρτα της Βενετίας, 1964). Η Γεωμετρική τεκμηρίωση είναι ένα υποσύνολο της συνολικής ολοκληρωμένης τεκμηρίωσης ενός μνημείου (βιβλιογραφικής, ιστορικής, αρχαιολογικής, αρχιτεκτονικής, χαρτογραφικής, νομικής κ.τ.λ.) [8].

Με τον όρο ”Γεωμετρική Τεκμηρίωση Μνημείου” ορίζεται η καταγραφή της θέσης, του μεγέθους και της μορφής ενός μνημείου σε μια συγκεκριμένη χρονική στιγμή, στο χώρο των τριών διαστάσεων. Σκοπός είναι η παρουσίαση της κατάστασης στην οποία βρίσκεται τη χρονική στιγμή της τεκμηρίωσης. Έτσι είναι δυνατή η παρουσίαση των στοιχείων λεπτομερειών που περιλαμβάνονται σε αυτή και ο εντοπισμός πιθανών κατασκευαστικών αποκλίσεων ή φθορών που έχει υποστεί. Η ακριβής μετρική καταγραφή του μνημείου δίνει τη δυνατότητα και σε άλλους ειδικούς επιστήμονες (στατικούς, αρχιτέκτονες, αρχαιολόγους, ιστορικούς) να εκτιμήσουν τη σοβαρότητα των αποκλίσεων ή φθορών και να προτείνουν τις μεθόδους και διαδικασίες παρέμβασης για την αποκατάστασή τους. [9].

Μια σειρά συμβάσεων και συμφωνιών από διεθνείς οργανισμούς (UNESCO, ICOMOS, κλπ.) αναφέρονται στη σπουδαιότητα της τεκμηρίωσης, αλλά και στις απαραίτητες ενέργειες με σκοπό την αποτύπωση, αρχειοθέτηση, διαχείριση, διάθεση δεδομένων κλπ., με προεξέχουσα τη Χάρτα της Βενετίας, που συντάχθηκε το 1964, και πιο συγκεκριμένα τα άρθρα 2 και 16,σύμφωνα με τα οποία ορίζονται τα παρακάτω [10]:

- Άρθρο 2: «η συντήρηση και η αποκατάσταση των μνημείων αποτελεί έναν επιστημονικό κλάδο ο οποίος πρέπει να αποτείνεται στη συνεργασία όλων των επιστημών και όλων των τεχνών που μπορούν να συνεισφέρουν στη μελέτη και τη διάσωση της πολιτιστικής κληρονομιάς».
- Άρθρο 16: «οι εργασίες συντήρησης, αποκατάστασης και ανασκαφής θα πρέπει να βασίζονται σε εξακριβωμένη τεκμηρίωση, δηλαδή σε αναλυτικές και κριτικές εκθέσεις, εικονογραφημένες με σχέδια και φωτογραφίες...»[11].

Ένα τρισδιάστατο μοντέλο όμως μπορεί να χρησιμοποιηθεί και σε περισσότερες εφαρμογές μηχανικών. Για έναν μηχανικό η ανταλλαγή της πληροφορίας μέχρι πρόσφατα γινόταν σε δύο διαστάσεις λόγω των περιορισμών που έθετε η τεχνολογία. Τώρα πια όμως, με την εξέλιξη της τεχνολογίας είναι δυνατή η ανταλλαγή πληροφορίας σε καθαρή τρισδιάστατη μορφή. Αυτό ανοίγει πολλές νέες προοπτικές όσον αφορά τη μελέτη σε κάθε αντικείμενο μηχανικού καθώς η εποπτεία της οποιασδήποτε κατασκευής είναι πλέον άμεση και στις τρεις διαστάσεις. Παρακάτω αναφέρονται κάποιες από τις καινούριες δυνατότητες που δίνονται όσον αφορά τη μελέτη:

- Καλύτερη εποπτεία του έργου
- Δυναμικές προμετρήσεις και προϋπολογισμοί
- Εξομοιώσεις με μεγαλύτερο βαθμό ασφάλειας
- Μείωση κόστους κατασκευής

Η οπτικοποίηση ενός αντικειμένου και η παραγωγή του τρισδιάστατου μοντέλου του, είναι ένας πολύ χρήσιμος τρόπος παρουσίασής του για διάφορους τομείς της επιστήμης και της τεχνολογίας όπως είναι η τοπογραφία, η οδοποιία, η αρχιτεκτονική, η χωροταξία κ.α..

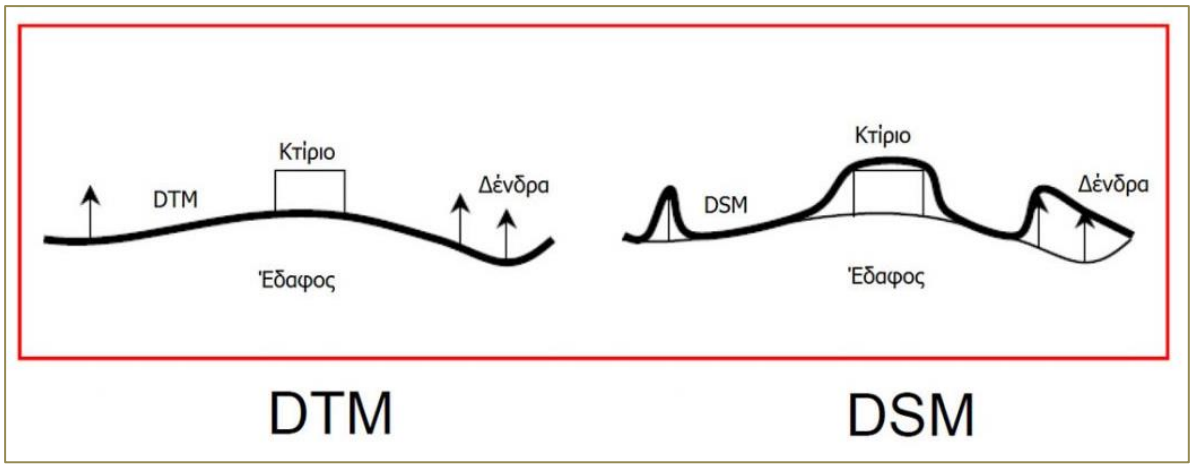
Οι περιπτώσεις που είναι πλησιέστερα στο αντικείμενο του μηχανικού αφορούν κυρίως αναπαραστάσεις

- ψηφιακών μοντέλων εδάφους,
- αστικών περιοχών,
- αρχιτεκτονικών σχεδίων και
- μελλοντικών αρχιτεκτονικών εφαρμογών (ανέγερση νέων κτιρίων, αναπλάσεις, αναπαλαιώσεις κ.λπ.),

Τα παραπάνω μοντέλα, μπορούν να χρησιμοποιηθούν και για τη δημιουργία άλλων προϊόντων, όπως για παράδειγμα σε προσομοιώσεις πτήσεων για την εκπαίδευση πιλότων.

Πιο συγκεκριμένα στον τομέα της Χαρτογραφίας, η δημιουργία τρισδιάστατων μοντέλων είναι ένας από τους βασικούς στόχους της, έτσι ώστε να είναι η όσο το δυνατόν πιο κατανοητή στον άνθρωπο απεικόνιση του φυσικού ανάγλυφου. Έτσι μπορεί κανείς να αντλήσει πληροφορίες με μία πιο ρεαλιστική αναπαράσταση της περιοχής που απεικονίζεται [12].

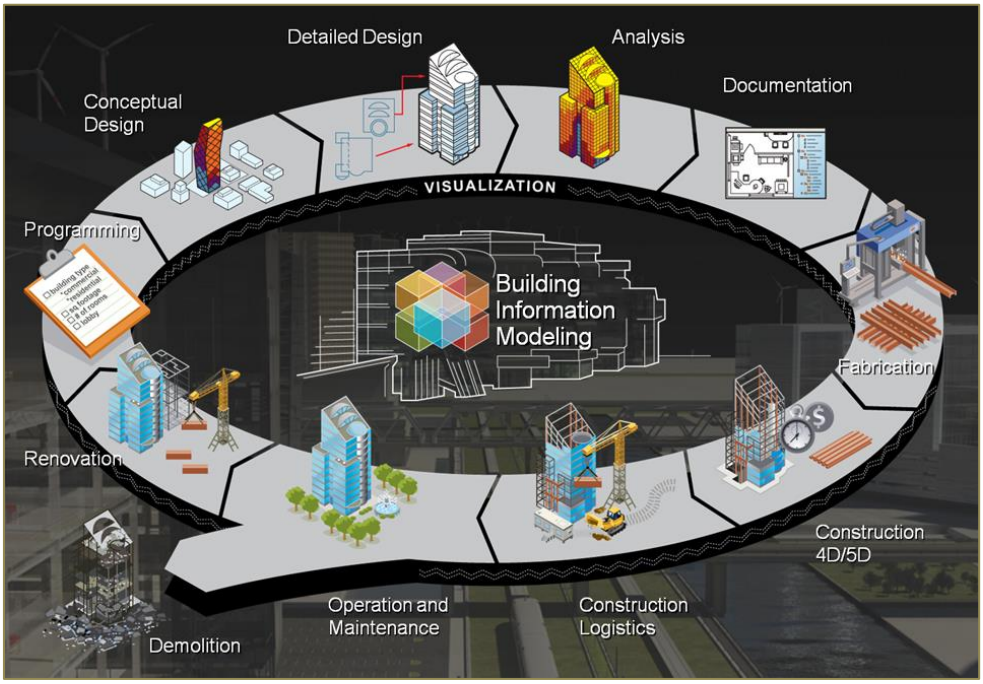
Η τρισδιάστατη απεικόνιση του αναγλύφου, γίνεται με τη δημιουργία του Ψηφιακού Μοντέλου Εδάφους (ΨΜΕ ή DTM). Ένα Ψηφιακό Μοντέλο Εδάφους είναι μια ψηφιακή αναπαράσταση τμήματος της φυσικής γήινης επιφάνειας, η οποία δημιουργείται από ένα σύνολο σημείων σε ένα τρισδιάστατο σύστημα αναφοράς (X,Y,Z), το οποίο προσεγγίζει την πραγματική επιφάνεια. Όταν συμπεριλαμβάνεται στην πληροφορία και το υψόμετρο του εδάφους, τότε το μοντέλο που δημιουργείται το Ψηφιακό Μοντέλο Υψομέτρου (DEM), ενώ όταν δημιουργείται μοντέλο μίας δομημένης περιοχής, να απεικονίζονται επίσης όλες οι κατασκευές που υπάρχουν σ' αυτήν, παράγοντας ένα Τρισδιάστατο Μοντέλο Επιφανείας (DSM)(Εικόνα 1.2) [13][14].



Εικόνα 1.2: Διαφορά μεταξύ DTM και DSM.
Πηγή: http://www.terra.gr/el/3d-data-3d-building-dtm/

Μία άλλη χρήση ενός τρισδιάστατου ψηφιακού μοντέλου είναι στην αρχιτεκτονική. Η τρισδιάστατη απεικόνιση ενός αρχιτεκτονικού σχεδίου είναι μία σχετικά απλή διαδικασία σε σχέση με την παραγωγή αρχιτεκτονικής μακέτας με αποτέλεσμα να τείνει να την αντικαταστήσει. Το τρισδιάστατο ρεαλιστικό μοντέλο ενός κτιρίου, μπορεί να παρουσιαστεί υπό διάφορες συνθήκες φωτισμού, κίνησης, κ.λπ. προσομοιάζοντας την πραγματικότητα καλύτερα από μία απλή μακέτα, καλύπτοντας έτσι τις βασικές απαιτήσεις της.

Ένα πολύ σημαντικό εργαλείο στο χώρο των μελετών, της αρχιτεκτονικής αλλά και των κατασκευών αποτελεί το BIM (Building Information Modeling)(Εικόνα 1.3). Σύμφωνα με την ιστοσελίδα buildipedia.com, το BIM είναι : *μια ολοκληρωμένη ψηφιακή αναπαράσταση των φυσικών και λειτουργικών χαρακτηριστικών μιας υποδομής. Ένα μοντέλο BIM αποτελεί μια πηγή πληροφοριών για ένα κτίριο – υποδομή, δημιουργώντας έτσι μια αξιόπιστη βάση για λήψη βελτιωμένων αποφάσεων σε όλο τον κύκλο ζωής, που είναι διαθέσιμη από τα πιο πρώιμα στάδια της σχεδιαστικής σύλληψης έως την κατεδάφιση* [15][16].



Εικόνα 1.3: BIM.
Πηγή: http://buildipedia.com/aec-pros/design-news/the-daily-life-of-building-information-modeling-bim?print=1&tmpl=component

Πέρα όμως από την αρχιτεκτονική, τα τελευταία χρόνια εξελίσεται και ο τομέας του Κτηματολογίου, όπου γίνονται προσπάθειες για να γίνουν τρισδιάστατες κτηματολογικές καταγραφές. Με αυτόν τον τρόπο θα απεικονίζεται εύκολα το αστικό τοπίο, παρά την πολυπλοκότητά του λόγω των επικαλυπτόμενων κατασκευών και των σύνθετων εμπράγματων δικαιωμάτων. Ωστόσο, παρά την εξέλιξη της τεχνολογίας, η λειτουργία 3D Κτηματολογίου διεθνώς περιορίζεται σε πειραματικά μοντέλα και βασικές εφαρμογές [17].

Τέλος ένας άλλος πολύ σημαντικός τομέας που βρίσκουν χρήση τα τρισδιάστατα μοντέλα είναι η ιατρική. Οι γιατροί μπορούν να μελετήσουν ακριβή τρισδιάστατα μοντέλα μέρος ή ολόκληρου του σώματος, ενός ασθενή. Αυτά τα μοντέλα μπορούν να χρησιμοποιηθούν με πολλούς τρόπους όπως για παράδειγμα η ακριβής διάγνωση μίας, η επιλογή της κατάλληλης θεραπείας αλλά και ο προγραμματισμός μίας χειρουργικής επέμβασης.

Επιπλέον με την τρισδιάστατη εκτύπωση των μοντέλων αυτών, ένας γιατρός μπορεί να εξετάσει και να εξασκηθεί σε ένα προσαρμοσμένο 3D τυπωμένο μοντέλο πριν από τη χειρουργική επέμβαση.

Μία άλλη χρήση των τρισδιάστατων τυπωμένων μοντέλων είναι η δημιουργία προσθετικών μελών με χαμηλό κόστος ή αντικατάσταση μέρους τον οστών (π.χ. γόνατο) [18][19].

Τέλος, γίνεται προσπάθεια για τη δημιουργία τρισδιάστατων εκτυπωμένων ανθρώπινων οργάνων από βιοϋλικά έτσι ώστε να μην είναι πλέον απαραίτητη η δωρεά ανθρώπινων οργάνων [20].

Κεφάλαιο 2.

Παρουσίαση Εφαρμογής

Η εφαρμογή που δημιουργήθηκε, για το σκοπό της παρούσας εργασίας, είναι μία εφαρμογή που στοχεύει στην ανάδειξη, μέσω κάποιων παραδειγμάτων, του τρόπου παρουσίασης της πληροφορίας τρισδιάστατου μοντέλου.

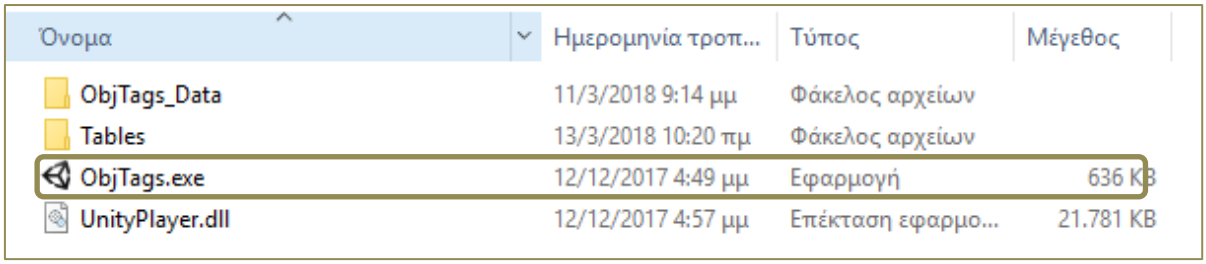
Η ανάδειξη της κατάλληλης πληροφορίας διαφέρει ανάλογα το χρήστη που χρησιμοποιεί το μοντέλο. Μέσω αυτής της εφαρμογής, δίνεται η δυνατότητα, αποθήκευσης και παρουσίασης της πληροφορίας που είναι απαραίτητη για κάθε χρήστη. Σε ότι αφορά τον τομέα της Πολιτιστικής Κληρονομιάς, που ανήκουν και τα μοντέλα που χρησιμοποιήθηκαν, οι χρήστες θα μπορούσαν να είναι αρχαιολόγοι, ερευνητές, μηχανικοί, επισκέπτες κ.α. [21].

Επιπλέον δίνεται στο χρήστη η δυνατότητα εισαγωγής δικού του μοντέλου στο οποίο μπορεί να περιηγηθεί ελεύθερα.

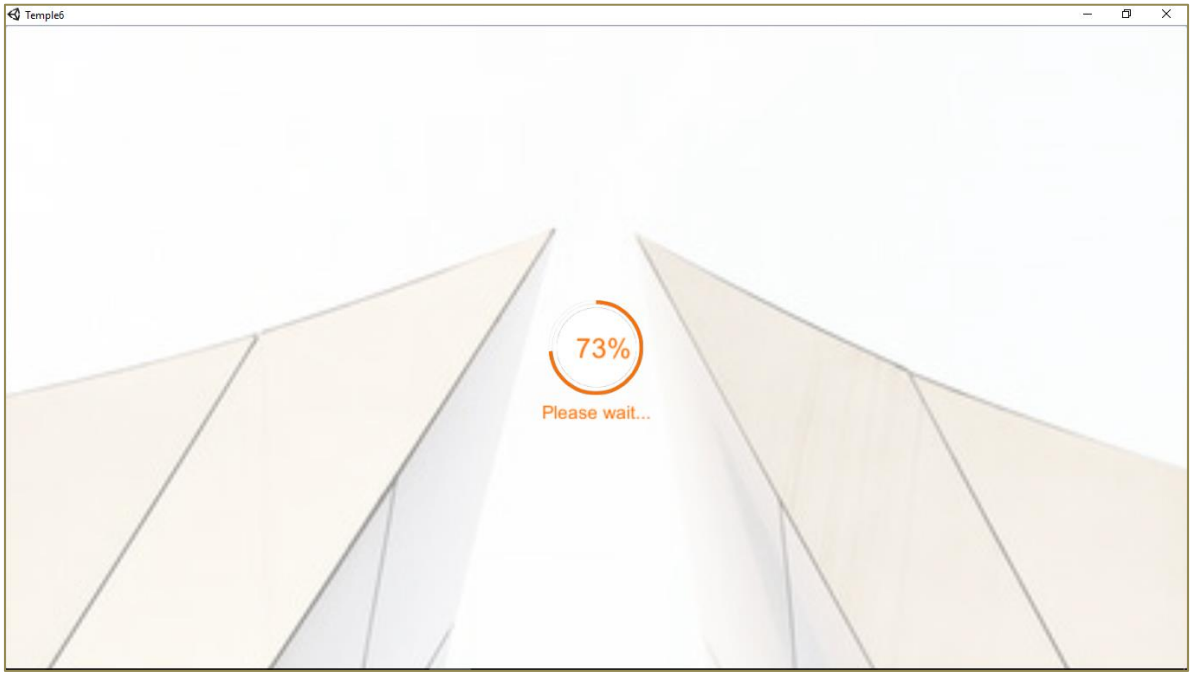
Τα δεδομένα που εισήχθησαν στην εφαρμογή, είναι δωρεάν τρισδιάστατα δεδομένα, από τα οποία υπάρχει μεγάλο εύρος επιλογών στο διαδίκτυο.

2.1 Εκκίνηση Εφαρμογής

Η εφαρμογή αποτελείται από έναν φάκελο που περιέχει τα αρχεία της. Επιλέγοντας το εκτελέσιμο αρχείο (Εικόνα 2.1), αρχίζει η εφαρμογή να φορτώνει. Στην Εικόνα 2.2, παρουσιάζεται η οθόνη που φορτώνει η εφαρμογή.



Εικόνα 2.1: Περιεχόμενα Φακέλου Εφαρμογής.



Εικόνα 2.2: Φόρτωση Εφαρμογής.

Στη συνέχεια εμφανίζεται το αρχικό μενού της εφαρμογής (Εικόνα 2.3), το οποίο δίνει στο χρήστη τις παρακάτω επιλογές:

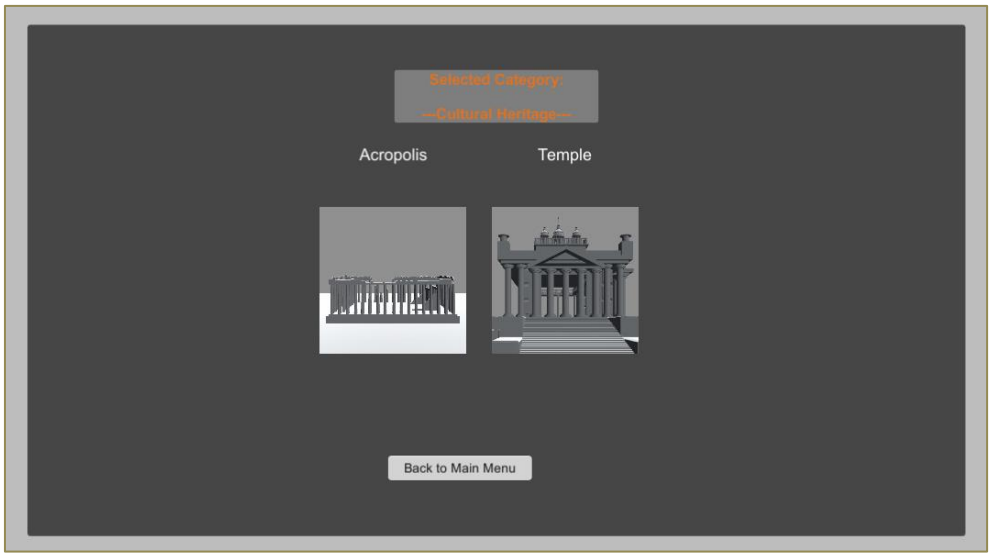
- Να ανοίξει παραδείγματα μοντέλων με ετικέτες
- Να φορτώσει δικό του μοντέλο
- Να κλείσει την εφαρμογή

Εικόνα 2.3: Αρχικό Μενού Εφαρμογής.

2.2 Μοντέλα με Ετικέτες Εφαρμογής

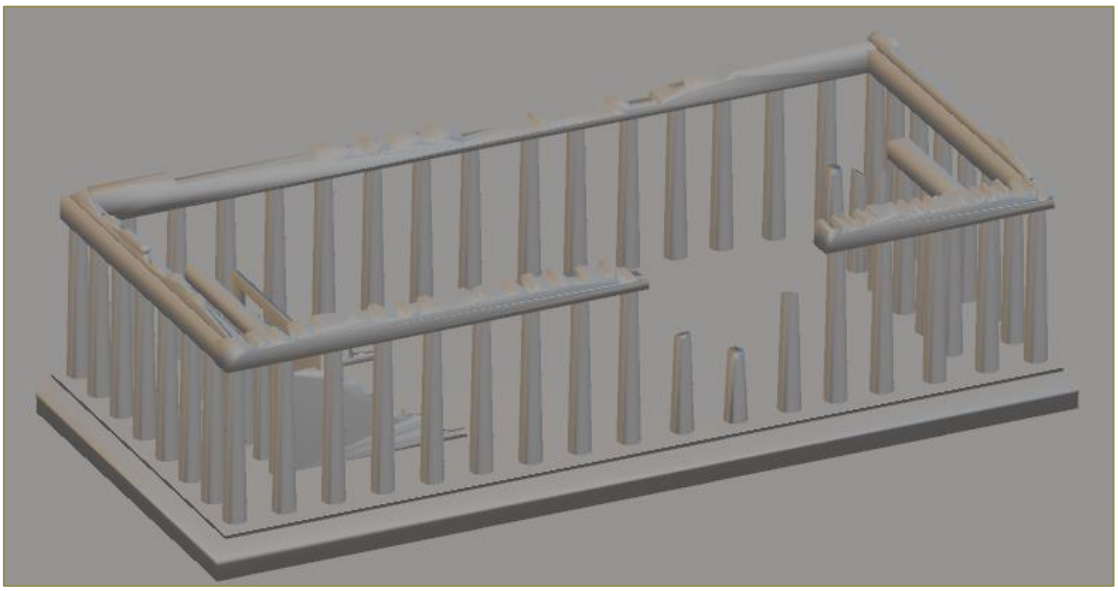
Επιλέγοντας ο χρήστης το κουμπί *Open Sample*, εμφανίζεται το υπομενού της Εικόνα 2.4, το οποίο δίνει τις παρακάτω επιλογές:

- να φορτωθεί ένα τρισδιάστατο μοντέλο του Παρθενώνα
- να φορτωθεί ένα τρισδιάστατο μοντέλο αρχαίου φανταστικού ναού
- να επιστρέψει ο χρήστης στο αρχικό μενού (Εικόνα 2.3)

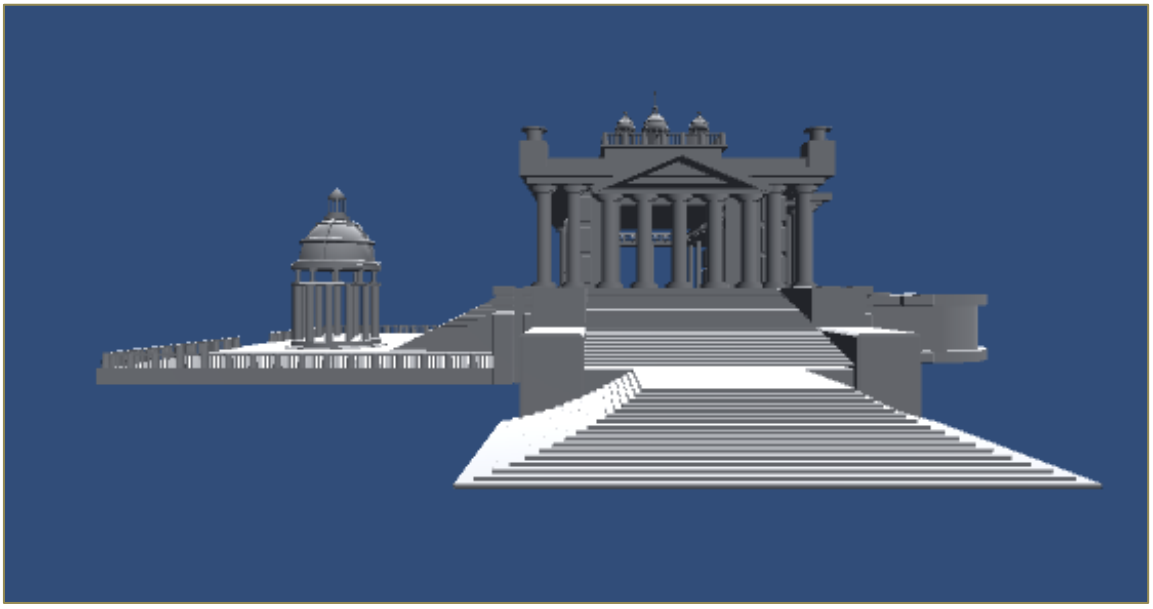


Εικόνα 2.4: Υπομενού με Παραδείγματα Εφαρμογής.

Τα μοντέλα που χρησιμοποιήθηκαν, είναι αυτό της Εικόνα 2.5 καθώς και αυτό της Εικόνα 2.6.



Εικόνα 2.5: Τρισδιάστατο Μοντέλο Παρθενώνα.
Πηγή: https://www.turbosquid.com/3d-models/acropolis-3ds-free/610885



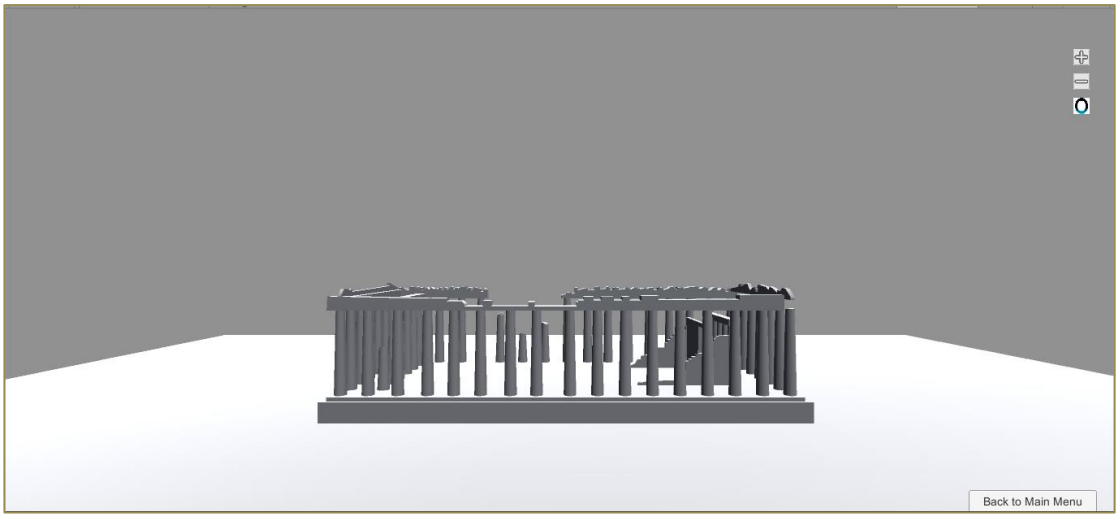
Εικόνα 2.6: Τρισδιάστατο Μοντέλο Φανταστικού Ναού.
Πηγή: https://free3d.com/3d-model/temple-98984.html

Οι δύο αυτές επιλογές, δίνουν τη δυνατότητα παρουσίασης των ετικετών των αντικειμένων με διαφορετικό τρόπο στο κάθε μοντέλο.

Επιλέγοντας τον Παρθενώνα, ο χρήστης μεταφέρεται σε ένα νέο παράθυρο (Εικόνα 2.7), το οποίο περιέχει το τρισδιάστατο μοντέλο. Οι δυνατότητες που δίνονται στο χρήστη σε ότι αφορά την περιήγηση είναι οι παρακάτω:

- Μετακίνηση
- Περιστροφή
- Μεγέθυνση
- Σμίκρυνση
- Επαναφορά κάμερας στην αρχική θέση

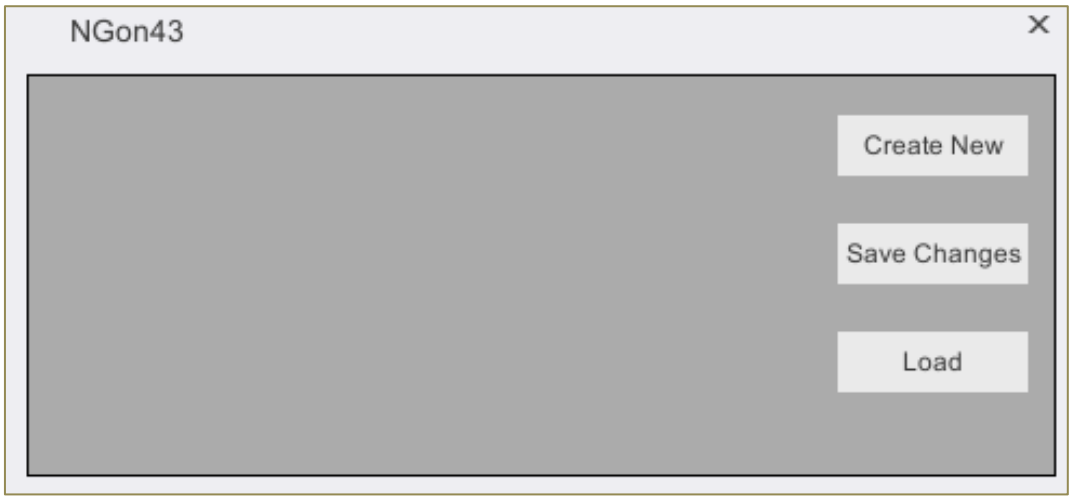
Αυτές τις δυνατότητες τις έχει ο χρήστης σε όλα τα τρισδιάστατα μοντέλα της εφαρμογής.



.

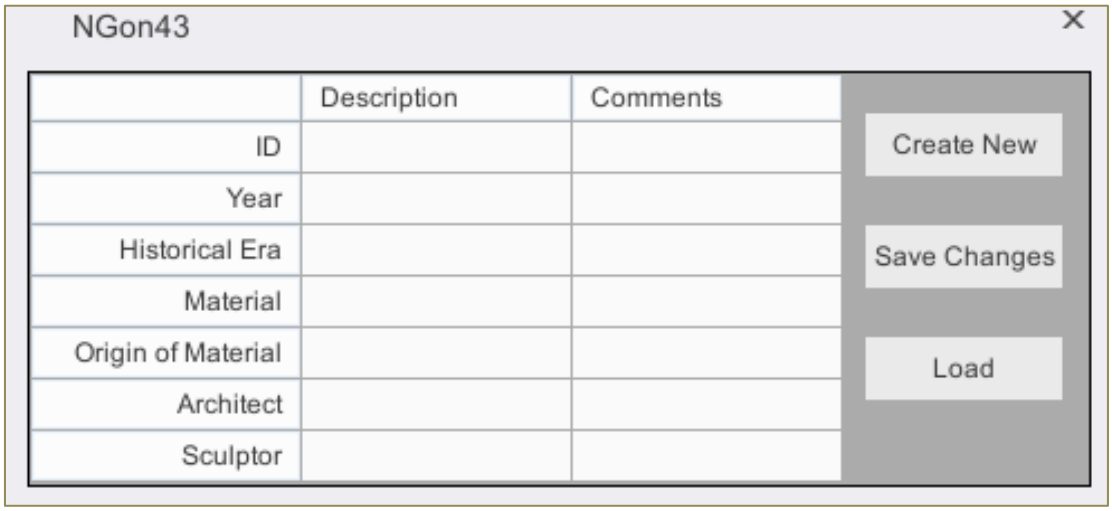
Εικόνα 2.7: Τρισδιάστατο Μοντέλο Παρθενώνα στην Εφαρμογή.

Στο μοντέλο του Παρθενώνα, επιλέχθηκε να γίνει παρουσίαση των δεδομένων σε μορφή πίνακα, έτσι ώστε ο χρήστης να έχει τη δυνατότητα προσθήκης περισσότερων πληροφοριών ανά αντικείμενο. Με διπλό πάτημα πάνω στο αντικείμενο αλλάζει το αρχικό του χρώμα σε κυανό και εμφανίζεται το παράθυρο της Εικόνα 2.8.



Εικόνα 2.8: Νέο Παράθυρο.

Πατώντας στο κουμπί *Create New*, δημιουργείται ένας νέος, κενός πίνακας ιδιοτήτων του αντικειμένου που επιλέχθηκε (Εικόνα 2.9).

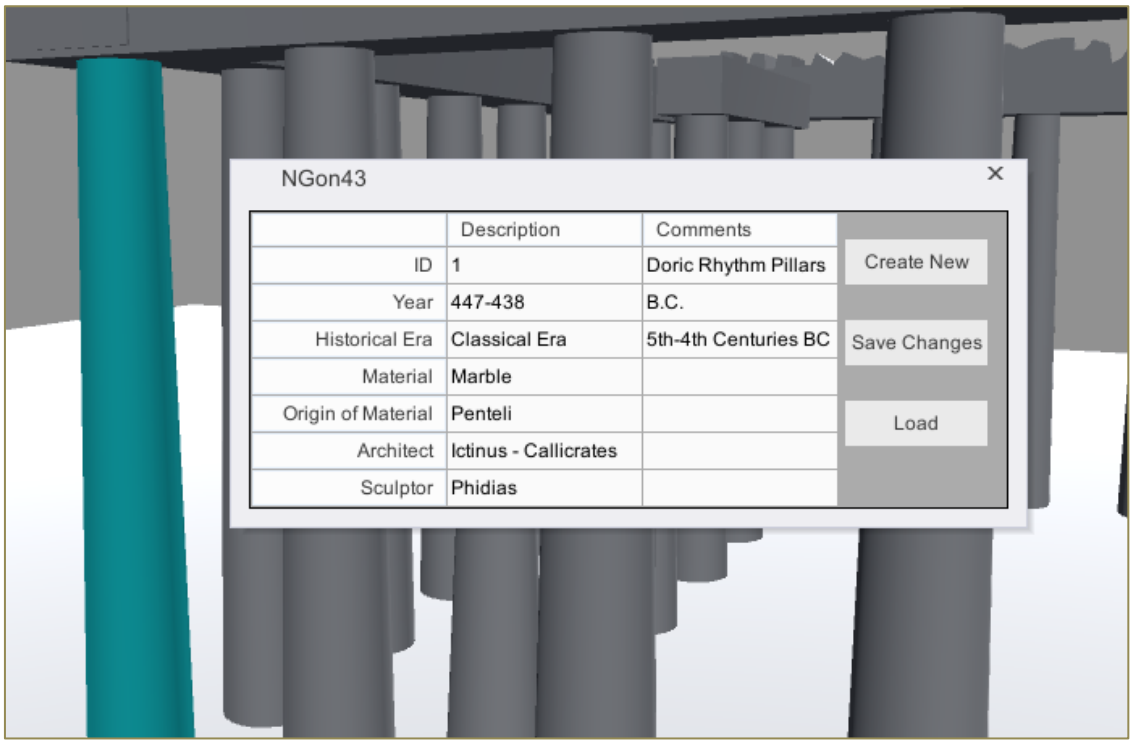


Εικόνα 2.9: Κενός Πίνακας Δεδομένων Αντικειμένου.

Επειδή πρόκειται για αρχαίο ναό, οι ιδιότητες που προστέθηκαν στον πίνακα είναι οι παρακάτω:

- ID
- Έτος
- Ιστορική Εποχή
- Υλικό
- Προέλευση Υλικού
- Αρχιτέκτονας
- Γλύπτης

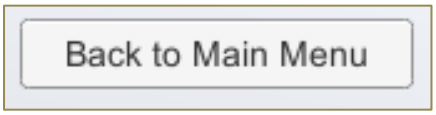
Στη συνέχεια ο χρήστης μπορεί να συμπληρώσει και να αποθηκεύσει, πατώντας το κουμπί *Save Changes*, τις αντίστοιχες πληροφορίες στα κελιά του πίνακα (Εικόνα 2.10).



Εικόνα 2.10: Πίνακας με Δεδομένα Αντικειμένου.

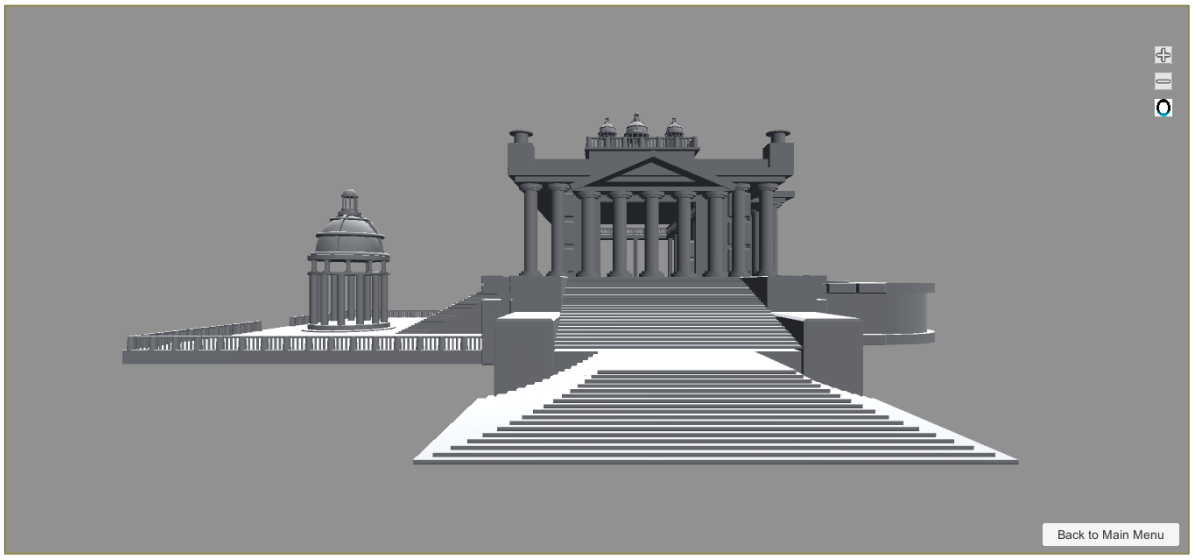
Πατώντας το κουμπί *Load*, αν το αντικείμενο περιέχει αποθηκευμένες πληροφορίες, ο πίνακας εμφανίζεται συμπληρωμένος.

Τέλος για την επιστροφή του χρήστη στο αρχικό μενού δημιουργήθηκε ένα ακόμη κουμπί το οποίο τοποθετήθηκε στο κάτω δεξιά μέρος της οθόνης (Εικόνα 2.11).



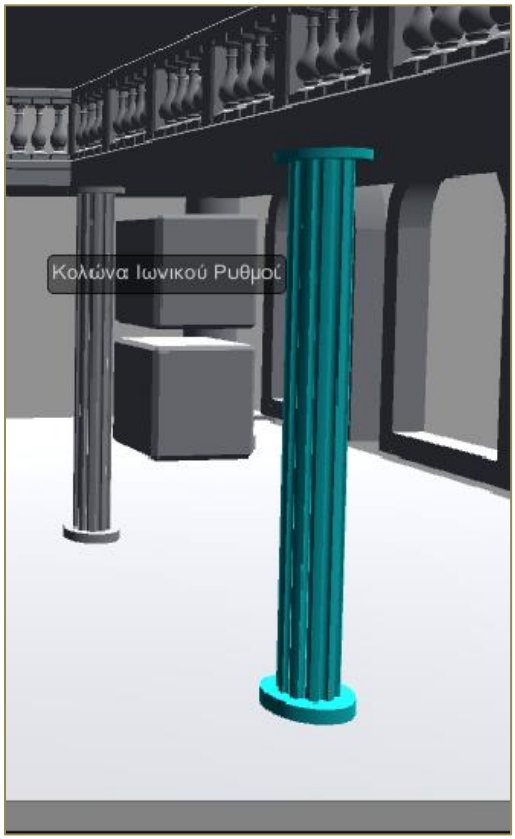
Εικόνα 2.11: Κουμπί επιστροφής στο Αρχικό Μενού.

Στην περίπτωση που ο χρήστης επιλέξει τον άλλο ναό (Εικόνα 2.12), η παρουσίαση των δεδομένων γίνεται με μία απλή ετικέτα. Πατώντας πάνω σε ένα αντικείμενο του ναού και σε αυτήν την περίπτωση, αλλάζει το αρχικό του χρώμα σε κυανό και εμφανίζεται η ετικέτα με τα δεδομένα του αντικειμένου.



Εικόνα 2.12: Τρισδιάστατο Μοντέλο Ναού στην Εφαρμογή.

Αυτό ο τρόπος παρουσίασης είναι χρήσιμος για απλά αντικείμενα, που δεν χρειάζονται αποθήκευση σύνθετης πληροφορίας και ο χρήστης μπορεί να εμφανίσει τα δεδομένα με ένα απλό πάτημα πάνω στο αντικείμενο (Εικόνα 2.13).



Εικόνα 2.13: Εμφάνιση Απλής Ετικέτας.

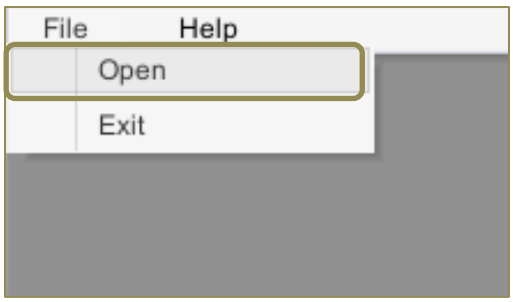
Τέλος, σε περίπτωση που ο χρήστης επιθυμεί να επιστρέψει στο αρχικό μενού, αρκεί να επιλέξει και σε αυτήν την περίπτωση το κουμπί που βρίσκεται κάτω δεξιά στην οθόνη (*Back to Main Menu*).

2.3 Φόρτωση Νέου Μοντέλου

Η τελευταία επιλογή που δίνεται στο χρήστη, είναι η φόρτωση δικού του τρισδιάστατου μοντέλου. Επιλέγοντας το κουμπί *Load*, εμφανίζεται το παράθυρο της Εικόνα 2.14.

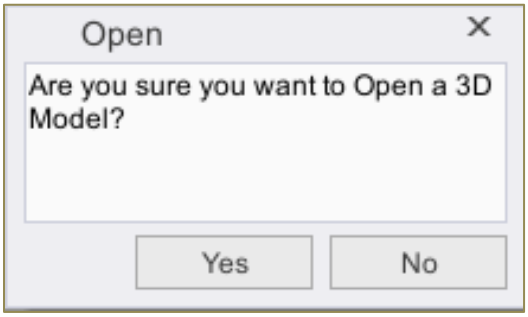
Εικόνα 2.14: Κενό Περιβάλλον με Σκοπό την Εισαγωγή Τρισδιάστατου Μοντέλου.

Για την εισαγωγή του νέου τρισδιάστατου μοντέλου, αρκεί ο χρήστης να πατήσει στο μενού της εφαρμογής, *File* και στην συνέχεια *Open* (Εικόνα 2.15).



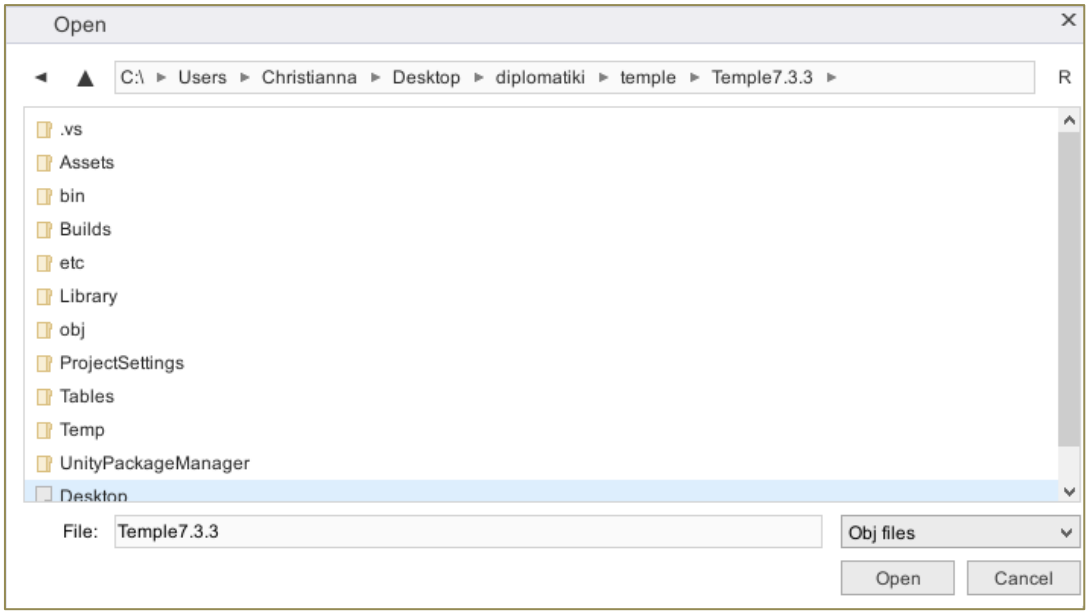
Εικόνα 2.15: Άνοιγμα Μοντέλου.

Πατώντας *Open* εμφανίζεται στο χρήστη ένα νέο παράθυρο επιβεβαίωσης της ενέργειας (Εικόνα 2.16).



Εικόνα 2.16: Μήνυμα Επιβεβαίωσης για Άνοιγμα νέου Αρχείου.

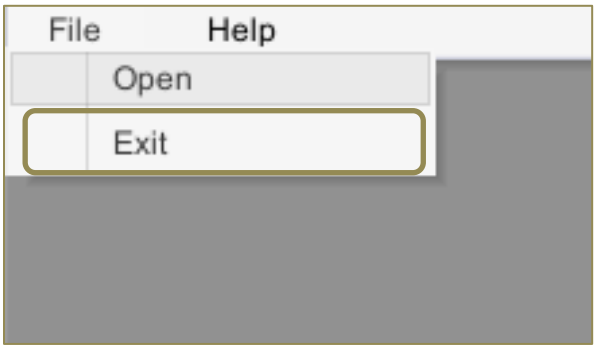
Επιλέγοντας το κουμπί *Yes*, ανοίγει το παράθυρο επιλογής του αρχείου (Εικόνα 2.17).



Εικόνα 2.17: Παράθυρο Επιλογής Αρχείου.

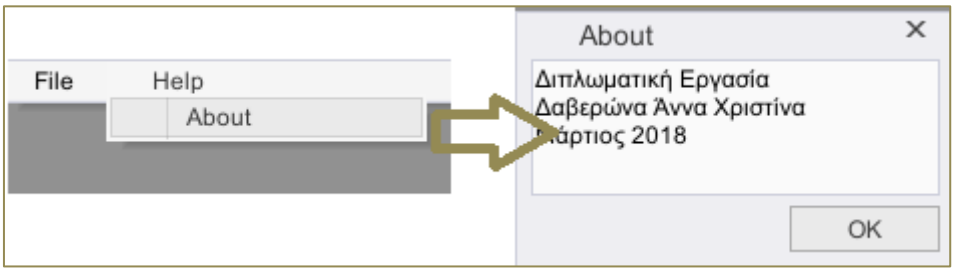
Τα αρχεία που μπορεί να επιλέξει ο χρήστης είναι της μορφής ".obj". Επιλέγοντας το μοντέλο, ο χρήστης έχει τις ίδιες δυνατότητες περιήγησης που είχε και στα υπόλοιπα μοντέλα.

Για την έξοδο από την εφαρμογή αρκεί να πατηθεί το κουμπί Exit από το μενού (Εικόνα 2.18).



Εικόνα 2.18: Τερματισμός Εφαρμογής.

Πατώντας το κουμπί Help και στη συνέχεια About, εμφανίζονται πληροφορίες για την εφαρμογή (Εικόνα 2.19).



Εικόνα 2.19: Πληροφορίες Εφαρμογής.

Κεφάλαιο 3.

Λογισμικά που χρησιμοποιήθηκαν

Σε αυτό το κεφάλαιο, παρουσιάζονται τα λογισμικά που χρησιμοποιήθηκαν για τη δημιουργία της εφαρμογής. Αναλυτικότερα, έγινε η χρήση των προγραμμάτων Unity και Microsoft Visual Studio.

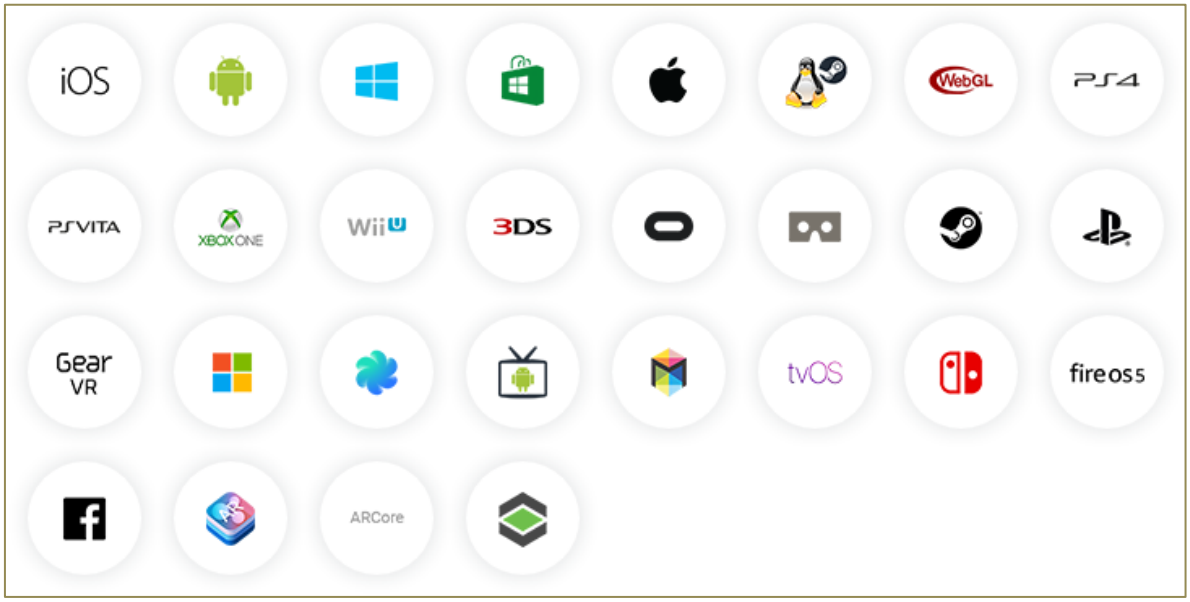
Η άδεια χρήσης των δύο αυτών λογισμικών παρέχεται με δύο τρόπους, δωρεάν και επί πληρωμής έκδοση. Για τη δημιουργία της παρούσας εφαρμογής χρησιμοποιήθηκε η δωρεάν έκδοση του Unity και του Microsoft Visual Studio, διότι οι δυνατότητες που παρέχονται καλύπτουν τις ανάγκες υλοποίησης της.

3.1 Unity

Το Unity (Εικόνα 3.1) είναι ένα σύστημα λογισμικού, ανεξάρτητο πλατφόρμας, σχεδιασμένο για τη δημιουργία και την ανάπτυξη βιντεοπαιχνιδιών.

Εικόνα 3.1: Λογότυπο Unity.
Πηγή: https://unity3d.com

Το λογισμικό αυτό, δίνει στο χρήστη τη δυνατότητα ανάπτυξης δισδιάστατων και τρισδιάστατων βιντεοπαιχνιδιών για υπολογιστές, παιχνιδομηχανές, φορητές συσκευές και ιστοσελίδες. Σήμερα, το Unity υποστηρίζει την ανάπτυξη βιντεοπαιχνιδιών σε περισσότερες από 25 πλατφόρμες (Εικόνα 3.2).



Εικόνα 3.2: Πλατφόρμες με Δυνατότητα Ανάπτυξής Βιντεοπαιχνιδιών μέσω του Λογισμικού "Unity".
Πηγή: https://unity3d.com/unity

Οι γλώσσες προγραμματισμού που υποστηρίζει το Unity, είναι:

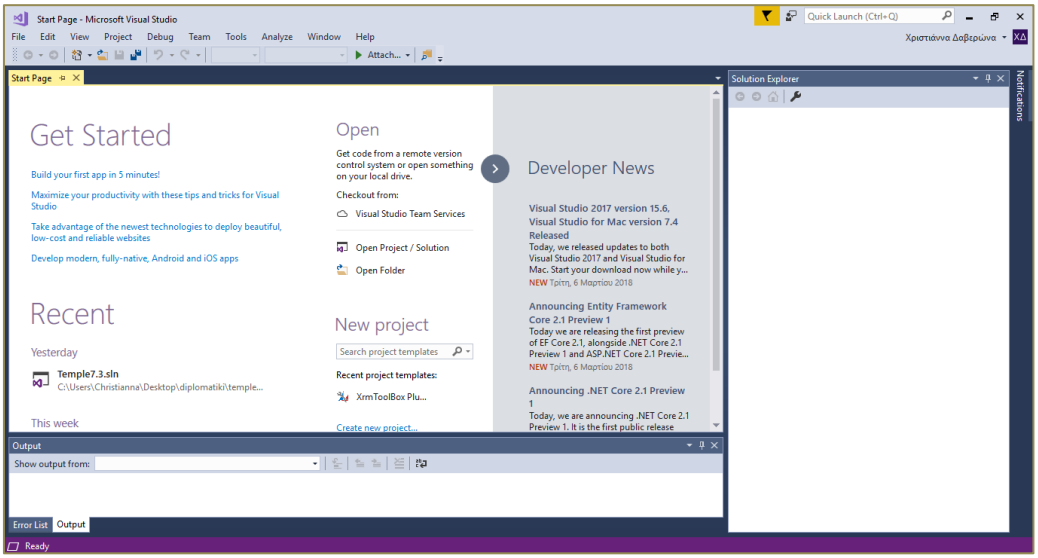
- C#
- JavaScript (UnityScript)
- C++

Η C# και η JavaScript είναι γλώσσες που διατίθενται στη δωρεάν έκδοση του Unity, ενώ η C++ μόνο στην έκδοση επί πληρωμής. Η γλώσσα που επιλέχθηκε ήταν η C#, διότι είναι πιο διαδεδομένη με περισσότερους χρήστες από ότι η JavaScript με αποτέλεσμα να υπάρχουν περισσότερα παραδείγματα στο διαδίκτυο, κάνοντας έτσι την εκμάθησή της πιο εύκολη για έναν αρχάριο χρήστη. Κύρια πηγή εκμάθησης και παραδειγμάτων ήταν το YouTube, η επίσημη ιστοσελίδα του Unity [22], αλλά και κάποια forum [23][24][25].

3.2 Microsoft Visual Studio

Το Microsoft Visual Studio (Εικόνα 3.3), είναι ένα ολοκληρωμένο περιβάλλον ανάπτυξης εφαρμογών, της Microsoft, στο οποίο γίνεται η δημιουργία και η επεξεργασία των αρχείων εντολών (script) (Εικόνα 3.4). Η εγκατάστασή του, γίνεται από την επίσημη ιστοσελίδα https://www.visualstudio.com/downloads/.

Εικόνα 3.3: Λογότυπο Microsoft Visual Studio.
Πηγή: https://www.visualstudio.com



Εικόνα 3.4: Περιβάλλον Microsoft Visual Studio 2017.

Το λογισμικό αυτό υποστηρίζει τις παρακάτω γλώσσες προγραμματισμού [26]:

- C
- C++
- C++/CLI
- Visual Basic.NET
- C#
- F#
- JavaScript
- TypeScript
- XML
- XSLT
- HTML
- CSS

Πρόκειται για ένα αρκετά εύχρηστο λογισμικό συγγραφής κώδικα, το οποίο δίνει στο χρήστη τη δυνατότητα να εντοπίσει και να διορθώσει εύκολα τα λάθη του, προτείνοντάς του πιθανές λύσεις. Επιπλέον μπορεί να μεταφράσει κώδικα από μία γλώσσα σε μία άλλη από αυτές που υποστηρίζει.

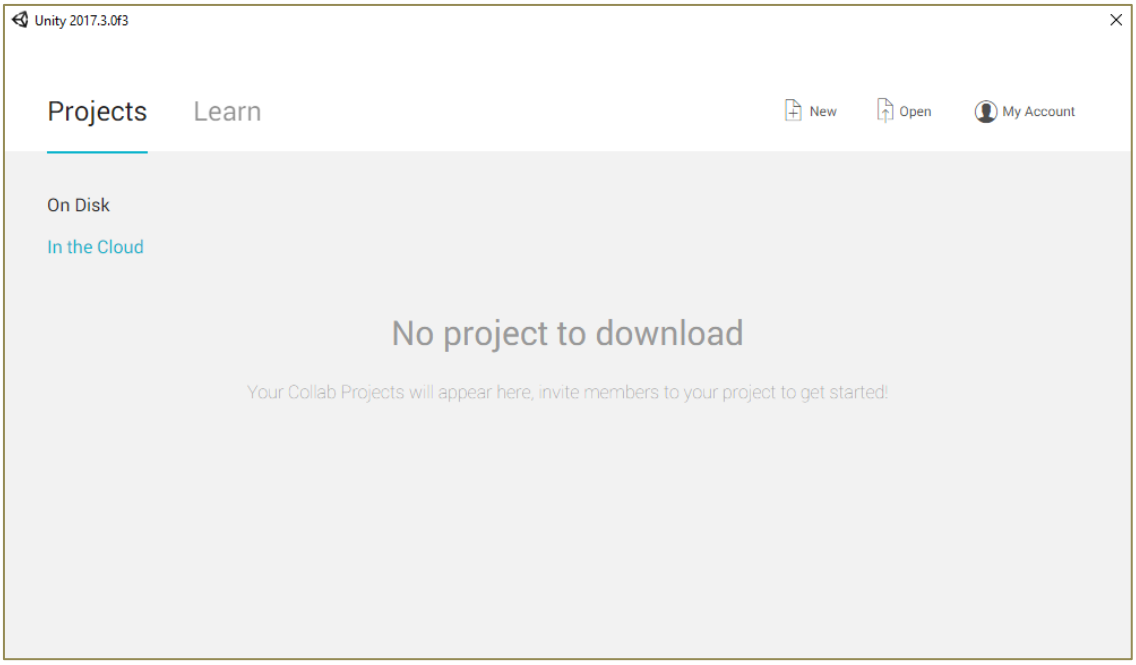
Κεφάλαιο 4.

Δημιουργία Εφαρμογής

Στο Κεφάλαιο αυτό περιγράφονται οι διαδικασίες που ακολουθήθηκαν για τη δημιουργία της εφαρμογής. Αναλύονται τα επιμέρους στοιχεία και η λειτουργία τους.

4.1 Εγκατάσταση και Εκκίνηση του Unity

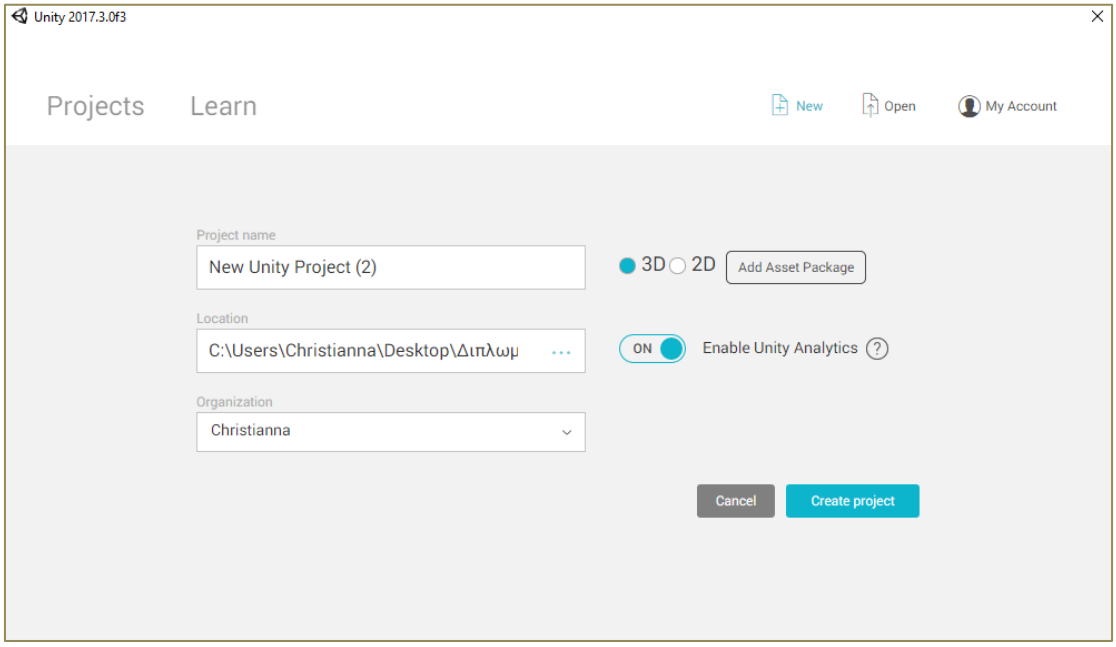
Αρχικά έγινε η λήψη του λογισμικού από την επίσημη ιστοσελίδα [27] και στη συνέχεια έγινε η εγκατάστασή και η εκκίνηση του. Κατά την εκκίνηση του Unity, εμφανίζεται το παράθυρο της Εικόνα 4.1.



Εικόνα 4.1: Αρχικό Παράθυρο Unity

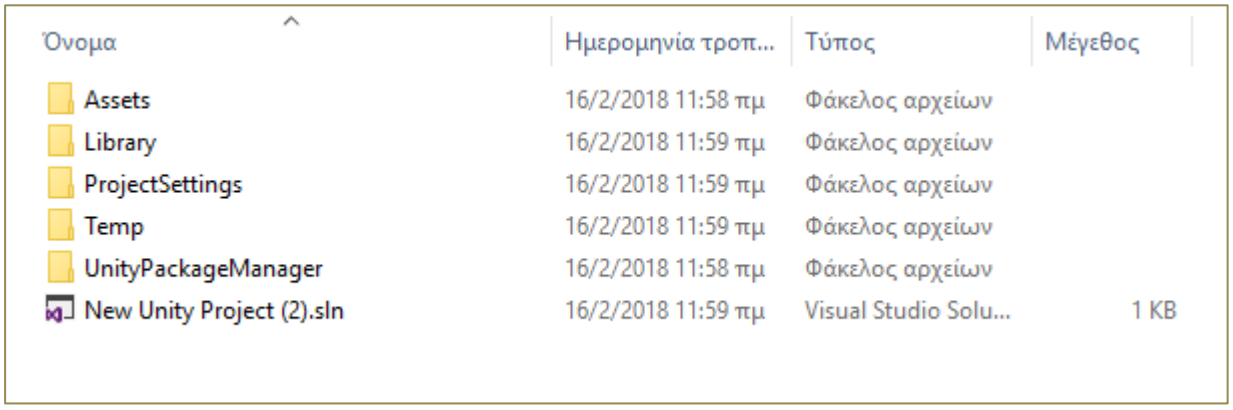
Πατώντας *new,* στο αρχικό παράθυρο που ανοίγει, προκύπτει το παράθυρο της Εικόνα 4.2, όπου εμφανίζονται οι επιλογές για τη δημιουργία ενός νέου project. Στο παράθυρο αυτό καθορίζονται τα εξής στοιχεία του νέου project:

- όνομα
- φάκελος αποθήκευσης
- όνομα οργανισμού που το δημιουργεί
- επιλογή αν θα είναι 2D ή 3D



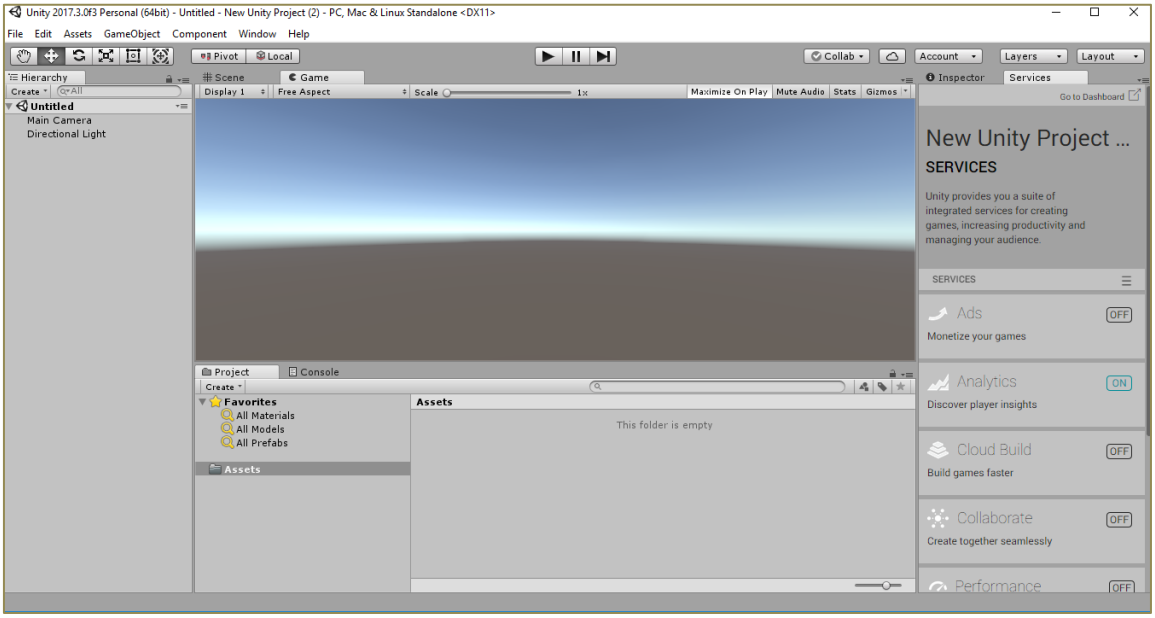
Εικόνα 4.2: Επιλογές για τη Δημιουργία Νέου Project.

Συμπληρώνοντας και επιλέγοντας τα παραπάνω, δίνεται στο χρήστη η δυνατότητα να πατήσει το κουμπί create project το οποίο, δημιουργεί ένα νέο φάκελο στη θέση που επιλέχθηκε. Ο φάκελος αυτός, περιέχει κάποιους φακέλους (Εικόνα 4.3), οι οποίοι είναι απαραίτητοι για τη λειτουργία του project.



Εικόνα 4.3: Περιεχόμενα Φακέλου Νέου Project.

Με τη δημιουργία του νέου project, ανοίγει αυτόματα μία νέα σκηνή στο Unity, όπως φαίνεται στην Εικόνα 4.4.



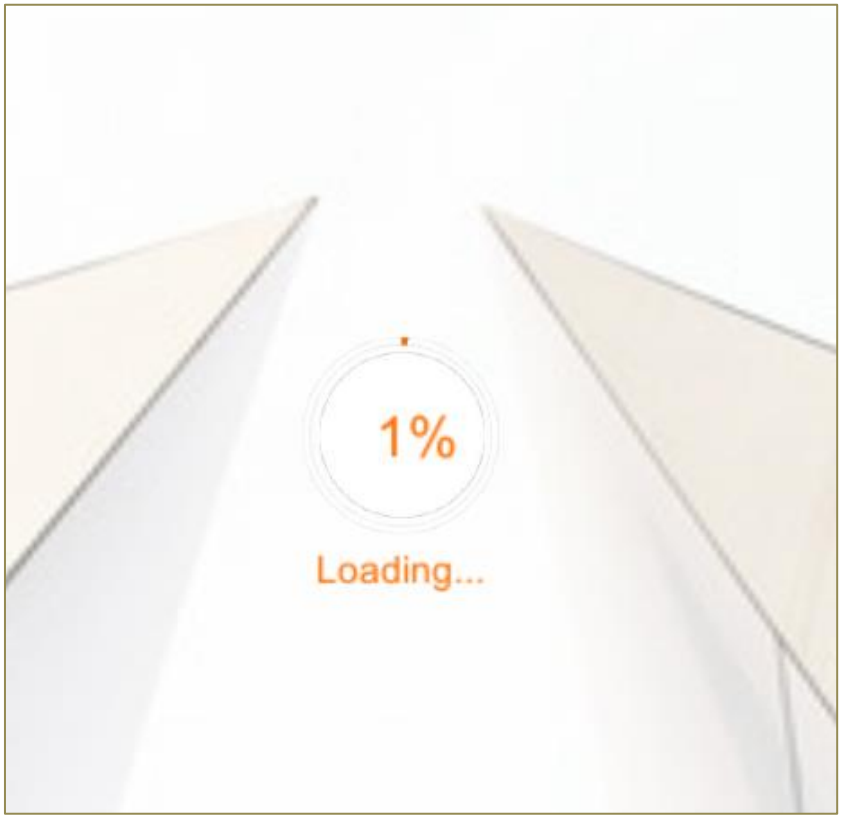
Εικόνα 4.4: Περιβάλλον Χρήστη Unity.

Το Unity, σε κάθε νέο project, εμπεριέχει μία κάμερα για να παρουσιάζει ότι έχει δημιουργηθεί στη νέα σκηνή και ένα φως για να εμφανίζεται και να φωτίζει η σκηνή κατά την εκτέλεση της εφαρμογής.

4.2 Δημιουργία Περιβάλλον Χρήστη της Εφαρμογής

a) Φόρτωση Εφαρμογής

Κατά την εκκίνηση της εφαρμογής, παρουσιάζεται ένας κύκλος με το ποσοστό που έχει φορτώσει, όπως φαίνεται στην Εικόνα 4.5. Για την δημιουργία του, χρησιμοποιήθηκε έτοιμο πακέτο με αυτό το περιεχόμενο [28], το οποίο παρέχεται δωρεάν από το κατάστημα του Unity, Asset Store. Από το συγκεκριμένο πακέτο χρησιμοποιήθηκε ένα script το οποίο τροποποιήθηκε και προσαρμόστηκε στην εφαρμογή. Στο Παράρτημα Ι αναγράφεται ο συγκεκριμένος κώδικας που χρησιμοποιήθηκε.



Εικόνα 4.5: Φόρτωση Εφαρμογής

b) Δημιουργία Αρχικού Μενού

Το αρχικό μενού, εμφανίζεται στο χρήστη, όταν φορτωθεί η εφαρμογή. Για την κατασκευή του, δημιουργήθηκε μία νέα σκηνή στην οποία προστέθηκαν τα παρακάτω κουμπιά:

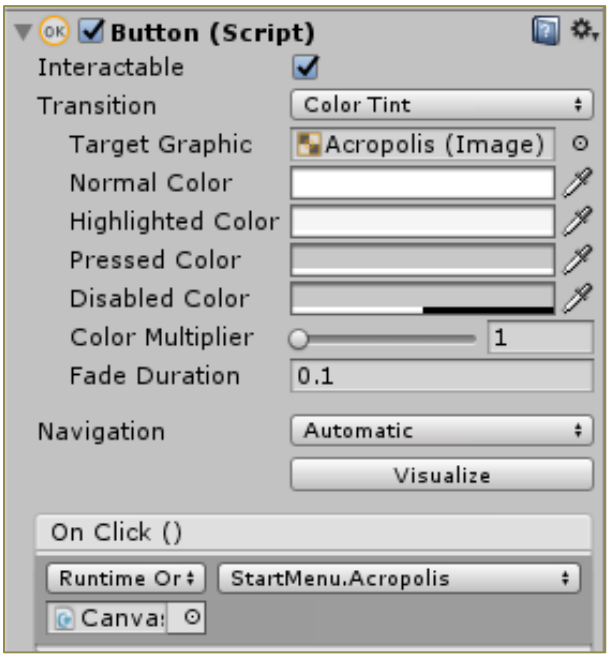
- Κουμπί για τη φόρτωση υπομενού που περιέχει έτοιμα μοντέλα με ετικέτες
- Φόρτωση νέου τρισδιάστατου μοντέλου (Load)
- Έξοδος από την εφαρμογή (Exit)

Πατώντας το πρώτο κουμπί, φορτώνεται στην οθόνη ένα νέο υπομενού με δυνατότητα επιλογής μεταξύ δύο μοντέλων, όπως αυτά αναλύθηκαν στο Κεφάλαιο 2. Για να φορτωθεί ο ναός που επιλέχθηκε από το χρήστη δημιουργήθηκε ένα script, το οποίο ονομάστηκε StartMenu, που περιέχει μία συνάρτηση για την κάθε σκηνή, η οποία συνδέει το κουμπί με τη νέα σκηνή. Ο κώδικας αυτός βρίσκεται στο Παράρτημα Ι. Παρακάτω αναφέρεται ένα παράδειγμα το οποίο όταν η συνάρτηση αυτή συνδεθεί με κάποιο κουμπί, τότε πατώντας το, θα ανοίγει η ανάλογη σκηνή που στην προκειμένη περίπτωση θα είναι αυτή που φορτώνει το τρισδιάστατο μοντέλο του Παρθενώνα.

```
    public void Acropolis()

{
```

```
    UnityEngine.Application.LoadLevel("sample_acropolis");

}
```

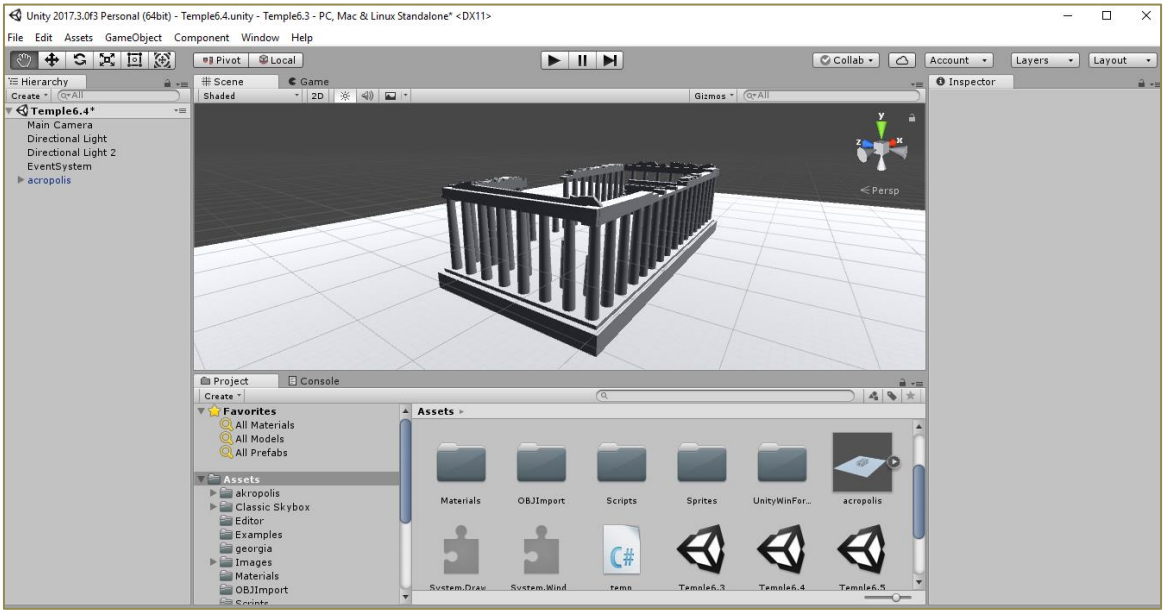
Η συνάρτηση αυτή προστίθεται στο κουμπί που έχει δημιουργηθεί στο Unity, όπως φαίνεται στην Εικόνα 4.6.



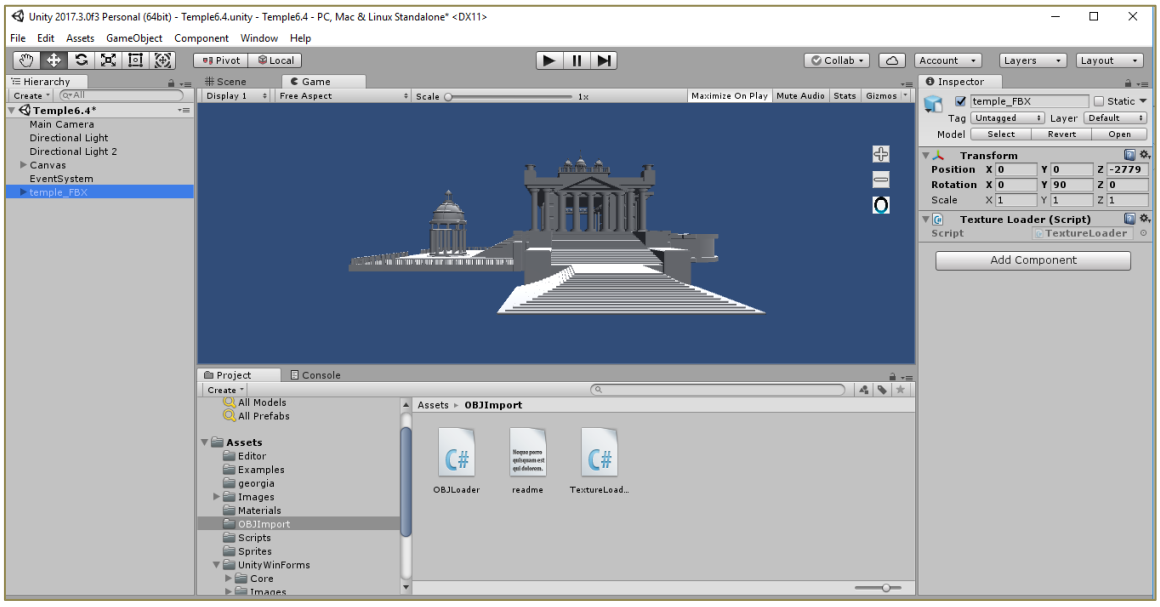
Εικόνα 4.6: Προσθήκη Λειτουργίας Κουμπιού.

c) Περιήγηση

Η δημιουργία του κύριου μέρους της εφαρμογής αποτελείται από διάφορες λειτουργίες. Για τις δοκιμές και τη δημιουργία του κώδικα, χρησιμοποιήθηκαν δωρεάν τρισδιάστατα μοντέλα που υπάρχουν στο διαδίκτυο ως παραδείγματα. Πιο συγκεκριμένα χρησιμοποιήθηκε ένα μοντέλο του Παρθενώνα (Εικόνα 4.7) καθώς και ένα μοντέλο ενός αρχαίου φανταστικού ναού (Εικόνα 4.8). Τα αρχεία των τρισδιάστατων μοντέλων είχαν επέκταση αρχείου ".obj".

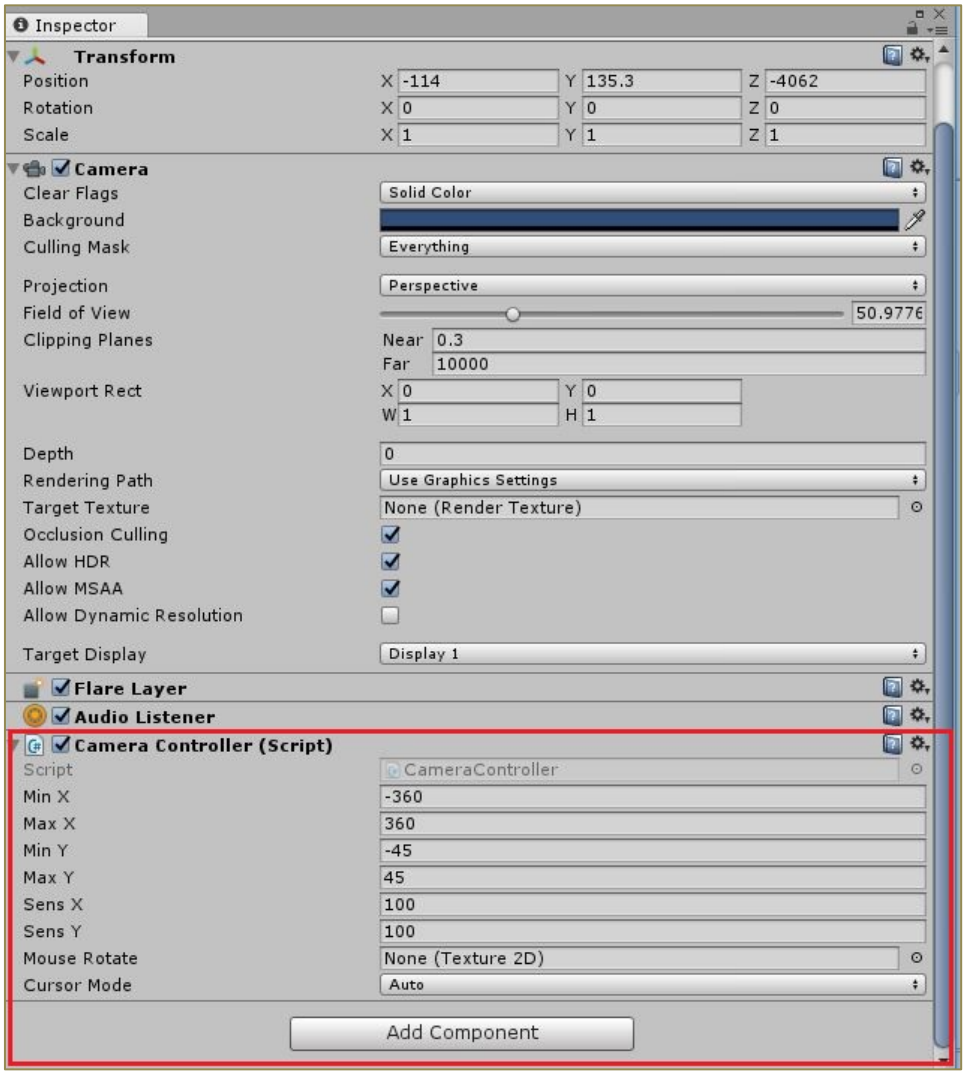


Εικόνα 4.7: Τρισδιάστατο Μοντέλο Παρθενώνα στο Unity.
Πηγή: https://www.turbosquid.com/3d-models/acropolis-3ds-free/610885



Εικόνα 4.8: Τρισδιάστατο Μοντέλο Αρχαίου Ναού στο Unity.
Πηγή: https://free3d.com/3d-model/temple-98984.html

Η περιήγηση στο εκάστοτε μοντέλο, έγινε με τη δημιουργία ενός αρχείου, το οποίο προστέθηκε στις λειτουργίες της κάμερας (Εικόνα 4.9). Το αρχείο αυτό ονομάστηκε "Camera Controller" και στην επόμενη παράγραφο αναλύεται το περιεχόμενό του.



Εικόνα 4.9: Εισαγωγή του Αρχείου Περιήγησης στην Κάμερα.

Οι βασικές λειτουργίες αυτού του script μαζί με τον κώδικα δημιουργίας της κάθε μίας λειτουργίας, είναι οι εξής:

- Μετακίνηση στο χώρο, χρησιμοποιώντας τα βελάκια του πληκτρολογίου.

Η μετακίνηση στο οριζόντιο επίπεδο, έγινε με την εντολή:

```
var x = Input.GetAxis("Horizontal") * Time.deltaTime * moveSpeed;
```

Ενώ στο κάθετο:

```
var z = Input.GetAxis("Vertical") * Time.deltaTime * moveSpeed;
```

Η εντολή transform αλλάζει τη θέση με βάση τις νέες συντεταγμένες.

```
transform.Translate(x, 0, z);

}
```

- Αλλαγή του ύψους της θέσης της κάμερας

Η αλλαγή του ύψους της θέσης της κάμερας επιλέχθηκε να γίνει , χρησιμοποιώντας τη ροδέλα (Mouse ScrollWheel). Έτσι, δημιουργήθηκε μία μεταβλητή scroll, στην οποία καταχωρείται το ύψος εκείνη τη στιγμή.

```
float scroll = Input.GetAxis ("Mouse ScrollWheel");

Για να αλλάξει η θέση χρησιμοποιείται πάλι η εντολή transform.

transform.Translate (0, scroll * zoomSpeed, 0, Space.World);
```

- Μεγέθυνση και σμίκρυνση (Εικόνα 4.10).

Η μεγέθυνση καθώς και η σμίκρυνση του αντικειμένου, γίνονται από δύο κουμπιά στην οθόνη (Εικόνα 4.10). Για τη δημιουργία αυτών των δύο εντολών, δημιουργήθηκε μία συνάρτηση για τη μεγέθυνση που ονομάστηκε ClickZoomIn και μία για τη σμίκρυνση που ονομάστηκε ClickZoomOut.

```
public void ClickZoomIn(){
```

Η παρακάτω γραμμή κώδικα, μειώνει το οπτικό πεδίο της κάμερας με αποτέλεσμα ο χρήστης να βλέπει το αντικείμενο πιο κοντά.

```
GetComponent<Camera> ().fieldOfView--;
```

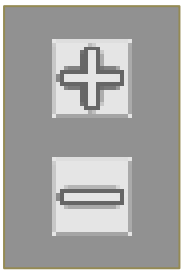
Επιπλέον ορίζεται το ελάχιστο οπτικό πεδίο της κάμερας, έτσι ώστε ο χρήστης να μην μπορεί να κάνει μεγέθυνση πέρα από αυτό.

```
if (GetComponent<Camera> ().fieldOfView < 10.00f) {

GetComponent<Camera> ().fieldOfView = 10.00f;

}

}
```

Για τη σμίκρυνση του αντικειμένου, χρησιμοποιήθηκε αντίστοιχος κώδικας, με τη διαφορά ότι το οπτικό πεδίο της κάμερας μεγαλώνει.

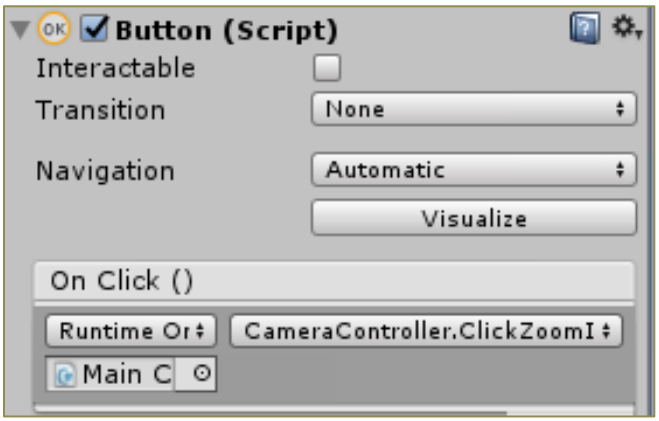
```
public void ClickZoomOut(){
```

```
GetComponent<Camera> ().fieldOfView++;

if (GetComponent<Camera> ().fieldOfView > 90.00f) {

GetComponent<Camera> ().fieldOfView = 90.00f;

}

}
```



Εικόνα 4.10: Κουμπιά Σμίκρυνσης και Μεγέθυνσης

Τα δύο αυτά κουμπιά, δημιουργήθηκαν στο Unity, και στη συνέχεια προστέθηκε η συνάρτηση του κάθε κουμπιού στο script Button (Εικόνα 4.11), το οποίο προστίθεται αυτόματα από το πρόγραμμα, μόλις δημιουργηθεί ένα νέο κουμπί και κάνει την εικόνα να έχει εφέ κουμπιού, δηλαδή ο χρήστης να μπορεί να κάνει κλικ πάνω του.



Εικόνα 4.11: Συνάρτηση Μεγέθυνσης στο script Button.

- Περιστροφή της κάμερας, πατώντας παρατεταμένα το δεξί κλικ.

Για να γίνει η περιστροφή της κάμερας αρχικά ελέγχεται αν ο χρήστης έχει πατήσει το δεξί κουμπί του πληκτρολογίου. Το κουμπί αυτό αντιστοιχεί στον κωδικό 1.

```
if (Input.GetMouseButton (1)) {
```

Με την παρακάτω εντολή, τοποθετείται ο κέρσορας στην μέση της οθόνης.

```
Cursor.SetCursor (mouseRotate, hotspot, cursorMode);
```

Στη συνέχεια υπολογίζεται η περιστροφή κατά τον άξονα X και Y.

```
rotationX += Input.GetAxis ("Mouse X") * sensX * Time.deltaTime;

rotationY += Input.GetAxis ("Mouse Y") * sensY * Time.deltaTime;

rotationX = Mathf.Clamp (rotationX, minX, maxX);

rotationY = Mathf.Clamp (rotationY, minY, maxY);
```

Τέλος, μετατρέπει τη γωνία όσο είναι η περιστροφή που έκανε ο χρήστης με το ποντίκι.

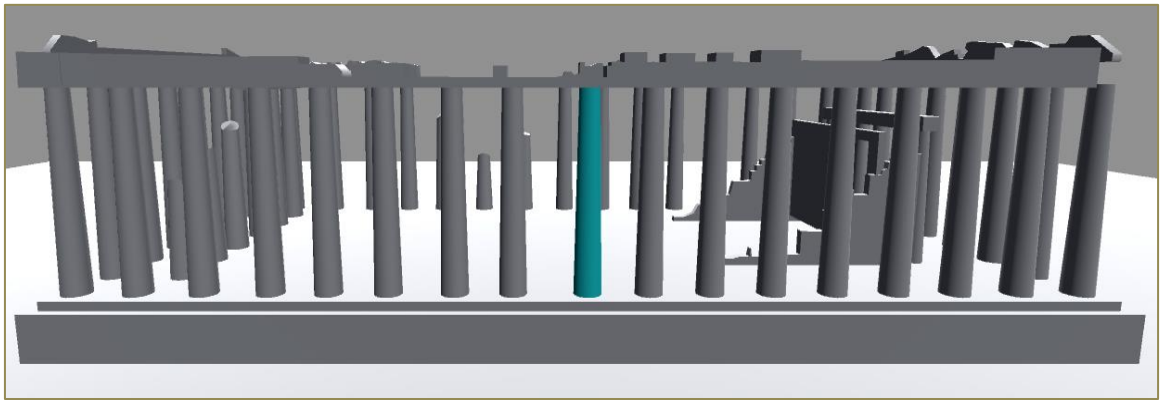
```
transform.localEulerAngles = new Vector3 (-rotationY, -rotationX, 0);

}
```

Σε περίπτωση που ο χρήστης δεν κάνει κάποια αλλαγή, ο κέρσορας μένει στην ίδια θέση.

```
else

Cursor.SetCursor(null, hotspot, cursorMode);
```

d) Επιλογή Αντικειμένων

Η επιλογή μέρους του τρισδιάστατου μοντέλου είναι μία απαραίτητη λειτουργία για την εισαγωγή ετικέτας στο αντικείμενο που επιλέχθηκε. Ο κώδικας που περιγράφεται παρακάτω, δίνει τη δυνατότητα στο χρήστη κάνοντας διπλό κλικ πάνω σε ένα αντικείμενο του τρισδιάστατου μοντέλου, να το επιλέξει και στη συνέχεια να ανοίξει ένα νέο παράθυρο, ώστε να συμπληρώσει τις πληροφορίες του αντικειμένου. Όταν το αντικείμενο επιλεχθεί, αλλάζει το αρχικό του χρώμα και γίνεται κυανό (Εικόνα 4.12), για την αποεπιλογή του αρκεί ο χρήστης να πατήσει από το πληκτρολόγιο το πλήκτρο G. Το περιεχόμενο του νέου παραθύρου που ανοίγει περιγράφεται στην επόμενη παράγραφο.



Εικόνα 4.12: Το επιλεγμένο αντικείμενο εμφανίζεται με κυανό χρώμα.

Σε αυτό το script, αρχικά γίνεται έλεγχος αν ο χρήστης πάτησε δύο φορές το αριστερό κλικ. Αυτό επιτυγχάνεται μετρώντας το χρόνο μεταξύ των δύο κλικ.

```
IEnumerator trapDoubleClicks(float timer)
```

```
{

float endTime = Time.time + timer;

while (Time.time < endTime)

{

if (Input.GetMouseButtonDown(0))

{
```

Όταν ο χρήστης έχει κάνει διπλό κλικ, τότε δημιουργείται ένα νέο παράθυρο.

```
var NewWindow = new NewTable();
```

Με την παρακάτω εντολή εμφανίζεται το νέο παράθυρο, το οποίο είναι αυτό που περιέχει τον πίνακα, μέσα στον οποίο ο χρήστης θα δημιουργεί ετικέτες. Ο κώδικας που αναλύεται στην επόμενη παράγραφο, είναι αυτός που εξηγεί τι περιέχει το νέο αυτό παράθυρο.

```
NewWindow.Show();
```

Στη συνέχεια γίνεται αλλαγή του χρώματος από αυτό που έχει το αντικείμενο σε κυανό.

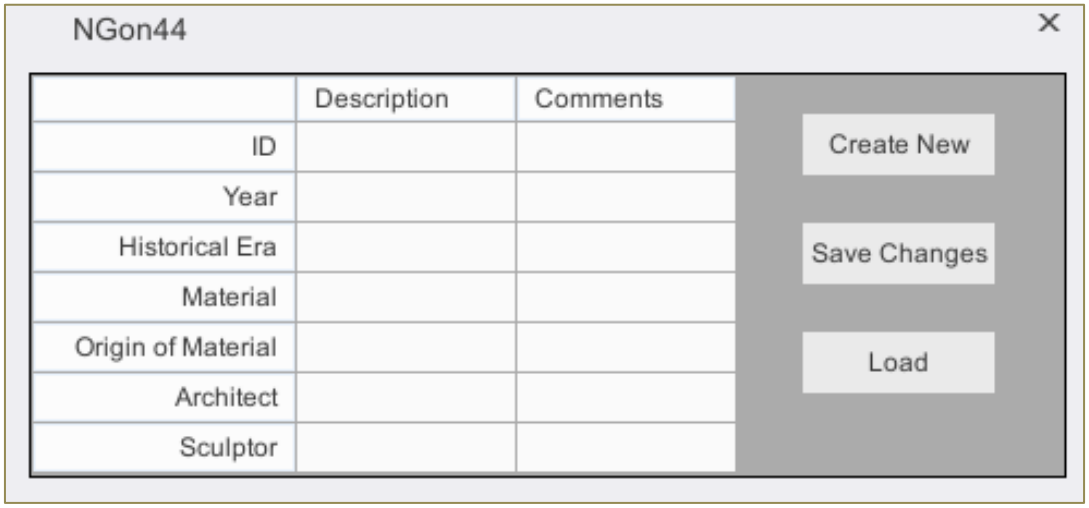
```
startColor = GetComponent<Renderer>().material.color;

GetComponent<Renderer>().material.color = Color.cyan;
```

Το διπλό κλικ ενεργοποιείται μόνο όταν η διαφορά από το ένα πάτημα στο άλλο είναι μικρότερη από 0.4/sec.

```
yield return new WaitForSeconds(0.4f);

clickEnable = true;

}
```

e) Δημιουργία Ετικετών

Για την περίπτωση πολλών πληροφοριών ανά αντικείμενο, θεωρήθηκε καλύτερη επιλογή η απεικόνισή τους σε μορφή πίνακα (Εικόνα 4.13).



Εικόνα 4.13: Κενός Πίνακας Ιδιοτήτων

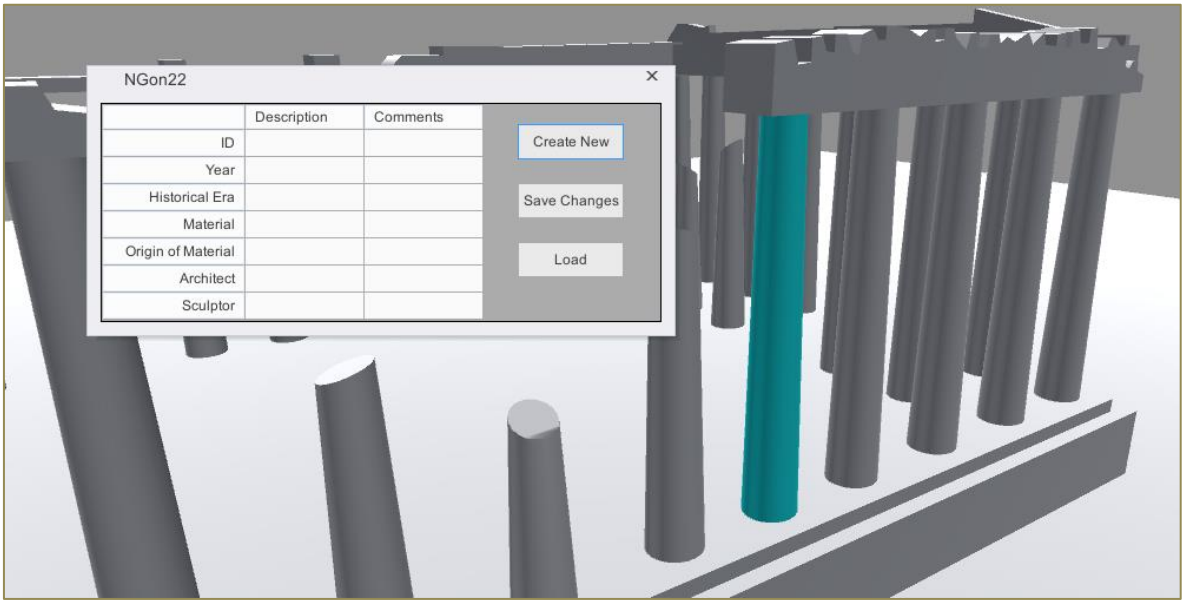
Το script κατασκευής του πίνακα, αρχικά δημιουργεί ένα νέο κενό πίνακα και στη συνέχεια ορίζονται κάποια χαρακτηριστικά για την εμφάνισή του, όπως για παράδειγμα σε ποιο σημείο ανοίγει.

```
var table = new TableView();

table.Anchor = AnchorStyles.Left | AnchorStyles.Right | AnchorStyles.Top | AnchorStyles.Bottom;

table.Location = new Point(12, 32);
```

Στη συνέχεια προστίθεται ο καινούριος πίνακας.

```
Controls.Add(table);
```

Κάνοντας διπλό κλικ πάνω σε οποιοδήποτε σημείο του αντικειμένου, ανοίγει ένας πίνακας ιδιοτήτων. Στην Εικόνα 4.14 παρουσιάζεται το παράθυρο που ανοίγει, με τις ιδιότητες του αντικειμένου που επιλέχθηκε.



Εικόνα 4.14: Πίνακας Ιδιοτήτων Κολόνας.

Πατώντας το κουμπί Create New δημιουργούνται οι γραμμές και οι στήλες του πίνακα. Αυτό γίνεται αρχικά δημιουργώντας το κουμπί και ορίζοντας τα χαρακτηριστικά του, όπως σε ποιο σημείο θα βρίσκεται, τι κείμενο θα γράφει μέσα στο κουμπί κ.α..

```
Button ButonCreateNewTable = new Button();
ButonCreateNewTable.Anchor = AnchorStyles.Right | AnchorStyles.Bottom;
ButonCreateNewTable.Location = new Point(200, 80);
ButonCreateNewTable.TabIndex = 0;
ButonCreateNewTable.Size = new Size(90, 30);
ButonCreateNewTable.Text = "Create New";
```

Με την παρακάτω εντολή, ορίζεται τι θα συμβεί όταν ο χρήστης πατήσει πάνω σε αυτό.

```
ButonCreateNewTable.Click += (object sender, EventArgs args) =>
    {
```

Αρχικά δημιουργούνται οι στήλες.

```
        for (int i = 1; i < 3; i++)

        {
```

Για τη στήλη 1 επιλέχθηκε για τίτλος Περιγραφή, ενώ για τη στήλη 2 επιλέχθηκε ο τίτλος Σχόλιο.

```
            if (i == 1)
                table.Columns.Add(i.ToString(), "Description");

            if (i == 2)
                table.Columns.Add(i.ToString(), "Comments");

        }
```

Στη συνέχεια δημιουργούνται εφτά γραμμές, όσες και οι ιδιότητες που επιλέχθηκαν.

```
        table.Rows.Add(7);
```

```
        table.Refresh();
    };
```

Τέλος προστίθεται το νέο κουμπί.

```
    Controls.Add(ButonCreateNewTable);
```

Για την αποθήκευση των πληροφοριών που εισχωρήθηκαν στα κελιά του πίνακα, αρκεί ο χρήστης να πατήσει το κουμπί save από τον πίνακα. Πατώντας αυτό το κουμπί δημιουργείται αυτόματα ένα νέο αρχείο με ονομασία τον κωδικό του αντικειμένου που επιλέχθηκε και κατάληξη .txt. Το αρχείο αυτό αποθηκεύεται στο φάκελο της εφαρμογής, σε ένα ξεχωριστό φάκελο με την ονομασία Tables, στην επιφάνεια εργασίας του χρήστη.

Ομοίως με το κουμπί *Create New*, αρχικά δημιουργείται ένα νέο κουμπί.

```
Button ButonSaveChanges = new Button();
ButonSaveChanges.Anchor = AnchorStyles.Right | AnchorStyles.Bottom;
ButonSaveChanges.Location = new Point(200, 130);
ButonSaveChanges.TabIndex = 0;
ButonSaveChanges.Size = new Size(90, 30);
ButonSaveChanges.Text = "Save Changes";
ButonSaveChanges.Click += (object sender, EventArgs args) =>
    {
```

Τα δεδομένα του πίνακα καταγράφονται.

```
        var data = "";
        for (int i = 0; i < table.Rows.Count; i++)
        {
            for (int k = 0; k < table.Columns.Count; k++)
            {
                var item = table.Rows[i][k];
                if (item != null)
                    data += item.ToString();

                data += ';';
            }

            data += "\r\n";
        }
```

Ορίζεται η διαδρομή στην οποία θα αποθηκευτεί το αρχείο.

```
string path =
Environment.ExpandEnvironmentVariables(@"C:\Users\%USERNAME%\Desktop\ObjTags\Ta
bles\" + Text + ".txt");
```

Η παρακάτω εντολή αποθηκεύει τα δεδομένα στο αρχείο, στη διαδρομή που ορίστηκε.

```
File.WriteAllText(path,data);
    };

Controls.Add(ButonSaveChanges);
```

Στη συνέχεια, δίνεται η δυνατότητα στο χρήστη είτε να δημιουργήσει ένα νέο αρχείο για το συγκεκριμένο αντικείμενο, είτε να φορτώσει αυτό που ήδη έχει δημιουργηθεί πατώντας το κουμπί *Load*.

Πατώντας το κουμπί *Load*, φορτώνεται αν υπάρχει ο πίνακας που έχει δημιουργηθεί. Σε περίπτωση που δεν υπάρχει αποθηκευμένος πίνακας, αυτό το κουμπί δεν πραγματοποιεί κάποια ενέργεια. Τα βήματα που ακολουθούν για τη δημιουργία του κουμπιού, είναι όμοια με αυτά που ακολουθήθηκαν και για το κουμπί *Create New*.

```
Button ButtonLoad = new Button();
ButtonLoad.Anchor = AnchorStyles.Right | AnchorStyles.Bottom;
ButtonLoad.Location = new Point(200, 180);
ButtonLoad.TabIndex = 0;
ButtonLoad.Size = new Size(90, 30);
ButtonLoad.Text = "Load";
ButtonLoad.Click += (object sender, EventArgs args) =>
      {
string path =
Environment.ExpandEnvironmentVariables(@"C:\Users\%USERNAME%\Desktop\ObjTags\Tables\" + Text + ".txt");
```

Σε αυτήν την περίπτωση, γίνεται αναζήτηση του αρχείου στη διαδρομή που ορίστηκε και το αρχείο διαβάζεται.

```
var lines = File.ReadAllLines(path);
```

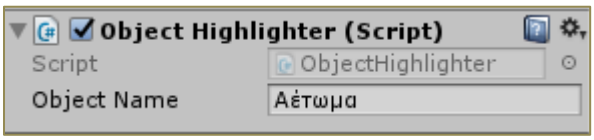
Στη συνέχεια δημιουργείται ένας νέος πίνακας συμπληρωμένος με τα δεδομένα που έχουν αποθηκευτεί.

```
if (lines.Length > 0)
        {
           var columnsCount = lines[0].Count(s => s == ';');
           for (int i = 0; i < columnsCount; i++)
              table.Columns.Add(i.ToString(), i.ToString());

           for (int i = 0; i < lines.Length; i++)
           {
              var items = lines[i].Split(';');

              table.Rows.Add(items);
           }
        }

        table.Refresh();

     };
     Controls.Add(ButtonLoad);

  }

  }
```

Για την περίπτωση που η απεικόνιση της πληροφορίας μπορεί να γίνει και με μία απλή ετικέτα (Εικόνα 4.15), δημιουργήθηκε μία συνάρτηση στο αρχείο κώδικα που επιλέγει τα αντικείμενα.

Εικόνα 4.15: Απλή Ετικέτα Αντικειμένου.

Η συνάρτηση αυτή τοποθετείται στα αντικείμενα του μοντέλου και περιέχει ένα κενό πεδίο για την προσθήκη της πληροφορίας (Εικόνα 4.16).



Εικόνα 4.16: Πεδίο για τη Προσθήκη Νέας Ετικέτας

Ο παρακάτω κώδικας, αυτό που κάνει είναι αν ο χρήστης συμπληρώσει περιεχόμενο στο κενό πεδίο, να εμφανίσει σε ένα νέο κουτάκι, την πληροφορία αυτή.

```
public void DisplayName()
  {
    if (displayObjectName == true)

            {
GUI.Box(new Rect(Event.current.mousePosition.x - 155, Event.current.mousePosition.y, 150,
25), objectName);

    }
  }
```

f) Φόρτωση Νέου Μοντέλου

Για τη δυνατότητα επιλογής ενός νέου μοντέλου, εκτός εφαρμογής, δημιουργήθηκε ένα μενού με τη μορφή του μενού των Windows (Εικόνα 4.17), στο οποίο προστέθηκαν οι επιλογές File και Help. Η κάθε επιλογή δίνει στο χρήστη κάποιες δυνατότητες που αναλύονται στο Κεφάλαιο 2.

Εικόνα 4.17: Μενού.

Για να έχει η εφαρμογή αυτή τη μορφή χρησιμοποιήθηκε ένα πακέτο [29], το οποίο δίνει τη δυνατότητα δημιουργίας βασικών και προσαρμοσμένων στοιχείων ελέγχου.

Το μενού αυτό δημιουργήθηκε και προστέθηκε με τον παρακάτω κώδικα, στον οποίο ορίζονται η θέση, οι διαστάσεις και το χρώμα.

```
MenuStrip menu = new MenuStrip();
```

```
    menu.Anchor = AnchorStyles.Left | AnchorStyles.Right | AnchorStyles.Top | AnchorStyles.Bottom;
    menu.BackColor = BackColor;
    menu.Location = new Point(-2, -2);
    menu.Size = Size;
    Controls.Add(menu);
```

Στη συνέχεια δημιουργήθηκαν τα κουμπιά. Η δημιουργία του πρώτου, δηλαδή του *File*, έγινε ως εξής:

```
var itemFile = new ToolStripMenuItem("File");
    itemFile.BackColor = BackColor;
    itemFile.ForeColor = System.Drawing.Color.Black;
    itemFile.Height = 22;       itemFile.Width = 52;
    menu.Items.Add(itemFile);
```

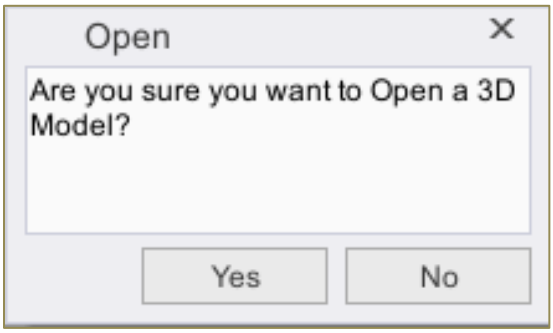
Δηλαδή δημιουργήθηκε το κουμπί και προστέθηκε στη συγκεκριμένη θέση.

Στη συνέχεια προστέθηκαν τα κουμπιά τα οποία ενεργοποιούνται όταν ο χρήστης πατήσει πάνω στο κουμπί *File*.

Το κουμπί *Open* δημιουργήθηκε ως εξής:

```
var itemOpen = new ToolStripMenuItem(“Open”);
    itemFile.DropDownItems.Add(itemOpen);
```

Σε περίπτωση που επιλεχθεί από το χρήστη, ανοίγει ένα παράθυρο το οποίο περιέχει ένα μήνυμα (Εικόνα 4.18).



Εικόνα 4.18: Μήνυμα με Επιλογές Ναι και Όχι.

```
    itemOpen.Click += (s, a) =>
    {
       MessageBoxOpen.Show("Are you sure you want to Open a 3D Model?", "Open");


    };
```

Στο σημείο αυτό με την ενεργοποίηση του *Open*, καλείται από το κώδικα του αρχείου MessageBoxOpen, η συνάρτηση που θα το εμφανίσει. Ο κώδικας αυτού του αρχείου βρίσκεται στο Παράρτημα Ι.

Παρακάτω αναλύονται οι ενέργειες που πραγματοποιούνται, όταν ο χρήστης επιλέξει *Yes* στο μήνυμα.

```
formButtonYes.Click += (object sender, EventArgs args) =>
        {
```

```
        {
```

Αρχικά ανοίγει ένα παράθυρο για την εύρεση του επιθυμητού αρχείου που πρόκειται να φορτωθεί.

```
            var ofd = new OpenFileDialog();
```

Το αρχείο που θα επιλεχθεί θα είναι της μορφής ".obj", ωστόσο μπορεί ο χρήστης να αλλάξει το φίλτρο και να του εμφανιστούν όλα τα αρχεία.

```
Ofd.Filter = “Obj files|*.obj|All files|*.*”;

            ofd.ShowDialog((ofdForm, result) =>
            {

               if (result == DialogResult.OK)
               {
```

Τέλος καλείται η συνάρτηση από ένα τρίτο αρχείο, η οποία ζητάει το όνομα του αρχείου για να το φορτώσει.

```
                                    OBJLoader.LoadOBJFile(ofd.FileName);
               }

            });
```

Ο κώδικας που χρησιμοποιήθηκε για να φορτώνονται τα τρισδιάστατα μοντέλα, διατίθεται δωρεάν στο κατάστημα του Unity [30].

Συμπεράσματα – Προτάσεις

Με την ολοκλήρωση αυτής της Διπλωματικής Εργασίας, δημιουργήθηκε η εφαρμογή που περιγράφηκε η οποία αποτελεί ένα χρήσιμο εργαλείο για την αποθήκευση, την επεξεργασία και την παρουσίαση πληθώρας πληροφοριών ανάλογα με την ανάγκη της εκάστοτε μελέτης. Καθίσταται εφικτή η λεπτομερής παρουσίαση και κατανόηση μέρους ή και της συνολικής εικόνας του αντικειμένου που μελετάται, με αποτέλεσμα τα συμπεράσματα που εξάγονται να διακρίνονται για την ακρίβεια και την πληρότητά τους. Επίσης η χρήση αυτής της εφαρμογής έχει το πλεονέκτημα ότι μπορεί να χρησιμοποιηθεί από μεγάλο εύρος χρηστών και να προσαρμοστεί ανάλογα με τις ανάγκες της εκάστοτε μελέτης.

Επιπλέον, με το αποτέλεσμα της εργασίας αυτής προκύπτει ότι τα διαθέσιμα εργαλεία ελεύθερου λογισμικού, αρκούν για τη δημιουργία μίας τέτοιας εφαρμογής. Ακομή ήταν δυνατή και η παράκαμψη τυχόν δυσκολιών που συναντήθηκαν κατά τη διάρκεια εκπόνησης της, μέσω της μεγάλης κοινότητας χρηστών του λογισμικού Unity αλλά και της γλώσσας προγραμματισμού C#, που υπάρχουν στο διαδίκτυο, οι οποίες συνεχώς αυξάνονται σε χρήστες και εξελίσσονται στην επίλυση ζητημάτων. Ωστόσο ο βέλτιστος τρόπος εκμάθησης, ήταν η συστηματική ενασχόληση με αυτά και η γενικότερη διερεύνηση στο διαδίκτυο.

Ακόμη έχει γίνει προσπάθεια ώστε ο σχεδιασμός της εφαρμογής να γίνει με τέτοιο τρόπο ώστε να είναι ιδιαίτερα εύχρηστη και φιλική προς το χρήστη. Τα περιβάλλον της, βασίζεται σε συνηθισμένες απεικονιστικές μεθόδους με τις οποίες ο χρήστης είναι εξοικειωμένος, λόγω μεγάλου εύρους παρόμοιων παρουσιάσεων σε άλλες εφαρμογές.

Λόγω του μεγάλου εύρους εφαρμογών, η εργασία αυτή έχει αρκετές προοπτικές εξέλιξης ανάλογα με την περίπτωση που πρόκειται να χρησιμοποιηθεί η εφαρμογή. Ενδεικτικά αναφέρονται κάποιες χρήσιμες προτάσεις, ιδιαίτερα για εφαρμογές μηχανικών, όπως για παράδειγμα η προσθήκη γεωγραφικής πληροφορίας του αντικειμένου, η παρουσίαση του μοντέλου σε επίπεδα, η αύξηση των δυνατοτήτων του χρήστη, η επεξεργασία της παρουσίασης του μοντέλου κ.α..

Η εφαρμογή αυτή μπορεί να χρησιμοποιηθεί σε πολλούς τομείς ενισχύοντας τη διαδραστικότητα μεταξύ αντικειμένων και φυσικών προσώπων. Για παράδειγμα ένας τέτοιος τομέας εφαρμογής θα μπορούσε να είναι η εκπαίδευση μέσω παρουσιάσεων των πληροφοριών από τρισδιάστατα μοντέλα, επιταχύνοντας τον χρόνο εκμάθησης, αφού αναπαριστούν την θεωρία με τον πιο παραστατικό τρόπο .

Κλείνοντας, αξίζει να αναφερθεί ότι οι προοπτικές εξέλιξης είναι πολλές καθώς υπάρχει η τάση να διευρυνθούν οι χρήσεις του και σε άλλους τομείς, διότι προσφέρει αμφίδρομη ανταλλαγή πληροφοριών και επικοινωνία.

Βιβλιογραφία

[1] Doulamis, A., Ioannides, M., Doulamis, N., Hadjiprocopis, A., Fritsch, D., Balet, O., Julien, M., Protopapadakis, E., Makantasis, K., Weinlinger,G., S.Johnsons, P., Klein, M. , Fellner, D., Stork, A., Santos, P.: 4D Reconstruction of the Past (2013)

[2] https://www.pcsteps.gr/145776-%CF%80%CF%8E%CF%82-%CF%83%CF%87%CE%B5%CE%B4%CE%B9%CE%AC%CE%B6%CE%BF%CF%85%CE%BC%CE%B5-3d-%CE%BC%CE%BF%CE%BD%CF%84%CE%AD%CE%BB%CE%B1-blender/ (2/2018)

[3] Kyriakaki, G., Doulamis, A., Doulamis, N., Ioannides, M., Makantasis, K., Protopapadakis, E., Hadjiprocopis, A., Wenzel, K., Fritsch, D., Klein, M., Weinlinger, G.: 4D Reconstruction of Tangible Cultural Heritage Objects from Web-Retrieved Images. σελ. 431-451 (2014)

[4] Boehler, W.: Scanning for Cultural Heritage Recording, σελ. 3 (2002)

[5] Pavlidis, G., Koutsoudis, A., Arnaoutoglou, F., Tsioukas, V., Chamzas, C.: Methods for 3D digitization of cultural heritage. J. Cult. Herit. 8(1), σελ. 93–98 (2007)

[6] http://www.ipet.gr/digitech2/index.php?option=com_content&task=view&id=59&Itemid=53 (02/2018)

[7] Georgopoulos, A.: 3D Virtual Reconstrustion of Archaeological Monuments. σελ. 155-164 (2014)

[8] Σημειώσεις Μαθήματος "Αποτυπώσεις Μνημείων ", DOCUMO_01_Μνημεία.pdf, Διαφάνειες Διάλεξης 04.10.2017.

[9] Σημειώσεις Μαθήματος "Αποτυπώσεις Μνημείων ", Γεωδαιτικός_Εξοπλισμός_Δίκτυα.pdf, Διαφάνειες Διάλεξης 18.10.2017 & 25.10.2017

[10] Πατιάς, Π.: Φωτογραμμετρία και Τεκμηρίωση Αρχαιολογικών Χώρων και Ευρημάτων με απλά λόγια. Ανάσκαμμα 2.2008, σελ. 69-79 (2008)

[11] www.charta-von-venedig.de (01/2018)

[12] Άγα, Ε., Γκανιάτσα, Ε., Καρράς, Γ., Μπιτζιλέκη, Χ., Νάκος Β.: Ρεαλιστική Οπτικοποίηση 3D Μοντέλων με ψηφιακές εικόνες σε εφαρμογές μεγάλης κλίμακας.

[13] Τσούλος, Λ., Σκοπελίτη, Α., Στάμου, Λ.: Χαρτογραφική Σύνθεση και Απόδοση σε Ψηφιακό Περιβάλλον. (2015)

[14] Σημειώσεις Μαθήματος "Φωτογραμμετρία ΙΙ", Foto_II_08_2010_DTM.pdf, (2010)

[15] http://buildipedia.com/aec-pros/design-news/the-daily-life-of-building-information-modeling-bim?print=1&tmpl=component

[16] https://www.b2green.gr/el/post/483/building-information-modeling-(bim)-orismos-ta-ofeli-kai-oi-efarmoges

[17] Δημοπούλου, Ε.: nD Κτηματολόγιο. (2015)

[18] https://www.asme.org/engineering-topics/articles/manufacturing-design/top-5-ways-3d-printing-changing-medical-field (02/2018)

[19] https://formlabs.com/blog/3d-printed-medical-models/ (01/2018)

[20] Vermeulen, N., Haddow, G., Seymour, T.Faulkner-Jones, A., Shu, W.: 3D bioprint me: a socioethical view of bioprinting human organs and tissues. σελ. 1-7 (2017)

[21] Doulamis, A., Doulamis, N., Ioannidis, C., Chrysouli, C., Grammalidis, N., Dimitropoulos, K., Potsiou, C., Stathopoulou, E.K. , Ioannides, M.: 5D Modelling: An efficient approach for creating spatiotemporal predictive 3D maps of large-scale cultural resources. σελ. 61–68 (2015)

[22] https://unity3d.com/learn/tutorials/s/scripting

[23] https://forum.unity.com/

[24] https://stackoverflow.com/questions/tagged/c%23

[25] https://social.msdn.microsoft.com/Forums/en-US/home

[26] https://www.visualstudio.com/ (01/2018)

[27] https://store.unity.com/

[28] https://assetstore.unity.com/packages/tools/loading-screen-animation-98505

[29] https://github.com/Meragon/Unity-WinForms (12/2017)

[30] https://assetstore.unity.com/packages/tools/modeling/runtime-obj-importer-49547 (01/2018)

Παράρτημα Ι: Κώδικας Εφαρμογής

Σε αυτό το παράρτημα παρατίθεται όλος ο κώδικας της εφαρμογής μαζί με κάποιες βασικές σημειώσεις.

Κώδικας Φόρτωσης της Εφαρμογής

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

public class loadingtext : MonoBehaviour {

    private RectTransform rectComponent;
    private Image imageComp;

    public float speed = 280f;
    public Text text;
    public Text textNormal;

    [SerializeField]
    private int scene;

    // Use this for initialization
    void Start () {
        rectComponent = GetComponent<RectTransform>();
        imageComp = rectComponent.GetComponent<Image>();
        imageComp.fillAmount = 0.0f;
    }

      // Update is called once per frame
      void Update ()
    {
        int a = 0;
        if (imageComp.fillAmount != 1f)
        {
            imageComp.fillAmount = imageComp.fillAmount + Time.deltaTime *
speed;
            a = (int)(imageComp.fillAmount * 100);
            if (a > 0 && a <= 50)
            {
                textNormal.text = "Loading...";
            }

            else if (a > 50 && a < 100)
            {
                textNormal.text = "Please wait...";
            }
            else {

            }
```

```
            text.text = a + "%";
        }
        else
        {
            imageComp.fillAmount = 0.0f;
            text.text = "0%";
        }
        if (a == 100)
        {
            Application.LoadLevel("scene1");
        }
    }


    }
```

Κώδικας Αρχικού Μενού

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

public class StartMenu : MonoBehaviour {
    public void Architecture()
    {
        UnityEngine.Application.LoadLevel("engineering_architecture");
    }
    public void HumanHeart()
    {
        UnityEngine.Application.LoadLevel("anatomy_heart");
    }
    public void Discovolus()
    {
        UnityEngine.Application.LoadLevel("cultural_heritage_discobolus");
    }
    public void BackToMainMenu()
    {
        UnityEngine.Application.LoadLevel("scene1");
    }
    public void DNA()
    {
        UnityEngine.Application.LoadLevel("education_DNA");
    }
    public void SolarSystem()
    {
        UnityEngine.Application.LoadLevel("education_solar_system");
    }
    public void New()
    {
        UnityEngine.Application.LoadLevel("scene2");
    }
        public void Acropolis()
    {
        UnityEngine.Application.LoadLevel("cultural_heritage_acropolis");
    }
    public void Exit()
    {
        UnityEngine.Application.Quit();
    }

}
```

Κώδικας Περιήγησης Κάμερας

```
//Στο script αυτό χρησιμοποιήθηκαν οι παρακάτω βιβλιοθήκες:

using UnityEngine;

using UnityEngine.SceneManagement;

public class CameraController : MonoBehaviour {

//Δήλωση Μεταβλητών

private float moveSpeed = 130.0f;

private float zoomSpeed = 20.0f;

public float minX = -360.0f;

public float maxX = 360.0f;

public float minY = -45.0f;

public float maxY = 45.0f;

public float sensX = 100.0f;

public float sensY = 100.0f;

float rotationY = 0.0f;

float rotationX = 0.0f;

public Texture2D mouseRotate;

public CursorMode cursorMode=CursorMode.Auto;

private Vector2 hotspot=Vector2.zero; //τοποθετεί τον κέρσορα στο κέντρο της οθόνης


void Update () {

//Περιστροφή της Κάμερας

if (Input.GetMouseButton (1)) {

Cursor.SetCursor (mouseRotate, hotspot, cursorMode);


rotationX += Input.GetAxis ("Mouse X") * sensX * Time.deltaTime;
```

```
rotationY += Input.GetAxis ("Mouse Y") * sensY * Time.deltaTime;


rotationX = Mathf.Clamp (rotationX, minX, maxX); //Περιστοφή κατα Χ

rotationY = Mathf.Clamp (rotationY, minY, maxY); //Περιστροφη κατα Υ


transform.localEulerAngles = new Vector3 (-rotationY, -rotationX, 0);

}

 else

Cursor.SetCursor(null, hotspot, cursorMode);


//Αλλαγή Ύψους Θέσης Κάμερας

float scroll = Input.GetAxis ("Mouse ScrollWheel");

transform.Translate (0, scroll * zoomSpeed, 0, Space.World); //Μετακινεί την κάμερα πάνω ή κατω


//Μετακίνηση στο Χώρο

var x = Input.GetAxis("Horizontal") * Time.deltaTime * moveSpeed; //Μετακίνει δεξιά ή αριστερά

var z = Input.GetAxis("Vertical") * Time.deltaTime * moveSpeed; //Μετακινεί μπροστά και πίσω

transform.Translate(x, 0, z);

}


//Μεγέθυνση

public void ClickZoomIn(){

GetComponent<Camera> ().fieldOfView--;

if (GetComponent<Camera> ().fieldOfView < 10.00f) {

GetComponent<Camera> ().fieldOfView = 10.00f;

}
```

```
}

//Σμίκρυνση

public void ClickZoomOut(){

GetComponent<Camera> ().fieldOfView++;

if (GetComponent<Camera> ().fieldOfView > 90.00f) {

GetComponent<Camera> ().fieldOfView = 90.00f;

}

}

}
```

Κώδικας Επιλογής Αντικειμένου

```
//Στο script αυτό χρησιμοποιήθηκαν οι παρακάτω βιβλιοθήκες:

using System.Collections;

using System.Collections.Generic;

using UnityEngine;

using UnityEngine.EventSystems;

using UnityEngine.SceneManagement;

using System.Windows.Forms;


public class Tags : MonoBehaviour

{

public float DoubleClickTimeout = 0.2f;

private bool clickEnable = true;

private Color startColor;


void Start()

{

startColor = GetComponent<Renderer>().material.color;

MeshCollider mc = gameObject.AddComponent(typeof(MeshCollider)) as MeshCollider;

}


void OnMouseUp()

{

if (clickEnable)

{

clickEnable = false;
```

```
void OnMouseUpAsButton()
{
if (Input.GetMouseButtonDown(0))

GetComponent<Renderer>().material.color = startColor;
}
//Ανίχνευση Διπλού Κλικ
IEnumerator trapDoubleClicks(float timer)
{
float endTime = Time.time + timer;
while (Time.time < endTime)
{

if (Input.GetMouseButtonDown(0))
{
// Εμφάνιση του Νέου Παραθύρου
var NewWindow = new NewTable();
NewWindow.Show();

startColor = GetComponent<Renderer>().material.color;
GetComponent<Renderer>().material.color = Color.cyan;

Debug.Log("Double click!");
yield return new WaitForSeconds(0.4f);
clickEnable = true;
```

```
}
yield return 0;
}
clickEnable = true;
yield return 0;
}
}
```

Κώδικας Δημιουργίας Πίνακα Ετικετών

```
//Στο script αυτό χρησιμοποιήθηκαν οι παρακάτω βιβλιοθήκες:
using System;
using System.Collections.Generic;
using System.Linq;
using System.Text;
using System.Drawing;
using System.Windows.Forms;
using System.IO;


public class NewTable: Form
{

    public NewTable()
    {
      //Στοιχεία του Πίνακα
        var table = new TableView();
       table.Anchor    =    AnchorStyles.Left    |    AnchorStyles.Right    |
      AnchorStyles.Top | AnchorStyles.Bottom;
        table.Location = new Point(12, 32);

        Controls.Add(table);

        table.FillToBottom(12);
        table.FillToRight(12);



        //Προσθήκη τριών κουμπιών:

       //Πρώτο Κουμπί: Δημιουργία Νέου Κενού Πίνακα
        Button ButonCreateNewTable = new Button();
        ButonCreateNewTable.Anchor       =       AnchorStyles.Right       |
AnchorStyles.Bottom;
        ButonCreateNewTable.Location = new Point(200, 80);
        ButonCreateNewTable.TabIndex = 0;
        ButonCreateNewTable.Size = new Size(90, 30);
        ButonCreateNewTable.Text = "Create New";
        ButonCreateNewTable.Click += (object sender, EventArgs args) =>
        {
            //Δημιουργία Πίνακα:

            //Δημιουργία Στηλών

            for (int i = 1; i < 3; i++)

            {
                if (i == 1)
                    table.Columns.Add(i.ToString(), "Description");

                if (i == 2)
                    table.Columns.Add(i.ToString(), "Comments");
```

```
        }

        //Δημιουργία Γραμμών


        table.Rows.Add(7);

        table.Refresh();
    };
    Controls.Add(ButonCreateNewTable);

   //Δεύτερο Κουμπί: Αποθήκευση Υπάρχοντος Πίνακα
    Button ButonSaveChanges = new Button();
    ButonSaveChanges.Anchor = AnchorStyles.Right | AnchorStyles.Bottom;
    ButonSaveChanges.Location = new Point(200, 130);
    ButonSaveChanges.TabIndex = 0;
    ButonSaveChanges.Size = new Size(90, 30);
    ButonSaveChanges.Text = "Save Changes";
    ButonSaveChanges.Click += (object sender, EventArgs args) =>
    {


        var data = "";
        for (int i = 0; i < table.Rows.Count; i++)
        {
            for (int k = 0; k < table.Columns.Count; k++)
            {
                var item = table.Rows[i][k];
                if (item != null)
                    data += item.ToString();

                data += ';';
            }

            data += "\r\n";
        }
              string                          path                          =
Environment.ExpandEnvironmentVariables(@"C:\Users\%USERNAME%\Desktop\
ObjTags\Tables\" + Text + ".txt");
        File.WriteAllText(path,data);


    };
    Controls.Add(ButonSaveChanges);


//Τρίτο Κουμπί: Φόρτωση του Πίνακα αν υπάρχει. Σε περίπτωση που δεν
υπάρχει αποθηκευμένος πίνακας.
   Button ButtonLoad = new Button();
    ButtonLoad.Anchor = AnchorStyles.Right | AnchorStyles.Bottom;
    ButtonLoad.Location = new Point(200, 180);
    ButtonLoad.TabIndex = 0;
    ButtonLoad.Size = new Size(90, 30);
```

```
        ButtonLoad.Text = "Load";
        ButtonLoad.Click += (object sender, EventArgs args) =>
        {
                  string                          path                          =
    Environment.ExpandEnvironmentVariables(@"C:\Users\%USERNAME%\Desktop\
    ObjTags\Tables\" + Text + ".txt");
              var lines = File.ReadAllLines(path);
              if (lines.Length > 0)
              {
                  var columnsCount = lines[0].Count(s => s == ';');
                  for (int i = 0; i < columnsCount; i++)
                      table.Columns.Add(i.ToString(), i.ToString());

                  for (int i = 0; i < lines.Length; i++)
                  {
                      var items = lines[i].Split(';');

                      table.Rows.Add(items);
                  }
              }

              table.Refresh();

        };
        Controls.Add(ButtonLoad);


    }

    }
```

Κώδικας Εισαγωγής Αντικειμένων στην Εφαρμογή

```
using System;
using UnityEngine;
using System.Collections;
using System.Collections.Generic;
using System.Globalization;
using System.IO;
using System.Linq;
using System.ComponentModel;
using System.Text.RegularExpressions;
using System.Drawing;
#if UNITY_EDITOR
using UnityEditor;
#endif

public class OBJLoader : MonoBehaviour

{

    public static bool splitByMaterial = false;
    public static string[] searchPaths = new string[] { "",
"%FileName%_Textures" + Path.DirectorySeparatorChar };
    public float DoubleClickTimeout = 0.2f;
    private bool clickEnable = true;
    private UnityEngine.Color startColor;
    public UnityEngine.Color changedColor = UnityEngine.Color.black;
    public UnityEngine.Color originalColor = UnityEngine.Color.white;

    void Start()
    {

        startColor = GetComponent<Renderer>().material.color;
        MeshCollider mc = gameObject.AddComponent(typeof(MeshCollider)) as
MeshCollider;


    }
    void Awake()
    {
        originalColor = this.GetComponent<Renderer>().material.color;
    }

    void Update()
    {
        if (Input.GetKeyDown(KeyCode.G))
        {
            this.GetComponent<Renderer>().material.color = changedColor;
        }
        else if (Input.GetKeyUp(KeyCode.G))
        {
            this.GetComponent<Renderer>().material.color = originalColor;
        }
    }
```

```
    //structures
    struct OBJFace
    {
        public string materialName;
        public string meshName;
        public int[] indexes;
    }


    //functions
#if UNITY_EDITOR
    [MenuItem("GameObject/Import From OBJ")]
    static void ObjLoadMenu()
    {
        var pth = UnityEditor.EditorUtility.OpenFilePanel("Import OBJ", "",
"obj");
        if (!string.IsNullOrEmpty(pth))
        {
            System.Diagnostics.Stopwatch s = new
System.Diagnostics.Stopwatch();
            s.Start();
            LoadOBJFile(pth);
            Debug.Log("OBJ load took " + s.ElapsedMilliseconds + "ms");
            s.Stop();
        }
    }
#endif

    public static Vector3 ParseVectorFromCMPS(string[] cmps)
    {
        float x = float.Parse(cmps[1]);
        float y = float.Parse(cmps[2]);
        if (cmps.Length == 4)
        {
            float z = float.Parse(cmps[3]) - 3500;
            return new Vector3(x, y, z);
        }
        return new Vector2(x, y);
    }
    public static UnityEngine.Color ParseColorFromCMPS(string[] cmps, float
scalar = 1.0f)
    {
        float Kr = float.Parse(cmps[1]) * scalar;
        float Kg = float.Parse(cmps[2]) * scalar;
        float Kb = float.Parse(cmps[3]) * scalar;
        return new UnityEngine.Color(Kr, Kg, Kb);
    }

    public static string OBJGetFilePath(string path, string basePath,
string fileName)
    {
        foreach (string sp in searchPaths)
```

```
        {
            string s = sp.Replace("%FileName%", fileName);
            if (File.Exists(basePath + s + path))
            {
                return basePath + s + path;
            }
            else if (File.Exists(path))
            {
                return path;
            }
        }

        return null;
    }


    public static Material[] LoadMTLFile(string fn)
    {
        Material currentMaterial = null;
        List<Material> matlList = new List<Material>();
        FileInfo mtlFileInfo = new FileInfo(fn);
        string baseFileName = Path.GetFileNameWithoutExtension(fn);
        string mtlFileDirectory = mtlFileInfo.Directory.FullName +
Path.DirectorySeparatorChar;
        foreach (string ln in File.ReadAllLines(fn))
        {
            string l = ln.Trim().Replace("  ", " ");
            string[] cmps = l.Split(' ');
            string data = l.Remove(0, l.IndexOf(' ') + 1);

            if (cmps[0] == "newmtl")
            {
                if (currentMaterial != null)
                {
                    matlList.Add(currentMaterial);
                }
                currentMaterial = new Material(Shader.Find("Standard
(Specular setup)"));
                currentMaterial.name = data;
            }
            else if (cmps[0] == "Kd")
            {
                currentMaterial.SetColor("_Color",
ParseColorFromCMPS(cmps));
            }
            else if (cmps[0] == "map_Kd")
            {
                //TEXTURE
                string fpth = OBJGetFilePath(data, mtlFileDirectory,
baseFileName);
                if (fpth != null)
                    currentMaterial.SetTexture("_MainTex",
TextureLoader.LoadTexture(fpth));
            }
```

```
            else if (cmps[0] == "map_Bump")
            {
                //TEXTURE
                string fpth = OBJGetFilePath(data, mtlFileDirectory,
baseFileName);
                if (fpth != null)
                {
                    currentMaterial.SetTexture("_BumpMap",
TextureLoader.LoadTexture(fpth, true));
                    currentMaterial.EnableKeyword("_NORMALMAP");
                }
            }
            else if (cmps[0] == "Ks")
            {
                currentMaterial.SetColor("_SpecColor",
ParseColorFromCMPS(cmps));
            }
            else if (cmps[0] == "Ka")
            {
                currentMaterial.SetColor("_EmissionColor",
ParseColorFromCMPS(cmps, 0.05f));
                currentMaterial.EnableKeyword("_EMISSION");
            }
            else if (cmps[0] == "d")
            {
                float visibility = float.Parse(cmps[1]);
                if (visibility < 1)
                {
                    UnityEngine.Color temp = currentMaterial.color;

                    temp.a = visibility;
                    currentMaterial.SetColor("_Color", temp);

                    //TRANSPARENCY ENABLER
                    currentMaterial.SetFloat("_Mode", 3);
                    currentMaterial.SetInt("_SrcBlend",
(int)UnityEngine.Rendering.BlendMode.SrcAlpha);
                    currentMaterial.SetInt("_DstBlend",
(int)UnityEngine.Rendering.BlendMode.OneMinusSrcAlpha);
                    currentMaterial.SetInt("_ZWrite", 0);
                    currentMaterial.DisableKeyword("_ALPHATEST_ON");
                    currentMaterial.EnableKeyword("_ALPHABLEND_ON");
                    currentMaterial.DisableKeyword("_ALPHAPREMULTIPLY_ON");
                    currentMaterial.renderQueue = 3000;
                }

            }
            else if (cmps[0] == "Ns")
            {
                float Ns = float.Parse(cmps[1]);
                Ns = (Ns / 1000);
                currentMaterial.SetFloat("_Glossiness", Ns);

            }
```

```
        }
        if (currentMaterial != null)
        {
            matlList.Add(currentMaterial);
        }
        return matlList.ToArray();
    }

    public static GameObject LoadOBJFile(string fn)
    {


        string meshName = Path.GetFileNameWithoutExtension(fn);

        bool hasNormals = false;
        //OBJ LISTS
        List<Vector3> vertices = new List<Vector3>();
        List<Vector3> normals = new List<Vector3>();
        List<Vector2> uvs = new List<Vector2>();
        //UMESH LISTS
        List<Vector3> uvertices = new List<Vector3>();
        List<Vector3> unormals = new List<Vector3>();
        List<Vector2> uuvs = new List<Vector2>();
        //MESH CONSTRUCTION
        List<string> materialNames = new List<string>();
        List<string> objectNames = new List<string>();
        Dictionary<string, int> hashtable = new Dictionary<string, int>();
        List<OBJFace> faceList = new List<OBJFace>();
        string cmaterial = "";
        string cmesh = "default";
        //CACHE
        Material[] materialCache = null;
        //save this info for later
        FileInfo OBJFileInfo = new FileInfo(fn);


        foreach (string ln in File.ReadAllLines(fn))
        {
            if (ln.Length > 0 && ln[0] != '#')
            {
                string l = ln.Trim().Replace("  ", " ");
                string[] cmps = l.Split(' ');
                string data = l.Remove(0, l.IndexOf(' ') + 1);

                if (cmps[0] == "mtllib")
                {
                    //load cache
                    string pth = OBJGetFilePath(data,
OBJFileInfo.Directory.FullName + Path.DirectorySeparatorChar, meshName);
                    if (pth != null)
                        materialCache = LoadMTLFile(pth);

                }
```

```
                else if ((cmps[0] == "g" || cmps[0] == "o") &&
splitByMaterial == false)
                {
                    cmesh = data;
                    if (!objectNames.Contains(cmesh))
                    {
                        objectNames.Add(cmesh);
                    }
                }
                else if (cmps[0] == "usemtl")
                {
                    cmaterial = data;
                    if (!materialNames.Contains(cmaterial))
                    {
                        materialNames.Add(cmaterial);
                    }

                    if (splitByMaterial)
                    {
                        if (!objectNames.Contains(cmaterial))
                        {
                            objectNames.Add(cmaterial);
                        }
                    }
                }
                else if (cmps[0] == "v")
                {
                    //VERTEX
                    vertices.Add(ParseVectorFromCMPS(cmps));
                }
                else if (cmps[0] == "vn")
                {
                    //VERTEX NORMAL
                    normals.Add(ParseVectorFromCMPS(cmps));
                }
                else if (cmps[0] == "vt")
                {
                    //VERTEX UV
                    uvs.Add(ParseVectorFromCMPS(cmps));
                }
                else if (cmps[0] == "f")
                {
                    int[] indexes = new int[cmps.Length - 1];
                    for (int i = 1; i < cmps.Length; i++)
                    {
                        string felement = cmps[i];
                        int vertexIndex = -1;
                        int normalIndex = -1;
                        int uvIndex = -1;
                        if (felement.Contains("//"))
                        {
                            //doubleslash, no UVS.
                            string[] elementComps = felement.Split('/');
                            vertexIndex = int.Parse(elementComps[0]) - 1;
```

```
                    normalIndex = int.Parse(elementComps[2]) - 1;
                }
                else if (felement.Count(x => x == '/') == 2)
                {
                    //contains everything
                    string[] elementComps = felement.Split('/');
                    vertexIndex = int.Parse(elementComps[0]) - 1;
                    uvIndex = int.Parse(elementComps[1]) - 1;
                    normalIndex = int.Parse(elementComps[2]) - 1;
                }
                else if (!felement.Contains("/"))
                {
                    //just vertex inedx
                    vertexIndex = int.Parse(felement) - 1;
                }
                else
                {
                    //vertex and uv
                    string[] elementComps = felement.Split('/');
                    vertexIndex = int.Parse(elementComps[0]) - 1;
                    uvIndex = int.Parse(elementComps[1]) - 1;
                }
                string hashEntry = vertexIndex + "|" + normalIndex
+ "|" + uvIndex;
                if (hashtable.ContainsKey(hashEntry))
                {
                    indexes[i - 1] = hashtable[hashEntry];
                }
                else
                {
                    //create a new hash entry
                    indexes[i - 1] = hashtable.Count;
                    hashtable[hashEntry] = hashtable.Count;
                    uvertices.Add(vertices[vertexIndex]);
                    if (normalIndex < 0 || (normalIndex >
(normals.Count - 1)))
                    {
                        unormals.Add(Vector3.zero);
                    }
                    else
                    {
                        hasNormals = true;
                        unormals.Add(normals[normalIndex]);
                    }
                    if (uvIndex < 0 || (uvIndex > (uvs.Count - 1)))
                    {
                        uuvs.Add(Vector2.zero);
                    }
                    else
                    {
                        uuvs.Add(uvs[uvIndex]);
                    }

                }
```

```
                    }
                    if (indexes.Length < 5 && indexes.Length >= 3)
                    {
                        OBJFace f1 = new OBJFace();
                        f1.materialName = cmaterial;
                        f1.indexes = new int[] { indexes[0], indexes[1],
indexes[2] };
                        f1.meshName = (splitByMaterial) ? cmaterial :
cmesh;
                        faceList.Add(f1);
                        if (indexes.Length > 3)
                        {

                            OBJFace f2 = new OBJFace();
                            f2.materialName = cmaterial;
                            f2.meshName = (splitByMaterial) ? cmaterial :
cmesh;
                            f2.indexes = new int[] { indexes[2],
indexes[3], indexes[0] };
                            faceList.Add(f2);
                        }
                    }
                }
            }
        }

        if (objectNames.Count == 0)
            objectNames.Add("default");


        //build objects

        GameObject parentObject = new GameObject(meshName);




        foreach (string obj in objectNames)
        {
            GameObject subObject = new GameObject(obj);
            subObject.transform.parent = parentObject.transform;
            subObject.transform.localScale = new Vector3(-1, 1, 1);
            //Create mesh
            Mesh m = new Mesh();
            m.name = obj;
            //LISTS FOR REORDERING
            List<Vector3> processedVertices = new List<Vector3>();
            List<Vector3> processedNormals = new List<Vector3>();
            List<Vector2> processedUVs = new List<Vector2>();
            List<int[]> processedIndexes = new List<int[]>();
            Dictionary<int, int> remapTable = new Dictionary<int, int>();
            //POPULATE MESH
            List<string> meshMaterialNames = new List<string>();
```

```
            OBJFace[] ofaces = faceList.Where(x => x.meshName ==
obj).ToArray();
            foreach (string mn in materialNames)
            {
                OBJFace[] faces = ofaces.Where(x => x.materialName ==
mn).ToArray();
                if (faces.Length > 0)
                {
                    int[] indexes = new int[0];
                    foreach (OBJFace f in faces)
                    {
                        int l = indexes.Length;
                        System.Array.Resize(ref indexes, l +
f.indexes.Length);
                        System.Array.Copy(f.indexes, 0, indexes, l,
f.indexes.Length);
                    }
                    meshMaterialNames.Add(mn);
                    if (m.subMeshCount != meshMaterialNames.Count)
                        m.subMeshCount = meshMaterialNames.Count;

                    for (int i = 0; i < indexes.Length; i++)
                    {
                        int idx = indexes[i];
                        //build remap table
                        if (remapTable.ContainsKey(idx))
                        {
                            //ezpz
                            indexes[i] = remapTable[idx];
                        }
                        else
                        {
                            processedVertices.Add(uvertices[idx]);
                            processedNormals.Add(unormals[idx]);
                            processedUVs.Add(uuvs[idx]);
                            remapTable[idx] = processedVertices.Count - 1;
                            indexes[i] = remapTable[idx];
                        }
                    }

                    processedIndexes.Add(indexes);
                }
                else
                {

                }
            }

            //apply stuff
            m.vertices = processedVertices.ToArray();
            m.normals = processedNormals.ToArray();
            m.uv = processedUVs.ToArray();

            for (int i = 0; i < processedIndexes.Count; i++)
```

```
            {
                m.SetTriangles(processedIndexes[i], i);
            }

            if (!hasNormals)
            {
                m.RecalculateNormals();
            }
            m.RecalculateBounds();
            ;

            MeshFilter mf = subObject.AddComponent<MeshFilter>();
            MeshRenderer mr = subObject.AddComponent<MeshRenderer>();

            Material[] processedMaterials = new
Material[meshMaterialNames.Count];
            for (int i = 0; i < meshMaterialNames.Count; i++)
            {

                if (materialCache == null)
                {
                    processedMaterials[i] = new
Material(Shader.Find("Standard (Specular setup)"));
                }
                else
                {
                    Material mfn = Array.Find(materialCache, x => x.name ==
meshMaterialNames[i]); ;
                    if (mfn == null)
                    {
                        processedMaterials[i] = new
Material(Shader.Find("Standard (Specular setup)"));
                    }
                    else
                    {
                        processedMaterials[i] = mfn;
                    }

                }
                processedMaterials[i].name = meshMaterialNames[i];
            }

            mr.materials = processedMaterials;
            mf.mesh = m;

        }

        return parentObject;



    }
```

Κώδικας Δημιουργίας Μενού με τη Μορφή των Windows

```
using System;
using System.Collections.Generic;
using System.Linq;
using System.Text;
using System.Drawing;
using System.Windows.Forms;
using UnityEngine;



public class FormEx2 : Form
{

    private static int yOffset;
    private readonly object formObjectsWindow;


    public FormEx2()
    {
        Anchor = AnchorStyles.Left | AnchorStyles.Top | AnchorStyles.Right;
        BackColor = System.Drawing.Color.FromArgb(0xF5, 0xF5, 0xF5);
        ControlBox = true; // Close button.
        Location = new Point(0, yOffset);
        Height = 22;
        FormBorderStyle = FormBorderStyle.FixedDialog; //με .none δεν
αλλαζει διαστασεις το πλαισιο



        Width =
System.Windows.Forms.Screen.PrimaryScreen.WorkingArea.Width;

        uwfMovable = false;

        yOffset += Height;


        //-----------------------------------------------------------------
------------------------------

        //Δημιουργία Μενού

        MenuStrip menu = new MenuStrip();
        menu.Anchor = AnchorStyles.Left | AnchorStyles.Right |
AnchorStyles.Top | AnchorStyles.Bottom;
        menu.BackColor = BackColor;
        menu.Location = new Point(-2, -2);
        menu.Size = Size;
        Controls.Add(menu);
```

```
        //-------------------------------------------------------------
--------------------------------

        //Δημιουργία File


        var itemFile = new ToolStripMenuItem("File");
        itemFile.BackColor = BackColor;
        itemFile.ForeColor = System.Drawing.Color.Black;
        itemFile.Height = 22; // There is no AutoSize yet.
        itemFile.Width = 52;
        menu.Items.Add(itemFile);


        //-------------------------------------------------------------
--------------------------------

        //File-->Open


        var itemOpen = new ToolStripMenuItem("Open");
        itemFile.DropDownItems.Add(itemOpen);

        itemOpen.Click += (s, a) =>
        {
            MessageBoxOpen.Show("Are you sure you want to Open a 3D
Model?", "Open");


        };



            //-------------------------------------------------------------
-----------------------------------

        //File-->Exit


        var itemExit = new ToolStripMenuItem("Exit");
        itemExit.Click += (s, a) =>
        {

            MessageBoxQuit.Show("Are you sure you want to Exit?", "Exit");


        };
        itemFile.DropDownItems.Add(itemExit);


         //-------------------------------------------------------------
--------------------------------
        //Δημιουργία Help
```

```
        var itemHelp = new ToolStripMenuItem("Help");
        itemHelp.BackColor = BackColor;
        itemHelp.ForeColor = System.Drawing.Color.Black;
        itemHelp.Height = 22;
        itemHelp.Width = 72;
        menu.Items.Add(itemHelp);

        //---------------------------------------------------------------
--------------------------------
        //Help-->About


        var itemAbout = new ToolStripMenuItem("About");
        itemAbout.Click += (s, a) => { MessageBox.Show("Διπλωματική
Εργασία\nΔαβερώνα Άννα Χριστίνα\nΜάρτιος 2018", "About"); };
        itemHelp.DropDownItems.Add(itemAbout);
    }

}
```

Κώδικας Δημιουργίας Νέου Παραθύρου με Μήνυμα

```
using System;
using System.Collections.Generic;
using System.Linq;
using System.Text;
using System.Drawing;
using System.Windows.Forms;
using UnityEngine;
using UnityEngine.SceneManagement;

namespace System.Windows.Forms
{
    public class MessageBoxOpen
    {
        private static Form _last;


        public static Form Last
        {
            get
            {
                if (_last != null && (_last.IsDisposed || _last.Disposing))
                    _last = null;
                return _last;
            }
            private set { _last = value; }
        }

        public static DialogResult Show(string text)
        {
            return Show(text, "");
        }
        public static DialogResult Show(string text, string caption)
```

```
        {
            return Show(text, caption, new System.Drawing.Font("Arial",
12));
        }
        public static DialogResult Show(string text, string caption,
System.Drawing.Font textFont)
        {
            Form form = new Form();
            form.Size = new Size(220, 96 + 32);
            form.Location = new Point(
                Screen.PrimaryScreen.WorkingArea.Width / 2 - form.Width /
2,
                Screen.PrimaryScreen.WorkingArea.Height / 2 - form.Height /
2);
            form.Text = caption;
            form.TopMost = true;

            GroupBox formGroup = new GroupBox();
            formGroup.Anchor = AnchorStyles.Left | AnchorStyles.Bottom |
AnchorStyles.Right | AnchorStyles.Top;
            formGroup.BackColor = System.Drawing.Color.White;
            formGroup.BorderColor = form.uwfBorderColor;
            formGroup.Size = new Size(form.Width - 16, form.Height -
(int)form.uwfHeaderHeight - 8 - 28);
            formGroup.Location = new Point(8, (int)form.uwfHeaderHeight);

            form.Controls.Add(formGroup);

            TextBox formLabel = new TextBox();
            formLabel.Anchor = AnchorStyles.Left | AnchorStyles.Bottom |
AnchorStyles.Right | AnchorStyles.Top;
            formLabel.Font = textFont;
            formLabel.Size = formGroup.Size;
            formLabel.Text = text;
            formLabel.ReadOnly = false;
            formLabel.Multiline = true;

            formGroup.Controls.Add(formLabel);


            Button formButtonYes = new Button();
            formButtonYes.Anchor = AnchorStyles.Right |
AnchorStyles.Bottom;
            formButtonYes.Location = new Point(form.Width - 90 -
formButtonYes.Width, form.Height - formButtonYes.Height - 8);
            formButtonYes.TabIndex = 0;
            formButtonYes.Text = "Yes";
            formButtonYes.Click += (object sender, EventArgs args) =>
            {
                {
                    var ofd = new OpenFileDialog();
                    ofd.Filter = "Obj files|*.obj|All files|*.*";

                    ofd.ShowDialog((ofdForm, result) =>
```

```
                    {

                        if (result == DialogResult.OK)
                        {


                        OBJLoader.LoadOBJFile(ofd.FileName);
                        }


                    });




                };
                form.Close();
            };

            Button formButtonNo = new Button();
            formButtonNo.Anchor = AnchorStyles.Right | AnchorStyles.Bottom;
            formButtonNo.Location = new Point(form.Width - 8 -
formButtonNo.Width, form.Height - formButtonNo.Height - 8);
            formButtonNo.TabIndex = 0;
            formButtonNo.Text = "No";
            formButtonNo.Click += (object sender, EventArgs args) =>
            {
                form.Close();
            };

            form.Controls.Add(formButtonYes);
            form.Controls.Add(formButtonNo);
            form.Show();

            Last = form;

            return DialogResult.OK;
        }

    }

}
```