Πανεπιστήμιο Πειραιώς – Τμήμα Πληροφορικής

Πρόγραμμα Μεταπτυχιακών Σπουδών

«Πληροφορική»

Μεταπτυχιακή Διατριβή

Τίτλος Διατριβής	**Το πέρασμα από τον φυσικό κόσμο στην εικονική πραγματικότητα.** **From Natural World to Virtual Reality.**
Ονοματεπώνυμο Φοιτητή	**Χαρουτιούν Σπαρταλιάν**
Πατρώνυμο	**Τατίκ**
Αριθμός Μητρώου	**ΜΠΠΛ17051**
Επιβλέπων	**Θεμιστοκλής Παναγιωτόπουλος, Καθηγητής**

Ημερομηνία Παράδοσης **Ιούνιος 2021**

Τριμελής Εξεταστική Επιτροπή

Θ. Παναγιωτόπουλος	Δ. Αποστόλου	Α. Πικράκης
Καθηγητής	Καθηγητής	Επ. Καθηγητής

Εισαγωγή.

Η μεταπτυχιακή αυτή διατριβή χωρίζεται σε 2 σκέλη. Το πρώτο σκέλος αφορά την δημιουργία ενός τρισδιάστατου μοντέλου από οποιοδήποτε αντικείμενο θα μπορούσαμε να έχουμε στα χέρια μας. Το δεύτερο σκέλος αφορά την χρήση αυτού του τρισδιάστατου μοντέλου σε μια εφαρμογή, ένα παιχνίδι εν προκειμένω, ως εικονικό αντικείμενο.

Στο πρώτο σκέλος θα εξετάσουμε τους τρόπους που υπάρχουν στην διάθεση μας, για την δημιουργία τρισδιάστατου μοντέλου με την μέθοδο της φωτογραμμετρίας. Θα αναλύσουμε μερικές από αυτές και θα εμβαθύνουμε σε μια, με την οποία θα δημιουργήσουμε το δικό μας μοντέλο.

Στο δεύτερο μέρος θα περάσουμε στην πλατφόρμα της Unity και με απλά βήματα θα δομήσουμε μια εφαρμογή στην οποία το τρισδιάστατο μοντέλο μας θα έχει πρωταγωνιστικό ρόλο.

Θα δείξουμε βήμα προς βήμα πως εισάγουμε το νέο μοντέλο στο παιχνίδι, πως διαμορφώνουμε το περιβάλλον του παιχνιδιού, πως θα εισάγουμε τα υπόλοιπα μοντέλα στο παιχνίδι μας και πως αυτά θα αλληλοεπιδρούν, καθώς και την λογική που ακολουθήσαμε σε κάθε μας βήμα.

Θα χρειαστεί να γράψουμε κώδικα σε γλώσσα C#, τον οποίο θα εξηγήσουμε αναλυτικά. Με τον κώδικα που θα γράψουμε θα ελέγχουμε την κίνηση του παίχτη μας, την συμπεριφορά των αντικειμένων, το σύστημα συλλογής πόντων και άλλα.

Το αποτέλεσμα είναι ένα παιχνίδι τύπου arcade που μπορεί κανείς να παίξει σε web περιβάλλον στο PC του.

Βασικός στόχος μας είναι να εμφυσήσουμε στον αναγνώστη μας την ιδέα ότι, οτιδήποτε κρατά στο χέρι του, θα μπορούσε να αποτελέσει εικονικό αντικείμενο στην επόμενη εφαρμογή του. Μια εφαρμογή την οποία θα μπορούσε να δημιουργήσει μόνος του, με την χρήση ενός απλού υπολογιστή και μιας φωτογραφικής μηχανής ή ενός κινητού τηλεφώνου. Χωρίς την χρήση εξειδικευμένου δαπανηρού εξοπλισμού και ακριβών προγραμμάτων. Μια καλή αρχή για όποιον θα ήθελε να πειραματιστεί στον χώρο αυτό.

Abstract

This particular postgraduate thesis is divided into two sections. The first one involves the creation of a three-dimensional model of any object we could have around us. The second section includes incorporating this three-dimensional model into an application, a game in this case, as a virtual object.

In the first section, we are going to examine all the available means to create a three-dimensional model, following the method of photogrammetry. We are going to analyze the properties of some and delve into one, which we will use to create our own model.

In the second part, we will access the platform Unity and following simple steps, we will develop an application, in which our three-dimensional model will play a leading part.

More specifically, we will demonstrate how to insert the new model into the game in stages, as well as how to form the game environment, how to incorporate the rest of the models in our game, how these will interact and finally, we will explain the rationale behind each step.

It will be necessary to write a code in C# programming language, which we are going to analytically explain. With the aforementioned code, we will be able to control every move of our player, the behavior of the objects, the system of point collection and many other features.

The result includes an arcade game that can be played on a PC, in a web environment.

Our main objective is to implant the idea that every object that someone might have in their surroundings could constitute a virtual entity in a future application, which they could create on their own, using a PC and a camera or a mobile phone. All the above does not require expensive special equipment or programs. Overall, it is a good start for anyone who would like to experiment in this particular field.

Περιεχόμενα

Φωτογραμμετρία

Η φωτογραμμετρία είναι η επιστήμη και η τεχνολογία της απόκτησης αξιόπιστων πληροφοριών σχετικά με τα φυσικά αντικείμενα και του περιβάλλοντος μέσω της διαδικασίας εγγραφής, μέτρησης και ερμηνείας φωτογραφικών εικόνων.

Η φωτογραμμετρία εμφανίστηκε στα μέσα του 19ου αιώνα, σχεδόν ταυτόχρονα με την εμφάνιση της ίδιας της φωτογραφίας. Η χρήση φωτογραφιών για τη δημιουργία τοπογραφικών χαρτών προτάθηκε για πρώτη φορά από τον Γάλλο επιθεωρητή Dominique F. Arago το 1840 περίπου. Ο όρος φωτογραμμετρία επινοήθηκε από τον Πρώσο αρχιτέκτονα Albrecht Meydenbauer, που εμφανίστηκε στο άρθρο του 1867 «Die Photometrographie».

Υπάρχουν πολλές παραλλαγές της φωτογραμμετρίας. Ένα παράδειγμα είναι η εξαγωγή τρισδιάστατων μετρήσεων από δισδιάστατα δεδομένα (δηλ. Εικόνες). Για παράδειγμα, η απόσταση μεταξύ δύο σημείων που βρίσκονται σε επίπεδο παράλληλο προς το επίπεδο φωτογραφικής εικόνας μπορεί να προσδιοριστεί μετρώντας την απόστασή τους στην εικόνα, εάν είναι γνωστή η κλίμακα της εικόνας. Ένα άλλο είναι κατοπτρική ανάκλαση ή μεταλλικότητα από φωτογραφίες υλικών για σκοπούς φυσικής απόδοσης.

Η φωτογραμμετρία μικρής εμβέλειας αναφέρεται στη συλλογή φωτογραφιών από μικρότερη απόσταση από την παραδοσιακή εναέρια φωτογραμμετρία. Η φωτογραμμετρική ανάλυση μπορεί να εφαρμοστεί σε μία φωτογραφία ή μπορεί να χρησιμοποιήσει φωτογράφηση υψηλής ταχύτητας και τηλεπισκόπηση για να ανιχνεύσει, να μετρήσει και να καταγράψει σύνθετα πεδία κίνησης δισδιάστατων και τρισδιάστατων μοντέλων.

Μέθοδοι φωτογραμμετρίας

Η φωτογραμμετρία χρησιμοποιεί μεθόδους από πολλούς κλάδους, όπως οπτική και προβολική γεωμετρία. Η ψηφιακή λήψη εικόνας και η φωτογραμμετρική επεξεργασία περιλαμβάνουν πολλά στάδια, τα οποία επιτρέπουν τη δημιουργία ψηφιακών μοντέλων 2D ή 3D του αντικειμένου ως τελικό προϊόν.

Οι αλγόριθμοι φωτογραμμετρίας συνήθως προσπαθούν να ελαχιστοποιήσουν το άθροισμα των τετραγώνων σφαλμάτων στις συντεταγμένες και τις σχετικές μετατοπίσεις των σημείων αναφοράς. Αυτή η ελαχιστοποίηση είναι γνωστή ως ρύθμιση δέσμης και εκτελείται συχνά χρησιμοποιώντας τον αλγόριθμο Levenberg – Marquardt.

Στερεοφωτογραμμετρία

Μια ειδική περίπτωση, που ονομάζεται στερεοφωτογραμμετρία, περιλαμβάνει την εκτίμηση των τρισδιάστατων συντεταγμένων σημείων σε ένα αντικείμενο που χρησιμοποιεί μετρήσεις που γίνονται σε δύο ή περισσότερες φωτογραφικές εικόνες που λαμβάνονται από διαφορετικές θέσεις (βλέπε στερεοσκοπία). Τα κοινά σημεία προσδιορίζονται σε κάθε εικόνα. Μια γραμμή όρασης (ή ακτίνας) μπορεί να κατασκευαστεί από τη θέση της κάμερας έως το σημείο του αντικειμένου. Είναι η τομή αυτών των ακτίνων (τριγωνοποίηση) που καθορίζει την τρισδιάστατη θέση του σημείου. Οι πιο εξελιγμένοι αλγόριθμοι μπορούν να εκμεταλλευτούν άλλες πληροφορίες σχετικά με τη σκηνή που είναι γνωστή εκ των προτέρων, για παράδειγμα συμμετρίες, σε ορισμένες περιπτώσεις επιτρέποντας ανακατασκευές τρισδιάστατων συντεταγμένων από μία μόνο θέση κάμερας. Η στερεοφωτομετρία εμφανίζεται ως μια ισχυρή τεχνική μέτρησης χωρίς επαφή για τον προσδιορισμό δυναμικών χαρακτηριστικών και μορφών λειτουργίας κατασκευών.

Εφαρμογές

Η φωτογραμμετρία χρησιμοποιείται σε τομείς όπως η τοπογραφική χαρτογράφηση, η αρχιτεκτονική, η μηχανική, η κατασκευή, ο ποιοτικός έλεγχος, η αστυνομική έρευνα, η πολιτιστική κληρονομιά και η γεωλογία. Οι αρχαιολόγοι το χρησιμοποιούν για να παράγουν γρήγορα σχέδια μεγάλων ή πολύπλοκων τοποθεσιών, και οι μετεωρολόγοι το χρησιμοποιούν για να προσδιορίσουν την ταχύτητα του ανέμου των ανεμοστρόβιλων όταν δεν μπορούν να ληφθούν αντικειμενικά καιρικά δεδομένα. Η φωτογραμμετρία χρησιμοποιήθηκε εκτενώς για τη δημιουργία φωτορεαλιστικών περιβαλλοντικών στοιχείων για βιντεοπαιχνίδια.

Χαρτογράφηση

Η φωτογράφηση είναι η διαδικασία δημιουργίας ενός χάρτη με "χαρτογραφικές βελτιώσεις" που έχουν σχεδιαστεί από ένα «μωσαϊκό» φωτογραφιών που είναι μια σύνθετη φωτογραφική εικόνα του εδάφους, ή πιο συγκεκριμένα, ως ένα ελεγχόμενο φωτο-μωσαϊκό όπου μεμονωμένες φωτογραφίες διορθώνονται για κλίση και φέρονται σε κοινή κλίμακα (τουλάχιστον σε ορισμένα

σημεία ελέγχου). Η διόρθωση των εικόνων επιτυγχάνεται γενικά με την προσαρμογή των προβαλλόμενων εικόνων κάθε φωτογραφίας σε ένα σύνολο τεσσάρων σημείων ελέγχου των οποίων οι θέσεις έχουν προέλθει από έναν υπάρχοντα χάρτη ή από μετρήσεις εδάφους. Όταν, αυτές οι διορθωμένες, κλιμακωτές φωτογραφίες τοποθετούνται σε ένα πλέγμα σημείων ελέγχου, μια καλή αντιστοίχιση μπορεί να επιτευχθεί. Η φωτογραμμετρία φαίνεται να είναι ο μόνος τρόπος για να εκμεταλλευτούμε μελλοντικά πηγές δεδομένων, όπως εικόνες από αεροσκάφη μεγάλου υψομέτρου και δορυφορικές εικόνες.

Αρχαιολογία

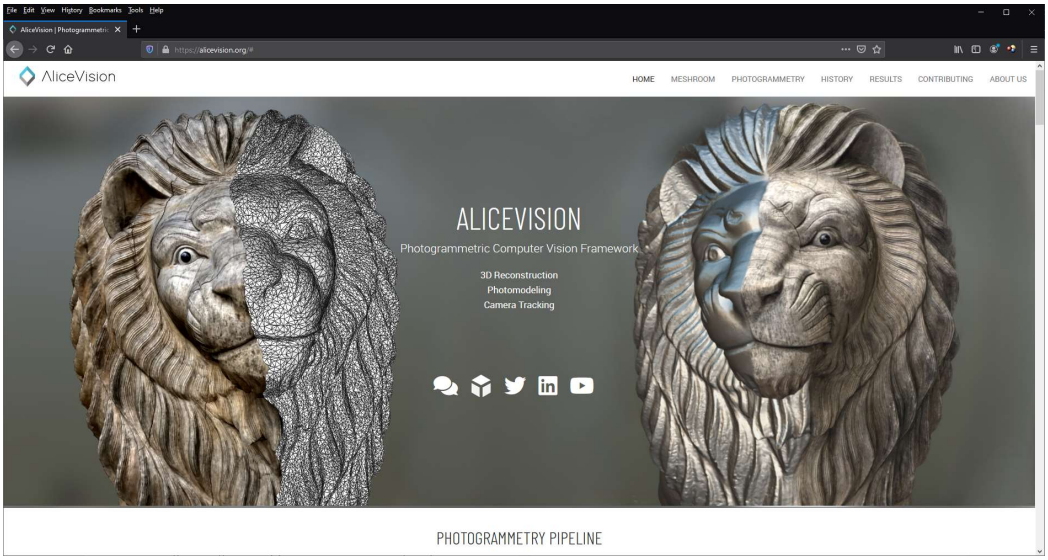
Η εναέρια φωτογραφία έχει εφαρμοστεί ευρέως για τη χαρτογράφηση υπολειμμάτων επιφανειών και εκθέσεων ανασκαφών σε αρχαιολογικούς χώρους. Η φωτογραμμετρία χρησιμοποιείται ολοένα και περισσότερο στη θαλάσσια αρχαιολογία λόγω της σχετικής ευκολίας χαρτογράφησης τοποθεσιών σε σύγκριση με τις παραδοσιακές μεθόδους, επιτρέποντας τη δημιουργία τρισδιάστατων χαρτών που μπορούν να αποδοθούν στην εικονική πραγματικότητα.

3D μοντελοποίηση

Η εφαρμογή της φωτογραμμετρίας στην δημιουργία τρισδιαστατων μοντέλων είναι και αυτή που θα μας απασχολήσει κυρίως στην μελέτη μας στην εργασία αυτή. Είναι μια κάπως παρόμοια εφαρμογή με αυτή της αρχαιολογίας, και είναι η σάρωση αντικειμένων για αυτόματη δημιουργία τρισδιάστατων μοντέλων από αυτά. Το παραγόμενο μοντέλο συχνά εξακολουθεί να περιέχει κενά, οπότε απαιτείται επιπλέον καθαρισμός με λογισμικό όπως το MeshLab, το netfabb ή το MeshMixer. Κάποια από αυτά τα λογισμικά θα αναλυθούν παρακάτω στην εργασία μας. Γενικά η φωτογραμμετρία κερδίζει καθημερινά έδαφος και εξαπλώνεται σε διάφορες εφαρμογές. Ακόμη και το γνωστό σε όλους μας Google Earth χρησιμοποιεί φωτογραμμετρία για τη δημιουργία τρισδιάστατων εικόνων.

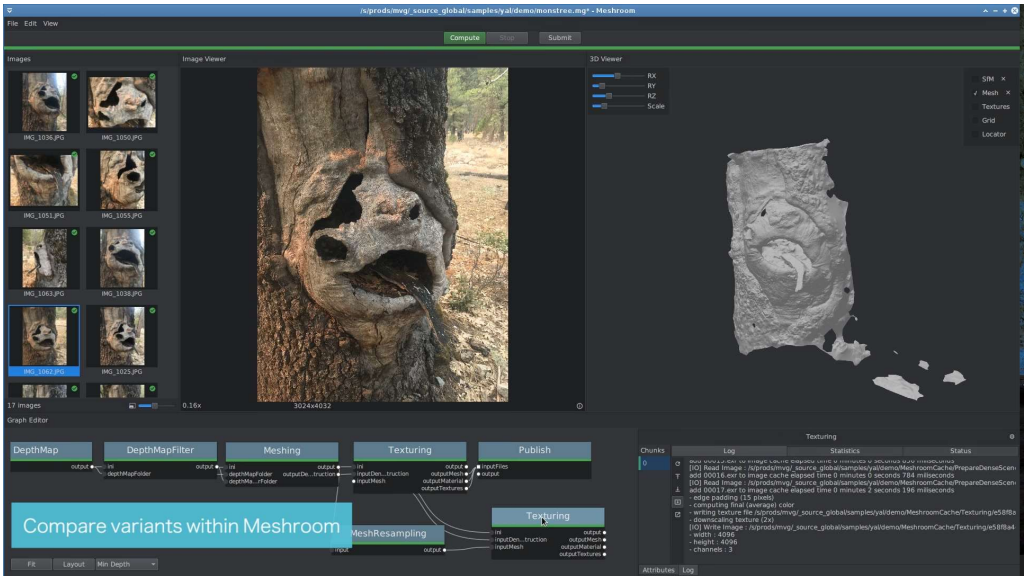
Προγράμματα Φωτογραμμετρίας

Meshroom



Το Meshroom είναι ένα πρόγραμμα φωτογραμμετρίας που κατασκευάστηκε από την AliceVision. Ο στόχος του Meshroom είναι η εξαγωγή διακριτικών ομάδων εικονοστοιχείων που, σε κάποιο βαθμό, είναι αναλλοίωτα στην αλλαγή απόψεων κάμερας κατά τη λήψη εικόνων. Ως εκ τούτου, ένα χαρακτηριστικό στη σκηνή θα πρέπει να έχει παρόμοιες περιγραφές χαρακτηριστικών σε όλες τις εικόνες.

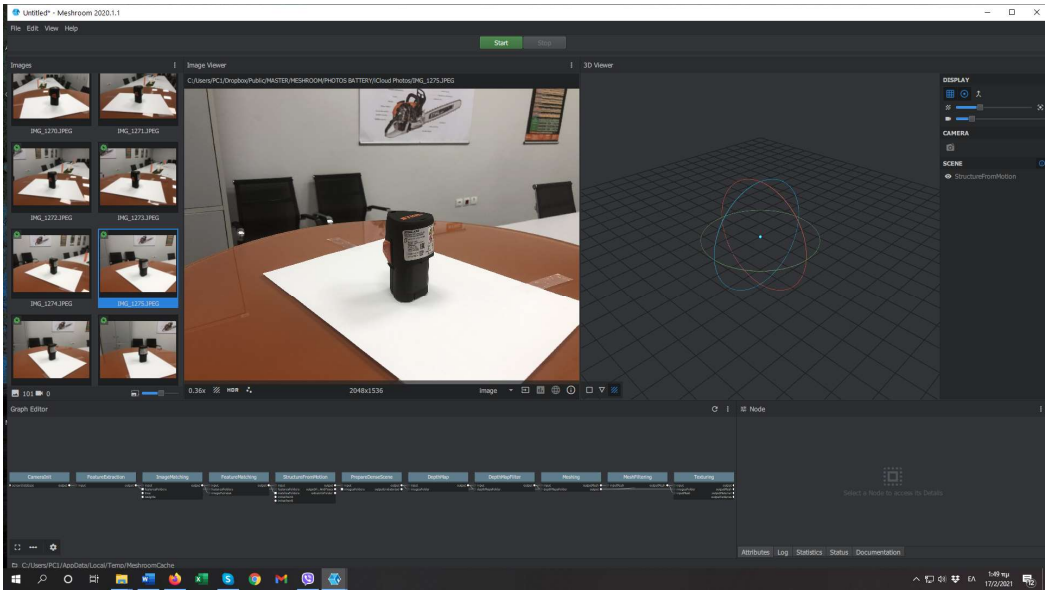
Η πιο γνωστή μέθοδος ανίχνευσης χαρακτηριστικών είναι ο αλγόριθμος SIFT (Scale-invariant feature transform). Ο αρχικός στόχος του SIFT είναι η εξαγωγή διακριτικών επιδιορθώσεων σε μια πρώτη εικόνα που μπορεί να συγκριθεί με διακριτικά στοιχεία μιας δεύτερης εικόνας ανεξάρτητα από την περιστροφή, τη μετάφραση και την κλίμακα. Δεδομένου ότι μια σχετική λεπτομέρεια υπάρχει μόνο σε μια συγκεκριμένη κλίμακα, τα εξαγόμενα στοιχεία επικεντρώνονται σε σταθερά σημεία ενδιαφέροντος.



Η βασική ιδέα είναι ότι, σε κάποιο βαθμό, μπορεί κανείς να χρησιμοποιήσει την αναλλοίωτη SIFT για να αντιμετωπίσει τους μετασχηματισμούς εικόνας που συμβαίνουν όταν οι απόψεις αλλάζουν κατά τη λήψη εικόνας. Η αναπαράσταση μιας εικόνας σε διαφορετικές

κλίμακες, πραγματοποιείται τεχνικά με τον υπολογισμό μιας πυραμίδας από εικόνες χαμηλής κλίμακας. Κάθε εικόνα δημιουργεί διάφορα σημεία αναφοράς. Το κάθε σημείο αναφοράς σχηματίζεται από την περιγραφή των υπολογισμών αυτών των μετασχηματισμών.

Η περιγραφή, η οποία συνήθως αποθηκεύεται σε 128 bit, αποτελείται από στατιστικά στοιχεία των βαθμίδων που υπολογίζονται σε περιοχές γύρω από το σημείο αναφοράς. Το μέγεθος της περιοχής καθορίζεται από την κλίμακα του σημείου αναφοράς, και ο προσανατολισμός καθορίζεται από τον κυρίαρχο άξονα.

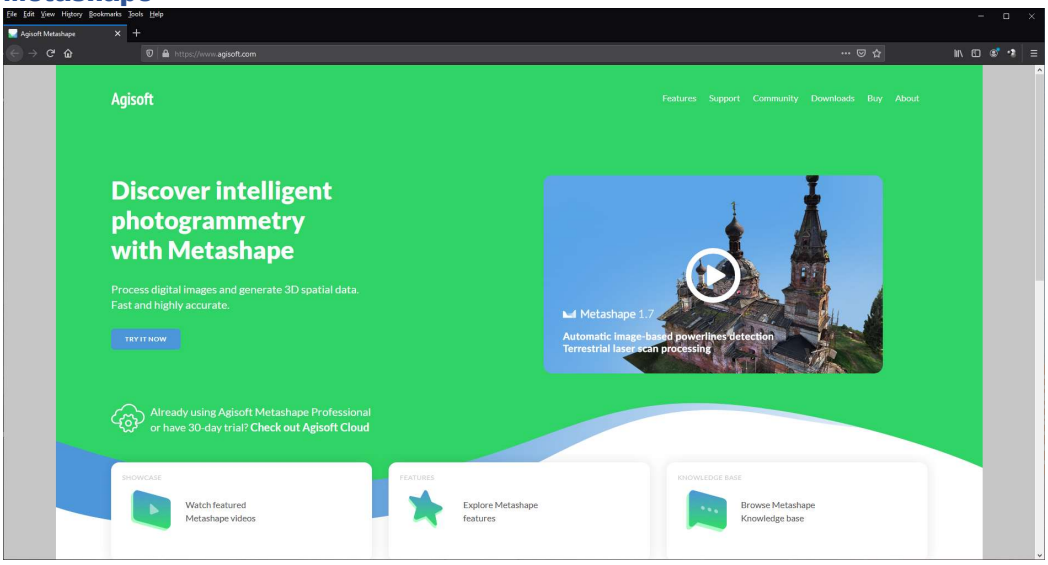


Το Meshroom είναι δωρεάν αλλά για να λειτουργήσει απαιτεί αυξημένες δυνατότητες από την κάρτα γραφικών του υπολογιστή μας. Για την ακρίβεια απαιτεί κάρτα γραφικών με ενεργοποιημένο NVIDIA CUDA cuda-10 συμβατό με υπολογιστική δυνατότητα από 3.0 έως 7.5. Διαφορετικά η διαδικασία δημιουργίας του μοντέλου αποτυγχάνει.

Το πρόγραμμα, καθώς και όλες τις απαραίτητες πληροφορίες μπορεί καείς να τα βρει στην παρακάτω διεύθυνση:

https://alicevision.org/#meshroom

Metashape

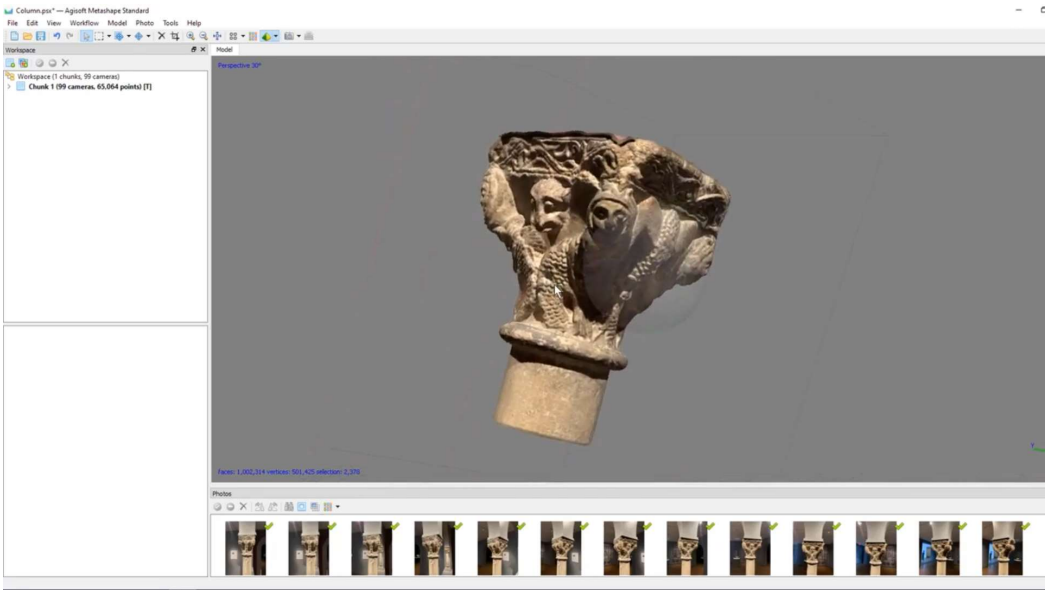


Σε γενικές γραμμές, ένα έργο επεξεργασίας φωτογραμμετρίας στο Metashape είναι η κατασκευή ενός τρισδιάστατου μοντέλου με υφή. Η διαδικασία επεξεργασίας δεδομένων εικόνας με το Agisoft Metashape αποτελείται από δύο κύρια βήματα.

1. Το πρώτο βήμα ονομάζεται ευθυγράμμιση. Περιλαμβάνει εναέριο τριγωνισμό (AT) και ρύθμιση δέσμης (BBA). Σε αυτό το στάδιο, το Metashape αναζητά σημεία δυνατοτήτων στις εικόνες και ταιριάζει σε όλες τις εικόνες σε σημεία αναφοράς. Το πρόγραμμα βρίσκει επίσης τη θέση της κάμερας για κάθε εικόνα και βελτιώνει τις παραμέτρους βαθμονόμησης της κάμερας. Τα αποτελέσματα αυτών των διαδικασιών απεικονίζονται με τη μορφή ενός αραιού σημείου νέφους και ενός συνόλου καμεραθέσεις. Το σύννεφο αραιών σημείων αντιπροσωπεύει τα αποτελέσματα της ευθυγράμμισης της εικόνας και δεν θα χρησιμοποιηθεί άμεσα σε περαιτέρω επεξεργασία Αλλά το αραιό σημείο cloudis είναι απαραίτητο για τον προσδιορισμό των χαρτών. Ωστόσο, μπορεί να εξαχθεί για περαιτέρω χρήση σε εξωτερικά προγράμματα.

2. Το δεύτερο βήμα είναι η δημιουργία επιφάνειας σε 3D (πλέγμα) Το πολυγωνικό μοντέλο (πλέγμα) μπορεί να έχει υφή για φωτορεαλιστική ψηφιακή αναπαράσταση του αντικειμένου / σκηνής και να εξάγεται σε πολλές μορφές συμβατές με λογισμικό μετά την επεξεργασία, τόσο για ροές εργασίας CAD όσο και για μοντελοποίηση 3D. Το cloud πυκνό σημείο μπορεί να κατασκευαστεί από το Metashape με βάση τις εκτιμώμενες θέσεις της κάμερας.

Το πρόγραμμα έχει δοκιμαστική έκδοση 30 ημερών η οποία είναι πλήρως λειτουργική. Από εκεί και πέρα για την χρήση του προγράμματος χρειάζεται η αγορά μιας άδειας από την κατασκευάστρια εταιρία. Οι άδειες μπορεί να είναι ατομικές, ομαδικές ή εκπαιδευτικού τύπου που μπορούν να αγοραστούν από εκπαιδευτικά ιδρύματα.



Οι απαιτήσεις συστήματος για να λειτουργήσει εύρυθμα το Metashape παρατίθενται παρακάτω:

Ελάχιστη διαμόρφωση

• Windows 7 SP 1 ή μεταγενέστερη έκδοση (64 bit), Mac OS X High Sierra ή μεταγενέστερη έκδοση, Debian / Ubuntu με GLIBC 2.13+ (64 bit)

• Επεξεργαστής Intel Core 2 Duo ή ισοδύναμο

• 4 GB RAM

Συνιστώμενη διαμόρφωση

• Windows 7 SP 1 ή μεταγενέστερη έκδοση (64 bit), Mac OS X Mojave ή μεταγενέστερη έκδοση, Debian / Ubuntu με GLIBC 2.13+ (64 bit)

• Επεξεργαστής Intel Core i7 ή AMD Ryzen 7

• 32 GB RAM

Συστάσεις GPU

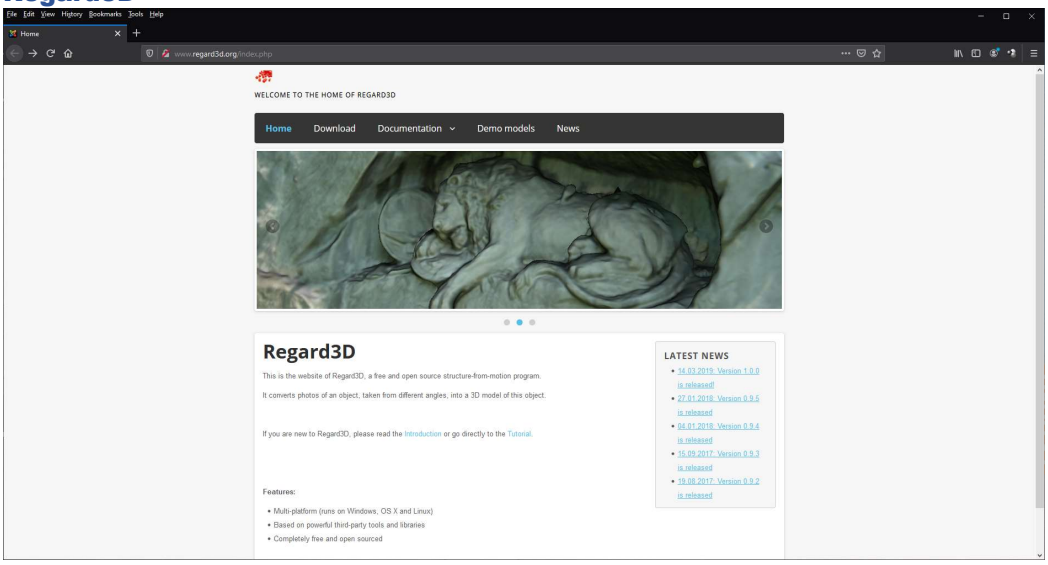
Το Metashape υποστηρίζει ταχεία αντιστοίχιση εικόνων ανακατασκευή χαρτών βάθους χάρτες βάθους με βάση τη δημιουργία μοντέλου ανάμειξη υφής; φωτοσυνεπής λειτουργία βελτίωσης πλέγματος λόγω εκμετάλλευσης γραφικού υλικού (GPU). Το Metashape για να λειτουργήσει απαιτεί αυξημένες δυνατότητες από την κάρτα γραφικών του υπολογιστή μας. Συγκεκριμένα μια από τις παρακάτω σειρές είναι απαραίτητο να υπάρχουν.

- NVIDIA GeForce GTX 6xx series and later with CUDA support.
- AMD Radeon R9 series and later with OpenCL 1.1 support.

Το πρόγραμμα, καθώς και όλες τις απαραίτητες πληροφορίες μπορεί καείς να τα βρει στην παρακάτω διεύθυνση:

https://www.agisoft.com/

Regard3D



Το Regard3D είναι ένα πρόγραμμα που μπορεί να δημιουργήσει μοντέλα 3D από αντικείμενα χρησιμοποιώντας μια σειρά φωτογραφιών που λαμβάνονται από αυτό το αντικείμενο από διαφορετικές οπτικές γωνίες.

Για να επιτευχθεί αυτό το Regard3D εκτελεί τα ακόλουθα βήματα:

Για κάθε εικόνα, εντοπίζονται σημεία-κλειδιά. Τέτοια χαρακτηριστικά σημεία σε ένα αντικείμενο είναι αυτά που έχουν μεγάλη πιθανότητα να βρεθούν σε διαφορετικές εικόνες του ίδιου αντικειμένου, για παράδειγμα γωνίες, άκρα κ.λπ.

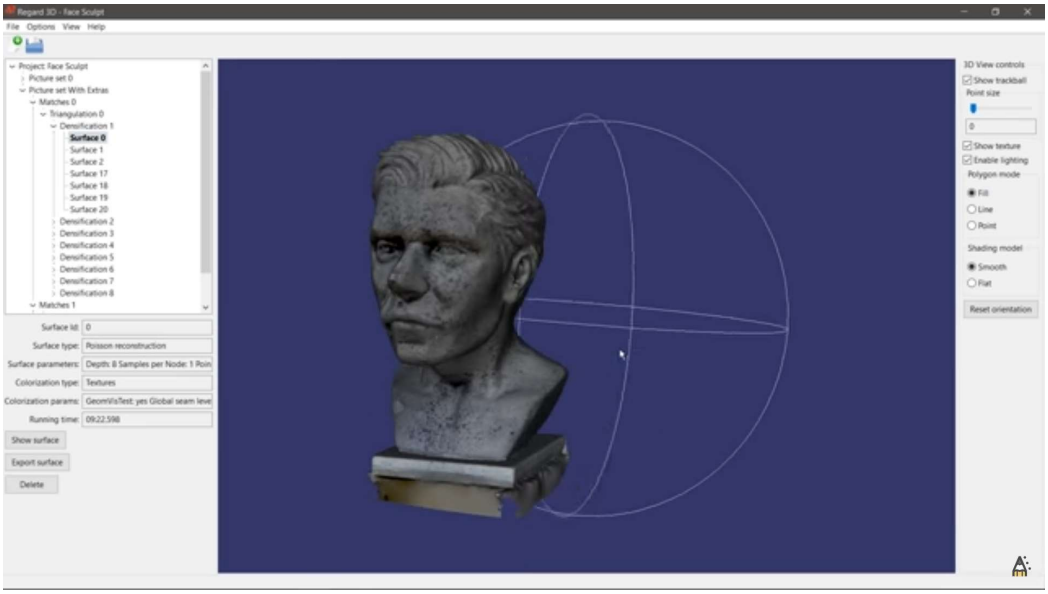
Για κάθε χαρακτηριστικό σημείο, υπολογίζεται ένας μαθηματικός περιγραφέας. Οι περιγραφείς του ίδιου σημείου σε ένα αντικείμενο σε διαφορετικές εικόνες (φαίνεται από διαφορετικές οπτικές γωνίες) είναι παρόμοιοι. Το Regard3D χρησιμοποιεί το LIOP (Local Order Intensity Order) για το σκοπό αυτό. Οι περιγραφείς από διαφορετικές εικόνες ταιριάζουν και φιλτράρονται γεωμετρικά. Το αποτέλεσμα αυτού του βήματος είναι μια συλλογή «ταιριασμένων σημείων» μεταξύ κάθε ζεύγους εικόνων.

Το επόμενο βήμα είναι η φάση τριγωνισμού. Όλα τα ταιριασμένα σημεία όλων των ζευγών εικόνων χρησιμοποιούνται για τον υπολογισμό:

- Η τρισδιάστατη θέση και το χαρακτηριστικό της "κάμερας", δηλαδή όπου τραβήχτηκε κάθε εικόνα και τα οπτικά χαρακτηριστικά της κάμερας.
- Υπολογίζεται η τρισδιάστατη θέση κάθε " ταιριασμένου σημείου ".

Το αποτέλεσμα της τριγωνικής φάσης είναι ένα αραιό νέφος σημείου.

Το τελευταίο βήμα ονομάζεται "Surface Generation". Τα σημειακά σύννεφα χρησιμοποιούνται για τη δημιουργία μιας επιφάνειας, είτε με χρωματιστές κορυφές είτε με υφή. Με το βήμα αυτό ολοκληρώνεται το τρισδιάστατο μοντέλο.



Στην καλύτερη εφαρμογή των παραπάνω βημάτων βοηθούν οι παρακάτω προϋποθέσεις:

- Περισσότερες εικόνες είναι καλύτερες.
- Όλες οι εικόνες πρέπει να λαμβάνονται από μια ελαφρώς διαφορετική άποψη από τις άλλες.
- Οι εικόνες υψηλότερης ανάλυσης εικόνες θα παράγουν ένα πιο λεπτομερές μοντέλο.
- Το αντικείμενο θα πρέπει να φωτογραφηθεί από όλες τις πιθανές γωνίες για να αποφευχθούν τρύπες στο μοντέλο που προκύπτει.

Το Regard3D είναι εντελώς δωρεάν για χρήση. Όλες οι εργασίες που έχουν δημιουργηθεί με το Regard3D μπορούν να χρησιμοποιηθούν ελεύθερα, για εμπορικούς ή μη εμπορικούς σκοπούς.

Οι απαιτήσεις συστήματος για να λειτουργήσει εύρυθμα το Regard3D παρατίθενται παρακάτω:

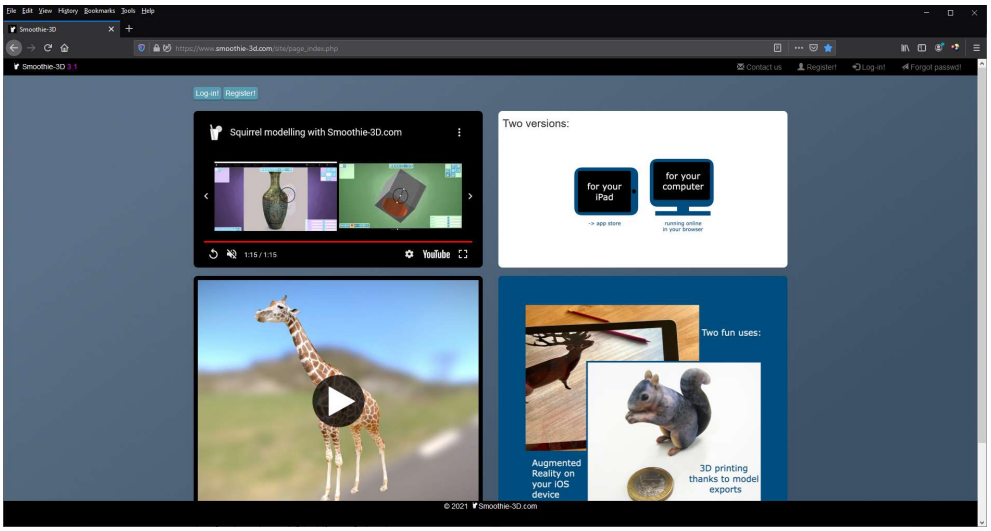
- Το Regard3D είναι ένα πρόγραμμα μόνο 64 bit. Τρέχει μόνο σε εκδόσεις 64 bit των Windows και Mac OS X
- Windows 7 ή νεότερα (δεν εκτελείται σε Windows XP, Vista δεν έχουν δοκιμαστεί)
- Mac OS X 10.7 ή νεότερο
- Κάρτα γραφικών με δυνατότητα OpenGL / chip
- Μεγαλύτερα έργα (πολλές εικόνες υψηλής ανάλυσης) απαιτούν γρήγορο υπολογιστή με πολύ μνήμη RAM (ελάχιστο 4 GB, συνιστάται 8 GB ή περισσότερο)

Το πρόγραμμα, καθώς και όλες τις απαραίτητες πληροφορίες μπορεί καείς να τα βρει στην παρακάτω διεύθυνση:

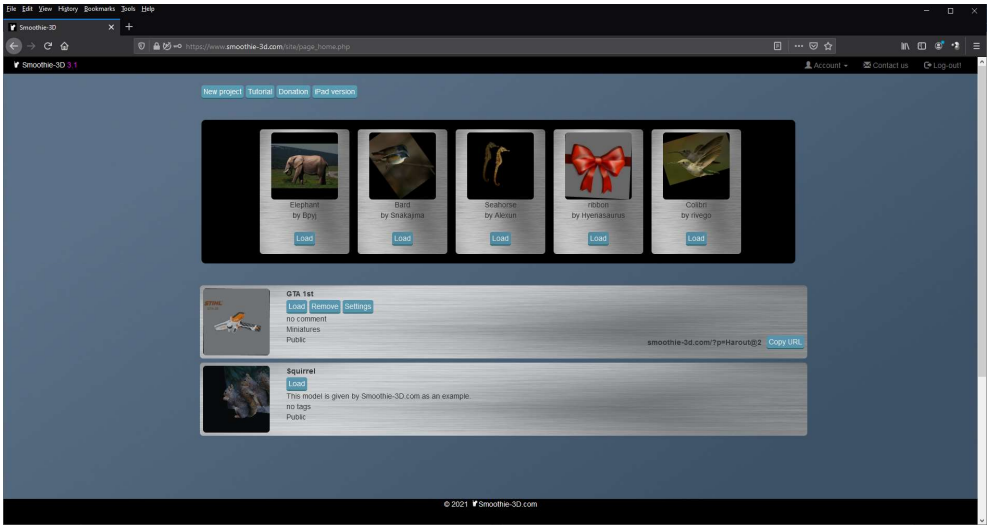
http://www.regard3d.org/index.php

Smoothie-3D

Το Smoothie-3D είναι ένα Web-Based εργαλείο στο οποίο κανείς μπορεί να δημιουργήσει ένα τρισδιάστατο αντικείμενο από δισδιάστατες φωτογραφίες – απεικονίσεις του αντικειμένου αυτού.



Όλη η επεξεργασία γίνεται στην ιστοσελίδα με πόρους που παρέχει το ίδιο το εργαλείο. Αυτό το καθιστά ικανό να δουλέψει σε οποιοδήποτε υπολογιστή ασχέτως δυνατοτήτων, πράμα που το καθιστά πιο συμφέρον από οποιαδήποτε άλλη λύση από αυτές που εξετάσαμε παραπάνω. Για τον λόγο αυτό επιλέξαμε να δουλέψουμε στην εργασία μας με αυτό το εργαλείο για να δείξουμε ότι η δημιουργία ενός 3D αντικειμένου είναι τόσο προσιτή όσο ένας καφές. Για την χρήση του προγράμματος το site μας ζητάει μια συμβολική συνδρομή της τάξεως των 2,5€ κατά την εγγραφή μας. Ύστερα η χρήση είναι δωρεάν.

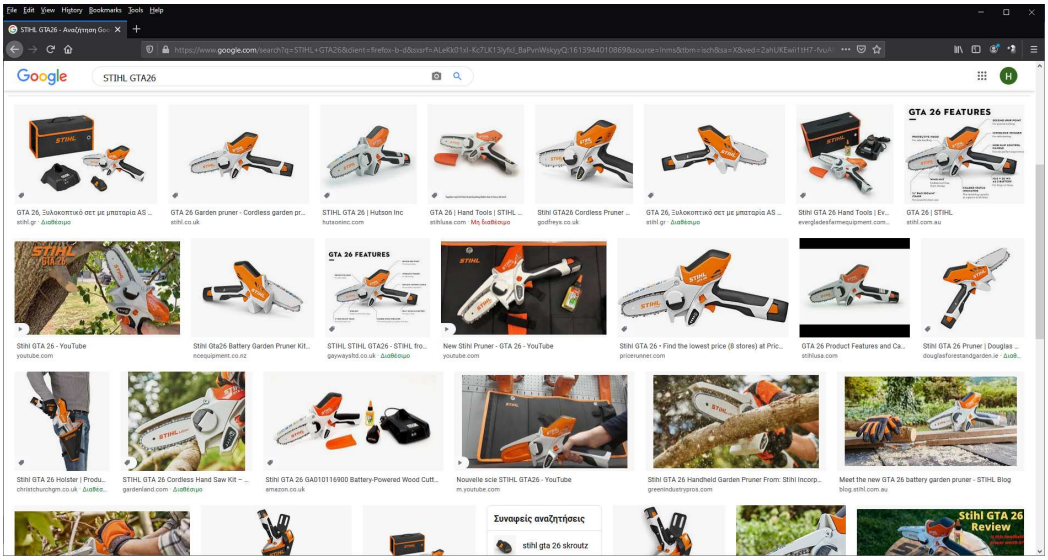


Το πρόγραμμα, καθώς και όλες τις απαραίτητες πληροφορίες μπορεί καείς να τα βρει στην παρακάτω διεύθυνση:

https://www.smoothie-3d.com/site/page_index.php

Η δημιουργία του 3D μοντέλου.

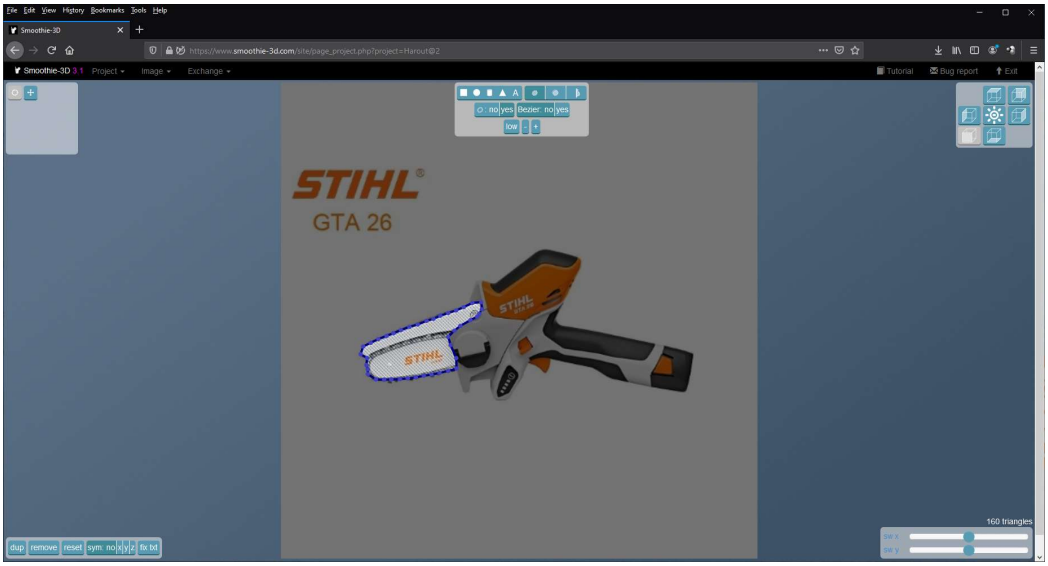
Για τις ανάγκες της εργασίας μας θα δημιουργήσουμε το τρισδιάστατο μοντέλο ενός μηχανήματος της εταιρία STIHL που λέγεται GTA26. Πρόκειται για ένα αλυσοπρίονο χειρός που έχει παρουσιάσει η εταιρία τον τελευταίο χρόνο. Το μοντέλο μας θα το δημιουργήσουμε από φωτογραφίες του μηχανήματος που μπορούμε να βρούμε κάνοντας μια απλή αναζήτηση του GTA26 στο διαδίκτυο ή στο site της κατασκευάστριας εταιρίας.



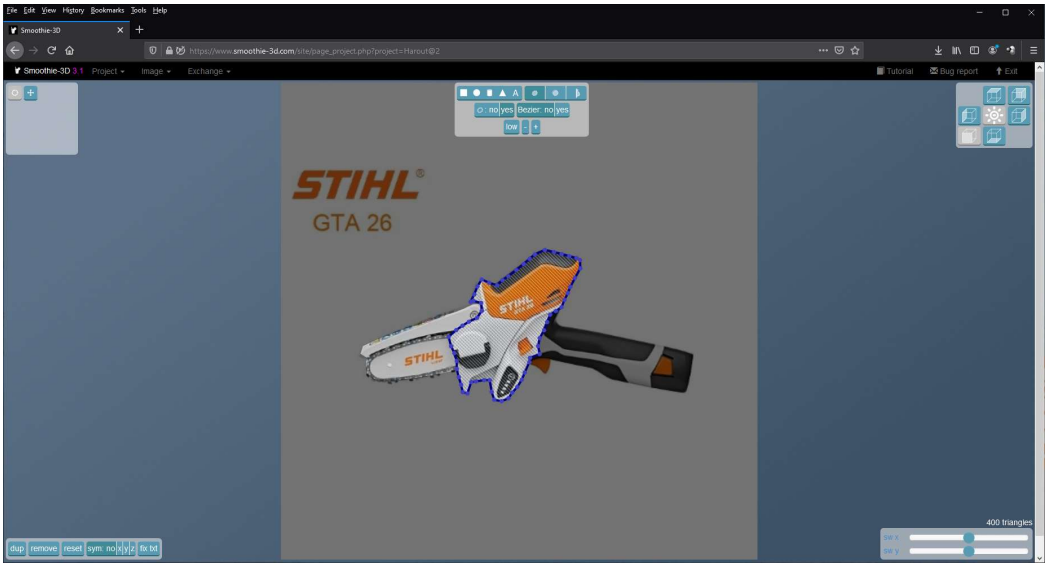
Μετά την εγγραφή μας στο site του Smoothie-3D θα ανοίξουμε ένα νέο project και θα ανεβάσουμε τις φωτογραφίες με τις οποίες θέλουμε να δουλέψουμε. Στην περίπτωση μας ο βασικός «καμβάς» μας θα είναι η παρακάτω αντιπροσωπευτική εικόνα του μοντέλου.

Μόλις ανέβει η φωτογραφία μας στην πλατφόρμα, το περιβάλλον μας δίνει την δυνατότητα να «ζωγραφίσουμε» νοητά πάνω στον καμβά μας. Το αντικείμενο που πάμε να σχεδιάσουμε έχει ουσιαστικά 4 όγκους οι οποίοι θα πρέπει να είναι διακριτοί. Το μοντέλο μας δεν θα χρειαστεί ιδιαίτερη λεπτομέρεια μιας και θα το χρησιμοποιήσουμε σε μικρή κλίμακα στην εφαρμογή μας. Παρόλα αυτά το Smoothie-3D μας δίνει την δυνατότητα να δημιουργήσομε ένα αρκετά λεπτομερές μοντέλο. Όσο περισσότερες όψεις του μοντέλου φορτώσουμε στο σχέδιο και όσο καλύτερα δουλέψουμε τον κάθε όγκο ξεχωριστά, τόσο ανώτερο και πιστότερο θα είναι το αποτέλεσμα του τρισδιάστατου μοντέλου που θα αποκομίσουμε.

Έχοντας την φωτογραφία μας σαν οδηγό σχεδιάζουμε τα κύρια μέρη του μοντέλου μας. Ο σχεδιασμός γίνεται αρχικά με την δημιουργία περιγραμμάτων γύρω από τα μέρη αυτά. Κάθε μέρος του μοντέλο που πρέπει να διακρίνεται από διαφορετικό γεωμετρικό όγκο στο τελικό μοντέλο πρέπει να σχεδιαστεί και με ξεχωριστό δικό του περίγραμμα. Έτσι λοιπόν ξεκινάμε σχεδιάζοντας το περίγραμμα την «Λάμας-Αλυσίδας-Προφυλακτήρα» όπως φαίνεται παρακάτω στην φωτογραφία.



Στην συνέχεια δημιουργούμε το περίγραμμα του όγκου της «Μηχανής» όπως φαίνεται παρακάτω στην φωτογραφία.

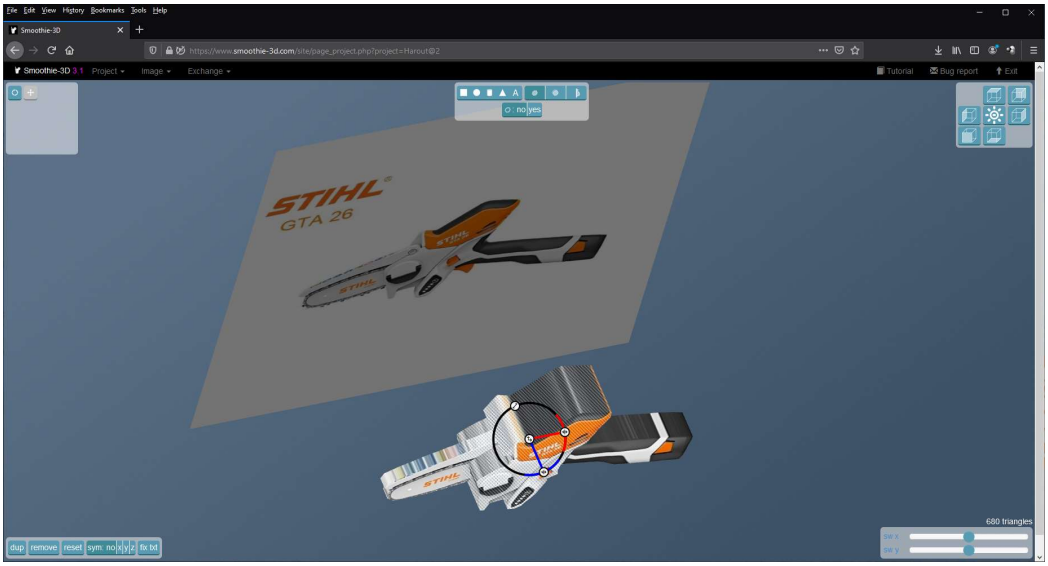


Τρίτο μέρος που θα πρέπει να σχεδιαστεί ξεχωριστά είναι το κομμάτι της «Λαβής» όπως φαίνεται παρακάτω στην φωτογραφία.

Τέταρτο και τελευταίο μέρος του μοντέλου που πρέπει να σχεδιαστεί ξεχωριστά είναι η «Σκανδάλη» όπως φαίνεται παρακάτω στην φωτογραφία.

Τώρα περνώντας πάλι από κάθε ένα από τα περιγράμματα που ήδη φτιάξαμε καθορίζουμε τα ιδιαίτερα μορφολογικά τους χαρακτηριστικά.

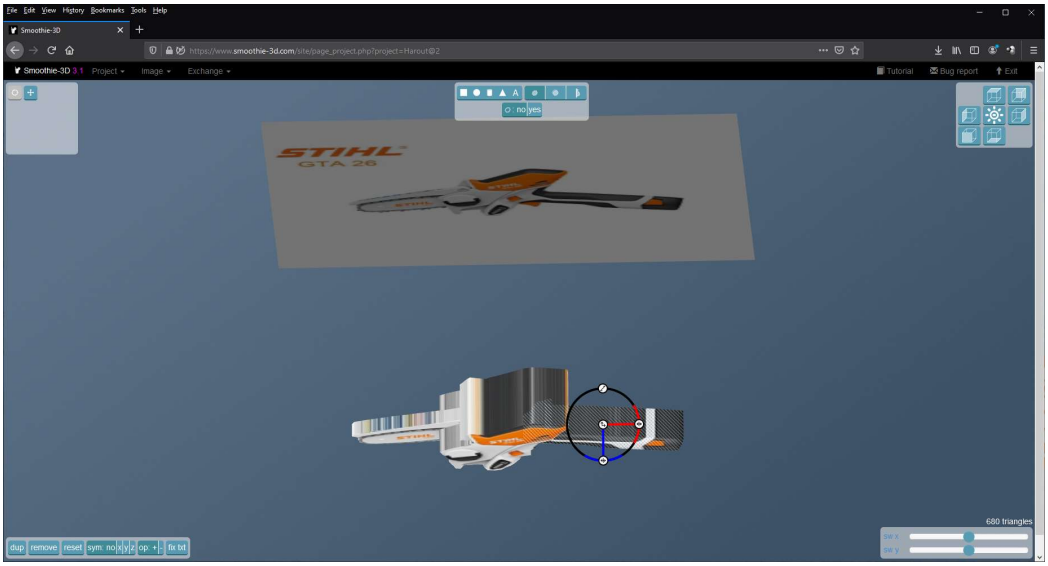
Το μέρος της «Λάμας-Αλυσίδας-Προφυλακτήρα» σχεδιάζεται αισθητά πιο λεπτό από υπόλοιπο σώμα του μοντέλου όπως είναι και στην πραγματικότητα.



Το μέρος της «Μηχανής» αποτελεί τον κύριο όγκο του μοντέλου όπως φαίνεται και από τις φωτογραφίες του φυσικού προϊόντος που παραθέτουμε.

Το κομμάτι της «Λαβής» πέρα του γεγονότος ότι βρίσκεται σε ένα ενδιάμεσο πάχος από αυτό της «Μηχανής» και αυτό της «Λάμας-Αλυσίδας-Προφυλακτήρα», πρέπει να πάρει κι έναν οβάλ σχήμα έτσι ώστε να πλησιάζει σε πιστότητα το πραγματικό αντικείμενο.

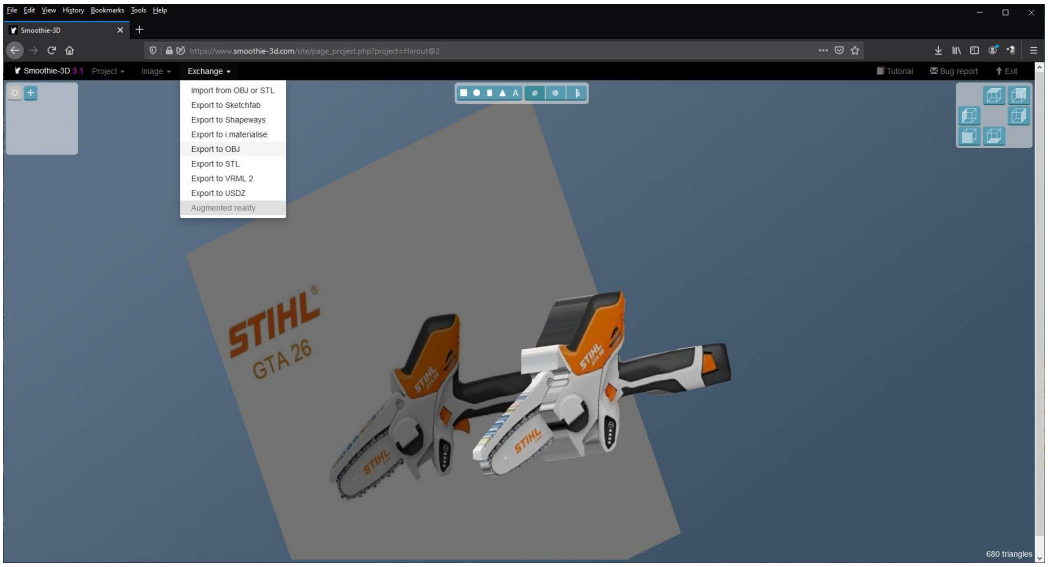

STIHL
GTA 26
680 triangles


STIHL
GTA 26
680 triangles

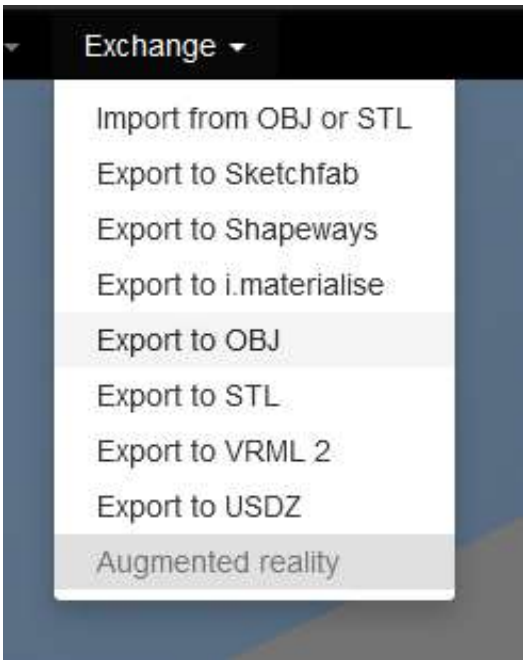
Τέλος σχεδιάζουμε και τον μικρό, αλλά σημαντικό για την λεπτομέρεια, όγκο του μέρους της «Σκανδάλης» και το προσαρμόζουμε στο μοντέλο μας.

Τώρα το μοντέλο μας έχει πλησιάσει σε πολύ ικανοποιητικό βαθμό το φυσικό προϊόν που έχουμε στα χέρια μας.

Αφού έχουμε τελειώσει με τον σχεδιασμό είμαστε έτοιμοι για την εξαγωγή του τελικού μας αντικειμένου.

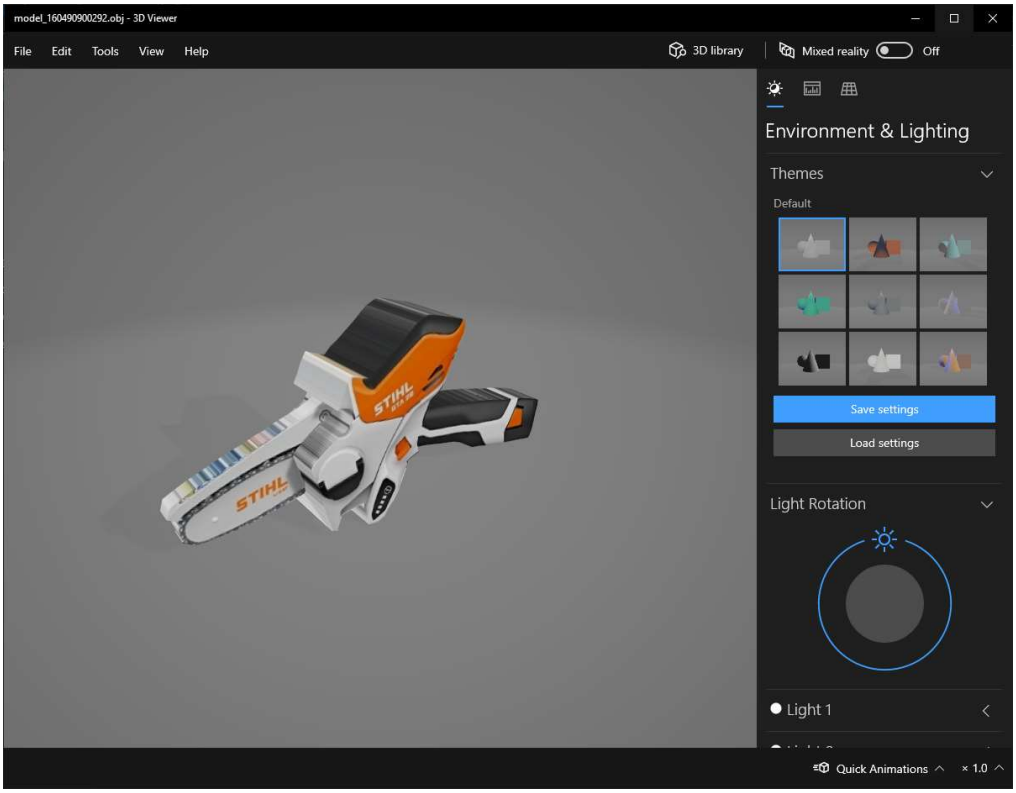


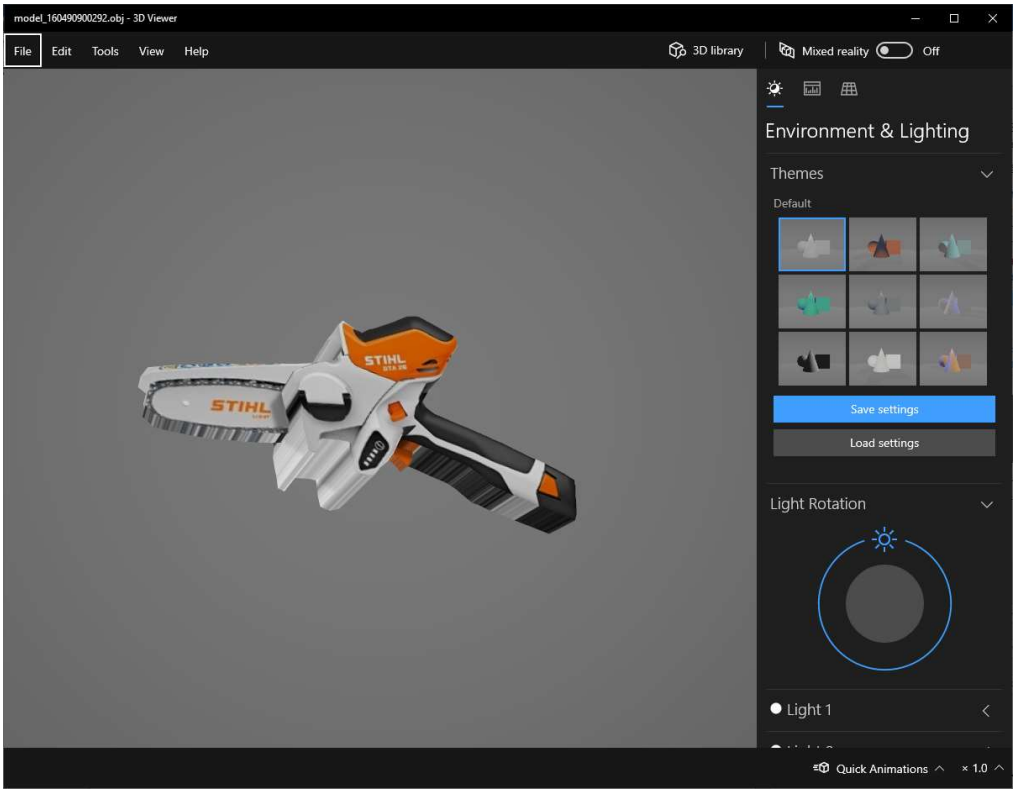
Το πρόγραμμα μας δίνει την δυνατότητα να εξάγουμε το σχέδιο μας σε διάφορα είδη αρχείου όπως φαίνεται παρακάτω.



Εμείς θα κάνουμε εξαγωγή σε OBJ αρχείο για να το χρησιμοποιήσουμε στην πλατφόρμα Unity.

Το τελικό αποτέλεσμα της εξαγωγής φαίνεται παρακάτω στις φωτογραφίες και είναι ιδιαιτέρως πιστό για την κλίμακα μεγέθους που θέλουμε να το χρησιμοποιήσουμε.

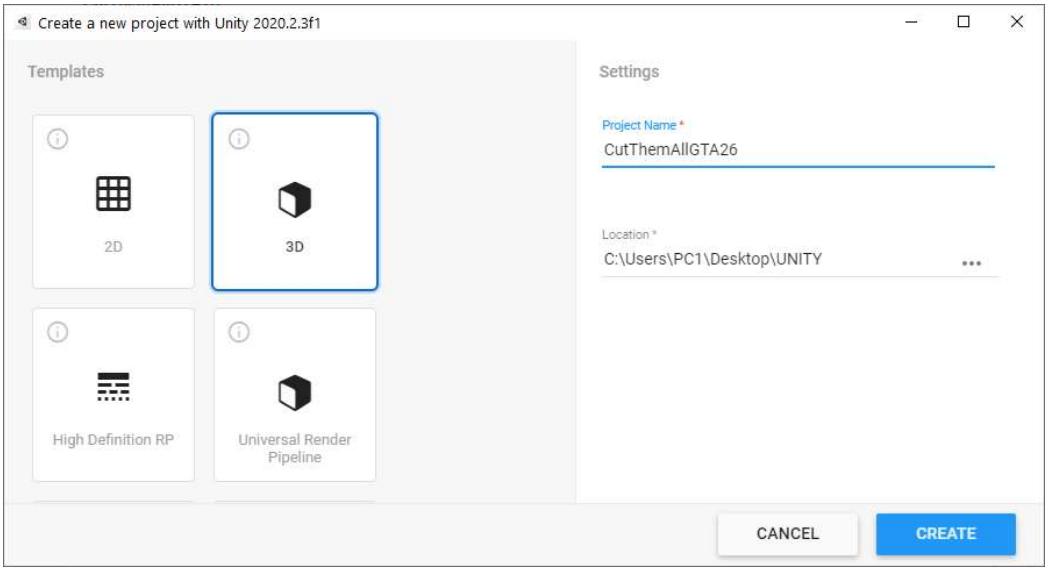




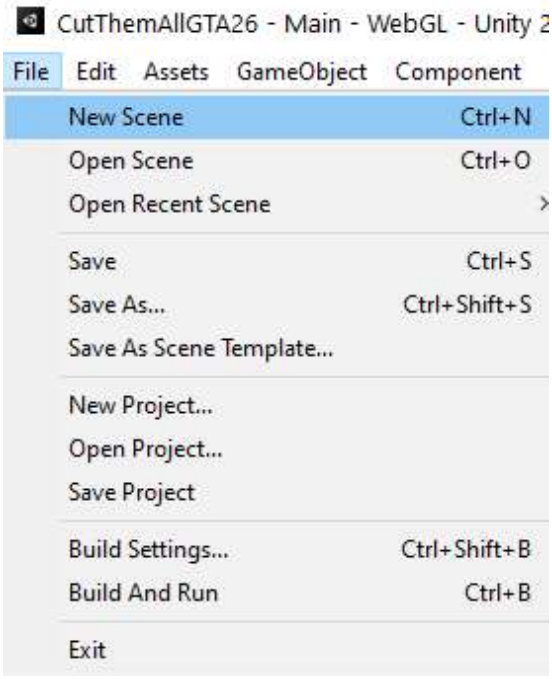
Η αρχική παραμετροποίηση στην Unity.

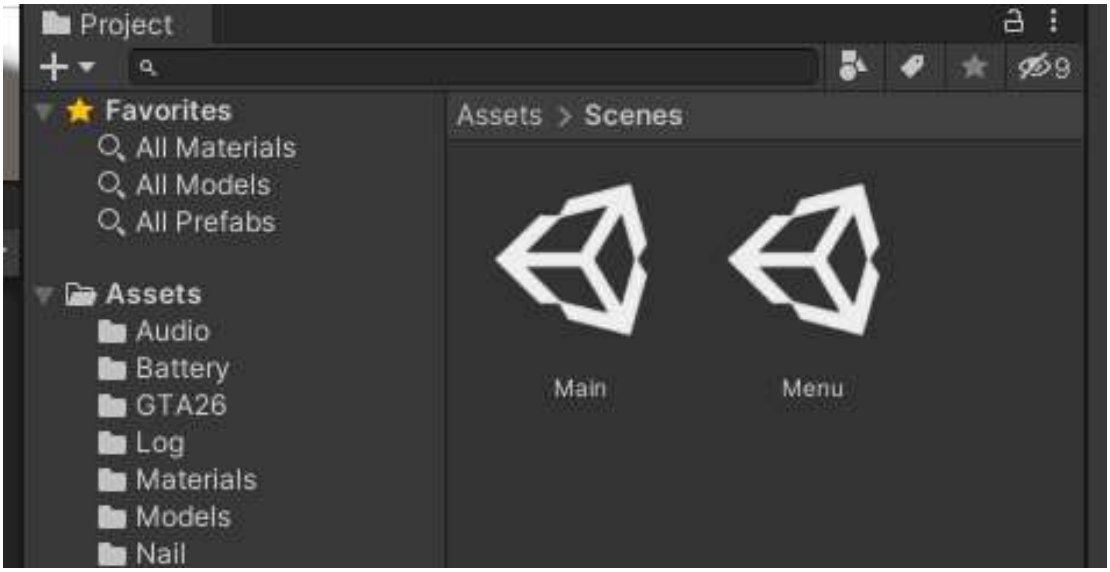
Από εδώ και πέρα θα περιγράψουμε βήμα προς βήμα όλη την διαδικασία με την οποία θα δημιουργήσουμε ένα ολοκληρωμένο παιχνίδι με την χρήση της πλατφόρμας unity. Στο παιχνίδι αυτό πρωταγωνιστικό ρόλο θα παίξει το αλυσοπρίονο το οποίο σχεδιάσαμε με την βοήθεια του Smoothie-3D.

Αρχικά ανοίγουμε το Unity Hub και δημιουργούμε ένα νέο project ορίζοντας το template το όνομα του project και την τοποθεσία όπου θα αποθηκευτεί όπως φαίνεται παρακάτω στην εικόνα:



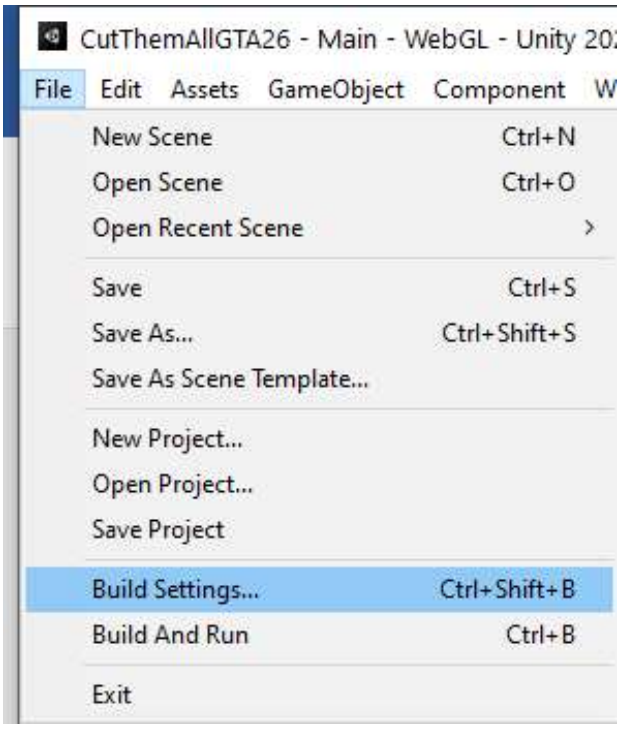
Στην συνέχεια πάμε και δημιουργούμε 2 κενές σκηνές στο νέο μας project.



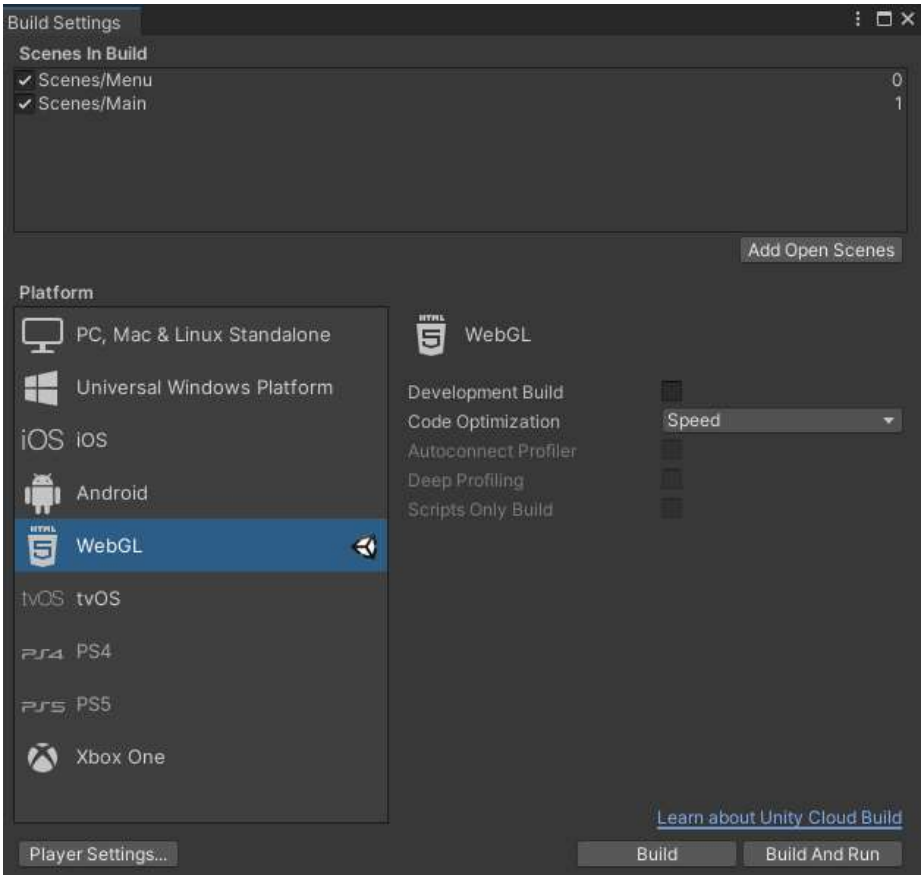


Οι δύο αυτές σκηνές είναι η Main που αποτελέι την κύρια σκηνή που φιλοξενεί το παιχνίδι μας και η Menu που είναι η εισογωγική σκηνή που φιλοξενεί το μενού έναρξης του παιχνιδιού μας καθώς και κάποιες επιλογές πληεοφοριών σχετικά με το τρόπο παιχνιδιού.

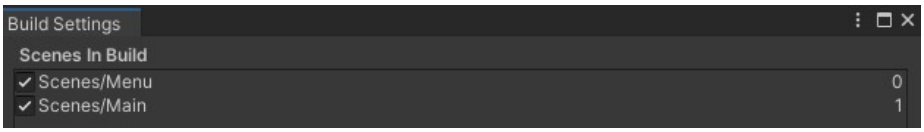
Αφού έχουμε τις σκηνές μας, πάμε και καθορίζουμε τις ρυθμίσεις της δομής που θα έχει το Project μας (Build Settings).



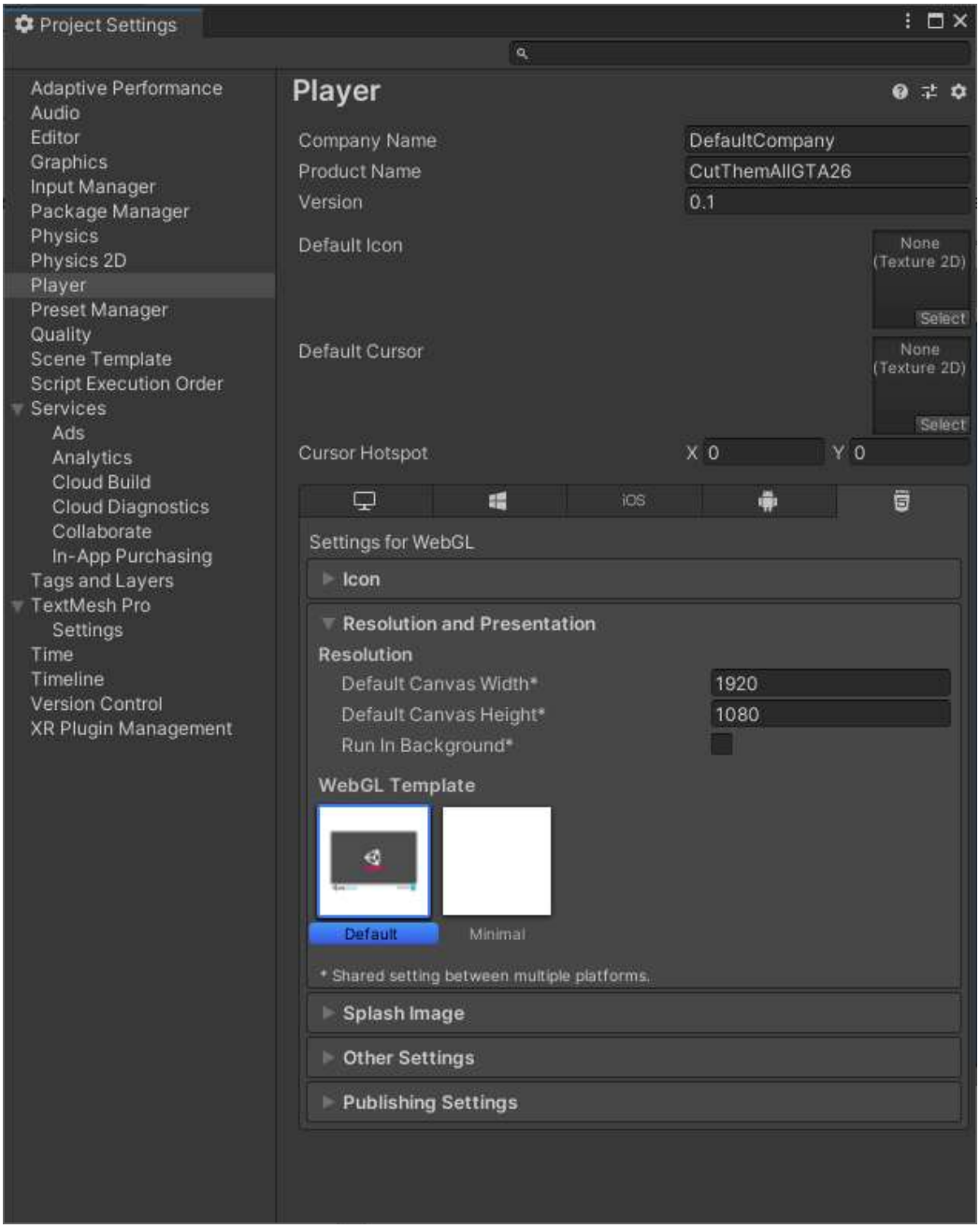
Εδώ ορίζουμε ότι η εφαρμογή που θα φτιάξουμε είναι web based, οπότε επιλέγουμε το WebGL.



Επίσης μιας και έχουμε φτιάξει ήδη τις σκηνές μας, ορίζουμε την σειρά τους. Όπως φαίνεται και στην εικόνα, πρώτα θα εμφανίζεται η σκηνή του μενού και στην συνέχεια με τα κατάλληλα κουμπιά θα περνάμε στην κύρια σκηνή.



Για το συγκεκριμένο project θα χρησιμοποιήσουμε ρυθμίσεις για Web και θα ορίσουμε την ανάλυση όπως φαίνεται παρακάτω στο Player Settings.

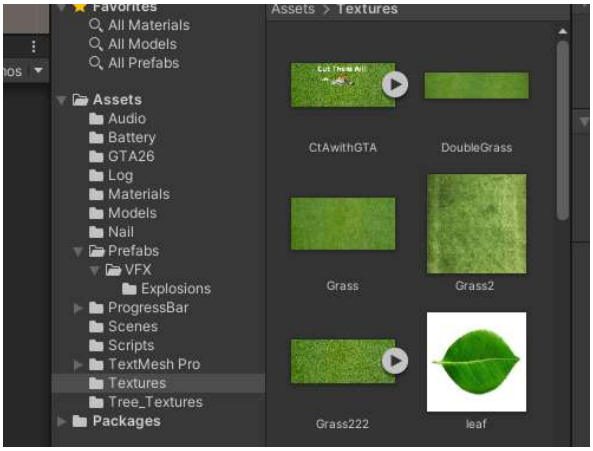


Όπως φαίνεται παραπάνω το παιχνίδι μας θα παίζεται σε ένα «κάδρο» 1920*1080. Το σκεπτικό μας είναι να μπορεί να το βλέπει ο παίχτης σε πλήρη οθόνη σε μια σύγχρονη οθόνη με ανάλυση 1920*1080.

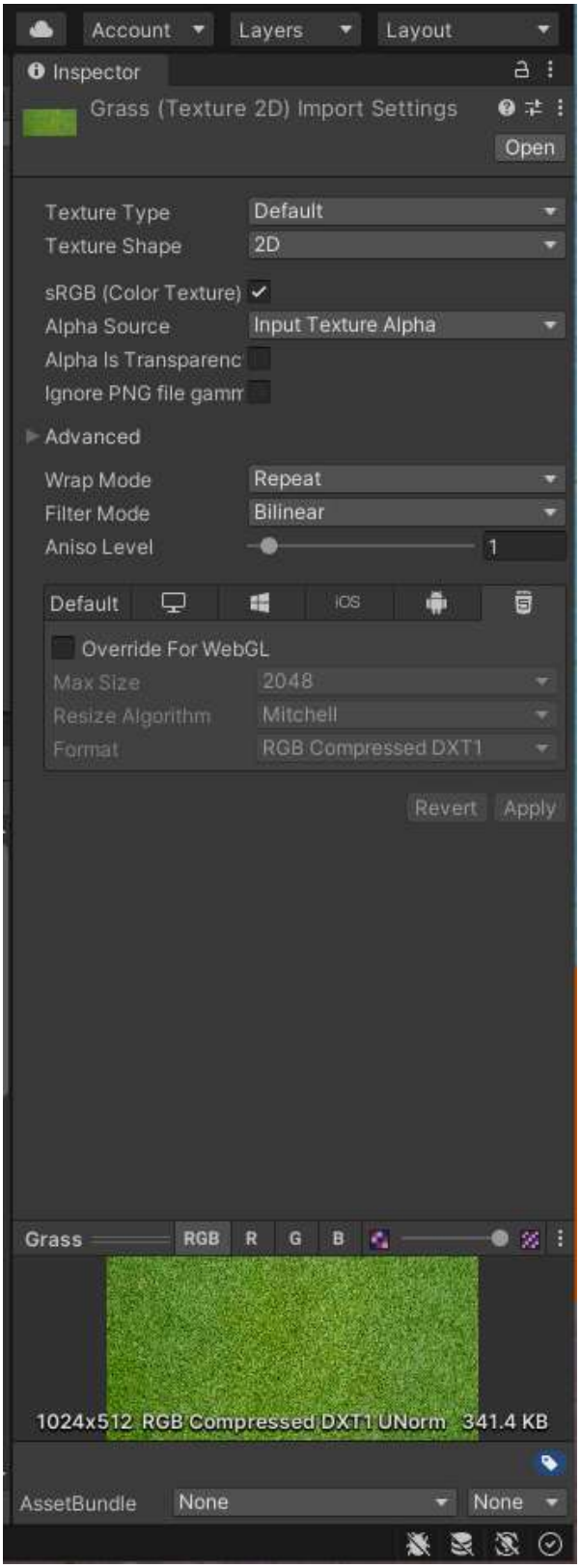
Το Παρασκήνιο.

Ξεκινάμε να χτίσουμε το παιχνίδι μας από πίσω προς τα εμπρός, από το βάθος προς την επιφάνεια. Το παιχνίδια μας σχετίζεται με ένα εργαλείο κήπου, άρα το παρασκήνιο μας πρέπει να είναι κάτι σχετικό με κήπο. Τι πιο σχετικό λοιπόν με κήπο από μια επιφάνεια με γκαζόν. Για τον λόγο αυτό επιστρατεύσαμε την κάμερα του κινητού μας για να τραβήξουμε μερικές φωτογραφίες από γκαζόν όπως οι παρακάτω.

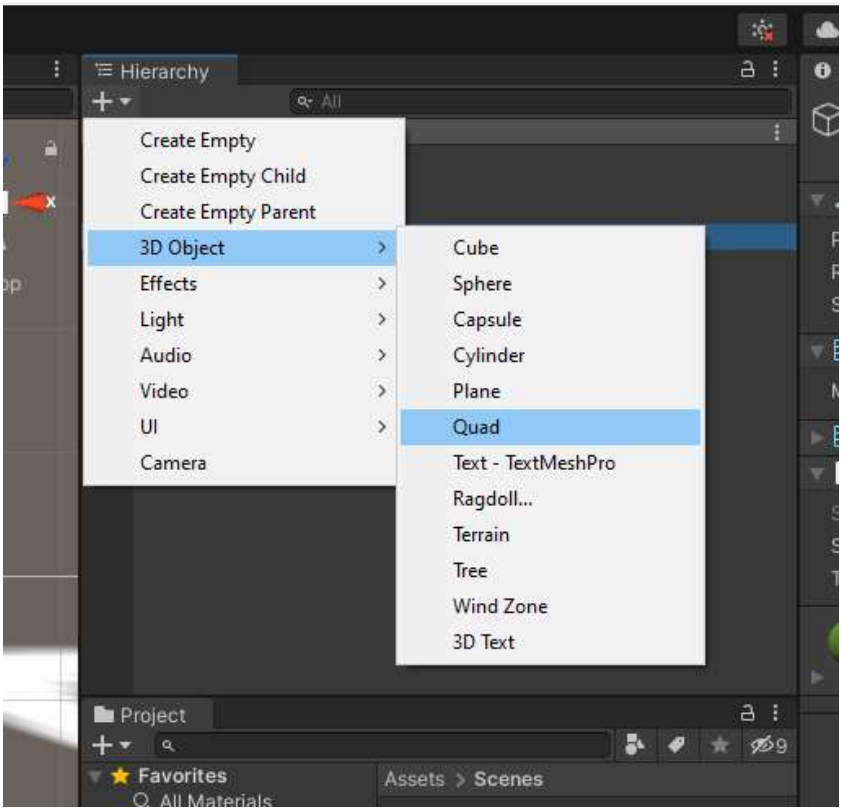
Μετά από δοκιμές επιλέξαμε την δεύτερη εικόνα ως πιο κατάλληλη για της ανάγκες του παιχνιδιού μας. Για να εισάγουμε την εικόνα αυτή στην unity ως παρασκήνιο ακολουθούμε τα παρακάτω βήματα. Εισάγουμε αρχικά με drag and drop το αντικείμενο (.jpg) της φωτογραφίας του γκαζόν στο φάκελο Textures της unity.



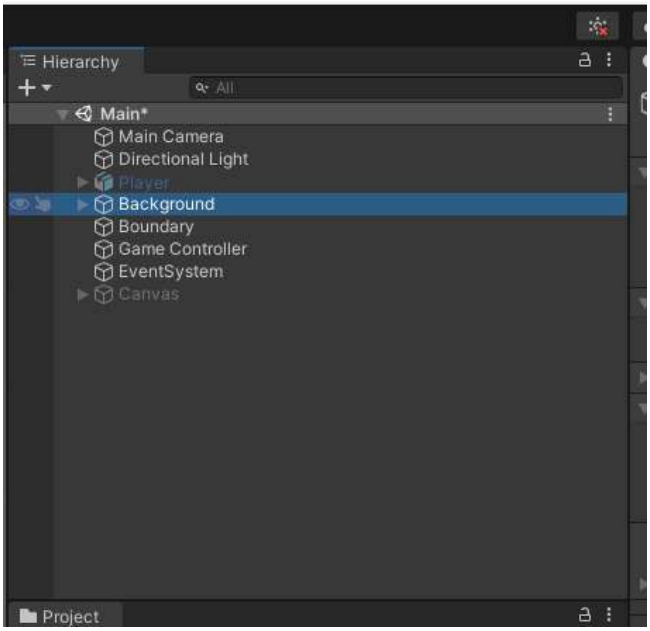
Κατά την εισαγωγή η unity δημιουργεί τις ανάλογες καταχωρήσεις στους υπόλοιπους φακέλους (π.χ. Materials) για να υποστηρίξει το αντικείμενο αυτό.


Account
Layers
Layout
Inspector
Grass (Texture 2D) Import Settings
Open
Texture Type
Default
Texture Shape
2D
sRGB (Color Texture)
Alpha Source
Input Texture Alpha
Alpha Is Transparenc
Ignore PNG file gamm
Advanced
Wrap Mode
Repeat
Filter Mode
Bilinear
Aniso Level
1
Default
iOS
Override For WebGL
Max Size
2048
Resize Algorithm
Mitchell
Format
RGB Compressed DXT1
Revert
Apply
Grass
RGB
R
G
B
1024x512 RGB Compressed DXT1 UNorm 341.4 KB
AssetBundle
None
None

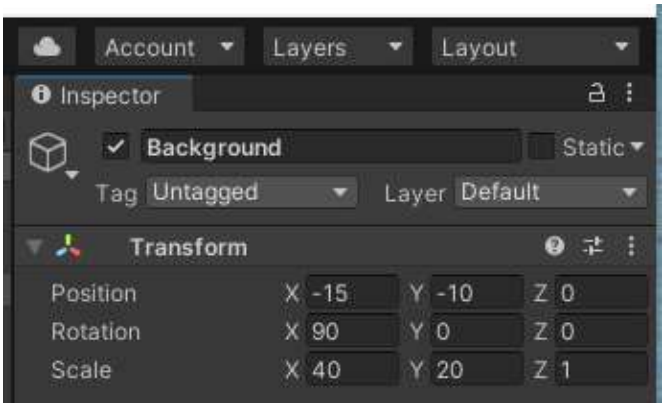
Ύστερα εισάγουμε αρχικά ένα στοιχείο Quad.



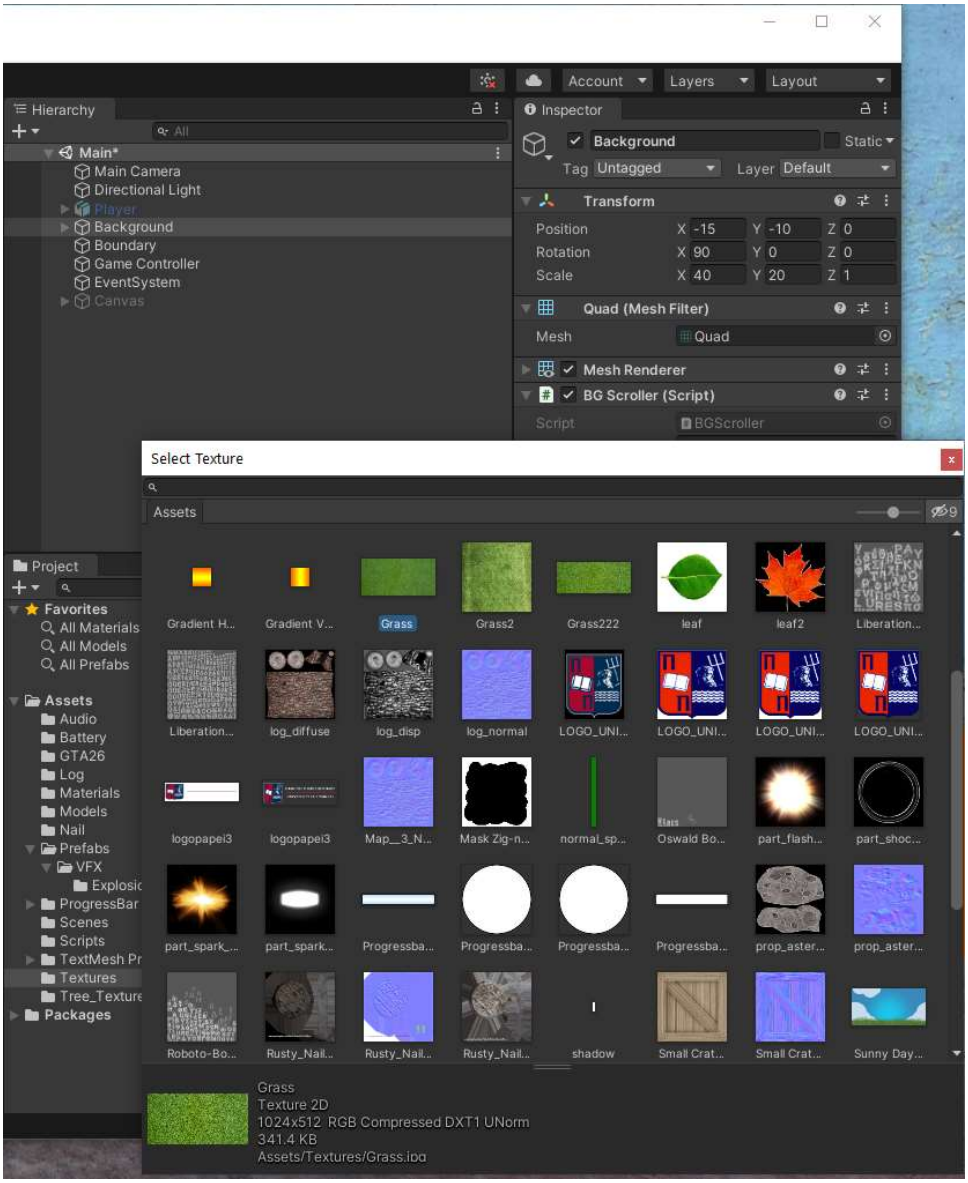
Και το μετονομάζουμε σε Background



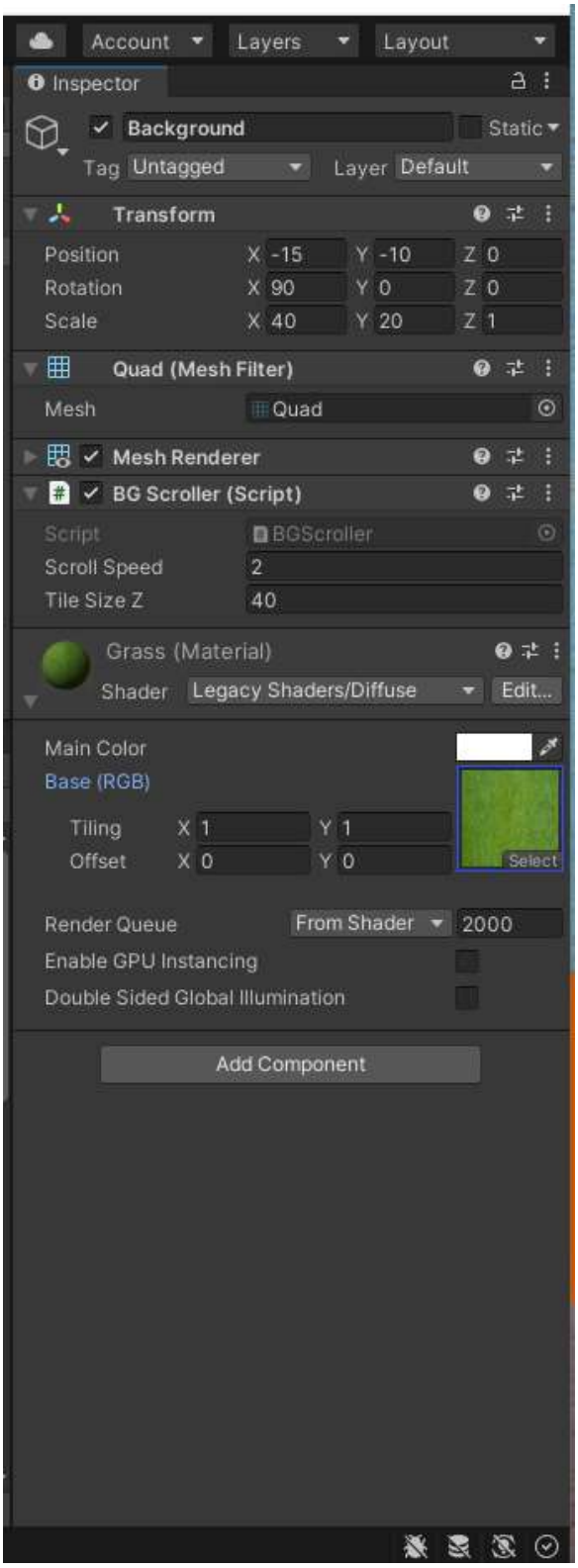
Δίνουμε τα παρακάτω στοιχεία μεγέθους και χωροταξικής διάταξης στο αντικείμενο μας.



Βρίσκουμε από τα assets το grass που είχαμε εισαγάγει στο φάκελο textures και το προσαρτούμε στο Background



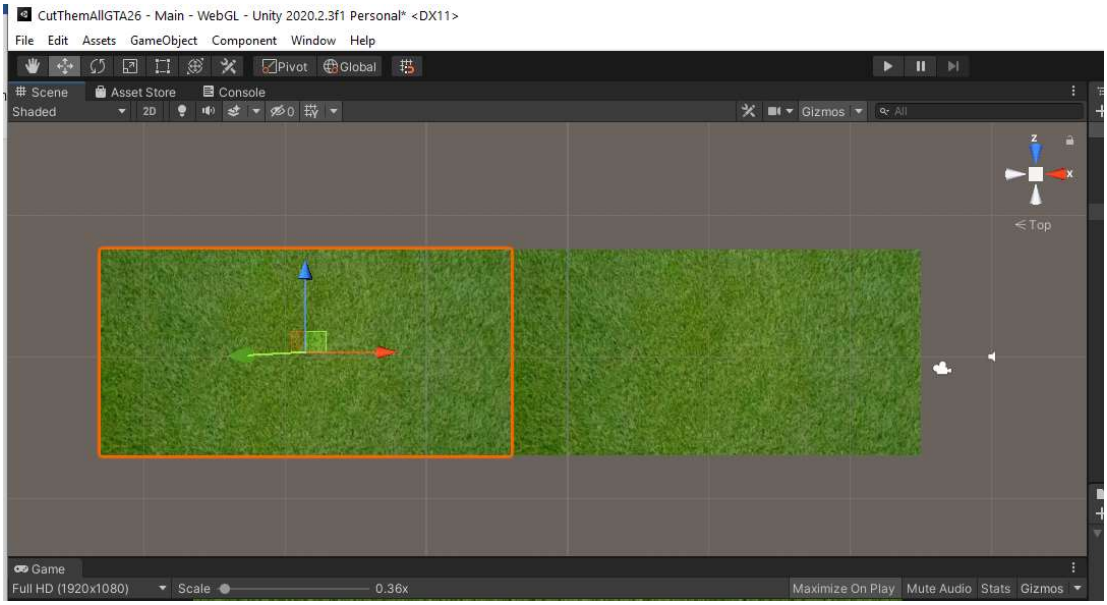
Επίσης επιλέγουμε το shader Diffuse από τα Legacy shaders διότι απεικονίζει καλυτέρα τον φωτισμό της φωτογραφίας του φόντου που θέλουμε για το γκαζόν μας και εφαρμόζουμε στο Quad στοιχείο τις παρακάτω ρυθμίσεις.



Το αποτέλεσμα στο Game φαίνεται ως εξής:

Ένας καθαρός κήπος με γκαζόν για να δουλέψουμε. Σε αυτό τον κήπο εμείς θα δώσουμε και μια κίνηση. Δεν θέλουμε να είναι στατικός. Θέλουμε να «κυλάει» προς την αντίθετη κατεύθυνση από αυτή που θα κινούνται τα υπόλοιπα αντικείμενα του παιχνιδιού έτσι ώστε να δίνεται η αίσθηση ότι κινούμαστε κι εμείς σαν «παίχτης» μέσα σε ένα κήπο, άσχετα με την κίνηση που θα δίνουμε εμείς για να ελέγξουμε το αντικείμενο «παίχτης» .

Για να γίνει αυτό τοποθετούμε ένα δεύτερο κομμάτι γκαζόν ακριβώς ίδιου μεγέθους δίπλα στο αρχικό μας κομμάτι.



Στην συνέχεια προσθέτουμε στο Background ένα κομμάτι κώδικα με το οποίο θα του προσδίδουμε κίνηση και θα το ανανεώνουμε με τρόπο τέτοιο που στον παίχτη του παιχνιδιού θα δίνεται η εντύπωση μιας ομαλής κίνησης του παρασκηνίου προς τα δεξιά.

Δημιουργούμε ένα νέο Script σε γλώσσα C# με το όνομα BGScroller με τον κώδικα που παρατίθεται παρακάτω:

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

public class BGScroller : MonoBehaviour
{
	public float scrollSpeed;
	public float tileSizeZ;

	private Vector3 startPosition;

	void Start()
	{
		startPosition = transform.position;
	}

	void Update()
	{
		float newPosition = Mathf.Repeat(Time.time * scrollSpeed, tileSizeZ);
		transform.position = startPosition + Vector3.right * newPosition;
	}
}
```

Με τον κώδικα αυτό προσδίδουμε κίνηση προς τα δεξιά στο αντικείμενο του παρασκηνίου. Έτσι λοιπόν όσο θα παίζουμε εμείς στο προσκήνιο από πίσω το παρασκήνιο θα δίνει την αίσθηση ότι όλη η σκηνή κινείται προς τα αριστερά.

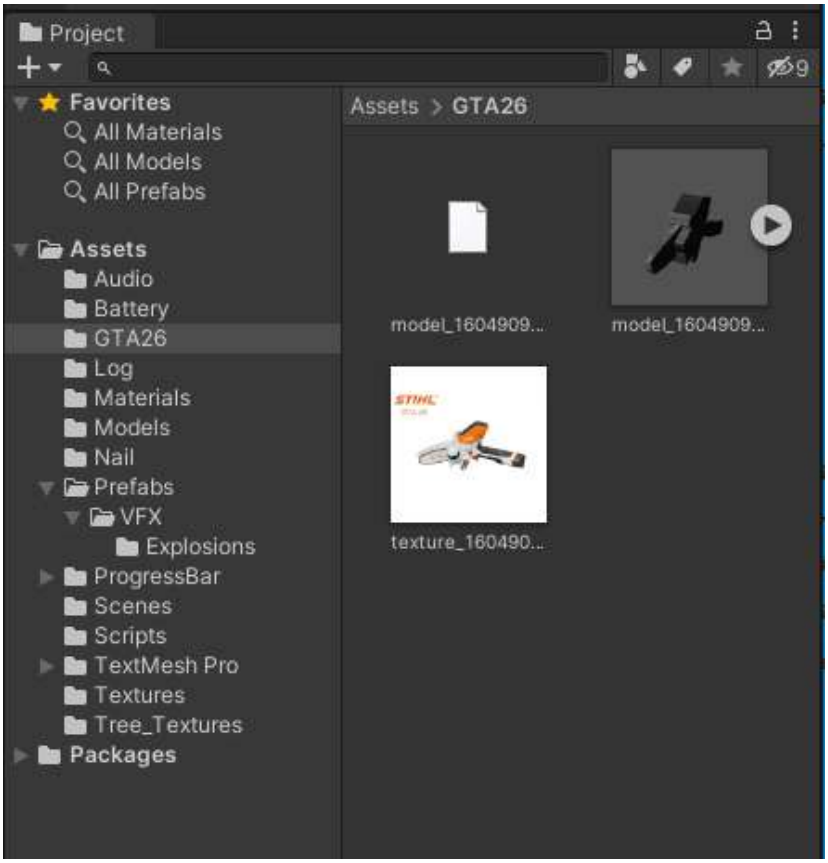
Το κώδικα αυτό τον προσαρτώ στο αντικείμενο Background. Με τον κώδικα αυτό προστίθενται 2 μεταβλητές στο αντικείμενο αυτό όπως φαίνεται παρακάτω.

Η μία μεταβλητή καθορίζει την ταχύτητα της κίνησης – κύλισης που θα κάνει το παρασκήνιο. Η δεύτερη καθορίζει το μέγεθος του πλαισίου μετά το οποίο θα ανανεώνεται η θέση της εικόνας έτσι ώστε να μην δημιουργούνται κενά κατά την κίνηση και να φαίνεται αυτή ομαλή.

Το αντικείμενο «Παίχτης».

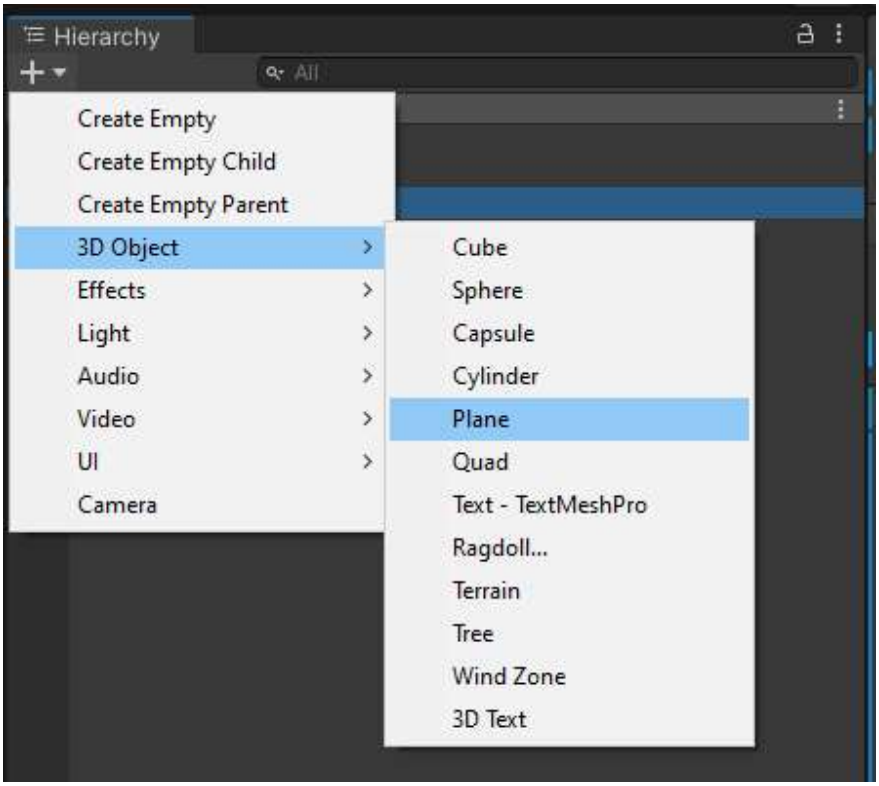
Πάμε τώρα να δημιουργήσουμε τον πρωταγωνιστή του παιχνιδιού μας που είναι ο κύριος παίχτης. Δεν πρωταγωνιστεί μόνο λόγο του ότι είναι ο παίχτης που χειριζόμαστε κατά το παιχνίδι αλλά κι επειδή το αντικείμενο του είναι το τρισδιάστατο αντικείμενο που δημιουργήσαμε εμείς από το μηδέν με την μέθοδο φωτογραμετρίας.

Αρχικά δημιουργούμε ένα φάκελο στα Assets μας με το όνομα GTA26. Εκεί εισάγουμε με drag and drop το αντικείμενο .obj του μοντέλου καθώς και τα παρελκόμενα αρχεία .mtl του μοντέλου καθώς και το texture αρχείο (.jpg) που χρειάζονται για να απεικονισθεί σωστά το μοντέλο.

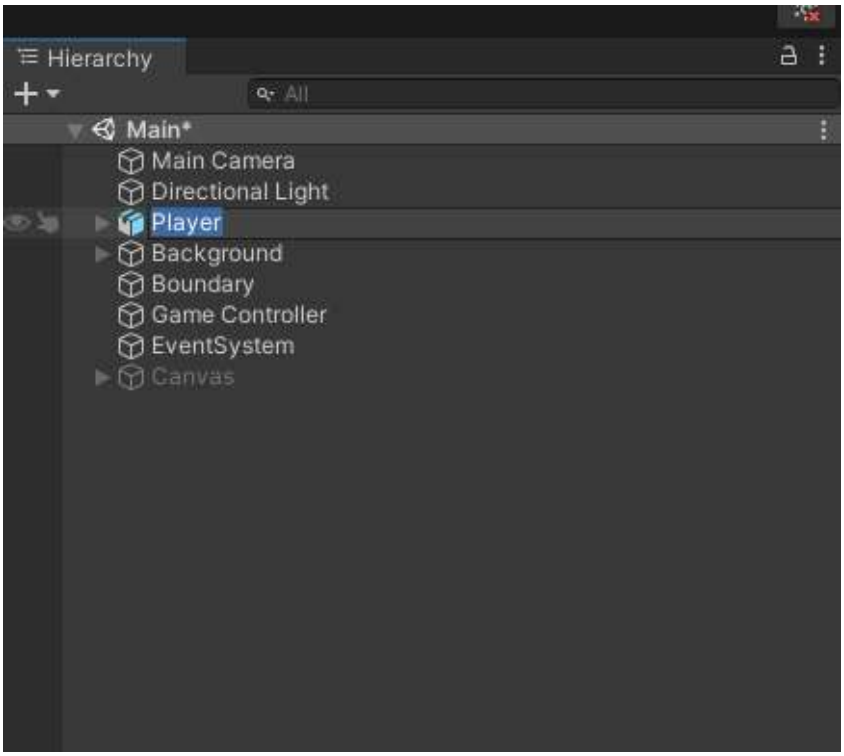


Κατά την εισαγωγή η unity δημιουργεί τις ανάλογες καταχωρήσεις στους υπόλοιπους φακέλους (π.χ. Materials) για να υποστηρίξει το αντικείμενο αυτό.

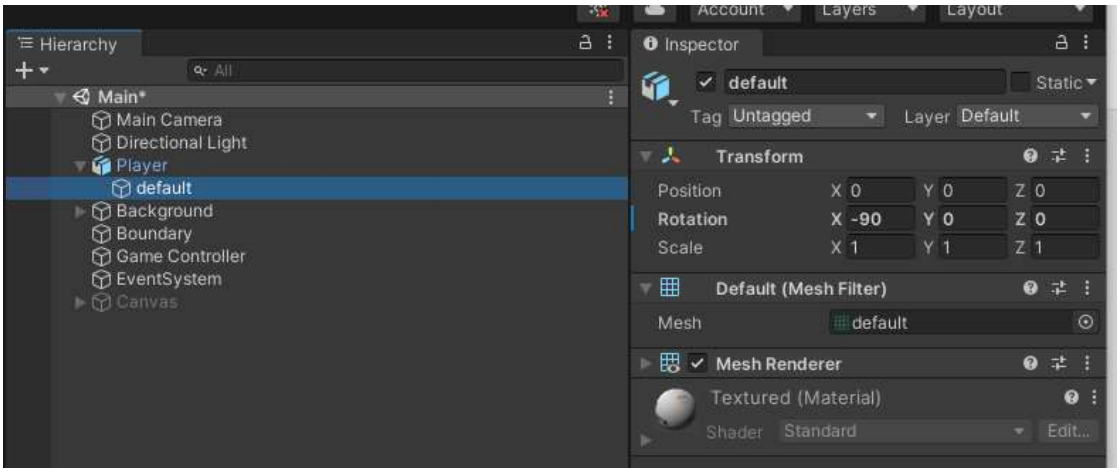
Ύστερα δημιουργούμε ένα κενό 3D αντικείμενο στην σκηνή μας.

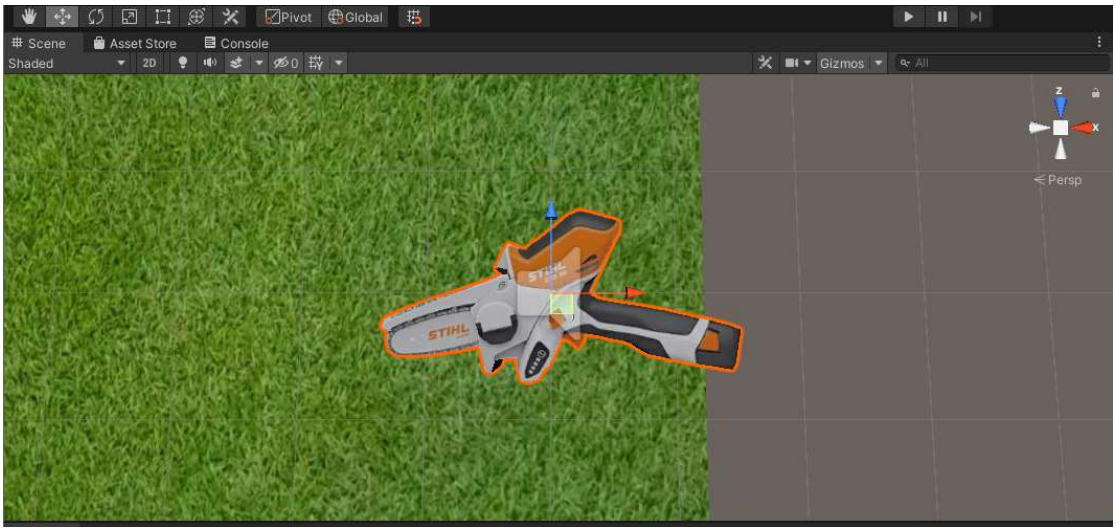


Και το ονομάζουμε Player.



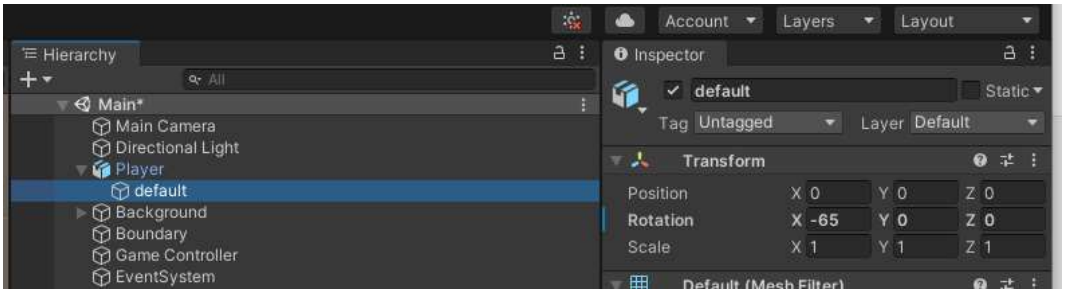
Στο αντικείμενο αυτό προσαρτώ το τρισδιάστατο αντικείμενο του GTA26 σαν παιδί του. Και του του καθορίζουμε την θέση στη σκηνή, την περιστροφή και την κλίμακα.



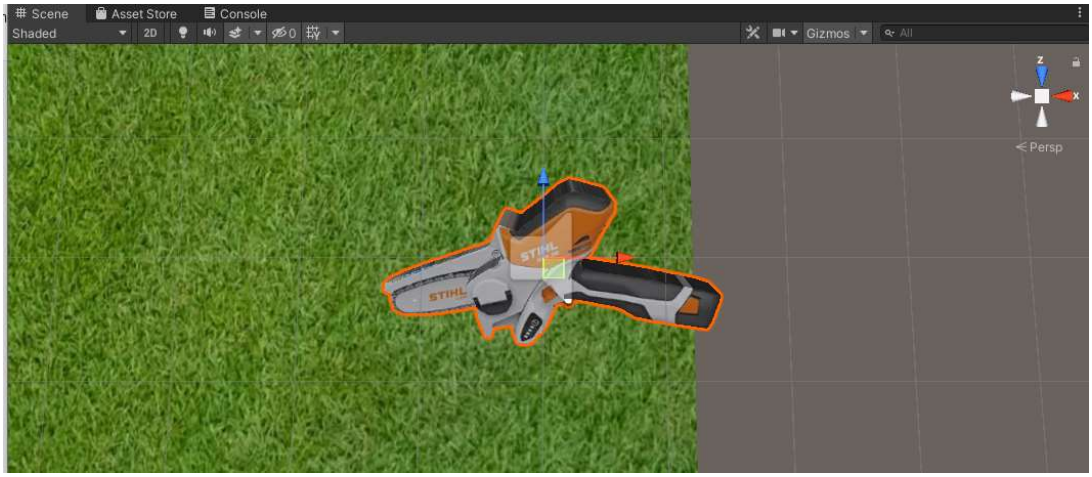


Παρατηρούμε όμως ότι με αυτήν την ρύθμιση βλέπουμε μόνο την μια πλαυρά του μοντέλου μας και χάνουμε την αίσθηση του τρισδιάστατου μοντέλου. Πράγμα που ακυρώνει ουσιαστικά την προσπάθεια μας για την δημιουργία του 3D μοντέλου. Γιατί το φτιάξαμε αν είναι να το χρησιμοποιήσουμε σαν δισδιάστατο;

Έτσι λοιπόν θα αλλάξουμε ελαφρώς την κλίση του μοντέλου μας για να του δώσουμε βάθος και να αναδείξουμε το γεγονός ότι είναι τρισδιάστατο. Αυτό επιτυγχάνεται με το να αλλάξουμε λίγο το transform όπως φαίνεται παρακάτω.

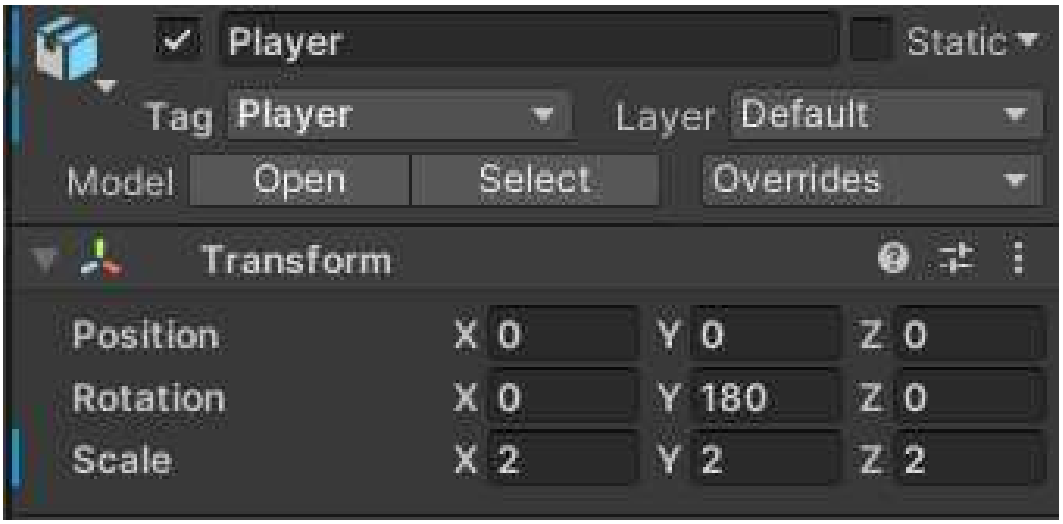


Και το πριόνι μας παίρνει μια ελαφριά κλίση που προσομοιώνει το τρόπο που θα το κράταγε ένα ανθρώπινο χέρι κατά την χρήση.



Μπορεί τώρα να φαίνεται ότι η διαφορά που κάνουμε είναι πολύ μικρή αλλά στην συνέχεια θα τονίσουμε ακόμη περισσότερο την τρισδιάστατη φύση του μοντέλου μας όταν θα του δώσουμε κίνηση. Πάμε να παραμετροποιήσουμε το αντικείμενο Player.

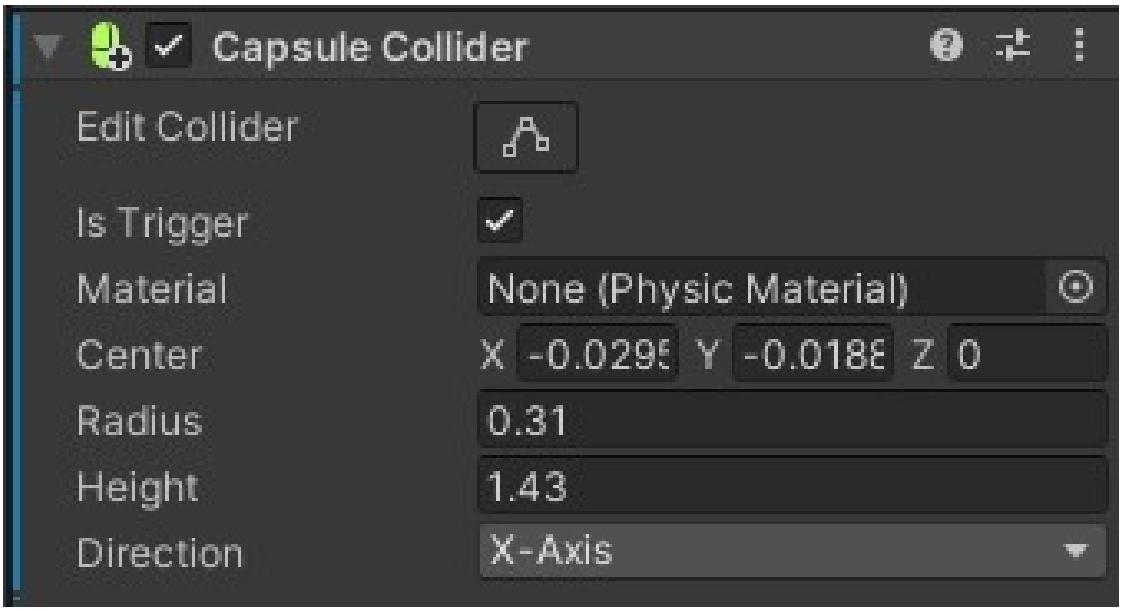
Αρχικά καθορίζουμε την θέση στη σκηνή, την περιστροφή και την κλίμακα.



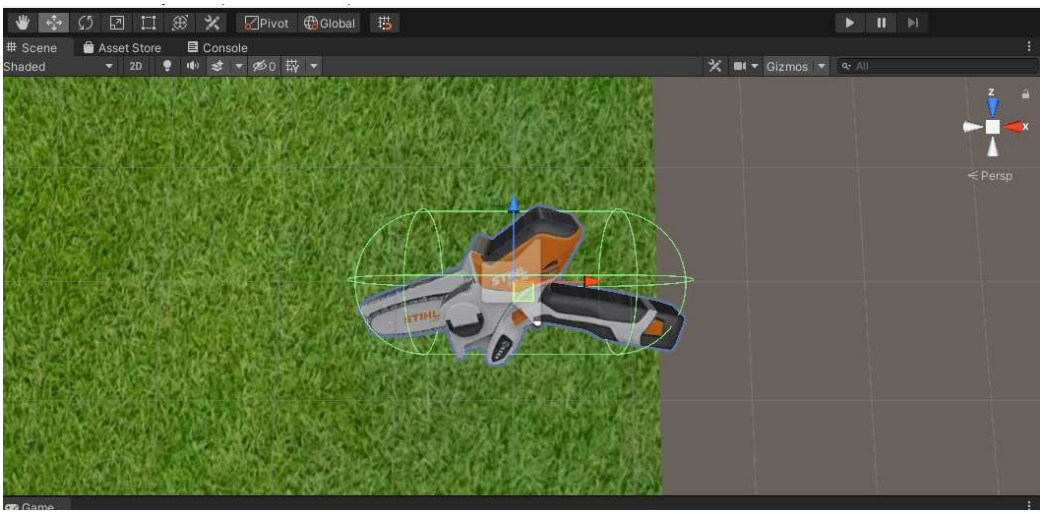
Ύστερα θα δώσουμε στο μοντέλο μας φυσική υπόσταση όπως φαίνεται παρακάτω.

Εδώ έχουμε δώσει μάζα στο αντικείμενο μας. Του έχουμε αφαιρέσει την βαρύτητα για να μην πέφτει από την σκηνή μας και έχουμε κλειδώσει την περιστροφή του στους άξονες Y και Z.

Στην συνέχεια θα προσδώσουμε στο μοντέλο μας ένα περίγραμμα κάψουλας όπως φαίνεται παρακάτω.



Με αυτόν τον τρόπο ουσιαστικά δηλώνουμε στην unity τα όρια του μοντέλου μας. Πράγμα που θα μας φανεί απαραίτητο όταν ο παίχτης θα αλληλοεπιδρά με τα υπόλοιπα στοιχεία του παιγνιδιού που θα προσθέσουμε.

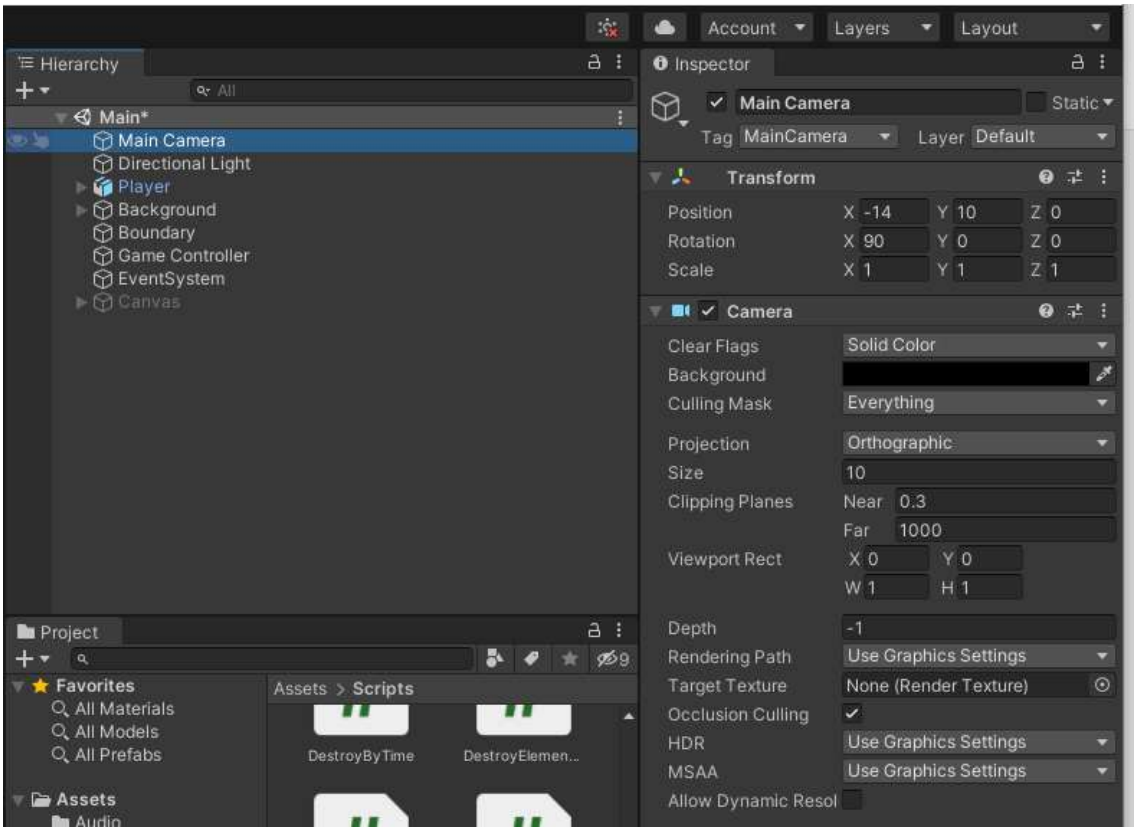


Με την κάψουλα λοιπόν αυτή να περικλείει το μοντέλο μας, οριοθετούμε τον χώρο του μοντέλου. Αν κάτι εισέλθει στον χώρο αυτό θεωρείται «σύγκρουση» με το αντικείμενο του παίχτη.

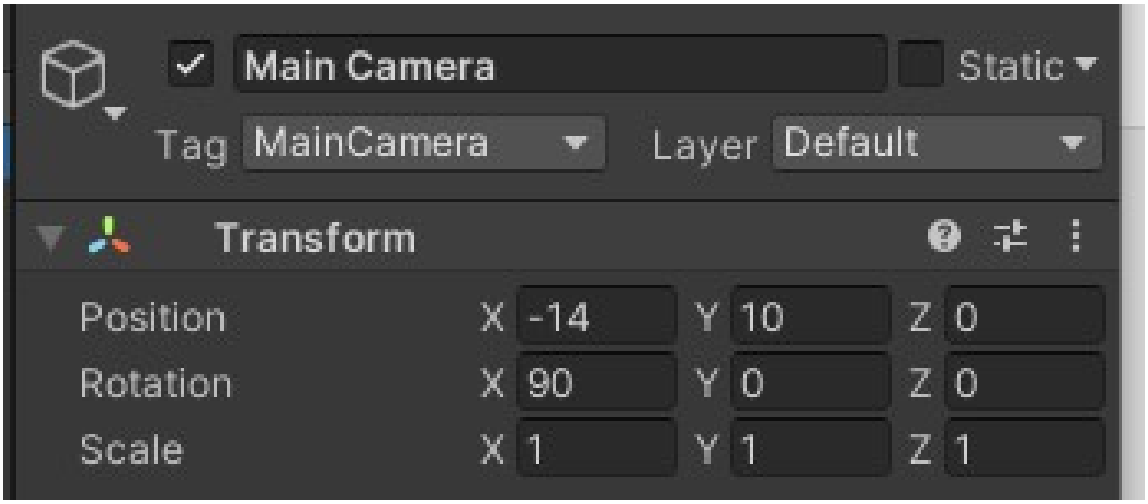
Σχετικά με την συμπεριφορά και τον έλεγχο του αντικειμένου «παίχτης» θα προσθέσουμε ένα κομμάτι κώδικα σε C# στην συνέχεια, αλλά τώρα έχουμε καλύψει το δομικό κομμάτι του αντικειμένου μέσα στο παιχνίδι.

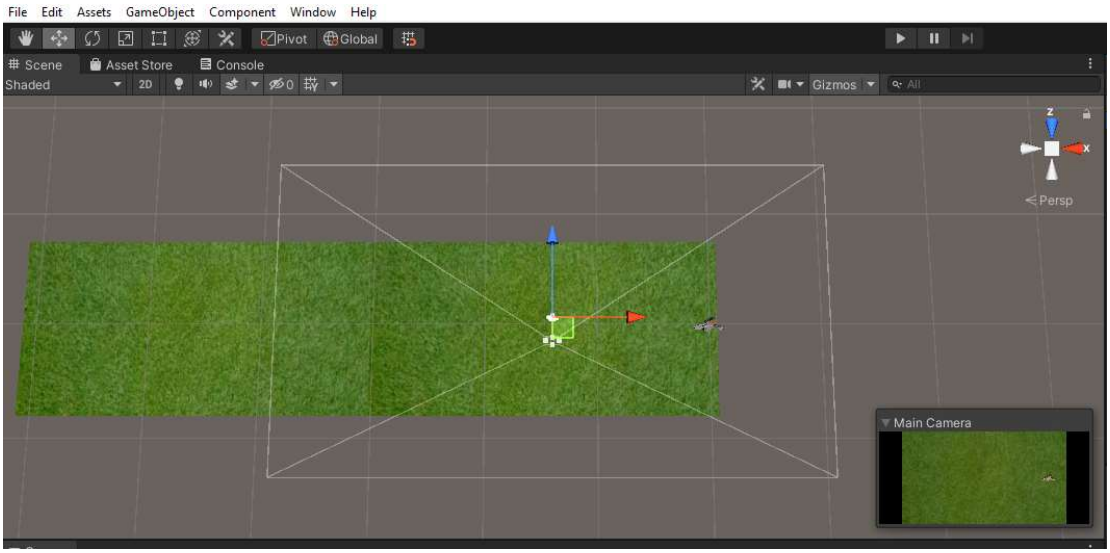
Κάμερα και Φώτα.

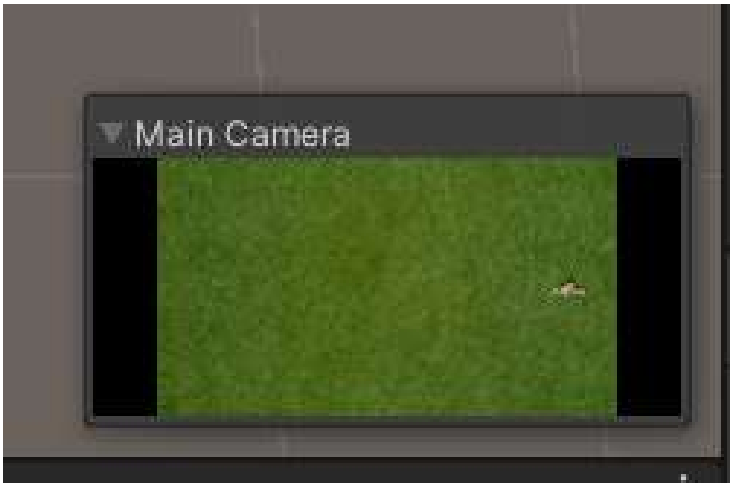
Σε αυτό το σημείο θα φτιάξουμε την κάμερα του παιχνιδιού μας και τον φωτισμό. Πάμε στο στοιχείο της κάμερας.



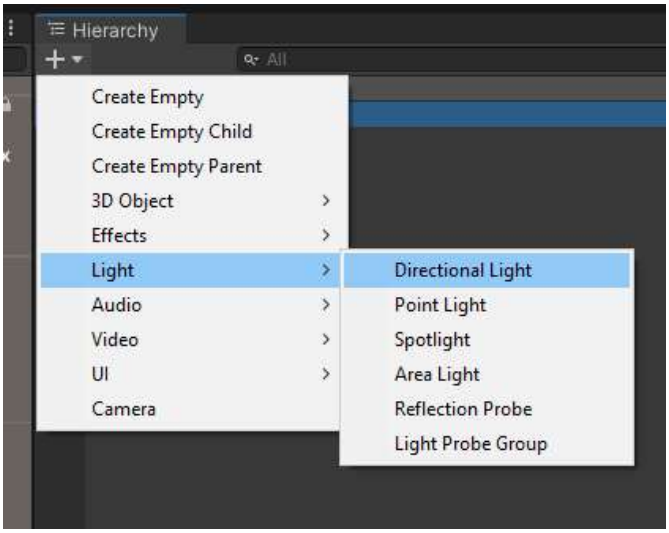
Εκεί επιλέγουμε η κάμερα μας να είναι ορθογραφική και η θέση της να είναι 10 μονάδες πάνω από τον παίχτη και να ξεκινά έχοντας τον 14 μονάδες αριστερά. Έτσι όταν ξεκινά το παιχνίδι ο παίκτης είναι στο δεξιό μέρος της οθόνης.

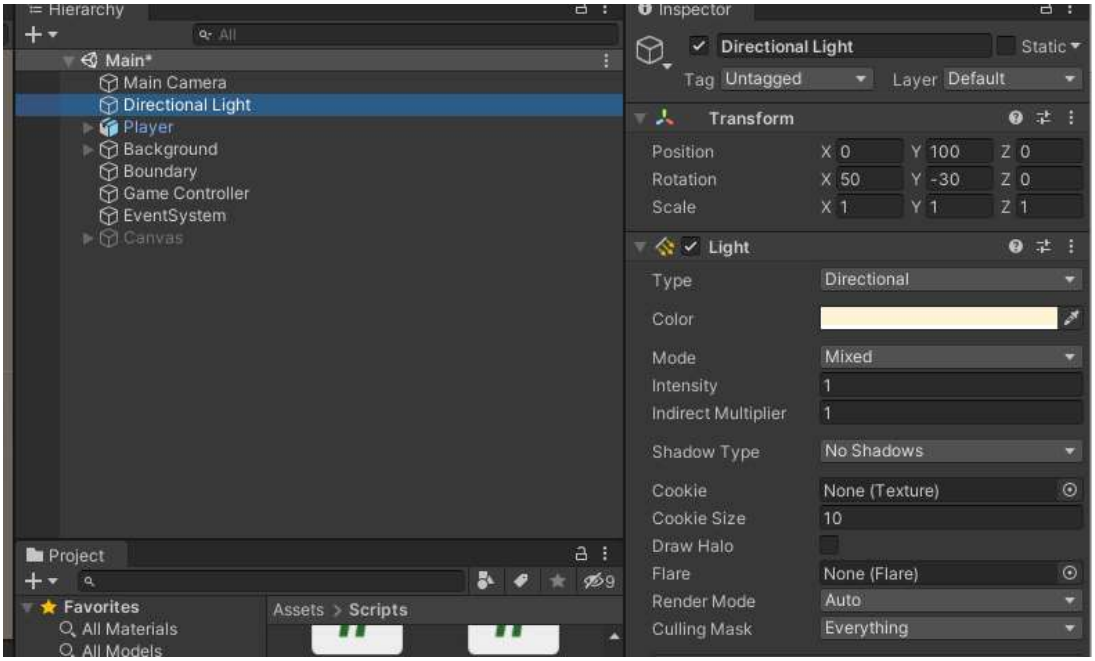




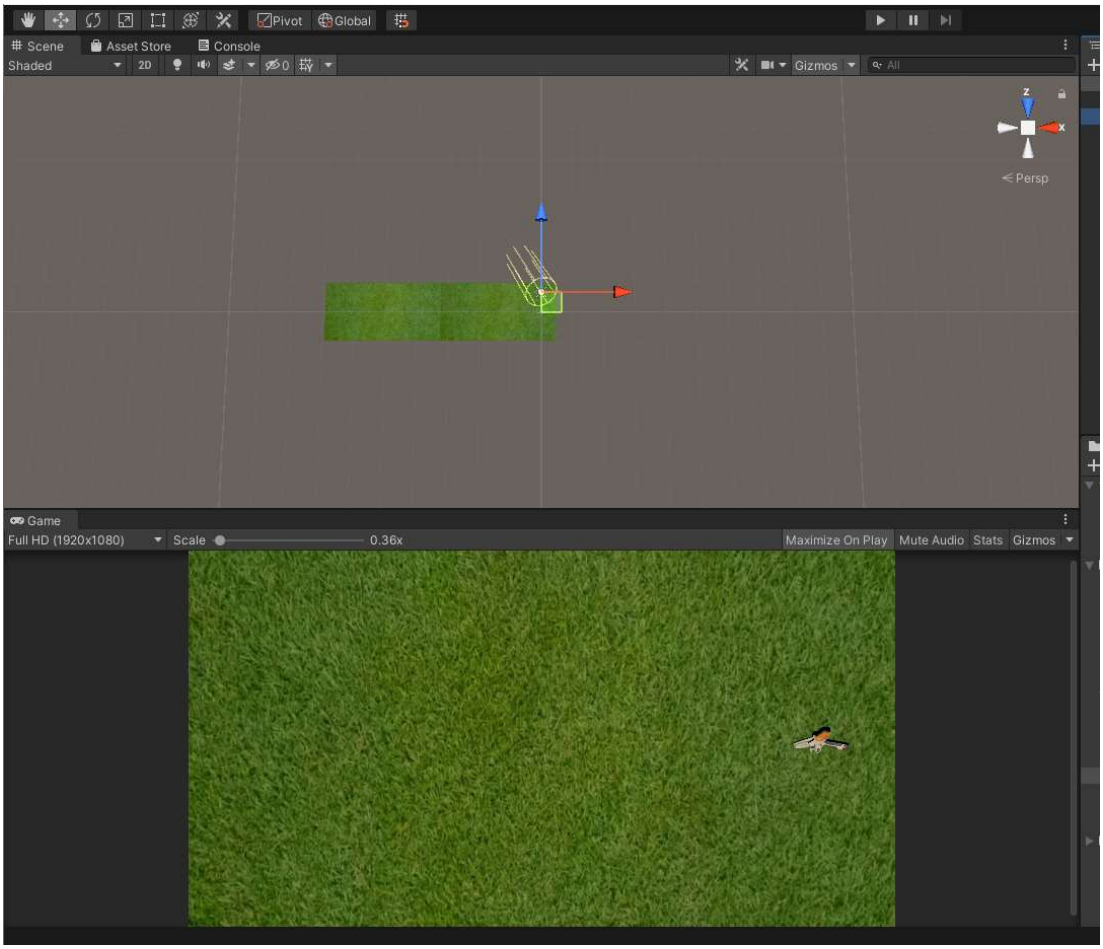


Για τον φωτισμό θα εισάγουμε ένα στοιχείο κατευθυνόμενου φωτισμού για να φτιάξουμε τον κύριο φωτισμό μας.



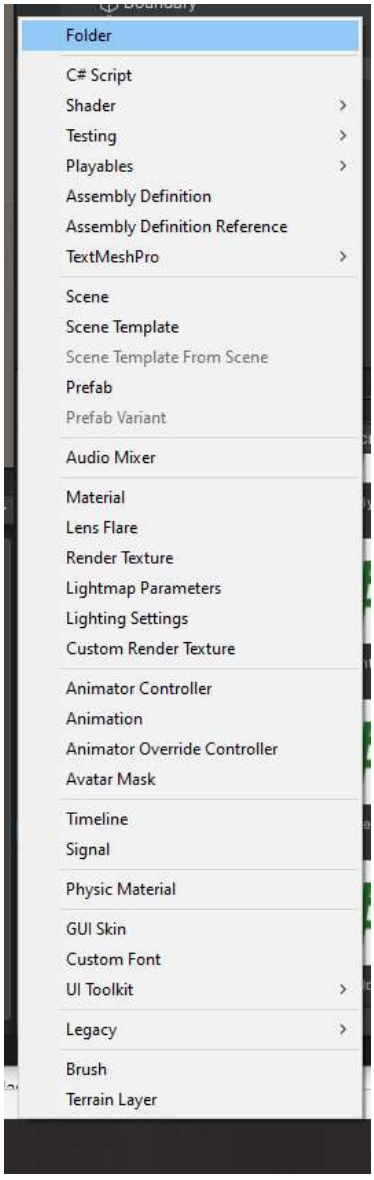


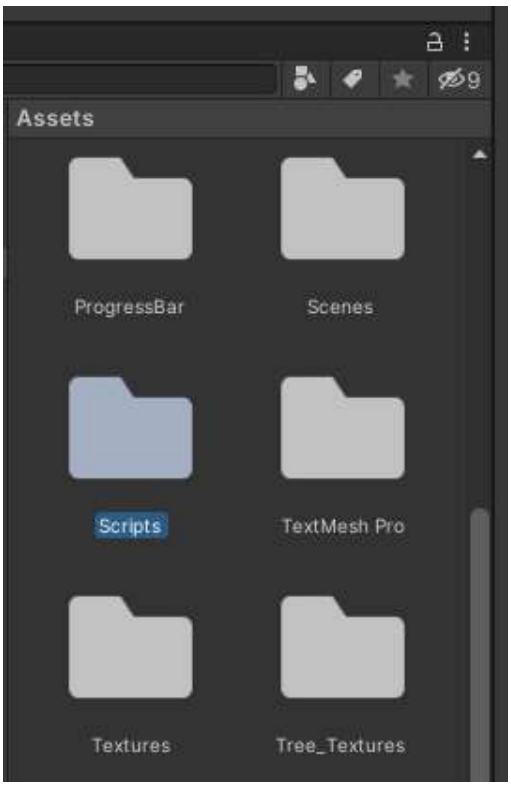
Για το φωτισμό του μοντέλου μας δεν χρειαζόμαστε κάτι ιδιαίτερο σε απαιτήσεις. Ορίζουμε την θέση του στο transform και παρατηρούμε αν το μοντέλο μας φωτίζεται ικανοποιητικά.



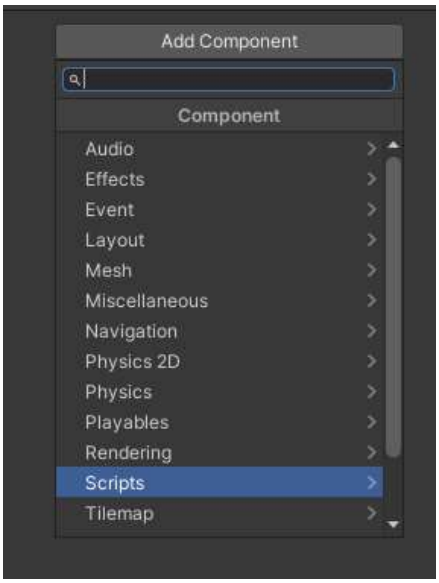
Κινώντας τον Παίχτη μας.

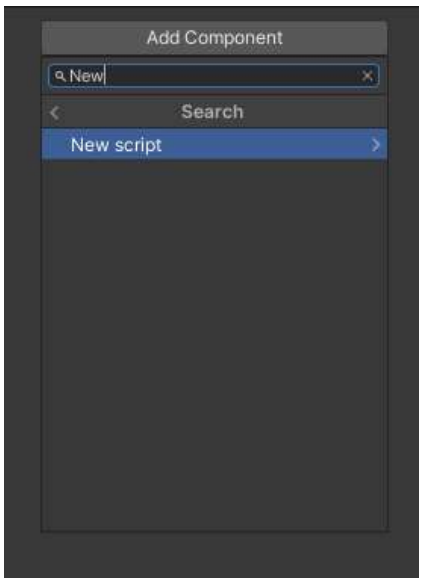
Για να ελέγξουμε την κίνηση του παίχτη μας πρέπει να εισάγουμε κάποιο κώδικα στην Unity και να τον παραμετροποιήσουμε. Αρχικά φτιάχνουμε έναν φάκελο όπου θα βάλουμε όλους τα αρχεία με κώδικα που θα προσθέσουμε.





Στην συνέχεια προσθέτουμε ένα στοιχείο νέου κώδικα στον παίχτη μας.





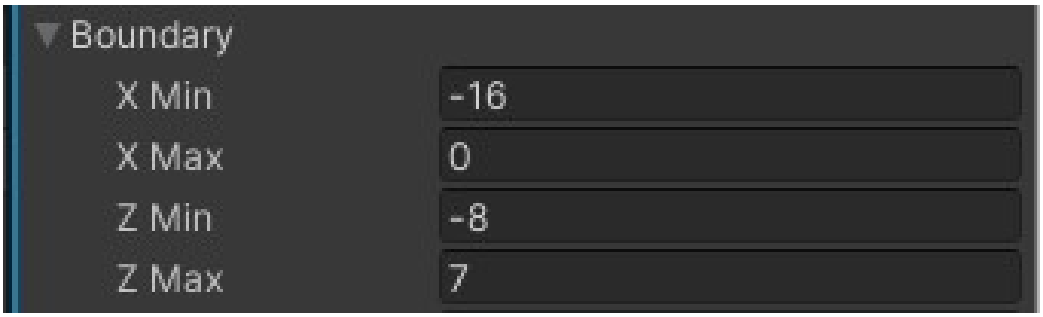
Προσθέτουμε τον παρακάτω κώδικα σε γλώσσα C#.

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

[System.Serializable]
public class Boundary
{
   public float xMin, xMax, zMin, zMax;
}

public class PlayerController : MonoBehaviour
{
   public float speed;
   public float tilt;
   public Boundary boundary;


   void Start()

   void FixedUpdate ()
   {
      float moveHorizontal = Input.GetAxis ("Horizontal");
      float moveVertical = Input.GetAxis ("Vertical");

      Vector3 movement = new Vector3 (moveHorizontal, 0.0f, moveVertical);
      GetComponent<Rigidbody>().velocity = movement * speed;

      GetComponent<Rigidbody>().position = new Vector3
```

```
        (
            Mathf.Clamp (GetComponent<Rigidbody>().position.x, boundary.xMin, boundary.xMax),
            0.0f,
            Mathf.Clamp (GetComponent<Rigidbody>().position.z, boundary.zMin, boundary.zMax)
        );

        GetComponent<Rigidbody>().rotation = Quaternion.Euler
(GetComponent<Rigidbody>().velocity.z * -tilt, 180.0f, 0.0f);
    }
}
```

Με τον κώδικα αυτό καθορίζουμε το πώς θα κινείται ο παίχτης μας και την ταχύτητα του. Κομμάτι που παραμετροποιούμε με το παρακάτω control.

Με την μεταβλητή Speed καθορίζουμε πόσο γρήγορα θα κινείται ο παίχτης μας. Με την μεταβλητή Tilt καθορίζουμε την κλίση που θα παίρνει ο παίχτης μας κατά την κίνηση του. Πριν που ορίσαμε την στάση του παίχτη μας με μια ελαφριά κλίση έτσι ώστε να φαίνεται το τρισδιάστατο μοντέλο, είπαμε ότι θα τονίσουμε την τρισδιάστατή φύση του μοντέλο και μέσα από τον κώδικα. Τώρα λοιπόν με την κλίση που θα δίνουμε στο μοντέλο μας κατά την κίνηση, αυτό θα περιστρέφεται στον άξονα του σαν να το κράταγε ένα χέρι και να το κουνά με φυσικότητα. Κατά την περιστροφή αυτή θα φαίνεται το μοντέλο μας και στις τρείς του διαστάσεις. Επίσης καθορίζουμε το μέχρι που θα κινείται ο παίχτης μας. Τα όρια της κίνησης του. Κομμάτι που παραμετροποιούμε με το παρακάτω control.



Φίλοι και Εχθροί του Παίχτη μας.

Πέρα από το αντικείμενο του παίχτη μας θα εισάγουμε στο παιχνίδι ακόμη πέντε στοιχεία. Θα εισάγουμε 2 στοιχεία που θα αποτελούν απειλή για τον παίχτη μας, 2 στοιχεία που θα δεν θα αποτελούν απειλή αλλά θα συμβάλουν στο να αυξομειώνει τους πόντους που θα συλλέγει και ένα στοιχείο που θα παρατείνει την ζωή του παίχτη μας στο παιχνίδι.

Τα στοιχεία αυτά είναι τα εξής:

Πέτρα: Το στοιχείο αυτό θα αποτελεί απειλή για τον παίχτη. Αν συγκρουστεί με αυτό η ζωή του τελειώνει και το ίδιο και το παιχνίδι.

Καρφί: Το στοιχείο αυτό θα αποτελεί απειλή για τον παίχτη. Αν συγκρουστεί με αυτό η ζωή του τελειώνει και το ίδιο και το παιχνίδι.

Κούτσουρο: Το στοιχείο αυτό δεν αποτελεί απειλή για τον παίχτη. Αν συγκρουστεί με αυτό θα κερδίζει 10 βαθμούς.

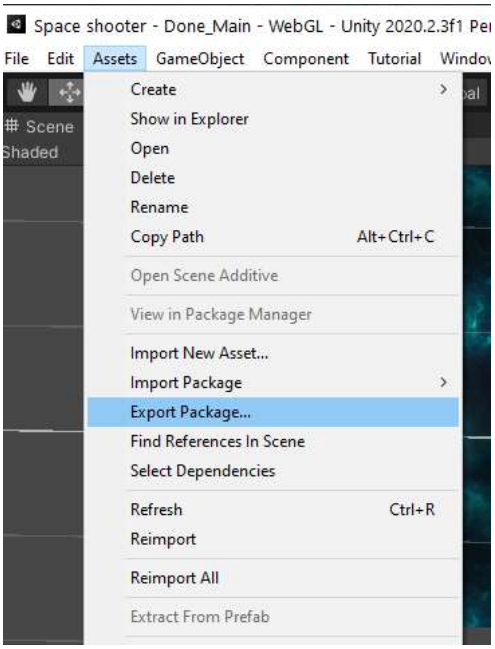
Φύλλο: Το στοιχείο αυτό δεν αποτελεί απειλή για τον παίχτη. Αν συγκρουστεί με αυτό όμως θα χάνει 2 βαθμούς, διότι δεν χρησιμοποιούμε αλυσοπρίονα για να κόψουμε φύλλα.

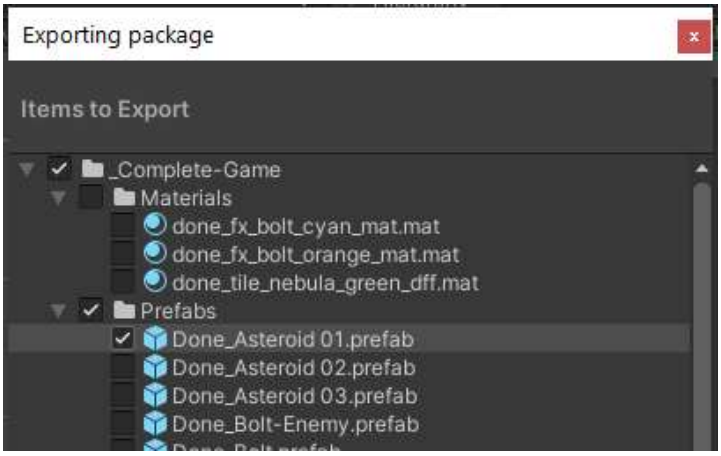
Μπαταρία: Το στοιχείο αυτό δεν αποτελεί απειλή για τον παίχτη. Αν συγκρουστεί με αυτό θα ανανεώνεται η διάρκεια ζωής της μπαταρίας του αλυσοπρίονου – παίχτη. Αν η στάθμη της μπαταρίας μηδενιστεί το παιχνίδι θα τελειώνει.

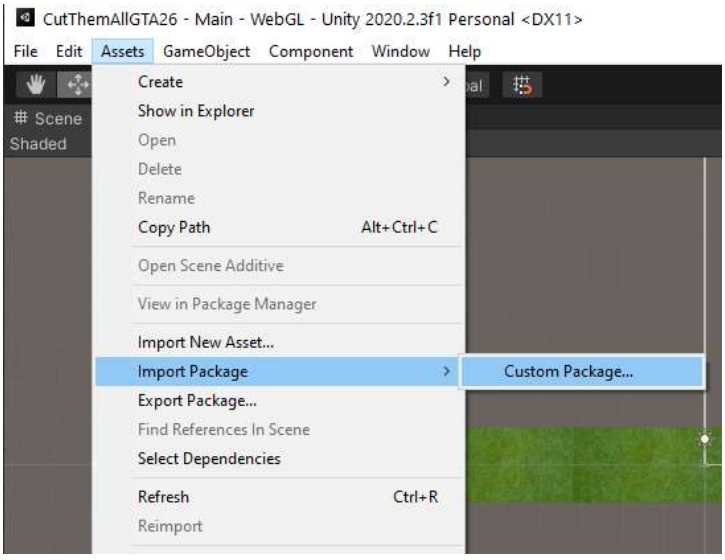
Το ενδιαφέρον όμως στην όλη διαδικασία είναι ότι για να εισάγουμε τα στοιχεία αυτά στο παιχνίδι μας, θα χρησιμοποιήσουμε τρείς διαφορετικές μεθόδους. Αν αυτές προστεθούν στην πρώτη μέθοδο με την οποία δημιουργήσαμε τον κύριο παίχτη, θα παραθέτουμε στον αναγνώστη της εργασίας αυτής τέσσερεις τρόπους για να εισάγει τα επιθυμητά για αυτόν αντικείμενα στο παιχνίδι του. Αυτό αποτελεί κι έναν από τους κύριους πυλώνες με τους οποίους θέλουμε να παροτρύνουμε τον αναγνώστη μας να ασχοληθεί με την δημιουργία μοντέλων και εφαρμογών, μιας και οι δυνατότητες είναι πραγματικά αμέτρητες.

Πέτρα

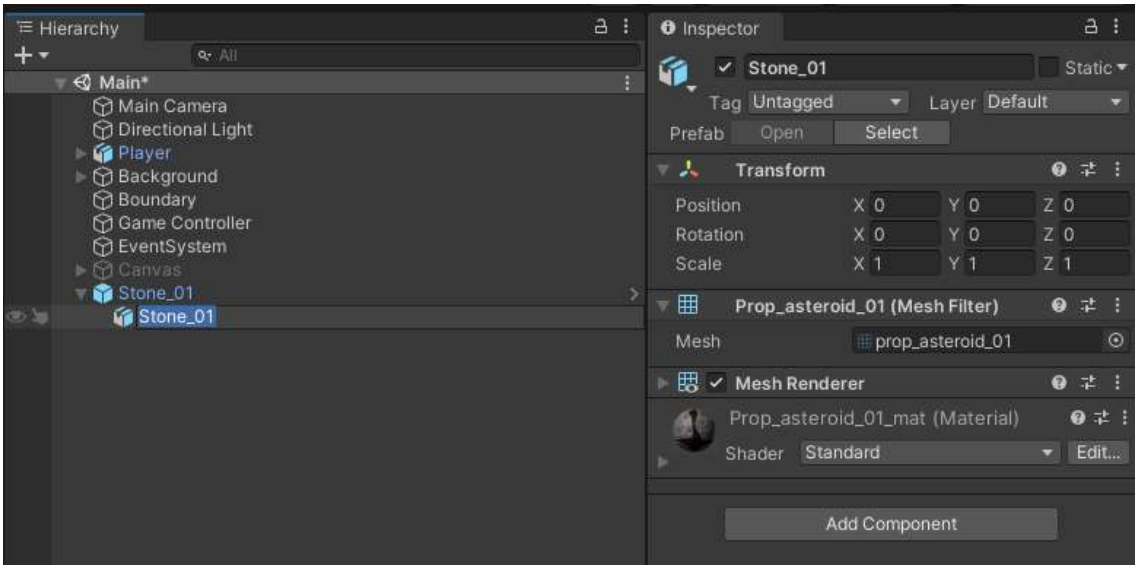
Πάμε να ξεκινήσουμε με τον πιο εύκολο θα λέγαμε τρόπο εισαγωγής ενός αντικειμένου στην unity. Μπαίνοντας στο asset store της unity μπορεί κανείς να βρει μια πληθώρα από αντικείμενα ,είτε δωρεάν είτε επί πληρωμή. Εμείς το αντικείμενο της πέτρας το βρήκαμε από ένα άλλο project thw unity το οποίο ήταν δωρεάν κι από το οποίο ήταν πολύ εύκολο με μερικά βήματα να το εξάγουμε σαν αντικείμενο και να το εισάγουμε στο δικό μας project. Ακολουθούν μερικές ενδεικτικές εικόνες από την διαδικασία.

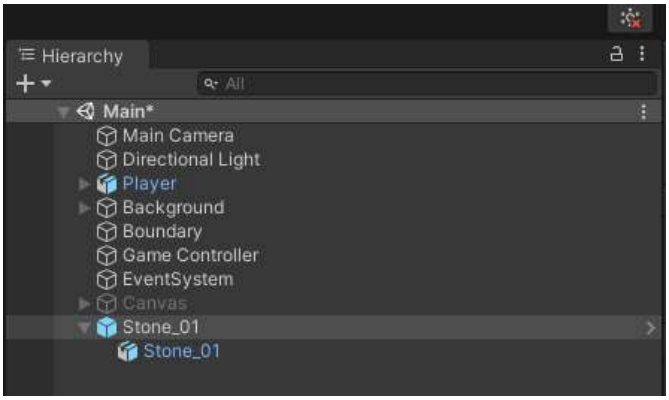

Space shooter - Done_Main - WebGL - Unity 2020.2.3f1 Pe
File Edit Assets GameObject Component Tutorial Windo
Scene
Shaded
Create
Show in Explorer
Open
Delete
Rename
Copy Path Alt+Ctrl+C
Open Scene Additive
View in Package Manager
Import New Asset...
Import Package
Export Package...
Find References In Scene
Select Dependencies
Refresh Ctrl+R
Reimport
Reimport All
Extract From Prefab


Exporting package
Items to Export
_Complete-Game
Materials
done_fx_bolt_cyan_mat.mat
done_fx_bolt_orange_mat.mat
done_tile_nebula_green_dff.mat
Prefabs
Done_Asteroid 01.prefab
Done_Asteroid 02.prefab
Done_Asteroid 03.prefab
Done_Bolt-Enemy.prefab

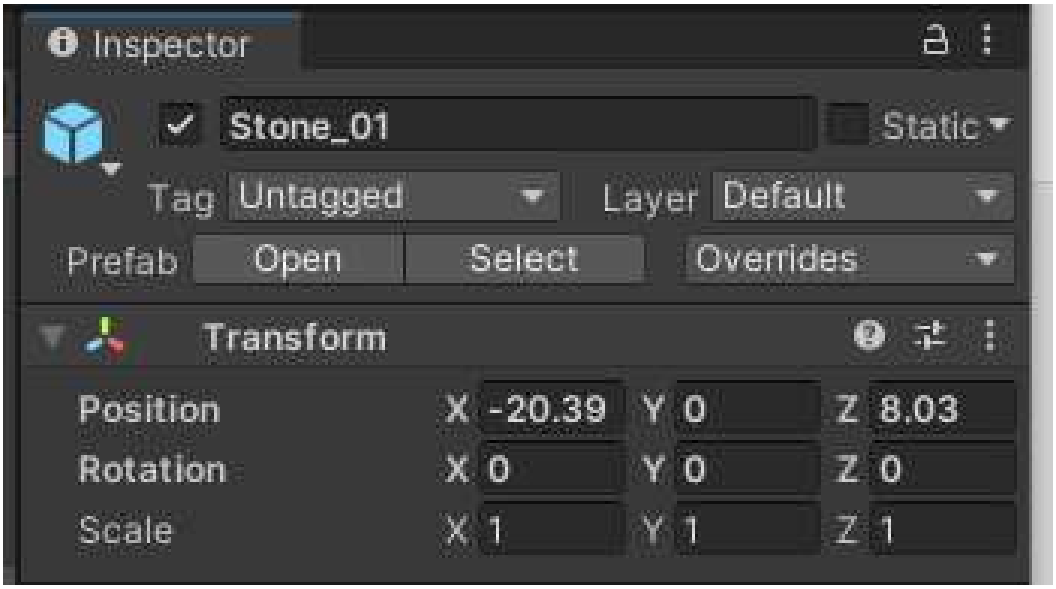

CutThemAllGTA26 - Main - WebGL - Unity 2020.2.3f1 Personal <DX11>
File Edit Assets GameObject Component Window Help
Scene
Shaded
Create
Show in Explorer
Open
Delete
Rename
Copy Path Alt+Ctrl+C
Open Scene Additive
View in Package Manager
Import New Asset...
Import Package
Custom Package...
Export Package...
Find References In Scene
Select Dependencies
Refresh Ctrl+R
Reimport

Δημιουργούμε ένα νέο 3D αντικείμενο και το ονομάζουμε Stone_01. Εκεί προσαρτούμε υπο-αντικείμενο το asset που έχουμε εισαγάγει.



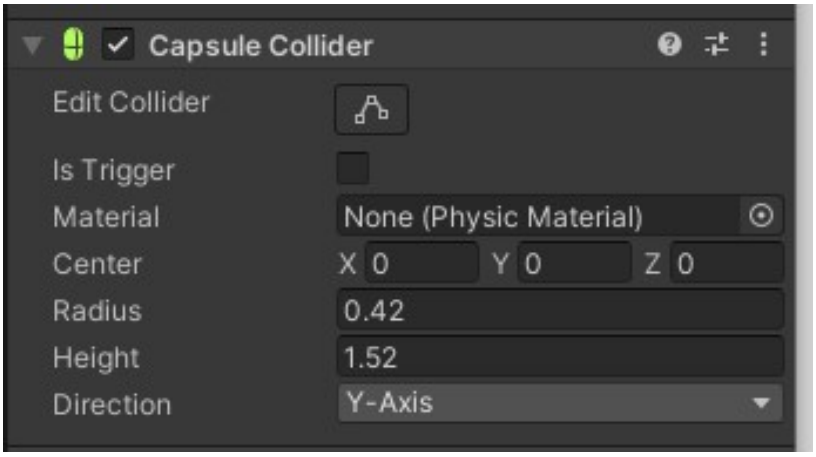


Αρχικά καθορίζουμε την θέση στη σκηνή, την περιστροφή και την κλίμακα.

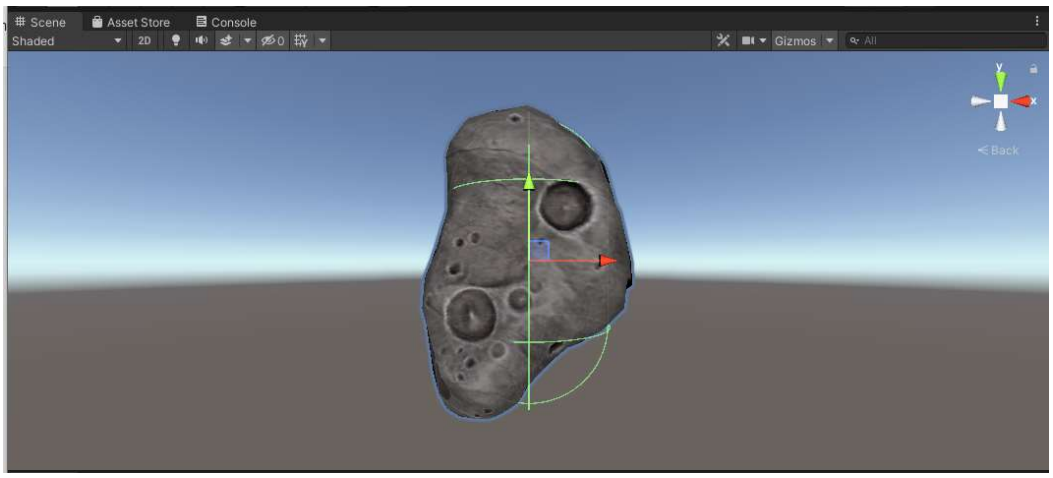


Ύστερα προσδίδουμε φυσικά χαρακτηριστικά.

Και τέλος περικλείουμε την πέτρα μας μέσα σε μια κάψουλα για να οριοθετήσουμε τα χώρο της κατά τις επερχόμενες συγκρούσεις.



Το αποτέλεσμα της «κάψουλας» φαίνεται παρακάτω στην εικόνα.



Πέρα όμως από την δομή της πέτρας μας, θα της δώσουμε και κίνηση. Αυτό επιτυγχάνεται με την προσθήκη ενός μικρούς κομματιού κώδικα. Φτιάχνουμε ένα νέο αρχείο στο φάκελο Scripts που το ονομάζουμε Mover. Εκεί τοποθετούμε τον παρακάτω κώδικα C#.

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

public class Mover : MonoBehaviour
{
    public float speed;

    void Start ()
    {
        GetComponent<Rigidbody>().velocity = transform.right * speed;
    }
}
```

Αυτόν τον κώδικα τον προσαρτώ στο αντικείμενο πέτρα. Από εκεί είμαι σε θέση να ελέγξω την ταχύτητα με την οποία θα κινείται αυτό το αντικείμενο προς τα δεξιά όπως φαίνεται παρακάτω.

Επιλέγω να δώσω μια ταχύτητα 5. Αν έβαζα αρνητική τιμή, το αντικείμενο θα κινηθεί προς τα αριστερά. Από το πρώτο κιόλας λεπτό όμως παρατηρούμε ότι με τον τρόπο αυτό έχουμε δώσει κίνηση στην πέτρα μας η οποία έρχεται ναι μεν προς τα δεξιά, αλλά έρχεται στατικά και «αφύσικα». Ποτέ μια πέτρα δεν ταξιδεύει στον αέρα στατικά. Πάντα έχει μια περιστροφή στον αέρα, είτε ελαφριά είτε πιο έντονη. Αυτή είναι η φυσική κίνηση και αυτήν θα προσπαθήσουμε να προσδώσουμε στο αντικείμενο μας, έτσι ώστε να είναι πιο αληθοφανές και πιο όμορφο το αποτέλεσμα του παιχνιδιού μας σαν εμπειρία. Για να το καταφέρουμε αυτό, φτιάχνουμε ένα νέο αρχείο στο φάκελο Scripts που το ονομάζουμε RandomRotator. Εκεί τοποθετούμε τον παρακάτω κώδικα C#.

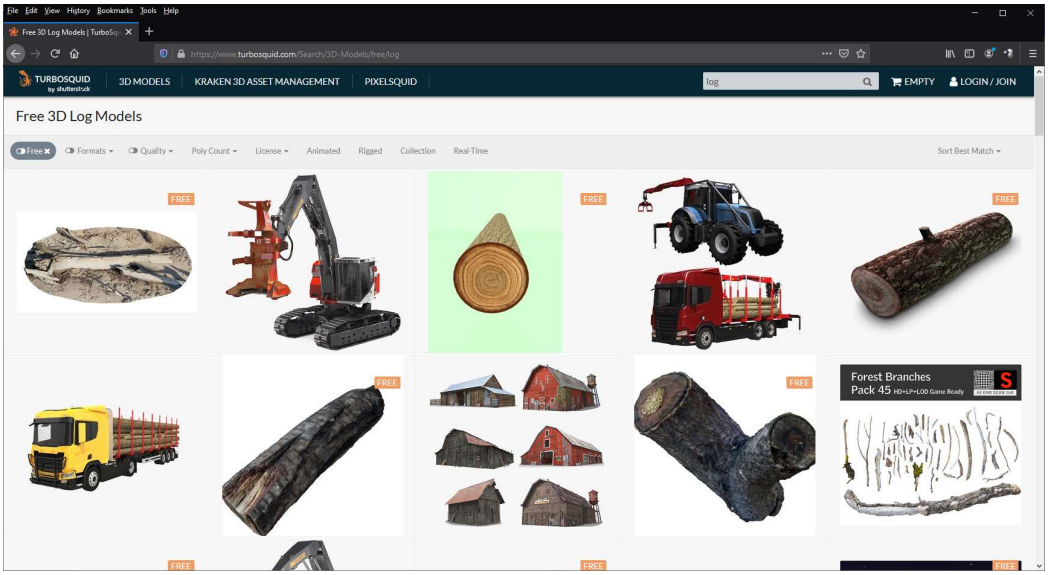
```
using System.Collections;
using System.Collections.Generic;
using UnityEngine;

public class RandomRotator : MonoBehaviour
{
    public float tumble;

    void Start ()
    {
        GetComponent<Rigidbody>().angularVelocity = Random.insideUnitSphere * tumble;
    }
}
```

Ήδη από το όνομα που δώσαμε στο αρχείο γίνεται κατανοητό ότι θέλουμε να δώσουμε στο αντικείμενο μας μια τυχαία περιστροφή. Αυτή η ιδιότητα του αντικειμένου είναι ελεγχόμενη από την μεταβλητή tumble όπως φαίνεται παρακάτω στην εικόνα.

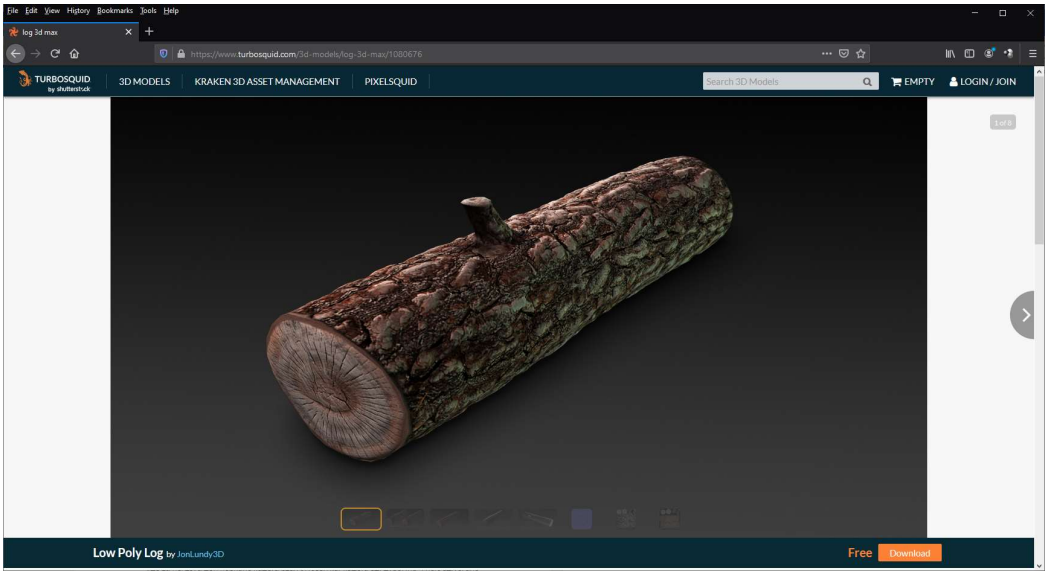
Επιλέγουμε να δώσουμε την τιμή 4 στην μεταβλητή αυτή μετά από δοκιμές, για να μοιάζει το αποτέλεσμα της κίνησης αληθοφανές. Μια υπερβολικά μεγάλη τιμή θα περιστρέφει το αντικείμενο μας αφύσικα γρήγορα, με αντίθετο αποτέλεσμα από αυτό που θέλουμε.

Κούτσουρο.

Ένα άλλο στοιχείο που θα προσθέσουμε στο παιχνίδι μας είναι ένα κούτσουρο. Αυτό το αντικείμενο θα το προσθέσουμε με έναν διαφορετικό τρόπο από αυτούς που έχουμε χρησιμοποιήσει μέχρι τώρα. Το διαδίκτυο έχει πολλές συλλογές από 3D αντικείμενα. Μια από αυτές είναι και η turbosquid όπου αναζητήσαμε ένα τρισδιάστατο αντικείμενο που να σχετίζεται με κορμό δέντρου ή κούτσουρο.

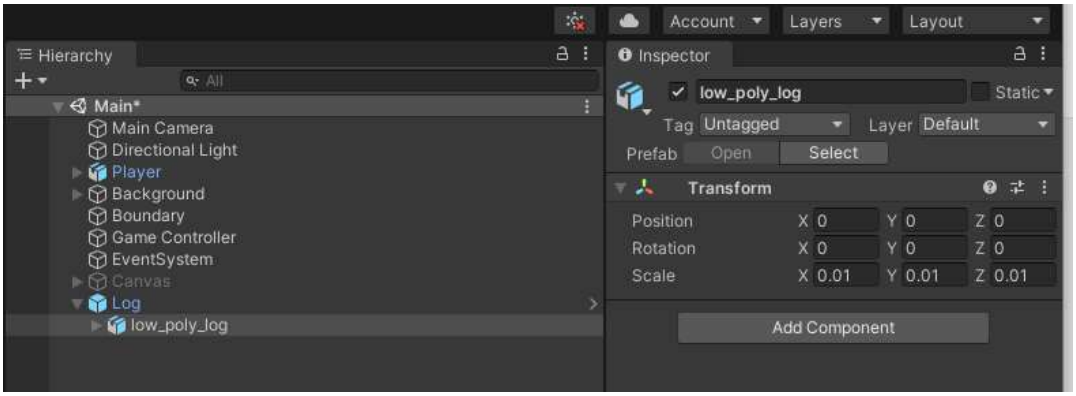


Από τα μοντέλα που βρήκαμε κάποια ήταν δωρεάν και κάποια επί πληρωμή. Εμείς επιλέξαμε ένα μοντέλο που θεωρήσαμε ότι θα ταιριάζει πιο πολύ στην φύση του παιχνιδιού που ετοιμάζομε.

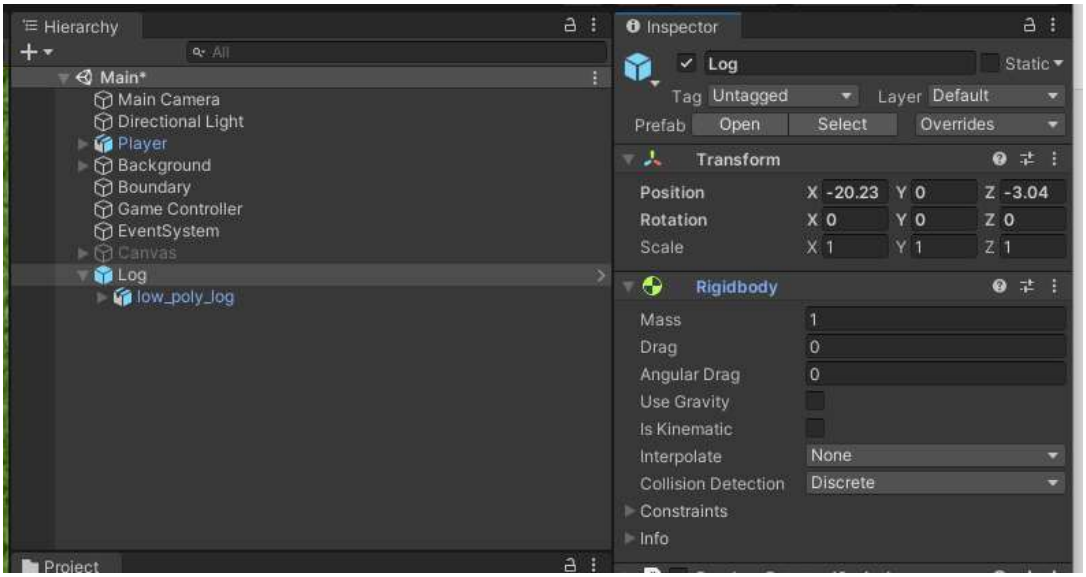


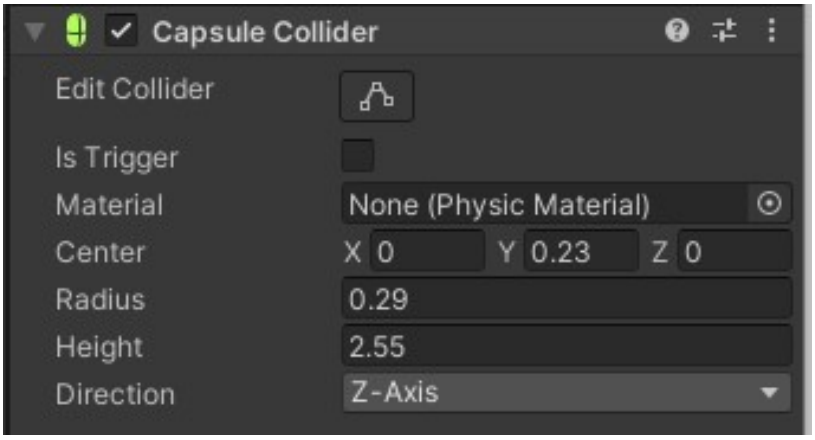
Η αναζήτηση στο διαδίκτυο για έτοιμα μοντέλα αποτελεί άλλον ένα τρόπο να βάλουμε στα παιχνίδια ή τις εφαρμογές μας τα επιθυμητά για εμάς μοντέλα.

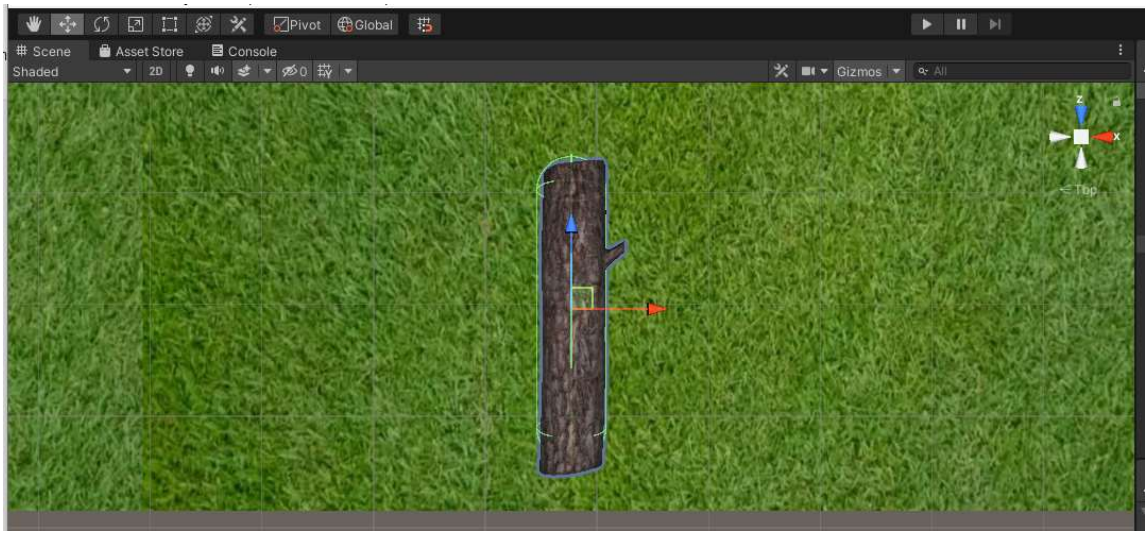
Αφού κατεβάσουμε το μοντέλο που μας αρέσει το εισάγουμε στην unity και το προετοιμάζουμε για το παιχνίδι μας. Δημιουργούμε ένα νέο 3D αντικείμενο και το ονομάζουμε Stone_01. Εκεί προσαρτούμε υπο-αντικείμενο το μοντέλο που έχουμε εισαγάγει.



Στην συνέχεια ακολουθούμε την ίδια διαδικασία που ακολουθήσαμε για την πέτρα. Προσθέτουμε Rigidbody και Capsule Collider



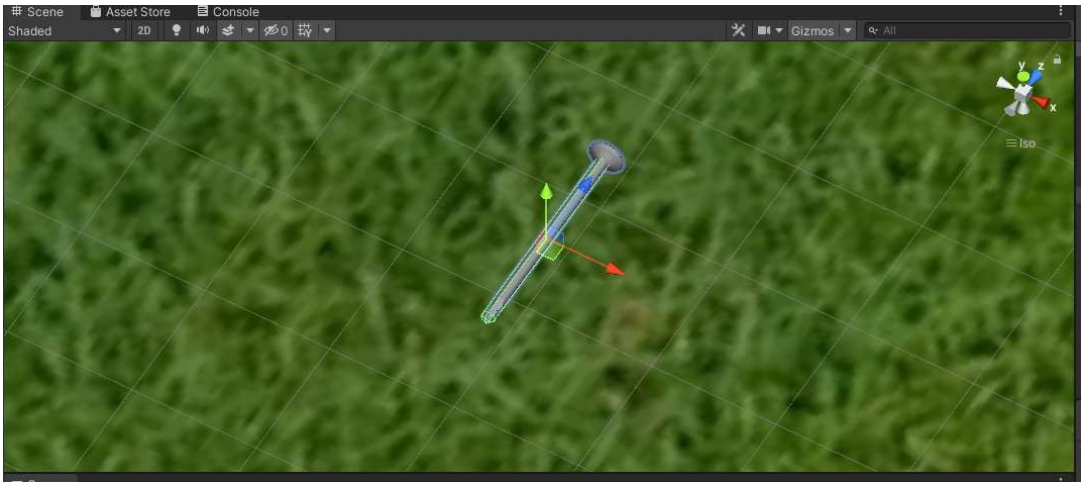




Επίσης για να δώσουμε κίνηση στο αντικείμενο αυτό θα προσθέσουμε τον κώδικα Mover και Random Rotator όπως κάναμε και στην πέτρα.

Καρφί

Άλλο ένα αντικείμενο που θα προσθέσουμε με τον ίδιο τρόπο με το κούτσουρο είναι ένα καρφί. Το βρίσκουμε κι αυτό από συλλογή τρισδιάστατων μοντέλων στο διαδίκτυο και ακολουθούμε την ίδια διαδικασία.

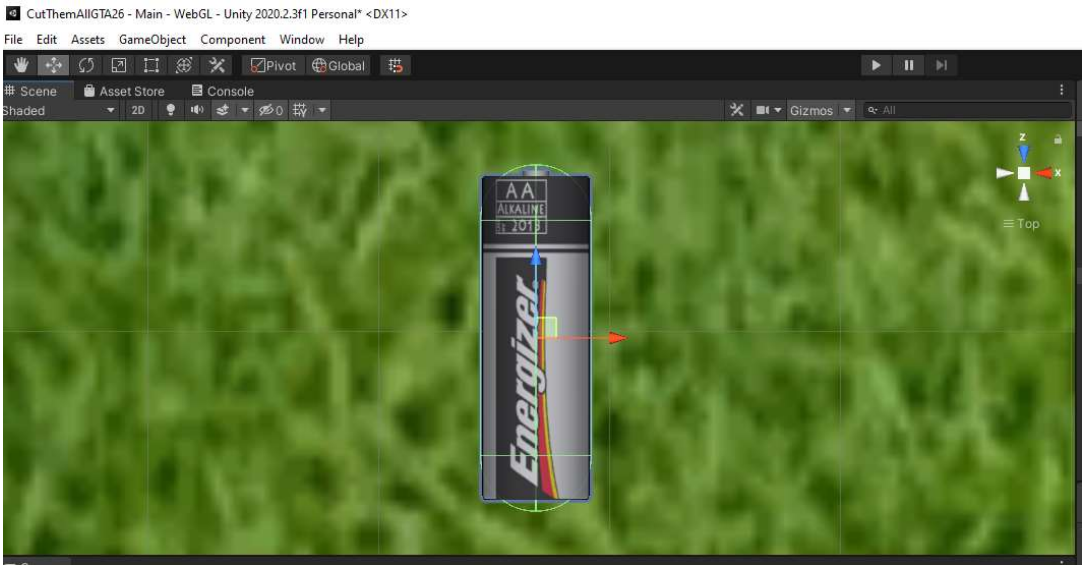


Μπαταρία

Το μηχανημα που αποτελεί τον κύριο παίχτη είναι ένα μηχανημα μπαταρίας. Οπότε θεωρήσαμε καλή ιδέα να προσθέσουμε ένα στοιχείο μπαταρίας στο παιχνίδι έτσι ώστε κάθε φορά που ο παίχτης θα συλλέγει μια μπαταρία να του ανανεώνει την ενέργεια.

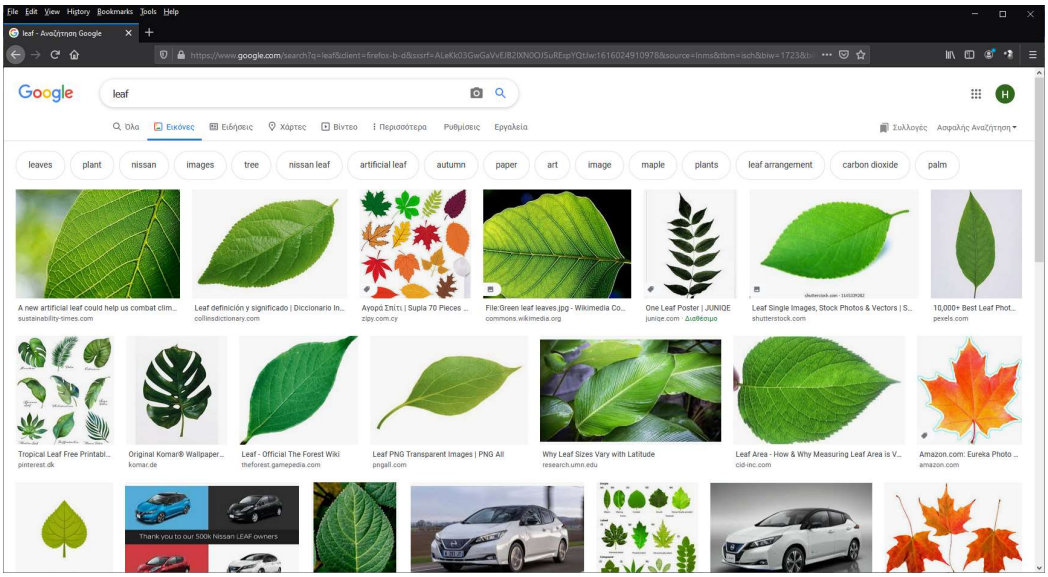
Η μπαταρία του συγκεκριμένου μηχανήματος είναι όπως απεικονίζεται στις παρακάτω φωτογραφίες.

Όπως γίνεται κατανοητό ένα τέτοιο αντικείμενο δεν θα ήταν άμεσα αναγνωρίσιμο σε έναν μη εξοικειωμένο με αυτά τα προϊόντα παίχτη. Έτσι λοιπόν θεωρήσαμε ότι είναι καλύτερο για το αντικείμενο μπαταρία να προσθέσουμε ένα τρισδιάστατο μοντέλο μιας ευρέως γνωστής μπαταρίας. Έτσι θα γινόταν άμεσα αντιληπτό από τον εκάστοτε παίχτη άμεσα. Βρήκαμε κι αυτό από συλλογή τρισδιάστατων μοντέλων στο διαδίκτυο και ακολουθούμε την ίδια διαδικασία.


CutThemAllGTA26 - Main - WebGL - Unity 2020.2.3f1 Personal* <DX11>
File Edit Assets GameObject Component Window Help
Pivot Global
Scene Asset Store Console
Shaded 2D Gizmos All
AA
ALKALINE
2018
Energizer
Top

Φύλλο

Στην περίπτωση του φύλλου σαν αντικείμενο ακολουθήσαμε μια ακόμη διαφορετική τεχνική, την τέταρτη κατά σειρά. Αναζητήσαμε εικόνες από φύλλα στο διαδίκτυο.



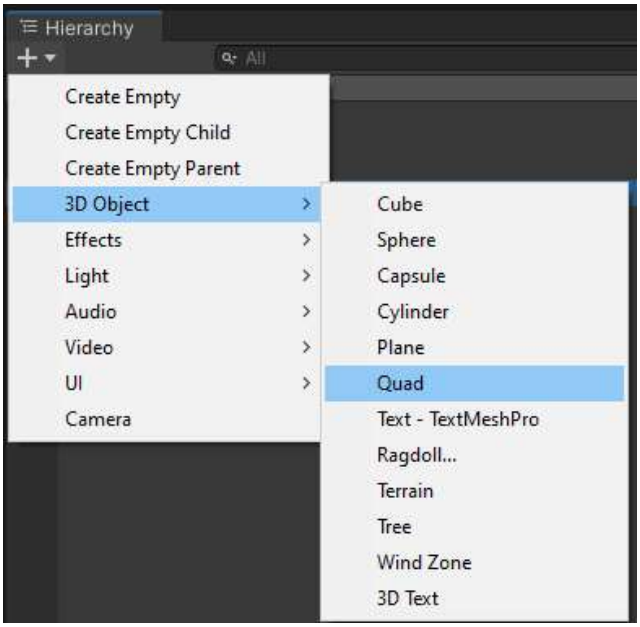
Από αυτά ξεχωρίσαμε μερικά που θα μπορούσαν να γίνουν η βάση για το αντικείμενο μας.

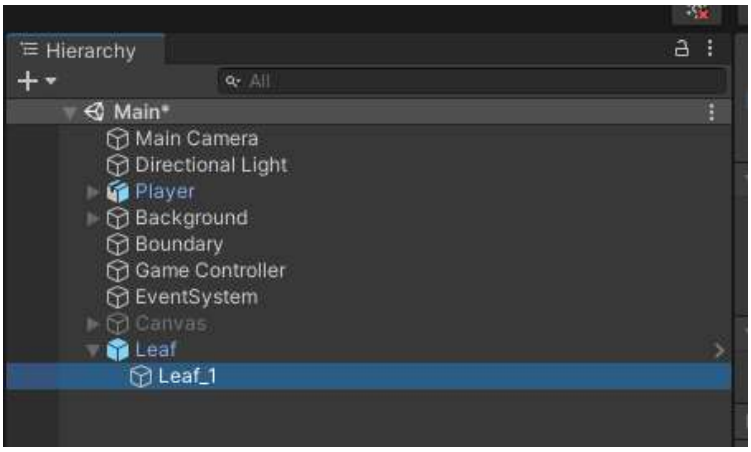
Επειδή το φόντο μας είναι πράσινο επιλέξαμε το κοκκινωπό φύλλο για να κάνει καλύτερη αντίθεση. Ύστερα πήραμε την εικόνα αυτή και αντιστρέψαμε το χρώμα του φόντου της από άσπρο σε μαύρο.


πριν

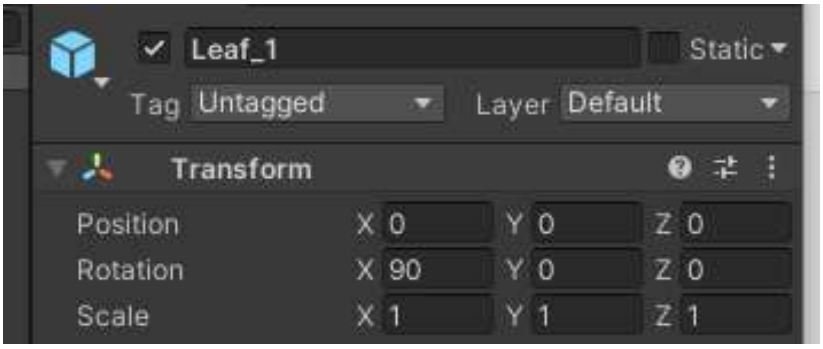
μετά 

Την αλλαγή αυτή την κάναμε για συγκεκριμένο λόγο που θα εξηγήσουμε αμέσως. Πάμε στην unity και δημιουργούμε ένα νέα 3D Quade αντικείμενο και το ονομάζουμε Leaf.

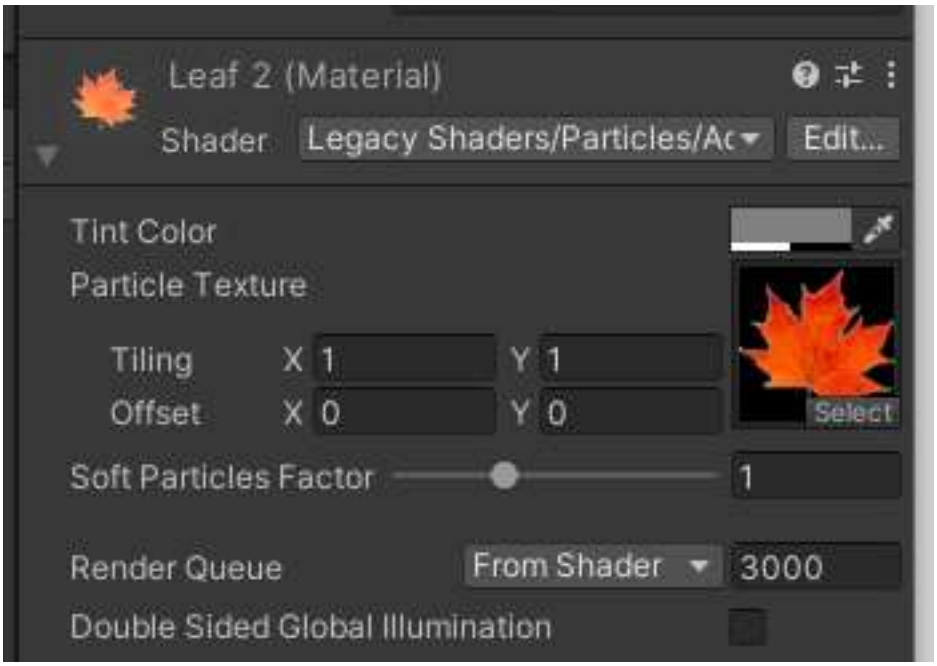




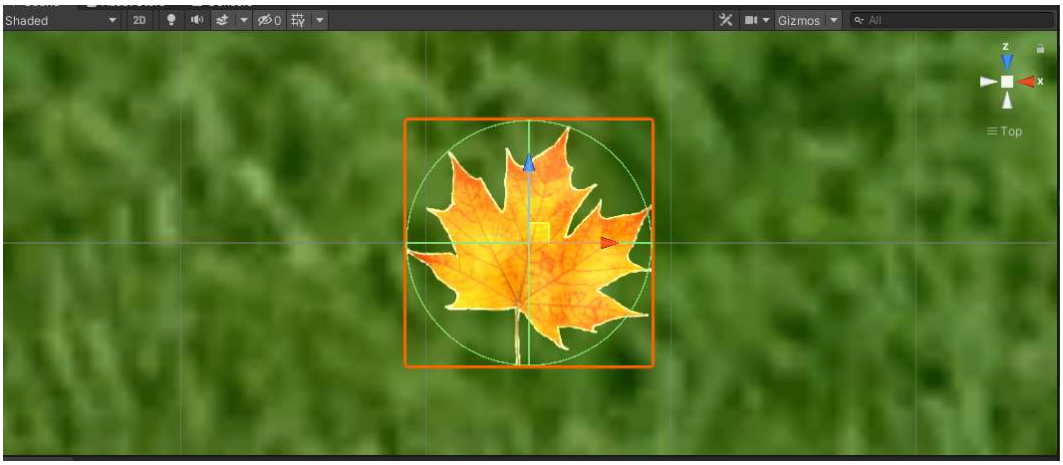
Εισάγουμε την φωτογραφία του φύλλου με το μαύρο φόντο στην unity και προσαρτούμε αυτήν την εικόνα στο πλακίδιο (Quad) που μόλις φτιάξαμε. Δίνουμε το επιθυμητό transform.



Στην συνέχεια επιλέγουμε το shader: Legacy shader / Particles / Additive



Αυτό το Shaderέχει σαν αποτέλεσμα να εξαφανίζει τα σκουρόχρωμα μέρη της εικόνας και να τονίζει τα πιο φωτεινά.



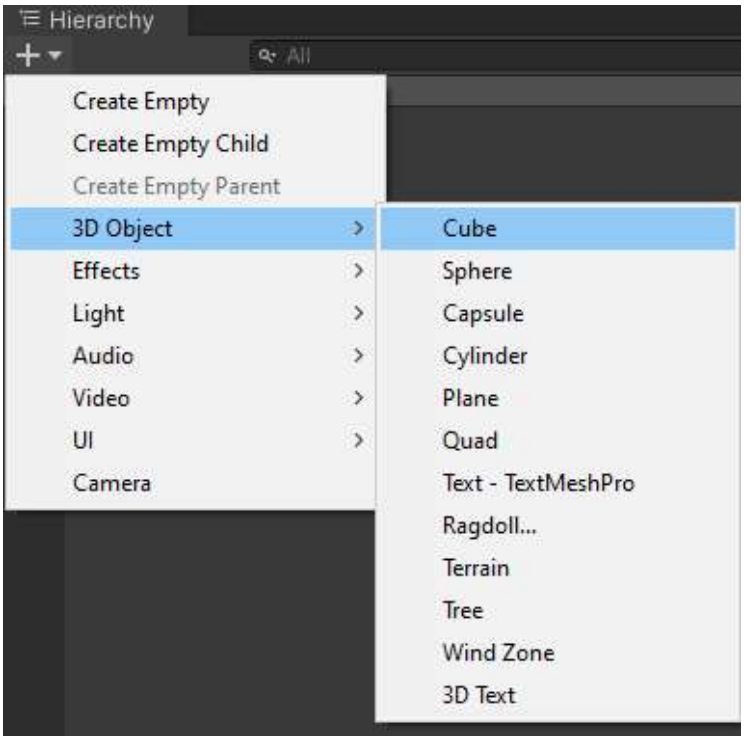
Σαν αποτέλεσμα το φύλλο μας φαίνεται πολύ καθαρά πάνω στο γρασίδι και το μαύρο φόντο ουσιαστικά εξαφανίζεται. Για αυτό το λόγο ήταν επιτακτική ανάγκη να αλλάξουμε το λευκό φόντο της της αρχικής φωτογραφίας.

Στο αντικείμενο αυτό θα προσθέσουμε ότι και στα υπόλοιπα αντικείμενα όπως Rigidbody, Mover Script και Random Rotator Scripts. Η μόνη διαφορά είναι ότι αντί για Capsule collider θα προσθέσουμε ένα Sphere collider το οποίο το καλύπτει επαρκώς.

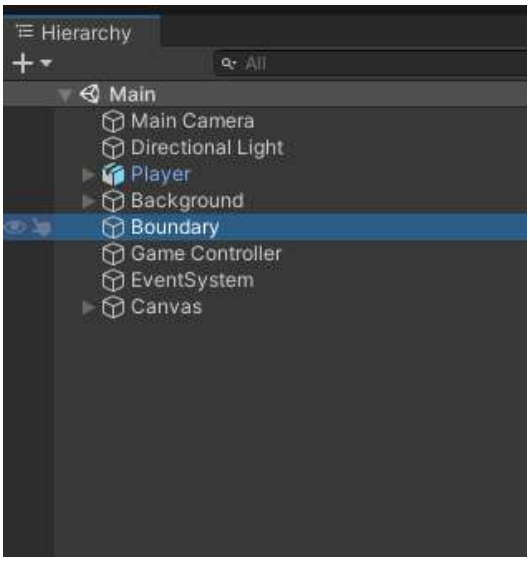
Το ιδιαίτερο στοιχεί με αυτό το αντικείμενο είναι το γεγονός ότι είναι ουσιαστικά μια δισδιάστατη εικόνα που με την κίνηση και τα χαρακτηριστικά που της έχουμε δώσει φέρεται και φαίνεται σαν ένα τρισδιάστατο μοντέλο. Ο συνδυασμός των Mover και Random Rotator κάνουν το φύλλο να κινείται στο γκαζόν μας σαν να το παρασέρνει ο άνεμος.

Βάζοντας Όρια.

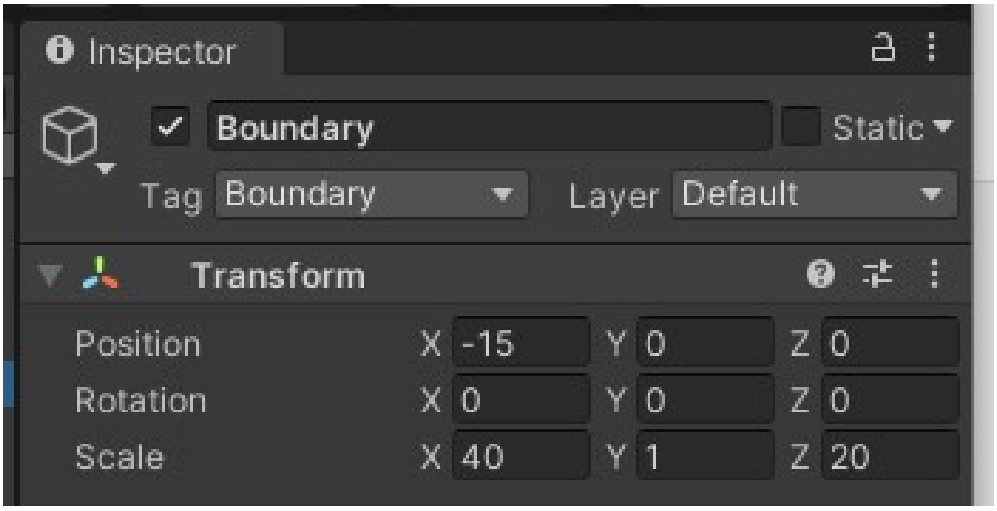
Σε αυτό το σημείο γίνεται αισθητό ότι πρέπει με κάποιο τρόπο να οριοθετήσουμε τον χώρο του παιχνιδιού. Δεν μπορεί για παράδειγμα τα αντικείμενα μας να ταξιδεύει αιώνια στο χώρο. Πρέπει να βάλουμε ένα όριο. Για να το πετύχουμε αυτό εισάγουμε ένα νέο αντικείμενο στο παιχνίδι τύπου Cube.



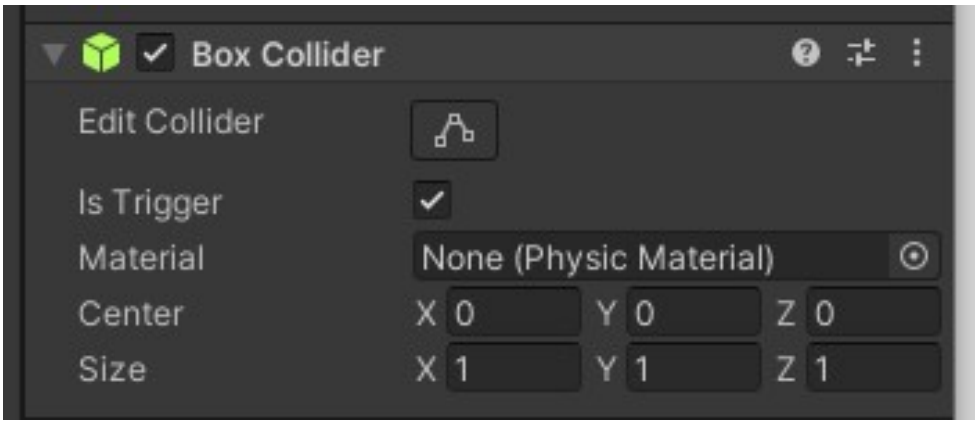
Το ονομάζουμε Boundary και το παραμετροποιούμε. Τα όρια μας θέλουμε να είναι τύπου κύβου όχι τόσο για τα πάνω- κάτω όρια μιας και δεν έχουμε κάθετη κίνηση στο παιχνίδι μας. Αλλά μας βοηθάνε τα αριστερά-δεξιά και μπρος- πίσω όρια που μας δίνει ο κύβος.



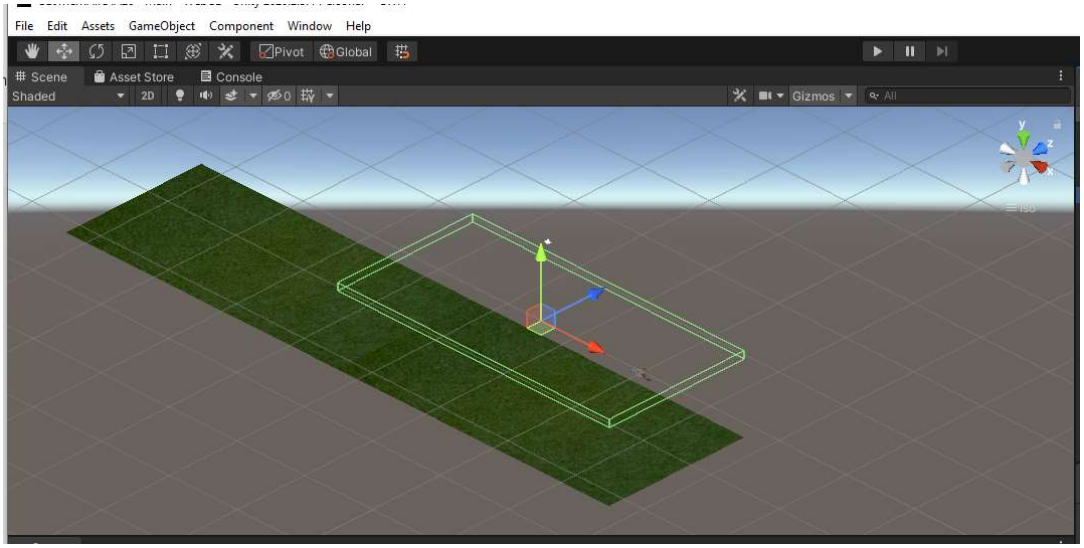
Καθορίζω το transform και το προθέτω ένα box collider.



Φτιάχνουμε ένα μεγάλο «κουτί» γύρο από τον παίχτη μας για να θέσουμε τα όρια του παιχνιδιού.



Το κουτί του Boundary πρέπει να καλύπτει ή να «αγκαλιάζει» όλη την επιφάνεια του παιχνιδιού.



Ύστερα πρέπει να ορίσουμε τις συγκρούσεις πάνω σε αυτό το πλαίσιο. Θέλουμε ότι πέφτει πάνω σε αυτό το πλαίσιο να καταστρέφεται. Αυτό γίνεται με την εισαγωγή ενός μικρού κομματιού κώδικα που φαίνεται παρακάτω.

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

public class DestroyByBoundary : MonoBehaviour
{
   void OnTriggerExit(Collider other)
   {
      Destroy(other.gameObject);
   }
}
```

Έτσι λοιπόν έχοντας ορίσει και το ανάλογο Box Collider, ότι συγκρουστεί με το πλέγμα του αντικειμένου Boundary καταστρέφεται ακαριαία.

Κατά την δοκιμή του παιχνιδιού παρατηρήσαμε ότι αφού τοποθετήσαμε το Boundary, ο παίχτης μας κινείται κανονικά μέσα σε αυτό. Αλλά κάποια από τα αντικείμενα μας όπως πέτρες, καρφιά, κούτσουρα, φύλλα και μπαταρίες ακουμπούσαν σε αυτό πρόωρα και εξαφανιζόντουσαν από το παιχνίδι πριν εκτελέσουν το έργο τους. Για το λόγο αυτό πήγαμε στο στοιχείο Rigidbody όλων αυτών των αντικειμένων και κλειδώσαμε την κίνηση τους στους άξονες Y και Z.

Βγαίνοντας για περιποίηση στον κήπου.

Στο σημείο αυτό θα πάμε να προσθέσουμε όλα τα αντικείμενα που έχουμε φτιάξει στο παιχνίδι μας έτσι ώστε να αρχίσει να σχηματίζεται κάποιο αποτέλεσμα. Το πρώτο πράγμα που παρατηρούμε είναι ότι όλα τα αντικείμενα κινούνται προς το παίχτη μας αλλά δεν αλληλοεπιδρούν όταν έρχονται σε επαφή μαζί του. Αυτό συμβαίνει διότι δεν έχουμε πει ακόμη στην unity τι να κάνει όταν θα συγκρούονται τα αντικείμενα με τον παίχτη μας.

Οι κατηγορίες αλληλεπιδράσεων χωρίζονται σε τρεις:

1) Αλληλεπίδραση μεταξύ παίχτη και πέτρα ή καρφί. Τέλος παιχνιδιού.

   Την αλληλεπίδραση αυτή την ορίζουμε με ένα κομμάτι κώδικα C# το οποίο προσαρτούμε στα αντικείμενα πέτρα και καρφί. Κοινώς αν ο παίχτης μας πέσει πάνω σε πέτρα ή σε καρφί αυτομάτως πεθαίνει και τελειώνει το παιχνίδι.

Ο κώδικας που θα χρησιμοποιήσουμε για αυτή την αλληλεπίδραση είναι ο παρακάτω:

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

public class DestroyByContact : MonoBehaviour
{
    public GameObject explosion;
    public GameObject playerExplosion;
    private GameController gameController;

    void Start()
    {
        GameObject gameControllerObject = GameObject.FindGameObjectWithTag("GameController");
        if (gameControllerObject != null)
        {
            gameController = gameControllerObject.GetComponent<GameController>();
        }
        if (gameController == null)
        {
            Debug.Log("Cannot find 'GameController' script");
        }
    }

    void OnTriggerEnter(Collider other)
    {
        if (other.tag == "Boundary")
        {
            return;
        }
        Instantiate(explosion, transform.position, transform.rotation);
        if (other.tag == "Player")
        {
            Instantiate(playerExplosion, other.transform.position, other.transform.rotation);
```

```
            gameController.GameOver();

        }
        Destroy(other.gameObject);
        Destroy(gameObject);
    }
}
```

2) Αλληλεπίδραση μεταξύ παίχτη και κούτσουρο ή φύλλο. Αύξηση ή μείωση πόντων.

Την αλληλεπίδραση αυτή την ορίζουμε με ένα κομμάτι κώδικα C# το οποίο προσαρτούμε στα αντικείμενα κούτσουρο και φύλλο. Κοινώς αν ο παίχτης μας πέσει πάνω σε κούτσουρο ή σε φύλλο θέλουμε αυτό το στοιχείο να καταστρέφεται και να προσθέτει ή να αφαιρεί πόντους από τον παίχτη ανάλογα με τις ρυθμίσεις μας.

Ο κώδικας που θα χρησιμοποιήσουμε για αυτή την αλληλεπίδραση είναι ο παρακάτω:

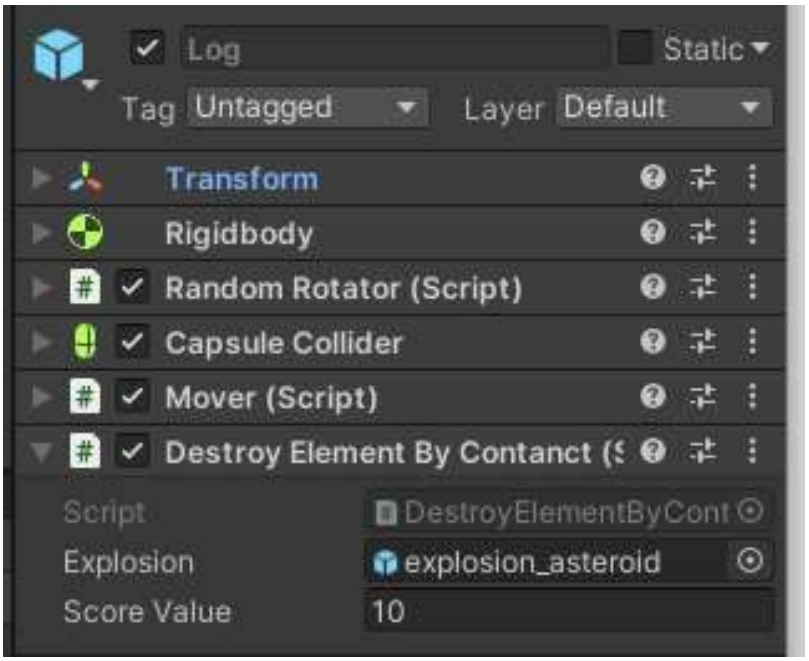
```
using System.Collections;
using System.Collections.Generic;
using UnityEngine;

public class DestroyElementByContanct : MonoBehaviour
{
    public GameObject explosion;
    public int scoreValue;
    private GameController gameController;

    void Start()
    {
        GameObject gameControllerObject = GameObject.FindGameObjectWithTag("GameController");
        if (gameControllerObject != null)
        {
            gameController = gameControllerObject.GetComponent<GameController>();
        }
        if (gameController == null)
        {
            Debug.Log("Cannot find 'GameController' script");
        }
    }


    void OnTriggerEnter(Collider other)
    {
        if (other.tag == "Boundary")
        {
            return;
```

```
            }
            Instantiate(explosion, transform.position, transform.rotation);

            gameController.AddScore(scoreValue);
            Destroy(gameObject);
        }
    }
```

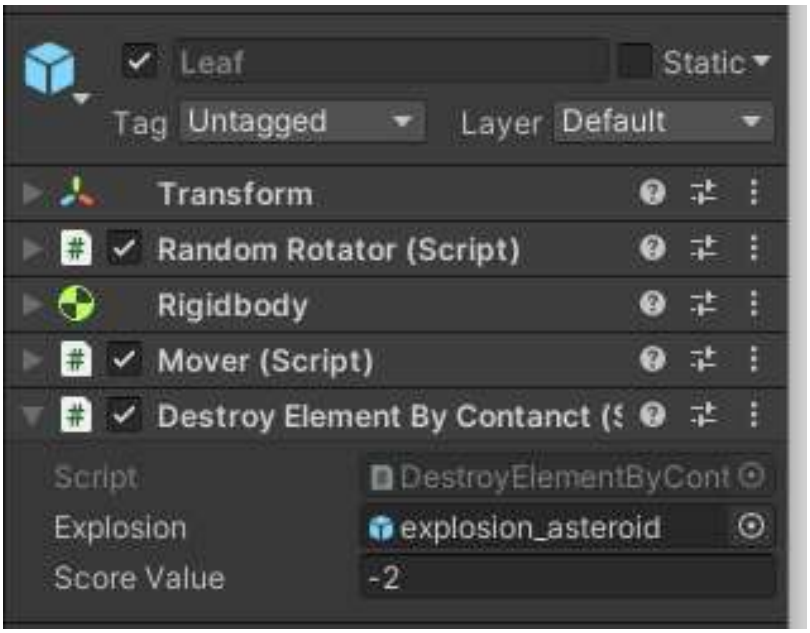
Θα παρατηρήσατε ότι μεταξύ του κώδικα DestroyByContact και του κώδικα DestroyElementByContact υπάρχει και μια ειδοποιός διαφορά. Αυτή είναι η μεταβλητή Score Value που υπάρχει στο DestroyElementByContact με την οποία θα ορίζουμε του βαθμούς που θα κερδίζει ο παίχτης μας από κάθε τέτοια σύγκρουση.

```
    public GameObject explosion;
        public int scoreValue;
        private GameController gameController;
```

Στην περίπτωση μας ο παίχτης θα κερδίζει 10 βαθμούς κάθε φορά που χτυπά ένα κούτσουρο.



Και θα χάνει 2 βαθμούς κάθε φορά που θα χτυπά ένα φύλλο.



Τον τρόπο με τον οποίο θα προστίθενται οι βαθμοί αυτοί στο γενικό Score του παίχτη θα γίνει με την ανάλογη διαμόρφωση που θα γίνει στον κώδικα GameController που θα αναλύσουμε λεπτομερώς αργότερα.

3) Αλληλεπίδραση μεταξύ παίχτη και μπαταρίας. Πλήρωση της μπάρας ενέργειας του παίχτη.

   Την αλληλεπίδραση αυτή την ορίζουμε με ένα κομμάτι κώδικα C# το οποίο προσαρτούμε στο αντικείμενο μπαταρία. Κοινώς αν ο παίχτης μας πέσει πάνω σε μια μπαταρία, θέλουμε αυτό το στοιχείο να καταστρέφεται και να γεμίζει η μπάρα ενέργειας του παίχτη που θα κατασκευάσουμε στην συνέχεια.

Ο κώδικας που θα χρησιμοποιήσουμε για αυτή την αλληλεπίδραση είναι ο παρακάτω:

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

public class ImportBattery : MonoBehaviour
{
    public GameObject import;
    private GameController gameController;

    void Start()
    {
        GameObject gameControllerObject =
GameObject.FindGameObjectWithTag("GameController");
        if (gameControllerObject != null)
        {
            gameController = gameControllerObject.GetComponent<GameController>();
        }
        if (gameController == null)
        {
            Debug.Log("Cannot find 'GameController' script");
        }
    }

    void OnTriggerEnter(Collider other)
    {
        if (other.tag == "Boundary")
        {
            return;
        }
        Instantiate(import, transform.position, transform.rotation);
        gameController.Charge = 100;
        Destroy(gameObject);
    }
}
```

Θα παρατηρήσουμε ότι σε όλους τους παραπάνω κώδικες υπάρχει το κομμάτι:

```
void Start()
   {
      GameObject gameControllerObject =
GameObject.FindGameObjectWithTag("GameController");
      if (gameControllerObject != null)
      {
         gameController = gameControllerObject.GetComponent<GameController>();
      }
      if (gameController == null)
      {
         Debug.Log("Cannot find 'GameController' script");
      }
   }
```

Αυτό μας εξασφαλίζει την επικοινωνία με τον κώδικα του ελέγχου του παιχνιδιού GameController έτσι ώστε να δίνεται η πληροφορία για:

- πότε καταστράφηκε το αλυσοπρίονο μας κατά μια σύγκρουση και πρέπει να έχουμε Game Over.
- πότε πιάσαμε μια μπαταρία και πρέπει να γεμίσει η μπάρα ενέργειας.
- πότε χτυπήσαμε ένα κούτσουρο ή ένα φύλο για να αλλάξει το σκορ μας.

Τον κώδικα του ελέγχου του παιχνιδιού GameController θα τον αναλύσουμε λεπτομερώς σε επόμενο κεφάλαιο της εργασίας μας και θα δούμε πώς η κάθε πληροφορία που λαμβάνει από τους επιμέρους κώδικες διαμορφώνουν την συμπεριφορά του παιχνιδιού.

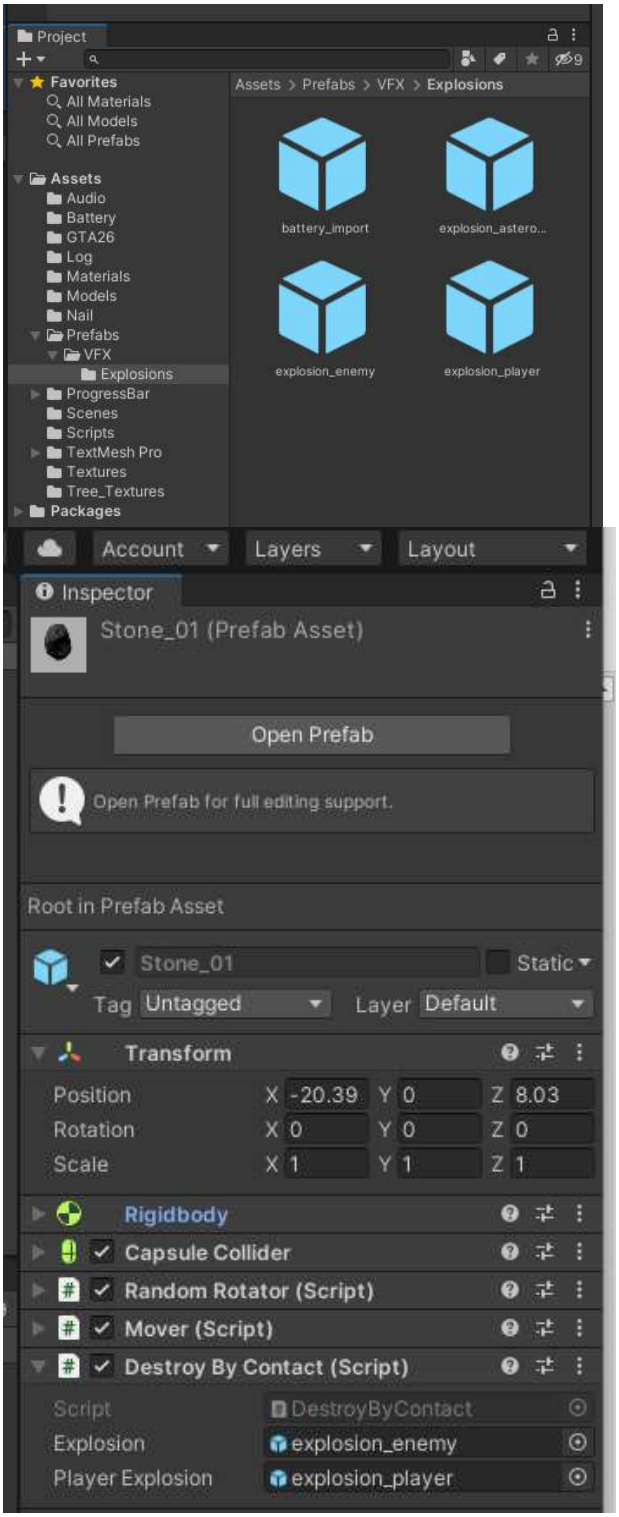
Οι Πρώτες Εκρήξεις.

Για να κάνουμε το παιχνίδι μας πιο εντυπωσιακό προσθέτουμε μερικά εφέ δράσης. Θα βάλουμε ένα εφέ έκρηξης όταν αντικείμενα συγκρούονται και καταστρέφονται ή όταν το αλυσοπρίονο μας καταστρέφει ένα κούτσουρο ή ένα φύλλο.

Στον παρακάτω κώδικα του DestroyByContact υπεύθυνα για αυτά τα εφέ είναι τα τονισμένα σημεία.

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

public class DestroyByContact : MonoBehaviour
{
    public GameObject explosion;
    public GameObject playerExplosion;
    private GameController gameController;

    void Start()
    {
        GameObject gameControllerObject = GameObject.FindGameObjectWithTag("GameController");
        if (gameControllerObject != null)
        {
            gameController = gameControllerObject.GetComponent<GameController>();
        }
        if (gameController == null)
        {
            Debug.Log("Cannot find 'GameController' script");
        }
    }

    void OnTriggerEnter(Collider other)
    {
        if (other.tag == "Boundary")
        {
            return;
        }
        Instantiate(explosion, transform.position, transform.rotation);
        if (other.tag == "Player")
        {
            Instantiate(playerExplosion, other.transform.position, other.transform.rotation);
            gameController.GameOver();
        }
        Destroy(other.gameObject);
        Destroy(gameObject);
    }
}
```

Όπως γίνεται κατανοητό από τον κώδικα πέρα από τον παίχτη, κατά την σύγκρουση καταστρέφεται και το αντικείμενο με οποίο συγκρούεται ο παίχτης. Κατά την διάρκεια αυτών των καταστροφών έχουμε προσθέσει και ένα εφέ έκρηξης που βρήκαμε στο asset store της unity.

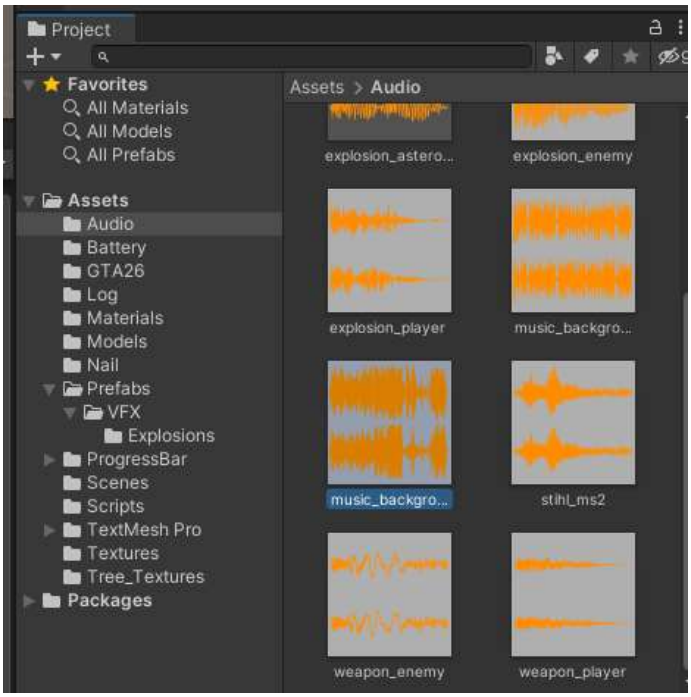


Με αυτές τις προσθήκες στον κώδικα υπαγορεύουμε στην unity να ξεκινήσει το εφέ explosion όταν το αντικείμενο συγκρουστεί με κάτι και το εφέ να γίνει ακριβώς στο σημείο της σύγκρουσης.

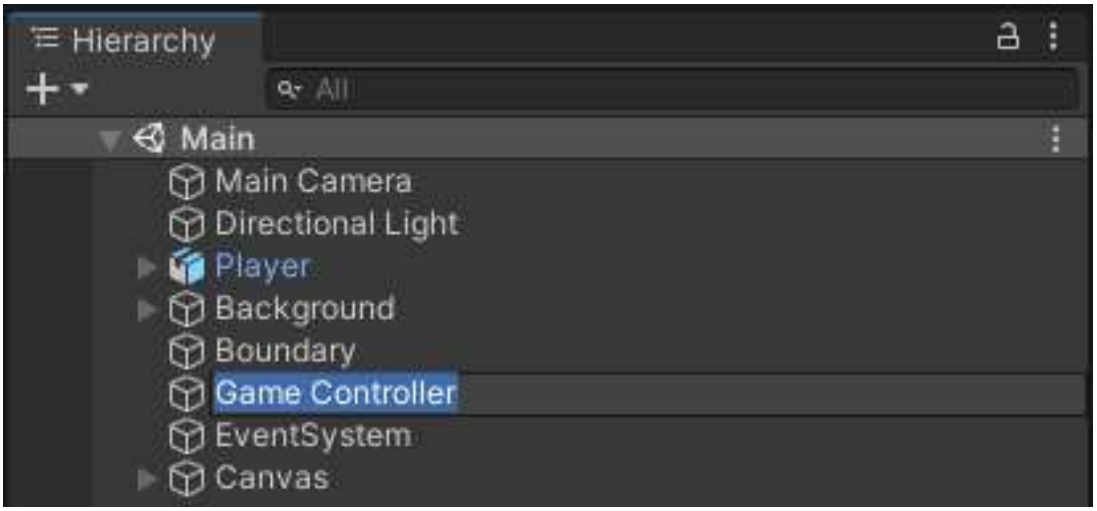
Ηχητική Κάλυψη.

Στο στάδιο αυτό αν προσπαθήσουμε να παίξουμε το παιχνίδι μας θα είναι άμεσα αισθητή η έλλειψη των ηχητικών μερών του παιχνιδιού. Η έλλειψη δηλαδή μουσικής την ώρα που παίζουμε και η έλλειψη των ηχητικών εφέ την ώρα που γίνονται η εκρήξεις ή την ώρα που πιάνουμε μια μπαταρία. Για να διανθίσουμε λοιπόν το παιχνίδι μας θα χρησιμοποιήσουμε τα ηχητικά assets που θα βρούμε στο διαδίκτυο.

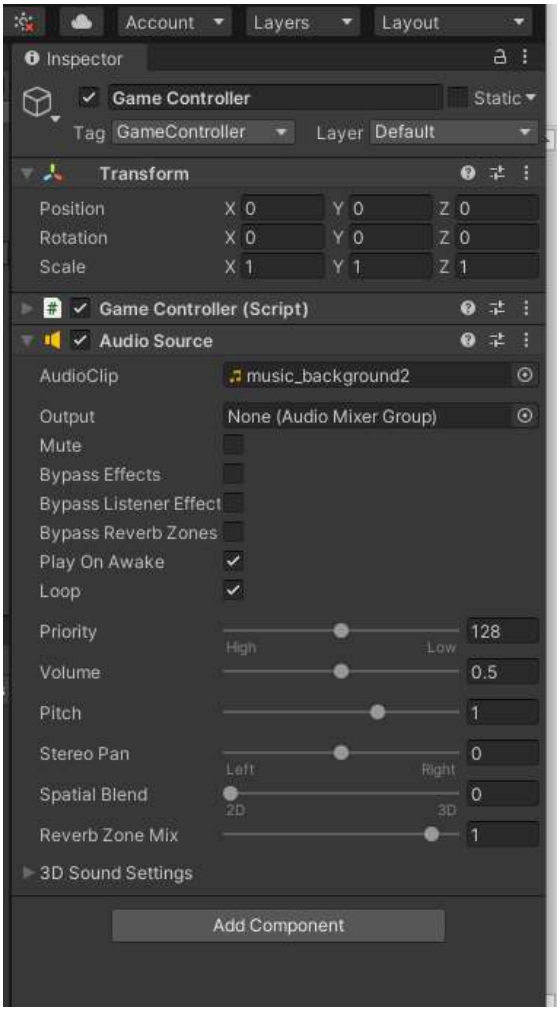
Θα βρούμε ήχους και μουσική ελεύθερης άδειας για να εμπλουτίσουμε το παιχνίδι μας. Αρχικά βρήκαμε ένα μουσικό κομμάτι για να παίζει σαν μουσικό χαλί καθ' όλη την διάρκεια του παιχνιδιού μας. Το εισάγουμε στη unity.



Ύστερα πάμε και δημιουργούμε ένα κενό 3D αντικείμενο με το όνομα Game Controller. Αυτό μαζί με το Player Controller θα είναι τα βασικότερα scripts του παιχνιδιού μας.



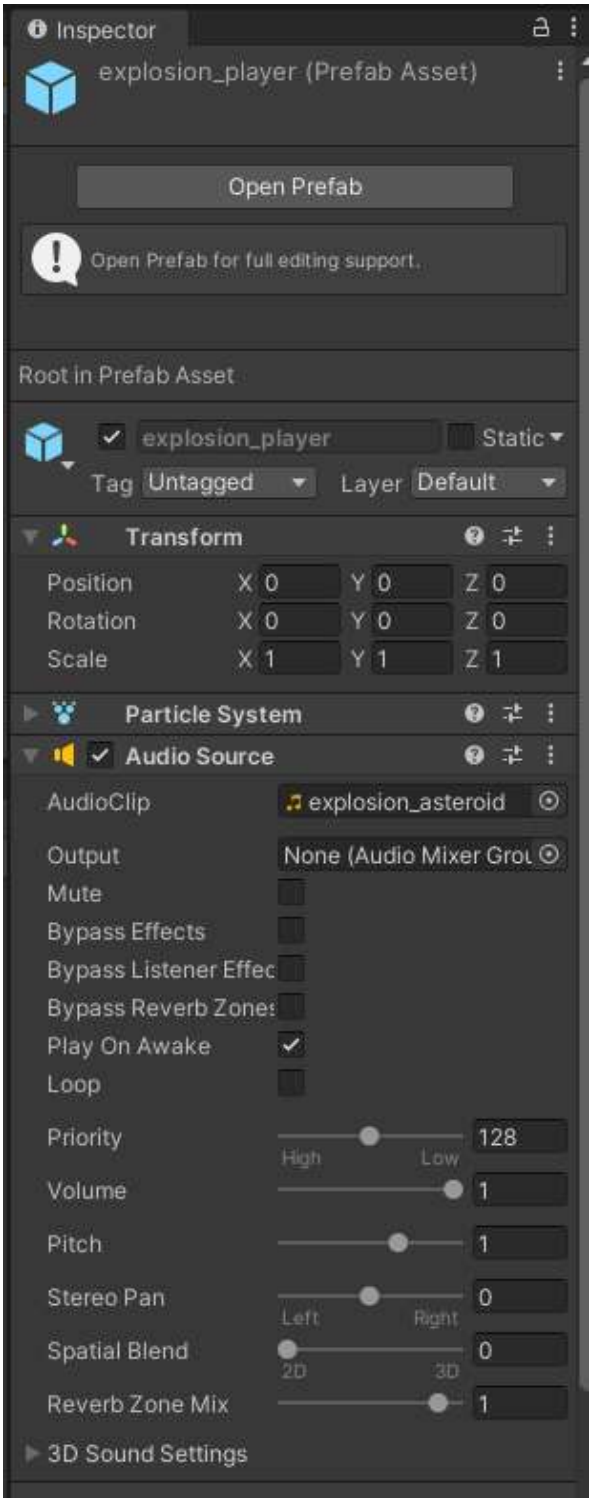
Τώρα πάμε και συσχετίζουμε το ηχητικό κομμάτι που ετοιμάσαμε με το αντικείμενο Game Controller έτσι ώστε να παίζει κατά την διάρκεια του παιχνιδιού.



Με τις επιλογές Play On Awake κα Loop υπαγορεύουμε ουσιαστικά να ξεκινά να παίζει το κομμάτι με το που ξεκινά το παιχνίδι, και να παίζει σε επανάληψη μετά το τέλος του. Έτσι θα παίζουμε με την συντροφιά μιας ωραίας μουσικής.

Επίσης σημαντικό είναι το να διανθίσουμε με ηχητικά εφέ κάθε αλληλεπίδραση που θα λαμβάνει χώρα στο παιχνίδι μας. Βρήκαμε από το διαδίκτυο έναν ήχο αλυσοπρίονου. Τον βάλαμε στην unity και τον συσχετίσαμε με το εφέ που ενεργοποιείται όταν ο παίχτης μας συγκρούεται με ένα κούτσουρο.

Το ίδιο ισχύει και για τα ηχητικά εφέ που προσθέσαμε στα γραφικά εφέ των εκρήξεων κατά τις συγκρούσεις μεταξύ του παίχτη μας και μιας πέτρας ή ενός καρφιού.



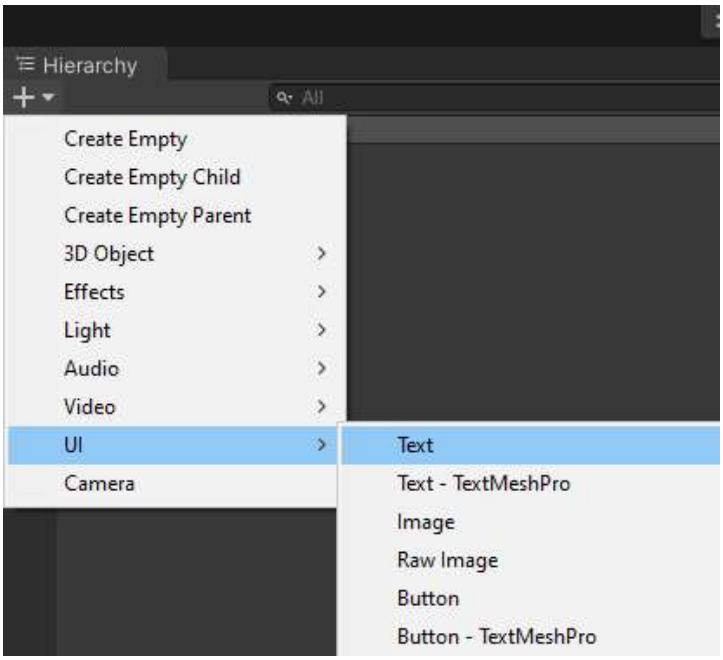
Πολύ σημαντικό είναι σε αυτή την περίπτωση να είναι ενεργοποιημένη η επιλογή Play On Awake. Με αυτό τον τρόπο το ηχητικό εφέ της έκρηξης θα ακουστεί ακριβώς την στιγμή που θα γίνεται η έκρηξη.

Score, Charge, Game Over, Restart

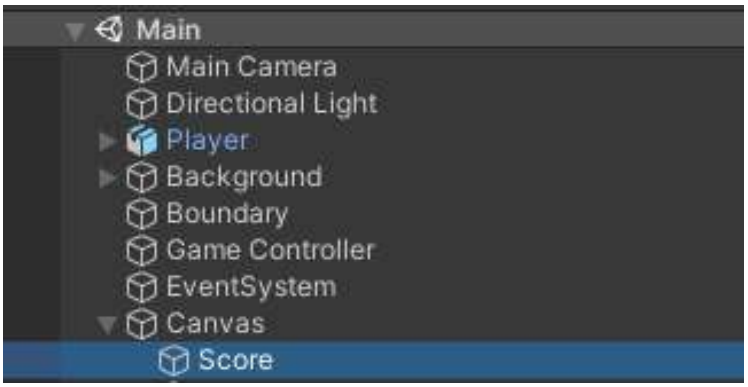
Στο σημείο αυτό θα φτιάξουμε όλα τα πεδία κειμένων και πληροφοριών του παιχνιδιού μας. Αυτά θα είναι τα εξής:

1) το πεδίο της βαθμολογίας του παίχτη (Score)
2) το πεδίο επανεκκίνησης του παιχνιδιού (Restart)
3) το πεδίο τέλους του παιχνιδιού (Game Over)
4) η μπάρα ενέργειας της μπαταρίας του παίχτη (Battery Life)

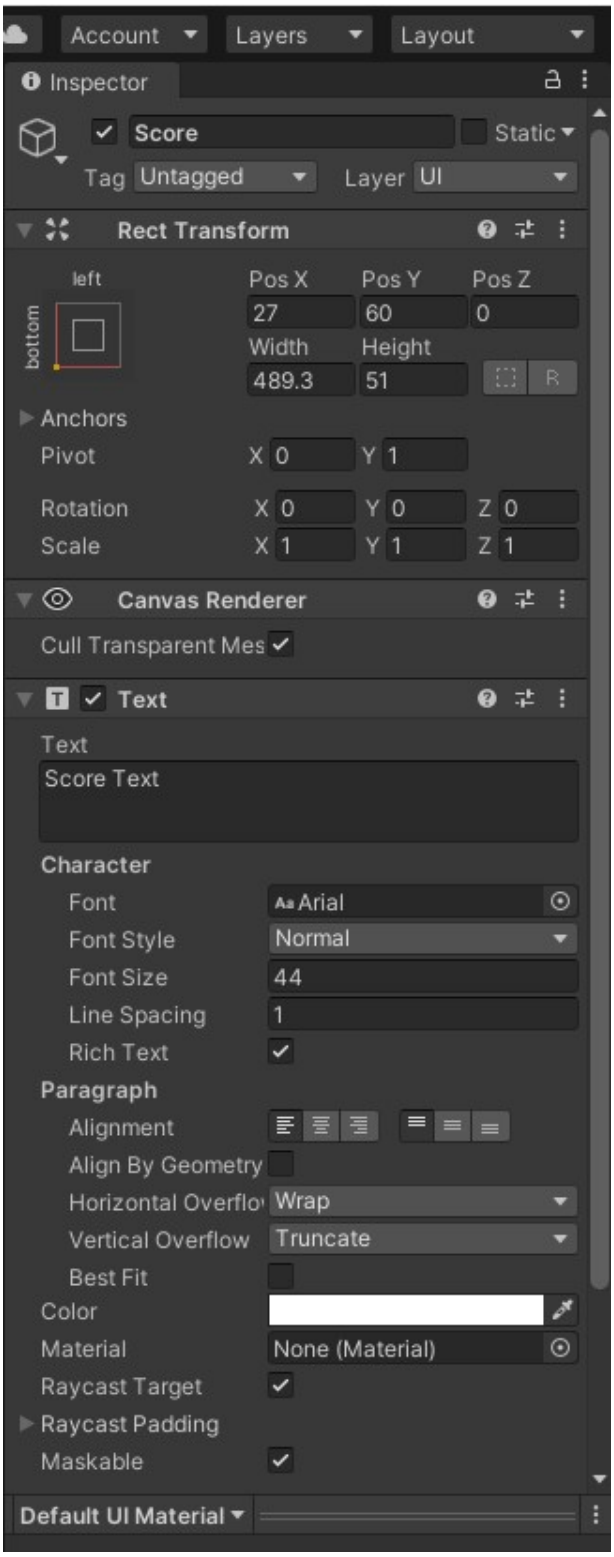
Παιχνίδι όμως χωρίς Score και πόντους δεν γίνεται. Σε αυτό το βήμα λοιπόν θα προσθέσουμε ένα σύστημα βαθμολογίας με το οποίο θα μετριέται η επίδοση του παίχτη μας. Για να το κάνουμε αυτό θα πάμε αρχικά και θα προσθέσουμε ένα νέο αντικείμενο στο παιχνίδι μας τύπου UI Text στο παιχνίδι μας.



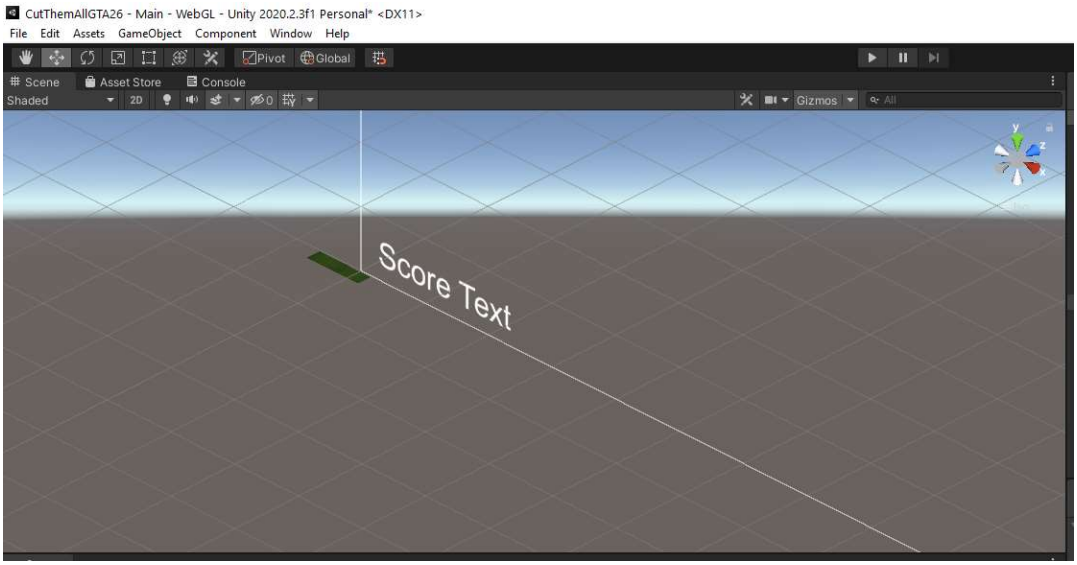
Αυτό θα δημιουργήσει ένα Canvas που θα πάρει το UI Text μας σαν υπο-αντικείμενο. Το ονομάζουμε Score.



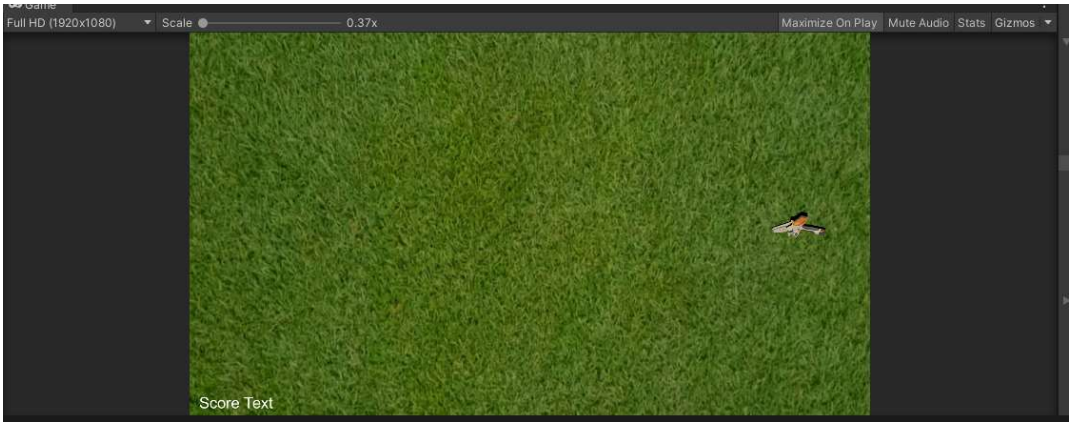
Στο πεδίο Score βάζουμε το λεκτικό Score Text και το τοποθετούμε στην κάτω αριστερή γωνία του παιχνιδιού μας με τις παρακάτω ρυθμίσεις.



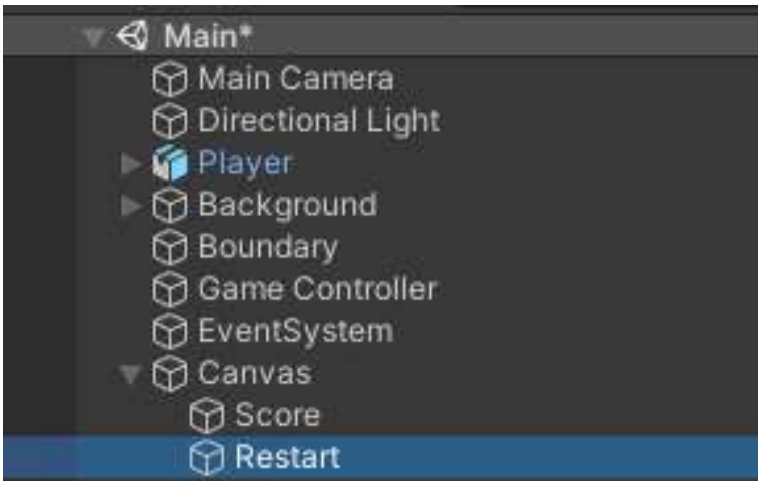
Καθ' όλη την διάρκεια των ρυθμίσεων πρέπει να παρατηρούμε το Scene View



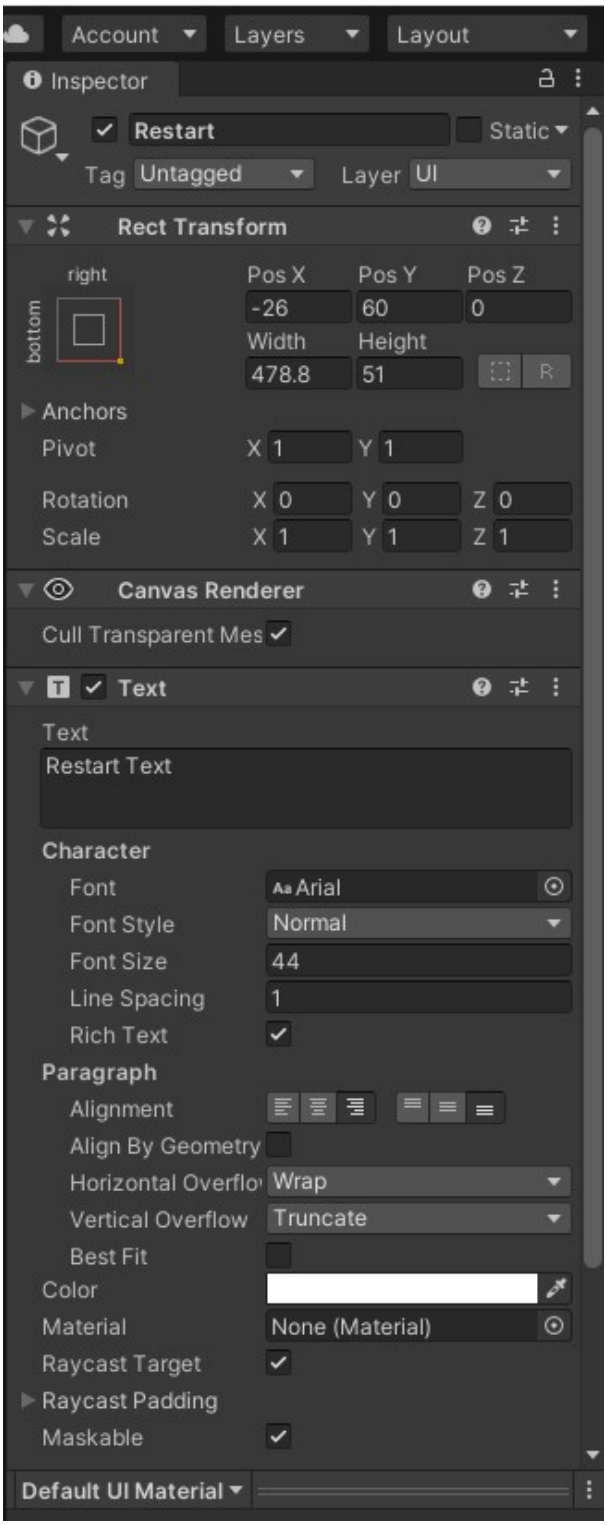
Καθώς και το Game View για να βλέπουμε πως θα είναι το τελικό μας αποτέλεσμα στο παιχνίδι.



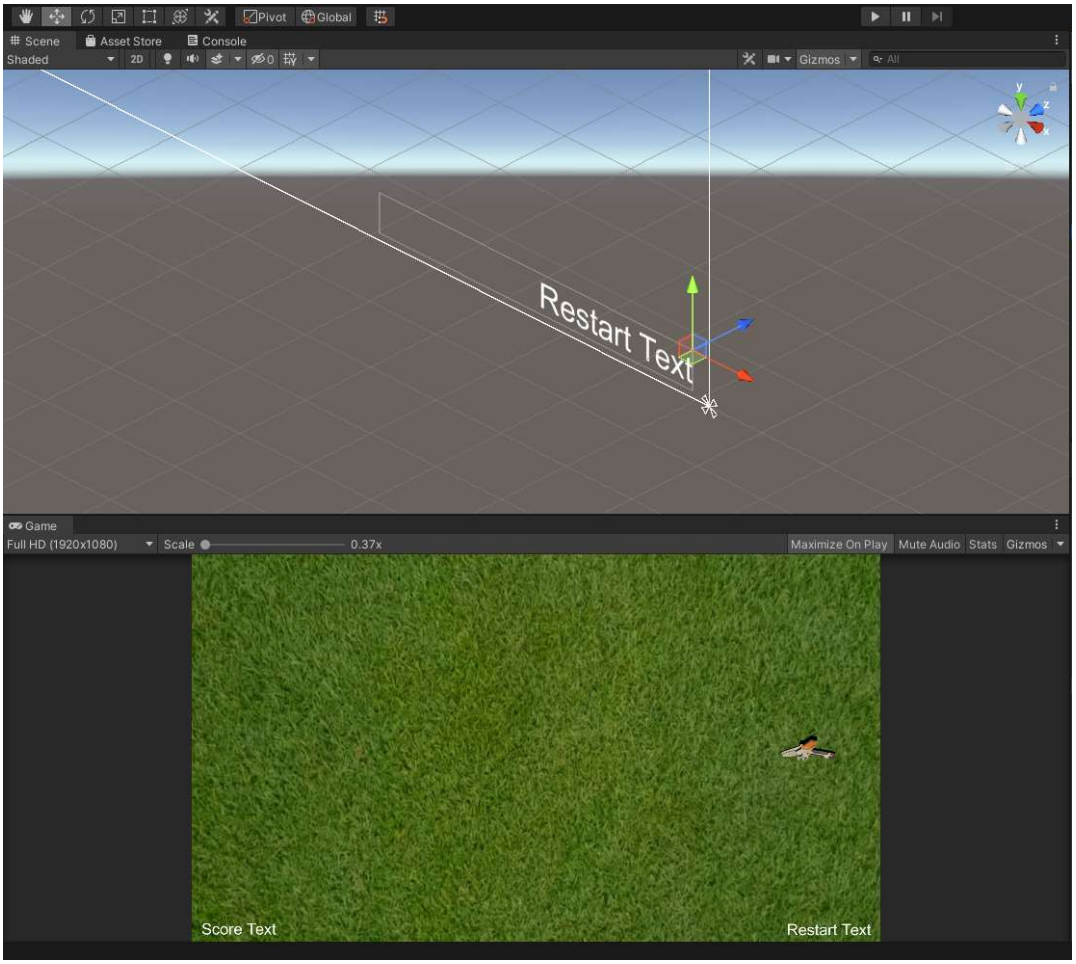
Με την ίδια λογική και το ίδιο τρόπο θα προσθέσουμε 2 ακόμη UI Texts στο ήδη υπάρχων Canvas. Το πρώτο θα είναι για το Restart.



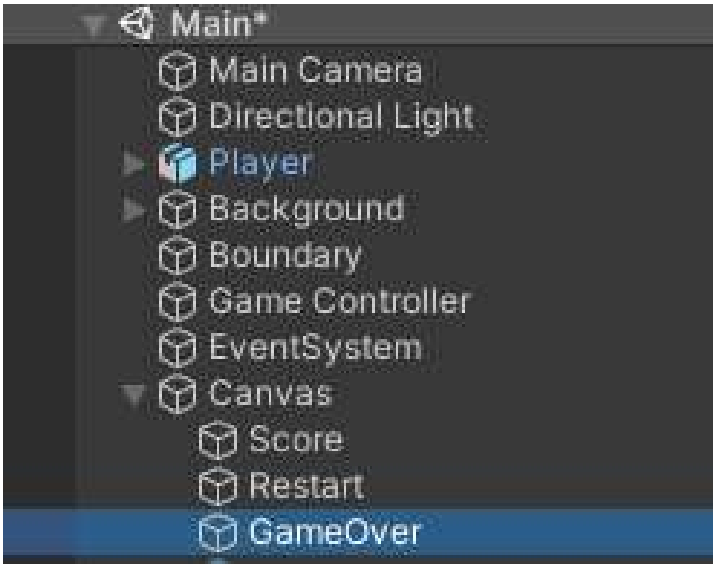
Στο πεδίο Restart βάζουμε το λεκτικό Restart Text και το τοποθετούμε στην κάτω δεξιά γωνία του παιχνιδιού μας με τις παρακάτω ρυθμίσεις.



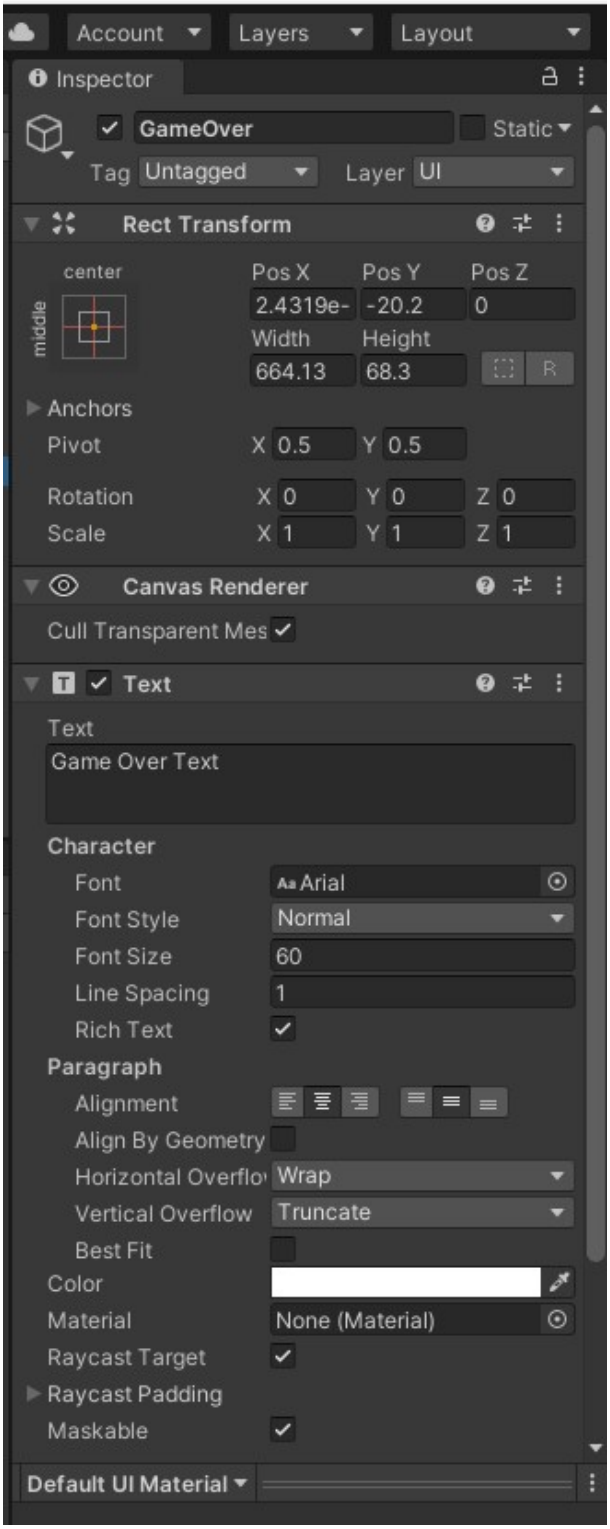
Το τελικό αποτέλεσμα πρέπει να δείχνει όπως φαίνεται στην παρακάτω εικόνα.



Τέλος θα προσθέσουμε το τρίτο και τελευταίο πεδίο κειμένου που είναι το «τέλος παιχνιδιού». Προσθέτουμε ένα ακόμη UI Text.



Στο πεδίο Game Over βάζουμε το λεκτικό Game Over Text και το τοποθετούμε στο κέντρο του παιχνιδιού μας με τις παρακάτω ρυθμίσεις. Επίσης του δίνουμε και ελαφρά μεγαλύτερο μέγεθος γραμμάτων σε σχέση με τα άλλα πεδία που φτιάξαμε.

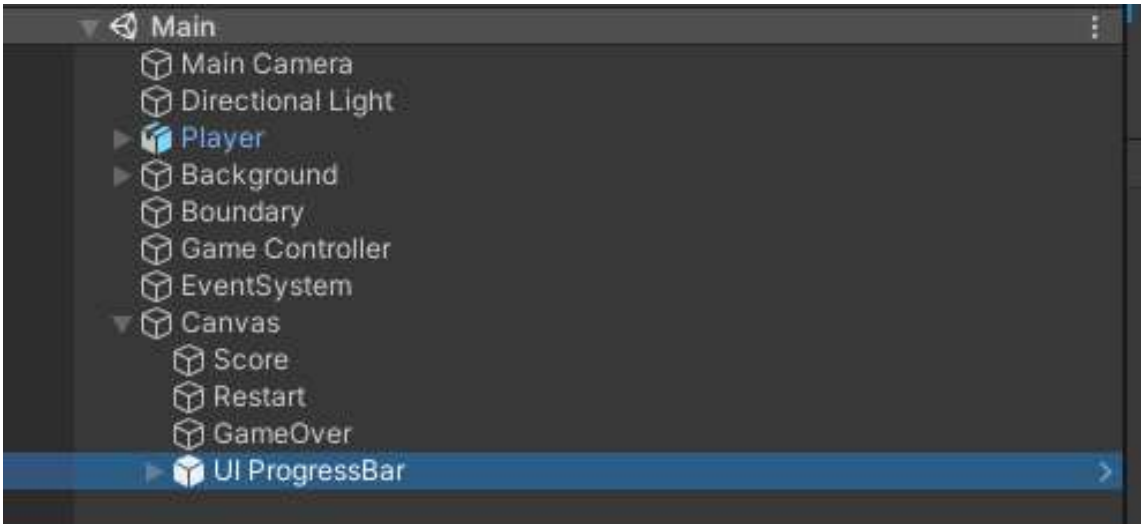


Το τελικό αποτέλεσμα πρέπει να δείχνει όπως φαίνεται στην παρακάτω εικόνα.

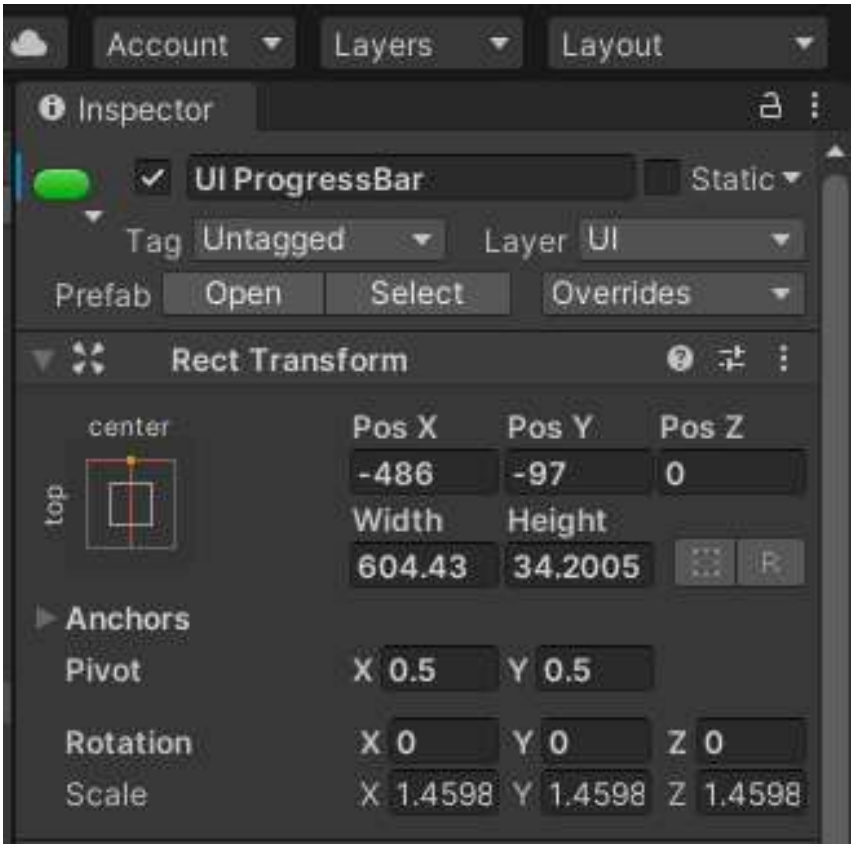
Τώρα έχουμε δομήσει όλα τα λεκτικά πεδία του παιχνιδιού μας και τα έχουμε τοποθετήσει στο παιχνίδι μας. Μας λείπει όμως ένα στοιχείο του παιχνιδιού που έχει και λεκτικό κομμάτι αλλά έχει κι έναν ιδιαίτερο ρόλο στο παιχνίδι μιας εκτός του ότι δίνει την πληροφορία της φόρτισης της μπαταρίας στον παίχτη μας, θα έχει την ιδιότητα να τερματίσει το παιχνίδι αν αυτή φτάσει στο τέλος της, δηλαδή στην τιμή 0.

Για τις ανάγκες αυτού του αντικειμένου θα χρησιμοποιήσουμε ένα progress bar που θα βρούμε σαν βάση από το asset store της unity και θα το παραμετροποιησούμε σύμφωνα με τις ανάγκες του παιχνιδιού μας για να γίνει το Battery Bar που θα μας πληροφορεί για το επίπεδο φόρτησης της μπαταρίας του αλυσοπριόνου μας. Το μοντέλο του αλυσοπρίονου που έχει το ρόλο του παίχτη, είναι ένα μηχάνημα μπαταρίας. Για αυτό το λόγο φτιάξαμε το αντικείμενο μπαταρία που θα μπορεί να συλλέγει ο παίχτης μας για να ανανεώνει την ενέργεια του και για αυτό το λόγο θα φτιάξουμε και την Battery Bar για να το πληροφορούμε για το επίπεδο φόρτισης της μπαταρίας του.

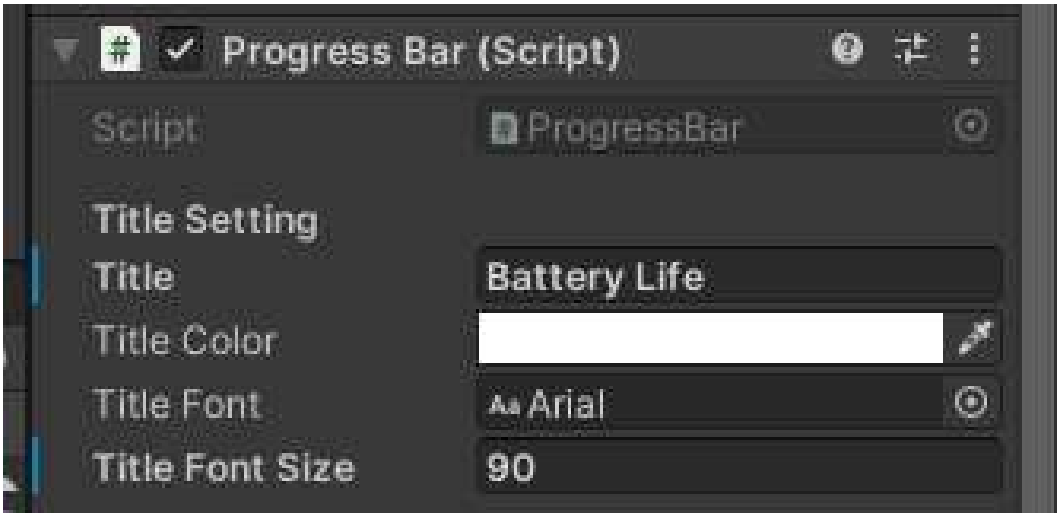
Για την προσθήκη του νέου αντικειμένου θα εισάγωουμε το αντικείμενο αρχικά στην unity. Στην συνέχεια θα το προσθέσουμε σαν υπο-αντικείμενο στον ήδη υπάρχον Canvasμιας και αποτελεί ένα UI αντικείμενο.



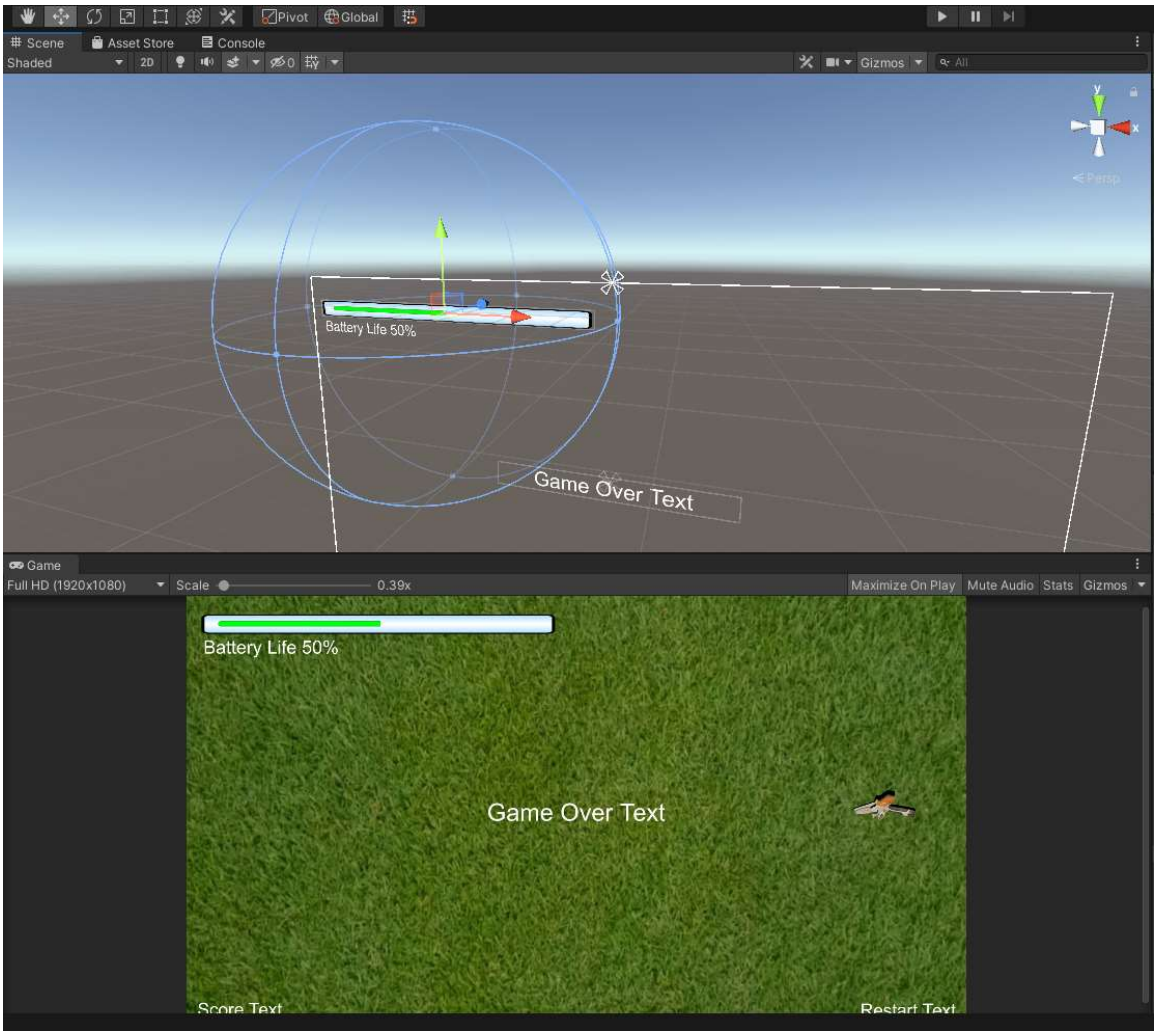
Στην συνέχεια θα δώσουμε στην νέα μπάρα τα χαρακτηριστικά που θέλουμε. Αρχικά την τοποθετούμε πάνω αριστερά στο παιχνίδι μας.



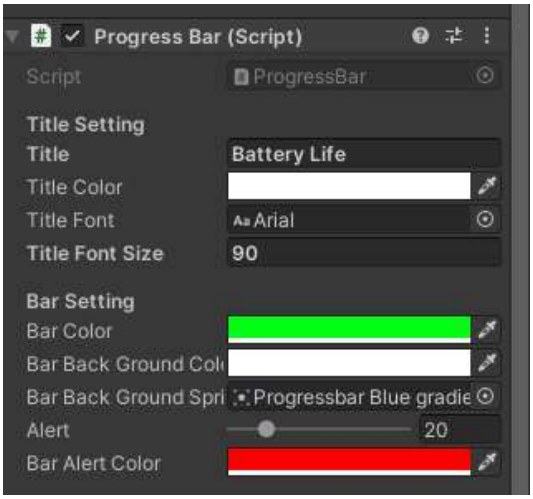
Δίνουμε τον τίτλο Battery Life



Το τελικό αποτέλεσμα πρέπει να δείχνει όπως φαίνεται στην παρακάτω εικόνα.



Πέρα όμως από την τοποθέτηση της μπάρας καθορίζουμε ότι το κύριο χρώμα της μπάρας θα είναι πράσινο, αλλά καθώς θα μειώνεται, μόλις φτάσει στο 20% θα γίνεται κόκκινη.



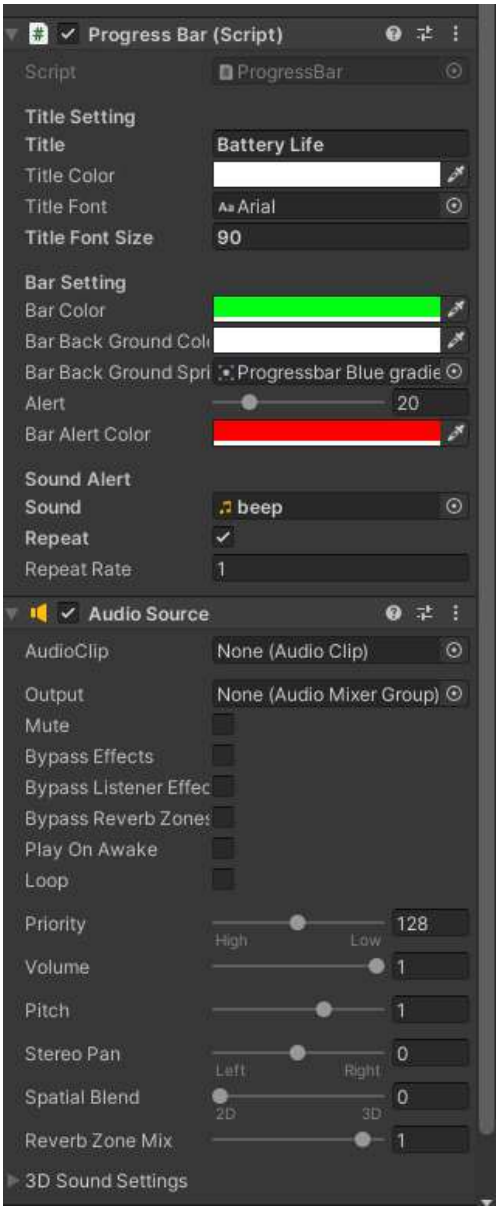
Για την λειτουργία της μπάρας που περιγράφουμε, κάναμε χρήση του παρακάτω κώδικα C#.

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


[ExecuteInEditMode]

public class ProgressBar : MonoBehaviour
{

    [Header("Title Setting")]
    public string Title;
    public Color TitleColor;
    public Font TitleFont;
    public int TitleFontSize = 10;

    [Header("Bar Setting")]
    public Color BarColor;
    public Color BarBackGroundColor;
    public Sprite BarBackGroundSprite;
    [Range(1f, 100f)]
    public int Alert = 20;
    public Color BarAlertColor;

    [Header("Sound Alert")]
    public AudioClip sound;
    public bool repeat = false;
    public float RepeatRate = 1f;
```

```
    private Image bar, barBackground;
    private float nextPlay;
    private AudioSource audiosource;
    private Text txtTitle;
    private float barValue;
    public float BarValue
    {
        get { return barValue; }

        set
        {
            value = Mathf.Clamp(value, 0, 100);
            barValue = value;
            UpdateValue(barValue);

        }
    }



    private void Awake()
    {
        bar = transform.Find("Bar").GetComponent<Image>();
        barBackground = GetComponent<Image>();
        txtTitle = transform.Find("Text").GetComponent<Text>();
        barBackground = transform.Find("BarBackground").GetComponent<Image>();
        audiosource = GetComponent<AudioSource>();
    }

    private void Start()
    {
        txtTitle.text = Title;
        txtTitle.color = TitleColor;
        txtTitle.font = TitleFont;
        txtTitle.fontSize = TitleFontSize;

        bar.color = BarColor;
        barBackground.color = BarBackGroundColor;
        barBackground.sprite = BarBackGroundSprite;

        UpdateValue(barValue);


    }
```

```
    void UpdateValue(float val)
    {
        bar.fillAmount = val / 100;
        txtTitle.text = Title + " " + val + "%";

        if (Alert >= val)
        {
            bar.color = BarAlertColor;
        }
        else
        {
            bar.color = BarColor;
        }

    }


    private void Update()
    {
        if (!Application.isPlaying)
        {
            UpdateValue(50);
            txtTitle.color = TitleColor;
            txtTitle.font = TitleFont;
            txtTitle.fontSize = TitleFontSize;

            bar.color = BarColor;
            barBackground.color = BarBackGroundColor;

            barBackground.sprite = BarBackGroundSprite;
        }
        else
        {
            if (Alert >= barValue && Time.time > nextPlay)
            {
                nextPlay = Time.time + RepeatRate;
                audiosource.PlayOneShot(sound);
            }
        }
    }

}
```

Όπως φαίνεται κι από τον κώδικα παραπάνω, προσθέσαμε κι ένα ύφος συναγερμού στην μπάρα μας για να προειδοποιούμε τον παίχτη μας. Αυτό επαναλαμβάνεται κάθε 1% από το 20% μέχρι το 0% της μπάρας.



Τέλος εισαγάγαμε κι ένα ήχο συναγερμού τύπου «beep» και τον συσχετίσαμε με το αντικείμενο της μπάρας μας έτσι ώστε να ηχεί και να προειδοποιεί τον παίχτη μας. Τώρα όμως πρέπει να κάνουμε την μπάρα μας να ξεκινά μαζί με το παιχνίδι και μειώνεται με συγκεκριμένο ρυθμό κατά την διάρκεια του. Στην περίπτωση μας θέλουμε να μειώνεται 1% ανά δευτερόλεπτο παιχνιδιού. Έτσι ο παίχτης μας αν δεν ανανεώσει την ενέργεια του αρπάζοντας μια μπαταρία, θα έχει στην διάθεση του χρόνο παιχνιδιού 100 δευτερολέπτων μιας και μόλις η μπάρα ενέργειας του μηδενίσει θα τερματιστεί το παιχνίδι του με μια θεαματική έκρηξη.

Αυτό το καταφέρνουμε με την προσθήκη ενός μικρού κομματιού κώδικα C# που θα προσθέσουμε στο κώδικα του Game Controller ως μια ρουτίνα που θα τρέχει με την εκκίνηση του παιχνιδιού. Θα το δούμε αναλυτικά στην συνέχεια της εργασίας μας.

Ο Έλεγχος του Παιχνιδιού.

Στο σημείο αυτό έχουμε φτιάξει όλα τα δομικά μέρη του παιχνιδιού μας. Αυτό που μας απομένει είναι φτιάξουμε είναι ο έλεγχος του παιχνιδιού (Game Controller) και ο έλεγχος του παίχτη (Player Controller) που είναι ουσιαστικά 2 scripts από κώδικα C# που σε συνδυασμό, δίνουν σάρκα και οστά στο παιχνίδι μας. Τον έλεγχο του παίχτη τον έχουμε θίξει ήδη νωρίτερα στην εργασία μας, αλλά πολύ επιφανειακά. Τώρα θα τον αναλύσουμε εις βάθος έτσι ώστε να γίνει κατανοητό το κάθε κομμάτι. Παραθέτουμε αρχικά τον κώδικα του Game Controller.

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


public class GameController : MonoBehaviour
{
    public GameObject stone;
            public GameObject nail;
            public GameObject log;
            public GameObject leaf;
            public GameObject battery;
    public ProgressBar progressBar;
    public Vector3 spawnValues;
    public float startWait;
    public int stoneCount;
    public float stonespawnWait;
    public float stonewaveWait;
            public int nailCount;
    public float nailspawnWait;
    public float nailwaveWait;
            public int logCount;
    public float logspawnWait;
    public float logwaveWait;
            public int leafCount;
    public float leafspawnWait;
    public float leafwaveWait;
            public int batteryCount;
    public float batteryspawnWait;
    public float batterywaveWait;

    public Text scoreText;
    public Text restartText;
    public Text gameOverText;

    private bool gameOver;
    private bool restart;
```

```
    private int score;
    public int Charge;


    void Start ()
    {

        gameOver = false;
        restart = false;
        restartText.text = "";
        gameOverText.text = "";
        score = 0;
        Charge = 100;
        UpdateScore();
        StartCoroutine (SpawnWaves1 ());
                    StartCoroutine (SpawnWaves2 ());
                    StartCoroutine (SpawnWaves3 ());
                    StartCoroutine (SpawnWaves4 ());
                    StartCoroutine (SpawnWaves5 ());
        StartCoroutine("decreaseCharge");
    }

    void Update()
    {
        {
            progressBar.BarValue = Charge;
        }

        if (restart)
        {
            if (Input.GetKeyDown(KeyCode.R))
            {
                SceneManager.LoadScene(SceneManager.GetActiveScene().buildIndex);
            }
        }


    }

    IEnumerator SpawnWaves1 ()
    {
        yield return new WaitForSeconds (startWait);
        while (true)
        {
            for (int i = 0; i < stoneCount; i++)
            {
```

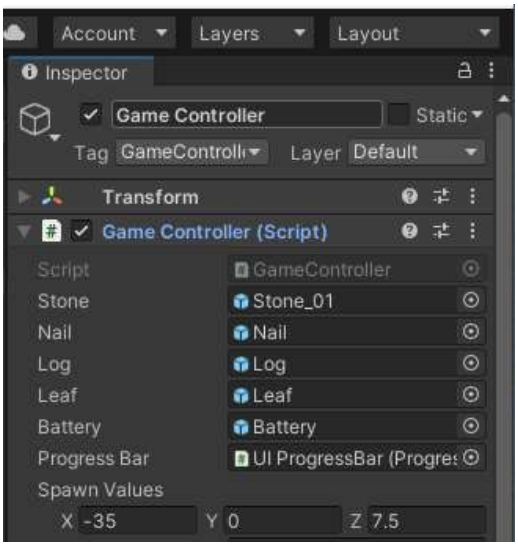
```
            Vector3 spawnPosition = new Vector3 (spawnValues.x, spawnValues.y,
Random.Range (-spawnValues.z, spawnValues.z));
            Quaternion spawnRotation = Quaternion.identity;
            Instantiate (stone, spawnPosition, spawnRotation);
            yield return new WaitForSeconds (stonespawnWait);
        }
        yield return new WaitForSeconds (stonewaveWait);

        if (gameOver)
        {
            restartText.text = "Press 'R' for Restart";
            restart = true;
            StopCoroutine("decreaseCharge");
            Charge = 100;
            break;
        }
    }
     }


     IEnumerator SpawnWaves2 ()
  {
    yield return new WaitForSeconds (startWait);
    while (true)
    {

        for (int i = 0; i < nailCount; i++)
        {
            Vector3 spawnPosition = new Vector3 (spawnValues.x, spawnValues.y,
Random.Range (-spawnValues.z, spawnValues.z));
            Quaternion spawnRotation = Quaternion.identity;
            Instantiate (nail, spawnPosition, spawnRotation);
            yield return new WaitForSeconds (nailspawnWait);
        }
        yield return new WaitForSeconds (nailwaveWait);

        if (gameOver)
        {
            restartText.text = "Press 'R' for Restart";
            restart = true;
            break;
        }
    }
     }

     IEnumerator SpawnWaves3 ()
```

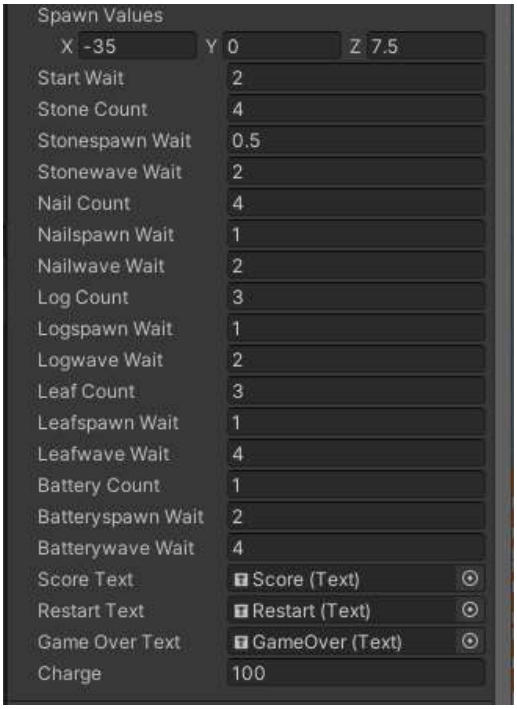
```
  {
     yield return new WaitForSeconds (startWait);
     while (true)
     {

        for (int i = 0; i < logCount; i++)
        {
           Vector3 spawnPosition = new Vector3 (spawnValues.x, spawnValues.y,
Random.Range (-spawnValues.z, spawnValues.z));
           Quaternion spawnRotation = Quaternion.identity;
           Instantiate (log, spawnPosition, spawnRotation);
           yield return new WaitForSeconds (logspawnWait);
        }
        yield return new WaitForSeconds (logwaveWait);

        if (gameOver)
        {
           restartText.text = "Press 'R' for Restart";
           restart = true;
           break;
        }
     }
         }

         IEnumerator SpawnWaves4 ()
  {
     yield return new WaitForSeconds (startWait);
     while (true)
     {

        for (int i = 0; i < leafCount; i++)
        {
           Vector3 spawnPosition = new Vector3 (spawnValues.x, spawnValues.y,
Random.Range (-spawnValues.z, spawnValues.z));
           Quaternion spawnRotation = Quaternion.identity;
           Instantiate (leaf, spawnPosition, spawnRotation);
           yield return new WaitForSeconds (leafspawnWait);
        }
        yield return new WaitForSeconds (leafwaveWait);

        if (gameOver)
        {
           restartText.text = "Press 'R' for Restart";
           restart = true;
           break;
        }
```

```
    }

        }


        IEnumerator SpawnWaves5 ()
  {
    yield return new WaitForSeconds (startWait);
    while (true)
    {

      for (int i = 0; i < batteryCount; i++)
      {
        Vector3 spawnPosition = new Vector3 (spawnValues.x, spawnValues.y,
Random.Range (-spawnValues.z, spawnValues.z));
        Quaternion spawnRotation = Quaternion.identity;
        Instantiate (battery, spawnPosition, spawnRotation);
        yield return new WaitForSeconds (batteryspawnWait);
      }
      yield return new WaitForSeconds (batterywaveWait);

      if (gameOver)
      {
        restartText.text = "Press 'R' for Restart";
        restart = true;
        break;
      }
    }
  }

  IEnumerator decreaseCharge ()
  {

      yield return new WaitForSeconds(1F);
      Charge -= 1;
      StartCoroutine("decreaseCharge");



  }

  public void AddScore(int newScoreValue)
  {
    score += newScoreValue;
    UpdateScore();
  }
```

```
    void UpdateScore()
    {
        scoreText.text = "Score: " + score;
    }

    public void GameOver()
    {
        gameOverText.text = "Game Over!";
        gameOver = true;
    }

}
```

Στο αντικείμενο Game Controller που έχουμε φτιάξει ήδη έχουμε συσχετίσει μόνο το μουσικό κομμάτι που θέλουμε να ακούγεται κατά την διάρκεια του παιχνιδιού. Τώρα που συσχετίζουμε και το παραπάνω κομμάτι κώδικα, θα εμφανιστούν οι παρακάτω μεταβλητές.





Μπορεί τώρα να φαίνονται πολλές μεταβλητές αλλά με τον τρόπο αυτό έχουμε τον απόλυτο έλεγχο του παιχνιδιού μας. Σε κάθε ένα από τα στοιχεία Stone, Nail, Log, Leaf και Battery, έχουμε ορίσει τις μεταβλητές Spawn Wait, Wave Wait και Count. Με αυτά ορίζουμε τα παρακάτω:

Spawn Wait: είναι ο ρυθμός στοιχείων που γεννάμε ανά δευτερόλεπτο κατά την διάρκεια ενός κύματος.

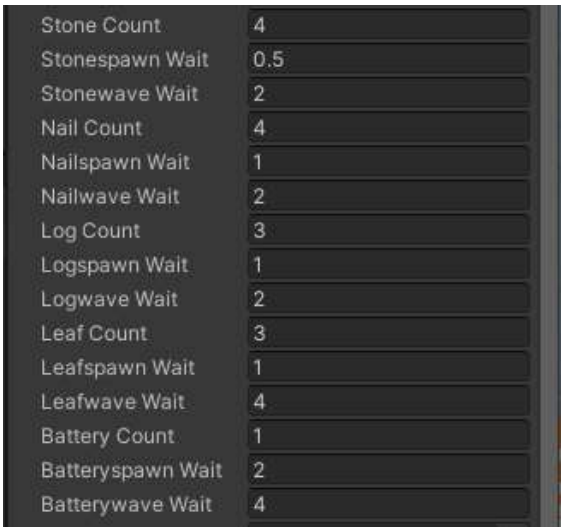
Wave Wait: είναι ο χρόνος αναμονής που μετρά το σύστημα πριν εξαπολύσει ένα νέο κύμα στοιχείων.

Count: είναι το πλήθος των στοιχείων που εξαπολύονται ανά κύμα.

Όλα αυτά ορίζονται εδώ στον κώδικα μας:

```
public int stoneCount;
    public float stonespawnWait;
    public float stonewaveWait;
        public int nailCount;
    public float nailspawnWait;
    public float nailwaveWait;
        public int logCount;
    public float logspawnWait;
    public float logwaveWait;
        public int leafCount;
    public float leafspawnWait;
    public float leafwaveWait;
        public int batteryCount;
    public float batteryspawnWait;
    public float batterywaveWait;
```

και με αυτόν τον τρόπο ελέγχουμε ακριβώς την συχνότητα που θα εμφανίζεται το κάθε στοιχείο στο παιχνίδι μας. Με αυτόν τον τρόπο μπορούμε να επηρεάσουμε την δυσκολία του παιχνιδιού. Για παράδειγμα μπορούμε να φέρουμε περισσότερες πέτρες για να απειλούμε τον παίχτη μας ή να φέρουμε περισσότερα κούτσουρα για να τον βοηθήσουμε να μαζέψει πόντους. Επίσης μπορούμε να φέρουμε πιο αραιά μπαταρίες για να κινδυνεύει να ξεμείνει από ενέργεια.



Επίσης ορίζουμε τον χρόνο εκκίνησης, που είναι ο χρόνος κατά τον οποίο στην αρχή του παιχνιδιού δεν θα εμφανιστεί κανένα στοιχείο, για να δώσουμε την ευκαιρία στον παίχτη μας να προετοιμαστεί.

```
public float startWait;
```

Με τις ανάλογες μεταβλητές αντικειμένων παιχνιδιού ορίζουμε ποιο στοιχείο που έχουμε στην unity θα είναι η πέτρα, ποιο το κούτσουρο κτλ.

```
    public GameObject stone;
        public GameObject nail;
        public GameObject log;
```

```
public GameObject leaf;
public GameObject battery;
```

Τα κύματα των αντικειμένων πρέπει να εξαπολύονται από συγκεκριμένο σημείο της οθόνης.
Αυτό το ορίζουμε από το :
public Vector3 spawnValues;

Με τον τρόπο αυτό ορίζουμε στο πρόγραμμα να γεννά τα αντικείμενα ακριβώς λίγο έξω από το αριστερό όριο της οθόνης μας και ύστερα να έρχονται αυτά προς τον παίχτη μας.
Με το ξεκίνημα του παιχνιδιού δίνουμε εντολή μέσω του κώδικα μας να ξεκινήσουν μαζί και κάποιες ρουτίνες (Coroutine) οι οποίες θα εφαρμόζουν όλα όσα έχουμε ορίσει μέχρι τώρα.

```
StartCoroutine (SpawnWaves1 ());
        StartCoroutine (SpawnWaves2 ());
        StartCoroutine (SpawnWaves3 ());
        StartCoroutine (SpawnWaves4 ());
        StartCoroutine (SpawnWaves5 ());
```

Η κάθε μια αυτό αυτές τις ρουτίνες ελέγχει την εμφάνιση ενός από τα στοιχεία του παιχνιδιού μας όπως φαίνεται παρακάτω για την περίπτωση της πέτρας:

```
IEnumerator SpawnWaves1 ()
{
    yield return new WaitForSeconds (startWait);
    while (true)
    {
        for (int i = 0; i < stoneCount; i++)
        {
            Vector3 spawnPosition = new Vector3 (spawnValues.x, spawnValues.y, Random.Range (-spawnValues.z, spawnValues.z));
            Quaternion spawnRotation = Quaternion.identity;
            Instantiate (stone, spawnPosition, spawnRotation);
            yield return new WaitForSeconds (stonespawnWait);
        }
        yield return new WaitForSeconds (stonewaveWait);

        if (gameOver)
        {
            restartText.text = "Press 'R' for Restart";
            restart = true;
```

```
            StopCoroutine("decreaseCharge");
            Charge = 100;
            break;
        }
    }
        }
```

Ορίζουμε ακόμη το αντικείμενο της μπάρας ενέργειας μας.

```
public ProgressBar progressBar;
```

Επίσης ορίζουμε τις μεταβλητές με τις οποίες θα ελέγχουμε τα λεκτικά της βαθμολογίας, της επανεκκίνησης και του τέλους παιχνιδιού.

```
public Text scoreText;
public Text restartText;
public Text gameOverText;
```

Ορίζουμε ακόμη τον τύπο των μεταβλητών αυτών, καθώς και της μεταβλητής της φόρτισης.

```
private bool gameOver;
private bool restart;
private int score;
public int Charge;
```

Κατά την εκκίνηση του παιχνιδιού ορίζουμε τα παρακάτω:

```
void Start ()
{

    gameOver = false;
    restart = false;
    restartText.text = "";
    gameOverText.text = "";
    score = 0;
    Charge = 100;
    UpdateScore();
```

Όπως φαίνεται το σκορ μας είναι 0 και η φόρτιση της μπαταρίας μας είναι 100.

Παράλληλα ξεκινά και μια ρουτίνα μαζί με την εκκίνηση του παιχνιδιού που αφορά την αποφόρτιση της μπαταρίας μας.

```
    StartCoroutine("decreaseCharge");
```

Η ρουτίνα αυτή φαίνεται παρακάτω:

```
IEnumerator decreaseCharge ()
{

        yield return new WaitForSeconds(1F);
        Charge -= 1;
        StartCoroutine("decreaseCharge");
```

```
}
```

Με αυτόν τον κώδικα ορίζουμε την τιμή φόρτισης της μπαταρίας στο 100% όταν ξεκινά το παιχνίδι κι όσο διαρκεί το παιχνίδι αυτή μειώνεται 1% ανά δευτερόλεπτο. Σε συνδυασμό με τον προηγούμενο κώδικα που αφορούσε την μπάρα μας, μόλις αυτή φτάσει στο 20%, θα γίνει κόκκινη και ανά ένα δευτερόλεπτο που θα μειώνεται θα κάνει κι ένα προειδοποιητικό beep για το παίχτη.

Η τιμή της μπάρας μας ανανεώνεται συνέχεια σύμφωνα με την μεταβλητή Charge.

```
void Update()
{
   {
      progressBar.BarValue = Charge;
   }
```

Οι τιμές των μεταβλητών game over και restart είναι τύπου Boolean και αρχικά είναι ανενεργές. Ενεργοποιούνται αν κάτι προκαλέσει το τέλος του παιχνιδιού.

```
public void GameOver()
{
   gameOverText.text = "Game Over!";
   gameOver = true;
}
```

Και

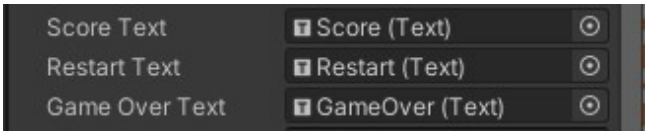
```
      if (gameOver)
      {
         restartText.text = "Press 'R' for Restart";
         restart = true;
         StopCoroutine("decreaseCharge");
         Charge = 100;
         break;
      }
```

Αν κάποιος πατήσει το κουμπί R το παιχνίδι δρομολογεί την επανεκκίνηση της σκηνής του παιχνιδιού όπως φαίνεται παρακάτω:

```
   if (restart)
   {
      if (Input.GetKeyDown(KeyCode.R))
      {
         SceneManager.LoadScene(SceneManager.GetActiveScene().buildIndex);
      }
   }
```

Με τις παραπάνω παρεμβάσεις προσθέσαμε τις μεταβλητές και τα ανάλογα πεδία κειμένων για την επανεκκίνηση και το τέλος του παιχνιδιού. Στην συνέχεια έχουμε προσθέσει την if με την οποία θα σπάει η συνέχεια του παιχνιδιού αν υπάρχει GameOver, το κείμενο που θα εμφανίζει όταν θα έχουμε GameOver την δυνατότητα του παίχτη να ξαναπαίξει και σαφώς την λειτουργία της επανεκκίνησης με το πάτημα του πλήκτρου R.

Αντιστοιχίζουμε τα πεδία που έχουμε φτιάξει με τα ανάλογα αντικείμενα που φτιάξαμε σε προηγούμενο κεφάλαιο.



Τέλος παραθέτουμε τον κώδικα με τον οποίο ελέγχουμε και ανανεώνουμε το σκορ του παίχτη μας:

```
public void AddScore(int newScoreValue)
    {
        score += newScoreValue;
        UpdateScore();
    }

    void UpdateScore()
    {
        scoreText.text = "Score: " + score;
    }
```

Με τις παραπάνω προσθήκες στον κώδικα έχουμε προσθέσει τις μεταβλητές με τις οποίες θα μετράμε το score. Οι βαθμοί όμως που θα συλλέγουμε δεν υπάρχουν σαν πληροφορία στον κώδικα του Game Controller. Για τον λόγο αυτό έχουμε δημιουργήσει την μεταβλητή AssScore με την οποία συλλέγουμε τους βαθμούς από τον κώδικα Destroy Element By Contanct όπως φαίνεται παρακάτω στα τονισμένα σημεία του κώδικα:

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

public class DestroyElementByContanct : MonoBehaviour
{
    public GameObject explosion;
    public int scoreValue;
    private GameController gameController;

    void Start()
    {
        GameObject gameControllerObject = GameObject.FindGameObjectWithTag("GameController");
        if (gameControllerObject != null)
        {
            gameController = gameControllerObject.GetComponent<GameController>();
        }
        if (gameController == null)
        {
            Debug.Log("Cannot find 'GameController' script");
        }
    }
```

```
    void OnTriggerEnter(Collider other)
    {
        if (other.tag == "Boundary")
        {
            return;
        }
        Instantiate(explosion, transform.position, transform.rotation);

        gameController.AddScore(scoreValue);
        Destroy(gameObject);
    }
}
```

Με την παραπάνω προσθήκη στον κώδικα έχουμε συνδέσει την μεταβλητή ScoreValue του κώδικα GameController με αυτή του DestroyElementByContanct.

Το κομμάτι του κώδικα:

```
        GameObject gameControllerObject = GameObject.FindGameObjectWithTag("GameController");
        if (gameControllerObject != null)
        {
            gameController = gameControllerObject.GetComponent<GameController>();
        }
        if (gameController == null)
        {
            Debug.Log("Cannot find 'GameController' script");
        }
```

Το χρειαζόμαστε για να επικοινωνήσουν τα δύο αντικείμενα κώδικα μεταξύ τους και να γίνει η επιθυμητή ανταλλαγή πληροφοριών.

Τέλος ενέργειας και τέλος παιχνιδιού.

Νωρίτερα παραθέσαμε τον αρχικό κώδικα του Player Controller με τον οποίο δίναμε κίνηση στον παίχτη μας. Παρακάτω παραθέτουμε τον τελικό κώδικα με τις ανάλογες προσθήκες που προέκυψαν από την εξέλιξη της κατασκευής του παιχνιδιού μας.

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

[System.Serializable]
public class Boundary
{
   public float xMin, xMax, zMin, zMax;
}

public class PlayerController : MonoBehaviour
{
   public float speed;
   public float tilt;
   public Boundary boundary;
   public GameObject playerExplosion;
   private GameController gameController;

   void Start()
   {
      GameObject gameControllerObject =
GameObject.FindGameObjectWithTag("GameController");
      if (gameControllerObject != null)
      {
         gameController = gameControllerObject.GetComponent<GameController>();
      }
      if (gameController == null)
      {
         Debug.Log("Cannot find 'GameController' script");
      }
   }

   void FixedUpdate ()
   {
      float moveHorizontal = Input.GetAxis ("Horizontal");
      float moveVertical = Input.GetAxis ("Vertical");

      Vector3 movement = new Vector3 (moveHorizontal, 0.0f, moveVertical);
      GetComponent<Rigidbody>().velocity = movement * speed;

      GetComponent<Rigidbody>().position = new Vector3
      (
```

```
            Mathf.Clamp (GetComponent<Rigidbody>().position.x, boundary.xMin, boundary.xMax),
            0.0f,
            Mathf.Clamp (GetComponent<Rigidbody>().position.z, boundary.zMin, boundary.zMax)
        );

        GetComponent<Rigidbody>().rotation = Quaternion.Euler (GetComponent<Rigidbody>().velocity.z * -tilt, 180.0f, 0.0f);
    }

    private void Update()
    {
        if (gameController.Charge == 0)
        {
            Instantiate(playerExplosion, transform.position, transform.rotation);
            gameController.GameOver();
            Destroy(gameObject);
        }

    }
}
```

Όπως γίνεται αντιληπτό 2 είναι η βασικές προσθήκες. Πρώτον μπήκε ο κώδικας επικοινωνίας με το script GameController:

```
void Start()
    {
        GameObject gameControllerObject = GameObject.FindGameObjectWithTag("GameController");
        if (gameControllerObject != null)
        {
            gameController = gameControllerObject.GetComponent<GameController>();
        }
        if (gameController == null)
        {
            Debug.Log("Cannot find 'GameController' script");
        }
    }
```

Και δεύτερον μπήκε ο κώδικας με τον οποίο υπαγορεύουμε ότι αν η μεταβλητή Charge του GameController φτάσει στην τιμή μηδέν ο παίχτης μας θα αυτοκαταστραφεί και θα δώσει Game Over.

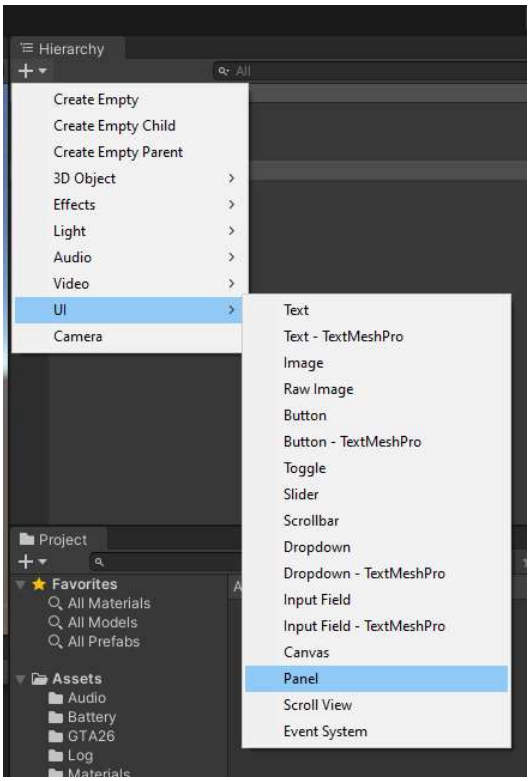
```
private void Update()
    {
        if (gameController.Charge == 0)
        {
            Instantiate(playerExplosion, transform.position, transform.rotation);
            gameController.GameOver();
            Destroy(gameObject);
        }
```

Το Μενού του Παιχνιδιού.

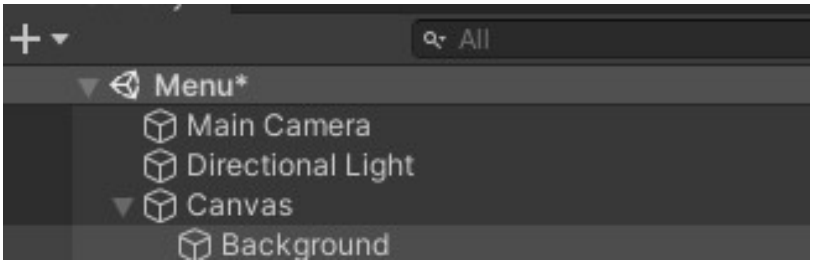
Από την αρχή του παιχνιδιού δημιουργήσαμε δύο σκηνές. Μια για το κύριο παιχνίδι μας και μια για το εισαγωγικό μενού του παιχνιδιού. Τώρα ήρθε η στιγμή να φτιάξουμε αυτό το μενού το οποίο θα καλωσορίζει και θα ενημερώνει τους παίχτες του παιχνιδιού.

Με την χρήση απλών εργαλείων των windows όπως η ζωγραφική φτιάχνουμε μια νέα εικόνα που θα χρησιμοποιήσουμε σαν background εικόνα.

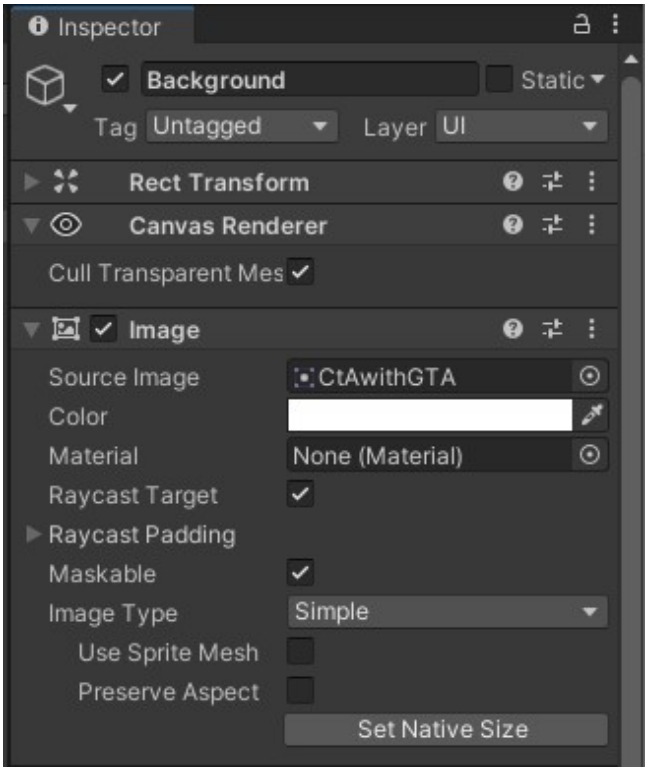
Έτσι λοιπόν πάμε στην νέα σκηνή και προσθέτουμε ένα νέο αντικείμενο Panel.



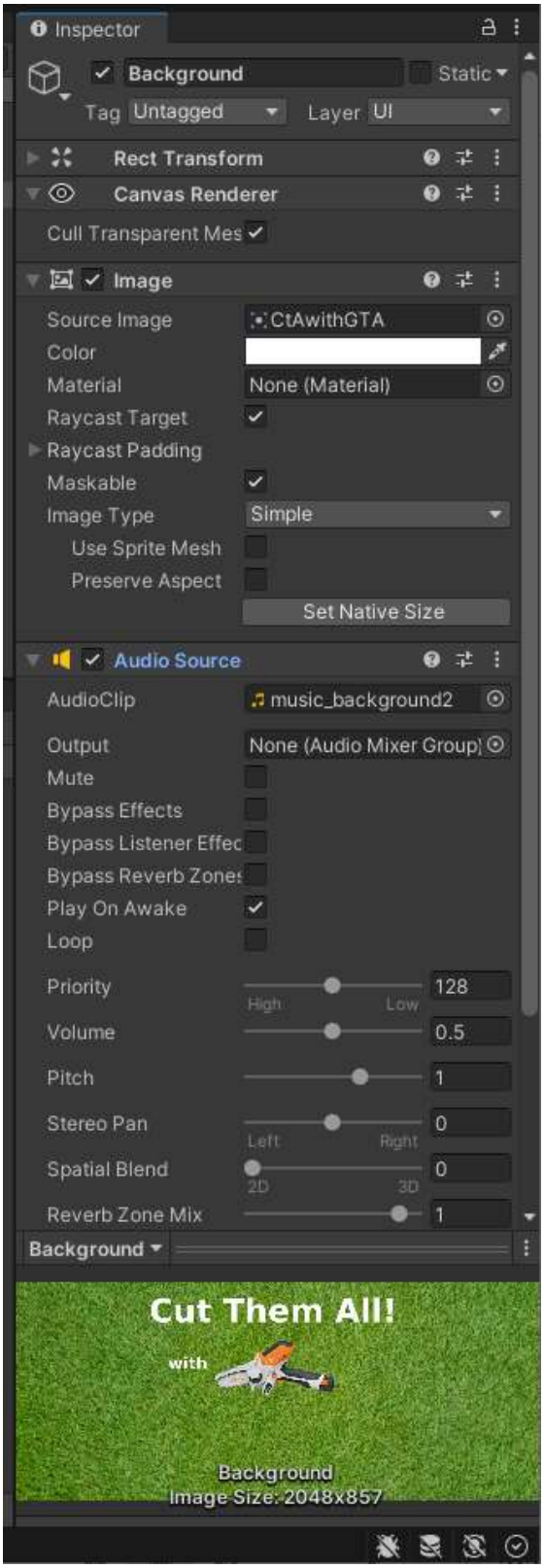
Αυτή η κίνηση θα δημιουργήσει ένα νέο αντικείμενο Canvas το οποίο θα φιλοξενήσει το αντικείμενο Panel που μόλις δημιουργήσαμε. Μετονομάζουμε το αντικείμενο Panel σε Background.



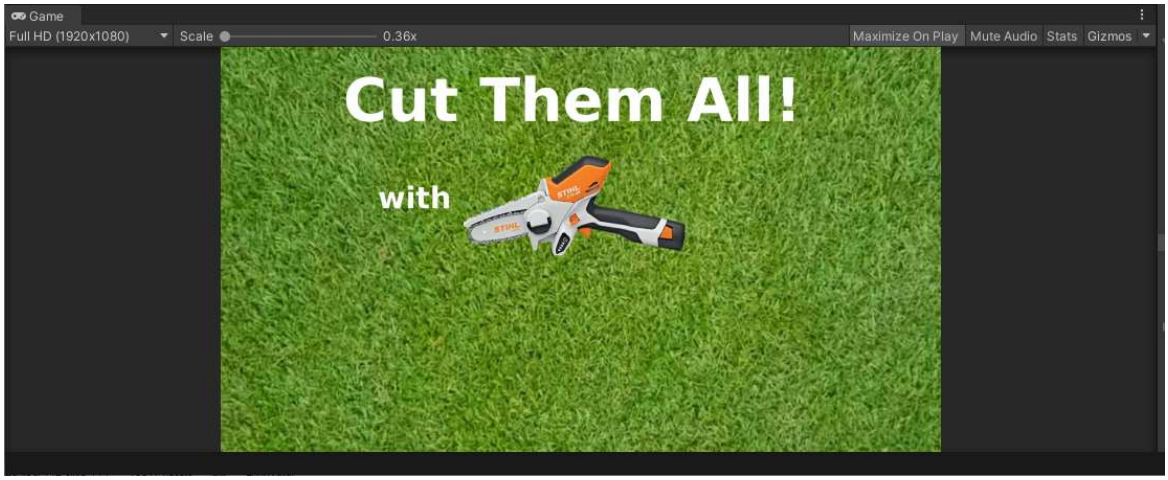
Πάμε και συσχετίζουμε την εικόνα που μόλις φτιάξαμε στην ζωγραφική με το νέο αντικείμενο που φτιάξαμε στην Unity.



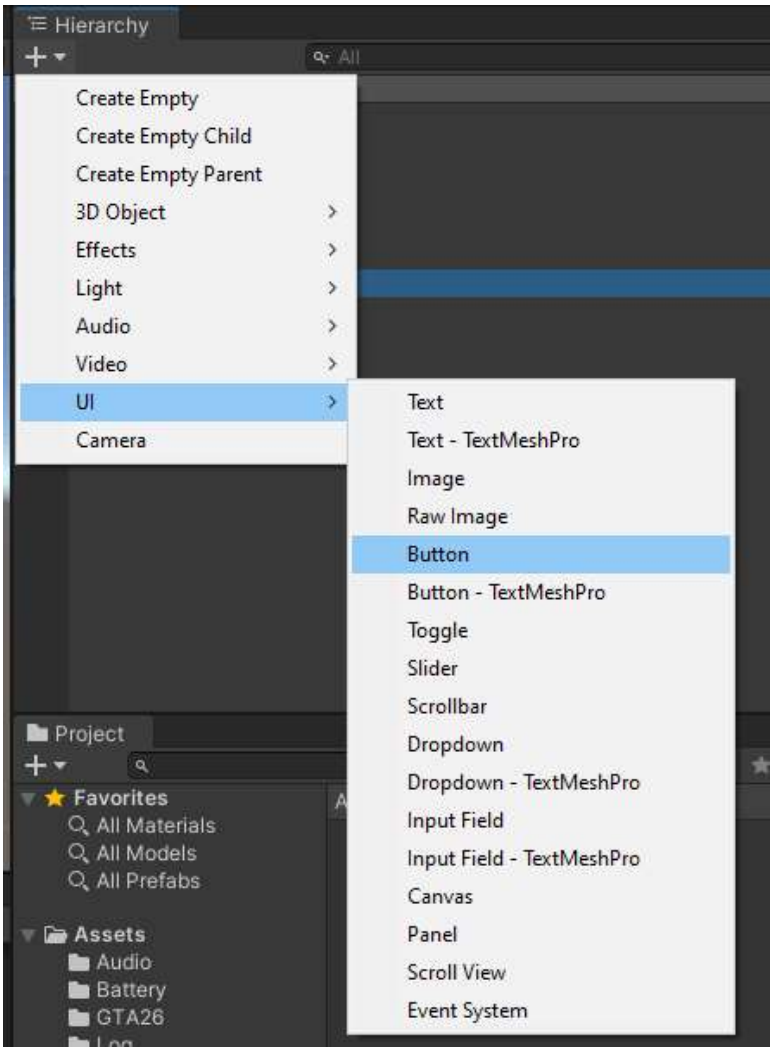
Επίσης συσχετίζουμε κι ένα ηχητικό κομμάτι με το αντικείμενο αυτό, έτσι ώστε να ξεκινάει μουσική με το που μπαίνει ο παίχτης στο μενού του παιχνιδιού μας.


Inspector
Background
Static
Tag Untagged
Layer UI
Rect Transform
Canvas Renderer
Cull Transparent Mes
Image
Source Image CtAwithGTA
Color
Material None (Material)
Raycast Target
Raycast Padding
Maskable
Image Type Simple
Use Sprite Mesh
Preserve Aspect
Set Native Size
Audio Source
AudioClip music_background2
Output None (Audio Mixer Group)
Mute
Bypass Effects
Bypass Listener Effec
Bypass Reverb Zones
Play On Awake
Loop
Priority 128
High Low
Volume 0.5
Pitch 1
Stereo Pan 0
Left Right
Spatial Blend 0
2D 3D
Reverb Zone Mix 1
Background
Cut Them All!
with
Background
Image Size: 2048x857

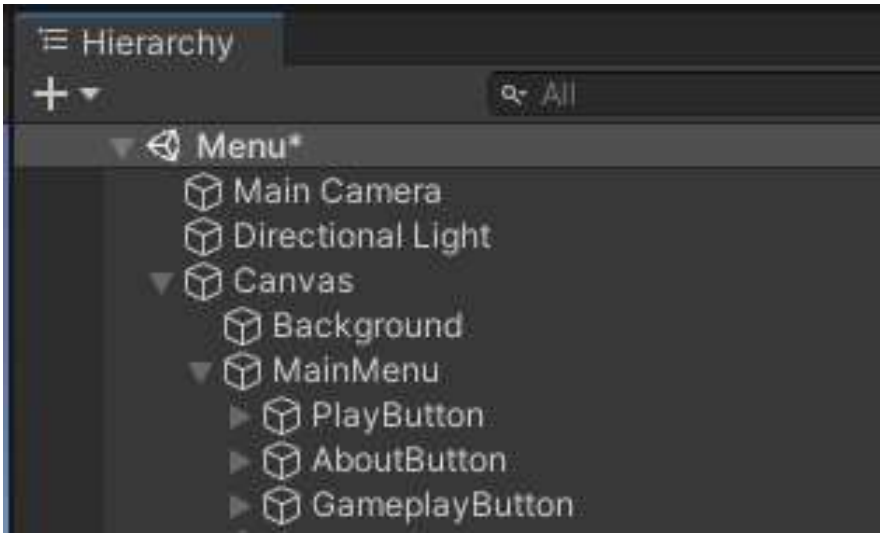
Το τελικό αποτέλεσμα στο παιχνίδι θα δείχνει έτσι.



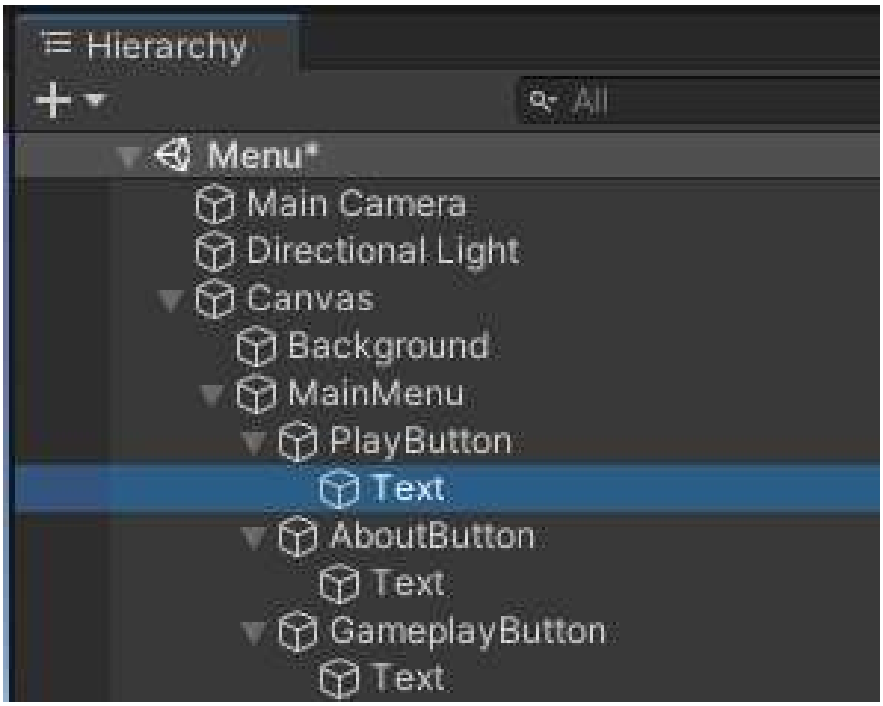
Πάμε τώρα να προσθέσουμε τα επιμέρους στοιχεία του μενού μας. Θα αρχίσουμε προσθέτοντας κουμπιά που θα είναι απαραίτητα στο μενού μας.



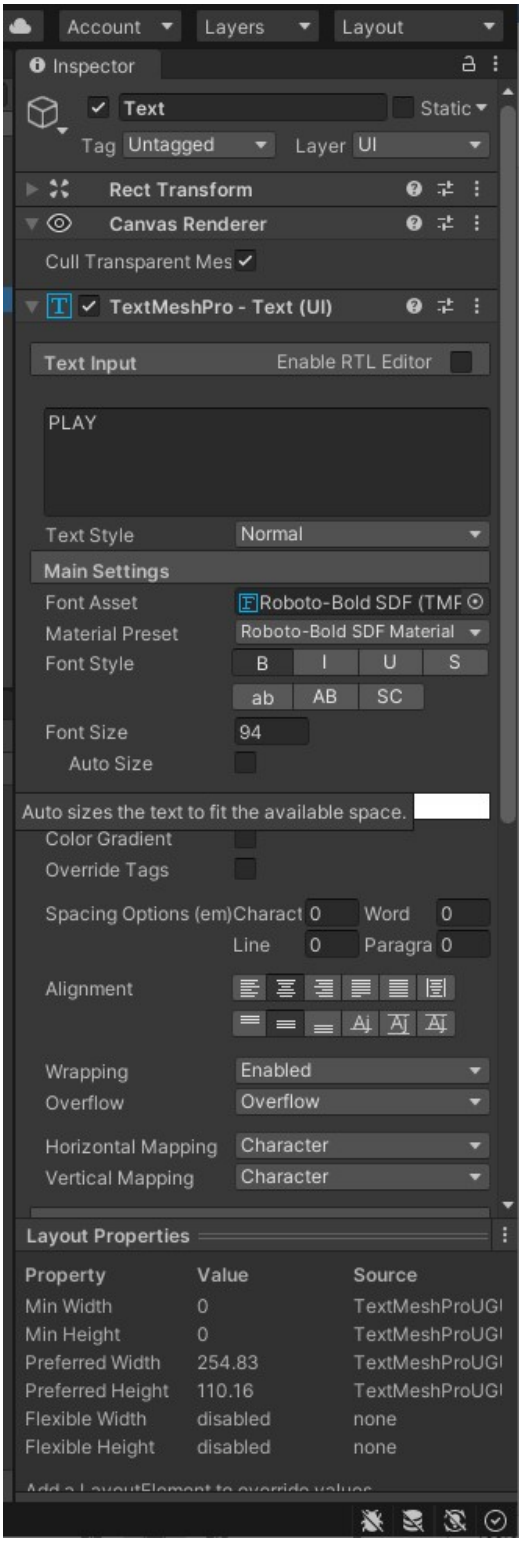
Θα φτιάξουμε ένα κενό αντικείμενο με τον όνομα Main Menu και κάτω από αυτό θα προσθέσουμε τρία κουμπιά. Ένα PlayButton, ένα AboutButton κι ένα Gameplay Button.



Κάτω από το κάθε κουμπί δημιουργούμε ένα text αντικείμενο στο οποίο τοποθετούμε το λεκτικό του κάθε κουμπιού καθώς και την μορφοποίηση του.

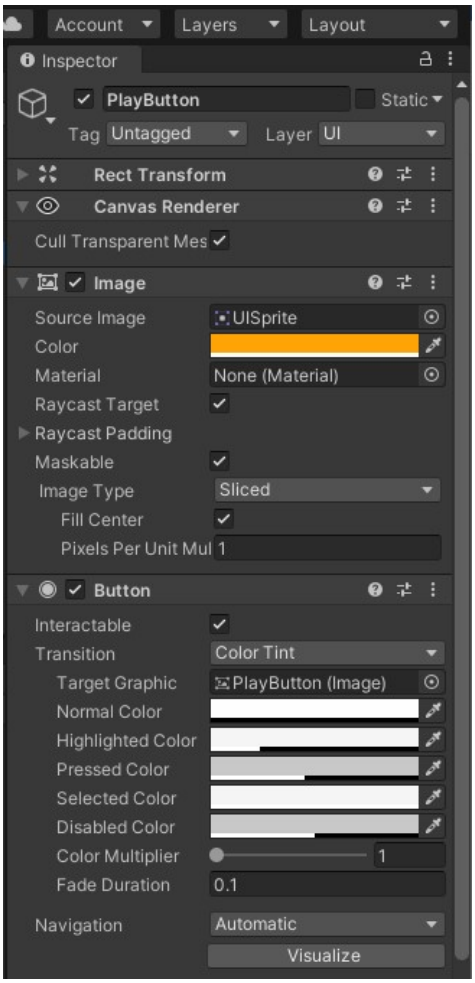


Εκεί θα καθορίσουμε την γραμματοσειρά που θα έχουν τα γράμματα πάνω στο κουμπί, το μέγεθος τους, την στοίχιση τους και πολλά άλλα.



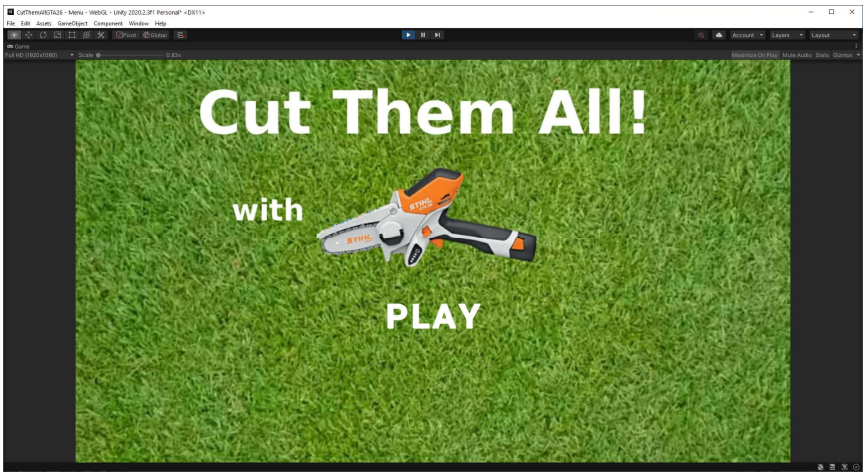
Στην παραπάνω εικόνα απεικονίζονται οι ρυθμίσεις για το κείμενο στο κουμπί Play.

Αντίστοιχα πηγαίνοντας ένα επίπεδο πιο πάνω στις ρυθμίσεις του κουμπιού Play θα καθορίσουμε άλλα δομικά στοιχεία του κουμπιού όπως το χρώμα του και τις αντιδράσεις του όταν περνάμε το κέρσορα από πάνω του και όταν τελικά το επιλέγουμε με κλικ.

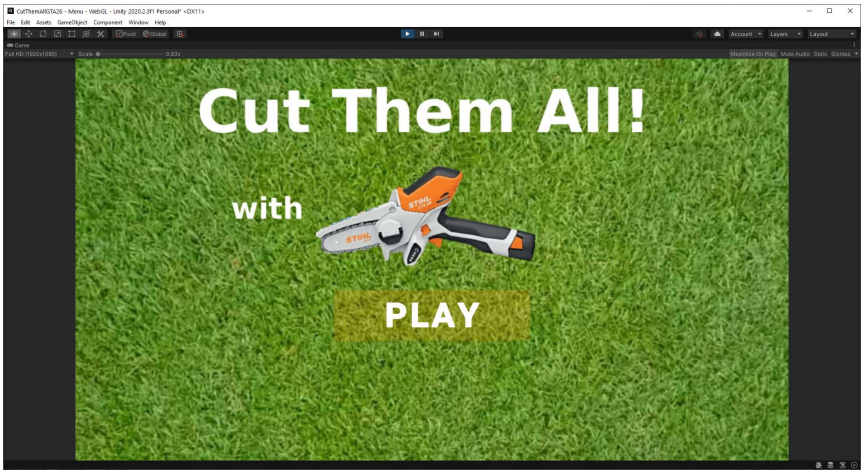


Τέλος ρυθμίζουμε τι θα γίνεται όταν ο παίχτης πατήσει το κουμπί αυτό. Στην περίπτωση του κουμπιού Play όπως γίνεται κατανοητό θα βάλουμε να ξεκινά η σκηνή Main, που ισοδυναμεί με το να ξεκινάει το παιχνίδι μας.

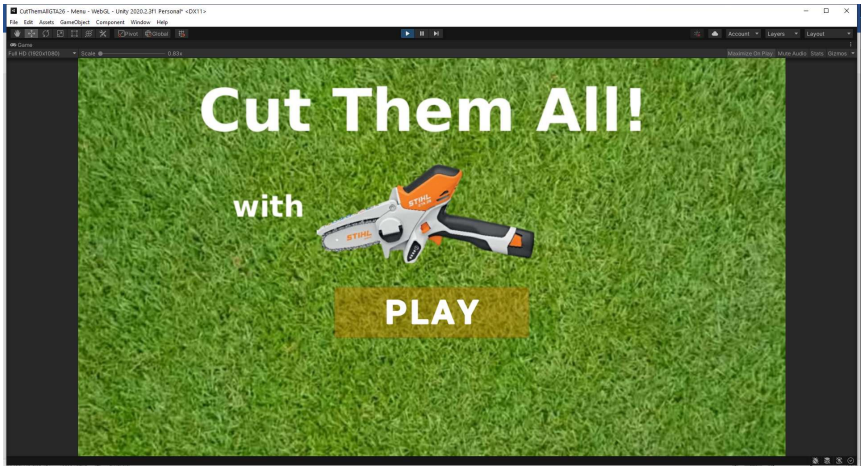
Με την τοποθέτηση του κουμπιού Play το τελικό αποτέλεσμα θα δείχνει έτσι.



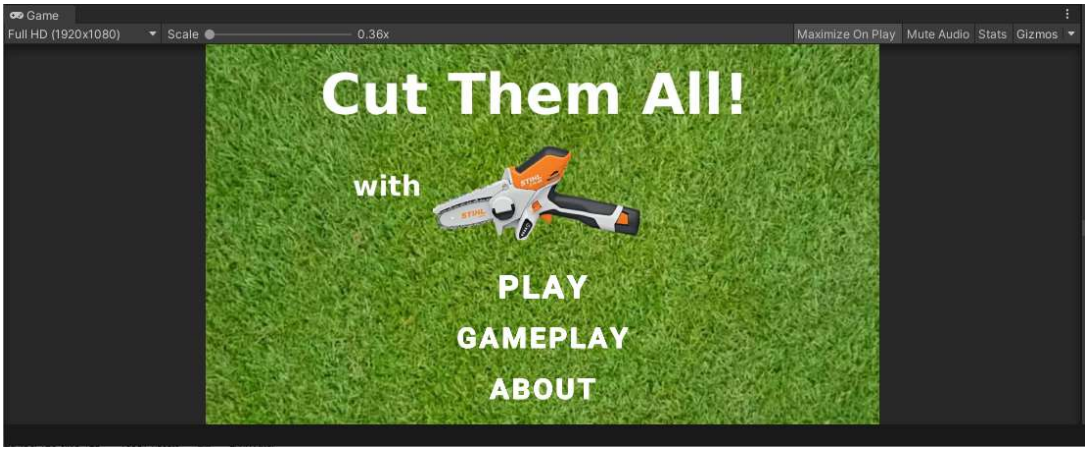
Όταν ο παίχτης θα περνά τον κέρσορα πάνω από το κουμπί θα εμφανίζεται το πλαίσιο του.



Όταν ο παίχτης θα επιλέγει το κουμπί με κλικ αυτό το πλαίσιο θα γίνεται πιο έντονο.

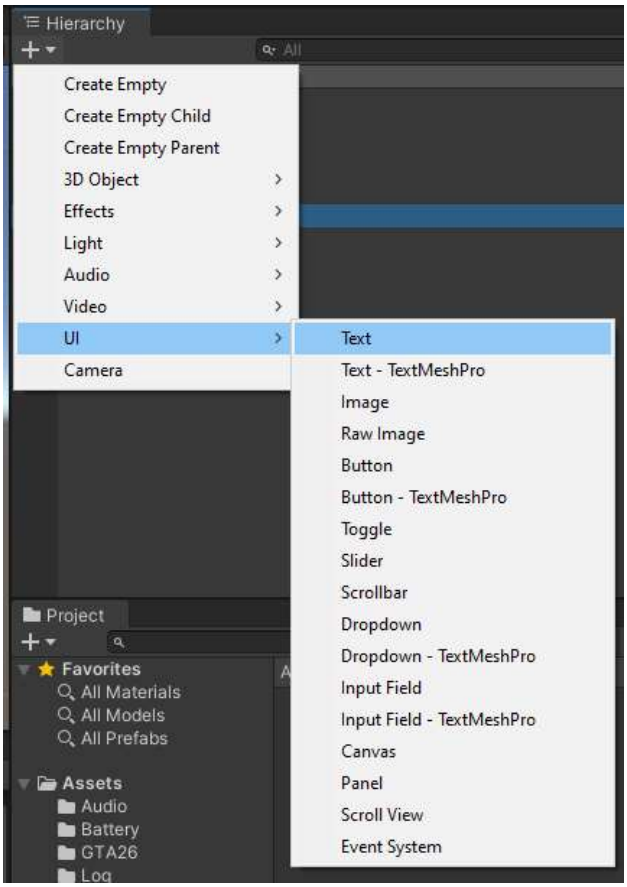


Κάνουμε το ίδιο με τις ρυθμίσεις και τις τοποθετήσει και των άλλων δύο κουμπιών του παιχνιδιού μας και το τελικό αποτέλεσμα του μενού μας είναι όπως φαίνεται στην παρακάτω εικόνα.



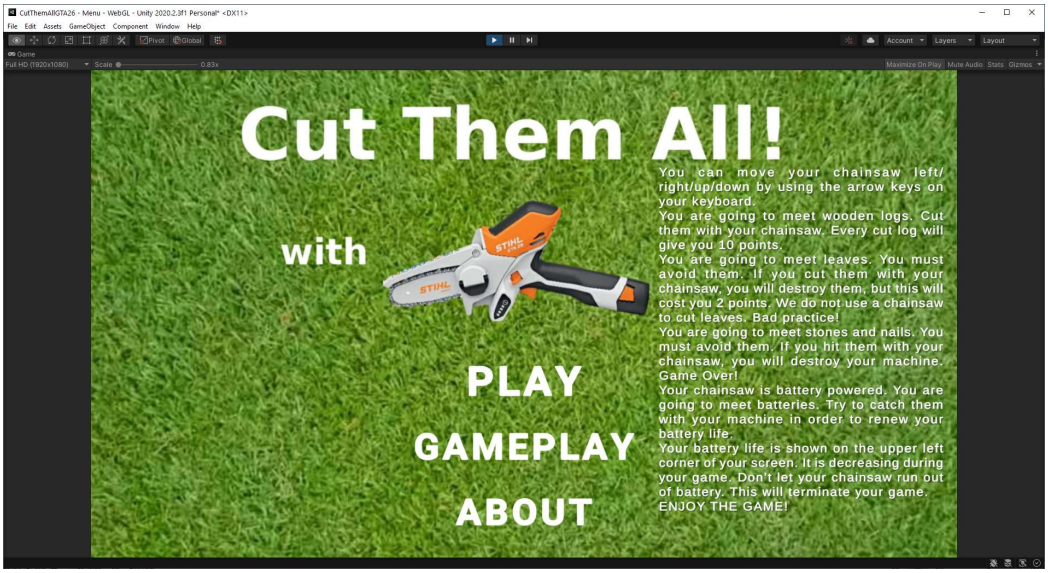
Θα πρέπει μόνο να ορίσουμε τι θα κάνουν τα δύο νέα κουμπιά στο παιχνίδι μας. Όπως γίνεται κατανοητό από τους τίτλους των κουμπιών, το κουμπί Gameplay θα δίνει πληροφορίες για τον τρόπο παιχνιδιού και το κουμπί About θα δίνει πληροφορίες σχετικές με το παιχνίδι.

Για το κουμπί Gameplay λοιπόν θα δημιουργήσουμε ένα απλό πλαίσιο κειμένου UI Text.

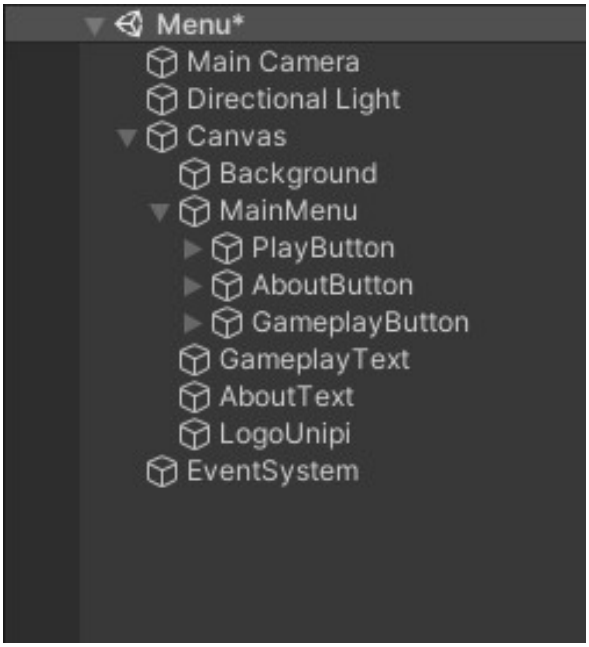


Σε αυτό το πλαίσιο κειμένου θα γράψουμε αναλυτικά όλους τους κανόνες που διέπουν το παιχνίδι μας. Με τον τρόπο αυτό ο παίχτης θα είναι σε θέση να ξέρει πώς πρέπει να παίξει, τι να επιδιώκει αλλά και τι να αποφεύγει κατά την διάρκεια του παιχνιδιού. Δεν είναι σωστό να αφήσουμε το

παίχτη μας να τα βρει μόνος του ψάχνοντας και χάνοντας. Τοποθετούμε το κείμενο αυτό σε εμφανές σημείο στην οθόνη του μενού μας, χωρίς να επικαλύπτουμε άλλα αντικείμενα.

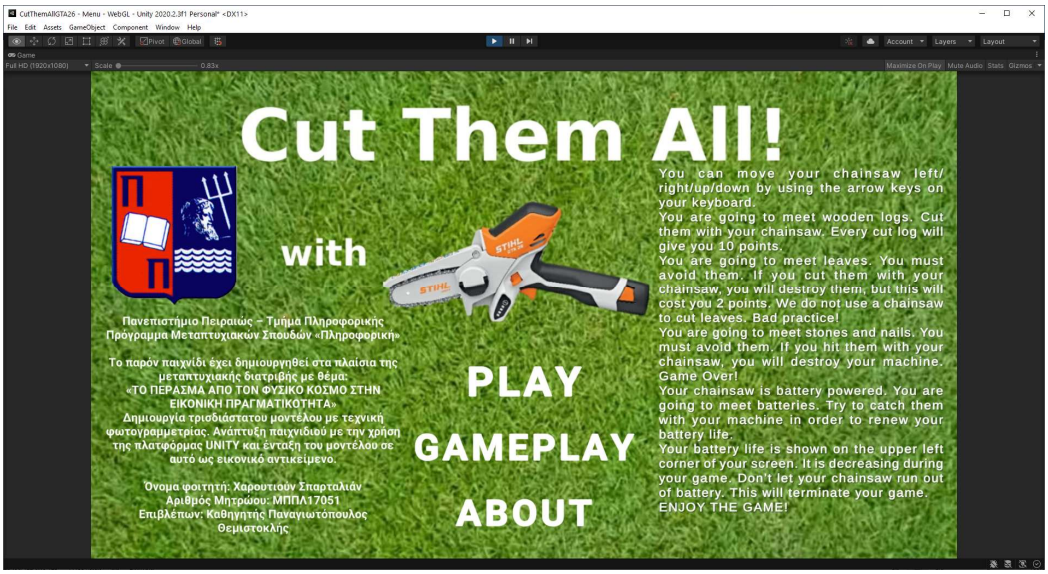


Για το κουμπί About ένα ακόμη πλαίσιο κειμένου με τις πληροφορίες που αφορούν τον σκοπό της κατασκευής ης εργασίας μας, καθώς και στοιχεία του φοιτητή που το κατασκεύασε, του επιβλέποντος καθηγητή και του προγράμματος σπουδών του πανεπιστημίου μας. Προσθέτουμε ακόμη και μια εικόνα του Πανεπιστημίου Πειραιώς.



Τα τοποθετούμε όλα στο μενού μας χωρίς να επικαλύπτουμε άλλα αντικείμενα.

Το τελικό αποτέλεσμα θα δείχνει έτσι.



Για να δώσουμε την δυνατότητα στον παίχτη να ενεργοποιεί και να διαβάζει όποιο πληροφορία τον ενδιαφέρει και για να μην τον βαραίνουμε με παραπάνω πληροφορίες, θα του δώσουμε την δυνατότητα να εμφανίζει και να εξαφανίζει τις πληροφορίες αυτές από την οθόνη του με την χρήση των κουμπιών που φτιάξαμε προηγουμένως.

Για να το πετύχουμε λοιπόν αυτό, θα γράψουμε ένα μικρό κομμάτι κώδικα σε C# με τον οποίο θα ενεργοποιούμε και θα απενεργοποιούμε τα αντικείμενα που θέλουμε. Ο κώδικας μα είναι ο παρακάτω:

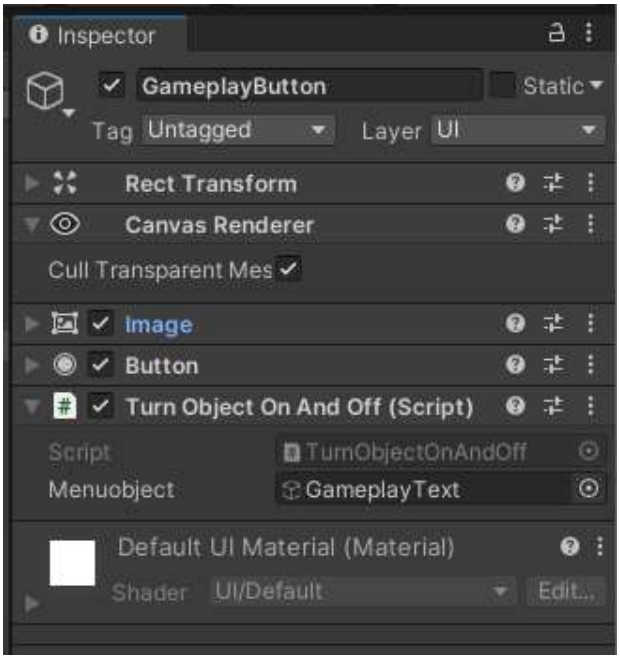
```
using System.Collections;
using System.Collections.Generic;
using UnityEngine;
using UnityEngine.UI;

public class TurnObjectOnAndOff : MonoBehaviour
{

    [SerializeField]
    private GameObject menuobject;

    private bool objectIsEnabled;


    // Start is called before the first frame update
    void Start()
    {
        gameObject.GetComponent<Button>().onClick.AddListener(TurnOnAndOff);
        objectIsEnabled = false;
        menuobject.SetActive(objectIsEnabled);

    }
```

```
    private void TurnOnAndOff()
    {
        objectIsEnabled ^= true;
        menuobject.SetActive(objectIsEnabled);
    }

}
```

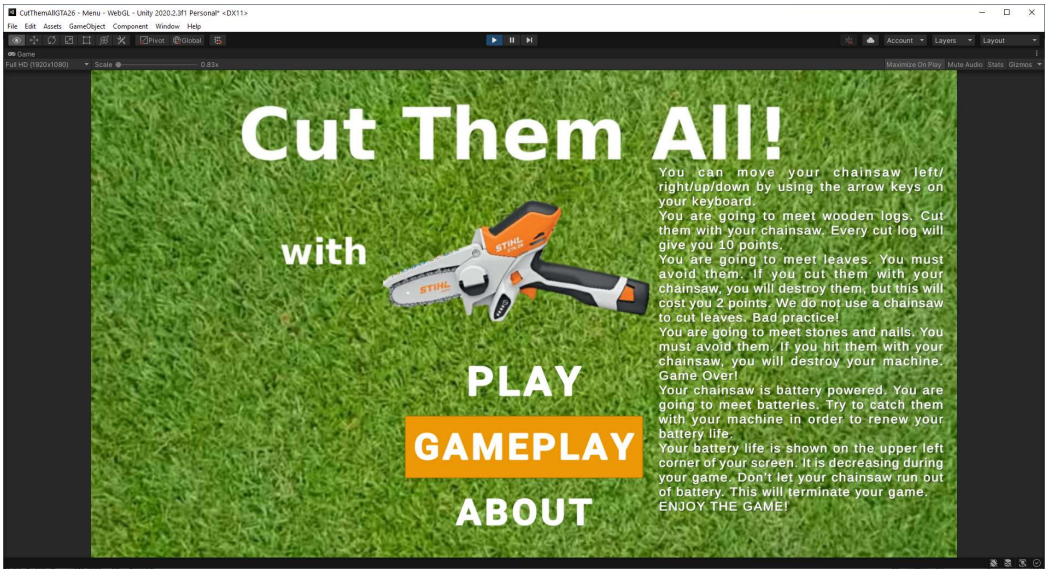
Με τον τρόπο αυτό τα κουμπιά Gameplay και About λειτουργούν σαν On-Off διακόπτες για τα αντικείμενα που θα τους ορίσουμε στην μεταβλητή menuobject.

Έτσι λοιπόν τα στοιχεία text GameplayText, AboutText και LogoUnipi είναι αρχικά απενεργοποιημένα.

Προσαρτούμε τον κώδικα TurnObjectOnAndOff στο κουμπί Gameplay με menuobject το GameplayText όπως φαίνεται παρακάτω.

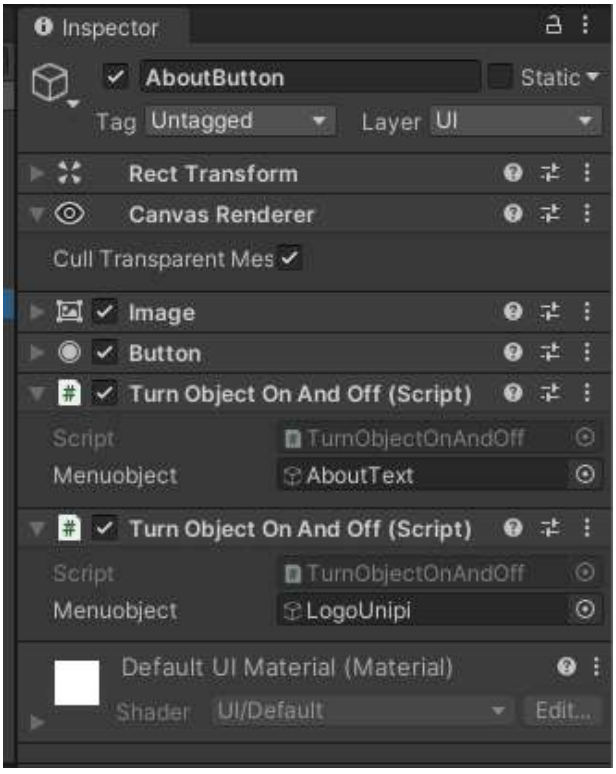


Έτσι αν ο παίχτης πατήσει το κουμπί Gameplay θα του εμφανιστεί το σχετικό κείμενο και θα έχει την δυνατότητα να το διαβάσει με την ησυχία του.

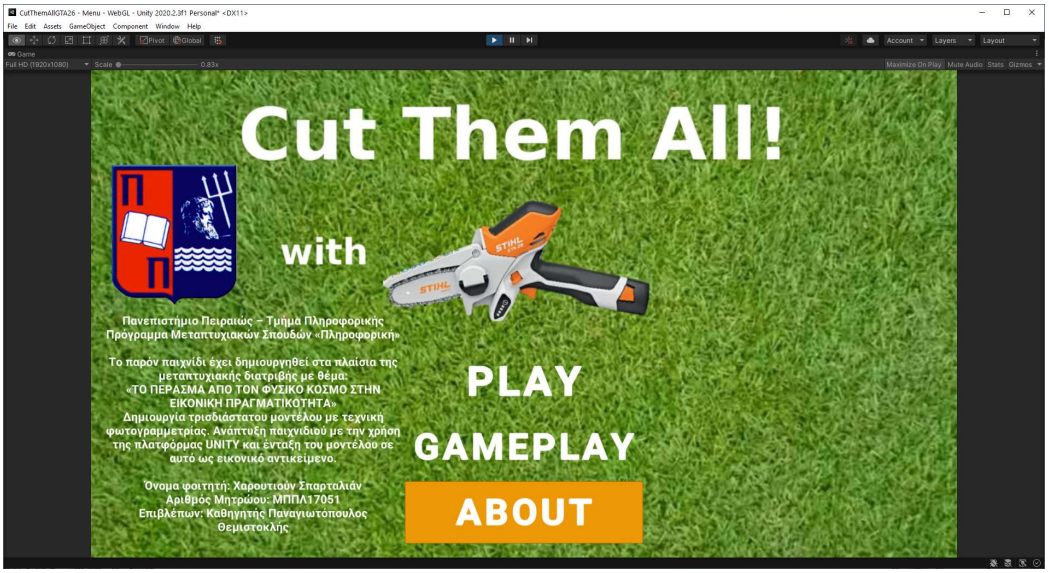


Αν ξαναπατήσει το κουμπί Gameplay το κείμενο θα εξαφανιστεί.

Θα κάνουμε τους ίδιους συσχετισμούς κώδικα και αντικειμένων για το κουμπί About όπως φαίνεται παρακάτω.



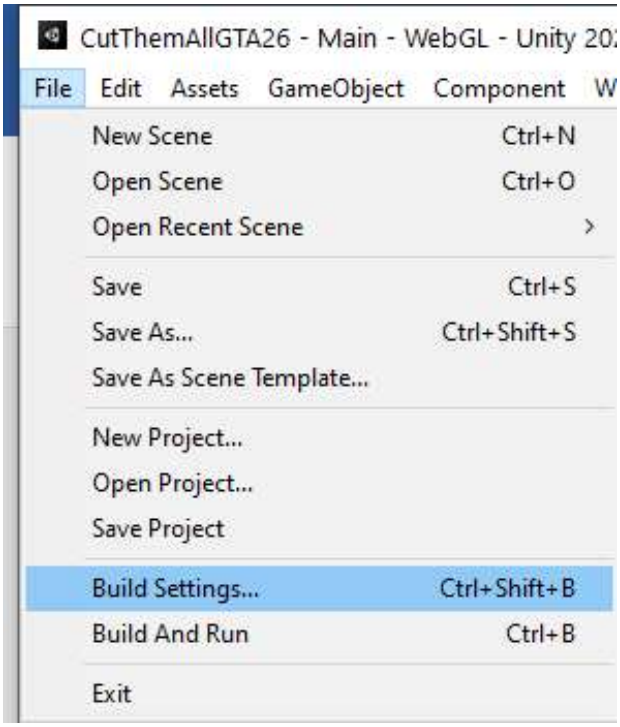
Μόνο που αυτή την φορά θα το κάνουμε δύο φορές για να συσχετίσουμε και το αντικείμενο κειμένου AboutText αλλά και το LogoUnipi που θέλουμε να εμφανίζονται και να εξαφανίζονται ταυτόχρονα. Έτσι όταν ο παίχτης πατήσει το κουμπί About θα εμφανίζεται το σχετικό κείμενο και το σήμα του Πανεπιστημίου Πειραιώς.



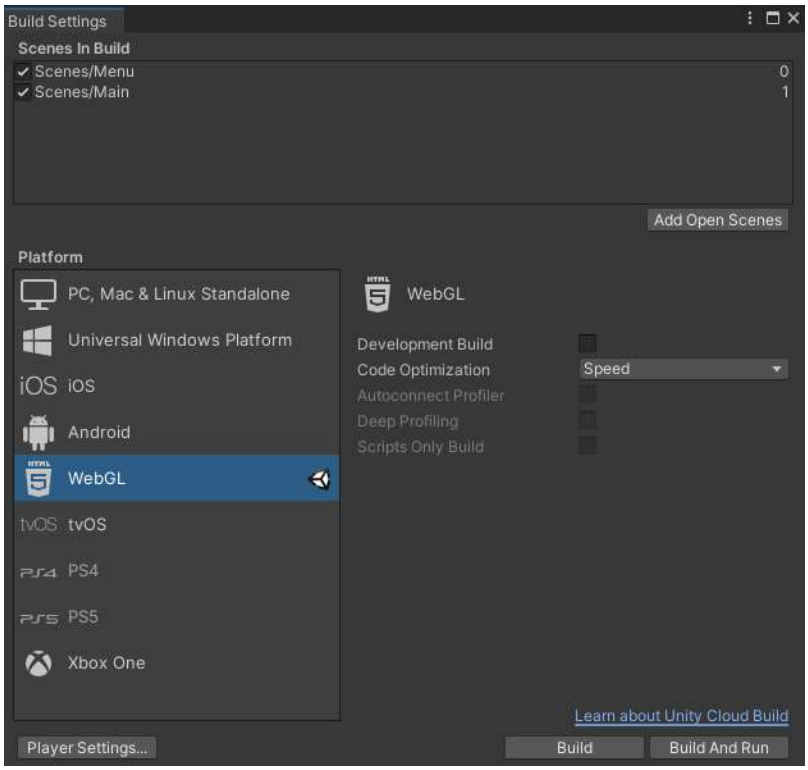
Αν ξαναπατήσει το κουμπί About τα δύο αντικείμενα θα εξαφανίζονται.

Το Τελικό Χτίσιμο και το WEB Upload.

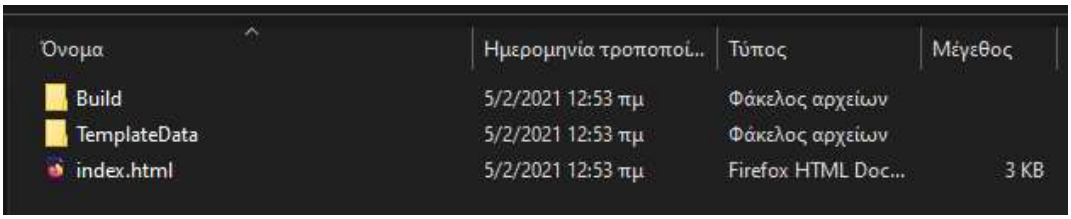
Το παιχνίδι μας είναι έτοιμο. Το μόνο που μας απομένει είναι να το εξάγουμε σε μια μορφή που να μπορεί να παίξει και κάποιος εκτός της unity. Από την αρχή εμείς έχουμε κάνει την δομή μας σαν παιχνίδι Web. Οπότε θα πάμε στα Build Settings.



Και θα κάνουμε Build.



Θα προκύψουν τα παρακάτω αρχεία



Τα αρχεία αυτά περιέχουν όλο μας το παιχνίδι και ότι αυτό χρειάζεται για να παιχτεί σε μορφή Web.

Εμείς με την βοήθεια του www.000webhost.com όπου μπορεί κανείς να ανεβάσει ένα site στο διαδίκτυο, ανεβάσαμε τα παραπάνω αρχεία για να μπορούμε να παίξουμε το παιχνίδι μας από οποιοδήποτε υπολογιστή με πρόσβαση στο διαδίκτυο.

Σας προσκαλούμε να παίξετε κι εσείς CutThemAllGTA26 με το να επισκεφθείτε την παρακάτω διεύθυνση:

https://russky-pressure.000webhostapp.com/?fbclid=IwAR086bjn9u6hfwoEfoxbzTpF-v14hGSf7MzkHdfXXXYqAtr4xb8tqoOf6hQ

Καλή διασκέδαση!

Τελικά Συμπεράσματα - Περίληψη

Στο σημείο αυτό θα ήθελα να παραθέσω τα τελικά συμπεράσματα που αποκόμισα από την γενικότερη ενασχόληση μου με την προετοιμασία της μεταπτυχιακής αυτής διατριβής και όχι μόνο.

Προερχόμενος από σπουδές διοικητικών και οικονομικών επιστημών και ενός πρώτου μεταπτυχιακού στα Logistics, οι εργασιακές μου ανάγκες με ώθησαν να ασχοληθώ όλο και πιο στενά με θέματα που σχετίζονταν με την πληροφορική. Για τον λόγο αυτό αποφάσισα να παρακολουθήσω ένα δεύτερο μεταπτυχιακό πρόγραμμα. Επέλεξα το μεταπτυχιακό πρόγραμμα «Πληροφορικής» του Πανεπιστήμιού Πειραιώς μιας και ταίριαζε απόλυτα στο προφίλ των γνωσιακών αναγκών που ήθελα να καλύψω. Μπήκα με μηδενικές γνώσεις σε γλώσσες προγραμματισμού και κώδικα. Μόνο μου εφόδιο ήταν η καλή μου σχέση με την τεχνολογία και η αέναη διάθεση μου για ανακάλυψη νέων γνωσιακών οριζόντων. Από την πρώτη κιόλας ημέρα παρακολούθησης των μαθημάτων έγινε αντιληπτό ότι το μεταπτυχιακό πρόγραμμα προϋποθέτει μεγάλη προσπάθεια και αφοσίωση από την μεριά μου μιας και κάθε μάθημα πραγματευόταν έννοιες νέες και δύσκολες, αλλά συνάμα πολύ ενδιαφέρουσες. Το έντονο εργασιακό μου πρόγραμμα ανέβαζε τον βαθμό δυσκολίας όλο και περισσότερο κάνοντας το όλο εγχείρημα μια μεγάλη πρόκληση.

Τώρα πια, μετά το πέρας όλων αυτών των δυσκολιών, νιώθω απόλυτα δικαιωμένος από τις επιλογές μου. Από το πρόγραμμα αυτό βγαίνω με πολύ διευρυμένη αντίληψη σε θέματα πληροφορικής, βγαίνω με καλές βάσεις γνώσεων προγραμματισμού σε γλώσσες όπως C, C# και Java. Γλώσσες προγραμματισμού και εργαλεία που ανοίγουν μπροστά στον καθένα μας ένα απέραντο πεδίο δημιουργικών δυνατοτήτων. Στο πρόγραμμα αυτό μας δόθηκε η ευκαιρία να γνωρίσουμε δημιουργικά εργαλεία όπως η Unity με την οποία μπορεί να δώσει κανείς σάρκα και οστά σε οτιδήποτε έχει φανταστεί. Για την εισαγωγή στον κόσμο της Unity θα ήθελα να ευχαριστήσω τον καθηγητή μου, κύριο Θεμιστοκλή Παναγιωτόπουλο, που μέσα από τον μάθημα της εικονικής πραγματικότητας με ενέπνευσε για την δημιουργία της διατριβής αυτής, και με καθοδήγησε ως επιβλέπων καθηγητής για την ολοκλήρωσή της.

Πόροι – Πηγές - Βιβλιογραφία

Για την δημιουργία τους μεταπτυχιακής διατριβής χρησιμοποιήθηκε ένα σταθερός υπολογιστής Lenovo ThinkCentre M920x με τα παρακάτω χαρακτηριστικά:

Επεξεργαστής:	Intel(R) Core(TM) i7-8700T CPU @ 2.40GHz 2.40 GHz
Μνήμη RAM:	16,0 GB DDR4 2400 MHz
Αποθηκευτικό μέσο:	512 GB M.2 PCIe SSD
Κάρτα γραφικών:	AMD ATI Radeon RX 560 Series 4GB
Λειτουργικό σύστημα:	Windows 10 Pro GR 64-bit

Η έκδοση της Unity που χρησιμοποιήθηκε είναι η: 2020.2.3f1.3936 Personal.

Η έκδοση του Unity Hub που χρησιμοποιήθηκε είναι η: 2.4.2

Το τρισδιάστατο μοντέλο για το κούτσουρο, το καρφί και την μπαταρία συλλέχθηκαν από την παρακάτω διεύθυνση:

https://www.turbosquid.com/

Το τρισδιάστατο μοντέλο για την πέτρα, τα ηχητικά και τα εφέ για τις εκρήξεις συλλέχθηκαν από την παρακάτω διεύθυνση:

https://assetstore.unity.com/

Η εικόνα για το φύλλο συλλέχθηκε από την παρακάτω διεύθυνση και επεξεργάστηκε με το εργαλείο ζωγραφικής των Windows:

https://www.amazon.com/Eureka-Photo-Image-Cut-Outs-Package/dp/B003D8BD7U

Η μουσική για το παρασκήνιο συλλέχθηκε από την παρακάτω διεύθυνση:

https://mixkit.co/free-stock-music/

Το ηχητικό εφέ με τον ήχο αλυσοπρίονου συλλέχθηκε από την παρακάτω διεύθυνση:

https://www.stihl.gr/Ixos-klisis-Alysopriono-STIHL.aspx

Τα ηχητικά στοιχεία επεξεργάστηκαν με το εργαλείο Audacity που μπορεί να βρει κανείς, να κατεβάσει και να χρησιμοποιήσει εντελώς δωρεάν από την παρακάτω διεύθυνση:

https://www.audacityteam.org/