ΣΧΟΛΗ ΜΗΧΑΝΙΚΩΝ

ΤΜΗΜΑ ΜΗΧΑΝΙΚΩΝ ΠΛΗΡΟΦΟΡΙΚΗΣ
ΚΑΙ ΗΛΕΚΤΡΟΝΙΚΩΝ ΣΥΣΤΗΜΑΤΩΝ

ΔΙΠΛΩΜΑΤΙΚΗ ΕΡΓΑΣΙΑ

«Δυναμική δημιουργία 2D/3D παιχνιδιού στο Unity3D»

Του φοιτητή
Βασιλείου Μάρτση
Αρ. Μητρώου: 154489

Επιβλέπων
Δρ Χριστίνα Βολιώτη
Διδάσκουσα ως Μετα-Διδάκτορας

Ημερομηνία 05/07/2022

Δυναμική δημιουργία 2D/3D παιχνιδιού στο Unity3D
Κωδικός Δ.Ε. 21326
Βασίλειος Μάρτσης
Χριστίνα Βολιώτη
Ημερομηνία ανάληψης Δ.Ε. 13-10-2021
Ημερομηνία περάτωσης Δ.Ε. 05/07/2022

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

Πρόλογος

Η εξέλιξη των τεχνολογιών των μηχανών παιχνιδιών (game engines) έχει φτάσει σε ένα επίπεδο ωριμότητας που επιτρέπει στις μηχανές παιχνιδιών να συνεισφέρουν σημαντικά στη δημιουργία ενός ποιοτικού παιχνιδιού εύκολα και αποδοτικά. Για να αναπτυχθεί ένα παιχνίδι θα πρέπει όλα τα μέλη της ομάδας ανάπτυξης παιχνιδιών να επικοινωνήσουν, συνεργαστούν και κατανοήσουν την ιδέα του παιχνιδιού για να μπορέσει στη συνέχεια να αναπτυχθεί το παιχνίδι επιτυχώς. Αυτή η διαδικασία μπορεί πολλές φορές να είναι απαιτητική και επαναλαμβανόμενη και αυτό γιατί τα μέλη της ομάδας, λόγω των διαφορετικών ρόλων, έχουν και διαφορετικό γνωστικό υπόβαθρο, π.χ. είναι ο ρόλος του σχεδιαστή (designer), του καλλιτέχνη (artist), του προγραμματιστή (programmer), του σχεδιαστή επιπέδου (level designer), του μηχανικού ήχου (sound engineer), του δοκιμαστή (tester), κ.ά..

Λαμβάνοντας τα παραπάνω υπόψη και στοχεύοντας στο να μειωθεί η διαδικασία αποτύπωσης της ιδέας αλλά και ανάπτυξης του παιχνιδιού, η συγκεκριμένη πτυχιακή προτείνει ένα εύχρηστο και απλό εργαλείο δυναμικής δημιουργίας είτε δισδιάστατου ή τρισδιάστατου παιχνιδιού στο Unity3D Game Engine. Η προστιθέμενη αξία είναι ότι η χρήση του εργαλείου δεν απαιτεί κανένα προγραμματιστικό υπόβαθρο, με αποτέλεσμα να μπορούν όλα τα μέλη της ομάδας ανάπτυξης παιχνιδιού να συμβάλλουν εξίσου στην ανάπτυξη του παιχνιδιού και να μην έχει ενεργό ρόλο μόνο ο προγραμματιστής.

Περίληψη

Σκοπός της παρούσας πτυχιακής εργασίας είναι η σχεδίαση και η ανάπτυξη ενός εργαλείου δημιουργίας είτε δισδιάστατου ή τρισδιάστατου παιχνιδιού δυναμικά στο Unity3D Game Engine. Η καινοτομία του συγκεκριμένου εργαλείου έγκειται στο ότι η χρήση του δεν απαιτεί προγραμματιστικό υπόβαθρο, δίνοντας έτσι τη δυνατότητα στον χρήστη να δημιουργήσει τα δικά του σενάρια παιχνιδιού, να εισάγει τα δικά του μοντέλα αντικειμένων (δισδιάστατα ή τρισδιάστατα) μέσα στο παιχνίδι έχοντας τα materials και textures της επιλογής του αλλά και να εισάγει animations σε κάθε ένα αντικείμενο μέσω μιας πολύ απλής και εύχρηστης διεπαφής. Το προτεινόμενο εργαλείο αποτελείται από δύο συστατικά, (α) το εργαλείο που συνοδεύεται από τη διεπαφή μέσω της οποίας ο χρήστης μπορεί εισάγει τα μοντέλα και τις ιδιότητες των αντικειμένων της σκηνής (π.χ. materials, textures, animations, κ.ά.) όπου στη συνέχεια δημιουργεί ένα Unity3D package, και (β) το ίδιο το Unity3D package που είναι υπεύθυνο για τη δυναμική δημιουργία του 2D/3D παιχνιδιού μέσα στο Unity3D Game Engine. Το πρώτο συστατικό πρόκειται για ένα πρόγραμμα κατασκευασμένο σε C# χρησιμοποιώντας το .NET Framework, ενώ το δεύτερο για ένα Unity3D package κατασκευασμένο πάλι σε C# στο Unity3D game engine.

Dynamic generation of a 2D/3D game in Unity3D

Vasileios Martsis

Abstract

The purpose of this thesis is to design and develop a tool to create either two-dimensional or three-dimensional game dynamically in the Unity3D Game Engine. The novelty of this tool is that it does not require a programming background while using it, thus allowing the user to create his/her own game scenarios, to import his/her own models (two-dimensional or three-dimensional) into the game having the materials and textures of his/her choice but also to add animations in each object through a very simple and easy-to-use interface. The proposed tool consists of two components, (a) the tool that comes with the interface through which the user can import the models and select the properties of the scene objects (e.g., materials, textures, animations, etc.) which then generates a Unity3D package, and (b) the Unity3D package itself which is responsible for the dynamic generation of the 2D/3D game within the Unity3D Game Engine. The first component is a program built in C # using the .NET Framework, while the second is a Unity3D package built in C # in the Unity3D game engine.

Ευχαριστίες

Στο σημείο αυτό θα ήθελα να ευχαριστήσω την κ. Βολιώτη Χριστίνας, Καθηγητή του Τμήματος Μηχανικών Πληροφορικής και Ηλεκτρονικών Συστημάτων του ΔΙ.ΠΑ.Ε., υπό την επίβλεψη της οποίας πραγματοποιήθηκε η πτυχιακή αυτή.

Περιεχόμενα

Κατάλογος Σχημάτων

Κατάλογος Πινάκων

Κεφάλαιο 1ο: Εισαγωγή

1.1 Κίνητρο & Περιγραφή Προβλήματος

Στη σημερινή εποχή οι απαιτήσεις των χρηστών στον τομέα των βιντεοπαιχνιδιών έχουν αυξηθεί σε μεγάλο βαθμό με αποτέλεσμα ο φόρτος εργασίας για την κατασκευή του παιχνιδιού αυτού να είναι υπερβολικά μεγάλος ώστε να μπορέσει να το διαχειριστεί ένα άτομο ή μια μικρή ομάδα ατόμων. Προκειμένου να μπορέσει μια σύγχρονη εταιρία κατασκευής βιντεοπαιχνιδιών να ανταπεξέλθει στις απαιτήσεις αυτές, αναγκάζεται να χωρίσει το προσωπικό της σε διάφορα τμήματα όπως τμήματα προγραμματισμού, κατασκευής σεναρίου, αφήγησης, κατασκευής περιβάλλοντος, κατασκευής χαρακτήρων, κατασκευής επιπέδων, κατασκευής animations, κατασκευής γραφικών, και πολλά άλλα ανάλογα με το μέγεθος του έργου. Πολλές φορές όμως υπάρχει η ανάγκη για συνεργασία δύο η παραπάνω διαφορετικών τμημάτων, ένα φαινόμενο που δημιουργεί προβλήματα καθώς τα διαφορετικά αυτά τμήματα καλούνται να κατανοήσουν και πολλές φορές να χρησιμοποιήσουν το έργο των άλλων τμημάτων προκειμένου να ολοκληρώσουν το δικό τους, με αποτέλεσμα να πρέπει τα μέλη αυτών τον τμημάτων να κατέχουν ή να αποκτήσουν γνώσεις σχετικά με τη δουλειά των άλλων τμημάτων. Κάτι τέτοιο είναι τις περισσότερες φορές τόσο χρονοβόρο όσο και κοστοβόρο για μια εταιρία, και είναι ένα σενάριο που η πλειοψηφία αυτών προσπαθεί να αποφύγει πραγματοποιώντας όσον τον δυνατόν καλύτερο διαχωρισμό και οργάνωση των έργων και της δουλειάς ανάλογα με τις γνώσεις των εργαζομένων του κάθε τμήματος.

1.2 Στόχος Διπλωματικής Εργασίας

Η παρούσα διπλωματική εργασία έρχεται να προτείνει λύση σε μία πολύ συγκεκριμένη όψη αυτού του προβλήματος που είναι η συνεργασία των υπολοίπων τμημάτων μιας εταιρίας κατασκευής παιχνιδιών με το τμήμα προγραμματισμού. Ποιο συγκεκριμένα ο στόχος της διπλωματικής είναι η δημιουργία ενός εργαλείου που θα δώσει την δυνατότητα σε οποιονδήποτε εκτός του τμήματος προγραμματισμού να μπορεί να δημιουργήσει ένα παιχνίδι χωρίς να έχει προγραμματιστικές γνώσεις, χρησιμοποιώντας δικά του αρχεία μοντέλων και γραφικών και ορίζοντας τις ιδιότητες του κάθε αντικειμένου στη σκηνή όπως αυτός επιθυμεί. Δύο βασικοί στόχοι του εργαλείου αυτού είναι η διαφάνεια και η ευχρηστία του, εννοώντας ότι ο τρόπος χρήσης του εργαλείου αυτού θα πρέπει να είναι ξεκάθαρος απέναντι σε έναν χρήστη οποιουδήποτε γνωστικού υπόβαθρου καθώς και να μπορεί να χρησιμοποιηθεί με απλά βήματα, χωρίς να είναι απαραίτητη η εκτέλεση περιττών επιπρόσθετων ενεργειών από πλευράς χρήστη.

1.3 Διάρθρωση Κειμένου

Το παρόν κείμενο αποτελεί την γραπτή αναφορά της διαδικασίας σχεδιασμού και υλοποίηση του εργαλείου που προαναφέρθηκε, καθώς και ένα εγχειρίδιο χρήσης του. Το κίνητρο και οι στόχοι που προσπαθήσει να πετύχει η συγκεκριμένη εργασία παρουσιάστηκαν σε αυτό το κεφάλαιο. Ωστόσο, στο επόμενο κεφάλαιο γίνεται η περιγραφή των διαφόρων μηχανών αναζήτησης στις οποίες θα μπορούσε να επεκταθεί μελλοντικά το εργαλείο αυτό, αναλύοντας τα πλεονεκτήματα και μειονεκτήματα του καθενός καθώς και τους λόγους για τους οποίους έγινε επιλογή του Unity3D για την υλοποίησή του.

Στο τρίτο κεφάλαιο περιγράφεται λεπτομερώς ο τρόπος διαμόρφωσης του τελικού συστήματος ενώ στο τέταρτο γίνεται αναφορά στον τρόπο χρήσης του εργαλείου. Τέλος στο πέμπτο κεφάλαιο αναφέρονται τα συμπεράσματα όπως και διάφορες προτάσεις βελτίωσης, δίνοντας προτάσεις για μελλοντικές έρευνες στο πρόβλημα.

1.4 Λογισμικό

Το προτεινόμενο εργαλείο αποτελείται από δύο συστατικά, (α) το εργαλείο που συνοδεύεται από τη διεπαφή μέσω της οποίας ο χρήστης μπορεί εισάγει τα μοντέλα και τις ιδιότητες των αντικειμένων της σκηνής (π.χ. materials, textures, animations, κ.ά.) όπου στη συνέχεια δημιουργεί ένα Unity3D package, και (β) το ίδιο το Unity3D package που είναι υπεύθυνο για τη δυναμική δημιουργία του 2D/3D παιχνιδιού μέσα στο Unity3D Game Engine. Το πρώτο συστατικό πρόκειται για ένα πρόγραμμα κατασκευασμένο σε C# χρησιμοποιώντας το .NET Framework, ενώ το δεύτερο για ένα Unity3D package κατασκευασμένο πάλι σε C# στο Unity3D game engine.

Κεφάλαιο 2ο: Περιγραφή Μηχανών Ανάπτυξης Παιχνιδιών

Ως μηχανή ανάπτυξης παιχνιδιών (Game Engine) ορίζεται ένα ολοκληρωμένο περιβάλλον ανάπτυξης, με μια έτοιμη σουίτα εργαλείων οπτικής ανάπτυξης και επαναχρησιμοποιήσιμων στοιχείων λογισμικού [1]. Απλοποιεί το πολύπλοκο έργο της ανάπτυξης παιχνιδιών, παρέχοντας ένα στρώμα αφαίρεσης, το οποίο κάνει πολλές μεγάλες εργασίες να φαίνονται πολύ εύκολες, ενώ η μηχανή παιχνιδιού κάνει όλη τη σκληρή δουλειά στο παρασκήνιο. Με άλλα λόγια, είναι ένα πλαίσιο που έχει σχεδιαστεί ειδικά για την κατασκευή και ανάπτυξη βιντεοπαιχνιδιών. Οι μηχανές παιχνιδιών χρησιμοποιούνται συνήθως για τη δημιουργία παιχνιδιών για κονσόλες, κινητές συσκευές και υπολογιστές. Μια μηχανή παιχνιδιών μπορεί να περιλαμβάνει μια μηχανή απόδοσης γραφικών 2D ή 3D που είναι συμβατή με διαφορετικές μορφές εισαγωγής, μια μηχανή φυσικής που προσομοιώνει τις πραγματικές δραστηριότητες, μια τεχνητή νοημοσύνη (AI) που ανταποκρίνεται αυτόματα στις ενέργειες του παίκτη, μια μηχανή ήχου που ελέγχει τα ηχητικά εφέ, μια μηχανή κινουμένων σχεδίων και μια σειρά από άλλα χαρακτηριστικά.

Τα πρώτα βιντεοπαιχνίδια αναπτύχθηκαν με τις δικές τους μηχανές απόδοσης, καθεμία ειδικά σχεδιασμένη για ένα παιχνίδι. Με την πάροδο του χρόνου, οι μηχανές παιχνιδιών εξελίχθηκαν από ιδιόκτητες, εσωτερικές μηχανές σε εμπορικά ανεπτυγμένους κινητήρες που είναι ευρέως διαθέσιμοι σήμερα. Οι προγραμματιστές παιχνιδιών, οι οποίοι έχουν εξαιρετικά υψηλή ζήτηση, μπορούν να απλοποιήσουν και να επιταχύνουν τη διαδικασία ανάπτυξης παιχνιδιών χρησιμοποιώντας εμπορικά ανεπτυγμένες μηχανές παιχνιδιών για την παραγωγή νέων παιχνιδιών ή για την επέκταση των υπαρχόντων παιχνιδιών σε πρόσθετες πλατφόρμες.

Για την υλοποίηση της συγκεκριμένης πτυχιακής επιλέχθηκε η μηχανή ανάπτυξης παιχνιδιών Unity3D. Η επιλογή αυτή δεν έγινε τυχαία αλλά μετά από αναλυτική καταγραφή των πλεονεκτημάτων και μειονεκτημάτων καθώς και των δυνατοτήτων της κάθε μηχανής ανάπτυξης παιχνιδιών, όπως παρατίθενται στις παρακάτω ενότητες.

2.1 Unity3D

Το Unity είναι μια μηχανή παιχνιδιών πολλαπλών πλατφορμών που αναπτύχθηκε από την Unity Technologies, η οποία χρησιμοποιείται κυρίως για την ανάπτυξη βιντεοπαιχνιδιών και προσομοιώσεων για υπολογιστές, κονσόλες και κινητές συσκευές. Ανακοινώθηκε για πρώτη φορά μόνο για το OS X, στο Παγκόσμιο Συνέδριο Προγραμματιστών της Apple το 2005, έκτοτε επεκτάθηκε σε 27 πλατφόρμες [2].

Εικόνα 1: Το λογότυπο του Unity3D

Το Unity είναι μια μηχανή παιχνιδιών για όλες τις χρήσεις που υποστηρίζει γραφικά 2D και 3D, λειτουργίες drag and drop και scripting μέσω C#.

Το Unity είναι ιδιαίτερα δημοφιλές για την ανάπτυξη παιχνιδιών για κινητά και μεγάλο μέρος της εστίασής τους είναι σε πλατφόρμες για κινητές συσκευές. Το 2D pipeline του Unity3D είναι μια πιο πρόσφατη προσθήκη στο game engine και είναι λιγότερο ώριμο από το 3D pipeline. Παρά το γεγονός αυτό, το Unity είναι μια επαρκής πλατφόρμα για την ανάπτυξη παιχνιδιών 2D, ακόμη και σε σύγκριση με άλλες αποκλειστικές μηχανές 2D, ιδιαίτερα εάν το παιχνίδι απευθύνεται σε πολλές φορητές συσκευές.

Το Unity είναι επίσης μια καλή επιλογή για την ανάπτυξη εικονικής πραγματικότητας (Virtual Reality – VR), αν και το VR είναι μια πολύ μικρή αγορά αυτή τη στιγμή. Το market των κινητών και PlayStation VR (PSVR) είναι τα μεγαλύτερα market εικονικής πραγματικότητας και το Unity είναι ήδη σε καλή θέση για να μεταφέρει παιχνίδια σε πολλές πλατφόρμες, όπως PS4 και PC ή market κινητών συσκευών.

Η μηχανή στοχεύει τα ακόλουθα API γραφικών: Direct3D σε Windows και Xbox One, OpenGL σε Linux, macOS και Windows, OpenGL ES σε Android και iOS, WebGL στον Ιστό, και ιδιόκτητα API στις κονσόλες βιντεοπαιχνιδιών.

Επιπλέον, το Unity υποστηρίζει τα χαμηλού επιπέδου API Metal σε iOS και macOS και Vulkan σε Android, Linux και Windows, καθώς και το Direct3D 12 σε Windows και Xbox One. Στα παιχνίδια 2D, το Unity επιτρέπει την εισαγωγή sprites και ένα προηγμένο 2D World Renderer.

Για τρισδιάστατα παιχνίδια, το Unity επιτρέπει την προδιαγραφή της συμπίεσης υφής και των ρυθμίσεων ανάλυσης για κάθε πλατφόρμα που υποστηρίζει η μηχανή παιχνιδιών και παρέχει υποστήριξη για [3]:

- bump mapping: Οι bump maps δημιουργούν την ψευδαίσθηση του βάθους και της υφής στην επιφάνεια ενός τρισδιάστατου μοντέλου χρησιμοποιώντας γραφικά υπολογιστή. Οι υφές δημιουργούνται τεχνητά στην επιφάνεια των αντικειμένων χρησιμοποιώντας κλίμακα του γκρι και απλά κόλπα φωτισμού, αντί να χρειάζεται να δημιουργηθούν με το χέρι μεμονωμένα χτυπήματα και ρωγμές [4].
- reflection mapping: Το reflection mapping όπως και το bump mapping, δημιουργεί την ψευδαίσθηση της ανάκλασης [5].
- parallax mapping: Το parallax mapping είναι μια τεχνική παρόμοια με την bump mapping, αλλά βασίζεται σε διαφορετικές αρχές. Ακριβώς όπως η κανονική χαρτογράφηση, είναι μια τεχνική που ενισχύει σημαντικά τη λεπτομέρεια μιας ανάγλυφης επιφάνειας και της δίνει μια αίσθηση βάθους. Αν και είναι ψευδαίσθηση, το parallax mapping είναι πολύ καλύτερη στη μετάδοση μιας αίσθησης βάθους και μαζί με την κανονική χαρτογράφηση δίνει απίστευτα ρεαλιστικά αποτελέσματα. Αν και το parallax mapping δεν είναι απαραίτητα μια τεχνική που σχετίζεται άμεσα με τον (προχωρημένο) φωτισμό, είναι μια λογική συνέχεια του bump mapping [6].
- screen space ambient occlusion (SSAO): Το εφέ εικόνας Screen Space Ambient Occlusion (SSAO) προσεγγίζει το Ambient Occlusion σε πραγματικό χρόνο, ως εφέ μετά την επεξεργασία εικόνας. Σκουραίνει πτυχές, τρύπες και επιφάνειες που είναι κοντά η μία στην άλλη. Στην πραγματική ζωή, τέτοιες περιοχές τείνουν να αποκλείουν ή να αποφράσσουν το φως του περιβάλλοντος, επομένως φαίνονται πιο σκούρες [7].
- δυναμικές σκιές με χρήση shadow maps: Μια τεχνική που ονομάζεται shadow mapping για την απόδοση σκιών σε πραγματικό χρόνο [8].
- render-to-texture: Το Render-To-Texture είναι μια μέθοδος για τη δημιουργία μιας ποικιλίας εφέ. Η βασική ιδέα είναι η δημιουργία μιας σκηνής, χρησιμοποιώντας textures που μπορούν να χρησιμοποιηθούν ξανά αργότερα [9].

- full-screen εφέ: Το Unity παρέχει μια σειρά από post-processing εφέ και full-screen εφέ μπορούν να βελτιώσουν σημαντικά την εμφάνιση της εφαρμογής σε πολύ μικρό χρόνο. Τα εφέ αυτά χρησιμοποιούνται για προσομοίωση φυσικών ιδιοτήτων κάμερας και φιλμ ή για τη δημιουργία στυλιζαρισμένων γραφικών.

Το Unity προσφέρει τις εξής υπηρεσίες σε προγραμματιστές [10]:

- Unity Ads: Ένα σύστημα ου καθιστά πολύ εύκολη την εισαγωγή διαφημίσεων στα παιχνίδια.
- Unity Analytics: Ένα σύστημα που παρέχει πληροφορίες σχετικά με την απόδοση του παιχνιδιού.
- Unity Certification: Μια πιστοποίηση που παρέχει η Unity3D στους χρήστες που την επιζητούν και την αξίζουν.
- Unity Cloud Build: Ένα σύστημα που επιτρέπει την εκτέλεση των project σε απομακρυσμένους υπολογιστές.
- Unity IAP: Ένα σύστημα που επιτρέπει πολύ εύκολα τους χρήστες του παιχνιδιού να κάνουν in-game αγορές εικονικών αντικειμένων με αντίτιμο πραγματικά χρήματα.
- Unity Multiplayer: Ένα σύστημα που επιτρέπει την δημιουργία multiplayer παιχνιδιών.
- Unity Performance Reporting: Το Unity Performance Reporting καταγράφει και συγκεντρώνει δεδομένα εξαιρέσεων, ώστε να βοηθήσει τον προγραμματιστεί να κατανοήσει τι συμβαίνει κατά τη διάρκεια του χρόνου εκτέλεσης και να βελτιστοποιήσει το έργο του πιο γρήγορα. Οι εξαιρέσεις ενημερώνονται στον Πίνακα ελέγχου υπηρεσιών ενώ το παιχνίδι εκτελείται ως ολοκληρωμένη έκδοση και ενώ βρίσκεται σε λειτουργία αναπαραγωγής εντός του Editor.
- Unity Collaborate: Μια υπηρεσία που διευκολύνει τις προγραμματιστικές ομάδες στο να δημιουργούν project, προσφέροντας προηγμένα εργαλεία κοινοποίησης του κώδικα.

Εκτός από αυτό, το Unity διαθέτει ένα κατάστημα asset όπου η κοινότητα προγραμματιστών μπορεί να κατεβάσει και να ανεβάσει εμπορικούς και δωρεάν πόρους, όπως textures, μοντέλα, πρόσθετα, επεκτάσεις επεξεργασίας, ακόμη και ολόκληρα παραδείγματα παιχνιδιών.

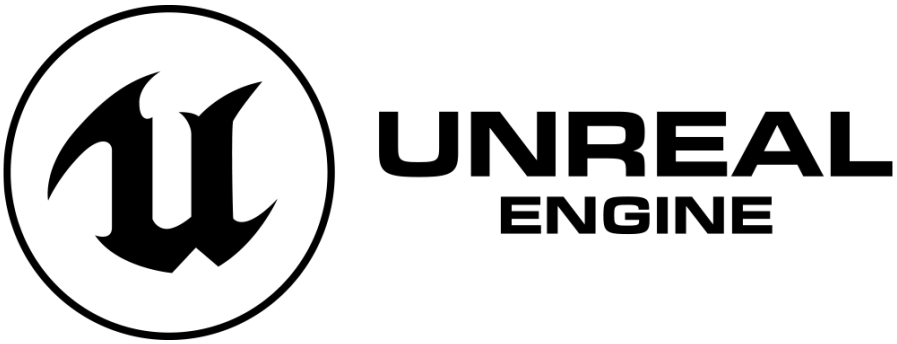
Το Unity είναι αξιοσημείωτο για την ικανότητά του να στοχεύει παιχνίδια για πολλές πλατφόρμες. Οι επί του παρόντος υποστηριζόμενες πλατφόρμες είναι Android, Android TV, Facebook Gameroom, Fire OS, Gear VR, Google Cardboard, Google Daydream, HTC Vive, iOS, Linux, macOS, Microsoft HoloLens, Nintendo 3DS οικογένεια, Nintendo Switch, Oculus Rift, PlayStation 4, PlayStation Vita, PlayStation VR, Samsung Smart TV, Tizen, tvOS, WebGL, Wii U, Windows, Windows Phone, Windows Store και Xbox One.

Το Unity είναι το προεπιλεγμένο εργαλείο ανάπτυξης λογισμικού (software development toolkit - SDK) για την πλατφόρμα κονσόλας βιντεοπαιχνιδιών Wii U της Nintendo, με ένα δωρεάν αντίγραφο που περιλαμβάνεται από τη Nintendo με κάθε άδεια προγραμματιστή Wii U. H Unity Technologies αποκαλεί αυτήν τη ομαδοποίηση ενός third-party SDK ως “industry first”.

2.2 Unreal Engine

Μία από τις πιο δημοφιλείς και ευρέως χρησιμοποιούμενες μηχανές παιχνιδιών είναι η Unreal Engine, η οποία ανήκει στην Epic Games. Πρόκειται ουσιαστικά για μια μηχανή ανάπτυξης παιχνιδιών πολλαπλών πλατφορμών που έχει σχεδιαστεί για επιχειρήσεις όλων των μεγεθών που βοηθά στη χρήση τεχνολογίας σε πραγματικό χρόνο για τη μετατροπή των ιδεών σε ελκυστικό οπτικό περιεχόμενο [11]. Αρχικά αναπτύχθηκε για shooters πρώτου προσώπου PC, έκτοτε έχει χρησιμοποιηθεί σε διάφορα είδη τρισδιάστατων (3D) παιχνιδιών και έχει υιοθετηθεί από άλλες

βιομηχανίες, κυρίως από τη βιομηχανία του κινηματογράφου και της τηλεόρασης. Γραμμένο σε C++, το Unreal Engine διαθέτει υψηλό βαθμό φορητότητας, υποστηρίζοντας ένα ευρύ φάσμα πλατφορμών για επιτραπέζιους υπολογιστές, κινητά, κονσόλες και VR.



Εικόνα 2: Το λογότυπο του Unreal Engine

Η αρχική έκδοση κυκλοφόρησε το 1998 και 19 χρόνια αργότερα συνεχίζει να χρησιμοποιείται για μερικά από τα μεγαλύτερα παιχνίδια. Η δύναμη του Unreal Engine είναι η ικανότητά του να τροποποιείται αρκετά ώστε τα παιχνίδια να μπορούν να μετατραπούν σε πολύ μοναδικές εμπειρίες. Ωστόσο, αυτό απαιτεί εξειδικευμένους προγραμματιστές με τεράστια εμπειρία [12].

Επιπλέον, η Epic Games εξαγόρασε την Quixel, η οποία διαθέτει μια τεράστια βιβλιοθήκη στοιχείων «φωτογραμμετρίας» [13] με εικόνες πραγματικού κόσμου που μπορούν να χρησιμοποιηθούν για τη δημιουργία κινούμενων εικόνων και βιντεοπαιχνιδιών. Οι χρήστες του Unreal Engine θα μπορούν να χρησιμοποιούν δωρεάν τα εργαλεία της Quixel (Bridge, Mixer) και όλα τα στοιχεία της βιβλιοθήκης Megascans της Quixel.

2.3 CryEngine

Το CryEngine, που παρουσιάστηκε για πρώτη φορά από τη μεγάλη εταιρεία ανάπτυξης, Crytek, στο πρώτο παιχνίδι Far Cry, είναι σίγουρα μια από τις πιο ισχυρές και κυρίαρχες μηχανές παιχνιδιών σήμερα [14]. Αυτό που κάνει το CryEngine αντάξιο των υπολοίπων μηχανών κατασκευής παιχνιδιών είναι οι γραφικές του ικανότητες που ξεπερνούν τις γραφικές δυνατότητες του Unity και είναι ισοδύναμες με αυτές που διαθέτει το Unreal. Αν και είναι μια βαριά και ισχυρή μηχανή παιχνιδιών, για να χρησιμοποιήσει κανείς το CryEngine αποτελεσματικά, χρειάζεται λίγο χρόνο και θεωρείται λίγο πιο δύσκολο για τους αρχάριους που δεν έχουν χρησιμοποιήσει άλλες μηχανές παιχνιδιών εκ των προτέρων.

Εικόνα 3: Το λογότυπο του CryENGINE

Το CryEngine υποστηρίζει την εικονική πραγματικότητα και έχει εκπληκτικά οπτικά εφέ, συμπεριλαμβανομένου του ογκομετρικού συστήματος ομίχλης και απόδοσης σύννεφων, το οποίο δίνει στα σύννεφα μια πλήρη 3D χωρική απόδοση και μια ρεαλιστική απεικόνιση για εφέ ομίχλης και καιρού. Επιπλέον, το καλύτερο μέρος της επιλογής της πλατφόρμας CryEngine για τη σχεδίαση παιχνιδιών είναι ότι δεν απαιτεί από τους χρήστες της να πληρώσουν το τέλος δικαιωμάτων κατά το σχεδιασμό του παιχνιδιού. Ωστόσο, πρέπει να πληρώσουν ένα σταθερό ποσό, δηλαδή 9,90 $ /- το μήνα για την απόκτηση πρόσβασης στο CryEngine. Επιπλέον, το CryEngine διαθέτει ένα αφιερωμένο φόρουμ Q&A που ονομάζεται CryEngine Answers, το οποίο περιλαμβάνει όλες τις αμφιβολίες και τις απορίες του χρήστη παρέχοντας καλύτερη εμπειρία κατά τη χρήση του.

2.4 Amazon Lumberyard

Το Lumberyard ήταν μια μηχανή παιχνιδιών beta AAA που έκτοτε αντικαταστάθηκε από το Open 3D Engine. Βασίζεται στο CryEngine της Crytek, αλλά η ομάδα προγραμματιστών στο Amazon έχει ήδη κάνει σημαντικές ενημερώσεις στο σύστημα. Οι συνεχείς αναβαθμίσεις από την πλευρά της Amazon θα φέρουν το Lumberyard πιο μακριά από το CryEngine. Για τον προγραμματισμό των παιχνιδιών μπορούν να χρησιμοποιηθούν οι γλώσσες προγραμματισμού C++ και Lua [15].

Εικόνα 4: Το λογότυπο του Amazon Lumberyard

Η οπτική τεχνολογία του Lumberyard βασίζεται στο CryEngine, που σημαίνει τρισδιάστατα περιβάλλοντα και μια ολόκληρη σειρά από εφέ σε πραγματικό χρόνο. Παρέχει εργαλεία για: καιρικά εφέ, πλαίσια κάμερας, φυσική υφάσματος, επεξεργαστές χαρακτήρων και animations, πρόγραμμα επεξεργασίας σωματιδίων, επεξεργαστής διεπαφής χρήστη κ.α..

Η Amazon περιλαμβάνει επίσης τον πηγαίο κώδικα για το Lumberyard. Οι ομάδες προγραμματιστών μπορούν να πάρουν τον πηγαίο κώδικα και να προσαρμόσουν τη μηχανή για να βελτιστοποιήσουν την ανάπτυξη παιχνιδιών για την ομάδα τους. Εκτός από τη βελτιστοποίηση, οι προγραμματιστές μπορούν να προσαρμόσουν τον πηγαίο κώδικα για να επηρεάσουν το gameplay, δηλ. τον τρόπο με τον οποίο οι παίκτες αλληλεπιδρούν με το παιχνίδι, όπως για παράδειγμα οι κανόνες (rules).

Το Lumberyard είναι μια δωρεάν μηχανή παιχνιδιού για χρήση. Δεν υπάρχουν δικαιώματα. Δεν υπάρχουν τέλη δανειοδότησης. Δεν υπάρχουν προκαταβολικές αγορές του λογισμικού. Εκεί που η Amazon βγάζει χρήματα είναι μέσω του Amazon Web Services (AWS), που είναι θυγατρική της Amazon που παρέχει κατ' απαίτηση πλατφόρμες υπολογιστικού νέφους και API σε ιδιώτες, εταιρείες και κυβερνήσεις. Φυσικά, η μηχανή παιχνιδιού έχει σχεδιαστεί για εύκολη συμβατότητα με την υπηρεσία Web της Amazon. Η Amazon έχει ξεκαθαρίσει ότι οι προγραμματιστές πληρώνουν μόνο για ό,τι χρειάζονται όπως το χρειάζονται.

Εκτός από την απρόσκοπτη ενσωμάτωση στο AWS, το Lumberyard περιλαμβάνει επίσης ενσωμάτωση Twitch, όπου με τη λειτουργία Twitch ChatPlay οι προγραμματιστές μπορούν εύκολα να ενσωματώσουν αλληλεπιδράσεις σε πραγματικό χρόνο με τους θεατές στο Twitch στο παιχνίδι. Αυτό θα μπορούσε να είναι για παράδειγμα εντολές συνομιλίας.

Ένα ακόμα σημαντικό χαρακτηριστικό είναι το γεγονός ότι μέρη του πηγαίου κώδικα μπορούν ακόμη και να συμπεριληφθούν στο τελικό παιχνίδι, επιτρέποντας στους παίκτες να το τροποποιήσουν όπως επιθυμούν. Επιπλέον, η ενσωμάτωση του AWS σημαίνει ότι ο κατασκευαστής του παιχνιδιού μπορεί πολύ εύκολα με την βοήθεια των υπηρεσιών αυτών της Amazon, να προσθέσει online λειτουργείες στο παιχνίδι του.

2.5 Godot

Το Godot πρόκειται για μια ακόμα ευέλικτη μηχανή κατασκευής παιχνιδιών, που δίνει την δυνατότητα κατασκευής τόσο τρισδιάστατων όσο και δισδιάστατων παιχνιδιών. Ο προγραμματισμός στο Godot γίνεται είτε σε GDScript (μία Python-like γλώσσα προγραμματισμού σχεδιασμένη για να δημιουργεί παιχνίδια με μηδενική ταλαιπωρία), είτε σε C# είτε σε C++, καθώς παρέχει και τη δυνατότητα visual scripting με χρήση nodes. Η κοινότητα του Godot έχει αναπτύξει επίσης υποστήριξη και για άλλες γλώσσες προγραμματισμού όπως Rust, Nim, D κ.α.. Είναι ανοιχτού κώδικα και όλη η δουλειά που αναπτύσσεται ανήκει αποκλειστικά στον δημιουργό. Παρέχονται επίσης όλα τα δικαιώματα πάνω σε αυτό χωρίς να χρειαστεί πληρωμή τελών/ δικαιωμάτων στο μέλλον. Αυτή είναι μια εξαιρετική επιλογή για έναν προγραμματιστή indie παιχνιδιών, αλλά πρέπει να ληφθεί υπόψη ότι δεν υπάρχει τόση τεκμηρίωση, οδηγοί/βίντεο όσο στο Unity ή το Unreal, καθώς η κοινότητα είναι πολύ μικρότερη για αυτήν τη μηχανή.

Εικόνα 5: Το λογότυπο του Godot

Το Godot συνοδεύεται από εκατοντάδες ενσωματωμένους κόμβους που κάνουν το σχεδιασμό παιχνιδιών πολύ εύκολο. Δίνει επίσης την δυνατότητα για δημιουργία [16]:

- custom behaviors: Η δυνατότητα δημιουργίας νέων behaviors, χρησιμοποιώντας τα ίδια events με αυτά που χρησιμοποιήθηκαν για την δημιουργία των κανώνων ενώς παιχνιδιού.
- custom editors: Custom παράθυρα επεξεργασίας του περιβάλλοντος του παιχνιδιού, όπως επιθυμεί ο χρήστης.
- node compositions με υποστήριξη για instancing και κληρονομικότητα: Η δυνατότητα του godot για την δημιουργία προγραμματιστικών κόμβων οι οποίοι μπορούν να αντιστοιχιστούν ως childs ενός άλλου κόμβου, με αποτέλεσμα μια διάταξη δέντρου. Ένας κόμβος μπορεί να περιέχει οποιονδήποτε αριθμό κόμβων ως παιδιά με την απαίτηση ότι όλα τα αδέρφια (άμεσα παιδιά ενός κόμβου) πρέπει να έχουν μοναδικά ονόματα

Παρέχει οπτικό επεξεργαστή με όλα τα απαραίτητα εργαλεία συγκροτημένα σε ένα context-sensitive UI. Σημαντικό χαρακτηριστικό είναι επίσης η παροχή pipeline για δημιουργία περιεχομένου για καλλιτέχνες, σχεδιαστές πίστας, animators και όλους τους ενδιάμεσους. Ένα άλλο χαρακτηριστικό είναι η ζωντανή επεξεργασία της σκηνής όπου οι αλλαγές δεν χάνονται μετά τη διακοπή του παιχνιδιού, ένα χαρακτηριστικό που λειτουργεί ακόμη και σε φορητές συσκευές. Επιτρέπει επίσης και την δημιουργία προσαρμοσμένων εργαλείων με ευκολία χρησιμοποιώντας το ενσωματωμένο σύστημα εργαλείων που παρέχεται.

2.6 OGRE

Το OGRE (Object-Oriented Graphics Rendering Engine) είναι μια scene-oriented, ευέλικτη 3D μηχανή γραμμένη σε C++ που έχει σχεδιαστεί για να διευκολύνει τους προγραμματιστές να παράγουν εφαρμογές χρησιμοποιώντας hardware-accelerated τρισδιάστατα γραφικά [17]. Η βιβλιοθήκη κλάσης αφαιρεί όλες τις λεπτομέρειες της χρήσης των υποκείμενων βιβλιοθηκών του συστήματος όπως το Direct3D και το OpenGL και παρέχει μια διεπαφή βασισμένη σε αντικείμενα κόσμου και άλλες έξυπνες κλάσεις. Το OGRE εξακολουθεί να χρησιμοποιείται, αλλά όχι τόσο όσο άλλα, αν και υπάρχει εδώ και αρκετό καιρό. Είναι μια μηχανή προγραμματισμού πιο χαμηλού επιπέδου, που σημαίνει ότι θα πρέπει να πραγματοποιηθεί από τον χρήστη περισσότερη κωδικοποίηση, σε αντίθεση με το Unity ή το Unreal που παρέχουν έτοιμα features, αποσυμφορώντας τον προγραμματιστή.

Εικόνα 6: Το λογότυπο του ORGE

Αυτό συμβαίνει επειδή το OGRE είναι κατά κύριο λόγο μια μηχανή rendering και δεν είναι μια πλήρης μηχανή παιχνιδιών όπως οι άλλες μηχανές παιχνιδιών σε αυτήν τη λίστα.

2.7 Σύγκριση μηχανών κατασκευής παιχνιδιών

Εφόσον έχουν αναφερθεί τα βασικά χαρακτηριστικά της κάθε μηχανής παιχνιδιών, πραγματοποιείται παρακάτω μία σύγκριση αυτών, εξηγώντας έτσι έμμεσα τον λόγο για τον οποίο έγινε επιλογή του Unity3D για την δημιουργία του UnitySceneGenerator εργαλείου της πτυχιακής αυτής.

Ξεκινώντας λοιπόν, συγκρίνοντας την απόδοση του Unity με το Unreal, καταλάβαμε ότι το Unity είναι καλύτερη πλατφόρμα για την ανάπτυξη παιχνιδιών για κινητά και 2D/3D, ενώ το Unreal είναι το καταλληλότερο για την ανάπτυξη παιχνιδιών με υψηλά γραφικά και φωτορεαλιστικά χαρακτηριστικά. Αυτό αποδεικνύεται ότι είναι η μεγαλύτερη διαφορά μεταξύ Unreal και Unity [18].

Το CryEngine, από την άλλη πλευρά, διαθέτει επίσης τις δυνατότητες δημιουργίας παιχνιδιών υψηλών γραφικών. Επιπλέον, συγκρίνοντας το CryEngine με το Unreal 2018 στην κλίμακα παροχής χαρακτηριστικών της πλατφόρμας επόμενης γενιάς με πιο ελκυστικό μοντέλο τιμολόγησης, το CryEngine σίγουρα ξεπερνά το Unreal με την οικονομική του δομή.

Όπως προαναφέρθηκε, το Unity3D φαίνεται να αποτελεί την καλύτερη επιλογή για αρχάριους. Η πολύ καλογραμμένη τεκμηρίωση, πολλά documentations και έτοιμα πρότυπα επιτρέπουν την εξερεύνηση των ποιο σημαντικών λειτουργειών του αρκετά γρήγορα. Επιπλέον, ο μεγαλύτερος αριθμός υποστηριζόμενων πλατφορμών και η μεγάλη ευελιξία του κινητήρα επιτρέπει την υλοποίηση οποιουδήποτε είδους παιχνιδιών. Το Unity είναι η μόνη μηχανή η οποία, πέρα από 3D, παρέχει επίσης αποκλειστική λειτουργία κατασκευής 2D παιχνιδιών, επομένως θα είναι η καλύτερη επιλογή για χρήστες που θέλουν να δημιουργήσουν ένα παιχνίδι δύο διαστάσεων. Το μειονέκτημα αυτής της μηχανής είναι αναμφίβολα ο σχετικά μικρός αριθμός εξειδικευμένων Editors. Τα modules που επιτρέπουν το animation, τη δημιουργία βίντεο ή την προσθήκη ηχητικών εφέ σε σύγκριση με τον ανταγωνισμό, επιτρέπουν μόνο βασικές λειτουργίες. Η μηχανή αυτή μπορεί να επεκταθεί χρησιμοποιώντας πρόσθετα και βιβλιοθήκες, αλλά πολλά από αυτά είναι επί πληρωμή, γεγονός που μπορεί να αποθαρρύνει τους πιθανούς χρήστες. Όσον αφορά την ποιότητα των γραφικών, το Unity φαίνεται να είναι το πιο αδύναμο από τα άλλα engines. Φυσικά, είναι δυνατό να δημιουργηθούν

παιχνίδια εξαιρετικής εμφάνισης με αυτό, αλλά οι μηχανές Unreal και CryEngine είναι πιο κατάλληλες για την απόδοση γραφικών. Ένα δοκιμαστικό παιχνίδι που εφαρμόστηκε στο Unity το 2019 [19], πήγε αρκετά καλά κατά τη διάρκεια των δοκιμών απόδοσης. Η εφαρμογή του παιχνιδιού σε αυτή τη μηχανή επωφελήθηκε περισσότερο από τη μείωση της ποιότητας γραφικών, η οποία επηρεάστηκε σε μεγάλο βαθμό από την πολιτική των προεπιλεγμένων ρυθμίσεων προγραμματιστών. Στη μηχανή Unity, μαζί με τη μείωση των ρυθμίσεων γραφικών, ορισμένοι μηχανισμοί γραφικών απενεργοποιούνται πλήρως, κάτι που δεν συνιστάτε η συμβαίνει σε άλλες μηχανές παιχνιδιών, όπου προτιμάται αντ' αυτού η προσέγγιση που συνίσταται στη μείωση της λεπτομέρειας των γραφικών εφέ που δημιουργούνται από τους μηχανισμούς αυτούς. Το παιχνίδι που υλοποιήθηκε στο Unity έχασε το μεγαλύτερο μέρος της απόδοσής του με την αύξηση της δασικής κάλυψης, γεγονός που μπορεί να υποδηλώνει ότι το συγκεκριμένο game engine μπορεί να μην είναι κατάλληλο για την υλοποίηση παιχνιδιών με έναν μεγάλο και πολύπλοκο κόσμο. Τα προβλήματα με την απόδοση του Unity επιβεβαιώνονται επίσης από το γεγονός ότι το μέσο frame rate μειώνεται σχετικά ελαφρά με τη μείωση της ανάλυσης της οθόνης.

Το Unreal Engine, σε αντίθεση με το Unity, δεν θα είναι απαραίτητα η σωστή επιλογή για αρχάριους. Το τεράστιο εύρος των διαθέσιμων επιλογών και προγραμμάτων επεξεργασίας μπορεί να είναι συντριπτικό κατά την πρώτη επαφή με αυτό το εργαλείο. Το Unreal Engine δίνει περισσότερη ελευθερία στους προγραμματιστές, και ως εκ τούτου αυτού του είδους η λειτουργία πρέπει να γίνεται από τους ίδιους τους χρήστες της μηχανής. Ωστόσο, το Unreal Engine μπορεί να είναι μια καλή επιλογή για χρήστες χωρίς δεξιότητες προγραμματισμού. Το συγγραφικό εργαλείο BluePrints επιτρέπει την δημιουργία ολόκληρης της λογικής του παιχνιδιού με τη χρήση γραφικών διαγραμμάτων, τα οποία μπορούν να αντικαταστήσουν πλήρως την ανάγκη εγγραφής κώδικα σε γλώσσα C++. Το Unreal Engine διακρίνεται για την υποστήριξή του σε πολλές πλατφόρμες υλικού-στόχου, την ικανότητα δημιουργίας γραφικών υψηλής ποιότητας, ένα αποδεδειγμένο και σχετικά εύχρηστο multiplayer σύστημα, έναν εξαιρετικό επεξεργαστή materials, την ικανότητα δημιουργίας προηγμένων animation, δημιουργία σκελετών για τους χαρακτήρες και συστήματα που υποστηρίζουν την εφαρμογή συμπεριφοράς τεχνητής νοημοσύνης. Σε δοκιμές απόδοσης, το Unreal Engine δεν είχε καλή απόδοση. Στα περισσότερα πειράματα, το παιχνίδι που εφαρμόστηκε σε αυτή τη μηχανή παιχνιδιών ήταν το λιγότερο αποτελεσματικό. Σε αντίθεση με το Unity, θα πρέπει να σημειωθεί, ωστόσο, ότι με την αύξηση των δασών, το παιχνίδι που γράφτηκε σε Unreal Engine έχασε τη μικρότερη απόδοση [19]. Η σωστή βελτιστοποίηση αυτού της μηχανής αποδεικνύεται επίσης από τη μεγαλύτερη αύξηση στην απόδοση καθώς μειώνεται η ανάλυση των γραφικών που εμφανίζονται. Τα σχετικά φτωχά αποτελέσματα που λήφθηκαν κατά τη διάρκεια των πειραμάτων μπορεί να υποδεικνύουν ότι η μηχανή δεν είναι καλά προσαρμοσμένη για να λειτουργεί σε ασθενέστερα υλικά συστήματα.

Το CryEngine είναι ένα εργαλείο που σίγουρα δεν μπορεί να συνιστάται σε άτομα χωρίς εμπειρία στη δημιουργία παιχνιδιών. Ένας μικρός αριθμός εκπαιδευτικών προγραμμάτων και η μάλλον κακώς γραπτή τεκμηρίωση είναι απίθανο να βοηθήσουν στην προσέλκυση νέων χρηστών. Σε σύγκριση με άλλα game engines, είναι επίσης ένα μάλλον ασήμαντο καθολικό εργαλείο, που επικεντρώνεται κυρίως στη δημιουργία παιχνιδιών First Person Shooter - FPS, πιθανώς παιχνιδιών σχετικών ειδών. Παρόλες τις ατέλειες του CryEngine, είναι ένα εργαλείο που δίνει τη δυνατότητα δημιουργίας γραφικών υψηλής ποιότητας, παρέχοντας πολλές μοναδικές λειτουργίες (π.χ. εξαιρετικός επεξεργαστής εδάφους, σύστημα animation εικονικών χαρακτήρων), συστήματα υψηλής ποιότητας που επιτρέπουν την εφαρμογή των συμπεριφορών των Non-Playing Character - NPC και πρόγραμμα επεξεργασίας που επιτρέπουν τον σχεδιασμό τρισδιάστατων αντικειμένων απευθείας στο engine. Στις

δοκιμές απόδοσης που εκτελέστηκαν από τους Andrzej BARCZAK και Hubert WOŹNIAK το 2019 [19], το παιχνίδι που υλοποιήθηκε στη μηχανή CryEngine με τις υψηλότερες ρυθμίσεις γραφικών και την υψηλότερη κάλυψη δασών λειτούργησε πιο αποτελεσματικά.

Το Amazon Lumberyard, Godot και ORGE από την άλλη είναι εντελώς δωρεάν, σε αντίθεση με τα προηγούμενα game engines που με τον έναν ή τον άλλο τρόπο είχαν μια τιμή στα προϊόντα ή υπηρεσίες τους. Και τα τρία παρέχουν υποστήριξη για την δημιουργία 2D γραφικών με εξαίρεση το Amazon Lumberyard το οποίο παρέχει υποστήριξη μόνο για 3D. Κυρίαρχο χαρακτηριστικό του Lumberyard, σε αντίθεση με το Godot και το ORGE, είναι η υποστήριξή που παρέχει για virtual reality σε συνδυασμό με την ευκολία χρήσης του ενώ του ORGE η ευελιξία του, καθώς παρέχει την δυνατότητα για προγραμματισμό χαμηλού επιπέδου σε C++. Κύριο χαρακτηριστικό του Godot από την άλλη είναι η υποστήριξη πελατών που παρέχει καθώς και η δυνατότητα προγραμματισμού σε πέντε γλώσσες: GDScript, C#, VisualScript, C++ και C.

Τα κύρια μειονεκτήματα των τριών τελευταίων γλωσσών (Amazon Lumberyard, Godot και ORGE) σε σχέση με των τριών προηγούμενων (Unity, Unreal Engine και CryENGINE), είναι η διαθεσιμότητα εκπαιδευτικού υλικού σχετικά με αυτά στο διαδίκτυο, ένα σημαντικό μειονέκτημα καθώς αποθαρρύνει πολλούς χρήστες από το να ασχοληθούν με την εκμάθησή τους, φοβούμενοι ότι δεν θα βρουν εύκολα λύση στα προβλήματά τους. Επίσης η έλλειψη συμβατότητας με πολλά διαφορετικά συστήματα και πλατφόρμες, αποθαρρύνει επίσης πολλούς χρήστες, οι οποίοι στοχεύουν στην κατηγορία αυτή, από το να τα χρησιμοποιήσουν.

Στον παρακάτω πίνακα έχει γίνει μια συγκεντρωτική μελέτη των μειονεκτημάτων και των πλεονεκτημάτων της κάθε μηχανή κατασκευής παιχνιδιών, και έχουν αποτυπωθεί τα αποτελέσματά τους.

	Ιδιοκτησία και τιμολόγηση	2D/3D	Ευκολία χρήσης	Ενσωμάτωση και συμβατότητα	Ευελιξία	Υλικό Εκμάθησης	Υποστήριξη VR	Υποστήριξη πελατών	Γλώσσες Προγραμματισμού
Unity3D	**	Both	*****	*****	****	*****	Ναι	****	C#
Unreal Engine	***	Both	*****	******	****	****	Ναι	***	C++ Python
CryENGINE	***	Both	****	******	****	***	Ναι	***	C++ C# Lua
Amazon Lumberyard	*****	3D	****	***	****	***	Ναι	***	C++ Lua

Godot	*****	Both	****	***	****	**	Όχι	*****	GDScript C# VisualScript C++ C
ORGE	*****	Both	**	***	*****	*	Όχι	****	C++

Table 1: Πίνακας σύγκρισης των διαφορετικών μηχανών κατασκευής παιχνιδιών

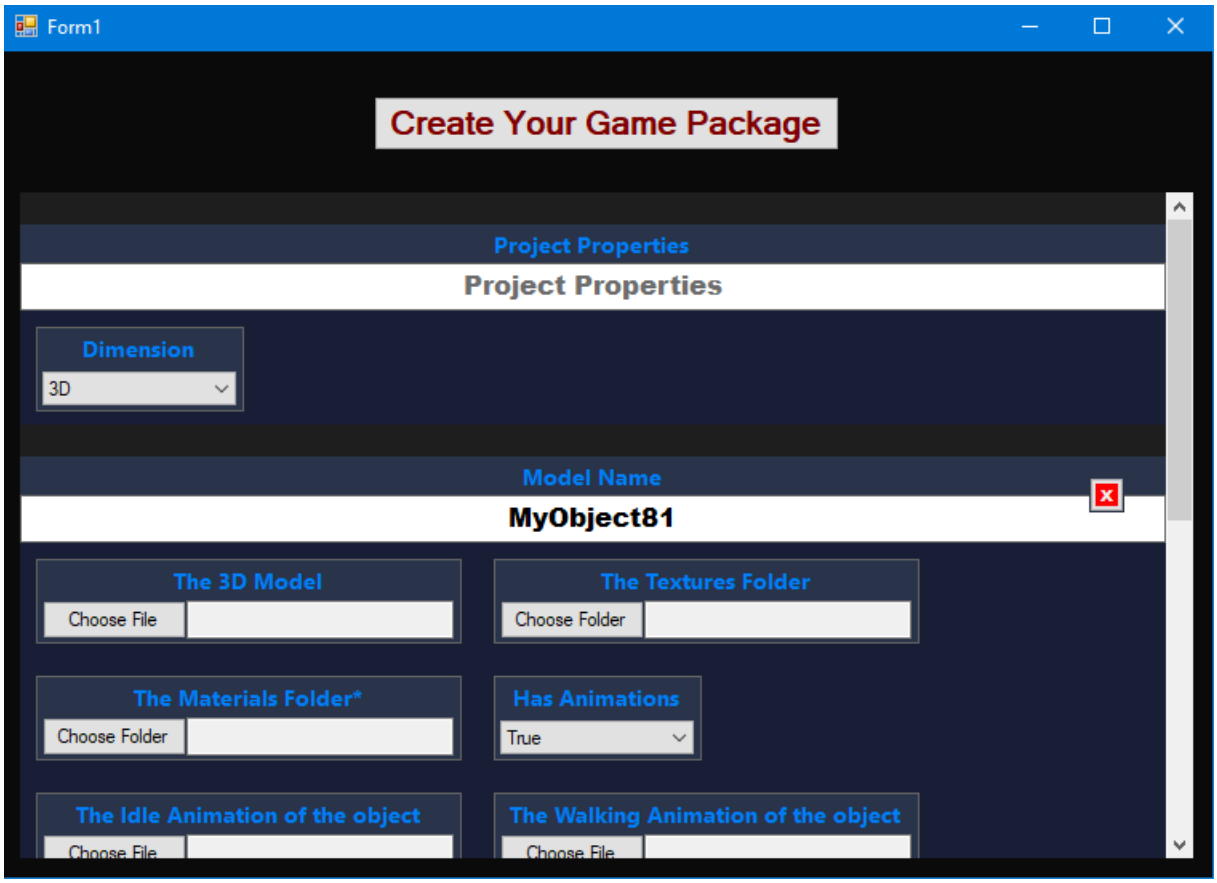
Καταλήγοντας λοιπόν, όπως φαίνεται και από τον παραπάνω πίνακα, το Unity3D υπερέχει, κατά μέσο όρο, των άλλων μηχανών, καθιστώντας το κατάλληλο για την συγκεκριμένη εργασία, μιας και η δυναμική δημιουργία παιχνιδιού απαιτεί ένα σύνολο των χαρακτηριστικών που αναφέρονται, και όχι έμφαση σε ένα από αυτά απαραίτητα. Το ποιο σημαντικό χαρακτηριστικό όμως που λήφθηκε υπ' όψιν ήταν η ποσότητα υλικού βοήθειας και εκμάθησης που υπάρχει διαθέσιμο στο διαδίκτυο, πράγμα που κατέστησε την υλοποίηση του συγκεκριμένου project πολύ πιο γρήγορη και εύκολη.

Κεφάλαιο 3ο: Σχεδίαση και Ανάπτυξη του Εργαλείου

3.1 Το εργαλείο UnityPackageGenerator

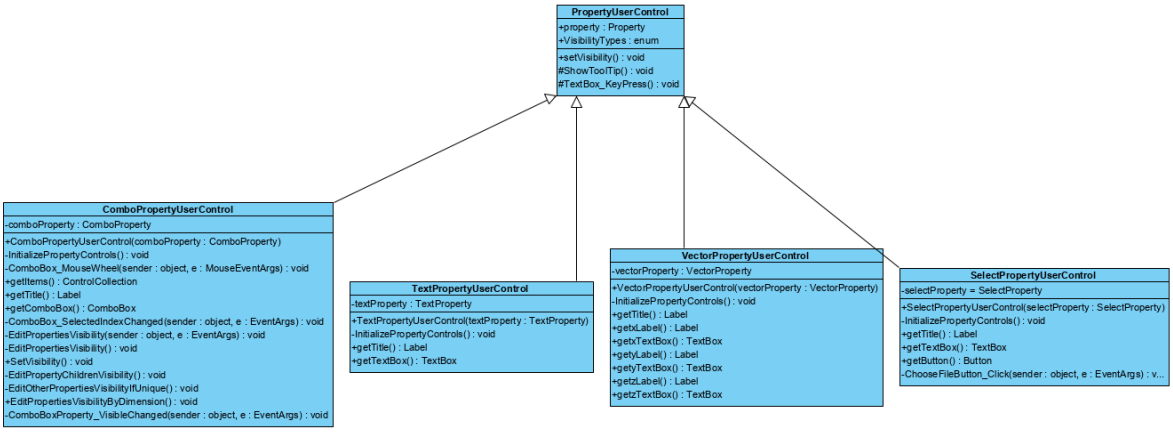
Στο κεφάλαιο αυτό παρουσιάζεται ο τρόπος σχεδίασης και ανάπτυξης του εργαλείου UnityPackageGenerator, το οποίο συλλέγει τις πληροφορίες του χρήστη σχετικά με τα αντικείμενα μιας σκηνής, καθώς και τα αρχεία αυτής δημιουργώντας δυναμικά στη συνέχεια το εργαλείο δυναμικής παραγωγής παιχνιδιού “JsonSceneGenerator”, με τη μορφή Unity Package. Το UnityPackageGenerator έχει αναπτυχθεί σε C# με χρήση του .NET Framework.

Το .NET Framework (προφέρεται ως "dot net") είναι ένα ιδιόκτητο πλαίσιο λογισμικού που αναπτύχθηκε από τη Microsoft και εκτελείται κυρίως σε Microsoft Windows. Ήταν η κυρίαρχη εφαρμογή της Κοινής Γλωσσικής Υποδομής (Common Language Infrastructure - CLI) έως ότου αντικατασταθεί από το έργο cross-platform .NET. Περιλαμβάνει μια μεγάλη βιβλιοθήκη κλάσης που ονομάζεται Framework Class Library (FCL) και παρέχει γλωσσική διαλειτουργικότητα (κάθε γλώσσα μπορεί να χρησιμοποιήσει κώδικα γραμμένο σε άλλες γλώσσες) σε πολλές γλώσσες προγραμματισμού. Τα προγράμματα που είναι γραμμένα για .NET Framework εκτελούνται σε περιβάλλον λογισμικού (σε αντίθεση με το περιβάλλον υλικού) που ονομάζεται Common Language Runtime (CLR). Το CLR είναι μια εικονική μηχανή εφαρμογών που παρέχει υπηρεσίες όπως ασφάλεια, διαχείριση μνήμης και διαχείριση εξαιρέσεων. Ως εκ τούτου, ο κώδικας υπολογιστή που γράφεται με χρήση .NET Framework ονομάζεται "διαχειριζόμενος κώδικας". Το FCL και το CLR μαζί αποτελούν το .NET Framework. Το FCL παρέχει διεπαφές χρήστη, πρόσβαση σε δεδομένα, συνδεσιμότητα βάσεων δεδομένων, κρυπτογραφία, ανάπτυξη διαδικτυακών εφαρμογών, αριθμητικούς αλγόριθμους και επικοινωνίες δικτύου. Οι προγραμματιστές παράγουν λογισμικό συνδυάζοντας τον πηγαίο κώδικα τους με το .NET Framework και άλλες βιβλιοθήκες. Το πλαίσιο προορίζεται να χρησιμοποιηθεί από τις περισσότερες νέες εφαρμογές που δημιουργούνται για την πλατφόρμα Windows. Η Microsoft παράγει επίσης ένα ενσωματωμένο περιβάλλον ανάπτυξης για λογισμικό .NET που ονομάζεται Visual Studio, πάνω στο οποίο αναπτύχθηκε και το UnityPackageGenerator εργαλείο.



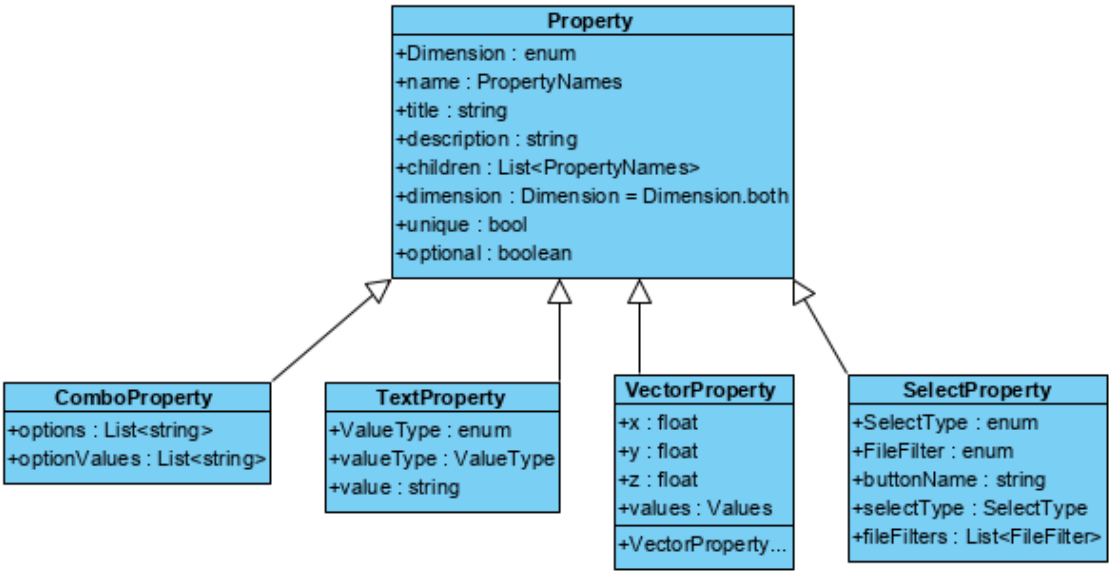
Εικόνα 7: Η διεπαφή του εργαλείου UnityPackageGenerator

Όπως απεικονίζεται παραπάνω, η διεπαφή του εργαλείου πρόκειται για μια προσομοίωση των αντικειμένων του αρχείου Json που θα περιλαμβάνει πληροφορίες σχετικά με τα αντικείμενα που θα εισαχθούν στη σκηνή. Πρόκειται για ένα σύνολο panels/categories από τα οποία το πρώτο είναι έχει πάντα το όνομα Project Properties ενώ τα υπόλοιπα έχουν το όνομα του μοντέλου που θα επιλέξει ο χρήστης να εισάγει. Όπως είναι αυτονόητο, το πρώτο category θα περιέχει τις πληροφορίες του αντικειμένου projectProperties του αρχείου Json, ενώ όλα τα υπόλοιπα τις πληροφορίες των αντικειμένων. Ο χρήστης έχει την δυνατότητα να προσθέσει κι άλλα panels αντικειμένων πατώντας το κουμπί “Add Another Object” που βρίσκεται στο τέλος κάθε category. Κάθε category με τη σειρά του περιέχει μια συλλογή τεσσάρων UserControls (ένα για κάθε είδους property αντικειμένου), που ονομάζονται PropertyUserControls. Αυτά τα τέσσερα UserControls κληρονομούν από την υπερκλάση PropertyUserControl. Στην παρακάτω εικόνα φαίνεται ποιο αναλυτικά η δομή των κλάσεων PropertyUserControls μέσω ενός UML διαγράμματος.



Εικόνα 8: Το UML διάγραμμα της κλάσης PropertyUserControls και των υποκλίσεών της

Όπως φαίνεται και στην εικόνα το κάθε ένα από αυτά τα PropertyUserControls κατέχει ένα αντικείμενο/μεταβλητή, comboProperty, textProperty, VectorProperty ή selectProperty, το οποίο κατά τη δημιουργία του userControl θα πάρει ως τιμή ένα αντικείμενο της κλάσης Property. Τα αντικείμενα της κλάσης Property παρέχουν πληροφορίες σχετικά με την εκάστοτε ιδιότητα του Json αντικειμένου που προστίθεται, όπως position, scale, hasPhysics κ.λπ.. Περισσότερες πληροφορίες σχετικά με την κλάση Property, αναγράφονται παρακάτω μέσω ενός UML διαγράμματος.

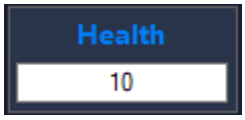


Εικόνα 9: Το UML διάγραμμα της κλάσης Property και των υποκλίσεών της

Έχοντας περιγράψει τη δομή των κλάσεων, ακολουθεί περεταίρω ανάλυσή τους όπως περιγραφή των μεθόδων και της λειτουργικότητας του καθενός.

Η κλάση TextBoxUserControl

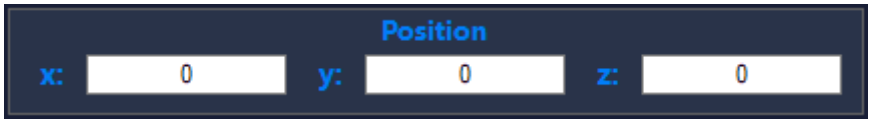
Η κλάση αυτή δημιουργεί ένα UserControl το οποίο θα περιέχει ένα απλό text box. Δημιουργείται προκειμένου να αναπαραστήσει ιδιότητες όπως το όνομα του αντικειμένου ή τη ζωή του. Σημαντικό χαρακτηριστικό του είναι το γεγονός ότι επιτρέπει στον χρήστη να εισάγει μόνο έγκυρους χαρακτήρες. Το τι εστί έγκυρος χαρακτήρας το ορίζει η μεταβλητή valueType του αντικειμένου textProperty. Για παράδειγμα άμα η μεταβλητή έχει οριστεί ως "TextProperty.ValueType.numeric" τότε ένα custom event θα προστεθεί στην κλάση αυτή το οποίο θα αποτρέψει το χρήστη να εισάγει οποιονδήποτε άλλο χαρακτήρα με το πληκτρολόγιο παρά αριθμούς και υποδιαστολές. Ένα παράδειγμα ενός τέτοιου TextBoxUserControl απεικονίζεται παρακάτω.



Εικόνα 10: Παράδειγμα TextBoxUserControl

- Η κλάση VectorPropertyUserControl

Η κλάση αυτή πρόκειται για ένα UserControl που περιέχει ένα σύνολο τριών text boxes με τίτλους "x", "y" και "z". Τα τρία αυτά text boxes απεικονίζουν ένα vector τριών διαστάσεων και μπορούν να πάρουν μόνο αριθμητικές float τιμές. Δημιουργείται για να αναπαραστήσει ιδιότητες όπως τη θέση, την περιστροφή ή το μέγεθος του αντικειμένου. Ένα παράδειγμα ενός τέτοιου VectorPropertyUserControl απεικονίζεται παρακάτω.



Εικόνα 11: Παράδειγμα VectorPropertyUserControl

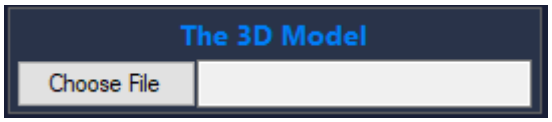
- Η κλάση SelectPropertyUserControl

Η κλάση αυτή δημιουργεί ένα UserControl το οποίο περιέχει ένα Button και ένα TextBox. Χρησιμοποιείται για properties οι οποίες έχουν να κάνουν με επιλογή αρχείων από τον υπολογιστή του χρήστη. Το button μπορεί να έχει δύο ονόματα είτε "Choose File" είτε "Choose Folder". Μετά από το πάτημα του, θα ανοίξει ένα μενού επιλογής αρχείου ή φακέλου αντίστοιχα. Αυτό καθορίζεται από την μεταβλητή selectType του αντικειμένου selectProperty.

Το μενού επιλογής αρχείου χρησιμοποιείται για properties όπως για παράδειγμα το 3D μοντέλο ή το background ενός 2D παιχνιδιού. Προκειμένου να εμφανιστεί το μενού επιλογής αρχείου, δημιουργείται ένα αντικείμενο της κλάσης OpenFileDialog το οποίο παίρνει ως τίτλο το όνομα της ιδιότητας και επιτρέπει τον χρήστη να επιλέξει αρχεία με συγκεκριμένες καταλήξεις που ορίζονται από τη μεταβλητή fileFilters του αντικειμένου selectProperty.

Το μενού επιλογής φακέλου, από την άλλη, χρησιμοποιείται για properties όπως για παράδειγμα τα textures του μοντέλου ή τα animations του. Η επιλογή φακέλου σε τέτοιες περιπτώσεις είναι απαραίτητη επειδή πρόκειται για μια συλλογή αρχείων και όχι μόνο ενός. Το μενού επιλογής φακέλου εμφανίζεται με τη δημιουργία ενός αντικειμένου της κλάσης CommonOpenFileDialog το οποίο παίρνει ως τίτλο το όνομα της ιδιότητας.

Μετά την επιλογή του αρχείου ή του φακέλου, το text box του UserControl παίρνει ως τιμή το path προς το επιλεγμένο αρχείο ή φάκελο. Ένα παράδειγμα ενός τέτοιου VectorPropertyUserControl απεικονίζεται παρακάτω.



Εικόνα 12: Παράδειγμα SelectPropertyUserControl

- Η κλάση ComboPropertyUserControl

Η κλάση αυτή δημιουργεί ένα UserControl το οποίο περιέχει απλά ένα combo box. Το UserControl αυτό χρησιμοποιείται για να αναπαραστήσει properties όπως “hasPhysics” ή “colliderType”, όπου ο χρήστης καλείται να επιλέξει μια τιμή από ένα σύνολο διαθέσιμων τιμών. Αρχικά το combo box αυτό είναι άδειο και παίρνει τιμές ανάλογα με τις τιμές που περιέχει η μεταβλητή optionValues του αντικειμένου comboProperty. Ένα σημαντικό πρόβλημα δημιουργείται όμως σε περιπτώσεις που ένα property είναι μοναδικό (δηλαδή αν μπορεί να υπάρχει σε πάνω από δύο αντικείμενα του αρχείου Json) ή σε περιπτώσεις που υπάρχει κάποια λογική σύνδεση μεταξύ των properties (για παράδειγμα το property “hasGravity” δεν πρέπει να εμφανίζεται σε περίπτωση που το property “hasPhysics” έχει οριστεί ως false). Τα user controls σε αυτές τις περιπτώσεις πρέπει να εμφανίζονται μόνο αν τους το επιτρέπουν οι συνθήκες.

Το αν είναι μοναδικό ή όχι ένα property καθορίζεται από τη μεταβλητή “isUnique” του αντικειμένου comboProperty. Τα αντικείμενα που μπορούν να οριστούν ως μοναδικά περιορίζονται σε αυτά που έχουν μόνο δύο τιμές true ή false. Σε περίπτωση που το αντικείμενο έχει οριστεί ως μοναδικό, εκτελείται η μέθοδος EditOtherPropertiesVisibilityIfUnique η οποία, με το που ορίζεται η τιμή του combo box σε true, εξαφανίζει όλα τα αντίστοιχα UserControls από κάθε άλλο category αντικειμένου.

Η λογική σύνδεση μεταξύ των properties καθορίζεται από τη μεταβλητή children του αντικειμένου comboProperty. Η μεταβλητή αυτή πρόκειται για έναν πίνακα που περιέχει άλλα properties τα οποία θεωρούνται children του property αυτού, και θα εξαφανιστούν σε περίπτωση που η τιμή του property αυτού έχει οριστεί ως false ή άμα το ίδιο το property έχει εξαφανιστεί. Την λειτουργικότητα αυτή καλείται να αναλάβει η μέθοδος EditOtherPropertiesVisibilityIfUnique η οποία κατά την αλλαγή της τιμής του combo box, εξαφανίζει τα κατάλληλα properties μέσα στο ίδιο category.

Για ειδική περίπτωση πρόκειται το property με όνομα Dimension, το οποίο είναι ένα projectProperty που δηλώνει τη διάσταση του παιχνιδιού. Η αλλαγή της τιμής του από 3D σε 2D και ανάποδα σημαίνει ότι κάποια από τα properties των αντικειμένων πρέπει να αντικατασταθούν από κάποια άλλα. Τη διαδικασία αυτή καλείται να υλοποιήσει η μέθοδος EditPropertiesVisibilityByDimension, η οποία λαμβάνοντας υπ’ όψη τη τιμή της μεταβλητής dimension του αντικειμένου comboProperty σε κάθε PropertyUserControl, εξαφανίζει τα properties που δεν συμφωνούν με την τιμή αυτή του αντικειμένου.

Ένα παράδειγμα ενός τέτοιου ComboPropertyUserControl απεικονίζεται παρακάτω.

Εικόνα 13: Παράδειγμα ComboPropertyUserControl

- Η δημιουργία του Json αρχείου

Αφού ο χρήστης έχει συμπληρώσει επαρκώς τα πεδία του κάθε category και πατήσει το κουμπί “Create Your Unity Package”, ένα παράθυρο επιλογής φακέλου θα ανοίξει στο οποίο ο χρήστης θα κληθεί να επιλέξει την τοποθεσία στην οποία θα δημιουργηθεί το Unity Package. Αφού λοιπόν ο χρήστης επιλέξει ένα έγκυρο path φακέλου, ξεκινάει η διαδικασία δημιουργίας του Json αρχείου που θα χρησιμοποιηθεί αργότερα για την δυναμική δημιουργία σκηνής και του παιχνιδιού από το εργαλείο JsonSceneGenerator. Για την δημιουργία του Json γίνεται χρήση της βιβλιοθήκης Newtonsoft, η οποία παρέχει τις κατάλληλες εντολές ώστε να υλοποιηθεί το serialization από και προς το Json object. Η μέθοδος CreateJson λοιπόν διαβάζει τις κατάλληλες τιμές των text boxes των PropertUserControls από κάθε category και τις αποθηκεύει ως properties στο Json object αναπαριστώντας τα αντικείμενα της σκηνής.

Στη συνέχεια ένα αρχείο Json δημιουργείται με χρήση της εντολής Write της βιβλιοθήκης StreamWriter. Το αρχείο αυτό θα τοποθετηθεί στον κατάλληλο φάκελο με όνομα Json μέσα στο προ δημιουργημένο Unity package.

- Η αντιγραφή των αρχείων

Μετά την επιτυχής δημιουργία του Json αρχείου, ακολουθεί η αντιγραφή των αρχείων του χρήστη, το path των οποίων αναγράφεται στο text box κάθε SelectUserControl. Τα αρχεία αυτά λοιπόν θα αντιγραφούν επίσης μέσα στους κατάλληλους φακέλους του προ δημιουργημένου Unity package.

- Η δημιουργία του τελικού Unity package

Αφού αντιγραφούν και τα αρχεία του χρήστη σειρά έχει η δημιουργία του τελικού Unity package. Το package αυτό θα έχει την μορφή ενός συμπιεσμένου .zip αρχείου. Το .zip αυτό αρχείο θα περιέχει την τελική έκδοση του Unity package, η οποία θα περιέχει το Json αρχείο καθώς και τα αρχεία του παιχνιδιού που έχει παρέχει ο χρήστης, τοποθετημένα στους κατάλληλους φακέλους εσωτερικά του Package. Η δημιουργία του .zip αρχείου γίνεται με τη χρήση της βιβλιοθήκης ZipFile, όπου με χρήση της εντολής “`ZipFile.CreateFromDirectory(startPath, zipPath))`” ο φάκελος του package ο οποίος βρίσκεται στο startPath, συμπιέζεται σε μορφή zip και αντιγράφεται στο zipPath που έχει επιλέξει ο χρήστης. Αν δεν υπάρξει κάποιο πρόβλημα, έχει πλέων δημιουργηθεί το εργαλείο με όνομα JsonSceneGenerator στη μορφή ενός UnityPackage, το οποίο είναι έτοιμο για την ενσωμάτωσή του στο Unity3D.

3.2 Το εργαλείο JsonSceneGenerator

Η ανάπτυξη του εργαλείου έγινε μέσα στο Unity3D Game Engine και υλοποιήθηκε ως Package ώστε ο χρήστης να μπορεί να το εισάγει εύκολα στο δικό του project μέσω του Unity Package Manager. Οι εφαρμογές το Unity μπορούν να «τρέξουν» σχεδόν σε όλα τα δημοφιλή λειτουργικά συστήματα (καθώς και σε κονσόλες). Αυτό κάνει την ανάπτυξη των εφαρμογών πολύ πιο γρήγορη. Βέβαια η κάθε εφαρμογή χρειάζεται την αντίστοιχη προσαρμογή για να μην υπάρχουν προβλήματα στο κάθε λειτουργικό σύστημα. Επιπλέον, το Unity χρησιμοποιεί μια γλώσσα υψηλού επιπέδου την C#

παρέχοντας όλες τις απαραίτητες βιβλιοθήκες, ενώ παράλληλα εξασφαλίζει τη σωστή λειτουργία της εφαρμογής.

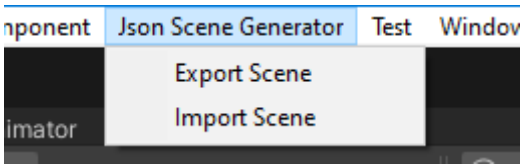
3.3 Δυναμική παραγωγή παιχνιδιού μέσω ενός αρχείου Json

Στην ενότητα αυτή, θα παρουσιαστεί ο τρόπος σχεδίασης και ανάπτυξης του εργαλείου για δυναμική παραγωγή μιας πρότυπης σκηνής παιχνιδιού. Κεντρική ιδέα είναι η υλοποίηση ενός εργαλείου το οποίο θα εξάγει την απαραίτητη πληροφορία από ένα δοθέν Json αρχείο και σε συνδυασμό με τα απαραίτητα αρχεία τα οποία θα παρέχει ο χρήστης (Models, Textures, Animations, κ.λπ.), θα δημιουργείται δυναμικά μία σκηνή και κατ' επέκταση το παιχνίδι μέσα στο Unity Game Engine. Έτσι, ακόμα και ένας άπειρος χρήστης του Unity3D θα μπορεί πολύ εύκολα να δημιουργήσει μια πρότυπη σκηνή παιχνιδιού.

3.3.1 Η δημιουργία του μενού επιλογών

Το Unity3D παρέχει τη δυνατότητα δημιουργίας editor scripts . Το Editor Scripting είναι ένα πολύ ισχυρό χαρακτηριστικό καθώς επιτρέπει στον προγραμματιστή να δημιουργεί εργαλεία και να επεκτείνει το Unity Editor ώστε να εκτελούν προσαρμοσμένες εργασίες που θα μπορούσαν να βοηθήσουν στην ανάπτυξη του έργου. Αυτό περιλαμβάνει ένα προσαρμοσμένο μενού, εργαλεία κινούμενων εικόνων και πολλά άλλα. Επιπροσθέτως παρέχει τη δυνατότητα στον προγραμματιστή να ορίσει πώς θα εμφανίζονται οι ιδιότητες στο Inspector με προσαρμοσμένα συρτάρια ιδιοτήτων, αλλά και να δημιουργήσει οποιονδήποτε αριθμό προσαρμοσμένων παραθύρων στο έργο του, τα οποία θα συμπεριφέρονται ακριβώς όπως το Inspector, το Scene ή οποιαδήποτε από τα άλλα ενσωματωμένα παράθυρα Unity.

Η δημιουργία της σκηνής και του παιχνιδιού λοιπόν ξεκινάει από την κλάση SceneImporterExecutor η οποία εμπεριέχει μία static μέθοδο στην οποία έχει δοθεί το attribute «[MenuItem("Json Scene Generator/Import Scene")]». Το attribute αυτό δημιουργεί ένα έξτρα μενού στο ήδη υπάρχον μενού του Unity Editor, δίνοντας στον χρήστη την δυνατότητα να εκτελέσει την συγκεκριμένη μέθοδο κάνοντας click κατάλληλο μενού.

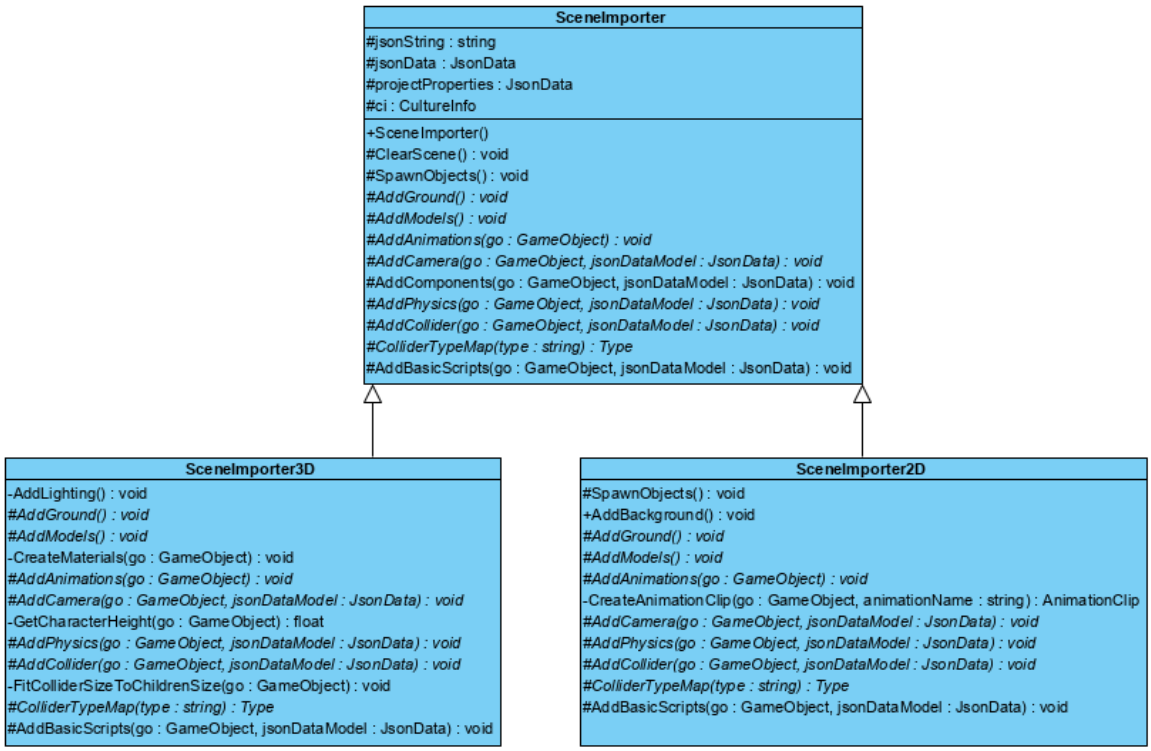


Εικόνα 14: Το μενού επιλογών, για την εκτέλεση του εργαλείου

Από τη στιγμή που θα γίνει η κλήση της, η μέθοδος αυτή θα εξάγει από το Json αρχείο τη διάσταση του project (3D ή 2D) και θα δημιουργήσει το αντίστοιχο αντικείμενο «SceneImporter3D» άμα πρόκειται για τρισδιάστατο project ή «SceneImporter2D» άμα πρόκειται για δισδιάστατο project. Επίσης με τη χρήση εντολών της βιβλιοθήκης «UnityEditor», αλλάζουν τα δύο properties του Unity Editor:

- defaultBehaviorMode, το οποίο θα οριστεί σε Mode3D ή Mode2D, και θα προσαρμόσει συγκεκριμένες λειτουργικότητες του Unity Editor ανάλογα με τη διάσταση του project.

- lastActiveSceneView.in2DMode, το οποίο θα οριστεί σε false ή true, και θα γυρίσει την κάμερα του παραθύρου Scene σε 3D ή 2D κάμερα ανάλογα με τη διάσταση του project.



Εικόνα 15: Το UML διάγραμμα του SceneImporter

3.3.2 Η ανάγνωση του αρχείου Json

Η ανάγνωση του Json αρχείου γίνεται με τι χρήση της βιβλιοθήκης UnityLitJSON, μίας .Net C# βιβλιοθήκης που διαχειρίζεται μετατροπές από και προς JSON(JavaScript Object Notation) strings. Η συγκεκριμένη βιβλιοθήκη πρόκειται για μια παραλλαγή της βιβλιοθήκης LitJSON, τροποποιημένη έτσι ώστε να λειτουργεί με το Unity3D [20].

Ποιο συγκεκριμένα χρησιμοποιώντας την εντολή «JsonMapper.ToObject(jsonString);», το περιεχόμενο του Json αρχείου μετατρέπεται σε μια σειρά από εμφωλευμένα Dictionaries, αναπαριστώντας έτσι κατάλληλα το Json object.

Προκειμένου όμως να μπορέσει να γίνει η χρήση του από το εργαλείο, το αρχείο Json αυτό θα πρέπει να έχει μια συγκεκριμένη μορφή. Αρχικά θα πρέπει να αποτελείται από δύο root αντικείμενα: το αντικείμενο projectProperties και το αντικείμενο gameObjects. Το αντικείμενο projectProperties θα περιέχει στη συνέχεια properties που έχουν να κάνουν με το project. Το αντικείμενο gameObjects θα εμπεριέχει στη συνέχεια τόσα αντικείμενα όσα και τα αντικείμενα της σκηνής. Το κάθε ένα από αυτά με τη σειρά του θα περιέχει properties που σχετίζονται με αυτό όπως το όνομά του ή τη θέση του στη σκηνή.

```
"projectProperties": {
    "dimension": "3d"
  },
  "gameObjects": {
    "Alien": {
      "name": "Alien",
      "hasAnimations": "true",
      "position": {
        "x": "-8",
        "y": "0",
        "z": "0"
      },
      "rotation": {
        "x": "0",
        "y": "0",
        "z": "0"
      },
      "scale": {
        "x": "1",
        "y": "1",
        "z": "1"
      },
      "colliderType": "capsuleCollider",
      "hasPhysics": "true",
      "hasGravity": "true",
      "health": "10"
    },
    "Room": {
      "name": "Room",
      "hasAnimations": "false",
      "position": {
        "x": "0",
        "y": "0",
        "z": "0"
      },
      "rotation": {
        "x": "0",
        "y": "0",
        "z": "0"
      },
      "scale": {
        "x": "1",
        "y": "1",
        "z": "1"
      },
      "colliderType": "meshCollider",
      "hasPhysics": "false",
      "health": "100"
    }
```


Εικόνα 16: Παράδειγμα περιεχομένου του Json αρχείου με ένα αντικείμενο στη σκηνή

3.3.3 Η δημιουργία του 3D παιχνιδιού

Έπειτα από τη δημιουργία του αντικειμένου SceneImporter3D μέσω του SceneImporterExecutor ξεκινάει η διαδικασία δυναμικής δημιουργίας της σκηνής και του παιχνιδιού [21]. Ο δομητής της κλάσης εφόσον μετατρέψει και αποθηκεύσει τα περιεχόμενα του αρχείου Json ως Dictionary στην

μεταβλητή JsonDataModel, προβαίνει στον καθαρισμό της σκηνής και στην προσθήκη των νέων GameObjects καλώντας τις δύο βασικές μεθόδους: ClearScene και SpawnObjects.

3.3.3.1 Η προετοιμασία της σκηνής

Το πρώτο πράγμα που θα πρέπει να γίνει είναι να καθαριστεί η σκηνή από οποιαδήποτε objects μπορεί να προϋπάρχουν. Την λειτουργεία αυτή καλείται να αναλάβει η μέθοδος ClearScene. Αυτό γίνεται με τη κλήση της μεθόδου DestroyImmediate για κάθε root GameObject στη σκηνή.

Στη συνέχεια ακολουθεί η προσθήκη των νέων GameObjects που ξεκινά από τη μέθοδο SpawnObjects η οποία αναλαμβάνει την προσθήκη φωτισμού, εδάφους και των μοντέλων που έχει παρέχει ο χρήστης καλώντας τις μεθόδους: AddLighting, AddGround και AddModels αντίστοιχα.

3.3.3.2 Ο φωτισμός

Ο φωτισμός στα σύγχρονα παιχνίδια κάνει εκτεταμένη χρήση του “Global Illumination”. Το Global Illumination, (ή «GI»), είναι ένας όρος που χρησιμοποιείται για να περιγράψει μια σειρά τεχνικών και μαθηματικών μοντέλων που προσπαθούν να προσομοιώσουν τη σύνθετη συμπεριφορά του φωτός καθώς αναπηδά και αλληλοεπιδρά με τον κόσμο. Η ακριβής προσομοίωση του παγκόσμιου φωτισμού είναι πρόκληση και μπορεί να είναι υπολογιστικά ακριβή. Εξαιτίας αυτού, τα παιχνίδια χρησιμοποιούν μια σειρά προσεγγίσεων για να χειριστούν αυτούς τους υπολογισμούς εκ των προτέρων και όχι κατά τη διάρκεια του παιχνιδιού [22].

Το Unity3D παρέχει κάποια έτοιμα εργαλεία ώστε να μπορέσει ο προγραμματιστής να πετύχει το είδος του φωτισμού που επιθυμεί [23]:

- Directional: Τα Directional Lights είναι πολύ χρήσιμα για την αναπαράσταση εφέ όπως το ηλιακό φως στη σκηνή. Συμπεριφερόμενα με πολλούς τρόπους όπως και ο ήλιος, τα Directional Lights μπορούν να θεωρηθούν ως μακρινές πηγές φωτός που υπάρχουν απείρως μακριά. Οι ακτίνες φωτός που εκπέμπονται από τα Directional Lights είναι παράλληλες μεταξύ τους και δεν αποκλίνουν όπως αυτές από άλλους τύπους φωτός. Ως αποτέλεσμα, οι σκιές που ρίχνουν τα Directional Lights φαίνονται ίδιες, ανεξάρτητα από τη θέση τους σε σχέση με την πηγή. Αυτό είναι χρήσιμο για τον φωτισμό εξωτερικών σκηνών.
- Point: Ένα Σημειακό Φως μπορεί να θεωρηθεί ως ένα σημείο στον τρισδιάστατο χώρο από το οποίο εκπέμπεται φως προς όλες τις κατευθύνσεις. Αυτά είναι χρήσιμα για τη δημιουργία εφέ όπως λαμπτήρες, λάμψη όπλων ή εκρήξεις όπου αναμένετε να εκπέμπεται φως από ένα αντικείμενο. Η ένταση του Point Lights in Unity μειώνεται τετραγωνικά από την πλήρη ένταση στο κέντρο του φωτός, στο μηδέν στο όριο της εμβέλειας του φωτός που ορίζεται από την ιδιότητα «Range» του στοιχείου στον Inspector. Η ένταση του φωτός είναι αντιστρόφως ανάλογη με το τετράγωνο της απόστασης από την πηγή. Αυτό είναι γνωστό ως «αντίστροφος τετράγωνος νόμος» και είναι παρόμοιο με το πώς συμπεριφέρεται το φως στον πραγματικό κόσμο.
- Spotlights: Οι προβολείς προβάλλουν έναν κώνο φωτός προς την εμπρός (+Z) κατεύθυνσή τους. Το πλάτος αυτού του κώνου ορίζεται από την παράμετρο «Spot Angle» του φωτός. Το φως θα «πέσει» από τη θέση της πηγής προς την έκταση του εύρους του φωτός, όπου τελικά θα μειωθεί στο μηδέν. Το φως μειώνεται επίσης στις άκρες του κώνου του Spotlight. Η διεύρυνση της γωνίας κηλίδας αυξάνει το πλάτος του κώνου και μαζί με αυτό, αυξάνει το μέγεθος αυτού του ξεθωριάσματος, που είναι γνωστό ως «penumbra». Οι προβολείς έχουν πολλές χρήσιμες εφαρμογές για τον φωτισμό σκηνής. Μπορούν να χρησιμοποιηθούν με εξαιρετικό αποτέλεσμα ως φώτα δρόμου, φώτα τοίχου ή να χρησιμοποιηθούν δυναμικά, για

τη δημιουργία εφέ όπως ένας φακός. Καθώς η περιοχή επιρροής τους μπορεί να ελεγχθεί με ακρίβεια, τα Spotlights είναι εξαιρετικά χρήσιμα για τη δημιουργία εστίασης σε έναν χαρακτήρα ή για τη δημιουργία δραματικών εφέ φωτισμού σκηνής

- Area Lights: Τα φώτα περιοχής μπορούν να θεωρηθούν παρόμοια με το softbox ενός φωτογράφου. Στο Unity ορίζονται ως ορθογώνια από τα οποία εκπέμπεται φως προς όλες τις κατευθύνσεις, μόνο από τη μία πλευρά - την κατεύθυνση +Z του αντικειμένου. Προς το παρόν διατιθέμενα μόνο σε Baked GI, αυτά τα φώτα περιοχής ανάβουν ομοιόμορφα σε όλη την επιφάνειά τους. Δεν υπάρχει χειροκίνητος έλεγχος για την εμβέλεια ενός Area Light, ωστόσο η ένταση θα μειωθεί στο αντίστροφο τετράγωνο της απόστασης καθώς απομακρύνεται από την πηγή. Το φως εκπέμπεται σε όλη την επιφάνεια ενός Φωτός Περιοχής παράγοντας ένα διάχυτο φως με απαλή σκίαση. Τα φώτα περιοχής είναι χρήσιμα σε περιπτώσεις δημιουργίας απαλών εφέ φωτισμού. Καθώς το φως εκπέμπεται προς όλες τις κατευθύνσεις σε όλη την επιφάνεια του φωτός, οι ακτίνες που παράγονται ταξιδεύουν προς πολλές κατευθύνσεις - δημιουργώντας ένα διάχυτο εφέ φωτισμού σε ένα θέμα. Μια κοινή χρήση για αυτό μπορεί να είναι ένα φωτιστικό οροφής ή ένα πάνελ με οπίσθιο φωτισμό.

Το εργαλείο λοιπόν προσθέτει στην σκηνή τον φωτισμό μέσο της μεθόδου AddLighting δημιουργώντας ένα GameObject με το όνομα “Sun” και Component “Light”. Η μεταβλητή Type του Component αυτού στη συνέχεια θα πάρει τη τιμή Directional, καθώς επιθυμούμε να αναπαραστήσουμε έναν εικονικό ήλιο.

3.3.3.3 Το έδαφος

Η μέθοδος AddGround προσθέτει ένα στοιχειώδες έδαφος στη σκηνή δημιουργώντας ένα Primitive Plane GameObject με το όνομα “Ground”. Το GameObject αυτό θα έχει το σχήμα τετραγώνου με διαστάσεις 100 επί 100, και θα είναι τοποθετημένο στο κέντρο της σκηνής.

3.3.3.4 Τα μοντέλα

Έπειτα η μέθοδος AddModels υλοποιεί την λειτουργικότητά της καλώντας έναν βρόγχο foreach ο οποίος εκτελείται για κάθε αντικείμενο μέσα στο περιεχόμενο του Json. Για κάθε αντικείμενο λοιπόν:

a. Προστίθεται στη σκηνή το αντίστοιχο μοντέλο ως GameObject που έχει παρέχει ο χρήστης στον φάκελο «Resources/GameObjects».
b. Άμα δεν υπάρχει ήδη κάποιος φάκελος με το όνομα Materials για κάθε αντικείμενο, προσθέτει δυναμικά materials στο GameObject με βάση τα textures που υπάρχουν στον φάκελο Textures του κάθε αντικειμένου τα οποία είναι υποχρεωμένος να παρέχει ο χρήστης από πριν. Αυτό επιτυγχάνεται διαβάζοντας το ονόματα των textures και αντιστοιχώντας το κάθε texture με το ίδιο το GameObject η το αντίστοιχο child GameObject που μπορεί να έχει. Εδώ πρέπει να σημειωθεί ότι κατά τη δημιουργία του material, άμα το αντίστοιχο texture του περιλαμβάνει στο όνομά του «_Normal», τότε το συγκεκριμένο texture ορίζεται ως Normal Map στο συγκεκριμένο material, αφού πρώτα μετατραπεί σε normal map στο ίδιο το Unity Engine χρησιμοποιώντας για ακόμα μια φορά την UnityEditor βιβλιοθήκη.
c. Άμα το GameObject ορίζεται ως Controllable στο Json, το Layer του αντικειμένου ορίζεται ως «TransparentFX», ώστε να διαφοροποιηθεί από τα υπόλοιπα GameObjects.
d. Ορίζεται, με βάση τις τιμές του Json, το Position, το Rotation και το Scale του GameObject.
e. Άμα το GameObject ορίζεται από το Json να έχει animations, εκτελείται η μέθοδος AddAnimations, ενώ άμα ορίζεται να έχει main camera, εκτελείται η μέθοδος AddCamera.
f. Τέλος εκτελούνται οι τρεις μέθοδοι, AddPhysics, AddCollider και AddBasicScripts, οι οποίες είναι υπεύθυνες για την προσθήκη των physics, των colliders και μερικών βασικών gameplay scripts αντίστοιχα.

3.3.3.5 Τα animations

Το κομμάτι της προσθήκης των animations καλείται να αναλάβει η μέθοδος AddAnimations. Προκειμένου ένα GameObject στο Unity3D να έχει Animations πρέπει να γίνει χρήση του Component "Animator". Από κει και πέρα, για να λειτουργήσει το animation θα πρέπει να οριστούν δύο βασικές μεταβλητές του Component αυτού, η μεταβλητή Avatar και η μεταβλητή Controller.

Ως Avatar ορίζεται το μοντέλο πάνω στο οποίο θα εφαρμοστούν τα animations και πρέπει προκειμένου να μην υπάρξουν σφάλματα συνίσταται να είναι το ίδιο το μοντέλο του GameObject [24].

Η μεταβλητή Controller πρόκειται για ένα αντικείμενο τύπου RuntimeAnimatorController, το οποίο αποτελεί μια built in βιβλιοθήκη του Unity3D που παρέχει επίσης και ένα πολύ βασικό εργαλείο που διαχειρίζονται τις καταστάσεις των animation ενός GameObject.

Το εργαλείο αυτό ονομάζεται Animator και πρόκειται για ένα γραφικό περιβάλλον στο οποίο μπορεί ο χρήστης να σχεδιάσει τη δομή και τις διασυνδέσεις μεταξύ των διαφορετικών καταστάσεων των διαφόρων animations που θα έχει το GameObject. Μέσο του εργαλείου αυτού λοιπόν ο χρήστης έχει τη δυνατότητα να ορίσει ποια animations θα εφαρμοστούν, το πότε θα γίνει η εναλλαγή από το ένα animation στο άλλο και το πώς θα γίνει η εναλλαγή αυτή (π.χ. χρονικό σημείο εναλλαγής).

Το πότε θα γίνει η εναλλαγή καθορίζεται από τις λεγόμενες Parameters του Animator οι οποίες είναι πρακτικά μεταβλητές διάφορον τύπων που κατά την αλλαγή τους κάνουν trigger τα διάφορα events που έχει ορίσει ο χρήστης μέσω της καρτέλας Conditions που βρίσκετε στα γραφικά βέλη που αντιπροσωπεύουν τις διασυνδέσεις των animations. Η εναλλαγή των μεταβλητών μπορεί να γίνει είτε χειροκίνητα είτε μέσω του κώδικα.

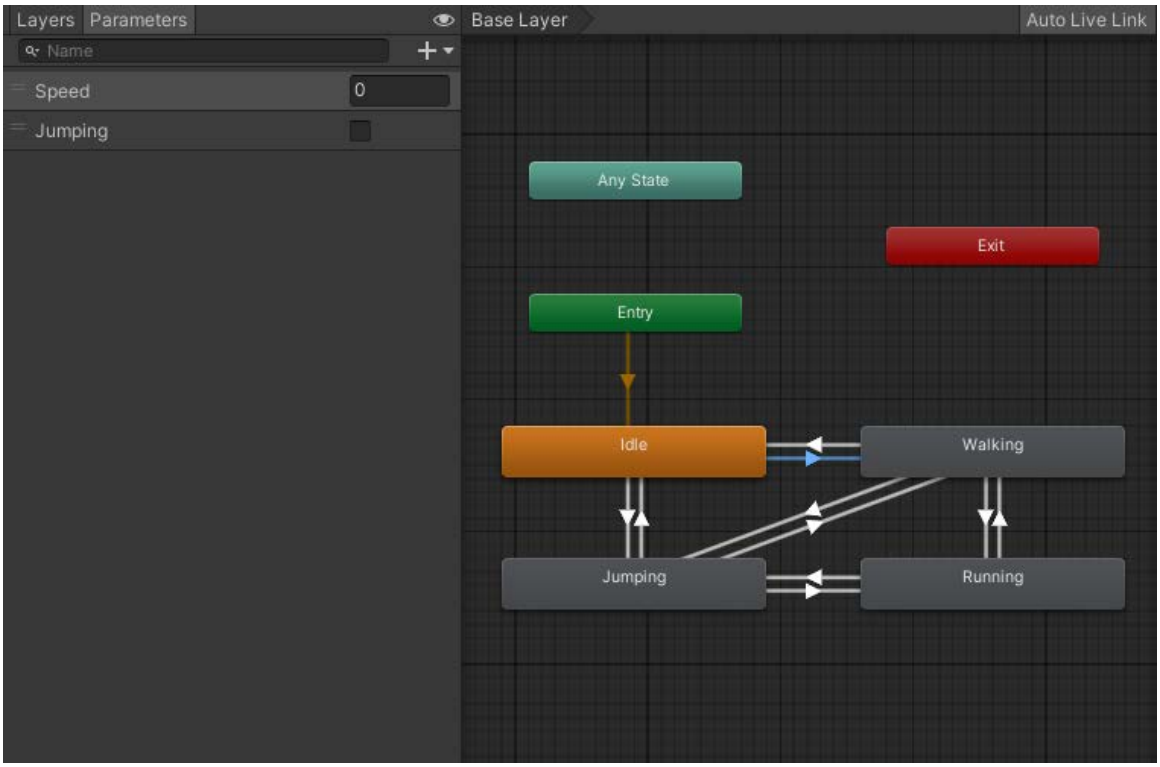
Για τους σκοπούς λοιπόν της δυναμικής προσθήκης animations σε κάθε αντικείμενο που έχει προμηθεύσει ο χρήστης του εργαλείου SceneImporter, έχει δημιουργηθεί ένα προκαθορισμένο RuntimeAnimatorContoller το οποίο έχει αποθηκευτεί στον φάκελο Prefabs κάτω από τον φάκελο Resources και περιέχει 4 βασικές καταστάσεις animation, τα οποία ονομάζονται AnimationClips:

- Idle: Την αδρανή κατάσταση του μοντέλου.
- Walking: Την κατάσταση κατά την οποίο το μοντέλο «περπατάει».
- Running: Την κατάσταση κατά την οποία το μοντέλο «τρέχει».
- Jumping: Την κατάσταση κατά την οποία το μοντέλο «πηδάει».

Οι καταστάσεις αυτές αλληλοσυνδέονται μεταξύ τους και η μετάβαση από τη μία κατάσταση στην άλλη επιτυγχάνεται από 2 βασικές προκαθορισμένες Parameters

- Speed: Μία float Parameter που αναπαριστά την ταχύτητα του GameObject.
- Jump: Μία bool Parameter η οποία ανάλογα με το αν είναι true ή false, ορίζει αν το GameObject βρίσκεται σε κατάσταση Jumping ή όχι αντίστοιχα.

Όσον αφορά το Conditions τα οποία θα ορίσουν το πότε θα γίνει η αλλαγή του AnimationClip, έχουν οριστεί κάποιες προκαθορισμένες τιμές στην Parameter Speed. Για παράδειγμα αν η τιμή της Parameter speed είναι μεγαλύτερη από την προκαθορισμένη ενδεικτική τιμή 7, το AnimationClip θα αλλάξει από Idle σε Walking, και ου το κάθε εξής, ενώ η Parameter speed θα ορίσει το πότε θα ενεργοποιηθεί το Jumping AnimationClip. Το προκαθορισμένο RuntimeAnimatorContoller φαίνεται αναλυτικά στην παρακάτω εικόνα.



Εικόνα 17: Το προκαθορισμένο RuntimeAnimatorController

Η πρώτη λειτουργεία της μεθόδου AddAnimations λοιπόν είναι προσθέσει ένα Component τύπου Animator στο GameObject.

Έπειτα πρέπει να ορίσει ένα Avatar για το Component αυτό. Αυτό επιτυγχάνεται φορτώνοντας ένα τυχαίο animation από αυτά που έχει παρέχει ο χρήστης, και ορίζοντας το μοντέλο του ως Avatar του Animator, εφόσον είναι σίγουρο ότι το μοντέλο του animation είναι το ίδιο με το μοντέλο, μιας και το animation προϋποθέτει να έχει φτιαχτεί για αυτό.

Αφού οριστεί το Avatar σειρά έχει η μεταβλητή Controller, η οποία θα πάρει ως τιμή το προκαθορισμένο RuntimeAnimatorController που έχει προαναφερθεί. Στη συνέχεια θα δημιουργηθεί ένα αντικείμενο της κλάσης AnimatorOverrideController, το οποίο είναι δίνει την δυνατότητα για επέκταση του υπάρχοντος RuntimeAnimatorController, αντικαθιστώντας τα animations που χρησιμοποιεί, κρατώντας όμως ταυτόχρονα την αυθεντική δομή, παραμέτρους και λογική που έχουν οριστεί. Αυτό επιτρέπει τη δημιουργία πολλαπλών παραλλαγών του ίδιου RuntimeAnimatorController, αλλά με το καθένα να χρησιμοποιεί διαφορετικά AnimationClips. Έπειτα θα πρέπει να αντιστοιχηθούν τα καινούρια animations με τις αντίστοιχες καταστάσεις τους. Εδώ είναι σημαντικό να αναφερθεί ότι τα AnimationClips πρέπει να έχουν προκαθορισμένα ονόματα προκειμένου να μπορέσουν να φορτωθούν και να αντιστοιχηθούν σωστά με τα animation των ήδη υπαρχόντων καταστάσεων. Τα ονόματα αυτά των animations θα πρέπει να είναι επ’ ακριβώς: “Idle”, “Walking”,”Running” ή “Jumping”. Από τη στιγμή που ικανοποιείται αυτή η προϋπόθεση, τα animations φορτώνονται στο σύστημα και εισάγονται στον νέο AnimatorOverrideController μέσω του παρακάτω set εντολών.

```
1. AnimationClipOverrides clipOverrides = new
   AnimationClipOverrides(animatorOverrideController.overridesCount);
2.
3.         animatorOverrideController.GetOverrides(clipOverrides);
4.
5.         clipOverrides["Idle"] = idleAnimationClip;
6.         clipOverrides["Walking"] = walkingAnimationClip;
7.         clipOverrides["Running"] = runningAnimationClip;
8.         clipOverrides["Jumping"] = jumpingAnimationClip;
9.
10.        animatorOverrideController.ApplyOverrides(clipOverrides);
```

Εικόνα 18: Set εντολών που εισάγουν τα animations στον νέο AnimatorOverrideController

Τέλος, η μεταβλητή RuntimeAnimatorController (ή Animator όπως φαίνεται στο GUI του Unity3D) του Component Animator του GameObject αντικαθίσταται με το καινούριο AnimatorOverrideController.

3.3.3.6 Η κάμερα

Προκειμένου ο χρήστης να έχει οπτική εικόνα της σκηνής κατά την εκτέλεση του παιχνιδιού (στο Play Mode), θα πρέπει να προστεθεί ένα βασικό Component σε ένα οποιοδήποτε GameObject της σκηνής. Οι κάμερες είναι τα υπεύθυνα εργαλεία για το rendering της σκηνής. Μπορούν να προσαρμοστούν, επεξεργαστούν μέσω κώδικα είτε να συνδυαστούν με άλλα εργαλεία για να επιτύχουν οποιουδήποτε είδους εφέ. Σε ένα σύγχρονο βιντεοπαιχνίδι η χρήση της κάμερας γίνεται με τρεις τρόπους: α) στατική κάμερα, β) κάμερα πρώτου προσώπου και γ) κάμερα τρίτου προσώπου. Στο συγκεκριμένο project γίνεται χρήση των 2 τελευταίων τρόπων.

Η μέθοδος AddCamera λοιπόν καλείται να υλοποιήσει έναν από τους δύο αυτούς τρόπους οπτικής αναπαράστασης ανάλογα με το πεδίο “HasCamera” του περιεχομένου του Json αρχείου. Το πεδίο αυτό είναι μοναδικό για κάθε μοντέλο, πράγμα που σημαίνει ότι μόνο ένα GameObject μέσα στη σκηνή θα έχει στην κατοχή του την τελική κάμερα. Ένας ακόμα περιορισμός είναι ότι το GameObject θα πρέπει να έχει οριστεί και ως Controllable. Από τη στιγμή που ικανοποιούνται αυτοί οι δύο περιορισμοί εκτελείται η μέθοδος για το επιλεγμένο GameObject.

3.3.3.6.1 Κάμερα πρώτου προσώπου

Η κάμερα πρώτου προσώπου πρόκειται για μια αναπαράσταση της ανθρώπινης όρασης. Κατά τη χρήση της κάμερας με αυτόν τον τρόπο ο χρήστης βλέπει την σκηνή μέσω της κάμερας όπως θα την έβλεπε στον πραγματικό κόσμο. Προκειμένου να επιτευχθεί αυτό η κάμερα πρέπει να βρίσκεται πάνω στο χαρακτήρα που θα χειρίζεται ο παίκτης στο ύψος που βρίσκονται τα μάτια. Μία άλλη σημαντική προϋπόθεση είναι το Component της κάμερας ή το GameObject που περιέχει το Component της κάμερας να είναι child του χαρακτήρα που θα χειρίζεται ο παίκτης, ώστε κατά την περιστροφή του να περιστρέφεται και η κάμερα, είτε η λειτουργία αυτή να προσομοιωθεί μέσω κώδικα.

Ο τρόπος με τον οποίο η μέθοδος AddCamera προσθέτει την κάμερα πρώτου προσώπου είναι η εξής:

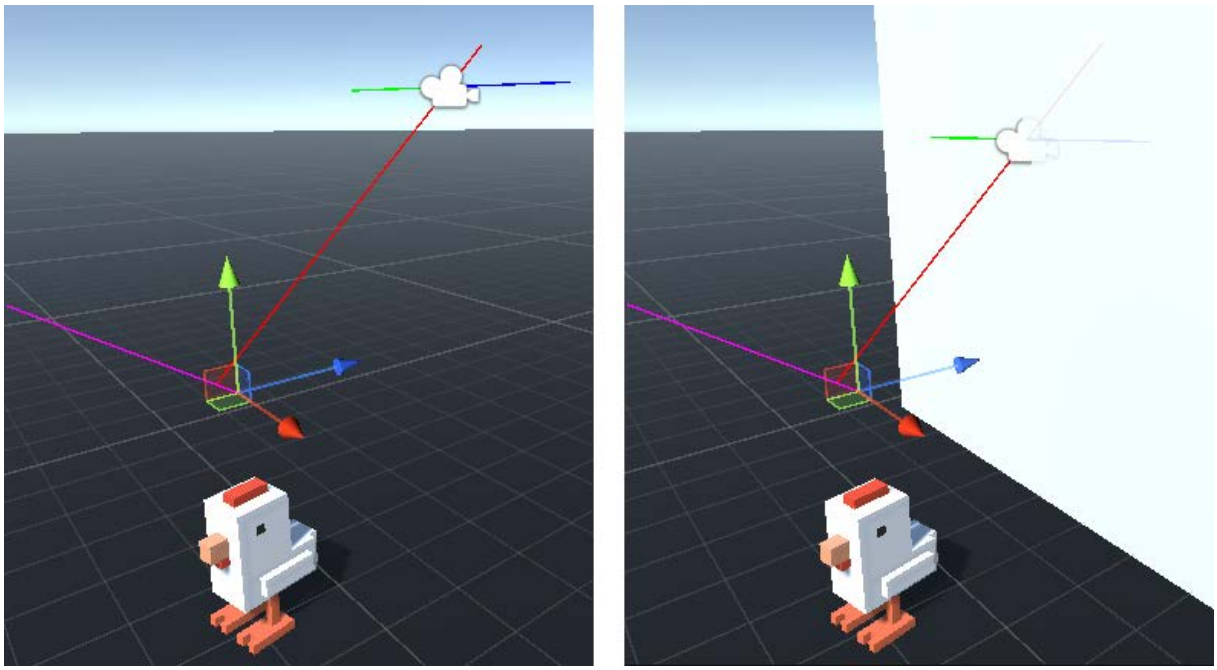
- Ένα GameObject δημιουργείται με το όνομα “Camera” στο οποίο προστίθεται το Component Camera, το οποίο συνοδεύεται και με το component AudioListener, του οποίου η λειτουργικότητα είναι να λαμβάνει τον ήχο μέσα στη σκηνή.
- Το GameObject “Camera” παίρνει τη θέση και τη περιστροφή των ματιών του παίκτη και γίνεται στη συνέχεια child του χαρακτήρα του παίκτη.
- Έπειτα η μεταβλητή CullingMask του Component Camera θα πάρει όλες τις τιμές εξαιρουμένης της τιμής “TranstparentFX”. Η μεταβλητή αυτή ορίζει ποια GameObjects θα γίνουν render μέσω της συγκεκριμένης κάμερας. Αφαιρώντας το Layer “TransparentFX” από τη, λίστα σημαίνει ότι η κάμερα δεν θα κάνει render οποιοδήποτε GameObject έχει Layer “TransparentFX”. Το συγκεκριμένο Layer έχει δοθεί μέχρι στιγμής στον χαρακτήρα του παίκτη, κάνοντάς τον με αυτόν τον τρόπο αόρατο στην κάμερα, ώστε να μην βλέπει ο παίκτης τον ίδιο του τον χαρακτήρα και να αποφευχθούν ανεπιθύμητα glitches.
- Τέλος προστίθεται στον χαρακτήρα του παίκτη το Script “CameraRotation”, που είναι υπεύθυνο για την κατακόρυφη κίνηση της κάμερας η οποία γίνεται με χρήση του ποντικιού του χρήστη. Η κατακόρυφη κίνηση της κάμερας περιορίζεται στις 180 μοίρες ώστε να είναι όσο ποιο ρεαλιστική γίνεται. Ο κώδικας που υλοποιεί αυτήν την λειτουργικότητα αυτή φαίνεται παρακάτω.

```
1.      Vector2 mouselook;
2.      Vector2 smoothV;
3.
4.      public float sencitivity = 5.0f;
5.      public float smoothing = 2.0f;
6.
7.      GameObject target;
8.
9.      // Use this for initialization
10.     void Start()
11.     {
12.         target = transform.root.gameObject;
13.     }
14.
15.     // Update is called once per frame
16.     void LateUpdate()
17.     {
18.         Vector2 md = new Vector2(Input.GetAxisRaw("Mouse X"),
    Input.GetAxisRaw("Mouse Y"));
19.
20.         md = Vector2.Scale(md, new Vector2(sencitivity * smoothing,
    sencitivity * smoothing));
21.         smoothV.x = Mathf.Lerp(smoothV.x, md.x, 1f / smoothing);
22.         smoothV.y = Mathf.Lerp(smoothV.y, md.y, 1f / smoothing);
23.         mouselook += smoothV;
24.         mouselook.y = Mathf.Clamp(mouselook.y, -90f, 90f);
25.
26.         transform.localRotation = Quaternion.AngleAxis(-mouselook.y,
    Vector3.right);
27.
28.         //Parent rotation
29.         target.transform.Rotate(Vector3.up * Input.GetAxis("Mouse X") *
    sencitivity);
30.     }
```

Εικόνα 19: Ο κώδικας του Script "CameraRotation", υπεύθυνος για την κατακόρυφη περιστροφή της κάμερας και περιστροφής του χαρακτήρα του παίκτη

3.3.3.6.2 Κάμερα τρίτου προσώπου

Η κάμερα τρίτου προσώπου είναι μια οπτική που αποδίδεται από σταθερή απόσταση πίσω και λίγο πάνω από τον χαρακτήρα του παίκτη. Στην οπτική αυτή η κάμερα ακολουθεί την θέση αλλά και την περιστροφή του χαρακτήρα του παίκτη, ενώ η σχετική απόσταση μεταξύ τους παραμένει σταθερή. Μια άλλη προϋπόθεση της κάμερας είναι να ανιχνεύει τις συγκρούσεις με άλλα αντικείμενα έτσι ώστε να αποτρέπετα το σενάριο του να περνάει μέσα από άλλα αντικείμενα που ενδέχεται να εμποδίσουν το οπτικό της πεδίο.



Εικόνα 20: Παράδειγμα διαχείρισης σύγκρουσης της κάμερας

Η μέθοδος AddCamera προσθέτει την κάμερα τρίτου προσώπου στο GameObject κάνοντας instantiate ως child στο GameObject ένα έτοιμο prefab με το όνομα “ThirdPersonCamera”, δίνοντάς του επίσης τη θέση και τη περιστροφή του GameObject. Το prefab αυτό πρόκειται για μια προ υλοποιημένη κάμερα τρίτου προσώπου η οποία αποτελείται από τέσσερα child GameObjects ιεραρχικά συνδεδεμένα μεταξύ τους.

- Το GameObject Camera: Περιέχει το component Camera.
- Το GameObject SteadyCameraPoint: Ορίζει την αρχική θέση της κάμερας. Περιέχει επίσης το script RotateCharacter το οποίο είναι υπεύθυνο για την οριζόντια περιστροφή του παίκτη και την κατακόρυφη περιστροφή του SteadyCameraPoint, ανάλογα με την κίνηση του ποντικιού. Το ο κώδικας είναι παρόμοιος με αυτόν της Εικόνας 3.
- Το GameObject SteadyPlayerPoint: Ορίζει ένα σημείο που θα βρίσκετε πάντα σταθερά στο κέντρο του χαρακτήρα του παίκτη.
- Το GameObject CameraCollisionPoint: Περιέχει το script CameraCollision το οποίο είναι υπεύθυνο για την αποφυγή συγκρούσεων της κάμερας με άλλα αντικείμενα. Αυτό επιτυγχάνετε εκτοξεύοντας ένα Ray από το SteadyCameraPoint προς το SteadyPlayerPoint και σε περίπτωση που το Ray χτυπήσει οτιδήποτε η κάμερα μεταφέρεται ποιο κοντά στον χαρακτήρα του παίκτη. Άμα η κάμερα έρθει πολύ κοντά στον παίκτη, τότε το layer του παίκτη αφόρητε από τη λίστα των Layers που κάνει render η κάμερα, καθιστώντας τον χαρακτήρα αόρατο σε αυτήν. Ο κώδικας που υλοποιεί την συγκεκριμένη λειτουργικότητα φαίνεται παρακάτω.

```
1. void Update()
2.     {
3. Ray backRay = new Ray(steadyCameraPoint.position, -
   steadyCameraPoint.forward * (cameraBackDistance + 1));
4.
5. FindClosestHitObject(backRay, out backHitPoint, out backDistance, ~(1 <<
   1));
6.
7.         //If the ray hits somthing move camera to the hitpoint
8.         if (backDistance != 0 && backDistance <= cameraBackDistance)
9.             transform.position = backHitPoint;
10.        //Return the camera to the wanted position
11.        else
12.            transform.localPosition = new Vector3(0, 0, -
   cameraBackDistance);
13. }
14.
15. Transform FindClosestHitObject(Ray ray, out Vector3 hitPoint, out float
   distance, int layerMask = -1)
16.     {
17.         RaycastHit[] hits = Physics.RaycastAll(ray, 1000, layerMask);
18.
19.         Transform closestHit = null;
20.         distance = 0;
21.         hitPoint = Vector3.zero;
22.
23.         foreach (RaycastHit hit in hits)
24.         {
25.             if (hit.transform.root.tag != "Player" && hit.transform !=
   transform && (closestHit == null || hit.distance < distance))
26.             {
27.                 // We have hit something that is:
28.                 // a) not us
29.                 // b) the first thing we hit (that is not us)
30.                 // c) or, if not b, is at least closer than the previous
   closest thing
31.
32.                 closestHit = hit.transform;
33.                 distance = hit.distance;
34.                 hitPoint = hit.point;
35.             }
36.         }
37.
38.         // closestHit is now either still null (i.e. we hit nothing) OR
   it contains the closest thing that is a valid thing to hit
39.
40.         return closestHit;
```

Εικόνα 21: Ο κώδικας του script CameraCollision, υπεύθυνος για την διαχείριση των συγκρούσεων της κάμερας με άλλα GameObjects

3.3.3.7 Τα physics

Τα Physics είναι ένα χαρακτηριστικό που πλέον κατέχει σχεδόν κάθε σύγχρονο παιχνίδι. Πρόκειται για την εισαγωγή κανόνων φυσικής πάνω στα διάφορα αντικείμενα της σκηνής όπως μάζα, βαρύτητα, επιτάχυνση, κ.λπ.. Αυτό επιτυγχάνεται μέσω του Component του Unity3D που ονομάζεται Rigidbody.

Την προσθήκη αυτή των Physics καλείται να αναλάβει η μέθοδος AddPhysics. Αν η παράμετρος "hasPhysics" του αντικειμένου στο αρχείο Json έχει οριστεί ως true, τότε θα του προστεθεί ένα Rigidbody component. Μια παρέμβαση γίνεται όμως στο Component, στη περίπτωση που το αντικείμενο έχει οριστεί ως "isControllable" στο αρχείο Json, και αυτή είναι να μείνει μονίμως σταθερή η περιστροφή του αντικειμένου μέσω τον physics, ώστε η κίνηση του χαρακτήρα του παίκτη να γίνεται αποκλειστικά μέσω του κώδικα και να στέκετε πάντα όρθιος. Μια ακόμα παρέμβαση γίνεται σε περίπτωση που η παράμετρος του αντικειμένου στο αρχείο Json "hasGravity" έχει οριστεί ως true ή false. Στην περίπτωση αυτή η επίδραση της βαρύτητας στο αντικείμενο θα ενεργοποιηθεί ή απενεργοποιηθεί αντίστοιχα.

3.3.3.8 Οι συγκρούσεις

Προκειμένου τα αντικείμενα μιας σκηνής να μπορούν να αλληλοεπιδράσουν μεταξύ τους θα πρέπει να εισαχθεί σε κάθε GameObject ένα Component που ονομάζεται Collider. Η λειτουργικότητα του Component αυτού είναι να προσομοιώνει τις συγκρούσεις μεταξύ των αντικειμένων ώστε να μην περνάει το ένα μέσα από το άλλο. Το Unity3D παρέχει έξι ειδών τρισδιάστατων colliders:

- BoxCollider,
- CapsuleCollider,
- MeshCollider,
- SphereCollider,
- TerrainCollider,
- WheelCollider.

Στα περισσότερα από αυτά τα colliders το σχήμα τους και η λειτουργικότητά τους υποδεικνύεται από το όνομα τους. Αυτό που ξεχωρίζει είναι το MeshCollider το οποίο πρόκειται για ένα collider το οποίο σχηματίζεται δυναμικά κατά την προσθήκη του αντίστοιχου component,

παίρνοντας το σχήμα του Mesh του μοντέλου του.

Το αρχείο Json περιέχει 2 παραμέτρους που σχετίζονται με τα colliders τις "hasColliders" και "colliderType". Η πρώτη, ανάλογα με την τιμή που θα πάρει (true η false), υποδεικνύει το αν το αντικείμενο θα έχει Colliders η όχι. Η δεύτερη υποδεικνύει το είδος του collider που θα του εισαχθεί.

Η μέθοδος που αναλαμβάνει την εισαγωγή colliders στα GameObjects είναι η μέθοδος AddCollider, η οποία θα εκτελεστεί αν η τιμή του "hasColliders" είναι true. Το πρώτο πράγμα που θα ελέγξει η μέθοδος είναι αν το συγκεκριμένο GameObject η κάποιο από τα child GameObjects του περιέχει Component MeshFilter, δηλαδή αν το GameObject είναι ένα μοντέλο που έχει κάποιο Mesh και δεν είναι απλά ένα αφηρημένο κενό GameObject.

Αν το GameObject ή κάποιο από τα child GameObjects του δεν περιέχει Component MeshFilter ή αν η τιμή της μεταβλητής "isControllable' του συγκεκριμένου αντικειμένου έχει οριστεί ως true, τότε το

GameObject αυτό θα λάβει έναν collider τύπου CapsuleCollider. Ο collider αυτός στη συνέχεια θα πρέπει να πάρει το μήκος και το πλάτος του GameObject ώστε η σύγκρουση να είναι ακριβής και ρεαλιστική. Την λειτουργία αυτή αναλαμβάνει η μέθοδος FitColliderSizeToChildrenSize, η οποία λειτουργεί με τον εξής τρόπο: Για κάθε child του GameObject προσθέτει τα μήκη και τα πλάτη των meshes τους χρησιμοποιώντας τη μέθοδο bounds.Encapsulate και στη συνέχεια ορίζει το μήκος και το πλάτος του collider ίσο με το μήκος και το πλάτος του Encapsulated Bound. Ο κώδικας της μεθόδου αυτής φαίνεται παρακάτω.

```
1.  void FitColliderSizeToChildrenSize(GameObject go)
2.      {
3.          if (go.GetComponent<Collider>() is CapsuleCollider)
4.          {
5.              bool hasBounds = false;
6.              Bounds bounds = new Bounds(Vector3.zero, Vector3.zero);
7.              for (int i = 0; i < go.transform.childCount; ++i)
8.              {
9.                  Renderer childRenderer =
    go.transform.GetChild(i).GetComponent<Renderer>();
10.                 if (childRenderer != null)
11.                     if (hasBounds)
12.                         bounds.Encapsulate(childRenderer.bounds);
13.                     else
14.                     {
15.                         bounds = childRenderer.bounds;
16.                         hasBounds = true;
17.                     }
18.             }
19.
20.             CapsuleCollider collider =
    (CapsuleCollider)go.GetComponent<Collider>();
21.             collider.center = bounds.center - go.transform.position;
22.             collider.height = bounds.size.y;
23.             collider.center = new Vector3(0, collider.center.y, 0);
24.         }
25.     }
```

Εικόνα 22: Ο κώδικας της μεθόδου FitColliderSizeToChildrenSize, η οποία θέτει το μήκος και το πλάτος του collider ίσο με αυτό του mesh του GameObject

3.3.3.9 Η εισαγωγή κάποιων βασικών scripts

Τέλος, θα προστεθούν κάποια βασικά script σε κάθε αντικείμενο που θα καθορίσουν τον ρόλο του στη σκηνή, θα του δώσουν κάποια βασικές ιδιότητες ή θα του δώσουν κάποια ειδική λειτουργικότητα ανάλογα με τον ρόλο αυτόν. Το πρώτο script που θα προστεθεί στο GameObject ως Component είναι το script Character ή Thing, το οποίο θα ορίσει το ήδος του αντικειμένου, δηλαδή αν πρόκειται για χαρακτήρα ή πράγμα αντίστοιχα. Το script αυτό περιέχει δύο μεταβλητές:

- maxhealth: Η float μεταβλητή αυτή ορίζει την ζωή που θα έχει ο παίκτης. Η μεταβλητή αυτή θα πάρει τιμή ανάλογα με την τιμή που έχει η αντίστοιχη μεταβλητή στο αρχείο Json.
- takesDamage: Η bool μεταβλητή αυτή ορίζει, αν ο παίκτης είναι ικανός να χάσει ζωή (πιθανών από κάποιο χτύπημα). Η μεταβλητή αυτή θα πάρει τιμή ανάλογα με την τιμή που έχει η αντίστοιχη μεταβλητή στο αρχείο Json.

Το script περιέχει επίσης και την μέθοδο TakeDamage(float damage) που κατά τη κλήση της θα κατεβάσει τη ζωή του παίκτη κατά ένα ποσό damage.

Το δεύτερο script που θα εισαχθεί ως component στο GameObject είναι το Script υπεύθυνο για την κίνηση του αντικειμένου που ονομάζεται PlayerController3D.

3.3.3.10 Η κίνηση του χαρακτήρα του παίκτη

Την κίνηση του χαρακτήρα του παίκτη αναλαμβάνει το script PlayerController3D. Πέρα από την κίνηση του χαρακτήρα, το script αυτό είναι υπεύθυνο και για την εναλλαγή των animations, μιας και σχετίζεται και αυτό με την κίνηση του παίκτη.

Το script αυτό έχει πέντε παραμετροποιήσιμες από τον χρήστη μεταβλητές

- Float walkSpeed: Η ταχύτητα του χαρακτήρα όταν περπατάει.
- Float runSpeed: Η ταχύτητα του χαρακτήρα όταν τρέχει.
- Float jumpForce: Το ύψος του άλματος του χαρακτήρα.
- LayerMask groundedMask: Το Layer το οποίο ορίζει τα αντικείμενα που το έχουν ως έδαφος για τον χαρακτήρα.
- Float groundedRayLength: Το μέγεθος της απόστασης κάτω από τον χαρακτήρα στην οποία ο χαρακτήρας θεωρείται ότι πατάει στο έδαφος.

Η κίνηση του παίκτη συμβαίνει μέσα στη μέθοδο FixedUpdate η οποία για κάθε Frame του παιχνιδιού ορίζει το velocity του Component Rigidbody ίσο με “new Vector3(moveVector.x, rb.velocity.y, moveVector.z)” όπου moveVector, μια Vector3 μεταβλητή η οποία παίρνει τιμή από τον παρακάτω κώδικα.

```
1. Vector3 playerMovementInput = new
   Vector3(Input.GetAxisRaw("Horizontal"), 0,
   Input.GetAxisRaw("Vertical")).normalized;
2.
3. Vector3 moveVector = transform.TransformDirection(playerMovementInput) *
   speed;
```

Εικόνα 23: Το Vector3 που ορίζει την τιμή της κίνησης του αντικειμένου

Έπειτα με το πάτημα του κουμπιού shift ορίζεται η αλλαγή της ταχύτητας του αντικειμένου από walkSpeed σε runSpeed, ενώ με το πάτημα του κουμπιού Space εφαρμόζεται μια κάθετη δύναμη στο αντικείμενο, ίση με την τιμή του jumpForce, η οποία προσομοιώνει το πήδημα. Το πήδημα βέβαια θα συμβεί μόνο άμα ισχύει ο παρακάτω περιορισμός: άμα δηλαδή τα πόδια του χαρακτήρα απέχουν από οτιδήποτε έχει οριστεί ως έδαφος για τον χαρακτήρα (μέσο της τιμής του groundedMask), τόσο όσο ορίζει η τιμή του groundedRayLength.

Αφού έχει εξασφαλιστεί η κίνηση του αντικειμένου σειρά έχει η ανάλλαγή των animations. Η αλλαγή των animations όπως έχει προαναφερθεί συμβαίνει μέσω της αλλαγής των δύο Parameters στο Component Animator, τις Speed και Jumping. Η παράμετρος Speed παίρνει τιμή μέσω της μεθόδου Update όπου καταλήγει να ισούται με το μέγεθος της επιτάχυνσης του αντικειμένου, μέσω της

εντολής ‘`GetComponent<Animator>().SetFloat("Speed", rb.velocity.magnitude)`”, ενώ η παράμετρος Jumping αλλάζει από true σε false και το αντίθετο κατά το πήδημα.

3.3.4 Η δημιουργία του 2D παιχνιδιού

Παρομοίως με την 3D σκηνή, έπειτα από τη δημιουργία του αντικειμένου SceneImporter2D μέσω του SceneImporterExecutor ξεκινάει η διαδικασία δυναμικής δημιουργίας της σκηνής [25]. Ο δομητής της κλάσης εφόσον μετατρέψει και αποθηκεύσει τα περιεχόμενα του αρχείου Json ως Dictionary στην μεταβλητή JsonDataModel, προβαίνει στον καθαρισμό της σκηνής και στην προσθήκη των νέων GameObjects καλώντας τις 2 βασικές μεθόδους: ClearScene και SpawnObjects.

3.3.4.1 Η προετοιμασία της σκηνής

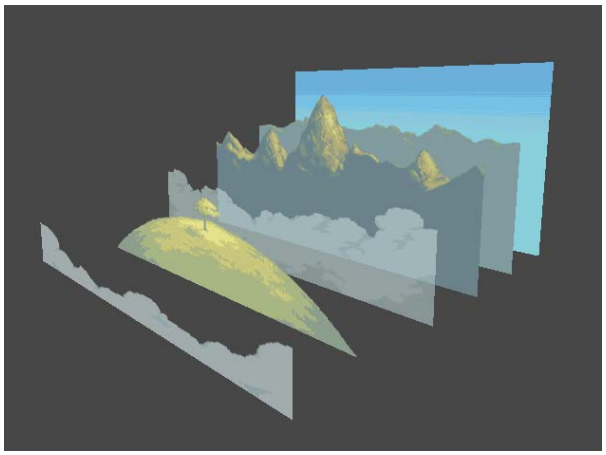
Όπως και κατά τη δημιουργία της 3D σκηνής, το πρώτο πράγμα που θα πρέπει να γίνει είναι να καθαριστεί η σκηνή από οποιαδήποτε objects μπορεί να προϋπάρχουν. Την λειτουργία αυτή καλείται να αναλάβει η μέθοδος ClearScene. Αυτό γίνεται με τη κλήση της μεθόδου DestroyImmediate για κάθε root GameObject στη σκηνή.

Στη συνέχεια ακολουθεί η προσθήκη των νέων GameObjects που ξεκινά από τη μέθοδο SpawnObjects η οποία αναλαμβάνει την προσθήκη εδάφους και των μοντέλων που έχει παρέχει ο χρήστης καλώντας τις μεθόδους: AddGround και AddModels αντίστοιχα, ενώ σε αντίθεση με τη 3D σκηνή θα προστεθεί και ένα background, με τη χρήση της μεθόδου AddBackground.

3.3.4.2 Το background

Το background είναι ένα πολύ σημαντικό αισθητικό στοιχείο ενός 2D παιχνιδιού. Υπάρχουν 2 τεχνικές υλοποίησης ενός 2D background.

- Ο πρώτος τρόπος είναι να γίνει χρήση του Parallax effect το οποίο πρόκειται για ένα σύνολο από background εικόνες, τοποθετημένες η μία πίσω από την άλλη σε διαφορετικές αποστάσεις. Οι εικόνες αυτές κινούνται ελαφρώς σχετικά με την θέση του χαρακτήρα του παίκτη δίνοντας έτσι την αίσθηση του βάθους στον παίκτη.



Εικόνα 24: Παράδειγμα Parallax effect

- Ο δεύτερος τρόπος, ο οποίος είναι και αυτός που χρησιμοποιεί το εργαλείο για να εισάγει 2D background στη σκηνή μέσω της μεθόδου AddBackground, είναι μια στατική εικόνα.

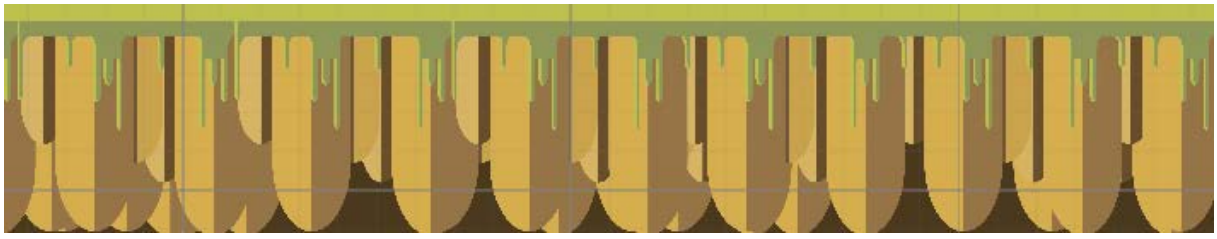
Προκειμένου να εισαχθεί το background θα δημιουργηθεί ένα empty GameObject στο κέντρο της σκηνής με το όνομα "Background". Η συντεταγμένη "z" της θέσης του όπως θα πάρει την τιμή 1 που θα υποδηλώνει ότι αυτό το sprite θα βρίσκετε πίσω από όλα τα μελλοντικά sprites που θα εισαχθούν στη σκηνή. Στη συνέχεια θα προστεθεί ένα Component τύπου SpriteRenderer. Το Component αυτό είναι ένα πολύ σημαντικό στοιχείο για χρήση κατά την ανάπτυξη 2D projects, καθώς καθορίζει τον τρόπο που θα γίνει render ένα texture. Το Component αυτό έχει περιέχει το property sprite το οποίο θα πάρει ως τιμή το texture sprite (δηλαδή την εικόνα) "Background" από τα αρχεία του χρήστη. Επειδή όπως οι εικόνες που θα παραδώσει ο χρήστης μπορεί πολλές φορές να μην είναι στη μορφή Sprite αλλά στη μορφή Default, θα πρέπει πρώτα να μετατραπούν σε sprite. Αυτό γίνεται με την παρακάτω εντολή.

```
1.  Sprite.Create(texture2D, new Rect(0, 0, texture2D.width, texture2D.height), new
    Vector2(0.5f, 0.5f));
```

Εικόνα 25: Η μετατροπή της εικόνας σε μορφή sprite

3.3.4.3 Το έδαφος

Έπειτα η μέθοδος AddGround προσθέτει ένα στοιχειώδες έδαφος στη σκηνή. Το έδαφος αυτό είναι ένα σύνολο από αντίγραφα ενός προκαθορισμένου sprite τοποθετημένα το ένα δίπλα στο άλλο έχοντας τυχαία τιμή στη συντεταγμένη "x" στη θέση τους. Έτσι δημιουργείται ένα έδαφος όπως αυτό της παρακάτω εικόνας.



Εικόνα 26: Πρότυπο δημιουργίας εδάφους

3.3.4.4 Τα υπόλοιπα αντικείμενα

Έπειτα η μέθοδος AddModels υλοποιεί την λειτουργικότητά της καλώντας έναν βρόγχο foreach ο οποίος εκτελείται για κάθε αντικείμενο μέσα στο περιεχόμενο του Json. Για κάθε αντικείμενο λοιπόν:

a. Δημιουργείται ένα GameObject για κάθε αντικείμενο μέσα στο Json αρχείο και προστίθεται σε αυτό το Component SpriteRenderer. Έπειτα φορτώνεται από τα αρχεία του χρήστη η αντίστοιχη εικόνα και αφού μετατραπεί σε sprite μέσω της εντολής της εντολής στην Εικόνα 11 προστίθεται στη σκηνή.
b. Άμα το GameObject ορίζεται ως Controllable στο Json, το Layer του αντικειμένου ορίζεται ως «TransparentFX», ώστε να διαφοροποιηθεί από τα υπόλοιπα GameObjects.
c. Ορίζεται, με βάση τις τιμές του Json, το Position, το Rotation και το Scale του GameObject.

d. Άμα το GameObject ορίζεται από το Json να έχει animations, εκτελείται η μέθοδος AddAnimations, ενώ άμα ορίζεται να έχει main camera, εκτελείται η μέθοδος AddCamera.
e. Τέλος εκτελούνται οι τρεις μέθοδοι, AddPhysics, AddCollider και AddBasicScripts, οι οποίες είναι υπεύθυνες για την προσθήκη των physics, των colliders και μερικών βασικών gameplay scripts αντίστοιχα.

3.3.4.5 Τα animations

Το κομμάτι της προσθήκης των animations καλείται να αναλάβει η μέθοδος AddAnimations. Προκειμένου ένα GameObject στο Unity3D να έχει Animations πρέπει να γίνει χρήση του Component "Animator". Από κει και πέρα, για να λειτουργήσει το animation θα πρέπει να οριστεί η βασική μεταβλητή του Component αυτού, η μεταβλητή Controller.

Για τους σκοπούς λοιπόν της δυναμικής προσθήκης animations σε κάθε αντικείμενο που έχει προμηθεύσει ο χρήστης του εργαλείου SceneImporter, χρησιμοποιείται το προκαθορισμένο RuntimeAnimatorContoller που έχει χρησιμοποιηθεί και κατά την προσθήκη 3D animations.

Η πρώτη λειτουργεία της μεθόδου AddAnimations λοιπόν είναι προσθέσει ένα Component τύπου Animator στο GameObject. Έπειτα πρέπει να οριστεί η μεταβλητή Controller, η οποία θα πάρει ως τιμή το προκαθορισμένο RuntimeAnimatorController που έχει προαναφερθεί. Στη συνέχεια θα δημιουργηθεί ένα αντικείμενο της κλάσης AnimatorOverrideController. Εδώ πρέπει να σημειωθεί ότι ο χρήστης πρέπει να παραδώσει τα animations με έναν συγκεκριμένο τρόπο προκειμένου να χρησιμοποιηθούν από το εργαλείο.

Ένα 2D animation δεν είναι τίποτα άλλο παρά εικόνες που εναλλάσσονται γρήγορα η μία μετά την άλλη [26]. Γι' αυτό θα πρέπει και ο χρήστης να παραδώσει ως αρχεία, τις τρεις καταστάσεις ("Idle", "Walking", "Running" και "Jumping) με τη μορφή φακέλων με αντίστοιχα ονόματα, που το καθένα θα περιέχει με τη σειρά τις εικόνες που θα αποτελέσουν το τελικό animation. Στη συνέχεια η μέθοδος θα αντιστοιχεί τα overridden AnimationClips του AnimatorOverrideController με τα καινούρια animations τα οποία θα δημιουργηθούν από τη μέθοδο CreateAnimationClip που παίρνει ως παραμέτρους το GameObject και το όνομα του animation, και η οποία θα επεξηγηθεί παρακάτω.

Το πρώτο βήμα της μεθόδου είναι να φορτώσει σε μια μεταβλητή πίνακα textures2D όλες τις εικόνες του φακέλου του οποίου το όνομα είναι ίδιο με το όνομα της δεύτερης παραμέτρου της μεθόδου. Έπειτα κάθε εικόνα μετατρέπεται για άλλη μια φορά σε sprite με τη χρήση της μεθόδου της Εικόνα 11 και να αποθηκευτεί σε μια μεταβλητή πίνακα sprites. Στη συνέχεια δημιουργείτε δυναμικά ένα αντικείμενο της κλάσης AnimationClip το οποίο θα πάρει ως όνομα το όνομα του animation και ένα προκαθορισμένο frameRate ίσο με 25 fps (frames per seconds). Συνεχίζοντας θα πρέπει το κάθε ένα από αυτά τα animations αν κάνουνε loop δηλαδή να επαναλαμβάνονται ξανά και ξανά από την αρχή έως ότου αλλάξουν, εκτός του animation "Jumping" το οποίο θέλουμε να εκτελεστεί μία φορά και να παραμείνει στην τελική του εικόνα έως ότου αλλάξει. Αυτό επιτυγχάνεται με τη χρήση των παρακάτω εντολών:

```
1. AnimationClipSettings settings =
   AnimationUtility.GetAnimationClipSettings(animClip);
2.             settings.loopTime = true;
3.             AnimationUtility.SetAnimationClipSettings(animClip,
   settings);
```

Εικόνα 27: Ο κώδικας που αλλάζει το loopTime του animation

Σειρά έχει να δημιουργηθεί το animation χρησιμοποιώντας τα sprites της μεταβλητής sprites. Προκειμένου να δημιουργηθεί ένα animation από εικόνες που εναλλάσσονται θα πρέπει να ακολουθηθεί η παρακάτω διαδικασία:

- Πρώτα θα δημιουργηθεί ένα αντικείμενο spriteBinding της κλάσης EditorCurveBinding στο οποίο θα οριστούν οι εξής τιμές στις μεταβλητές του: type = typeof(SpriteRenderer), Path = "", και propertyName = "m_Sprite". Αυτές οι τρεις παράμετροι δηλώνουν αντίστοιχα: ποιο Component θα επηρεαστεί από το sprite binding αυτό, το path του Object που θα γίνει animated ("" σηματοδοτεί το root Object), και όνομα του property που θα γίνει animated.
- Στη συνέχεια θα δημιουργηθεί ένα keyFrame για κάθε εικόνα το οποίο θα προστεθεί το ένα σε απόσταση μερικά εκατοστά του δευτερολέπτου από το άλλο, ορίζοντας μια αρχική ταχύτητα για το animation.

Ο κώδικας που υλοποιεί την παραπάνω λειτουργία φαίνεται παρακάτω.

```
1. //Create animationClip Curve of type SpriteRenderer
2.         EditorCurveBinding spriteBinding = new EditorCurveBinding();
3.         spriteBinding.type = typeof(SpriteRenderer);
4.         spriteBinding.path = "";
5.         spriteBinding.propertyName = "m_Sprite";
6.
7.         //Create
8.         ObjectReferenceKeyframe[] spriteKeyFrames = new
   ObjectReferenceKeyframe[sprites.Length];
9.         for (int i = 0; i < (sprites.Length); i++)
10.        {
11.            spriteKeyFrames[i] = new ObjectReferenceKeyframe();
12.            spriteKeyFrames[i].time = i / 10f;
13.            spriteKeyFrames[i].value = sprites[i];
14.        }
15.
16.        AnimationUtility.SetObjectReferenceCurve(animClip,
   spriteBinding, spriteKeyFrames);
```

Εικόνα 28: Ο κώδικας που δημιουργεί τα AnimationClips

3.3.4.6 Η κάμερα

Η κάμερα στα 2D παιχνίδια είναι ένα πολύ απλό concept. Υπάρχουν δύο τρόποι χρήσης της κάμερας:

- Στατική κάμερα: Μια κάμερα που παραμένει ακίνητη σε μια συγκεκριμένη θέση κάνοντας render ολόκληρη τη σκηνή.
- Κινούμενη κάμερα: Η κάμερα είναι child ενός GameObject με αποτέλεσμα να ακολουθά πιστά παντού τη θέση του παίκτη, ή απλά αιωρείται γύρο από κάποιο αντικείμενο.

Το εργαλείο κάνει χρήση της δεύτερης μεθόδου δημιουργώντας ένα αντικείμενο “MainCamera”, με Component Camera και βάζοντας το, ως child, στο αντικείμενο του αρχείου, που έχει την τιμή true ως τιμή στο property “hasCamera”. Έπειτα από τη στιγμή που μιλάμε για δισδιάστατη προβολή της κάμερας, η αίσθηση του βάθους δεν είναι απαραίτητη, οπότε θα πρέπει να γίνει μετατροπή της κάμερας από Perspective σε Orthographic. Αυτό γίνεται αλλάζοντας τη μεταβλητή Projection του Component Camera από Perspective σε Orthographic.

3.3.4.7 Τα physics

Όπως και κατά την δημιουργία του 3D παιχνιδιού θα πρέπει να εισαχθούν κάποια Physics σε κάθε αντικείμενο ώστε να επιτευχθεί μια κάποια ρεαλιστικότατα. Η διαφορά με μια τρισδιάστατη σκηνή είναι ότι τα physics τώρα πρέπει να ακολουθούν τη λογική ενός δισδιάστατου κόσμου. [27] Physics θα λάβουν για ακόμα μια φορά μόνο τα αντικείμενα τα οποία έχουν true ως τιμή στο πεδίο “hasCamera”. Από τη στιγμή που ικανοποιείται αυτός ο περιορισμός, θα προστεθεί στο GameObject το Component Rigidbody2D, το οποίο είναι μια δισδιάστατη παραλλαγή του Component Rigidbody. Αν όμως η τιμή του πεδίου “isControllable” στο αρχείο Json είναι true, θα πρέπει η περιστροφή του αντικειμένου μέσω τον physics να μείνει μονίμως σταθερή, ώστε η κίνηση του χαρακτήρα του παίκτη να γίνεται αποκλειστικά μέσω του κώδικα και να στέκετε πάντα όρθιος. Αυτό γίνεται παγώνοντας τις συντεταγμένες “x” και “y” συντεταγμένες στο πεδίο Constraints στις παραμέτρους του Rigidbody2D Component. Μια ακόμα παρέμβαση γίνεται στο Rigidbody2D σε περίπτωση που η παράμετρος του αντικειμένου στο αρχείο Json “hasGravity” έχει οριστεί ως true ή false. Στην περίπτωση αυτή η επίδραση της βαρύτητας στο αντικείμενο θα ενεργοποιηθεί ή απενεργοποιηθεί αντίστοιχα ορίζοντας την τιμή της μεταβλητής gravityScale του RigidBody2D σε “0”.

3.3.4.8 Οι συγκρούσεις

Σε αντίθεση με τη δημιουργία της τρισδιάστατης σκηνής, η εισαγωγή colliders σε 2D αντικείμενα είναι ποιο απλή διαδικασία. Αρχικά θα εισαχθεί πάλι σε κάθε GameObject ένα Component που ονομάζεται Collider. Το Unity3D παρέχει έξι ειδών δισδιάστατων colliders: BoxCollider2D, CapsuleCollider2D, PolygonCollider2D, CircleCollider2D, CompositeCollider2D, EdgeCollider2D. Σε όλα αυτά τα colliders το σχήμα τους και η λειτουργικότητά τους υποδεικνύεται από το όνομα τους. Αυτό που ξεχωρίζει είναι το PolygonCollider2D το οποίο πρόκειται για ένα collider το οποίο σχηματίζεται δυναμικά κατά την προσθήκη του αντίστοιχου component, παίρνοντας το σχήμα του texture του Sprite πάνω στο οποίο εφαρμόζεται.

Όπως και για τη 3D σκηνή, το αρχείο Json περιέχει 2 παραμέτρους που σχετίζονται με τα colliders τις “hasColliders” και “colliderType”. Η πρώτη, ανάλογα με την τιμή που θα πάρει (true η false), υποδεικνύει το αν το αντικείμενο θα έχει Colliders η όχι. Η δεύτερη υποδεικνύει το είδος του collider που θα του εισαχθεί.

3.3.4.9 Η εισαγωγή κάποιων βασικών scripts

Τέλος, θα προστεθούν κάποια βασικά scrip ως components σε κάθε αντικείμενο που θα ορίζουν τις ιδιότητες και θα υλοποιούν κάποιες βασικές λειτουργείες του αντικειμένου. Ανάλογα με το αν το αντικείμενο θεωρείτε Character ή Thing (πράγμα που ορίζεται από την τιμή της παραμέτρου

“isControllable”: true αν πρόκειται για χαρακτήρα και false αν πρόκειται για πράγμα). Όπως και στη 3D version, το script αυτό περιέχει τις δύο μεταβλητές maxHealth και takesDamage καθώς και την μέθοδο TakeDamage, που σχετίζονται με τη ζωή του παίκτη.

Το δεύτερο script που θα εισαχθεί ως component στο GameObject είναι το Script υπεύθυνο για την κίνηση του αντικειμένου που ονομάζεται PlayerController2D.

3.3.4.10 Η κίνηση του χαρακτήρα του παίκτη

Την κίνηση του χαρακτήρα του παίκτη αναλαμβάνει το script PlayerController2D. Πέρα από την κίνηση του χαρακτήρα, το script αυτό είναι υπεύθυνο και για την εναλλαγή των animations, μιας και σχετίζεται και αυτό με την κίνηση του παίκτη [28].

Το script αυτό έχει εννιά παραμετροποιήσιμες από τον χρήστη μεταβλητές

- Float walkingSpeed: Η ταχύτητα του χαρακτήρα όταν περπατάει.
- Float runningSpeed: Η ταχύτητα του χαρακτήρα όταν τρέχει.
- Float m_JumpForce: Το ύψος του άλματος του χαρακτήρα.
- Float m_CrouchSpeed: Η Ποσότητα maxSpeed που εφαρμόζεται κατά το σκύψιμο (όπου 1 = 100%)
- Float m_MovementSmoothing: Η ποσότητα εξομάλυνσης της κίνησης.
- Bool m_AirControl: Ορίζει εάν ένας παίκτης μπορεί ή όχι να στρίψει ενώ πηδά.
- LayerMask m_WhatIsGround: Μια μάσκα που καθορίζει τι θεωρείτο έδαφος για τον χαρακτήρα.
- Transform m_CeilingCheck: Μια θέση πάνω στον χαρακτήρα που δείχνει σε ποιο σημείο θα ελέγχεται η σύγκρουση με τυχόν οροφή.
- Collider2D m_CrouchDisableCollider: Ένας collider που θα απενεργοποιείται κατά τη διάρκεια του σκυψίματος του χαρακτήρα.

Η κίνηση του παίκτη συμβαίνει μέσα στη μέθοδο Update η οποία για κάθε Frame εκτελεί τον παρακάτω κώδικα.

```
1. // Move the character by finding the target velocity
2. Vector3 targetVelocity = new Vector2(move * speed,
   m_Rigidbody2D.velocity.y);
3. // And then smoothing it out and applying it to the character
4. m_Rigidbody2D.velocity = Vector3.SmoothDamp(m_Rigidbody2D.velocity,
   targetVelocity, ref m_Velocity, m_MovementSmoothing);
```

Εικόνα 29: Οι εντολές που υλοποιούν την κίνηση του παίκτη

Επίσης κατά την κίνηση του χαρακτήρα αριστερά και δεξιά, θα πρέπει ταυτόχρονα να εναλλάσσεται και η φορά του sprite. Την λειτουργικότητα αυτή αναλαμβάνει η μέθοδος Flip η οποία καλείται κατά την αλλαγή κατεύθυνσης και πολλαπλασιάζει τις διαστάσεις του GameObject του χαρακτήρα του παίκτη με -1, αναποδογυρίζοντας το sprite.

3.4 Δυναμική εξαγωγή 3D παιχνιδιού σε ένα αρχείο Json

Το δεύτερο κομμάτι της λειτουργικότητας του εργαλείου, είναι η εξαγωγή του παιχνιδιού του Unity3D σε έναν εξωτερικό φάκελο. Αυτό περιλαμβάνει την εξαγωγή των 3D GameObjects ως .fbx αρχεία, των 2D GameObjects ως εικόνες και των ιδιοτήτων τους ως αντικείμενα σε ένα αρχείο Json. Με αυτόν τον τρόπο μία σκηνή μπορεί πολύ εύκολα να μεταφερθεί από ένα project σε ένα άλλο, ακόμα και από ένα Game Engine σε ένα άλλο, χωρίς να χρειαστεί να ανακατασκευαστεί από την αρχή.

Η διαδικασία της εξαγωγής της σκηνής και κατ' επέκταση του παιχνιδιού ξεκινάει αφού ο χρήστης πατήσει την επιλογή "Export Scene" από το μενού επιλογών (βλέπε Εικόνα 1). Η επιλογή αυτή είναι συνδεδεμένη με την static μέθοδο Execute της κλάσης SceneExporterExecutor.

Η πρώτη λειτουργία λοιπόν της μεθόδου Execute είναι να αποθηκεύσει, στη μεταβλητή sceneExportPath, το path στο οποίο θα αποθηκευτούν τα αντικείμενα και το αρχείο Json. Αυτό επιτυγχάνεται με την εκτέλεση της εντολής SaveFolderPanel της βιβλιοθήκης EditorUtility, η οποία εμφανίζει ένα "Open Folder" dialog και κατά τη επιλογή ενός φακέλου από τον χρήστη επιστρέφει το path του φακέλου.

Στη συνέχεια άμα η μεταβλητή sceneExportPath δεν είναι κενή, εμφανίζεται ένα παράθυρο επιβεβαίωσης της εξαγωγής, μέσω της εντολής DisplayDialog, το οποίο προειδοποιεί και τον χρήστη να μην κλείσει το Unity3D κατά τη διάρκεια της διαδικασίας. Από την στιγμή που ο χρήστης θα πατήσει επιβεβαίωση, δημιουργείται ένας φάκελος με το όνομα "ExportedScene" ο οποίος θα περιέχει όλα τα αρχεία της εξαγωγής και στη συνέχεια ξεκινάει η ίδια η διαδικασία της εξαγωγής, η οποία περιγράφεται ποιο αναλυτικά παρακάτω.

3.4.1 Δημιουργία του αρχείου Json

Η διαδικασία ξεκινά με την δημιουργία του αρχείου Json. Ένα νέο αντικείμενο JsonData με το όνομα "jsonData" δημιουργείται, το οποίο θα περιέχει τα δύο βασικά αντικείμενα: το αντικείμενο "projectProperties" στο οποίο αποθηκεύονται κάποιες βασικές ιδιότητες όλου του project και το αντικείμενο "gameObjects" όπου αποθηκεύονται οι ιδιότητες των GameObjects της σκηνής. Στην εκάστοτε έκδοση του εργαλείου, το μόνο project property που αποθηκεύεται είναι η διάσταση του project, δηλαδή "3D" ή "2D". Αυτό επιτυγχάνεται με τη χρήση της εντολής "`SceneView.lastActiveSceneView.in2DMode`" η οποία ελέγχει αν κάμερα του παραθύρου Scene είναι σε δισδιάστατο mode ή σε τρισδιάστατο mode.

Αφού αποθηκευτούν τα project properties σειρά έχουν οι ιδιότητες των GameObject. Η διαδικασία αυτή ξεκινάει με έναν βρόγχο ο οποίος εκτελείται για κάθε root GameObject της σκηνής τα οποία επιστρέφει η συνάρτηση "`UnityEngine.SceneManagement.SceneManager.GetActiveScene().GetRootGameObjects()`".

Για κάθε GameObject λοιπόν της σκηνής (εξαιρουμένων αυτών με όνομα "Sun", "Ground", "Background" και "MainCamera" διότι δημιουργούνται δυναμικά κατά τη διάρκεια της εκτέλεσης του εργαλείου δημιουργίας της σκηνής), δημιουργείται ένα Json αντικείμενο με όνομα το όνομα του GameObject. Έπειτα για κάθε τέτοιο Json αντικείμενο δημιουργούνται μερικά ακόμα αντικείμενα ως children σε αυτό το αντικείμενο, τα οποία υποδηλώνουν τις ιδιότητες του κάθε αντικειμένου. Τα αντικείμενα αυτά παρουσιάζονται αναλυτικά παρακάτω:

- “name”: Το όνομα του GameObject.
- “position”: Η θέση του GameObject.
- “rotation”: Η περιστροφή του GameObject.
- “scale”: Οι διαστάσεις του GameObject.
- “hasPhysics”: Η ιδιότητα αυτή παίρνει την τιμή true αν το GameObject έχει Rigidbody ή Rigidbody2D Component.
- “hasMainCamera”: Η ιδιότητα αυτή θα πάρει την τιμή true αν βρεθεί μέσα στο GameObject το Component Camera. Η αναζήτηση του Component αυτού γίνεται με χρήση της εντολής “`GetComponentInChildren<Camera>()`”.
- ”cameraType”: Η ιδιότητα αυτή θα πάρει τιμή ‘thirdPerson” άμα μέσα στο GameObject βρεθεί το Component CameraCollision.
- “isControllable”: Η ιδιότητα αυτή θα πάρει τιμή true αν μέσα στο GameObject βρεθεί το Component Character3D, Character2D, PlayerController3D ή PlayerController2D.
- “health”: Η ιδιότητα αυτή θα πάρει τιμή ανάλογα με τη μεταβλητή maxHealth του Component Entity, αν αυτό βρεθεί μέσα στο GameObject.
- “colliderType”: Η ιδιότητα αυτή θα πάρει τιμή ανάλογα με το ίδιος του collider που υπάρχει πάνω στο GameObject, αν αυτό υπάρχει.
- “hasCravity”: Η ιδιότητα αυτή θα πάρει τιμή true αν η μεταβλητή hasGravity του Component Rigidbody είναι true ή αν η τιμή της μεταβλητής GravityScale του Component Rigidbody2D είναι μεγαλύτερη από το μηδέν. Όλα αυτά φυσικά αν το Component Rigidbody υπάρχει στο GameObject.

Από τη στιγμή που θα δημιουργηθεί το περιεχόμενο του Json αρχείου, η εντολή “JsonMapper.ToJson”, θα μετατρέψει το Json string που δημιουργήθηκε σε Json object και στη συνέχεια η εντολή File.WriteAllText θα δημιουργήσει ένα νέο αρχείο, με περιεχόμενά του το Json object και με κατάληξη “.json”, στο path που έχει αποθηκευτεί στην μεταβλητή sceneExportPath.

3.4.2 Η εξαγωγή των 3D GameObjects

Η εξαγωγή των 3D αντικειμένων υλοποιείται από τη μέθοδο ExportModelFiles3D, η οποία καλείται να εκτελέσει τα παρακάτω βήματα.

3.4.2.1 Εξαγωγή των GameObjects ως .fbx αρχεία

Για την εξαγωγή των GameObjects ως .fbx αρχεία, γίνεται χρήση της βιβλιοθήκης UnityFBXExporter. Η μέθοδος ExportGameObjToFBX της κλάσης FBXExporter, παίρνει ως παραμέτρους το GameObject προς εξαγωγή, το Path εξαγωγής, και δύο bool τιμές οι οποίες είναι ορισμένες ως true και ορίζουν το αν τα materials και τα textures θα εξαχθούν μαζί με το GameObject.

3.4.2.2 Εξαγωγή των animations του κάθε GameObject ως .anim αρχεία

Η εξαγωγή των animations είναι μια σχετικά απλή διαδικασία. Αποτελείτε από έναν βρόγχο ο οποίος για κάθε animationClip στη μεταβλητή runtimeAnimatorController του Component Animator του GameObject, αφού λάβει το path του animation clip το οποίο βρίσκετε κάπου στα αρχεία του project με τη χρήση της μεθόδου GetAssetPath της κλάσης AssetDatabase, αντιγράφει το αρχείο του

animation στο δοθέν path καλώντας την εντολή File.Copy με overload την bool μεταβλητή true ώστε να γίνει override αν τυχόν το αρχείο υπάρχει ήδη στην ίδια τοποθεσία.

3.4.3 Η εξαγωγή των 2D GameObjects

Η εξαγωγή των 2D αντικειμένων διαφέρει λίγο από αυτή των 3D διότι αυτή τη φορά δεν έχουμε .fbx και .anim αρχεία αλλά απλά αρχεία εικόνων. Προκειμένου λοιπόν να εξαχθούν τα 2D αντικείμενα καλείτε η μέθοδος ExportModelFiles2D, η οποία εκτελεί τα παρακάτω βήματα.

3.4.3.1 Εξαγωγή των GameObjects ως αρχεία εικόνας

Ένα 2D GameObject δεν είναι τίποτα άλλο παρά ένα κενό GameObject με Component SpriteRenderer. Το Component αυτό είναι που δίνει υλική υπόσταση στο GameObject και ποιο συγκεκριμένα η μεταβλητή Sprite αυτού. Στη μεταβλητή sprite αποθηκεύεται το αρχείο εικόνας ως sprite. Το μόνο που χρειάζεται να γίνει για την εξαγωγή της εικόνας αυτής είναι η χρήση της μεθόδου GetAssetPath της κλάσης AssetDatabase με παράμετρο το sprite αυτό, επιστρέφοντας το path της εικόνας στα αρχεία του project. Από τη στιγμή που έχει βρεθεί το path γίνεται αντιγραφή της εικόνας στο δοθέν path με χρήση, για ακόμα μια φορά, της μεθόδου Copy.

3.4.3.2 Εξαγωγή των animations του κάθε GameObject ως αρχεία εικόνων

Ένα 2D animation δεν είναι τίποτα άλλο παρά μια σειρά από εικόνες που εναλλάσσονται γρήγορα. Προκειμένου να εξαχθεί το animation ενός 2D GameObject θα δημιουργηθεί ένας φάκελος για κάθε είδος animation (δηλαδή “Idle”, “Walking”,”Running”,”Jumping”), ο οποίος θα έχει και το αντίστοιχο όνομα του animation. Έπειτα γίνεται χρήση ενός βρόγχου, ο οποίος θα εκτελεστεί για κάθε animationClip των animationClips του runtimeAnimatorController που βρίσκεται πάνω στο εκάστοτε GameObject και θα επιστρέψει όλα τα animation clips του αντικειμένου. Από τη στιγμή που έχουν παρθεί τα animation clips, θα πρέπει τώρα να εξαχθούν από αυτά τα sprites που το αποτελούν. Την λειτουργεία αυτή αναλαμβάνει η μέθοδος GetSpritesFromClip όπως υποδεικνύει και το όνομά της. Προκειμένου να εξαχθούν τα sprites θα εκτελεστεί ένας βρόγχος για κάθε keyFrame του animationClip και να αποθηκευτεί το εκάστοτε sprite σε έναν πίνακα sprites. Ο κώδικας της μεθόδου αυτής φαίνεται ποιο αναλυτικά παρακάτω.

```
1. public static List<Sprite> GetSpritesFromClip(AnimationClip
   animationClip)
2.     {
3.         List<Sprite> sprites = new List<Sprite>();
4.         if (animationClip != null)
5.             foreach (EditorCurveBinding binding in
   AnimationUtility.GetObjectReferenceCurveBindings(animationClip))
6.             {
7.                 ObjectReferenceKeyframe[] keyframes =
   AnimationUtility.GetObjectReferenceCurve(animationClip, binding);
8.                 foreach (ObjectReferenceKeyframe frame in keyframes)
9.                     sprites.Add((Sprite)frame.value);
10.            }
11.
12.        return sprites;
13.    }
```

Εικόνα 30: Ο κώδικας της μεθόδου GetSpritesFromClip, η οποία επιστρέφει τα sprites που αποτελούν ένα animationClip

Αφού έχουν εξαχθεί επιτυχώς τα sprites του animationClip θα γίνει χρήση ενός βρόγχου ο οποίος για κάθε sprite, θα αντιγράψει τα sprites ως εικόνες από τα αρχεία του project με χρήση και πάλι των μεθόδων GetAssetPath και Copy.

Κεφάλαιο 4ο: Παράδειγμα Δυναμικής Παραγωγής 3D Παιχνιδιού

Στο κεφάλαιο αυτό παρουσιάζεται μέσω παραδειγμάτων ο τρόπος χρήσης των δύο εργαλείων UnityPackageGenerator και JsonSceneGenerator, καθώς και το πως συνεργάζονται μεταξύ τους ώστε να επιτευχθεί η δυναμική παραγωγή ενός 3D παιχνιδιού. Τα δύο αυτά εργαλεία θα παρουσιαστούν ξεχωριστά διότι λειτουργούν σε διαφορετικές πλατφόρμες, καθώς το πρώτο έχει δημιουργηθεί σε C# με τη χρήση του .NET framework, ενώ το δεύτερο σε C# με τη χρήση του Unity3D game engine.

4.1 Παράδειγμα δημιουργίας 3D παιχνιδιού

4.1.1 Παράδειγμα χρήσης του εργαλείου UnityPackageGenerator

Το UnityPackageGenerator είναι το εργαλείο υπεύθυνο για την δημιουργία του Unity3D package που αργότερα θα χρησιμοποιηθεί Unity για την δυναμική παραγωγή σκηνής. Το εργαλείο αυτό συλλέγει πληροφορία από το χρήστη όπως τα αρχεία των αντικειμένων της σκηνής, δημιουργώντας παράλληλα και το αρχείο Json που περιέχει λεπτομέρειες σχετικά με τα αντικείμενα αυτά.

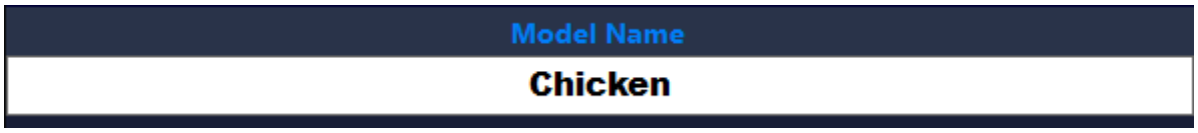
Η εκτέλεση του εργαλείου αυτού ξεκινάει με την εκτέλεση του .exe αρχείου του προγράμματος. Μετά την εκτέλεση του θα εμφανιστεί στην οθόνη ένα interface (βλέπε Εικόνα 1) το οποίο θα περιέχει δύο κατηγορίες η κάθε μια από της οποίες περιέχει τα δικά της properties. Η πρώτη κατηγορία θα έχει πάντα το όνομα ProjectProperties και θα περιλαμβάνει properties τα οποία σχετίζονται με ολόκληρο project και όχι με κάθε αντικείμενο της σκηνής ξεχωριστά (π.χ. Το dimension της σκηνής). Οι υπόλοιπες κατηγορίες αναπαριστούν η κάθε μια ένα αντικείμενο και περιλαμβάνουν properties που σχετίζονται με το αντικείμενο της σκηνής αυτό.

4.1.1.1 Τα Properties

Κατά την εκτέλεση της εφαρμογής λοιπόν, το πρώτο πράγμα που καλείται να επιλέξει ο χρήστης, από το property Dimension, την διάσταση του project: 3D ή 2D. Στο παράδειγμα αυτό επιλέχθηκε η επιλογή 3D. Η επιλογή του αυτή θα εξαφανίσει κάποια properties και θα εμφανίσει κάποια άλλα ανάλογα με το αν σχετίζονται με τη διάσταση αυτή.

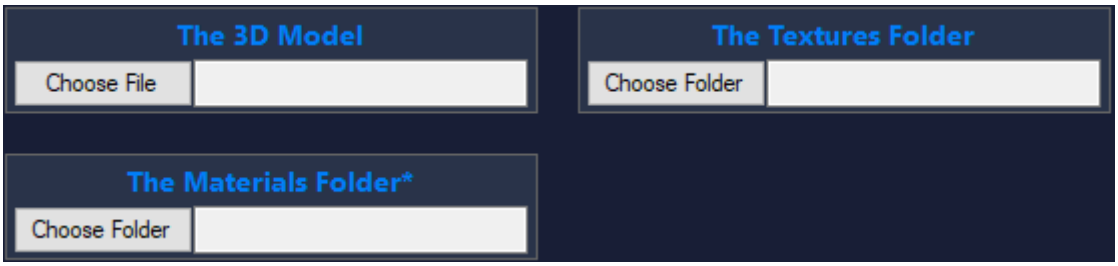
Εικόνα 31: Το property “Dimension”, υπεύθυνο για την επιλογή της διάστασης της σκηνής

Προχωρώντας στις παρακάτω κατηγορίες που σχετίζονται με τα αντικείμενα της σκηνής έχοντας επιλέξει ως Dimension την επιλογή 3D, το πρώτο property που θα συναντήσει ο χρήστης είναι το property “Model Name”, το οποίο είναι ένα property τύπου text box, η τιμή του οποίου, όπως υποδεικνύει και το όνομά του, θα αποτελέσει το όνομα του αντικειμένου. Για το συγκεκριμένο παράδειγμα ορίζεται το όνομα του μοντέλου ως “Chicken”



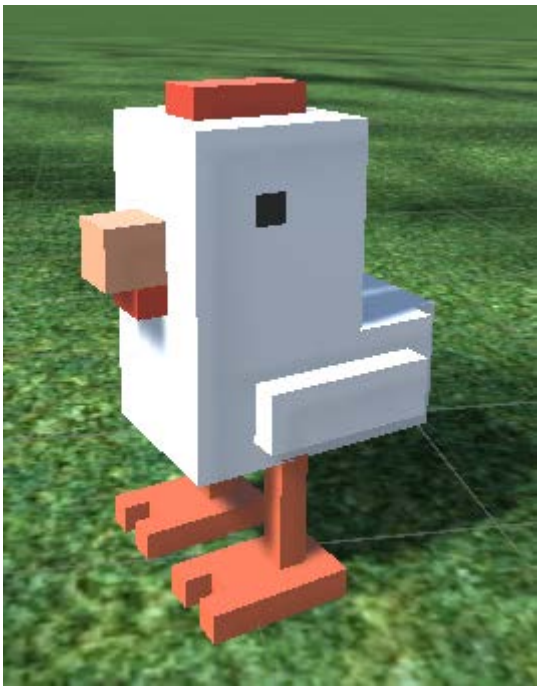
Εικόνα 32: Το property "Model Name"

Τα τρία επόμενα properties που θα συναντήσει ο χρήστης είναι τα properties “The 3D Model”, “The Textures Folder” και “The Materials Folder”. Αυτά τα properties τύπου select, ζητάν από τον χρήστη να επιλέξει αντίστοιχα το .fbx μοντέλο [29] του αντικειμένου, τον φάκελο που περιέχει τα textures του και τον φάκελο που περιέχει τα materials του αντικειμένου, από ένα μενού επιλογής αρχείων ή φακέλων. Η επιλογή του φακέλου των materials είναι προαιρετική, πράγμα που υποδεικνύει ο αστερίσκος δεξιά από το όνομα του property.



Εικόνα 33: Τα properties που σχετίζονται με την επιλογή του path των αρχείων του μοντέλου, των textures και των materials του μοντέλου”

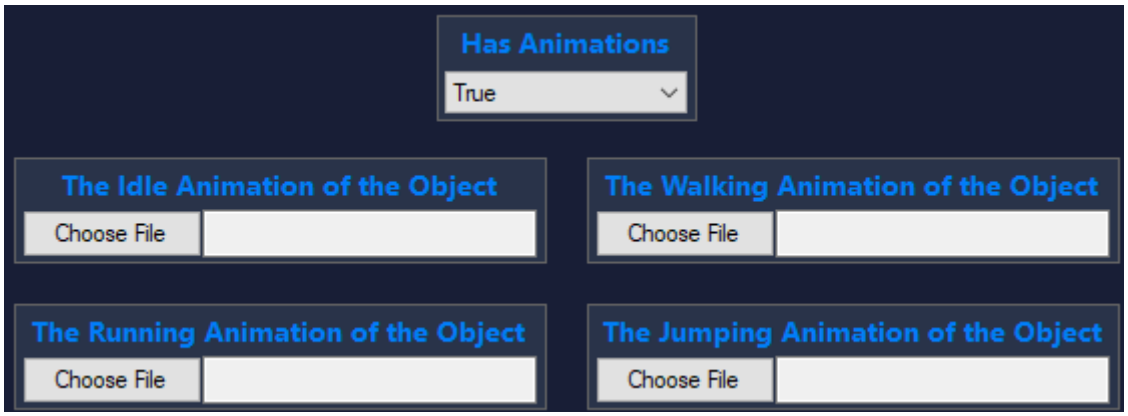
Για το παράδειγμα αυτό επιλέγεται από τον υπολογιστή το path του μοντέλου “Chicken.fbx”, το οποίο φαίνεται στην Εικόνα 35, μαζί με ta paths των φακέλων που περιέχουν τα αντίστοιχα materials και textures του.



Εικόνα 34: Το μοντέλο "Chicken.fbx" του παραδείγματος

Το επόμενο στη σειρά property ονομάζεται “Has Animations” και πρόκειται για ένα property τύπου combo box το οποίο περιέχει δύο τιμές: true και false. Αν ο χρήστης επιλέξει την τιμή true, τέσσερα ακόμα select properties θα εμφανιστούν τα οποία έχουν να κάνουν με την επιλογή των αρχείων animation του μοντέλου. Τα τέσσερα αυτά properties είναι τα εξής:

- “The Idle Animation of the object”: Όπου ο χρήστης καλείται να επιλέξει το path για το .fbx αρχείο που αποτελεί το Idle animation της σκηνής.
- “The Walking Animation of the object”: Όπου ο χρήστης καλείται να επιλέξει το path για το .fbx αρχείο που αποτελεί το Walking animation της σκηνής.
- “The Running Animation of the object”: Όπου ο χρήστης καλείται να επιλέξει το path για το .fbx αρχείο που αποτελεί το Running animation της σκηνής, το οποίο μπορεί να είναι το ίδιο με το Walking animation.
- “The Jumping Animation of the object”: Όπου ο χρήστης καλείται να επιλέξει το path για το .fbx αρχείο που αποτελεί το Jumping animation της σκηνής.

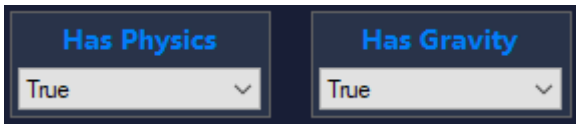


Εικόνα 35: Τα properties που σχετίζονται με την επιλογή του path των αρχείων των animation του μοντέλου.

Σειρά έχουν τα τρία properties που σχετίζονται με την επιλογή της θέσης της περιστροφής και του μεγέθους του αντικειμένου. Τα properties αυτά ονομάζονται αντίστοιχα “Position”, “Rotation” και “Scale”, και είναι properties τύπου Vector, διότι απαιτούν την εισαγωγή τριών αριθμητικών τιμών (“x”, “y” και “z”), που αποτελούν την θέση του αντικειμένου πάνω στους τρισδιάστατους άξονες της σκηνής του Unity3D.

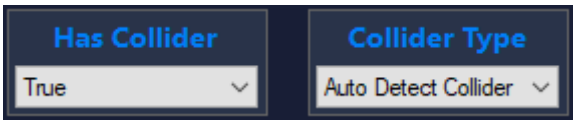
Εικόνα 36: Τα properties που έχουν να κάνουν με την εισαγωγή τριών αριθμητικών τιμών που αποτελούν την θέση του αντικειμένου πάνω στους τρισδιάστατους άξονες της σκηνής του Unity3D

Στη συνέχεια ο χρήστης μπορεί να επιλέξει αν επιθυμεί το αντικείμενο να έχει Physics και αν ναι, τότε αν επιθυμεί το αντικείμενο αυτό να επηρεάζεται από την δύναμη της βαρύτητας. Την επιλογή αυτή μπορεί να την κάνει μέσω των δύο properties “Has Physics” και “Has Gravity”, δύο properties τύπου combo box τα οποία περιέχουν δύο επιλογές true ή false. Το δεύτερο θα εμφανιστεί μόνο σε περίπτωση που η τιμή του πρώτου είναι true.



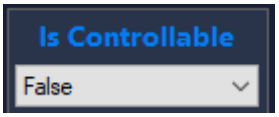
Εικόνα 37: Τα properties που σχετίζονται με τα physics του αντικειμένου

Συνεχίζοντας, ο χρήστης θα συναντήσει το property με το όνομα “Has Collider”, ένα combo box property το οποίο υποδεικνύει αν το αντικείμενο θα έχει colliders ή όχι. Αν πάρει την τιμή true, θα εμφανίσει ένα άλλο property με το όνομα “Collider Type”, επίσης ένα combo box property με 5 επιλογές: "Auto Detect Collider", "Box Collider", "Capsule Collider", "Sphere Collider", "Terrain Collider". Οι επιλογές αυτές αντιστοιχούν στο είδος του collider που θέλει ο χρήστης να έχει το αντικείμενο.



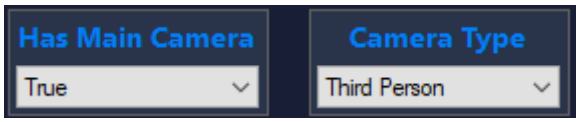
Εικόνα 38: Τα properties που σχετίζονται με τα colliders του αντικειμένου

Έπειτα, το property “Is Controllable”, υποδεικνύει το αν ο παίκτης θα έχει ή όχι τον έλεγχο της κίνησης του αντικειμένου αυτού. Το property αυτό είναι μοναδικό στο σύνολο των αντικειμένων μιας σκηνής, πράγμα που σημαίνει ότι αν πάρει την τιμή true κανένα άλλο αντικείμενο δεν θα μπορεί να πάρει να πάρει την τιμή true στο συγκεκριμένο του property.



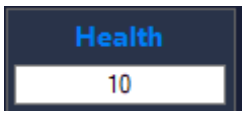
Εικόνα 39: Το property "Is Controllable"

Στη συνέχεια εμφανίζεται το combo box property “Has Main Camera”, το οποίο αν πάρει την τιμή true υποδεικνύει ότι το αντικείμενο θα έχει πάνω του την κάμερα του παιχνιδιού. Το αντικείμενο αυτό είναι επίσης μοναδικό ανάμεσα στα αντικείμενα της σκηνής, θέτοντας τον κανόνα της μίας κάμερας ανά σκηνή. Αν λοιπόν το property αυτό πάρει την τιμή true, ένα ακόμα property θα εμφανιστεί με το όνομα “Camera Type”, το οποίο υποδεικνύει το είδος της κάμερας. Υπάρχουν δύο επιλογές κάμερας που αντιστοιχούν και στις δύο επιλογές του combo box property αυτού: κάμερα πρώτου προσώπου (first person camera) και κάμερα τρίτου προσώπου (third person camera). Για τους σκοπούς του παραδείγματος γίνεται η επιλογή κάμερας τρίτου προσώπου.



Εικόνα 40: Τα properties που σχετίζονται με τη κάμερα της σκηνής

Το property “Health”, είναι ένα text box property που μπορεί να πάρει μόνο αριθμητικές float τιμές. Το property αυτό, όπως φαίνεται και από το όνομά του, υποδεικνύει τη ζωή που θα έχει το αντικείμενο, η οποία μπορεί να μεταβληθεί αν δεχθεί ζημιά.



Εικόνα 41: Το property "Heath", που σχετίζεται με τη ζωή του αντικειμένου

4.1.1.2 Επιπλέον λειτουργίες

Στον χρήστη δίνεται, όπως προαναφέρθηκε, η δυνατότητα να προσθέσει και άλλα αντικείμενα στη σκηνή. Την λειτουργεία αυτή έχει το κουμπί “ADD ANOTHER OBJECT” το οποίο βρίσκεται στο τέλος της κάθε κατηγορίας, και προσθέτει άλλη μία κατηγορία/αντικείμενο στη διεπαφή. Η κάθε κατηγορία μπορεί στη συνέχεια να διαγραφεί με το πάτημα του κόκκινου κουμπιού “X” που βρίσκετε στη πάνω δεξιά γωνία της κάθε κατηγορίας.

Τέλος προκειμένου να ξεκινήσει η διαδικασία δημιουργίας του Unity package, ο χρήστης πρέπει να πατήσει το κουμπί “Create Your Unity Package” που βρίσκετε πάντα στην κορυφή του interface. Αν ο χρήστης δεν έχει αφήσει κάποιο υποχρεωτικό property κενό σε οποιαδήποτε κατηγορία, με το πάτημα αυτού του κουμπιού ο χρήστης θα κληθεί να επιλέξει την τοποθεσία στην οποία επιθυμεί να εξάγει το Unity package, με τη χρήση ενός μενού επιλογής φακέλου, και αφού επιλέξει έναν έγκυρο φάκελο θα δημιουργηθεί ένα συμπιεσμένο αρχείο .zip με όνομα JsonSceneGenerator, το οποίο θα περιέχει το Unity package αυτό.

4.1.2 Παράδειγμα χρήσης του εργαλείου JsonSceneGenerator

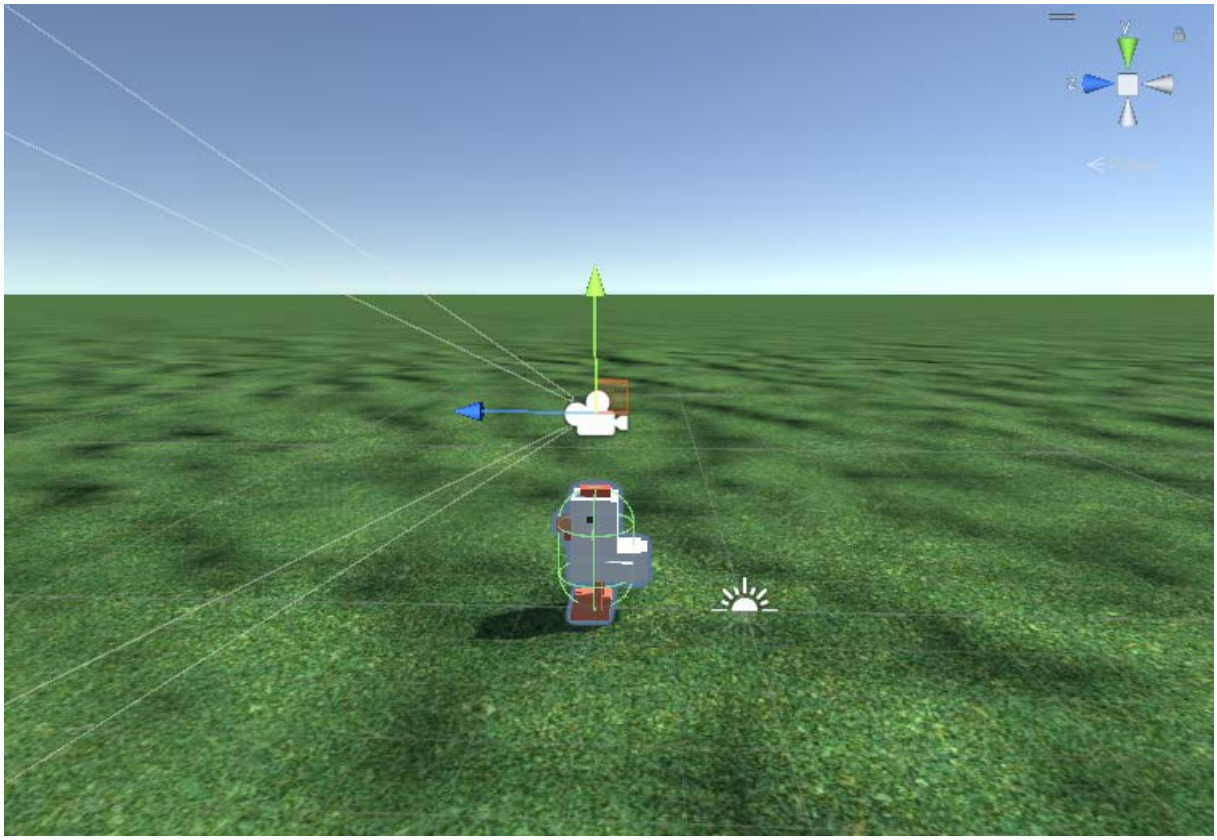
Το JsonSceneGenerator είναι το εργαλείο υπεύθυνο για την δυναμική δημιουργία της σκηνής στο Unity3D game engine. Προκειμένου να δημιουργηθεί η σκηνή, το εργαλείο χρησιμοποιεί τα αρχεία των αντικειμένων της σκηνής όπως μοντέλα και textures και ένα Json αρχείο που περιέχει πληροφορία σχετικά με το κάθε αντικείμενο της σκηνής.

Το εργαλείο αυτό έχει δημιουργηθεί ως Unity package πράγμα που σημαίνει ότι ο χρήστης θα πρέπει πρώτα να το εισάγει στο Unity3D. Ο ευκολότερος τρόπος να το κάνει αυτό, είναι μέσω του Package Manager του Unity3D. Ανοίγοντας λοιπόν το Package Manager ο χρήστης θα πρέπει να πατήσει το κουμπί πάνω αριστερά με το σύμβολο “+”, και στη συνέχεια να επιλέξει “Add package from disk”. Η επιλογή αυτή θα εμφανίσει ένα μενού επιλογής αρχείου, όπου ο χρήστης θα κληθεί να επιλέξει το .json αρχείο που συνήθως έχει το όνομα “package.json”. Από τη στιγμή που θα ολοκληρώσει τα παραπάνω βήματα και εισάγει το πακέτο στο Unity3D, θα εμφανιστεί άλλη μία επιπλέον επιλογή στο μενού του game engine με το όνομα “Json Scene Generator”, ανάμεσα στην επιλογή Component και

Window. Η επιλογή αυτή εμπεριέχει άλλες δύο επιλογές, με το όνομα “Export Scene” και “Import Scene” (Βλέπε Εικόνα 8). Με το πάτημα της πρώτης επιλογής ο χρήστης θα κληθεί, μέσο ενός μενού επιλογής φακέλου, να επιλέξει την τοποθεσία στην οποία θέλει να εξαχθεί η τρέχον σκηνή που έχει δημιουργήσει. Με το πάτημα της δεύτερης επιλογής θα δημιουργηθεί δυναμικά μια νέα σκηνή, με βάση τα αρχεία της σκηνής που έχει προ παρέχει ο χρήστης, συμπεριλαμβανομένου και του Json αρχείου.

4.1.3 Αποτέλεσμα

Αν το αρχείο Json και τα αρχεία του χρήστη έχουν δοθεί στη σωστή μορφή, μετά το πάτημα της επιλογής “Json Scene Generator/Import Scene” από το μενού επιλογών, η σκηνή θα πρέπει να δείχνει όπως στην παρακάτω εικόνα.



Εικόνα 42: Παράδειγμα τελικής δημιουργίας 3D σκηνής με ένα αντικείμενο

4.2 Παράδειγμα δημιουργίας 2D παιχνιδιού

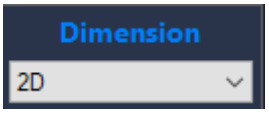
4.2.1 Παράδειγμα χρήσης του εργαλείου UnityPackageGenerator

Το UnityPackageGenerator είναι το εργαλείο υπεύθυνο για την δημιουργία του Unity3D package που αργότερα θα χρησιμοποιηθεί Unity για την δυναμική παραγωγή σκηνής. Το εργαλείο αυτό συλλέγει πληροφορία από το χρήστη όπως τα αρχεία των αντικειμένων της σκηνής, δημιουργώντας παράλληλα και το αρχείο Json που περιέχει λεπτομέρειες σχετικά με τα αντικείμενα αυτά.

Η εκτέλεση του εργαλείου αυτού ξεκινάει με την εκτέλεση του .exe αρχείου του προγράμματος. Μετά την εκτέλεση του θα εμφανιστεί στην οθόνη ένα interface (βλέπε Εικόνα 1) το οποίο θα περιέχει δύο κατηγορίες η κάθε μια από της οποίες περιέχει τα δικά της properties. Η πρώτη κατηγορία θα έχει πάντα το όνομα ProjectProperties και θα περιλαμβάνει properties τα οποία σχετίζονται με ολόκληρο project και όχι με κάθε αντικείμενο της σκηνής ξεχωριστά (π.χ. Το dimension της σκηνής). Οι υπόλοιπες κατηγορίες αναπαριστούν η κάθε μια ένα αντικείμενο και περιλαμβάνουν properties που σχετίζονται με το αντικείμενο της σκηνής αυτό.

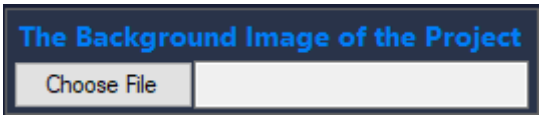
4.2.1.1 Τα Properties

Κατά την εκτέλεση της εφαρμογής λοιπόν, το πρώτο πράγμα που καλείται να επιλέξει ο χρήστης, από το property Dimension, την διάσταση του project: 3D ή 2D. Για τους σκεπούς του παραδείγματος αυτού επιλέχθηκε η επιλογή 2D. Η επιλογή του αυτή θα εξαφανίσει κάποια properties και θα εμφανίσει κάποια άλλα ανάλογα με το αν σχετίζονται με τη διάσταση αυτή.



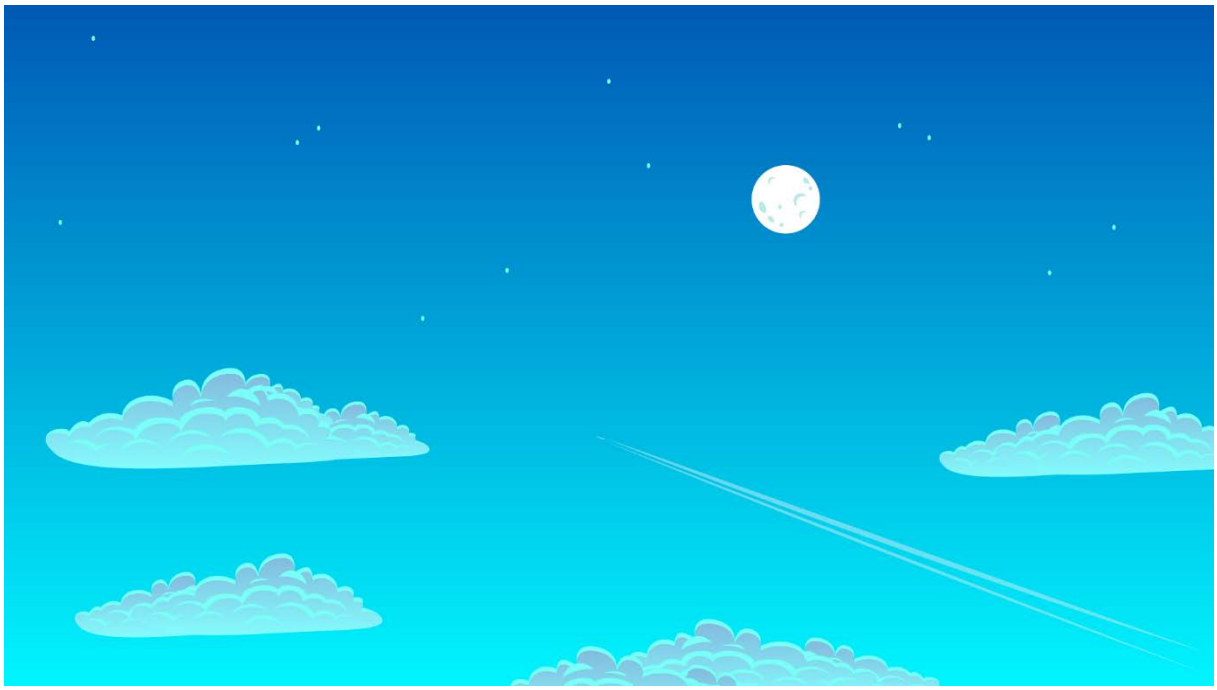
Εικόνα 43: Το property “Dimension”, υπεύθυνο για την επιλογή της διάστασης της σκηνής

Η επιλογή του αυτή θα εξαφανίσει κάποια properties και θα εμφανίσει κάποια άλλα ανάλογα με το αν σχετίζονται με τη διάσταση αυτή. Για παράδειγμα εφόσον ο χρήστης επιλέξει την επιλογή 2D, θα εμφανιστεί επίσης και το property Background, ένα property τύπου select, όπου ο χρήστης καλείται να εισάγει το path στον υπολογιστή του, για το αρχείο εικόνας που θα αποτελέσει το background της σκηνής.



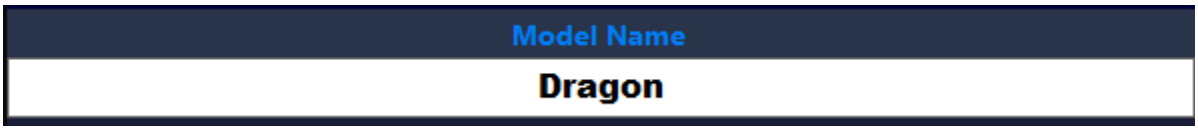
Εικόνα 44: Το property “Background”

Για το συγκεκριμένο παράδειγμα έγινε επιλογή της Εικόνα 47 ως background εικόνας.



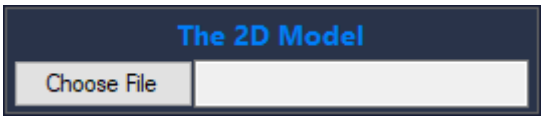
Εικόνα 45: Η background εικόνα που χρησιμοποιήθηκε στο παράδειγμα δημιουργίας 2D σκηνής

Προχωρώντας στις παρακάτω κατηγορίες που σχετίζονται με τα αντικείμενα της σκηνής έχοντας επιλέξει ως Dimension την επιλογή 2D, το πρώτο property που θα συναντήσει ο χρήστης είναι το property “Model Name”, το οποίο είναι ένα property τύπου text box, η τιμή του οποίου, όπως υποδεικνύει και το όνομά του, θα αποτελέσει το όνομα του αντικειμένου. Για τους σκοπούς του παραδείγματος αυτού, ορίζεται το όνομα του μοντέλου ως “Dragon”



Εικόνα 46: Το property "Model Name"

Το πρώτο property που θα συναντήσει ο χρήστης είναι το property “The 2D Model”, ένα select property το οποίο θα ζητήσει από τον χρήστη ένα αρχείο εικόνας.



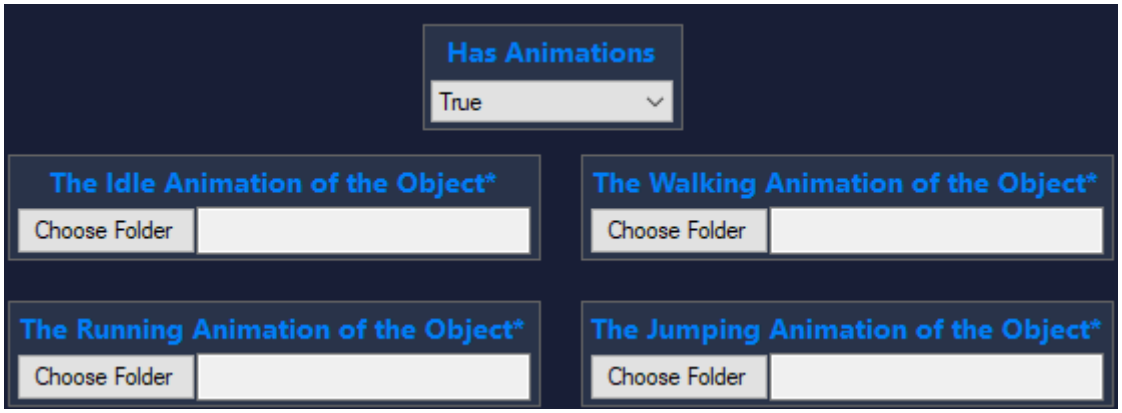
Εικόνα 47:Τα properties που σχετίζονται με την επιλογή του path του 2D μοντέλου”

Το 2D μοντέλο που θα χρησιμοποιηθεί για το συγκεκριμένο παράδειγμα είναι ένα αρχείο εικόνας με το όνομα “Dragon.png” και απεικονίζεται παρακάτω.

Εικόνα 48: Η εικόνα "Dragon.png" που θα χρησιμοποιηθεί στο παράδειγμα ως αντικείμενο

Το επόμενο στη σειρά property ονομάζεται “Has Animations” και πρόκειται για ένα property τύπου combo box το οποίο περιέχει δύο τιμές: true και false. Αν ο χρήστης επιλέξει την τιμή true, τέσσερα ακόμα select properties θα εμφανιστούν τα οποία έχουν να κάνουν με την επιλογή των αρχείων animation του μοντέλου. Τα τέσσερα αυτά properties είναι τα εξής:

- “The Idle Animation of the object”: Όπου ο χρήστης καλείται να επιλέξει το path για τον φάκελο που θα περιέχει τις εικόνες που αποτελούν το Idle animation της σκηνής.
- “The Walking Animation of the object”: Όπου ο χρήστης καλείται να επιλέξει το path για τον φάκελο που θα περιέχει τις εικόνες που αποτελούν το Walking animation της σκηνής.
- “The Running Animation of the object”: Όπου ο χρήστης καλείται να επιλέξει το path για τον φάκελο που θα περιέχει τις εικόνες που αποτελούν το Running animation της σκηνής, το οποίο μπορεί να είναι το ίδιο με το Walking animation.
- “The Jumping Animation of the object”: Όπου ο χρήστης καλείται να επιλέξει το path για τον φάκελο που θα περιέχει τις εικόνες που αποτελούν το Jumping animation της σκηνής.

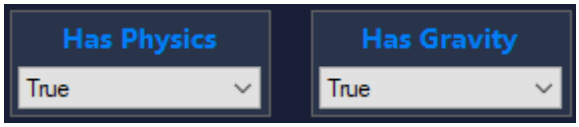


Εικόνα 49: Τα properties που σχετίζονται με την επιλογή του path των αρχείων των animation του μοντέλου.

Σειρά έχουν τα τρία properties που σχετίζονται με την επιλογή της θέσης της περιστροφής και του μεγέθους του αντικειμένου. Τα properties αυτά ονομάζονται αντίστοιχα “Position”, “Rotation” και “Scale”, και είναι properties τύπου Vector, διότι απαιτούν την εισαγωγή τριών αριθμητικών τιμών (“x”, “y” και “z”), που αποτελούν την θέση του αντικειμένου πάνω στους τρισδιάστατους άξονες της σκηνής του Unity3D. Παρόλο που το παράδειγμα αναφέρεται σε 2D σκηνή, η τρίτη διάσταση ορίζει αν το αντικείμενο θα εμφανίζεται μπροστά ή πίσω από τα άλλα.

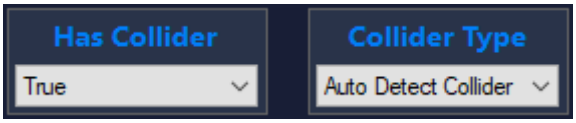
Εικόνα 50: Τα properties που έχουν να κάνουν με την εισαγωγή τριών αριθμητικών τιμών που αποτελούν την θέση του αντικειμένου πάνω στους τρισδιάστατους άξονες της σκηνής του Unity3D

Στη συνέχεια ο χρήστης μπορεί να επιλέξει αν επιθυμεί το αντικείμενο να έχει Physics και αν ναι, τότε αν επιθυμεί το αντικείμενο αυτό να επηρεάζεται από την δύναμη της βαρύτητας. Την επιλογή αυτή μπορεί να την κάνει μέσω των δύο properties “Has Physics” και “Has Gravity”, δύο properties τύπου combo box τα οποία περιέχουν δύο επιλογές true ή false. Το δεύτερο θα εμφανιστεί μόνο σε περίπτωση που η τιμή του πρώτου είναι true.



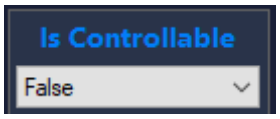
Εικόνα 51: Τα properties που σχετίζονται με τα physics του αντικειμένου

Συνεχίζοντας, ο χρήστης θα συναντήσει το property με το όνομα “Has Collider”, ένα combo box property το οποίο υποδεικνύει αν το αντικείμενο θα έχει colliders ή όχι. Αν πάρει την τιμή true, θα εμφανίσει ένα άλλο property με το όνομα “Collider Type”, επίσης ένα combo box property με 5 επιλογές: "Auto Detect Collider 2D", "Box Collider 2D", "Capsule Collider 2D", "Circle Collider 2D", "Composite Collider 2D" , “Edge Collider 2D”. Οι επιλογές αυτές αντιστοιχούν στο είδος του collider που θέλει ο χρήστης να έχει το αντικείμενο.



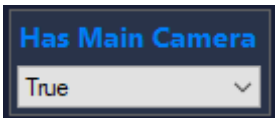
Εικόνα 52: Τα properties που σχετίζονται με τα colliders του αντικειμένου

Έπειτα, το property “Is Controllable”, υποδεικνύει το αν ο παίκτης θα έχει ή όχι τον έλεγχο της κίνησης του αντικειμένου αυτού. Το property αυτό είναι μοναδικό στο σύνολο των αντικειμένων μιας σκηνής, πράγμα που σημαίνει ότι αν πάρει την τιμή true κανένα άλλο αντικείμενο δεν θα μπορεί να πάρει να πάρει την τιμή true στο συγκεκριμένο του property.



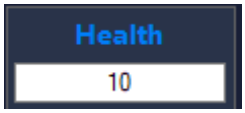
Εικόνα 53: Το property "Is Controllable"

Στη συνέχεια εμφανίζεται το combo box property “Has Main Camera”, το οποίο αν πάρει την τιμή true υποδεικνύει ότι το αντικείμενο θα έχει πάνω του την κάμερα του παιχνιδιού. Το αντικείμενο αυτό είναι επίσης μοναδικό ανάμεσα στα αντικείμενα της σκηνής, θέτοντας τον κανόνα της μίας κάμερας ανά σκηνή. Αυτό σημαίνει ότι η κάμερα θα ακολουθεί τον χαρακτήρα του παίκτη κρατώντας μία σταθερή απόσταση.



Εικόνα 54: Το property που σχετίζεται με τη κάμερα της σκηνής

Το property “Health”, είναι ένα text box property που μπορεί να πάρει μόνο αριθμητικές float τιμές. Το property αυτό, όπως φαίνεται και από το όνομά του, υποδεικνύει τη ζωή που θα έχει το αντικείμενο, η οποία μπορεί να μεταβληθεί αν δεχθεί ζημιά.



Εικόνα 55: Το property "Heath", που σχετίζεται με τη ζωή του αντικειμένου

4.2.1.2 Επιπλέον λειτουργίες

Στον χρήστη δίνεται, όπως προαναφέρθηκε, η δυνατότητα να προσθέσει και άλλα αντικείμενα στη σκηνή. Την λειτουργεία αυτή έχει το κουμπί “ADD ANOTHER OBJECT” το οποίο βρίσκεται στο τέλος της κάθε κατηγορίας, και προσθέτει άλλη μία κατηγορία/αντικείμενο στη διεπαφή. Η κάθε κατηγορία μπορεί στη συνέχεια να διαγραφεί με το πάτημα του κόκκινου κουμπιού “X” που βρίσκετε στη πάνω δεξιά γωνία της κάθε κατηγορίας.

Τέλος προκειμένου να ξεκινήσει η διαδικασία δημιουργίας του Unity package, ο χρήστης πρέπει να πατήσει το κουμπί “Create Your Unity Package” που βρίσκετε πάντα στην κορυφή του interface. Αν ο χρήστης δεν έχει αφήσει κάποιο υποχρεωτικό property κενό σε οποιαδήποτε κατηγορία, με το πάτημα αυτού του κουμπιού ο χρήστης θα κληθεί να επιλέξει την τοποθεσία στην οποία επιθυμεί να εξάγει το Unity package, με τη χρήση ενός μενού επιλογής φακέλου, και αφού επιλέξει έναν έγκυρο φάκελο θα δημιουργηθεί ένα συμπιεσμένο αρχείο .zip με όνομα JsonSceneGenerator, το οποίο θα περιέχει το Unity package αυτό.

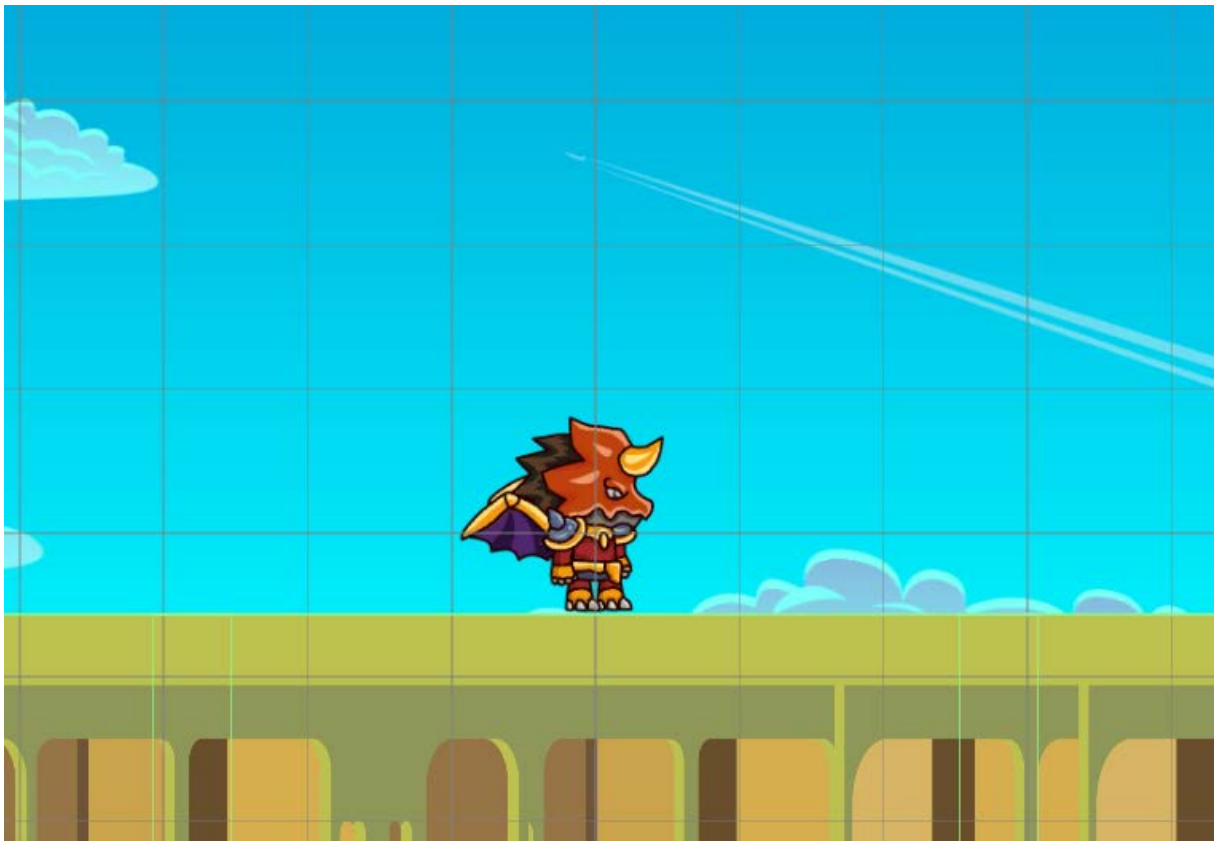
4.2.2 Παράδειγμα χρήσης του εργαλείου JsonSceneGenerator

Το JsonSceneGenerator είναι το εργαλείο υπεύθυνο για την δυναμική δημιουργία της σκηνής στο Unity3D game engine. Προκειμένου να δημιουργηθεί η σκηνή, το εργαλείο χρησιμοποιεί τα αρχεία των αντικειμένων της σκηνής όπως μοντέλα και textures και ένα Json αρχείο που περιέχει πληροφορία σχετικά με το κάθε αντικείμενο της σκηνής.

Το εργαλείο αυτό έχει δημιουργηθεί ως Unity package πράγμα που σημαίνει ότι ο χρήστης θα πρέπει πρώτα να το εισάγει στο Unity3D. Ο ευκολότερος τρόπος να το κάνει αυτό, είναι μέσω του Package Manager του Unity3D. Ανοίγοντας λοιπόν το Package Manager ο χρήστης θα πρέπει να πατήσει το κουμπί πάνω αριστερά με το σύμβολο "+", και στη συνέχεια να επιλέξει "Add package from disk". Η επιλογή αυτή θα εμφανίσει ένα μενού επιλογής αρχείου, όπου ο χρήστης θα κληθεί να επιλέξει το .json αρχείο που συνήθως έχει το όνομα "package.json". Από τη στιγμή που θα ολοκληρώσει τα παραπάνω βήματα και εισάγει το πακέτο στο Unity3D, θα εμφανιστεί άλλη μία επιπλέον επιλογή στο μενού του game engine με το όνομα "Json Scene Generator", ανάμεσα στην επιλογή Component και Window. Η επιλογή αυτή εμπεριέχει άλλες δύο επιλογές, με το όνομα "Export Scene" και "Import Scene" (Βλέπε Εικόνα 8). Με το πάτημα της πρώτης επιλογής ο χρήστης θα κληθεί, μέσο ενός μενού επιλογής φακέλου, να επιλέξει την τοποθεσία στην οποία θέλει να εξαχθεί η τρέχον σκηνή που έχει δημιουργήσει. Με το πάτημα της δεύτερης επιλογής θα δημιουργηθεί δυναμικά μια νέα σκηνή, με βάση τα αρχεία της σκηνής που έχει προ παρέχει ο χρήστης, συμπεριλαμβανομένου και του Json αρχείου.

4.2.3 Αποτέλεσμα

Αν το αρχείο Json και τα αρχεία του χρήστη έχουν δοθεί στη σωστή μορφή, μετά το πάτημα της επιλογής "Json Scene Generator/Import Scene" από το μενού επιλογών, η σκηνή θα πρέπει να δείχνει όπως στην παρακάτω εικόνα.



Εικόνα 56: Παράδειγμα τελικής δημιουργίας 2D σκηνής με ένα αντικείμενο

Κεφάλαιο 5ο: Συμπεράσματα ή/και Προτάσεις Βελτίωσης

5.1 Συμπεράσματα

Ένα μεγάλο πρόβλημα σε μία σύγχρονη εταιρία ανάπτυξης βιντεοπαιχνιδιών αποτελεί η ανάγκη συνεργασίας των διαφορετικών τμημάτων της εταιρίας η οποία δημιουργείται λόγο του αυξημένου όγκου και απαιτήσεων των χρηστών από τα σημερινά βιντεοπαιχνίδια. Ορισμένα προβλήματα εμφανίζονται κατά την προσπάθεια της συνεργασίας αυτής, όπως η ανάγκη για συνεννόηση και ανταλλαγή γνώσεων μεταξύ του προσωπικού των διαφόρων αυτών τμημάτων ή ακόμα χρήση γνώσεων ή εργαλείων ενός τμήματος από ένα άλλο. Η παρούσα πτυχιακή πρότεινε λύση σε μια πολύ συγκεκριμένη πτυχή αυτού του προβλήματος, τη συνεργασία του τμήματος δημιουργίας γραφικών και μοντελοποίησης με το τμήμα προγραμματισμού και πιο συγκεκριμένα την δοκιμή των διαφόρων μοντέλων ή γραφικών στοιχείων που έχει κατασκευάσει το τμήμα δημιουργίας γραφικών και μοντελοποίησης πάνω σε μια Unity3D σκηνή.

Αυτό επιτυγχάνεται με τη χρήση δύο εργαλείων που αναπτύχθηκαν στο πλαίσιο της πτυχιακής αυτής τα οποία επιτρέπουν έναν άπειρο προγραμματιστικά χρήστη να δημιουργήσει δυναμικά μια τέτοια Unity3D σκηνή. Με τη χρήση του πρώτου εργαλείου ονόματι “UnityPackageGenerator” ανεπτυγμένο σε C# με τη χρήση του .NET Framework, ο χρήστης καλείτε να περιγράψει τις ιδιότητες των αντικειμένων της σκηνής, συμπληρώνοντας τα απαραίτητα πεδία στο πρόγραμμα, καθώς και να παρέχει τα απαραίτητα μοντέλα και γραφικά στοιχεία που θα αποτελούν τα αντικείμενα της σκηνής. Με την επιτυχή ολοκλήρωση των βημάτων αυτών, θα δημιουργηθεί ένα Unity Package το οποίο θα περιέχει εσωτερικά τα αρχεία που παρέδωσε ο χρήστης καθώς και ένα αρχείο Json που θα περιέχει τις περιγραφές των αντικειμένων αυτών. Το Unity Package αυτό πρόκειται για το δεύτερο εργαλείο ονόματι “JsonSceneGenerator” με το οποίο, μετά την εισαγωγή του στο Unity3D ως package, ο χρήστης θα μπορεί πολύ εύκολα, με το πάτημα ενός κουμπιού, να δημιουργήσει την σκηνή που επιθυμεί μέσα στο Unity Editor.

Με τη χρήση αυτών των εργαλείων ένας οποιοσδήποτε χρήστης μπορεί πλέον να δημιουργήσει την σκηνή και κατ’ επέκταση το παιχνίδι που επιθυμεί χωρίς να είναι απαραίτητο να κατέχει προγραμματιστικές γνώσεις, αποσυμφορώντας το προσωπικό μιας εταιρίας κατασκευής βιντεοπαιχνιδιών. Πέρα από τις εταιρίες πληροφορικής το λογισμικό αυτό μπορεί να βρει ενδιαφέρον και στο ευρύτερο κοινό το οποίο μπορεί να έχει ως σκοπό την δημιουργία ενός προτύπου παιχνιδιού χωρίς πολύ κόπο και ιδιαίτερες γνώσεις πάνω στο Unity.

5.2 Προτάσεις βελτίωσης

Το λογισμικό που παρουσιάζεται στη πτυχιακή αυτή χρήζει αρκετές βελτιώσεις. Μέχρι στιγμής κατά τη δημιουργία της σκηνής υλοποιείται ένα σύνολο πολύ βασικών λειτουργιών ενός παιχνιδιού όπως η τοποθέτηση των αντικειμένων του αρχείου Json στην επιθυμητή θέση με την επιθυμητή περιστροφή και τις επιθυμητές διαστάσεις, εισάγοντας πάνω σε αυτά διάφορες ιδιότητες όπως colliders, physics, animations, textures, κ.α. Παρακάτω γίνεται αναφορά στα διάφορα τμήματα αυτού, παρουσιάζοντας πιθανούς τρόπους βελτίωσης της λειτουργικότητας τους και της εμπειρίας του χρήστη.

5.2.1 Βελτίωση της εισαγωγής των μοντέλων

Ο τρόπος με τον οποίο γίνεται η εισαγωγή και εξαγωγή των τρισδιάστατων μοντέλων είναι απλός. Ο χρήστης παρέχει ένα .fbx αρχείο το οποίο με την εκτέλεση του προγράμματος υλοποιείται μέσα στη σκηνή. Τα προβλήματα ξεκινούν με την εισαγωγή των textures και των materials, καθώς η τεχνολογία αναπαράστασης των γραφικών του Unity3D διαφέρει από αυτήν των λογισμικών μοντελοποίησης όπως Blender, Maya κ.λπ., με αποτέλεσμα το Unity να μη μπορεί αυτόματα να αντιστοιχίσει το μοντέλο με τα γραφικά του στοιχεία. Ο τρόπος με τον οποίο επιλύεται το παρών πρόβλημα στη παρούσα έκδοση του εργαλείου είναι η δυναμική αντιστοίχιση των textures των μοντέλων με τα μοντέλα με βάση το όνομά τους. Όπως γίνεται κατανοητό, προκειμένου να γίνει κάτι τέτοιο θα πρέπει ο χρήστης να έχει χρησιμοποιήσει σωστή ονοματοδοσία τόσο στα μοντέλα όσο και στα textures ή materials που θα παραδώσει, πράγμα που δεν είναι πάντα εφικτό διότι ο δημιουργός του μοντέλου θα πρέπει να το γνωρίζει εκ των προτέρων. Μια σημαντική λοιπόν βελτίωση της εισαγωγής των μοντέλων στη σκηνή θα ήταν η περεταίρω αυτοματοποίηση της τοποθέτησης των textures ή των materials πάνω σε αυτά.

Ένας ακόμα περιορισμός που υπάρχει, όσον αφορά τα μοντέλα, είναι το ότι το πρόγραμμα μπορεί να δεχθεί μόνο μοντέλα τύπου FBX. Σημαντική βελτίωση λοιπόν θα μπορούσε να είναι η επέκταση του εργαλείου με τέτοιο τρόπο ώστε να μπορεί να δεχθεί και άλλων ειδών μοντέλα όπως OBJ ή 3DS. Ο ίδιος περιορισμός συναντάται και στα δισδιάστατα αντικείμενα καθώς το δισδιάστατο μοντέλο (δηλαδή ένα αρχείο εικόνας) μπορεί να είναι είτε της μορφής JPG είτε PNG.

5.2.2 Βελτίωση του συστήματος των animations

Τα 3D animations είναι αρχεία τύπου ANIM τα οποία περιέχουν πληροφορία σχετικά με την κίνηση του μοντέλου. Στην παρούσα έκδοση του εργαλείου ζητείται από τον χρήστη να παραδώσει τεσσάρων ειδών animations, δηλαδή τέσσερα .anim αρχεία τα οποία θα αντιστοιχούν σε τέσσερις καταστάσεις Idle, Walking, Running και Jumping. Στα 2D animations η λογική είναι η ίδια αλλά αντί για .anim αρχεία ο χρήστης καλείται να παραδώσει ένα σύνολο από εικόνες που θα αποτελούν το κάθε animation. Όπως γίνεται κατανοητό οι κινήσεις του animation περιορίζονται στις τέσσερις αυτές καταστάσεις. Μία σημαντική βελτίωση θα ήταν λοιπόν η επέκταση του συστήματος των animation ώστε να δέχεται περισσότερων ειδών καταστάσεις όπως για παράδειγμα Crouching ή Dancing.

Μία άλλη σημαντική ιδιότητα που λείπει από τη συγκεκριμένη έκδοση του συστήματος εισαγωγής των animations είναι η δυνατότητα αλλαγής της ταχύτητας των animations. Στο παρόν σύστημα τα animations έχουν οριστεί να παίζουν σε μια συγκεκριμένη προκαθορισμένη ταχύτητα. Για τα τρισδιάστατα animations αυτό συνήθως δεν αποτελεί πρόβλημα καθώς η προκαθορισμένη τους ταχύτητα είναι και ταχύτητα στην οποία προορίζονταν να παιχτούν. Τα δισδιάστατα animations όμως τα οποία δημιουργούνται δυναμικά από εικόνες, δίνεται ένα προκαθορισμένο διάστημα του ενός εκατοστού του δευτερολέπτου μεταξύ των εικόνων, πράγμα που ορίζει την ταχύτητα του animation. Η ταχύτητα στην οποία προοριζόταν να παιχτεί το animation πιθανών να διαφέρει από την ταχύτητα που όρισε το εργαλείο με αποτέλεσμα να απαιτούνται επιπλέον ενέργειες από την πλευρά του χρήστη για να επανέλθει στην κανονική αυθεντική του ταχύτητα, πράγμα που εναντιώνεται στην ιδέα της πλήρης αυτοματοποίησης της δημιουργίας της σκηνής. Ως λύση σε αυτό το πρόβλημα προτείνεται η εισαγωγή μιας επιπλέων ιδιότητας την οποία θα συμπληρώνει ο χρήστης, πριν την δημιουργία της σκηνής, η οποία θα ορίζει την ταχύτητα του animation.

5.2.3 Βελτίωση της κίνησης του χαρακτήρα του παίκτη

Η κίνηση του αντικειμένου, το οποίο ο χρήστης ως controllable και αποτελεί τον χαρακτήρα του παίκτη, επιτυγχάνεται με την χρήση ενός script ονόματι PlayerController3D και PlayerController2D άμα πρόκειται για τρισδιάστατο ή δισδιάστατο αντικείμενο αντίστοιχα.

Υπάρχουν δύο τρόποι για να επιτευχθεί η κίνηση ενός τρισδιάστατου χαρακτήρα μέσα σε ένα παιχνίδι στο Unity3D. Ο πρώτος τρόπος είναι με τη χρήση physics, πράγμα που δίνει την δυνατότητα στο GameObject να αλληλοεπιδρά πλήρως με τα physics των άλλων αντικειμένων. Ο δεύτερος τρόπος είναι με την χρήστη του Component CharacterController, ένα Component με τη χρήση του οποίου προσομοιώνεται, προγραμματιστικά αυτή τη φορά, μέρος των physics του αντικειμένου.

Με τη χρήση του πρώτου τρόπου η κίνηση του αντικειμένου επιτυγχάνεται με μεταβολή της επιτάχυνσης του αντικειμένου προς μία συγκεκριμένη κατεύθυνση, μέσο του Rigidbody Component. Η υλοποίηση αυτή φαντάζει εκ πρώτης όψεως ικανοποιητική, ορισμένα δυνατότητες όμως λείπουν από την λειτουργικότητα αυτής οι οποίες πρέπει να υλοποιηθούν προγραμματιστικά ενώ με το CharacterController Component θα ήταν ήδη ενσωματωμένες. Μερικά παραδείγματα αυτής της λειτουργικότητας είναι η δυνατότητα του χαρακτήρα να ανεβαίνει αυτόματα σκάλες χωρίς να χρειάζεται να πηδάει η δυνατότητα του χαρακτήρα να ανεβαίνει πλαγιές παραμετροποιήσιμης κλήσης.

Η χρήση του δεύτερου τρόπου παρόλα αυτά δεν λύνει όλα τα προβλήματα καθώς μπορεί να προκαλέσει καινούρια. Η εισαγωγή του CharacterController Component σε ένα αντικείμενο συνεπάγεται της αναγκαστικής εισαγωγής ενός Capsule Collider σε αυτό αποτρέποντας τη χρήση οποιουδήποτε άλλου είδους collider. Ένα από τα σημαντικότερα μειονεκτήματα του CharacterController αποτελεί επίσης το γεγονός ότι δεν είναι δυνατή η σύγκρουση του αντικειμένου με αντικείμενα που κατέχουν concave (κοίλους) colliders. Η βελτιστοποίηση που προτείνεται στο συγκεκριμένο κομμάτι, μπορεί να είναι είτε η δυνατότητα του χρήστη να επιλέξει την ενσωμάτωση CharacterController αντί της κίνησης μέσω physics, είτε ο προγραμματισμός των επιπλέων δυνατοτήτων που παρέχει το CharacterController, στην κίνηση μέσω physics. Ως επιπλέων θα μπορούσε να οριστεί η δυνατότητα του χρήστη να ορίσει εκ των προτέρων την ταχύτητα του αντικειμένου που αποτελεί τον χαρακτήρα του παίκτη καθώς και τη μάζα του.

Μια λιγότερο σημαντική, θα έλεγε κανείς, δυνατότητα που λείπει από την κίνηση τόσο των τρισδιάστατων χαρακτήρων όσο και των δισδιάστατων, είναι η δυνατότητα ανίχνευσης οροφής. Η δυνατότητα αυτή έρχεται μαζί με την δυνατότητα crouching του χαρακτήρα του παίκτη, και πρόκειται για την δυνατότητα του χαρακτήρα να ανιχνεύει αν από πάνω του υπάρχει οροφή ώστε να μπορέσει να επανέλθει στην “όρθια” κατάστασή του.

5.2.4 Βελτιστοποίηση του εδάφους

Στην τρέχουσα έκδοση του εργαλείου, κατά τη δυναμική δημιουργία μιας τρισδιάστατης σκηνής, το έδαφος δημιουργείται προσθέτοντας στη σκηνή ένα primitive αντικείμενο τύπου Plane το οποίο κατέχει ένα collider ώστε να μπορούν τα διάφορα αντικείμενα να σταθούν πάνω του. Είναι φανερό ότι η συγκεκριμένη υλοποίηση είναι φτωχή καθώς το έδαφος θα είναι επίπεδο, λείο και μονόχρωμο. Μία λύση στο πρόβλημα αυτό θα ήταν να δίνεται στον χρήστη η δυνατότητα να ορίζει ως έδαφος ένα δικό του μοντέλο ενώ μία δεύτερη λύση θα ήταν η δυναμική δημιουργία εδάφους με χρήση κάποιου Random Height Map βασισμένο σε κάποιο Seed έτσι ώστε να δημιουργείται ένα ρεαλιστικό έδαφος. Και η δύο αυτές λύσεις όμως αποτελούν πρόκληση καθώς αλλάζοντας το σχήμα του εδάφους τα

αντικείμενα που θα τοποθετηθούν σε αυτό θα πρέπει να εφαρμόζουν σωστά ώστε να μη δημιουργούνται κενά ανάμεσα σε αυτά και στο έδαφος ή ακόμα το έδαφος να περνάει από μέσα τους σε περίπτωση που δεν έχουν δοθεί κατάλληλες συντεταγμένες.

Κεφάλαιο 6ο: Βιβλιογραφία

[1] G. S. A. K. Farouk Messaoudi, Dissecting games engines: The case of Unity3D, 2014.

[2] J. Haas, «A History of the Unity Game Engine,» [Ηλεκτρονικό]. Available: https://digital.wpi.edu/downloads/2f75r821k. [Πρόσβαση 2022].

[3] Unity3D, «Unity Forum,» [Ηλεκτρονικό]. Available: https://forum.unity.com/. [Πρόσβαση 2022].

[4] Pluralsight, «Elliminate Texture Confusion: Bump, Normal and Displacement Maps,» Pluralsight, 2014. [Ηλεκτρονικό]. Available: https://www.pluralsight.com/blog/film-games/bump-normal-and-displacement-maps. [Πρόσβαση 2022].

[5] Unity, «Reflective Bumped Specular,» 2022. [Ηλεκτρονικό]. Available: https://docs.unity3d.com/Manual/shader-ReflectiveBumpedSpecular.html. [Πρόσβαση 2022].

[6] learnopengl, «Parallax Mapping,» 2017. [Ηλεκτρονικό]. Available: https://learnopengl.com/Advanced-Lighting/Parallax-Mapping. [Πρόσβαση 2022].

[7] Unity3D, «Screen Space Ambient Occlusion,» 2017. [Ηλεκτρονικό]. Available: https://docs.unity3d.com/550/Documentation/Manual/script-ScreenSpaceAmbientOcclusion.html. [Πρόσβαση 2022].

[8] U. 3d, «Shadow mapping,» Unity 3D, 2021. [Ηλεκτρονικό]. Available: https://docs.unity3d.com/Manual/shadow-mapping.html. [Πρόσβαση 2022].

[9] OpenGL, «Tutorial 14 : Render To Texture,» OpenGL, [Ηλεκτρονικό]. Available: http://www.opengl-tutorial.org/intermediate-tutorials/tutorial-14-render-to-texture/. [Πρόσβαση 2022].

[10] Unity3D, «Unity Services,» 2022. [Ηλεκτρονικό]. Available: https://dashboard.unity3d.com/login?redirectTo=Lw==. [Πρόσβαση 2022].

[11] U. E. 2, «Unreal Engine 2 Documentation,» [Ηλεκτρονικό]. Available: https://docs.unrealengine.com/udk/Two/WebHome.html. [Πρόσβαση 2022].

[12] U. Engine, «Unreal Engine Forum,» [Ηλεκτρονικό]. Available: https://forums.unrealengine.com/. [Πρόσβαση 2022].

[13] «Classic Tools Retrospective: Tim Sweeney on the first version of the Unreal Editor,» 1 9 2018. [Ηλεκτρονικό]. Available: https://www.gamedeveloper.com/design/classic-tools-retrospective-tim-sweeney-on-the-first-version-of-the-unreal-editor. [Πρόσβαση 2022].

[14] CryEngine, «CryEngine Documentation,» [Ηλεκτρονικό]. Available: https://docs.cryengine.com/. [Πρόσβαση 18 6 2022].

[15] A. Lumberyard, «Amazon Lumberyard Documentation,» [Ηλεκτρονικό]. Available: https://docs.aws.amazon.com/lumberyard/index.html. [Πρόσβαση 2022].

[16] GODOT, «GODOT ducumentation,» 2022. [Ηλεκτρονικό]. Available: https://docs.godotengine.org/en/stable/index.html. [Πρόσβαση 2022].

[17] Ogre3D, «Ogre3D Documentation,» [Ηλεκτρονικό]. Available: https://www.ogre3d.org/documentation. [Πρόσβαση 2022].

[18] A. U. Vinay Bhargav Vasudevamurt, «Serious game engines: Analysis and applications,» 2015. [Ηλεκτρονικό]. Available: https://ieeexplore.ieee.org/document/7293381. [Πρόσβαση 2022].

[19] A. BARCZAK και H. WOŹNIAK, «czasopisma,» 23 01 2019. [Ηλεκτρονικό]. Available: https://czasopisma.uph.edu.pl/studiainformatica/article/view/1864/1578. [Πρόσβαση 23 05 2022].

[20] T. Norton, «Learning C# by Developing Games with Unity 3D,» [Ηλεκτρονικό].

[21] P. K. Gizem Boyraz1, «Constructing A 3d Game With Unity 3d Game Engine,» [Ηλεκτρονικό]. Available: https://fruct.org/publications/acm28/files/Boy.pdf.

[22] W. Qi, «(Serious) Games Development: The State of the Art,» 2014. [Ηλεκτρονικό]. Available: https://www.semanticscholar.org/paper/Serious-%29-Games-Development-%3A-The-State-of-the-Art-Qi/100a766cf825fa8868149b23b065da531c4bf2e3?p2df. [Πρόσβαση 2022].

[23] Göteborg, «Comparison and evaluation of 3D mobile game engines,» 2014. [Ηλεκτρονικό]. Available: https://publications.lib.chalmers.se/records/fulltext/193979/193979.pdf. [Πρόσβαση 2022].

[24] D.-j. C. R. Nicholas Harshfield, «A Unity 3D framework for algorithm animation,» 2015. [Ηλεκτρονικό]. Available: https://ieeexplore.ieee.org/abstract/document/7272955. [Πρόσβαση 2022].

[25] J. Halpern, Developing 2D Games with Unity, Apress Berkeley, CA, 2019.

[26] H. A. S. Z. S. I. Syed Muhammad, «Hybrid Animation: Implementation of Two-Dimensional (2D) Animation,» 2019. [Ηλεκτρονικό]. Available: https://iopscience.iop.org/article/10.1088/1742-6596/1529/2/022093/meta. [Πρόσβαση 2022].

[27] V. Pereira, «Learning 2D Game Development By Example,» 2014. [Ηλεκτρονικό]. Available: https://books.google.gr/books?hl=en&lr=&id=S-paBAAAQBAJ&oi=fnd&pg=PT13&dq=2D+animations+game+development&ots=X9ii57kiAE&sig=SATsjzC1p512pEcn-qWjy4BPWSM&redir_esc=y#v=onepage&q=2D%20animations%20game%20development&f=false. [Πρόσβαση 2022].

[28] R. Rouse III, Game Design Theory & Practice, Texas, 2005.

[29] Q. N.-Y. YueGao, «3D model comparison using spatial structure circular descriptor,» 2009. [Ηλεκτρονικό]. Available: https://www.sciencedirect.com/science/article/pii/S0031320309002933?casa_token=23PSyTE5xHwAAAAA:u7JKY9boSiR94ugwUmkVOql3P0np3bjBi_b5qGrqx-_ybvuvOEOKnqoExgG7GcX4OAnn3RB82LQ. [Πρόσβαση 2022].

[30] Y. L. Hyungbum Ham, «An Empirical Study for Quantitative Evaluation of Game Satisfaction,» 11 9 2006. [Ηλεκτρονικό]. Available: https://ieeexplore.ieee.org/document/4021294. [Πρόσβαση 2022].

[31] S. Rogers, Level Up - The Guide to Great Video Game Design, West Sussex UK, 2010.

[32] I. D. D. P. S. A. A. P. M. H. S. d. F. Panagiotis Petridis1, «Game Engines Selection Framework for High-Fidelity Serious Applications,» 4 4 2012. [Ηλεκτρονικό]. Available: https://ibimapublishing.com/articles/IJIW/2012/418638/. [Πρόσβαση 2022].

[33] A. S. Heinrich Söbke, «Serious Games Architectures and Engines,» 6 11 2016. [Ηλεκτρονικό]. Available: https://link.springer.com/chapter/10.1007/978-3-319-46152-6_7. [Πρόσβαση 2022].

[34] J. Halpern, «Developing 2D Games with Unity Independent Game Programming with C#,» 2019. [Ηλεκτρονικό]. Available: https://link.springer.com/content/pdf/10.1007/978-1-4842-3772-4.pdf. [Πρόσβαση 2022].

[35] Aall3dp, «The Most Common 3D File Formats in 2022,» 2022. [Ηλεκτρονικό]. Available: https://all3dp.com/2/most-common-3d-file-formats-model/. [Πρόσβαση 2022].

ΠΑΡΑΡΤΗΜΑ Α : Τα εκτελέσιμα προγράμματα στο Github

https://github.com/vasilismartsis/Dynamic_Unity_Game_Generation

ΠΑΡΑΡΤΗΜΑ Β : Ο κώδικας του εργαλείου UnityPackageGenerator

Β.1 Η κλάση ComboPropertyUserControl:

Η κλάση αυτή υλοποιεί ένα User Control το οποίο περιέχει ένα combo box. Η κλάση αυτή περιέχει επίσης και τον κώδικα υπεύθυνο για την εμφάνιση η εξαφάνιση άλλων properties ανάλογα με την επιλογή στο combo box.

```
1.  using JsonCreator.PropertyModels;
2.  using System;
3.  using System.Windows.Forms;
4.  using static JsonCreator.SceneProperties;
5.
6.  namespace JsonCreator.UserControls
7.  {
8.      public partial class ComboPropertyUserControl : PropertyUserControl
9.      {
10.         ComboProperty comboProperty;
11.
12.         public ComboPropertyUserControl(ComboProperty comboProperty)
13.         {
14.             InitializeComponent();
15.
16.             this.comboProperty = comboProperty;
17.             property = comboProperty;
18.
19.             InitializePropertyControls();
20.         }
21.
22.         void InitializePropertyControls()
23.         {
24.             TitleLabel.Text = comboProperty.title;
25.             foreach (string value in comboProperty.optionValues)
26.             {
27.                 ComboBox.Items.Add(value);
28.             }
29.             if (!comboProperty.unique)
30.                 ComboBox.SelectedItem = ComboBox.Items[0];
31.             else
32.                 ComboBox.SelectedItem = ComboBox.Items[1];
33.             ComboBox.MouseWheel += new MouseEventHandler(ComboBox_MouseWheel);
34.             ShowToolTip(TitleLabel, comboProperty.description);
35.             ShowToolTip(ComboBox, comboProperty.description);
36.         }
37.
38.         void ComboBox_MouseWheel(object sender, MouseEventArgs e)
39.         {
40.             ((HandledMouseEventArgs)e).Handled = true;
41.         }
42.
43.         public ControlCollection getItems()
44.         {
45.             return Controls;
46.         }
```

```
47.
48.         public Label getTitle()
49.         {
50.             return TitleLabel;
51.         }
52.         public ComboBox getComboBox()
53.         {
54.             return ComboBox;
55.         }
56.
57.         private void ComboBox_SelectedIndexChanged(object sender, EventArgs e)
58.         {
59.             EditPropertiesVisibility();
60.         }
61.
62.         void EditPropertiesVisibility()
63.         {
64.             if (Parent != null)
65.             {
66.                 EditPropertyChildrenVisibility();
67.                 EditOtherPropertiesVisibilityIfUnique();
68.                 EditPropertiesVisibilityByDimension();
69.
70.                 SetVisibility();
71.             }
72.         }
73.
74.         public void SetVisibility()
75.         {
76.             if (Parent.Parent.Parent != null)
77.             {
78.                 foreach (CategoryUserControl categoryUserControl in
    Parent.Parent.Parent.Controls)
79.                 {
80.                     foreach (PropertyUserControl propertyUserControl in
    categoryUserControl.getFlowLayoutPanel().Controls)
81.                         if (propertyUserControl != this)
82.                             propertyUserControl.setVisibility();
83.
84.                     //categoryUserControl.Height = 0;
85.                 }
86.             }
87.         }
88.
89.         void EditPropertyChildrenVisibility()
90.         {
91.             if (comboProperty.children != null && comboProperty.children.Count > 0)
92.                 foreach (PropertyUserControl propertyUserControl in
    ((CategoryUserControl)Parent.Parent).getFlowLayoutPanel().Controls)
93.                     if (propertyUserControl != this)
94.                         if
    (comboProperty.children.Contains(propertyUserControl.property.name))
95.                             if (ComboBox.SelectedIndex == 1)
96.
    propertyUserControl.visibilityChecks[VisibilityTypes.childrenVisibility] = false;
97.                             else
98.
    propertyUserControl.visibilityChecks[VisibilityTypes.childrenVisibility] = true;
99.         }
100.
101.          void EditOtherPropertiesVisibilityIfUnique()
102.          {
103.              if (comboProperty.unique)
104.                  foreach (CategoryUserControl categoryUserControl in
    Parent.Parent.Parent.Controls)
105.                     foreach (PropertyUserControl propertyUserControl in
    categoryUserControl.getFlowLayoutPanel().Controls)
106.                         if (propertyUserControl != this &&
    propertyUserControl.property.name == property.name)
107.                             if (ComboBox.SelectedIndex == 0)
```

```
                            {
propertyUserControl.visibilityChecks[VisibilityTypes.uniqueVisibility] = false;
                            }
                            else
propertyUserControl.visibilityChecks[VisibilityTypes.uniqueVisibility] = true;
        }

        public void EditPropertiesVisibilityByDimension()
        {
            if (property.name == PropertyNames.dimension)
                foreach (CategoryUserControl categoryUserControl in
Parent.Parent.Parent.Controls)
                    foreach (PropertyUserControl propertyUserControl in
categoryUserControl.getFlowLayoutPanel().Controls)
                        if (propertyUserControl != this)
                            if (propertyUserControl.property.dimension.ToString() ==
"both" || propertyUserControl.property.dimension.ToString() == "_" +
ComboBox.Text.ToLower())
propertyUserControl.visibilityChecks[VisibilityTypes.dimetnionVisibility] = true;
                            else
propertyUserControl.visibilityChecks[VisibilityTypes.dimetnionVisibility] = false;
        }

        private void ComboBoxProperty_VisibleChanged(object sender, EventArgs e)
        {
            EditPropertyChildrenVisibility();
            SetVisibility();
        }

        private void EditPropertiesVisibilityTimer_Tick(object sender, EventArgs e)
        {
            EditPropertiesVisibilityByDimension();
            EditPropertyChildrenVisibility();
            SetVisibility();
        }
    }
}
```

B.2 Η κλάση TextPropertyUserControl

- Η κλάση αυτή υλοποιεί ένα User Control το οποίο περιέχει ένα textbox.

```
using JsonCreator.PropertyModels;
using System.Windows.Forms;

namespace JsonCreator.UserControls
{
    public partial class TextPropertyUserControl : PropertyUserControl
    {
        private TextProperty textProperty;

        public TextPropertyUserControl(TextProperty textProperty)
        {
            InitializeComponent();

            this.textProperty = textProperty;
            property = textProperty;

            InitializePropertyControls();
        }

        void InitializePropertyControls()
        {
```

```
            TitleLabel.Text = textProperty.title;
            TextBox.Text = textProperty.value;
            ShowToolTip(TitleLabel, textProperty.description);
            ShowToolTip(TextBox, textProperty.description);
            if (textProperty.valueType == TextProperty.ValueType.numeric)
                TextBox.KeyPress += TextBox_KeyPress;
        }

        public Label getTitle()
        {
            return TitleLabel;
        }
        public TextBox getTextBox()
        {
            return TextBox;
        }
    }
}
```

B.3 Η κλάση VectorPropertyUserControl

- Η κλάση αυτή υλοποιεί ένα User Control το οποίο περιέχει ένα συνδιασμό από textbox που αναπαριστούν ένα τρισδιάστατο Vector.

```
using JsonCreator.PropertyModels;
using System.Windows.Forms;

namespace JsonCreator.UserControls
{
    public partial class VectorPropertyUserControl : PropertyUserControl
    {
        private VectorProperty vectorProperty;

        public VectorPropertyUserControl(VectorProperty vectorProperty)
        {
            InitializeComponent();

            this.vectorProperty = vectorProperty;
            property = vectorProperty;

            InitializePropertyControls();
        }

        void InitializePropertyControls()
        {
            TitleLabel.Text = vectorProperty.title;
            xTextBox.Text = vectorProperty.values.x.ToString();
            yTextBox.Text = vectorProperty.values.y.ToString();
            zTextBox.Text = vectorProperty.values.z.ToString();
            ShowToolTip(TitleLabel, vectorProperty.description);
            ShowToolTip(xTextBox, vectorProperty.description);
            ShowToolTip(yTextBox, vectorProperty.description);
            ShowToolTip(zTextBox, vectorProperty.description);
            xTextBox.KeyPress += TextBox_KeyPress;
            yTextBox.KeyPress += TextBox_KeyPress;
            xTextBox.KeyPress += TextBox_KeyPress;
        }

        public Label getTitle()
        {
            return TitleLabel;
        }
        public Label getxLabel()
        {
            return xLabel;
        }
        public TextBox getxTextBox()
```

```
        {
            return xTextBox;
        }
        public Label getyLabel()
        {
            return yLabel;
        }
        public TextBox getyTextBox()
        {
            return yTextBox;
        }
        public Label getzLabel()
        {
            return zLabel;
        }
        public TextBox getzTextBox()
        {
            return zTextBox;
        }
    }
}

```

B.4 Η κλάση SelectPropertyUserControl

- Η κλάση αυτή υλοποιεί ένα User Control το οποίο περιέχει ένα Path Select Box για την επιλογή αρχείων και φακέλων από τον υπολογιστή.

```
using JsonCreator.PropertyModels;
using Microsoft.WindowsAPICodePack.Dialogs;
using System;
using System.Windows.Forms;

namespace JsonCreator.UserControls
{
    public partial class SelectPropertyUserControl : PropertyUserControl
    {
        private SelectProperty selectProperty;

        public SelectPropertyUserControl(SelectProperty selectProperty)
        {
            InitializeComponent();

            this.selectProperty = selectProperty;
            property = selectProperty;

            InitializePropertyControls();
        }

        void InitializePropertyControls()
        {
            TitleLabel.Text = selectProperty.title;
            ChooseButton.Text = selectProperty.buttonName;

            ShowToolTip(TitleLabel, selectProperty.description);
            ShowToolTip(TextBox, selectProperty.description);
        }

        public Label getTitle()
        {
            return TitleLabel;
        }

        public TextBox getTextBox()
        {
            return TextBox;
```

```
39.         }
40.
41.         public Button getButton()
42.         {
43.             return ChooseButton;
44.         }
45.
46.         private void ChooseFileButton_Click(object sender, EventArgs e)
47.         {
48.             if (selectProperty.selectType == SelectProperty.SelectType.file)
49.             {
50.                 string filterString = "Model files (";
51.                 foreach (SelectProperty.FileFilter fileFilter in
    selectProperty.fileFilters)
52.                 {
53.                     filterString += "*." + fileFilter.ToString() + ";";
54.                 }
55.                 filterString += ")|";
56.                 foreach (SelectProperty.FileFilter fileFilter in
    selectProperty.fileFilters)
57.                 {
58.                     filterString += "*." + fileFilter.ToString() + ";";
59.                 }
60.
61.                 OpenFileDialog openFileDialog = new OpenFileDialog
62.                 {
63.                     Filter = filterString.Remove(filterString.Length - 1, 1),
64.                     FilterIndex = 1,
65.                     RestoreDirectory = true,
66.                     FileName = "",
67.                     Title = selectProperty.title,
68.                     CheckFileExists = true,
69.                     CheckPathExists = true
70.                 };
71.
72.                 if (openFileDialog.ShowDialog() == DialogResult.OK)
73.                     TextBox.Text = openFileDialog.FileName;
74.             }
75.             else
76.             {
77.                 CommonOpenFileDialog dialog = new CommonOpenFileDialog();
78.                 dialog.IsFolderPicker = true;
79.                 dialog.EnsureFileExists = true;
80.                 dialog.EnsurePathExists = true;
81.                 dialog.Title = selectProperty.title;
82.
83.                 if (dialog.ShowDialog() == CommonFileDialogResult.Ok)
84.                     TextBox.Text = dialog.FileName;
85.             }
86.         }
87.     }
88. }
89.
```

B.5 Η κλάση CategoryUserControl

- Η κλάση αυτή υλοποιεί ένα UserControl που αναπαριστά μία κατηγορία που περιέχει μέσα πολλά PropertyUserControls (ConboPropertyUserControl, SelectPropertyUserControl, κ.λπ.).

```
1.  using JsonCreator.PropertyModels;
2.  using JsonCreator.UserControls;
3.  using System;
4.  using System.Collections.Generic;
5.  using System.Windows.Forms;
6.
7.  namespace JsonCreator
8.  {
```

```
9.      public partial class CategoryUserControl : UserControl
10.     {
11.         public bool isProjectPropertiesCategory;
12.
13.         public CategoryUserControl(bool isProjectPropertiesCategory = false)
14.         {
15.             InitializeComponent();
16.
17.             this.isProjectPropertiesCategory = isProjectPropertiesCategory;
18.
19.             InitializeCategoryControls();
20.             LoadPropertiesControls();
21.         }
22.
23.         void InitializeCategoryControls()
24.         {
25.             if (isProjectPropertiesCategory)
26.             {
27.                 CategoryTitleLabel.Text = "Project Properties";
28.                 TitleLabel.Visible = false;
29.                 TitleTextBox.Visible = false;
30.                 BlackPanel.Visible = false;
31.                 Controls.Remove(XButton);
32.                 Controls.Remove(AddButton);
33.             }
34.             else
35.             {
36.                 TitleLabel.Text = "Model Name";
37.                 TitleTextBox.Text = "MyObject" + new Random().Next(0, 1000);
38.                 ShowToolTip(TitleTextBox, "The name of the final object");
39.             }
40.         }
41.
42.         void LoadPropertiesControls()
43.         {
44.             List<Property> properties;
45.             if (isProjectPropertiesCategory)
46.                 properties = SceneProperties.getProjectProperties();
47.             else
48.                 properties = SceneProperties.getModelProperties();
49.
50.             foreach (Property property in properties)
51.             {
52.                 PropertyUserControl puc;
53.
54.                 if (property.GetType() == typeof(ComboProperty))
55.                     puc = new ComboPropertyUserControl((ComboProperty)property);
56.                 else if (property.GetType() == typeof(TextProperty))
57.                     puc = new TextPropertyUserControl((TextProperty)property);
58.                 else if (property.GetType() == typeof(VectorProperty))
59.                     puc = new VectorPropertyUserControl((VectorProperty)property);
60.                 else
61.                     puc = new SelectPropertyUserControl((SelectProperty)property);
62.
63.                 CategoryFlowLayoutPanel.Controls.Add(puc);
64.             }
65.         }
66.
67.         void ShowToolTip(Control uc, string text)
68.         {
69.             // Create the ToolTip and associate with the Form container.
70.             ToolTip toolTip = new ToolTip
71.             {
72.                 // Set up the delays for the ToolTip.
73.                 InitialDelay = 1,
74.                 ReshowDelay = 5,
75.                 // Force the ToolTip text to be displayed whether or not the form is
    active.
76.                 ShowAlways = true,
```

```
77.                 IsBalloon = true
78.             };
79.
80.             // Set up the ToolTip text for the Button and Checkbox.
81.             toolTip.SetToolTip(uc, text);
82.         }
83.
84.         private void AddButton_Click(object sender, EventArgs e)
85.         {
86.             MainForm mainForm = (MainForm)ParentForm;
87.             mainForm.AddCategory();
88.         }
89.
90.         private void XButton_Click(object sender, EventArgs e)
91.         {
92.             MainForm mainForm = (MainForm)ParentForm;
93.             mainForm.RemoveCategory(this);
94.         }
95.
96.         public FlowLayoutPanel getFlowLayoutPanel()
97.         {
98.             return CategoryFlowLayoutPanel;
99.         }
100.
101.          public Label getCategoryTitleLabel()
102.          {
103.              return CategoryTitleLabel;
104.          }
105.
106.          public Label getTitle()
107.          {
108.              return TitleLabel;
109.          }
110.
111.          public TextBox getTextBox()
112.          {
113.              return TitleTextBox;
114.          }
115.
116.          public void Category_Resize(object sender, EventArgs e)
117.          {
118.              if (!isProjectPropertiesCategory)
119.                  Height = CategoryFlowLayoutPanel.Height + 163;
120.              else
121.                  Height = CategoryFlowLayoutPanel.Height + 52;
122.          }
123.
124.          private void ResizeTimer_Tick(object sender, EventArgs e)
125.          {
126.              Category_Resize(sender, e);
127.          }
128.      }
129.  }
130.
```

B.6 Η κλάση MainForm

- Η κλάση αυτή υλοποιεί ένα Form και τοποθετεί μέσα του τις διαφορετικές κατηγορίες. Υλοποιεί επίσης και τη λειτουργικότητα του κουμπιού “Create My Unity Package”, αντιγράφοντας τα μοντέλα που επέλεξε ο χρήστης μαζί με τα απαραίτητα αρχεία τους, και δημιουργώντας ένα αρχείο Json.

```
1.  using JsonCreator.PropertyModels;
2.  using JsonCreator.UserControls;
3.  using Newtonsoft.Json.Linq;
4.  using System;
```

```
5.  using System.Collections.Generic;
6.  using System.Drawing;
7.  using System.IO;
8.  using System.IO.Compression;
9.  using System.Threading.Tasks;
10. using System.Windows.Forms;
11.
12. namespace JsonCreator
13. {
14.     public partial class MainForm : Form
15.     {
16.         List<CategoryUserControl> categories = new List<CategoryUserControl>();
17.         List<string> modelNames = new List<string>();
18.         List<string> modelPaths = new List<string>();
19.         List<string> texturesFolderPaths = new List<string>();
20.         List<string> materialsFolderPaths = new List<string>();
21.         Dictionary<string, List<string>> animationPaths = new Dictionary<string,
    List<string>>();
22.         string backgroundImagePath;
23.         bool is3DProject;
24.
25.         public MainForm()
26.         {
27.             InitializeComponent();
28.
29.             AddCategory(true);
30.             AddCategory();
31.         }
32.
33.         public void AddCategory(bool isProjectPropertiesCategory = false)
34.         {
35.             CategoryUserControl categoryUserControl = new
    CategoryUserControl(isProjectPropertiesCategory);
36.             categoryUserControl.Dock = DockStyle.Top;
37.             CategoriesPanel.Controls.Add(categoryUserControl);
38.             categoryUserControl.BringToFront();
39.             //category.Height = 0;
40.             categories.Add(categoryUserControl);
41.
42.             foreach (Control category in CategoriesPanel.Controls)
43.             {
44.                 if (category.GetType() == typeof(CategoryUserControl) &&
    !((CategoryUserControl)category).isProjectPropertiesCategory)
45.                     ((CategoryUserControl)category).getCategoryTitleLabel().Text =
    "Object #" + category.TabIndex;
46.             }
47.         }
48.
49.         public void RemoveCategory(Control category)
50.         {
51.             if (categories.Count > 2)
52.             {
53.                 CategoriesPanel.Controls.Remove(category);
54.
55.                 for (int i = 0; i < categories.Count; i++)
56.                     if (categories[i] == category)
57.                         categories.RemoveAt(i);
58.             }
59.         }
60.
61.         private void CreateGamePackageButton_Click(object sender, EventArgs e)
62.         {
63.             if (PropertiesValidated())
64.             {
65.                 ErrorLabel.Visible = false;
66.                 CreatePackageZip();
67.             }
68.             else
69.                 DisplayErrorMessage("Some inputs are left empty!");
```

```
70.         }
71.
72.         bool PropertiesValidated()
73.         {
74.             bool valid = true;
75.
76.             foreach (CategoryUserControl category in categories)
77.             {
78.                 TextBox objectNameTextBox = category.getTextBox();
79.                 Label objectNameLabel = category.getTitle();
80.
81.                 if (category.getTextBox().Text == "")
82.                 {
83.                     valid &= false;
84.
85.                     objectNameTextBox.BackColor = Color.Red;
86.                     objectNameTextBox.ForeColor = Color.White;
87.                     objectNameLabel.ForeColor = Color.Red;
88.                 }
89.                 else
90.                 {
91.                     objectNameTextBox.BackColor = Color.White;
92.                     objectNameTextBox.ForeColor = Color.Black;
93.                     objectNameLabel.ForeColor = Color.FromArgb(0, 126, 249);
94.                 }
95.
96.                 foreach (PropertyUserControl propertyUserControl in
    category.getFlowLayoutPanel().Controls)
97.                     if (propertyUserControl.Visible)
98.                     {
99.                         if (propertyUserControl.GetType() ==
    typeof(VectorPropertyUserControl))
100.                         {
101.                             TextBox xTextBox =
    ((VectorPropertyUserControl)propertyUserControl).getxTextBox();
102.                             TextBox yTextBox =
    ((VectorPropertyUserControl)propertyUserControl).getyTextBox();
103.                             TextBox zTextBox =
    ((VectorPropertyUserControl)propertyUserControl).getzTextBox();
104.
105.                             Label titleLabel =
    ((VectorPropertyUserControl)propertyUserControl).getTitle();
106.
107.                             if (propertyUserControl.property.optional == false &&
    xTextBox.Text == "" || yTextBox.Text == "" || zTextBox.Text == "")
108.                             {
109.                                 valid &= false;
110.
111.                                 xTextBox.BackColor = Color.Red;
112.                                 xTextBox.ForeColor = Color.White;
113.                                 yTextBox.BackColor = Color.Red;
114.                                 yTextBox.ForeColor = Color.White;
115.                                 zTextBox.BackColor = Color.Red;
116.                                 zTextBox.ForeColor = Color.White;
117.
118.                                 titleLabel.ForeColor = Color.Red;
119.                             }
120.                             else
121.                             {
122.                                 xTextBox.BackColor = Color.White;
123.                                 xTextBox.ForeColor = Color.Black;
124.                                 yTextBox.BackColor = Color.White;
125.                                 yTextBox.ForeColor = Color.Black;
126.                                 zTextBox.BackColor = Color.White;
127.                                 zTextBox.ForeColor = Color.Black;
128.
129.                                 titleLabel.ForeColor = Color.FromArgb(0, 126, 249);
130.                             }
131.                         }
```

```
132.                         else if (propertyUserControl.GetType() ==
    typeof(TextPropertyUserControl))
133.                         {
134.                             TextBox textBox =
    ((TextPropertyUserControl)propertyUserControl).getTextBox();
135.                             Label titleLabel =
    ((TextPropertyUserControl)propertyUserControl).getTitle();
136.
137.                             if (propertyUserControl.property.optional == false &&
    textBox.Text == "")
138.                             {
139.                                 valid &= false;
140.
141.                                 textBox.BackColor = Color.Red;
142.                                 textBox.ForeColor = Color.White;
143.                                 titleLabel.ForeColor = Color.Red;
144.                             }
145.                             else
146.                             {
147.                                 textBox.BackColor = Color.White;
148.                                 textBox.ForeColor = Color.Black;
149.                                 titleLabel.ForeColor = Color.FromArgb(0, 126, 249);
150.                             }
151.
152.                         }
153.                         else if (propertyUserControl.GetType() ==
    typeof(SelectPropertyUserControl))
154.                         {
155.                             TextBox textBox =
    ((SelectPropertyUserControl)propertyUserControl).getTextBox();
156.                             Label titleLabel =
    ((SelectPropertyUserControl)propertyUserControl).getTitle();
157.
158.                             if (propertyUserControl.property.optional == false &&
    textBox.Text == "")
159.                             {
160.                                 valid &= false;
161.
162.                                 textBox.BackColor = Color.Red;
163.                                 textBox.ForeColor = Color.White;
164.                                 titleLabel.ForeColor = Color.Red;
165.                             }
166.                             else
167.                             {
168.                                 textBox.BackColor = Color.White;
169.                                 textBox.ForeColor = Color.Black;
170.                                 titleLabel.ForeColor = Color.FromArgb(0, 126, 249);
171.                             }
172.                         }
173.                     }
174.             }
175.
176.             return valid;
177.         }
178.
179.         void DisplayErrorMessage(string message)
180.         {
181.             ErrorLabel.Visible = true;
182.             ErrorLabel.Text = message;
183.         }
184.
185.         void CreatePackageZip()
186.         {
187.             if
    (Directory.Exists("C:/NecessaryFiles/ExtractedPackage/JsonSceneGenerator"))
188.             {
189.                 System.IO.DirectoryInfo dir = new
    System.IO.DirectoryInfo(@"C:/NecessaryFiles");
190.                 SetAttributesNormal(dir);
```

```
191.
    Directory.Delete("C:/NecessaryFiles/ExtractedPackage/JsonSceneGenerator", true);
192.             }
193.
194.             ZipFile.ExtractToDirectory("NecessaryFiles/JsonSceneGenerator.zip",
    "C:/NecessaryFiles/ExtractedPackage");
195.
196.             //Extract("NecessaryFiles/JsonSceneGenerator.zip",
    "NecessaryFiles/ExtractedPackage");
197.
198.             SaveFileDialog saveFileDialog = new SaveFileDialog
199.             {
200.                 Filter = "zip files (*.zip)|*.zip",
201.                 FilterIndex = 1,
202.                 RestoreDirectory = true,
203.                 FileName = "JsonSceneGenerator",
204.                 Title = "Save Zip"
205.             };
206.             if (saveFileDialog.ShowDialog() == DialogResult.OK)
207.             {
208.                 CreateModelsFolders();
209.
210.                 string startPath =
    "C:/NecessaryFiles/ExtractedPackage/JsonSceneGenerator";
211.                 string zipPath = Path.GetFullPath(saveFileDialog.FileName);
212.                 File.Delete(zipPath);
213.                 ZipFile.CreateFromDirectory(startPath, zipPath);
214.             }
215.
216.             if (Directory.Exists("C:/NecessaryFiles"))
217.             {
218.                 System.IO.DirectoryInfo dir = new
    System.IO.DirectoryInfo(@"C:/NecessaryFiles");
219.                 SetAttributesNormal(dir);
220.                 Directory.Delete("C:/NecessaryFiles", true);
221.             }
222.         }
223.
224.         internal static void SetAttributesNormal(DirectoryInfo path)
225.         {
226.             // BFS folder permissions normalizer
227.             Queue<DirectoryInfo> dirs = new Queue<DirectoryInfo>();
228.             dirs.Enqueue(path);
229.             while (dirs.Count > 0)
230.             {
231.                 var dir = dirs.Dequeue();
232.                 dir.Attributes = FileAttributes.Normal;
233.                 Parallel.ForEach(dir.EnumerateFiles(), e => e.Attributes =
    FileAttributes.Normal);
234.                 foreach (var subDir in dir.GetDirectories())
235.                     dirs.Enqueue(subDir);
236.             }
237.         }
238.
239.         /*        void Extract(string zipPath, string extractPath)
240.                 {
241.                     try
242.                     {
243.                         ProcessStartInfo processStartInfo = new ProcessStartInfo
244.                         {
245.                             WindowStyle = ProcessWindowStyle.Hidden,
246.                             FileName = Path.GetFullPath(@"7z2107-extra/7za.exe"),
247.                             Arguments = "x \"" + zipPath + "\" -o\"" + extractPath +
    "\""
248.                         };
249.                         Process process = Process.Start(processStartInfo);
250.                         process.WaitForExit();
251.                         if (process.ExitCode != 0)
252.                         {
253.                             Console.WriteLine("Error extracting {0}.", extractPath);
```

```
                        }
                    }
                    catch (Exception e)
                    {
                        Console.WriteLine("Error extracting {0}: {1}", extractPath,
    e.Message);
                        throw;
                    }
                }*/

            void CreateModelsFolders()
            {
                CopyJson();
                if (!is3DProject)
                    CopyBackgroundImage();
                CopyModel();
                CopyTextures();
                CopyMaterials();
                CopyAnimations();
            }

            void CopyJson()
            {
                StreamWriter writer = new
    StreamWriter("C:/NecessaryFiles/ExtractedPackage/JsonSceneGenerator/JsonSceneGenerator/Js
    on/SceneJson.json");
                writer.Write(CreateJson());
                writer.Close();
            }

            void CopyBackgroundImage()
            {
                if (backgroundImagePath != "")
                    File.Copy(backgroundImagePath,
    "C:/NecessaryFiles/ExtractedPackage/JsonSceneGenerator/JsonSceneGenerator/Resources/Backg
    round/" + Path.GetFileName(backgroundImagePath), true);
            }

            void CopyModel()
            {
                int i = 0;
                foreach (string modelPath in modelPaths)
                {
                    if (modelPath != "")
                    {
                        string finalFolderName = modelNames[i];

    Directory.CreateDirectory("C:/NecessaryFiles/ExtractedPackage/JsonSceneGenerator/JsonScen
    eGenerator/Resources/GameObjects/" + finalFolderName + "/Models");
                        File.Copy(modelPath,
    "C:/NecessaryFiles/ExtractedPackage/JsonSceneGenerator/JsonSceneGenerator/Resources/GameO
    bjects/" + finalFolderName + "/Models/" + finalFolderName + Path.GetExtension(modelPath),
    true);

                        if (File.Exists(modelPath + ".meta"))
                            File.Copy(modelPath + ".meta",
    "C:/NecessaryFiles/ExtractedPackage/JsonSceneGenerator/JsonSceneGenerator/Resources/GameO
    bjects/" + finalFolderName + "/Models/" + finalFolderName + Path.GetExtension(modelPath)
    + ".meta", true);
                    }

                    i++;
                }
            }

            void CopyTextures()
            {
                int i = 0;
                foreach (string texturesFolderPath in texturesFolderPaths)
```

```
310.             {
311.                 if (texturesFolderPath != "")
312.                 {
313.                     string finalFolderName = modelNames[i];
314.
    Directory.CreateDirectory("C:/NecessaryFiles/ExtractedPackage/JsonSceneGenerator/JsonScen
    eGenerator/Resources/GameObjects/" + finalFolderName + "/Textures");
315.                     CopyFilesRecursively(texturesFolderPath,
    "C:/NecessaryFiles/ExtractedPackage/JsonSceneGenerator/JsonSceneGenerator/Resources/GameO
    bjects/" + finalFolderName + "/Textures");
316.                 }
317.
318.                 i++;
319.             }
320.         }
321.
322.         void CopyMaterials()
323.         {
324.             int i = 0;
325.             foreach (string materialsFolderPath in materialsFolderPaths)
326.             {
327.                 if (materialsFolderPath != "")
328.                 {
329.                     string finalFolderName = modelNames[i];
330.
    Directory.CreateDirectory("C:/NecessaryFiles/ExtractedPackage/JsonSceneGenerator/JsonScen
    eGenerator/Resources/GameObjects/" + finalFolderName + "/Materials");
331.                     CopyFilesRecursively(materialsFolderPath,
    "C:/NecessaryFiles/ExtractedPackage/JsonSceneGenerator/JsonSceneGenerator/Resources/GameO
    bjects/" + finalFolderName + "/Materials");
332.                 }
333.
334.                 i++;
335.             }
336.         }
337.
338.         void CopyAnimations()
339.         {
340.             if (is3DProject)
341.                 Copy3DAnimations();
342.             else
343.                 Copy2DAnimations();
344.         }
345.
346.         void Copy3DAnimations()
347.         {
348.             foreach (KeyValuePair<string, List<string>> animation in animationPaths)
349.             {
350.                 string modelName = animation.Key;
351.
352.
    Directory.CreateDirectory("C:/NecessaryFiles/ExtractedPackage/JsonSceneGenerator/JsonScen
    eGenerator/Resources/GameObjects/" + modelName + "/Animations");
353.
354.                 int i = 0;
355.                 foreach (string animationPath in animation.Value)
356.                 {
357.                     if (animationPath != "")
358.                     {
359.                         string finalFolderName = modelName;
360.                         switch (i)
361.                         {
362.                             case 0:
363.                                 File.Copy(animationPath,
    "C:/NecessaryFiles/ExtractedPackage/JsonSceneGenerator/JsonSceneGenerator/Resources/GameO
    bjects/" + modelName + "/Animations/Idle" + Path.GetExtension(animationPath), true);
364.                                 break;
365.                             case 1:
```

```
366.                                 File.Copy(animationPath,
    "C:/NecessaryFiles/ExtractedPackage/JsonSceneGenerator/JsonSceneGenerator/Resources/GameO
    bjects/" + modelName + "/Animations/Walking" + Path.GetExtension(animationPath), true);
367.                                 break;
368.                             case 2:
369.                                 File.Copy(animationPath,
    "C:/NecessaryFiles/ExtractedPackage/JsonSceneGenerator/JsonSceneGenerator/Resources/GameO
    bjects/" + modelName + "/Animations/Running" + Path.GetExtension(animationPath), true);
370.                                 break;
371.                             default:
372.                                 File.Copy(animationPath,
    "C:/NecessaryFiles/ExtractedPackage/JsonSceneGenerator/JsonSceneGenerator/Resources/GameO
    bjects/" + modelName + "/Animations/Jumping" + Path.GetExtension(animationPath), true);
373.                                 break;
374.                         }
375.                     }
376.
377.                     i++;
378.                 }
379.             }
380.         }
381.
382.         void Copy2DAnimations()
383.         {
384.             foreach (KeyValuePair<string, List<string>> animation in animationPaths)
385.             {
386.                 string modelName = animation.Key;
387.
388.
    Directory.CreateDirectory("C:/NecessaryFiles/ExtractedPackage/JsonSceneGenerator/JsonScen
    eGenerator/Resources/GameObjects/" + modelName + "/Animations");
389.
390.                 int i = 0;
391.                 foreach (string animationFolderPath in animation.Value)
392.                 {
393.                     if (animationFolderPath != "")
394.                     {
395.                         string finalFolderName = modelName;
396.                         switch (i)
397.                         {
398.                             case 0:
399.
    Directory.CreateDirectory("C:/NecessaryFiles/ExtractedPackage/JsonSceneGenerator/JsonScen
    eGenerator/Resources/GameObjects/" + modelName + "/Animations/Idle");
400.                                 CopyFilesRecursively(animationFolderPath,
    "C:/NecessaryFiles/ExtractedPackage/JsonSceneGenerator/JsonSceneGenerator/Resources/GameO
    bjects/" + modelName + "/Animations/Idle");
401.                                 break;
402.                             case 1:
403.
    Directory.CreateDirectory("C:/NecessaryFiles/ExtractedPackage/JsonSceneGenerator/JsonScen
    eGenerator/Resources/GameObjects/" + modelName + "/Animations/Walking");
404.                                 CopyFilesRecursively(animationFolderPath,
    "C:/NecessaryFiles/ExtractedPackage/JsonSceneGenerator/JsonSceneGenerator/Resources/GameO
    bjects/" + modelName + "/Animations/Walking");
405.                                 break;
406.                             case 2:
407.
    Directory.CreateDirectory("C:/NecessaryFiles/ExtractedPackage/JsonSceneGenerator/JsonScen
    eGenerator/Resources/GameObjects/" + modelName + "/Animations/Running");
408.                                 CopyFilesRecursively(animationFolderPath,
    "C:/NecessaryFiles/ExtractedPackage/JsonSceneGenerator/JsonSceneGenerator/Resources/GameO
    bjects/" + modelName + "/Animations/Running");
409.                                 break;
410.                             default:
411.
    Directory.CreateDirectory("C:/NecessaryFiles/ExtractedPackage/JsonSceneGenerator/JsonScen
    eGenerator/Resources/GameObjects/" + modelName + "/Animations/Jumping");
```

```
412.                                 CopyFilesRecursively(animationFolderPath,
    "C:/NecessaryFiles/ExtractedPackage/JsonSceneGenerator/JsonSceneGenerator/Resources/GameO
    bjects/" + modelName + "/Animations/Jumping");
413.                                 break;
414.                             }
415.                         }
416.
417.                         i++;
418.                     }
419.                 }
420.             }
421.
422.             private static void CopyFilesRecursively(string sourcePath, string targetPath)
423.             {
424.                 //Now Create all of the directories
425.                 foreach (string dirPath in Directory.GetDirectories(sourcePath, "*",
    SearchOption.AllDirectories))
426.                     Directory.CreateDirectory(dirPath.Replace(sourcePath, targetPath));
427.
428.                 //Copy all the files & Replaces any files with the same name
429.                 foreach (string newPath in Directory.GetFiles(sourcePath, "*.*",
    SearchOption.AllDirectories))
430.                     File.Copy(newPath, newPath.Replace(sourcePath, targetPath), true);
431.             }
432.
433.             string CreateJson()
434.             {
435.                 string json = string.Empty;
436.
437.                 JObject newJson = new JObject();
438.                 newJson["projectProperties"] = new JObject();
439.                 newJson["gameObjects"] = new JObject();
440.
441.                 foreach (CategoryUserControl category in categories)
442.                 {
443.                     if (!category.isProjectPropertiesCategory)
444.                     {
445.                         newJson["gameObjects"][category.getTextBox().Text] = new JObject();
446.
447.                         newJson["gameObjects"][category.getTextBox().Text]["name"] =
    category.getTextBox().Text;
448.                         modelNames.Add(category.getTextBox().Text);
449.
450.
    animationPaths[newJson["gameObjects"][category.getTextBox().Text]["name"].ToString()] =
    new List<string>();
451.                     }
452.
453.                     foreach (PropertyUserControl propertyUserControl in
    category.getFlowLayoutPanel().Controls)
454.                     {
455.                         if (propertyUserControl.Visible)
456.                         {
457.                             string propertyName =
    propertyUserControl.property.name.ToString();
458.
459.                             if (propertyUserControl.GetType() ==
    typeof(VectorPropertyUserControl))
460.                             {
461.                                 TextBox xTextBox =
    ((VectorPropertyUserControl)propertyUserControl).getxTextBox();
462.                                 TextBox yTextBox =
    ((VectorPropertyUserControl)propertyUserControl).getyTextBox();
463.                                 TextBox zTextBox =
    ((VectorPropertyUserControl)propertyUserControl).getzTextBox();
464.
465.                                 JProperty x = new JProperty("x", xTextBox.Text);
466.                                 JProperty y = new JProperty("y", yTextBox.Text);
467.                                 JProperty z = new JProperty("z", zTextBox.Text);
468.
```

```
469.                                JObject vector = new JObject(new JProperty(x), new
    JProperty(y), new JProperty(z));
470.
471.                                if (!category.isProjectPropertiesCategory)
472.
    newJson["gameObjects"][category.getTextBox().Text][propertyName] = vector;
473.                                else
474.                                    newJson["projectProperties"][propertyName] = vector;
475.                            }
476.                            else if (propertyUserControl.GetType() ==
    typeof(TextPropertyUserControl))
477.                            {
478.                                TextBox textBox =
    ((TextPropertyUserControl)propertyUserControl).getTextBox();
479.
480.                                if (!category.isProjectPropertiesCategory)
481.
    newJson["gameObjects"][category.getTextBox().Text][propertyName] = textBox.Text;
482.                                else
483.                                    newJson["projectProperties"][propertyName] =
    textBox.Text;
484.                            }
485.                            else if (propertyUserControl.GetType() ==
    typeof(ComboPropertyUserControl))
486.                            {
487.                                ComboBox comboBox =
    ((ComboPropertyUserControl)propertyUserControl).getComboBox();
488.
489.                                if (!category.isProjectPropertiesCategory)
490.
    newJson["gameObjects"][category.getTextBox().Text][propertyName] =
    ((ComboProperty)propertyUserControl.property).options[comboBox.SelectedIndex];
491.                                else
492.                                    newJson["projectProperties"][propertyName] =
    ((ComboProperty)propertyUserControl.property).options[comboBox.SelectedIndex];
493.
494.                                if (propertyUserControl.property.name ==
    SceneProperties.PropertyNames.dimension)
495.                                    if (comboBox.Text == "3D")
496.                                        is3DProject = true;
497.                                    else
498.                                        is3DProject = false;
499.                            }
500.                            else if (propertyUserControl.GetType() ==
    typeof(SelectPropertyUserControl))
501.                            {
502.                                TextBox textBox =
    ((SelectPropertyUserControl)propertyUserControl).getTextBox();
503.
504.                                if (propertyUserControl.property.name ==
    SceneProperties.PropertyNames.modelPath)
505.                                    modelPaths.Add(textBox.Text);
506.                                else if (propertyUserControl.property.name ==
    SceneProperties.PropertyNames.texturesFolderPath)
507.                                    texturesFolderPaths.Add(textBox.Text);
508.                                else if (propertyUserControl.property.name ==
    SceneProperties.PropertyNames.idleAnimationPath || propertyUserControl.property.name ==
    SceneProperties.PropertyNames.walkingAnimationPath || propertyUserControl.property.name
    == SceneProperties.PropertyNames.runningAnimationPath ||
    propertyUserControl.property.name == SceneProperties.PropertyNames.jumpingAnimationPath)
509.
    animationPaths[newJson["gameObjects"][category.getTextBox().Text]["name"].ToString()].Add
    (textBox.Text);
510.                                else if (propertyUserControl.property.name ==
    SceneProperties.PropertyNames.materialsFolderPath)
511.                                    materialsFolderPaths.Add(textBox.Text);
512.                                else
513.                                    backgroundImagePath = textBox.Text;
514.                            }
```

```
                    }
                }
            }

            json = newJson.ToString();

            return json;
        }
```

A.7 Η κλάση SceneProperties

- Η στατική αυτή κλάση είναι υπεύθυνη για την δημιουργία οντοτήτων που αναπαριστούν τα μοντέλα των διαφόρων Properties.

```
using JsonCreator.PropertyModels;
using System.Collections.Generic;

namespace JsonCreator
{
    public static class SceneProperties
    {
        public enum PropertyNames
        {
            dimension,
            background,
            modelPath,
            texturesFolderPath,
            materialsFolderPath,
            hasAnimations,
            idleAnimationPath,
            walkingAnimationPath,
            runningAnimationPath,
            jumpingAnimationPath,
            position,
            rotation,
            scale,
            hasCollider,
            colliderType,
            hasPhysics,
            hasGravity,
            isControllable,
            hasMainCamera,
            cameraType,
            health
        }

        public static List<Property> getProjectProperties()
        {
            return new List<Property>
            {
                new ComboProperty
                {
                    name = PropertyNames.dimension,
                    title = "Dimension",
                    description = "The dimension of the game.",
                    options = new List<string>() { "3d", "2d"  },
                    optionValues = new List<string>() { "3D", "2D" },
                    children = new List<PropertyNames>(){ PropertyNames.hasMainCamera }
                },
                new SelectProperty
                {
                    name = PropertyNames.background,
                    title = "The Background Image of the Project",
                    description = "Select the path to the background image of the
project.",
                    buttonName = "Choose File",
```

```
52.                     selectType = SelectProperty.SelectType.file,
53.                     fileFilters = new List<SelectProperty.FileFilter> {
    SelectProperty.FileFilter.jpg, SelectProperty.FileFilter.jpeg,
    SelectProperty.FileFilter.png },
54.                     dimension = Property.Dimension._2d,
55.                     optional = true
56.                 }
57.             };
58.         }
59.
60.         public static List<Property> getModelProperties()
61.         {
62.             return new List<Property>
63.             {
64.                 new SelectProperty
65.                 {
66.                     name = PropertyNames.modelPath,
67.                     title = "The 3D Model",
68.                     description = "Select the path to the 3D model. (Only .fbx are
    supported at the time)",
69.                     buttonName = "Choose File",
70.                     selectType = SelectProperty.SelectType.file,
71.                     fileFilters = new List<SelectProperty.FileFilter> {
    SelectProperty.FileFilter.fbx },
72.                     dimension = Property.Dimension._3d
73.                 },
74.                 new SelectProperty
75.                 {
76.                     name = PropertyNames.modelPath,
77.                     title = "The 2D Model",
78.                     description = "Select the path to the 2D model.",
79.                     buttonName = "Choose File",
80.                     selectType = SelectProperty.SelectType.file,
81.                     fileFilters = new List<SelectProperty.FileFilter> {
    SelectProperty.FileFilter.jpg, SelectProperty.FileFilter.jpeg,
    SelectProperty.FileFilter.png },
82.                     dimension = Property.Dimension._2d
83.                 },
84.                 new SelectProperty
85.                 {
86.                     name = PropertyNames.texturesFolderPath,
87.                     title = "The Textures Folder",
88.                     description = "Select the path to the textures folder.",
89.                     buttonName = "Choose Folder",
90.                     selectType = SelectProperty.SelectType.folder,
91.                     dimension = Property.Dimension._3d
92.                 },
93.                 new SelectProperty
94.                 {
95.                     name = PropertyNames.materialsFolderPath,
96.                     title = "The Materials Folder*",
97.                     description = "Select the path to the materials folder, if it exists.
    (Optional)",
98.                     buttonName = "Choose Folder",
99.                     selectType = SelectProperty.SelectType.folder,
100.                      dimension = Property.Dimension._3d,
101.                      optional = true
102.                 },
103.                 new ComboProperty
104.                 {
105.                     name = PropertyNames.hasAnimations,
106.                     title = "Has Animations",
107.                     description = "Set to true if the object has gravity.",
108.                     options = new List<string>() { "true", "false" },
109.                     optionValues = new List<string>() { "True", "False" },
110.                     children = new List<PropertyNames>() {
    PropertyNames.idleAnimationPath, PropertyNames.walkingAnimationPath,
    PropertyNames.runningAnimationPath, PropertyNames.jumpingAnimationPath },
111.                     dimension = Property.Dimension.both
```

```
112.                 },
113.                 new SelectProperty
114.                 {
115.                     name = PropertyNames.idleAnimationPath,
116.                     title = "The Idle Animation of the Object*",
117.                     description = "Select the path to the Idle animation of the
    object.",
118.                     buttonName = "Choose File",
119.                     selectType = SelectProperty.SelectType.file,
120.                     fileFilters = new List<SelectProperty.FileFilter> {
    SelectProperty.FileFilter.anim, SelectProperty.FileFilter.fbx },
121.                     dimension = Property.Dimension._3d,
122.                     optional = true
123.                 },
124.                 new SelectProperty
125.                 {
126.                     name = PropertyNames.idleAnimationPath,
127.                     title = "The Idle Animation of the Object*",
128.                     description = "Select the path to the folder that contains the
    images for the Idle animation of the object.",
129.                     buttonName = "Choose Folder",
130.                     selectType = SelectProperty.SelectType.folder,
131.                     dimension = Property.Dimension._2d,
132.                     optional = true
133.                 },
134.                 new SelectProperty
135.                 {
136.                     name = PropertyNames.walkingAnimationPath,
137.                     title = "The Walking Animation of the Object*",
138.                     description = "Select the path to the Walking animation of the
    object.",
139.                     buttonName = "Choose File",
140.                     selectType = SelectProperty.SelectType.file,
141.                     fileFilters = new List<SelectProperty.FileFilter> {
    SelectProperty.FileFilter.anim, SelectProperty.FileFilter.fbx },
142.                     dimension = Property.Dimension._3d,
143.                     optional = true
144.                 },
145.                 new SelectProperty
146.                 {
147.                     name = PropertyNames.walkingAnimationPath,
148.                     title = "The Walking Animation of the Object*",
149.                     description = "Select the path to the folder that contains the
    images for the Walking animation of the object.",
150.                     buttonName = "Choose Folder",
151.                     selectType = SelectProperty.SelectType.folder,
152.                     dimension = Property.Dimension._2d,
153.                     optional = true
154.                 },
155.                 new SelectProperty
156.                 {
157.                     name = PropertyNames.runningAnimationPath,
158.                     title = "The Running Animation of the Object*",
159.                     description = "Select the path to the Running animation of the
    object.",
160.                     buttonName = "Choose File",
161.                     selectType = SelectProperty.SelectType.file,
162.                     fileFilters = new List<SelectProperty.FileFilter> {
    SelectProperty.FileFilter.anim, SelectProperty.FileFilter.fbx },
163.                     dimension = Property.Dimension._3d,
164.                     optional = true
165.                 },
166.                 new SelectProperty
167.                 {
168.                     name = PropertyNames.runningAnimationPath,
169.                     title = "The Running Animation of the Object*",
170.                     description = "Select the path to the folder that contains the
    images for the Running animation of the object.",
171.                     buttonName = "Choose Folder",
172.                     selectType = SelectProperty.SelectType.folder,
```

```
173.                     dimension = Property.Dimension._2d,
174.                     optional = true
175.                 },
176.                 new SelectProperty
177.                 {
178.                     name = PropertyNames.jumpingAnimationPath,
179.                     title = "The Jumping Animation of the Object*",
180.                     description = "Select the path to the Jumping animation of the
    object.",
181.                     buttonName = "Choose File",
182.                     selectType = SelectProperty.SelectType.file,
183.                     fileFilters = new List<SelectProperty.FileFilter> {
    SelectProperty.FileFilter.anim, SelectProperty.FileFilter.fbx },
184.                     dimension = Property.Dimension._3d,
185.                     optional = true
186.                 },
187.                 new SelectProperty
188.                 {
189.                     name = PropertyNames.jumpingAnimationPath,
190.                     title = "The Jumping Animation of the Object*",
191.                     description = "Select the path to the folder that contains the
    images for the Jumping animation of the object.",
192.                     buttonName = "Choose Folder",
193.                     selectType = SelectProperty.SelectType.folder,
194.                     dimension = Property.Dimension._2d,
195.                     optional = true
196.                 },
197.                 new VectorProperty
198.                 {
199.                     name = PropertyNames.position,
200.                     title = "Position",
201.                     description = "The position of the Object.",
202.                     values = new VectorProperty.Values(0,0,0),
203.                     dimension = Property.Dimension.both
204.                 },
205.                 new VectorProperty
206.                 {
207.                     name = PropertyNames.rotation,
208.                     title = "Rotation",
209.                     description = "The rotation of the Object.",
210.                     values = new VectorProperty.Values(0,0,0),
211.                     dimension = Property.Dimension.both
212.                 },
213.                 new VectorProperty
214.                 {
215.                     name = PropertyNames.scale,
216.                     title = "Scale",
217.                     description = "The scale of the Object.",
218.                     values = new VectorProperty.Values(1,1,1),
219.                     dimension = Property.Dimension.both
220.                 },
221.                 new ComboProperty
222.                 {
223.                     name = PropertyNames.hasCollider,
224.                     title = "Has Collider",
225.                     description = "If set to true, this object will be able to handle
    collisions with other objects",
226.                     options = new List<string>() { "true", "false"  },
227.                     optionValues = new List<string>() { "True", "False" },
228.                     children = new List<PropertyNames>() { PropertyNames.colliderType
    },
229.                     dimension = Property.Dimension.both
230.                 },
231.                 new ComboProperty
232.                 {
233.                     name = PropertyNames.colliderType,
234.                     title = "Collider Type",
235.                     description = "The collider type of the object.",
```

```
236.                     options = new List<string>() { "meshCollider", "boxCollider",
    "capsuleCollider", "sphereCollider", "terrainCollider"  },
237.                     optionValues = new List<string>() { "Auto Detect Collider", "Box
    Collider", "Capsule Collider",  "Sphere Collider", "Terrain Collider" },
238.                     dimension = Property.Dimension._3d
239.                 },
240.                 new ComboProperty
241.                 {
242.                     name = PropertyNames.colliderType,
243.                     title = "Collider Type",
244.                     description = "The collider type of the object.",
245.                     options = new List<string>() { "polygonCollider2D",
    "boxCollider2D", "capsuleCollider2D", "circleCollider2D", "compositeCollider2D",
    "edgeCollider2D"  },
246.                     optionValues = new List<string>() { "Auto Detect Collider 2D", "Box
    Collider 2D", "Capsule Collider 2D", "Circle Collider 2D", "Composite Collider 2D", "Edge
    Collider 2D" },
247.                     dimension = Property.Dimension._2d
248.                 },
249.                 new ComboProperty
250.                 {
251.                     name = PropertyNames.hasPhysics,
252.                     title = "Has Physics",
253.                     description = "Set to true if the object has physics.",
254.                     options = new List<string>() { "true", "false"  },
255.                     optionValues = new List<string>() { "True", "False" },
256.                     dimension = Property.Dimension.both,
257.                     children = new List<PropertyNames>() { PropertyNames.hasGravity }
258.                 },
259.                 new ComboProperty
260.                 {
261.                     name = PropertyNames.hasGravity,
262.                     title = "Has Gravity",
263.                     description = "Set to true if the object has gravity.",
264.                     options = new List<string>() { "true", "false"  },
265.                     optionValues = new List<string>() { "True", "False" },
266.                     dimension = Property.Dimension.both
267.                 },
268.                 new ComboProperty
269.                 {
270.                     name = PropertyNames.isControllable,
271.                     title = "Is Controllable",
272.                     description = "Set to true if the object is suposed to be
    controlled by the player. (note: Only one character can be controllable.)",
273.                     options = new List<string>() { "true", "false"  },
274.                     optionValues = new List<string>() { "True", "False" },
275.                     dimension = Property.Dimension.both,
276.                     unique = true
277.                 },
278.                 new ComboProperty
279.                 {
280.                     name = PropertyNames.hasMainCamera,
281.                     title = "Has Main Camera",
282.                     description = "Set to true if the object has the main camera
    attached to it.",
283.                     options = new List<string>() { "true", "false"  },
284.                     optionValues = new List<string>() { "True", "False" },
285.                     dimension = Property.Dimension.both,
286.                     children = new List<PropertyNames>() { PropertyNames.cameraType },
287.                     unique = true
288.                 },
289.                 new ComboProperty
290.                 {
291.                     name = PropertyNames.cameraType,
292.                     title = "Camera Type",
293.                     description = "The type of the Camera.",
294.                     options = new List<string>() { "thirdPerson", "firstPerson"  },
295.                     optionValues = new List<string>() { "Third Person", "First Person"
    },
296.                     dimension = Property.Dimension._3d,
```

```
297.                 },
298.                 new TextProperty
299.                 {
300.                     name = PropertyNames.health,
301.                     title = "Health",
302.                     description = "The health of the object.",
303.                     valueType = TextProperty.ValueType.numeric,
304.                     value = "10"
305.                 }
306.             };
307.         }
308.     }
309. }
```

ΠΑΡΑΡΤΗΜΑ C : Ο κώδικας του εργαλείου JsonSceneGenerator

B.1 Η κλάση SceneImporterExecutor

- Η κλάση αυτή δημιουργεί το μενού "Json Scene Generator/Import Scene" και προετοιμάζει την σκηνή ανάλογα με τη διάσταση (3D ή 2D) δημιουργώντας έπειτα ένα instance της κλάσης SceneImporter3D ή SceneImporter2D.

```
1.  using LitJson;
2.  using System.IO;
3.  using UnityEditor;
4.
5.  public class SceneImporterExecutor
6.  {
7.      [MenuItem("Json Scene Generator/Import Scene")]
8.      public static void ExecuteScript()
9.      {
10.         string jsonString =
    File.ReadAllText("Packages/com.sleepydevs.json_scene_generator/Json/SceneJson.json");
11.         //Convert Json into Object
12.         JsonData jsonData = JsonMapper.ToObject(jsonString);
13.         //The general details of the project contained in Json
14.         JsonData projectProperties = jsonData["projectProperties"];
15.         string dimension = projectProperties["dimension"].ToString();
16.
17.         SceneImporter createScene;
18.         if (dimension == "3d")
19.         {
20.             EditorSettings.defaultBehaviorMode = EditorBehaviorMode.Mode3D;
21.             SceneView.lastActiveSceneView.in2DMode = false;
22.
23.             createScene = new SceneImporter3D();
24.         }
25.         else
26.         {
27.             EditorSettings.defaultBehaviorMode = EditorBehaviorMode.Mode2D;
28.             SceneView.lastActiveSceneView.in2DMode = true;
29.
30.             createScene = new SceneImporter2D();
31.         }
32.     }
33. }
```

B.2 Η κλάση SceneImporter

- Η κλάση αυτή περιέχει κώδικα που είναι κοινός για την δημιουργία 3D αλλά και 2D σκηνής.

```
1.  using LitJson;
2.  using System;
3.  using System.Globalization;
4.  using System.IO;
5.  using UnityEngine;
6.
7.  public abstract class SceneImporter
8.  {
9.      protected string jsonString;
10.     protected JsonData jsonData;
11.     protected JsonData projectProperties;
12.
13.     protected CultureInfo ci = (CultureInfo)CultureInfo.CurrentCulture.Clone();
14.
15.     public SceneImporter()
16.     {
```

```
17.         //Set up float.Parse() to corectly parse decimal numbers
18.         ci.NumberFormat.NumberDecimalSeparator = ".";
19.
20.         jsonString =
    File.ReadAllText("Packages/com.sleepydevs.json_scene_generator/Json/SceneJson.json");
21.         //Convert Json into Object
22.         jsonData = JsonMapper.ToObject(jsonString);
23.         //The general details of the project contained in Json
24.         projectProperties = jsonData["projectProperties"];
25.
26.         ClearScene();
27.         SpawnObjects();
28.     }
29.
30.     //Clear all previously instantiated GameObjects in the scene
31.     protected void ClearScene()
32.     {
33.         var sceneGameObjects = GameObject.FindObjectsOfType<GameObject>();
34.         foreach (GameObject go in sceneGameObjects)
35.             UnityEngine.Object.DestroyImmediate(go);
36.     }
37.
38.     //Begin spawning new GameObjects acording to Json
39.     protected virtual void SpawnObjects()
40.     {
41.         AddGround();
42.         AddModels();
43.     }
44.
45.     protected abstract void AddGround();
46.     protected abstract void AddModels();
47.     protected abstract void AddAnimations(GameObject go);
48.     protected abstract void AddCamera(GameObject go, JsonData jsonDataModel);
49.
50.     protected void AddComponents(GameObject go, JsonData jsonDataModel)
51.     {
52.         AddPhysics(go, jsonDataModel);
53.         AddCollider(go, jsonDataModel);
54.         AddBasicScripts(go, jsonDataModel);
55.     }
56.
57.     protected abstract void AddPhysics(GameObject go, JsonData jsonDataModel);
58.
59.     protected abstract void AddCollider(GameObject go, JsonData jsonDataModel);
60.
61.     //Match Json collider type string to unity Type
62.     protected abstract Type ColliderTypeMap(string type);
63.
64.     protected virtual void AddBasicScripts(GameObject go, JsonData jsonDataModel)
65.     {
66.         //Set the health of the GameObject
67.         if (jsonDataModel.Keys.Contains("health"))
68.         {
69.             go.GetComponent<Entity>().takesDamage = true;
70.             go.GetComponent<Entity>().maxHealth =
    float.Parse(jsonDataModel["health"].ToString(), ci);
71.         }
72.     }
73. }
```

C.3 Η κλάση SceneImporter3D:

- Η κλάση αυτή κληρονομεί από την κλάση SceneImporter και είναι υπεύθυνη για την δημιουργία και είναι υπεύθυνη για την δημιουργία 3D σκηνής.

```
1.  using LitJson;
```

```
2.  using System;
3.  using System.IO;
4.  using System.Linq;
5.  using UnityEditor;
6.  using UnityEditor.Animations;
7.  using UnityEngine;
8.
9.  public class SceneImporter3D : SceneImporter
10. {
11.     //Begin spawning new GameObjects acording to Json
12.     protected override void SpawnObjects()
13.     {
14.         AddLighting();
15.         base.SpawnObjects();
16.     }
17.
18.     //Add some lighting to the scene
19.     void AddLighting()
20.     {
21.         GameObject lightGameObject = new GameObject("Sun");
22.         lightGameObject.transform.eulerAngles = new Vector3(50, -30, 0);
23.         lightGameObject.AddComponent<Light>().type = LightType.Directional;
24.         lightGameObject.GetComponent<Light>().lightmapBakeType = LightmapBakeType.Mixed;
25.         lightGameObject.GetComponent<Light>().shadows = LightShadows.Soft;
26.     }
27.
28.     protected override void AddGround()
29.     {
30.         GameObject ground = GameObject.CreatePrimitive(PrimitiveType.Plane);
31.         ground.name = "Ground";
32.         ground.transform.localScale = new Vector3(100, 0.1f, 100);
33.         ground.GetComponent<Renderer>().material = new Material(Shader.Find("Standard"));
34.         ground.GetComponent<Renderer>().sharedMaterial.color = new Color32(0, 0, 0, 255);
35.         ground.GetComponent<MeshCollider>().convex = true;
36.         ground.GetComponent<MeshRenderer>().material =
    (Material)Resources.Load("Prefabs/Prefabs3D/Ground/Materials/Grass");
37.     }
38.
39.     protected override void AddModels()
40.     {
41.         //Iterate through each model in Json
42.         foreach (string key in jsonData["gameObjects"].Keys)
43.         {
44.             JsonData jsonDataModel = jsonData["gameObjects"][key];
45.             GameObject go;
46.             string modelName = jsonDataModel["name"].ToString();
47.
48.             go = GameObject.Instantiate(Resources.Load<GameObject>("GameObjects/" +
    modelName + "/Models/" + modelName));
49.             go.name = modelName;
50.
51.             //Create Materias
52.             //if
    (!AssetDatabase.IsValidFolder("Packages/com.sleepydevs.json_scene_generator/Resources/Gam
    eObjects/" + go.name + "/Materials"))
53.             CreateMaterials(go);
54.
55.             //Set the layer of the player to TransparentFX (Because I can't add new
    layers via scripting. I may find a way to bypass that later)
56.             if (jsonDataModel.Keys.Contains("isControllable") &&
    jsonDataModel["isControllable"].ToString() == "true")
57.             {
58.                 go.layer = 1;
59.                 foreach (Transform childTransform in
    go.GetComponentInChildren<Transform>())
60.                     childTransform.gameObject.layer = 1;
61.             }
62.
63.             go.transform.position = jsonDataModel.Keys.Contains("position") ? new
    Vector3(float.Parse(jsonDataModel["position"]["x"].ToString(), ci),
```

```
    float.Parse(jsonDataModel["position"]["y"].ToString(), ci),
    float.Parse(jsonDataModel["position"]["z"].ToString(), ci)) : Vector3.zero;
64.             go.transform.eulerAngles = jsonDataModel.Keys.Contains("rotation") ? new
    Vector3(float.Parse(jsonDataModel["rotation"]["x"].ToString(), ci),
    float.Parse(jsonDataModel["rotation"]["y"].ToString(), ci),
    float.Parse(jsonDataModel["rotation"]["z"].ToString(), ci)) : Vector3.zero;
65.             go.transform.localScale = jsonDataModel.Keys.Contains("scale") ? new
    Vector3(float.Parse(jsonDataModel["scale"]["x"].ToString(), ci),
    float.Parse(jsonDataModel["scale"]["y"].ToString(), ci),
    float.Parse(jsonDataModel["scale"]["z"].ToString(), ci)) : go.transform.localScale;
66.
67.             if (jsonDataModel.Keys.Contains("hasAnimations") &&
    jsonDataModel["hasAnimations"].ToString() == "true")
68.                 AddAnimations(go);
69.             if (jsonDataModel.Keys.Contains("hasMainCamera") &&
    jsonDataModel["hasMainCamera"].ToString() == "true")
70.                 AddCamera(go, jsonDataModel);
71.             AddComponents(go, jsonDataModel);
72.         }
73.     }
74.
75.     void CreateMaterials(GameObject go)
76.     {
77.         //Get all texture paths that end with .jpg or .jpeg, etc, from Textures folder
78.         string[] textures =
    Directory.GetFiles("Packages/com.sleepydevs.json_scene_generator/Resources/GameObjects/"
    + go.name + "/" + "Textures", "*.*", SearchOption.AllDirectories)
79.             .Where(s => s.EndsWith(".jpg") || s.EndsWith(".jpeg") ||
    s.EndsWith(".png")).ToArray();
80.         //Create an array with only the names of the textures in Textures folder
81.         string[] textureNames = new string[textures.Length];
82.         for (int i = 0; i < textureNames.Length; i++)
83.             textureNames[i] = Path.GetFileNameWithoutExtension(textures[i]);
84.
85.         foreach (Transform tr in go.GetComponentsInChildren<Transform>())
86.             if (tr.GetComponent<Renderer>())
87.             {
88.                 //Create material
89.                 Material material = new Material(Shader.Find("Standard"));
90.
91.                 if (textureNames.Contains(tr.name) || textureNames.Contains(tr.name +
    "_Normal"))
92.                 {
93.                     //Assign texture to material
94.                     if (textureNames.Contains(tr.name))
95.                         material.mainTexture =
    (Texture)AssetDatabase.LoadAssetAtPath(textures[Array.IndexOf(textureNames, tr.name)],
    typeof(Texture));
96.                     //Assign normal map to material
97.                     if (textureNames.Contains(tr.name + "_Normal"))
98.                     {
99.                         //Change normal texture type to normal map
100.                            string path = textures[Array.IndexOf(textureNames, tr.name +
    "_Normal")];
101.                            TextureImporter textureImporter = AssetImporter.GetAtPath(path)
    as TextureImporter;
102.                            textureImporter.textureType = TextureImporterType.NormalMap;
103.                            AssetDatabase.ImportAsset(path);
104.
105.                            material.EnableKeyword("_NORMALMAP");
106.                            material.SetTexture("_BumpMap",
    (Texture)AssetDatabase.LoadAssetAtPath(textures[Array.IndexOf(textureNames, tr.name +
    "_Normal")], typeof(Texture)));
107.                        }
108.
109.                    //Assign material to gameobject
110.                    tr.GetComponent<Renderer>().material = material;
111.                }
112.            }
```

```
113.      }
114.
115.      //Replace generalAnimationController's animations with GameObjects animations
116.      protected override void AddAnimations(GameObject go)
117.      {
118.          Avatar avatar;
119.          AnimationClip idleAnimationClip = Resources.Load<AnimationClip>("GameObjects/"
    + go.name + "/Animations/Idle");
120.          avatar = Resources.Load<Avatar>("GameObjects/" + go.name + "/Animations/Idle");
121.          AnimationClip walkingAnimationClip =
    Resources.Load<AnimationClip>("GameObjects/" + go.name + "/Animations/Walking");
122.          avatar = Resources.Load<Avatar>("GameObjects/" + go.name +
    "/Animations/Walking");
123.          AnimationClip runningAnimationClip =
    Resources.Load<AnimationClip>("GameObjects/" + go.name + "/Animations/Running");
124.          avatar = Resources.Load<Avatar>("GameObjects/" + go.name +
    "/Animations/Running");
125.          AnimationClip jumpingAnimationClip =
    Resources.Load<AnimationClip>("GameObjects/" + go.name + "/Animations/Jumping");
126.          avatar = Resources.Load<Avatar>("GameObjects/" + go.name +
    "/Animations/Jumping");
127.
128.          Animator animator = go.AddComponent<Animator>();
129.          AnimatorController generalAnimationController =
    Resources.Load<AnimatorController>("Prefabs/AnimationController/GeneralAnimationControlle
    r");
130.
131.          animator.runtimeAnimatorController = generalAnimationController;
132.
133.          if (avatar)
134.              animator.avatar = avatar;
135.
136.          AnimatorOverrideController animatorOverrideController = new
    AnimatorOverrideController(animator.runtimeAnimatorController);
137.
138.          AnimationClipOverrides clipOverrides = new
    AnimationClipOverrides(animatorOverrideController.overridesCount);
139.
140.          animatorOverrideController.GetOverrides(clipOverrides);
141.
142.          clipOverrides["Idle"] = idleAnimationClip;
143.          clipOverrides["Walking"] = walkingAnimationClip;
144.          clipOverrides["Running"] = runningAnimationClip;
145.          clipOverrides["Jumping"] = jumpingAnimationClip;
146.
147.          animatorOverrideController.ApplyOverrides(clipOverrides);
148.
149.          animator.runtimeAnimatorController = animatorOverrideController;
150.      }
151.
152.      //Add camera only to the main character that is marked as "isControllable"
153.      protected override void AddCamera(GameObject go, JsonData jsonDataModel)
154.      {
155.          if (jsonDataModel.Keys.Contains("cameraType") &&
    jsonDataModel["cameraType"].ToString() == "firstPerson")
156.          {
157.              GameObject firstPersonCamera = new GameObject("FirstPersonCamera");
158.              firstPersonCamera.transform.parent = go.transform;
159.
160.              GameObject mainCamera = new GameObject("FirstPersonCamera");
161.              mainCamera.tag = "MainCamera";
162.              mainCamera.transform.SetPositionAndRotation(go.transform.position + new
    Vector3(0, GetCharacterHeight(go), 0), go.transform.rotation);
163.              mainCamera.transform.parent = firstPersonCamera.transform;
164.              mainCamera.AddComponent<Camera>().cullingMask = ~(1 << 1);
165.              mainCamera.AddComponent(typeof(CameraRotation));
166.          }
167.          else
168.          {
```

```
169.              GameObject thirdPersonCamera =
    GameObject.Instantiate(Resources.Load<GameObject>("Prefabs/Prefabs3D/ThirdPersonCamera/Th
    irdPersonCamera"));
170.              thirdPersonCamera.name = "ThirdPersonCamera";
171.              thirdPersonCamera.transform.SetPositionAndRotation(go.transform.position +
    new Vector3(0, GetCharacterHeight(go) + 1, 0), go.transform.rotation);
172.              thirdPersonCamera.transform.parent = go.transform;
173.          }
174.      }
175.
176.      float GetCharacterHeight(GameObject go)
177.      {
178.          bool hasBounds = false;
179.          Bounds bounds = new Bounds(Vector3.zero, Vector3.zero);
180.          for (int i = 0; i < go.transform.childCount; ++i)
181.          {
182.              Renderer childRenderer = go.transform.GetChild(i).GetComponent<Renderer>();
183.              if (childRenderer != null)
184.                  if (hasBounds)
185.                      bounds.Encapsulate(childRenderer.bounds);
186.                  else
187.                  {
188.                      bounds = childRenderer.bounds;
189.                      hasBounds = true;
190.                  }
191.          }
192.
193.          return bounds.size.y;
194.      }
195.
196.      protected override void AddPhysics(GameObject go, JsonData jsonDataModel)
197.      {
198.          if (jsonDataModel.Keys.Contains("hasPhysics") &&
    jsonDataModel["hasPhysics"].ToString() == "true")
199.              go.AddComponent(typeof(Rigidbody));
200.
201.          if (jsonDataModel.Keys.Contains("isControllable") &&
    jsonDataModel["isControllable"].ToString() == "true")
202.              go.GetComponent<Rigidbody>().freezeRotation = true;
203.
204.          if (jsonDataModel.Keys.Contains("hadGravity") &&
    jsonDataModel["hadGravity"].ToString() == "true")
205.              go.GetComponent<Rigidbody>().useGravity = false;
206.      }
207.
208.      protected override void AddCollider(GameObject go, JsonData jsonDataModel)
209.      {
210.          if (go.GetComponentInChildren<MeshFilter>() == null ||
    (jsonDataModel.Keys.Contains("isControllable") &&
    jsonDataModel["isControllable"].ToString() == "true"))
211.          {
212.              go.AddComponent(typeof(CapsuleCollider));
213.              FitColliderSizeToChildrenSize(go);
214.          }
215.          else
216.          {
217.              if (jsonDataModel.Keys.Contains("hasCollider") &&
    jsonDataModel["hasCollider"].ToString() == "true" &&
    jsonDataModel.Keys.Contains("colliderType"))
218.                  if (go.GetComponent<MeshFilter>() != null)
219.                  {
220.
    go.AddComponent(ColliderTypeMap(jsonDataModel["colliderType"].ToString()));
221.
222.                      if (jsonDataModel["colliderType"].ToString() == "meshCollider" &&
    go.GetComponent<Rigidbody>())
223.                          go.GetComponent<MeshCollider>().convex = true;
224.                  }
225.                  else
```

```
226.                     foreach (MeshFilter meshFilter in
    go.transform.GetComponentsInChildren<MeshFilter>())
227.                     {
228.                         GameObject child = meshFilter.transform.gameObject;
229.
230.                         if (child.GetComponent<MeshFilter>() != null)
231.                         {
232.
    child.AddComponent(ColliderTypeMap(jsonDataModel["colliderType"].ToString()));
233.
234.                             if (jsonDataModel["colliderType"].ToString() ==
    "meshCollider" && go.GetComponent<Rigidbody>())
235.                                 child.GetComponent<MeshCollider>().convex = true;
236.                         }
237.                     }
238.             }
239.         }
240.
241.         //If the GameObject itself has no Mesh, the collider that will be given to it will
    be a capsule collider with size acording to it's chldren
242.         void FitColliderSizeToChildrenSize(GameObject go)
243.         {
244.             if (go.GetComponent<Collider>() is CapsuleCollider)
245.             {
246.                 bool hasBounds = false;
247.                 Bounds bounds = new Bounds(Vector3.zero, Vector3.zero);
248.                 for (int i = 0; i < go.transform.childCount; ++i)
249.                 {
250.                     Renderer childRenderer =
    go.transform.GetChild(i).GetComponent<Renderer>();
251.                     if (childRenderer != null)
252.                         if (hasBounds)
253.                             bounds.Encapsulate(childRenderer.bounds);
254.                         else
255.                         {
256.                             bounds = childRenderer.bounds;
257.                             hasBounds = true;
258.                         }
259.                 }
260.
261.                 CapsuleCollider collider = (CapsuleCollider)go.GetComponent<Collider>();
262.                 collider.center = bounds.center - go.transform.position;
263.                 collider.height = bounds.size.y;
264.                 collider.center = new Vector3(0, collider.center.y, 0);
265.             }
266.         }
267.
268.         //Match Json collider type string to unity Type
269.         protected override Type ColliderTypeMap(string type)
270.         {
271.             string lowerType = type.ToLower();
272.
273.             return lowerType switch
274.             {
275.                 "boxcollider" => typeof(BoxCollider),
276.                 "capsulecollider" => typeof(CapsuleCollider),
277.                 "meshcollider" => typeof(MeshCollider),
278.                 "spherecollider" => typeof(SphereCollider),
279.                 "terraincollider" => typeof(TerrainCollider),
280.                 _ => typeof(WheelCollider)
281.             };
282.         }
283.
284.         protected override void AddBasicScripts(GameObject go, JsonData jsonDataModel)
285.         {
286.             if (jsonDataModel.Keys.Contains("isControllable") &&
    jsonDataModel["isControllable"].ToString() == "true")
287.             {
288.                 //Attach basic character script
289.                 go.AddComponent(typeof(Character3D));
```

```
290.             //Attach movement script
291.             go.AddComponent(typeof(PlayerController3D));
292.         }
293.         else
294.             //Attach basic thing script
295.             go.AddComponent(typeof(Thing3D));
296.
297.         base.AddBasicScripts(go, jsonDataModel);
298.     }
299. }
```

C.4 Η κλάση SceneImporter2D

- Η κλάση αυτή κληρονομεί από την κλάση SceneImporter και είναι υπεύθυνη για την δημιουργία και είναι υπεύθυνη για την δημιουργία 2D σκηνής.

```
1.  using LitJson;
2.  using System;
3.  using UnityEditor;
4.  using UnityEditor.Animations;
5.  using UnityEngine;
6.
7.  public class SceneImporter2D : SceneImporter
8.  {
9.      GameObject background;
10.
11.     //Begin spawning new GameObjects acording to Json
12.     protected override void SpawnObjects()
13.     {
14.         AddBackground();
15.         base.SpawnObjects();
16.     }
17.
18.     void AddBackground()
19.     {
20.         if (Resources.Load<Texture2D>("Prefabs/Prefabs2D/Background"))
21.         {
22.             Texture2D texture2D =
    Resources.Load<Texture2D>("Prefabs/Prefabs2D/Background");
23.             Sprite sprite = Sprite.Create(texture2D, new Rect(0, 0, texture2D.width,
    texture2D.height), new Vector2(0.5f, 0.5f));
24.             sprite.name = texture2D.name;
25.
26.             background = new GameObject("Background");
27.             background.AddComponent<SpriteRenderer>().sprite = sprite;
28.             background.transform.position = new Vector3(0, 0, 1);
29.         }
30.     }
31.
32.     protected override void AddGround()
33.     {
34.         if (Resources.Load<Texture2D>("Prefabs/Prefabs2D/Ground"))
35.         {
36.             //GameObject ground = new GameObject("Ground");
37.             //ground.AddComponent<SpriteRenderer>().sprite =
    AssetDatabase.GetBuiltinExtraResource<Sprite>("UI/Skin/UISprite.psd");
38.             //ground.transform.localScale = new Vector3(1000, 50, 1);
39.             //ground.transform.position = new Vector3(0, -
    (ground.GetComponent<SpriteRenderer>().bounds.extents.y / 1.5f +
    background.GetComponent<SpriteRenderer>().bounds.extents.y), -1);
40.             //ground.AddComponent<BoxCollider2D>().size = new
    Vector2(ground.GetComponent<BoxCollider2D>().size.x, 0.14f);
41.             //ground.AddComponent<Rigidbody2D>().gravityScale = 0;
42.             //ground.GetComponent<Rigidbody2D>().constraints =
    RigidbodyConstraints2D.FreezeAll;
43.             //ground.GetComponent<SpriteRenderer>().color = new Color32(85, 29, 0, 255);
```

```
44.
45.             Texture2D texture2D = Resources.Load<Texture2D>("Prefabs/Prefabs2D/Ground");
46.             Sprite sprite = Sprite.Create(texture2D, new Rect(0, 0, texture2D.width,
    texture2D.height), new Vector2(0.5f, 0.5f));
47.             sprite.name = texture2D.name;
48.
49.             GameObject grounds = new GameObject("Grounds");
50.             GameObject ground;
51.             GameObject ground1;
52.
53.             for (int i = 0; i < 5; i++)
54.             {
55.                 ground = new GameObject("Ground");
56.                 ground.transform.parent = grounds.transform;
57.                 ground.AddComponent<SpriteRenderer>().sprite = sprite;
58.                 ground.transform.localScale = new Vector2(0.3f, 0.3f);
59.                 ground.transform.position = new Vector3(i * 5.6f, -
    (ground.GetComponent<SpriteRenderer>().bounds.extents.y / 1.5f +
    background.GetComponent<SpriteRenderer>().bounds.extents.y), -1);
60.                 ground.AddComponent<BoxCollider2D>();
61.                 ground.AddComponent<Rigidbody2D>().gravityScale = 0;
62.                 ground.GetComponent<Rigidbody2D>().constraints =
    RigidbodyConstraints2D.FreezeAll;
63.
64.                 if (i != 0)
65.                 {
66.                     ground1 = new GameObject("Ground");
67.                     ground1.transform.parent = grounds.transform;
68.                     ground1.AddComponent<SpriteRenderer>().sprite = sprite;
69.                     ground1.transform.localScale = new Vector2(0.3f, 0.3f);
70.                     ground1.transform.position = new Vector3(-i * 5.6f, -
    (ground1.GetComponent<SpriteRenderer>().bounds.extents.y / 1.5f +
    background.GetComponent<SpriteRenderer>().bounds.extents.y), -1);
71.                     ground1.AddComponent<BoxCollider2D>();
72.                     ground1.AddComponent<Rigidbody2D>().gravityScale = 0;
73.                     ground1.GetComponent<Rigidbody2D>().constraints =
    RigidbodyConstraints2D.FreezeAll;
74.                 }
75.             }
76.             for (int i = 0; i < 5; i++)
77.             {
78.                 ground = new GameObject("Ground");
79.                 ground.transform.parent = grounds.transform;
80.                 ground.AddComponent<SpriteRenderer>().sprite = sprite;
81.                 ground.transform.localScale = new Vector2(0.3f, 0.3f);
82.                 ground.transform.position = new Vector3(i *
    UnityEngine.Random.Range(1.6f, 5.6f), -
    (ground.GetComponent<SpriteRenderer>().bounds.extents.y / 1.5f +
    background.GetComponent<SpriteRenderer>().bounds.extents.y), -1);
83.
84.                 if (i != 0)
85.                 {
86.                     ground1 = new GameObject("Ground");
87.                     ground1.transform.parent = grounds.transform;
88.                     ground1.AddComponent<SpriteRenderer>().sprite = sprite;
89.                     ground1.transform.localScale = new Vector2(0.3f, 0.3f);
90.                     ground1.transform.position = new Vector3(-i *
    UnityEngine.Random.Range(1.6f, 5.6f), -
    (ground1.GetComponent<SpriteRenderer>().bounds.extents.y / 1.5f +
    background.GetComponent<SpriteRenderer>().bounds.extents.y), -1);
91.                 }
92.             }
93.
94.             grounds.transform.localScale = new Vector3(1, 1.81f, 1);
95.             grounds.transform.position = new Vector3(0, 3.7f, 0);
96.         }
97.     }
98.
99.     protected override void AddModels()
100.      {
```

```
101.          //Iterate through each model in Json
102.          foreach (string key in jsonData["gameObjects"].Keys)
103.          {
104.              JsonData jsonDataModel = jsonData["gameObjects"][key];
105.              GameObject go;
106.              string modelName = jsonDataModel["name"].ToString();
107.
108.              Texture2D texture2D = Resources.Load<Texture2D>("GameObjects/" + modelName
    + "/Models/" + modelName);
109.              Sprite sprite = Sprite.Create(texture2D, new Rect(0, 0, texture2D.width,
    texture2D.height), new Vector2(0.5f, 0.5f));
110.              sprite.name = texture2D.name;
111.
112.              go = new GameObject(modelName);
113.              go.AddComponent<SpriteRenderer>().sprite = sprite;
114.
115.              //Set the layer of the player to TransparentFX (Because I can't add new
    layers via scripting. I may find a way to bypass that later)
116.              if (jsonDataModel.Keys.Contains("isControllable") &&
    jsonDataModel["isControllable"].ToString() == "true")
117.              {
118.                  go.layer = 1;
119.                  foreach (Transform childTransform in
    go.GetComponentInChildren<Transform>())
120.                      childTransform.gameObject.layer = 1;
121.              }
122.
123.              go.transform.position = jsonDataModel.Keys.Contains("position") ? new
    Vector3(float.Parse(jsonDataModel["position"]["x"].ToString(), ci),
    float.Parse(jsonDataModel["position"]["y"].ToString(), ci),
    float.Parse(jsonDataModel["position"]["z"].ToString(), ci)) : Vector3.zero;
124.              go.transform.eulerAngles = jsonDataModel.Keys.Contains("rotation") ? new
    Vector3(float.Parse(jsonDataModel["rotation"]["x"].ToString(), ci),
    float.Parse(jsonDataModel["rotation"]["y"].ToString(), ci),
    float.Parse(jsonDataModel["rotation"]["z"].ToString(), ci)) : Vector3.zero;
125.              go.transform.localScale = jsonDataModel.Keys.Contains("scale") ? new
    Vector3(float.Parse(jsonDataModel["scale"]["x"].ToString(), ci),
    float.Parse(jsonDataModel["scale"]["y"].ToString(), ci),
    float.Parse(jsonDataModel["scale"]["z"].ToString(), ci)) : go.transform.localScale;
126.
127.              if (jsonDataModel.Keys.Contains("hasAnimations") &&
    jsonDataModel["hasAnimations"].ToString() == "true")
128.                  AddAnimations(go);
129.              if (jsonDataModel.Keys.Contains("hasMainCamera") &&
    jsonDataModel["hasMainCamera"].ToString() == "true")
130.                  AddCamera(go, jsonDataModel);
131.              AddComponents(go, jsonDataModel);
132.          }
133.      }
134.
135.      //Replace generalAnimationController's animations with GameObjects animations
136.      protected override void AddAnimations(GameObject go)
137.      {
138.          Animator animator = go.AddComponent<Animator>();
139.          AnimatorController generalAnimationController =
    Resources.Load<AnimatorController>("Prefabs/AnimationController/GeneralAnimationControlle
    r");
140.
141.          animator.runtimeAnimatorController = generalAnimationController;
142.
143.          AnimatorOverrideController animatorOverrideController = new
    AnimatorOverrideController(animator.runtimeAnimatorController);
144.
145.          AnimationClipOverrides clipOverrides = new
    AnimationClipOverrides(animatorOverrideController.overridesCount);
146.
147.          animatorOverrideController.GetOverrides(clipOverrides);
148.
149.          AnimationClip idleAnimationClip = CreateAnimationClip(go, "Idle");
```

```
150.          clipOverrides["Idle"] = idleAnimationClip;
151.          AnimationClip walkingAnimationClip = CreateAnimationClip(go, "Walking");
152.          clipOverrides["Walking"] = walkingAnimationClip;
153.          AnimationClip runningAnimationClip = CreateAnimationClip(go, "Running");
154.          clipOverrides["Running"] = runningAnimationClip;
155.          AnimationClip jumpingAnimationClip = CreateAnimationClip(go, "Jumping");
156.          clipOverrides["Jumping"] = jumpingAnimationClip;
157.
158.          animatorOverrideController.ApplyOverrides(clipOverrides);
159.
160.          animator.runtimeAnimatorController = animatorOverrideController;
161.      }
162.
163.      //Creates animation clip according to the animation images provided
164.      AnimationClip CreateAnimationClip(GameObject go, string animationName)
165.      {
166.          // load animation images
167.          Texture2D[] textures2D = Resources.LoadAll<Texture2D>("GameObjects/" + go.name
    + "/Animations/" + animationName);
168.          Sprite[] sprites = new Sprite[textures2D.Length];
169.          for (int i = 0; i < sprites.Length; i++)
170.          {
171.              sprites[i] = Sprite.Create(textures2D[i], new Rect(0, 0,
    textures2D[i].width, textures2D[i].height), new Vector2(0.5f, 0.5f));
172.              sprites[i].name = textures2D[i].name;
173.          }
174.
175.          //Create animation clip
176.          AnimationClip animClip = new AnimationClip();
177.          animClip.name = animationName;
178.          animClip.frameRate = 25;   // FPS
179.
180.          //Set LoopTime to true (Except of jumping animation)
181.          if (animationName != "Jumping")
182.          {
183.              AnimationClipSettings settings =
    AnimationUtility.GetAnimationClipSettings(animClip);
184.              settings.loopTime = true;
185.              AnimationUtility.SetAnimationClipSettings(animClip, settings);
186.          }
187.
188.          //Create animationClip Curve of type SpriteRenderer
189.          EditorCurveBinding spriteBinding = new EditorCurveBinding();
190.          spriteBinding.type = typeof(SpriteRenderer);
191.          spriteBinding.path = "";
192.          spriteBinding.propertyName = "m_Sprite";
193.
194.          //Create
195.          ObjectReferenceKeyframe[] spriteKeyFrames = new
    ObjectReferenceKeyframe[sprites.Length];
196.          for (int i = 0; i < (sprites.Length); i++)
197.          {
198.              spriteKeyFrames[i] = new ObjectReferenceKeyframe();
199.              spriteKeyFrames[i].time = i / 10f;
200.              spriteKeyFrames[i].value = sprites[i];
201.          }
202.
203.          AnimationUtility.SetObjectReferenceCurve(animClip, spriteBinding,
    spriteKeyFrames);
204.
205.          return animClip;
206.      }
207.
208.      //Add camera only to the main character that is marked as "isControllable"
209.      protected override void AddCamera(GameObject go, JsonData jsonDataModel)
210.      {
211.          GameObject mainCamera = new GameObject("MainCamera");
212.          mainCamera.AddComponent<Camera>().tag = "MainCamera";
213.          mainCamera.GetComponent<Camera>().orthographic = true;
```

```
214.          //mainCamera.orthographicSize =
    background.GetComponent<SpriteRenderer>().bounds.size.y * Screen.height / Screen.width *
    0.5f;
215.          mainCamera.transform.position = new Vector3(go.transform.position.x,
    go.transform.position.y, -200);
216.          mainCamera.transform.parent = go.transform;
217.      }
218.
219.      protected override void AddPhysics(GameObject go, JsonData jsonDataModel)
220.      {
221.          if (jsonDataModel.Keys.Contains("hasPhysics") &&
    jsonDataModel["hasPhysics"].ToString() == "true")
222.          {
223.              go.AddComponent(typeof(Rigidbody2D));
224.
225.              if (jsonDataModel.Keys.Contains("isControllable") &&
    jsonDataModel["isControllable"].ToString() == "true")
226.                  go.GetComponent<Rigidbody2D>().freezeRotation = true;
227.
228.              if (jsonDataModel.Keys.Contains("hasGravity") &&
    jsonDataModel["hasGravity"].ToString() == "false")
229.                  go.GetComponent<Rigidbody2D>().gravityScale = 0;
230.          }
231.      }
232.
233.      protected override void AddCollider(GameObject go, JsonData jsonDataModel)
234.      {
235.          if (jsonDataModel.Keys.Contains("colliderType"))
236.              go.AddComponent(ColliderTypeMap(jsonDataModel["colliderType"].ToString()));
237.      }
238.
239.      protected override Type ColliderTypeMap(string type)
240.      {
241.          string lowerType = type.ToLower();
242.
243.          return lowerType switch
244.          {
245.              "boxcollider2d" => typeof(BoxCollider2D),
246.              "capsulecollider2d" => typeof(CapsuleCollider2D),
247.              "polygoncollider2d" => typeof(PolygonCollider2D),
248.              "circlecollider2d" => typeof(CircleCollider2D),
249.              "compositecollider2d" => typeof(CompositeCollider2D),
250.              _ => typeof(EdgeCollider2D)
251.          };
252.      }
253.
254.      protected override void AddBasicScripts(GameObject go, JsonData jsonDataModel)
255.      {
256.          if (jsonDataModel.Keys.Contains("isControllable") &&
    jsonDataModel["isControllable"].ToString() == "true")
257.          {
258.              //Attach basic character script
259.              go.AddComponent(typeof(Character2D));
260.              //Attach movement script
261.              go.AddComponent(typeof(PlayerController2D));
262.          }
263.          else
264.              //Attach basic thing script
265.              go.AddComponent(typeof(Thing2D));
266.
267.          base.AddBasicScripts(go, jsonDataModel);
268.      }
269. }
```

B.5 Η κλάση AnimationClipOverrides:

- Η κλάση αυτή υλοποιεί την δυνατότητα αναζήτησης τιμών μιας λίστας List<KeyValuePair<AnimationClip, AnimationClip>> με βάση ένα string κλειδί.

```
1.  using System.Collections.Generic;
2.  using UnityEngine;
3.
4.  public class AnimationClipOverrides : List<KeyValuePair<AnimationClip, AnimationClip>>
5.  {
6.      public AnimationClipOverrides(int capacity) : base(capacity) { }
7.
8.      public AnimationClip this[string name]
9.      {
10.         get
11.         {
12.             return this.Find(x => x.Key.name.Equals(name)).Value;
13.         }
14.         set
15.         {
16.             int index = this.FindIndex(x => x.Key.name.Equals(name));
17.             if (index != -1)
18.                 this[index] = new KeyValuePair<AnimationClip,
    AnimationClip>(this[index].Key, value);
19.         }
20.     }
21.  }
```

C.6 Η κλάση PlayerController3D:

- Η κλάση αυτή υλοποιεί την κίνηση του 3D χαρακτήρα του παίκτη.

```
1.  using UnityEngine;
2.
3.  public class PlayerController3D : MonoBehaviour
4.  {
5.      [SerializeField] float walkSpeed = 8;
6.      [SerializeField] float runSpeed = 12;
7.      [SerializeField] float jumpForce = 10;
8.      [SerializeField] LayerMask groundedMask = ~0;
9.      [SerializeField] float groundedRayLength = 0.2f;
10.
11.     Rigidbody rb;
12.
13.     //Vector3 moveAmount;
14.     //Vector3 smoothMoveVelocity;
15.     Vector3 moveVector;
16.     bool grounded;
17.     float speed;
18.     bool jumping;
19.
20.     // Start is called before the first frame update
21.     void Start()
22.     {
23.         if (GetComponent<Rigidbody>())
24.             rb = GetComponent<Rigidbody>();
25.         else
26.             rb = gameObject.AddComponent<Rigidbody>();
27.
28.         speed = walkSpeed;
29.     }
30.
31.     // Update is called once per frame
32.     void Update()
33.     {
34.         Vector3 playerMovementInput = new Vector3(Input.GetAxisRaw("Horizontal"), 0,
    Input.GetAxisRaw("Vertical")).normalized;
35.
```

```
        moveVector = transform.TransformDirection(playerMovementInput) * speed;

        if (Input.GetKeyDown(KeyCode.LeftShift))
            speed = runSpeed;
        else if (Input.GetKeyUp(KeyCode.LeftShift))
            speed = walkSpeed;

        if (grounded && Input.GetButtonDown("Jump"))
        {
            rb.AddForce(transform.up * jumpForce, ForceMode.Impulse);

            jumping = true;
            Invoke(nameof(FinishedJumping), 1);
        }

        if (GetComponent<Animator>())
            if (jumping)
                GetComponent<Animator>().SetBool("Jumping", true);
            else
                GetComponent<Animator>().SetBool("Jumping", false);

        Ray groundedRay = new Ray(transform.position + new Vector3(0, 0.2f, 0), -
    transform.up);
        RaycastHit hit;

        if (groundedMask.value != 0)
        {
            if (Physics.Raycast(groundedRay, out hit, transform.localScale.y +
    groundedRayLength, groundedMask))
                grounded = true;
            else
                grounded = false;
        }
        else
        {
            if (Physics.Raycast(groundedRay, out hit, transform.localScale.y +
    groundedRayLength))
                grounded = true;
            else
                grounded = false;
        }

        if (GetComponent<Animator>())
            GetComponent<Animator>().SetFloat("Speed", rb.velocity.magnitude);
    }

    void FinishedJumping()
    {
        jumping = false;
    }

    private void FixedUpdate()
    {
        rb.velocity = new Vector3(moveVector.x, rb.velocity.y, moveVector.z);
    }
}
```

C.7 Η κλάση PlayerController2D:

- Η κλάση αυτή υλοποιεί την κίνηση του 2D χαρακτήρα του παίκτη.

```
using UnityEngine;
using UnityEngine.Events;

public class PlayerController2D : MonoBehaviour
{
    [SerializeField] float walkingSpeed = 8;
```

```
7.      [SerializeField] float runningSpeed = 12;
8.      float speed = 8;
9.      [SerializeField] private float m_JumpForce = 400f;                          // Amount
    of force added when the player jumps.
10.     [Range(0, 1)][SerializeField] private float m_CrouchSpeed = .36f;           // Amount
    of maxSpeed applied to crouching movement. 1 = 100%
11.     [Range(0, .3f)][SerializeField] private float m_MovementSmoothing = .05f;  // How
    much to smooth out the movement
12.     [SerializeField] private bool m_AirControl = true;                          // Whether
    or not a player can stear while jumping;
13.     [SerializeField] private LayerMask m_WhatIsGround = ~(1 << 1);
    // A mask determining what is ground to the character
14.     private Vector2 m_GroundCheck;                              // A position marking where
    to check if the player is grounded.
15.     [SerializeField] private Transform m_CeilingCheck;                          // A
    position marking where to check for ceilings
16.     [SerializeField] private Collider2D m_CrouchDisableCollider;                // A
    collider that will be disabled when crouching
17.
18.     const float k_GroundedRadius = .2f; // Radius of the overlap circle to determine if
    grounded
19.     private bool m_Grounded;            // Whether or not the player is grounded.
20.     const float k_CeilingRadius = .2f; // Radius of the overlap circle to determine if
    the player can stand up
21.     private Rigidbody2D m_Rigidbody2D;
22.     private bool m_FacingRight = true;  // For determining which way the player is
    currently facing.
23.     private Vector3 m_Velocity = Vector3.zero;
24.
25.     [Header("Events")]
26.     [Space]
27.
28.     public UnityEvent OnLandEvent;
29.
30.     [System.Serializable]
31.     public class BoolEvent : UnityEvent<bool> { }
32.
33.     public BoolEvent OnCrouchEvent;
34.     private bool m_wasCrouching = false;
35.
36.     private void Awake()
37.     {
38.         if (GetComponent<Rigidbody2D>())
39.             m_Rigidbody2D = GetComponent<Rigidbody2D>();
40.         else
41.             m_Rigidbody2D = gameObject.AddComponent<Rigidbody2D>();
42.
43.         if (OnLandEvent == null)
44.             OnLandEvent = new UnityEvent();
45.
46.         if (OnCrouchEvent == null)
47.             OnCrouchEvent = new BoolEvent();
48.     }
49.
50.     private void Update()
51.     {
52.         if (Input.GetKey(KeyCode.LeftShift))
53.             speed = runningSpeed;
54.         else
55.             speed = walkingSpeed;
56.
57.         Move(Input.GetAxisRaw("Horizontal"), Input.GetKeyDown(KeyCode.DownArrow),
    Input.GetButtonDown("Jump"));
58.
59.         GetComponent<Animator>().SetBool("Jumping", !m_Grounded);
60.         GetComponent<Animator>().SetFloat("Speed", m_Rigidbody2D.velocity.magnitude);
61.     }
62.
63.     private void FixedUpdate()
64.     {
```

```
65.         bool wasGrounded = m_Grounded;
66.         m_Grounded = false;
67.         m_GroundCheck = new Vector2(gameObject.transform.position.x,
    gameObject.transform.position.y - gameObject.GetComponent<Collider2D>().bounds.size.y);
68.
69.         // The player is grounded if a circlecast to the groundcheck position hits
    anything designated as ground
70.         // This can be done using layers instead but Sample Assets will not overwrite
    your project settings.
71.         Collider2D[] colliders = Physics2D.OverlapCircleAll(m_GroundCheck,
    k_GroundedRadius, m_WhatIsGround);
72.         for (int i = 0; i < colliders.Length; i++)
73.         {
74.             if (colliders[i].gameObject != gameObject)
75.             {
76.                 m_Grounded = true;
77.                 if (!wasGrounded)
78.                     OnLandEvent.Invoke();
79.             }
80.         }
81.     }
82.
83.     public void Move(float move, bool crouch, bool jump)
84.     {
85.         // If crouching, check to see if the character can stand up
86.         if (!crouch)
87.         {
88.             // If the character has a ceiling preventing them from standing up, keep them
    crouching
89.             //if (Physics2D.OverlapCircle(m_CeilingCheck.position, k_CeilingRadius,
    m_WhatIsGround))
90.             //{
91.             //    crouch = true;
92.             //}
93.         }
94.
95.         //only control the player if grounded or airControl is turned on
96.         if (m_Grounded || m_AirControl)
97.         {
98.             // If crouching
99.             if (crouch)
100.              {
101.                 if (!m_wasCrouching)
102.                 {
103.                     m_wasCrouching = true;
104.                     OnCrouchEvent.Invoke(true);
105.                 }
106.
107.                 // Reduce the speed by the crouchSpeed multiplier
108.                 move *= m_CrouchSpeed;
109.
110.                 // Disable one of the colliders when crouching
111.                 if (m_CrouchDisableCollider != null)
112.                     m_CrouchDisableCollider.enabled = false;
113.             }
114.             else
115.             {
116.                 // Enable the collider when not crouching
117.                 if (m_CrouchDisableCollider != null)
118.                     m_CrouchDisableCollider.enabled = true;
119.
120.                 if (m_wasCrouching)
121.                 {
122.                     m_wasCrouching = false;
123.                     OnCrouchEvent.Invoke(false);
124.                 }
125.             }
126.
127.             // Move the character by finding the target velocity
```

```
            Vector3 targetVelocity = new Vector2(move * speed,
    m_Rigidbody2D.velocity.y);
            // And then smoothing it out and applying it to the character
            m_Rigidbody2D.velocity = Vector3.SmoothDamp(m_Rigidbody2D.velocity,
    targetVelocity, ref m_Velocity, m_MovementSmoothing);

            // If the input is moving the player right and the player is facing left...
            if (move > 0 && !m_FacingRight)
            {
                // ... flip the player.
                Flip();
            }
            // Otherwise if the input is moving the player left and the player is
    facing right...
            else if (move < 0 && m_FacingRight)
            {
                // ... flip the player.
                Flip();
            }
        }

        // If the player should jump...
        if (m_Grounded && jump)
        {
            // Add a vertical force to the player.
            m_Grounded = false;
            m_Rigidbody2D.AddForce(new Vector2(0f, m_JumpForce));
        }
    }

    private void Flip()
    {
        // Switch the way the player is labelled as facing.
        m_FacingRight = !m_FacingRight;

        // Multiply the player's x local scale by -1.
        Vector3 theScale = transform.localScale;
        theScale.x *= -1;
        transform.localScale = theScale;
    }
}
```

B.8 Η κλάση CameraRotation:

- Η κλάση αυτή υλοποιεί την λειτουργικότητα της κάμερας πρώτου προσώπου.

```
using UnityEngine;

//This is on hand
public class CameraRotation : MonoBehaviour
{
    Vector2 mouselook;
    Vector2 smoothV;

    public float sencitivity = 5.0f;
    public float smoothing = 2.0f;

    GameObject target;

    // Use this for initialization
    void Start()
    {
        target = transform.root.gameObject;
    }

    // Update is called once per frame
```

```
21.     void LateUpdate()
22.     {
23.         Vector2 md = new Vector2(Input.GetAxisRaw("Mouse X"), Input.GetAxisRaw("Mouse
    Y"));
24.
25.         md = Vector2.Scale(md, new Vector2(sencitivity * smoothing, sencitivity *
    smoothing));
26.         smoothV.x = Mathf.Lerp(smoothV.x, md.x, 1f / smoothing);
27.         smoothV.y = Mathf.Lerp(smoothV.y, md.y, 1f / smoothing);
28.         mouselook += smoothV;
29.         mouselook.y = Mathf.Clamp(mouselook.y, -90f, 90f);
30.
31.         transform.localRotation = Quaternion.AngleAxis(-mouselook.y, Vector3.right);
32.
33.         //Parent rotation
34.         target.transform.Rotate(Vector3.up * Input.GetAxis("Mouse X") * sencitivity);
35.     }
36.  }
```

C.9 Η κλάση RotateCharacter:

- Η κλάση αυτή αποτελεί κομμάτι της λειτουργικότητας της κάμερας τρίτου προσώπου και είναι υπεύθυνη για την περιστροφή του χαρακτήρα του παίκτη, μαζί και της κάμερας.

```
1.  using UnityEngine;
2.
3.  //This is on SteadyCameraPoint
4.  public class RotateCharacter : MonoBehaviour
5.  {
6.      public Vector2 mouselook;
7.      public Vector2 smoothV;
8.
9.      public float sencitivity = 5.0f;
10.     public float smoothing = 2.0f;
11.
12.     GameObject target;
13.
14.     public Vector3 temp = Vector3.zero;
15.
16.     // Use this for initialization
17.     void Start()
18.     {
19.         target = transform.root.gameObject;
20.     }
21.
22.     // Update is called once per frame
23.     void LateUpdate()
24.     {
25.         Vector2 md = new Vector2(Input.GetAxisRaw("Mouse X"), Input.GetAxisRaw("Mouse
    Y"));
26.
27.         md = Vector2.Scale(md, new Vector2(sencitivity * smoothing, sencitivity *
    smoothing));
28.         smoothV.x = Mathf.Lerp(smoothV.x, md.x, 1f / smoothing);
29.         smoothV.y = Mathf.Lerp(smoothV.y, md.y, 1f / smoothing);
30.         mouselook += smoothV;
31.         mouselook.y = Mathf.Clamp(mouselook.y, -90f, 90f);
32.
33.         transform.localRotation = Quaternion.AngleAxis(-mouselook.y, Vector3.right);
34.         //transform.localEulerAngles = transform.localEulerAngles + new Vector3(0, -
    27.66f, 0);
35.         target.transform.localRotation = Quaternion.AngleAxis(mouselook.x + 90,
    Vector3.up);
36.     }
37.  }
```

C.10 Η κλάση CameraCollision:

- Η κλάση αυτή αποτελεί κομμάτι της λειτουργικότητας της κάμερας τρίτου προσώπου και είναι υπεύθυνη για την διαχείριση των συγκρούσεων της κάμερας τρίτου προσώπου με άλλα αντικείμενα.

```
1.  using UnityEngine;
2.
3.  public class CameraCollision : MonoBehaviour
4.  {
5.      [SerializeField] int cameraBackDistance = 3;
6.
7.      Transform steadyCameraPoint;
8.      Transform steadyPlayerPoint;
9.
10.     private void Start()
11.     {
12.         steadyCameraPoint =
    transform.root.GetComponentInChildren<RotateCharacter>().transform;
13.         steadyPlayerPoint = transform.parent.parent.GetChild(1);
14.     }
15.
16.     // Update is called once per frame
17.     void Update()
18.     {
19.         Ray rightRay = new Ray(transform.position, transform.right * 2);
20.         Debug.DrawRay(transform.position, transform.right * 2, Color.green);
21.         Ray leftRay = new Ray(transform.position, -transform.right * 2);
22.         Debug.DrawRay(transform.position, -transform.right * 2, Color.blue);
23.         Ray backRay = new Ray(steadyCameraPoint.position, -steadyCameraPoint.forward *
    (cameraBackDistance + 1));
24.         Debug.DrawRay(steadyCameraPoint.position, -steadyCameraPoint.forward *
    (cameraBackDistance + 1), Color.red);
25.         Ray stadyPlayerPointRay = new Ray(steadyPlayerPoint.position,
    transform.parent.transform.right * (cameraBackDistance + 1));
26.         Debug.DrawRay(steadyPlayerPoint.position, transform.parent.transform.right *
    (cameraBackDistance + 1), Color.magenta);
27.
28.         Vector3 rightHitPoint;
29.         Vector3 leftHitPoint;
30.         Vector3 backHitPoint;
31.         Vector3 stadyPlayerPointHitPoint;
32.         float rightDistance;
33.         float leftDistance;
34.         float backDistance;
35.         float stadyPlayerPointDistance;
36.
37.         FindClosestHitObject(rightRay, out rightHitPoint, out rightDistance);
38.         FindClosestHitObject(leftRay, out leftHitPoint, out leftDistance);
39.         FindClosestHitObject(backRay, out backHitPoint, out backDistance, ~(1 << 1));
40.         FindClosestHitObject(stadyPlayerPointRay, out stadyPlayerPointHitPoint, out
    stadyPlayerPointDistance);
41.
42.         //If the ray hits somthing move camera to the hitpoint
43.         if (backDistance != 0 && backDistance <= cameraBackDistance)
44.             transform.position = backHitPoint;
45.         //Return the camera to the wanted position
46.         else
47.             transform.localPosition = new Vector3(0, 0, -cameraBackDistance);
48.
49.         /* //If too close to a wall move stadyCameraPoint left
50.          if (stadyPlayerPointDistance > 0 && stadyPlayerPointDistance < 0.5)
51.              steadyCameraPoint.localPosition = stadyPlayerPointHitPoint;
52.          //Return the stadyCameraPoint to the wanted position
53.          else
54.              steadyCameraPoint.localPosition = new Vector3(0.4f,
    steadyCameraPoint.localPosition.y, 0);*/
```

```
55.
56.         //If camera is too close to the player camera won't see the player Layer
57.         if (Vector3.Distance(transform.root.position, Camera.main.transform.position) <
    2.5)
58.             Camera.main.cullingMask &= ~(1 << LayerMask.NameToLayer("TransparentFX"));
59.         else
60.             Camera.main.cullingMask = -1;
61.     }
62.
63.     Transform FindClosestHitObject(Ray ray, out Vector3 hitPoint, out float distance, int
    layerMask = -1)
64.     {
65.         RaycastHit[] hits = Physics.RaycastAll(ray, 1000, layerMask);
66.
67.         Transform closestHit = null;
68.         distance = 0;
69.         hitPoint = Vector3.zero;
70.
71.         foreach (RaycastHit hit in hits)
72.         {
73.             if (hit.transform.root.tag != "Player" && hit.transform != transform &&
    (closestHit == null || hit.distance < distance))
74.             {
75.                 // We have hit something that is:
76.                 // a) not us
77.                 // b) the first thing we hit (that is not us)
78.                 // c) or, if not b, is at least closer than the previous closest thing
79.
80.                 closestHit = hit.transform;
81.                 distance = hit.distance;
82.                 hitPoint = hit.point;
83.             }
84.         }
85.
86.         // closestHit is now either still null (i.e. we hit nothing) OR it contains the
    closest thing that is a valid thing to hit
87.
88.         return closestHit;
89.     }
90.  }
```