ΑΛΕΞΑΝΔΡΕΙΟ ΤΕΧΝΟΛΟΓΙΚΟ ΕΚΠΑΙΔΕΥΤΙΚΟ ΙΔΡΥΜΑ ΘΕΣΣΑΛΟΝΙΚΗΣ

ΣΧΟΛΗ ΤΕΧΝΟΛΟΓΙΚΩΝ ΕΦΑΡΜΟΓΩΝ

ΤΜΗΜΑ ΗΛΕΚΤΡΟΝΙΚΩΝ ΜΗΧΑΝΙΚΩΝ ΤΕ

Κατασκευή Παιχνιδιών σε γλώσσα C#

Video Game Development with the use of C#

ΠΤΥΧΙΑΚΗ ΕΡΓΑΣΊΑ
ΤΟΥ

ΔΑΦΝΙΩΤΗΣ ΔΗΜΗΤΡΙΟΣ
511126

ΚΩΔΙΚΟΣ ΠΤΥΧΙΑΚΗΣ: 17220Μ

Ημερ. Ανάληψης: 18/10/2017

Ημερ. Παράδοσης:16/5/2018

Επιβλέπων: Τζέκης Παναγιώτης, Αναπληρωτής Καθηγητής

Θεσσαλονίκη,2018

Περιεχόμενα

Περίληψη 5
Κεφάλαιο 1 C# 8
1.1 Ιστορία της C#(C sharp) και η ανάπτυξη της 8
1.2 Διακριτικά χαρακτηριστικά της C# 9
1.2.1 Portability (Φορητότητα) 9
1.2.2 Typing 10
1.2.3 Metaprogramming 10
1.2.4 Μέθοδοι και functions(λειτουργίες) 10
1.2.5 Πρόσβαση Μνήμης (Memory Access) 11
1.2.6 Πολυμορφισμός (Polymorphism) 12
1.2.7 Language Integrated Query – LINQ 12
1.2.8 Λειτουργικός προγραμματισμός(Functional programming) 12
1.3 Το συντακτικό (syntax) της C# 13
1.4 Τα σχέδια και οι στόχοι της C# για το μέλλον 13
1.5 Τι είναι το .NET Framework 14
1.6 Επεξήγηση συμαντικών όρων 15
Κεφάλαιο 2 Unity Game Engine 18
2.1 Τι είναι το Unity Game Engine 18
2.2 Game Engine 18
2.2.1 Τι είναι μία μηχανή παιχνιδιού (Game Engine) 19
2.2.2 Τι προσφέρει η Unity Game Engine 19
2.2.3 Σκοπώς των μηχανών παιχνιδιών 19
2.3 Τι είναι το Cross-platform 20
2.4 Application Programming Interface (API) 20
2.4.1 Τι είναι τα API 20
2.4.2 Σκοπός των API 21
2.4.3 Επεξήγηση των API που στοχεύχει η Unity 21
2.5 Ιστορία της Unity και η ανάπτυξη της 22
2.5.1 Licensing 23
2.6 Επεξήγηση συμαντικών όρων 24

Κεφάλαιο 3 Βασικά Εργαλεία για την ανάπτυξη ενός παιχνιδιού στην Unity Engine 27
3.1 Interface της Unity 28
3.2 Project Window 29
3.3 The Scene View 29
3.4 The Game View 30
3.5 Παράθυρο Ιεραρχίας (Hierarchy Window) 31
3.6 The Inspector Window or Inspector 32
3.7 Microsoft Visual Studio 34
3.8 Βασικά χαρακτηριστικά του Visual Studio 35
3.8.1 Code Editor 35
3.8.2 Debugger 36
3.9 Visual Studio Tools και αλληλεπίδραση με Unity 36
3.10 Επεξήγηση συμαντικών όρων 38
Κεφάλαιο 4 Δημιουργία Βιντεοπαιχνιδιών στην Unity 40
4.1 Διαχείρηση των Assets 40
4.2 Χωρισμός sprite και δημιουργία animation 41
4.2.1 Διαδικασία εξαγωγής Sprite από μια εικόνα 42
4.2.2 Δημιουργία Animation 44
4.3 Components που χρησιμοποιήθηκαν 50
4.4 Game Jack The Giant: Κώδικας και επεξήγησή του 50
4.4.1 Διάγραμμα Ροής Main Menu Scene 50
4.4.2 Διάγραμμα ροής Option Menu Scene 52
4.4.3 Διάγραμμα Ροής Highscore Scene 52
4.4.4 Διάγραμμα ροής Gameplay Scene 53
4.5 Game Flappy Bird: Κώδικας και επεξήγησή του 58
4.5.1 Διάγραμμα Ροής Main Menu scene 58
4.5.2 Διάγραμμα Ροής Gameplay scene 58
Επίλογος 63
APPENDIX A Jack the giant Code 65
APPENDIX B Flappy Bird Code 82
Μεταφρασμένη, στην Ελληνική γλώσσα βιβλιογραφία 90
Bibliography 90

Περίληψη

Η πτυχιακή αυτή εργασία με τίτλο κατασκευή παιχνιδιών σε γλώσσα C#, επικεντρώνεται όπως φαίνεται και από τον τίτλο της στην χρήση της γλώσσας προγραμματισμού C# και του περιβάλλοντος Unity για την δημιουργία βιντεοπαιχνιδιών.

Τα εργαλεία αυτά είναι σχετικά καινούργια (εμφανίστηκαν πρώτη φορά το 2000 και 2005 αντίστοιχα αν και στην τωρινή τους μορφή πολύ αργότερα.) Η αγορά στην οποία απευθύνονται μοιράζονταν πριν ανάμεσα σε εφαργμογές C++ με DirectX API ή από Java με OpenGL. Μάλιστα ο συγγραφέας της εργασίας αυτής ξεκίνησε την πτυχιακή του με Java, μόνο για να καταλήξει μετά από μικρό χρονικό διάστημα σε υπέρμετρες δυσκολίες portability και εμπλουτισμού παιχνιδιών με νέα στοιχεία. Ερευνώντας διαφορετικές επιλογές και διαβάζοντας κριτικές για τα διάφορα περιβάλλοντα και εργαλεία υποστήριξης δημιουργίας βιντεοπαιχνιδιών ο συγγραφέας ήρθε σε επαφή με τα εργαλεία αυτά. Στην πτυχιακή αυτή θα αναφερθούν όχι μόνο τα εργαλεία αλλά και λεπτομέρειες για αυτά, όπως το γιατί είναι χρήσιμα στους στόχους αυτής της εργασίας και το κυριότερο πως τα διαφορετικά αυτά εργαλεία μπορούν να συνδυαστούν μεταξύ τους.

Αναλυτικά στο πρώτο κεφάλαιο θα αναφερθούμε στην γλώσσα προγραμματισμού C# στην ιστορία της, στο πλαίσιο το οποίο αναπτύχθηκε καθώς και τα βασικά χαρακτηριστικά της. Στο δεύτερο κεφάλαιο θα αναφερθούμε στο τι είναι το Unity Engine και στην ιστορία του. Στο τρίτο κεφάλαιο θα γίνει η ανάλυση του interface της Unity και των βασικές δυνατοτήτων που μας προσφέρει για τους στόχους αυτής της πτυχιακής. Επίσης θα αναφερθούμε στο Visual Studio και τις βασικές του δυνατότητες καθώς αυτό το εργαλείο χρησιμοποιήθηκε για να γραφτεί ο κώδικας όπως και την αλληλεπίδραση Visual Studio με Unity. Στο τέταρτο κεφάλαιο θα αναφερθούμε στο πως μπορούμε να διαχειριστούμε τα assets μας σε ένα τέτοιο project όπως την κατασκευή ενός βιντεοπαιχνιδιού, πως από μία εικόνα μπορούμε να φτιάξουμε πολλά sprites και animation μέσα στην Unity, τα διαφορετικά components που χρησιμοποιήθηκαν και η εξήγηση του κώδικα. Τέλος θα αναφερθούμε πως μπορούμε να βελτιώσουμε τα βιντεοπαιχνίδια και τα συμπεράσματα μας από αυτό το project.

Abstract

This thesis titled “Video Game Development with the use of C#”, focuses on using the C# programming language and the Unity environment to create video games.

These tools are relatively new (they first appeared in 2000 and 2005 respectively, although in their current form much later). The market addressed by these tools was previously shared between C++ applications with DirectX API or Java with OpenGL. The author of this work has begun this thesis using Java only to end up with overwhelming difficulties in portability and game enrichment with new elements. By exploring different choices and reading reviews about the various video game creation environments and tools, the author was exposed to these tools. This dissertation will cover not only the tools but also important details about them, such as why they are usefull in the objectives of this work and most importanltly how these different tools can be combined with each other.

Analytically in the first chapter we will refer to the C# programming language in its history, the context in which it was developed and its basic features. In the second chapter we will discuss what is the Unity Engine and its history. In the third chapter we will analyze the interface of Unity and the basic capabilities it offers us for the objectives of this dissertation. We will also refer to Visual Studio and its basic features as this tool was used to write the code as well as its interaction with Unity. In the fourth chapter we will discuss how we can manage our game assets in a project such as the construction of a video game, how from one image we can make many sprites and animations within Unity, the different components used and the explanation of the code. We conclude this thesis with a brief outline of diverse possibilities to improve these video games as well as our findings from this project.

Κεφάλαιο 1 C#

Σε αυτό το κεφάλαιο θα παρουσιάσουμε συνοπτικά την γλώσσα προγραμματισμού C#. Θα αναφερθούμε στους λόγους δημιουργίας της γλώσσας καθώς και την ιστορία της, θα επισημάνουμε τα χαρακτηριστικά της C#.

Η εργασία δεν έχει σκοπό την ανάλυση της C# σε βάθος. Ωστόσο η αναφορά σε αυτήν είναι σημαντική καθώς είναι η γλώσσα προγραμματισμού που χρησιμοποιήθηκε για να δημιουργηθούν τα παιχνίδια για την πτυχιακή.

Η C# είναι μία πολλαπλών υποδειγμάτων (multi-paradigm) γλώσσα προγραμματισμού και περιλαμβάνει strong typing, imperative, declarative, functional, generic, object-oriented (class based) και component-oriented είναι οι προγραμματιστικές αρχές της γλώσσας αυτής. [1]

1.1 Ιστορία της C#(C sharp) και η ανάπτυξη της

Η C# αναπτύχθηκε από την Microsoft στο πλαίσιο του .NET framework .Η C# είναι μία απο της γλώσσες προγραμματισμού που σχεδιάστηκε για την Common Language Infrastructure(CLI). Κατά την διάρκεια ανάπτυξης της πλατφόρμας .NET οι βασικές class Libraries γράφτηκαν από έναν compiler με το όνομα Simple Managed C(SMC). Τον Ιανουάριο του 1999 ο Anders Hejlsberg συγκρότησε μια ομάδα για την δημιουργία μία νέας γλώσσας προγραμματισμού με το όνομα COOL (C-like Object Oriented Language). Ο κύριος σχεδιαστής και επικεφαλής αρχιτέκτονας της C# στην Microsoft είναι ο Anders Hejlsberg,προηγουμένως συμμετείχε στον σχεδιασμό της Turbo Pascal, Borland/Embarcadero Delphi και Visual J++. Όταν το project .NET ανακοινώθηκε δημοσίως τον Ιούλιο του 2000 στο Professional Developers Conference που διεξαγόταν στο Ορλάντο της Φλόριντας [2],η γλώσσα είχε αλλάξει το όνομα της σε C#, οπού αργότερα οι class libraries καθώς και το ASP.NET(Active Server Pages) runtime εισήχθησαν στην C#.

Έπειτα από το release της C# 2.0 τον Νοέμβριο του 2005, η C# και η Java έχουν εξελιχθεί σε όλο και πιο αποκλίνουσες τροχιές,όπου οι ομοιότητες μεταξύ τους είναι λιγότερες. Μία από της πρώτες μεγάλες αλλαγές ήρθε με την εισαγωγή του generic programming και στις δύο γλώσσες (C#, Java), με πολύ διαφορετικές εφαρμογές. Η C# κάνει χρήση της Reification για να εφοδιάσει με “first class” generic objects όπου

μπορούν να χρησιμοποιηθούν όπως κάθε άλλη class,με την παραγωγή κώδικα να εκτελείται σε class-load time. Επιπλέον η C# έχει προσθέσει αρκέτα σημαντικά χαρακτηριστικά για να φιλοξενήσει functional-style programming, με αποκορύφωμα τις επεκτάσεις LINQ που κυκλοφόρησαν με το C# 3.0 και το πλαίσιο υποστήριξης του για τις εκφράσεις Lambda(λ), extension methods και anonymous types. Αυτά τα χαρακτηριστικά επιτρέπουν στους προγρμματιστές της C# να χρησιμοποιούν λειτουργικές τεχνικές προγραμματισμού οπώς closures όταν είναι συμφέρουσα για την εφαρμογή τους. Οι επεκτάσεις LINQ και η εισαγωγή functions βοηθάνε τους προγραμματιστές να μειώσουν την έκταση του boilerplate code που περιλαμβάνεται σε κοινές εργασίες, όπως η αναζήτηση βάσης δεδομένων,αναλύοντας ενα αρχείο xml ή αναζητώντας μια δομή δεδομένων,αλλάζοντας την έμφαση στην πραγματική λογική του προγράμματος για να βελτιώσει την αναγνωσιμότητα και την συντηρησιμότητα (maintainability).

Το όνομα της C# εμπνεύστηκε απο το μουσικό σύμβολο της δίεσης (#) όπου δείχνει ότι η μουσική νότα θα πρέπει να ανυψωθεί το τονικό της ύψος κατά ένα ημιτόνιο. Αυτό είναι παρόμοιο με το όνομα γλώσσας της C++, όπου το ++ υποδηλώνει ότι μια μεταβλητή θα πρέπει να αυξηθεί κατά 1. Το συμβολο # μοίαζει επίσης με ένα σύμπλεγμα τεσσάρων συμβόλων + υποδηλώνοντας ότι η γλώσσα είναι μια αύξηση της C++. [1]

1.2 Διακριτικά χαρακτηριστικά της C#

Μερικά αξιοσημείωτα χαρακτηριστικά της C# που το διακρίνουν από την C, C++ και Java είναι:

1.2.1 Portability (Φορητότητα)

Η C# είναι η γλώσσα προγραμματισμού που αντικατοπτρίζει περισσότερο την υποκείμενη CLI (Common Language Infrastructure). Οι περισσότεροι από τους intrinsic types αντιστοιχούν σε value types που εφαρμόζονται από το CLI Framework. Ωστόσο η προσδιαγραφή της γλώσσας δεν δηλώνει τις απαιτήσεις δημιουργίας κώδικα του compiler: δηλαδή, δεν δηλώνει ότι ένας compiler C# πρέπει να στοχεύσει σε έναν CLR ή να δημιουργήσει μια CLI ή να δημιουργήσει οποιαδήποτε άλλη συγκεκριμένη μορφή. Θεωρητικά ένας compiler C# θα μπορούσε να παράγει κώδικα μηχανής όπως οι παραδοσιακοί compilers C++ και Fortran. [1]

1.2.2 Typing

Το C# υποστηρίζει το strong typing με implicit μεταβλητές δηλώσεις με τη λέξη var και τυχαία πληκτρολογημένες συστοιχίες με την λέξη κλειδί new [ ] που ακολουθείται από έναν αρχικοποιητή συλλογής (collection initializer).

Η C# υποστηρίζει έναν αυστηρό τύπο δεδομένων την Boolean. Δηλώσεις που λαμβάνουν συνθήκες οπώς “while” και “if”, απαιτούν μια έκφραση ενός τύπου που υλοποιεί τον πραγματικό χειριστή, όπως ο τύπος Boolean. Ενώ η C++ έχει επίσης τύπο Boolean, μπορεί να μετατραπεί ελεύθερα σε και από ακέραιους αριθμούς. Εκφράσεις όπως if(a) απαιτούν μόνο ότι το a είναι μετατρέψιμο σε bool επιτρέποντας το a να είναι int ή δείκτης(pointer).

Η C# έχει explicit support για συνδιακύμανση και αντιστοίχιση σε γενικούς τύπους, σε αντίθεση με την C++, η οποία έχει κάποιο βαθμό υποστήριξης για αντισταθμίσεις μέσω της σημασιολογίας των τύπων επιστροφής σε εικονικές μεθόδους.

Η C# δεν επιτρέπει global variables ή functions.Όλες οι μέθοδοι και τα μέλη πρέπει να δηλώνονται μέσα στις κλάσεις (classes). Τα στατικά μέλη των public classes μπορούν να υποκαταστήσουν τις global variables και functions. Οι τοπικές μεταβλητές(local variables) δεν μπορούν να σκιάζουν τις μεταβλητές του enclosing block, σε αντίθεση με την C και την C++. [1]

1.2.3 Metaprogramming

Το metaprogramming μέσω των χαρακτηριστικών της C# είναι μέρος της γλώσσας. Πολλά από αυτά τα χαρακτηριστικά αντιγράφουν την λειτουργικότητα των οδηγιών που εξαρτώνται απο την πλατφόρμα GCC και την Visual C++. Το metaprogramming είναι μια τεχνική προγραμματισμού στην οποία τα προγράμματα υπολογιστών έχουν την ικανότητα να αντιμετωπίζουν τα προγράμματα ως δεδομένα. Σημαίνει ότι ένα πρόγραμμα μπορεί να σχεδιαστεί για να διαβάζει, να παράγει, να αναλύει ή να μετασχηματίζει άλλα προγράμματα, ακόμα και να τροποποιείται κατά την λειτουργία. [3]

1.2.4 Μέθοδοι και functions(λειτουργίες)

Οι μέθοδοι στην γλώσσα προγρμματισμού είναι τα μέλη μιας class σε ένα project, μερικές μέθοδοι έχουν υπογραφές (signatures) και μερικές δεν έχουν. Οι μέθοδοι μπορεί

να είναι void(κενές) ή μπορούν να επιστρέψουν κάτι σαν string, integer, float, bool. Εάν μια μέθοδος είναι void,σημαίνει ότι η μέθοδος δεν επιστρέφει κανένα τύπο δεδομένων. Όπως και η C++ και σε αντίθεση με την Java, οι προγραμματιστές C# πρέπει να χρησιμοποιήσουν λέξεις κλειδιά για να επιτρέψουν την αντικατάσταση μεθόδων από τα subclasses (υποκατηγορίες).

Οι μέθοδοι επέκτασης στην C# επιτρέπουν στους προγρμματιστές να χρησιμοποιούν στατικές μεθόδους (static methods) σαν να ήταν μέθοδοι από τον πίνακα μεθόδων μιας class, επιτρέποντας στους προγραμματιστές να προσθέσουν μεθόδους που θεωρούν ότι θα έπρεπε να υπάρχουν σε αυτό το object και τα παράγωγα του.

Η C# υποστηρίζει strong typing function pointers μέσω της λέξης κλειδί “delegate”. Η C# έχει σημασιολογία που περιβάλλει συγκεκριμένα publish-subscribe style events, αν και στην C# χρησιμοποιούνται τα delegates για να το πράξει. Η C# κλήσεις συγχρονισμένης μεθόδους(synchronized method calls) όπως η Java μέσω του χαρακτηρισμού [MethodImpl (MethodImplOptions.Synchronized)], και έχει υποστήριξη για αμοιβαία αποκλειστικές κλειδαριές(exclusive locks) μέσω της λέξης κλειδί ”lock”. [1]

1.2.5 Πρόσβαση Μνήμης (Memory Access)

Η C# οι δείκτες διεύθυνσης μνήμης (memory address pointers) μπορούν να χρησιμοποιηθούν μόνο σε μπλοκ που χαρακτηρίζονται ειδικά ως μη ασφαλείς. Η πρόσβαση των περισσότερων object πραγματοποιείται μέσω ασφαλών object references, οι οποίες πάντοτε είτε οδηγούν σε “ζωντανό” object ή έχουν καθορισμένη μηδενική τιμή, είναι αδύνατο να αποκτηθεί μια αναφορά σε ένα “νεκρό” object (ένα object που έχει συλλεχθεί στα σκοπίδια (garbage)) ή σε ένα τυχαίο μπλοκ μνήμης. Ένας μη ασφαλής δείκτης μπορεί να δείχνει μια εμφάνιση ενός value-type, array, string ή ένα μπλοκ μνήμης που κατανέμεται σε μια στοίβα. Ο κώδικας που δεν έχει επισημανθεί ως μη ασφαλής μπορεί ακόμα να αποθηκεύει και να χειρίζεται τους δείκτες μέσω του τύπου “System.IntPtr”, αλλά δεν μπορεί να τους αποδώσει dereference.

Η διαχείριση μνήμης δεν μπορεί να απελευθερωθεί ρητά, αντίθετα συλλέγονται αυτόματα απορρίμματα. Η συλλογή σκουπιδιώ αντιμετωπίζει το πρόβλημα των διαρροών μνήμης απελευθερώνοντας τον προγραμματιστή από την ευθύνη για την αποδέσμευση της μνήμης που δεν χρειάζεται πλέον. [1]

1.2.6 Πολυμορφισμός (Polymorphism)

Σε αντίθεση με την C++, η C# δεν υποστηρίζει πολλαπλή κληρονομικότητα (inheritance), αν και μια class μπορεί να υλοποήσει οποιοδήποτε αριθμό διεπαφών. Αυτή ήταν μια απόφαση σχεδιασμού από τον επικεφαλή αρχιτέκοντα της γλώσσας Hejlsberg για να αποφευχθεί η επιπλοκή και να απλοποιηθούν οι αρχιτεκτονικές απαιτήσεις σε όλο το CLI. Κατά την εφαρμογή πολλαπλών διεπαφών που περιέχουν μια μέθοδο με την ίδια υπογραφή (signature), η C# επιτρέπει την εφαρμογή κάθε μεθόδου ανάλογα με ποια διεπαφή καλείται αυτή η μέθοδος ή όπως στην Java επιτρέπει την εφαρμογή της μεθόδου μία φορά και αυτή η επίκληση σε μία κλήση μέσω οποιασδήποτα διεπαφής της class. Ωστόσο, σε αντίθεση με την Java η C# υποστηρίζει την υπερφόρτωση του operator. Μόνο οι πιο συχνά υπερφορωμένοι operators στην C++ μπορούν να επιβαρύνονται με C#. [1]

1.2.7 Language Integrated Query – LINQ

Η C# έχει τη δυνατότητα να χρησιμοποιεί LINQ μέσω του Microsoft .NET Framework με το IEnumerable Interface. Ο προγραμματιστής μπορεί να αναζητήσει οποιαδήποτε κλάσση συλλογής .NET, έγγραφα XML, σύνολα δεδομένων ADO.NET και βάσης δεδομένων SQL. Υπάρχουν μερικά πλεονεκτήματα στην χρήση του LINQ στο C# και είναι τα εξής: υποστήριξη IntelliSense, ισχυρές δυνατότητες φιλτραρίσματος, type safe με δυνατότητα ελέγχου σφαλμάτων και φέρνει συνοχή για την αναζήτηση δεδομένων σε διάφορες πηγές. Υπάρχουν πολλές διαφορετικές δομές γλώσσας που μπορούν να χρησιμοποιηθούν με την LINQ και είναι εκφράσης λ(lambda), anonymous types, implicitly typed μεταβλητές, μέθοδοι επέκτασης και object initializers. [1]

1.2.8 Λειτουργικός προγραμματισμός(Functional programming)

Αν και πρωτίστως ήταν μία επιτακτική γλώσσα, η C# 2.0 προσέφερε περιορισμένη υποστήριξη για λειτουργικό προγραμματισμό μέσω first-class functions και closures με την μορφή anonymous delegates. Η C# 3.0 επέκτεινε την υποστήριξη για λειτουργικό προγραμματισμό με την εισαγωγή ελαφριάς σύνταξης για τις εκφράσεις λ, μεθόδους επέκτασης και μία λίστα κατανόησης συντακτικού (syntax) με την μορφή μια γλώσσας “query comprehension”. [1]

1.3 Το συντακτικό (syntax) της C#

Το βασικό συντακτικό της γλώσσας C# είναι παρόμοια με αυτή άλλων γλωσσών τύπου C, C++ και Java. Συγκεκριμένα: [1]

- Τα Semicolons(;) χρησιμοποιούνται για να δηλώσουν το τέλος μια δήλωσης.
- Οι κυρτές παρενθέσεις χρησιμοποιούνται για την ομαδοποίηση δηλώσεων.Οι δηλώσεις ομαδοποιούνται συνήθως σε μεθόδους function, classes και classes σε namespaces.
- Οι μεταβλητές αποδίδονται με τη χρήση ενός συμβόλου ισότητας(=), αλλά συγκρίνονται χρησιμοποιώντας δύο διαδοχικά σύμβολα ισότητας (==).
- Οι τετράγωνες αγκύλες ([ ]) χρησιμοποιούνται με συστοιχίες, τόσο για να τις δηλώσουν όσο και για να αποκτήσουν μια τιμή σε ένα δεδομένο δείκτη σε μία απο αυτές.

1.4 Τα σχέδια και οι στόχοι της C# για το μέλλον

Η ECMA έχει standard lists με τους στόχους σχεδιασμού(design goals) για την C#. [1]

- Η γλώσσα προορίζεται να είναι μία απλή, σύγχρονη, γενικής χρήσης, object-oriented γλώσσα προγραμματισμού.
- Η γλώσσα και οι εφαρμογές της πρέπει να παρέχουν υποστήριξη για αρχές software engineering όπως έλεγχος strong type, έλεγχος array bounds, ανίχνευση προσπαθειών για χρήση μη αρχικών μεταβλητών και αυτόματη συλλογή απορριμμάτων.Η δύναμη και η ανθεκτικότητα του software καθώς και η παραγγικότητα του προγραμματιστή είναι σημαντικά.
- Η γλώσα προορίζεται για χρήση στην ανάπτυξη software components κατάλληλων για ανάπτυξη σε κατανεμημένα περιβάλλοντα.
- Η φορητότητα είναι πολύ σημαντική για τον πηγαίο κώδικα(source code) και τους προγραμματιστές, ειδικά εκείνους που είναι ήδη εξοικειωμένοι με C και C++.
- Η υποστήριξη της διεθνοποίησης είναι πολύ σημαντική.
- Η C# προορίζεται να είναι κατάλληλη για την εγγραφή εφαρμογών τόσο για τα φιλοξενούμενα όσο και για τα ενσωματωμένα συστήματα,που κυμαίνονται από τα πολύ μεγάλα που χρησιμοποιούν εξελιγμένα λειτουργικά συστήματα μέχρι τα πολύ μικρά με ειδικές λειτουργίες.

- Παρόλο που οι εφαρμογές C#προορίζονται να είναι οικονομικές όσον αφορά τις απαιτήσεις ισχύος μνήμης και επεξεργασίας, η γλώσσα δεν προορίζεται να ανταγωνιστεί άμεσα την απόδοση και το μέγεθος με τη γλώσσα C ή την γλώσσα assembly.

Στην αρχή του κεφαλαίου της ενότητας για την ιστορία της C# έγινε η αναφορά του .Net Framework. Στην ενότητα που ακολουθεί θα γίνει περαιτέρω αναφορά για το τι είναι το .Net Framework.

1.5 Τι είναι το .NET Framework

Το project .NET Framework ανακοινώθηκε στο ίδιο Professional Developer Conference του 2000 με την C#. Το .NET Framework είναι ένα λογισμικό ανεπτυγμένο από την Microsoft όπου κυρίως τρέχει σε Microsoft Windows. Διαθέτη μία μέγαλη class library με το όνομα FCL(Framework Class Library). Με την χρήση της συγκεκριμένης βιβλιοθήκης μπορούμε και έχουμε αλληλεπίδραση μεταξύ διαφορετικών γλωσσών προγραμματισμού. Δηλαδή κάθε γλώσσα προγραμματισμού οπού θα υπάρχει αυτή η αλληλεπίδραση θα μπορεί να χρησιμοποιεί κώδικα γραμμένο σε άλλη γλώσσα. Τα προγράμματα που είναι γραμμένα για το .NET Framework εκτελούνται σε λογισμικό περιβάλλον CLR (Common Language Runtime). Μία εφαρμογή virtual machine οπού μας προμηθεύει με υπηρεσίες ασφάλεια,διαχείριση μνήμης και χειρισμός εξαιρέσεων (exception handling). [4] Η FCL και CLR μαζί απαρτίζουν το .NET Framework. [5]

Το CLR είναι ένα component της Microsoft’s .NET Framework όπου διαχειρίζεται την εκτέλεση των προγραμμάτων .NET. Όλα τα προγράμματα γραμμένα στο .NET Framework ανεξαρτήτος της γλώσσας προγραμματισμού εκτελούνται απο την CLR.

Το FCL παρέχει στον χρήστη interface, πρόσβαση δεδομένων, συνδεσιμότητα βάσεων δεδομένων, κρυπτογραφία ,ανάπτυξη εφαρμογών web ,αριθμητικοί αλγόριθμοι και επικοινωνίες δικτύου. Οι προγραμματιστές έχουν την δυνατότητα να παράγουν λογισμικό συνδυάζοντας τον κώδικα τους με το .NET Framework και άλλες βιβλιοθήκες.

Το framework προορίζεται να χρησιμοποιηθεί από τις περισσότερες νέες εφαρμογές που έχουν δημιουργηθελι για την πλατφόρα τον Windows. Η Microsoft δημιούργησε ένα ολοκληρωμένο περιβάλλον ανάπτυξης για μεγαλό μέρος του .NET software που ονομάζεται Visual Studio. Το .NET Framework οδήγησε στην δημιουργία μια οικογένιας .NET platforms με στόχο το mobile computing, ενσωματωμένες συσκευές,

εναλλακτικά λειτουργικά στυστήματα π.χ. (Android, iOS) και plug-ins για web browsers π.χ (Chrome, Firefox).

Στην παρακάτω ενότητα θα γίνει η επεξήγηση ορισμένων σημαντικών όρων όπου η απόδοση τους δεν μπόρει να γίνει με μία λέξη ή φράση στα ελληνικά.

1.6 Επεξήγηση συμαντικών όρων

- **ADO.NET** είναι μια τεχνολογία πρόσβασης δεδομένων από το Microsoft .NET framework που παρέχει επικοινωνία μεταξύ σχεσιακών και μη σχεσιακών συστημάτων μέσω ενός κοινού συνόλου στοιχείων. Το ADO.NET είναι ένα σύνολο στοιχείων λογισμικού που μπορούν να χρησιμοποιήσουν οι προγραμματιστές για την πρόσβαση σε δεδομένα και υπηρεσίες δεδομένων από μια βάση δεδομένων. [6]
- Τα **anonymous types** εμφανίζονται στην C#. Επιτρέπουν στους τύπους δεδομένων να ενσωματώνουν ένα σύνολο ιδιοτήτων σε ένα μεμονωμένο αντικείμενο χωρίς να χρειάζεται πρώτα να καθορίσουν ρητά έναν τύπο. Αυτό είναι ένα σημαντικό χαρακτηριστικό για την LINQ και είναι ενσωματωμένο στην C#. Δεδομένου ότι τα anonymous types δεν έχουν ονομασμένο τύπο πρέπει να αποθηκεύονται σε μεταβλητές που δηλώνονται χρησιμοποιώντας την λέξη κλειδί **“var”**,όπου λένε στον compiler της C# να χρησιμοποιήσει το type inference για τη μεταβλητή. [7]
- **Boilerplate code ή Boilerplate** στον προγραμματισμό αναφέρονται σε τμήματα του κώδικα που πρέπει να συμπεριληφθούν σε πολλά σημεία με ελάχιστη ή καθόλου αλλαγή. Συχνά χρησιμοποιείται όταν γίνεται αναφορά σε γλώσσες που θεωρούνται λεπτομερείς, δηλαδή ο προγραμματιστής πρέπει να γράψει πολύ κώδικα για να κάνει ελάχιστες εργασίες. [8]
- **Common Language Infrastructure (CLI Languages)** είναι μία ανοικτή προδιαγραφή, ένα τεχνικό πρότυπο (technical standard) που αναπτύχθηκε από την Microsoft και τυποποιήθηκε από την ISO και την ECMA που περιγράφει τον εκτελέσιμο κώδικα και το runtime περιβάλλον που επιτρέπει πολλαπλές high level γλώσσες προγραμματισμού να χρησιμοποιηθούν σε διαφορετικές πλατφόρμες υπολογιστών χωρίς να ξαναγραφούν για συγκεκριμένες αρχιτεκτονικές. [9]
- Τα **closures** στις γλώσσες προγραμματισμού είναι τεχνικές για την εφαρμογή της λεξικολογικής δέσμευσης ονόματος σε γλώσσες με first-class functions.

Λειτουργικά ένα closure είναι ένα αρχείο όπου αποθηκεύει μία function μαζί με ένα περιβάλλον, μία χαρτογράφηση που συνδέει κάθε ελεύθερη μεταβλητή της function με την τιμή ή την αναφορά στην οποία το όνομα είναι δεσμευμένο όταν το closure δημιουργήθηκε.

- **Declarative programming** (δηλωτικός προγραμματισμός) στην επιστήμη των υπολογιστών είναι ένα στυλ οικοδόμησης της δομής και των στοιχείων των προγραμμάτων των προγραμμάτων που εκφράζει τη λογική ενός υπολογισμού χωρίς να περιγράφει τη ροή ελέγχου. [10]
- Το **extension method** ή αλλιώς μέθοδος επέκτασης είναι μία μέθοδος που προστέθηκε σε ένα object αφότου το πρωτότυπο object υπέστη compile. Το τροποποιημένο object είναι σύχνα ένα πρωτότυπο ή ένας τύπος ή ένα class. Οι μέθοδοι επέκτασης είναι χαρακτηριστικά ορισμένων object-oriented γλωσσών προγραμματισμού. [11]
- **Functional Programming** (λειτουργικός προγραμματισμός) είναι ένα στυλ οικοδόμησης της δομής και των στοιχείων των προγραμμάτων που αντιμετωπίζει τον υπολογισμό ως αξιολόγηση των μαθηματικών λειτουργιών και αποφεύγει να αλλάξει κατάσταση και τα μεταβαλλόμενα δεδομένα. Είναι ένας τύπος declarative programming, πράγμα που σημαίνει ότι ο προγραμματισμός γίνεται με εκφράσεις ή declarations αντί για statements(δηλώσεις). [12]
- **Imperative Programming** (επιτακτικός προγραμματισμός) στην επιστήμη των υπολογιστών είναι ένα παράδειγμα προγραμματισμού που χρησιμοποιεί δηλώσεις που αλλάζουν την κατάσταση ενός προγράμματος. [13]
- Το **generic programming** είναι ένας στυλ προγραμματισμού στο οποίο γράφονται αλγόριθμοι με όρους τύπων που πρέπει να προσδιοριστούν αργότερα,οι οποίοι στην συνέχεια δημιουργούνται όταν απαιτείται για συγκεκριμένους τύπους που παρέχονται ως παράμετροι. [14]
- Οι επεκτάσεις **LINQ (Language Integrated Query)** είναι ένα εργαλείο του .NET Framework της Microsoft προσθέτει δυνατότητες αναζήτησης δεδομένων σε γλώσσες .NET. [15]
- Το **multi-paradigm** (πολλαπλών πραδειγμάτων) υπάρχει στις περισσότερες γλώσσες προγραμματισμού, υποστηρίζουν περισσότερα από ένα πρότυπα προγραμματισμού για να επιτρέψουν τους προγραμματιστές να χρησιμοποιήσουν

το πιο κατάλληλο στυλ προγραμματισμού και τις σχετικές γλωσσικές κατασκευές για μια δεδομένη εργασία. [16]

- **Object-oriented programming (OOP)** είναι ένα παράδειγμα προγραμματισμού που βασίζεται στην έννοια των objects (αντικειμένων), τα οποία μπορεί να περιέχουν δεδομένα με τη μορφή πεδίων, συχνά γνωστά ως ιδιότητες και τον κώδικα με την μορφή διαδικασιών συχνά γνωστά ως μέθοδοι. [17]
- Το **Reification** είναι η διαδικασία με την οποία μια αφηρημένη ιδέα για ένα πρόγραμμα υπολογιστή μετατρέπεται σε ένα σαφές μοντέλο δεδομένων ή σε ένα άλλο αντικείμενο που δημιουργείται σε μια γλώσσα προγραμματισμού. Το reification είναι μία από της ποιο συχνά χρησιμοποιούμενες τεχνικές εννοιολογικής ανάλυσης και εκπροσώπησης της γνώσης. [18]
- Το **Strong Type** ή αλλιώς **strong typing** (ισχυρή πληκτρολόγηση) αναφέρεται γενικά στην χρήση τύπων προκειμένου να συλλάβε και να αντικαταστήσει τον κώδικα και να εξασφαλίσει την ορθότητά του και σίγουρα να αποκλείσει ορισμένες κατηγορίες σφαλμάτων προγραμματισμού. Για αυτό υπάρχουν πολλοί κλάδοι strong typing που χρησιμοποιούντια για την επίτευξη αυτών τον στόχων. [19]
- Το **type inference** αναφέρεται στην αυτόματη ανίχνευση του τύπου δεδομένων μιας έκφρασης σε μια γλώσσα προγραμματισμού. [20]

Κεφάλαιο 2 Unity Game Engine

Εικόνα 2-1 Unity Technologies Logo https://en.wikipedia.org/wiki/File:Unity_Technologies_Logo.svg

Σε αυτό το κεφάλαιο θα παρουσιάσουμε την μηχανή παιχνιδιών Unity, τις δυνατότητες που δίνει στον προγραμματιστή και την ιστορία της. Το περιβάλλον για την δημιουργία ενός παιχνιδιού καθώς και τα εργαλεία μέσα στο λογισμικό της θα εξηγηθούν παρακάτω στο ίδιο κεφάλαιο.

2.1 Τι είναι το Unity Game Engine

Το Unity Game Engine είναι μια μηχανή παιχνιδιών πολλαπλών πλατφορμών (cross platform) που αναπτύχθηκε απο την Unity Technologies, η οποία χρησιμοποιείται κυρίως για την ανάπτυξη τρισδιάστατων και δισδιάστατων βιντεοπαιχνιδιών και προσομοιώσεων για υπολογιστές, κονσόλες και κινητές συσκευές. Η πρώτη της ανακοίνωση έγινε στο Παγκόσμιο Συνέδριο Προγραμματιστών της Apple το 2005 και θα ήταν μόνο για το OS X. Από τότε έχουν κυκλοφορήσει έξι μεγάλες εκδόσεις της Unity και έχει επεκταθεί σε 29 πλατφόρφες κάποιες από αυτές είναι iOS, Android, Windows, Linux/Steam OS, PS4, Xbox One και πολλές ακόμα. Όπως αναφέρθηκε παραπάνω το Unity είναι μία μηχανή παιχνιδιών πολλαπλών χρήσεων που υποστηρίζει γραφικά δισδιάστατα και τρισδιάστατα, λειτουργικότητα drag and drop και scripting στην γλώσσα C#. Δύο άλλες γλώσσες προγραμματσμού. Επίσης υποστηρίχθηκαν η Boo (καταργήθηκε με την κυκλοφορία της Unity 5) και η JavaScript. Για την τελευταία η διαδικασία απόρριψης της άρχισε τον Αύγουστο του 2017 μετά την κυκλοφορία του Unity 2017.1. [21]

2.2 Game Engine

Στην ενότητα που ακολουθεί θα γίνει αναφορά στο τι είναι μία μηχανή παιχνιδιού (Game Engine) καθώς και τι μας προσφέρει η Unity Engine.

2.2.1 Τι είναι μία μηχανή παιχνιδιού (Game Engine)

Μία μηχανή παιχνιδιού είναι ένα λογισμικό πλαίσιο (framework) σχεδιασμένο για την δημιουργία και την ανάπτυξη των βιντεοπαιχνιδιών. Οι προγραμματιστές τις χρησιμοποιούν για να δημιουργήσουν παιχνίδια για κονσόλες, κινητές συσκευές και υπολογιστές. Η βασική λειτουργικότητα που παρέχεται συνήθως από μια μηχανή παιχνιδιών περιλαμβάνει μία rendering μήχανη (renderer) για τα δισδιάστατα ή τρισδιάστατα γραφικά, μία μηχανή φυσικής ή ανίχνευση σύγκρουσης και απόκριση σύγκρουσης, ήχο, scripting, κινούμενα σχέδια (animations), τεχνητή νοημοσύνη, δικτύωση, streaming, διαχείριση μνήμης, threading, υποστήριξη εντοπισμού, scene graph ενώ μπορεί να περιλαμβάνει ακόμα και υποστήριξη βίντεο για κινηματογραφική προβολή. [22]

2.2.2 Τι προσφέρει η Unity Game Engine

Η Unity για τα δισδιάστατα παιχνίδια επιτρέπει την εισαγωγή των sprites και ενός προηγμένου δυσδιάστατου world renderer. Για τα τρισδιάστατα παιχνίδια επιτρέπει προδιαγραφές texture compression, mipmaps και ρυθμίσεων ανάλυσης για κάθε πλατφόρμα που υποστηρίζει η μηχανή παιχνιδιών και παρέχει υποστήριξη για bump mapping, reflection mapping, parallax mapping, screen space ambient occlusion (SSAO), dynamic shadows χρησιμοποιώντας shadow maps, render to texture και full screen post-processing effects. Η Unity προσφέρει επίσης υπηρεσίες σε προγραμματιστές όπως Unity Ads, Unity Analytics, Unity Certification, Unity Cloud Build, Unity Everyplay, Unity IAP, Unity Multiplayer, Unity Performance Reporting and unity Collaborate. [21]

2.2.3 Σκοπώς των μηχανών παιχνιδιών

Σε πολλές περιπτώσεις οι μηχανές παιχνιδιών παρέχουν μια σειρά εργαλείων οπτικής ανάπτυξης εκτός από τα επαναχρησιμοποιήσιμα στοιχεία λογισμικού. Αυτά τα εργαλεία παρέχονται γενικά σε ένα ολοκληρωμένο περιβάλλον που επιτρέπει την απλοποιημένη και ταχεία ανάπτυξη των παιχνιδιών με τρόπο που βασίζεται στην πληροφορία. Οι προγραμματιστές μηχανών παιχνιδιών αναπτύσσουν ισχυρές σουίτες λογισμικού (software suites) που περιλαμβάνουν πολλά στοιχεία που μπορεί να χρειαστεί ένας προγραμματιστής παιχνιδιών για να δημιουργήσει ένα παιχνίδι χωρίς "να *επανεφεύρει τον τροχό*". Οι περισσότερες σουίτες μηχανών παιχνιδιών παρέχουν

εγκαταστάσεις που διευκολύνουν την ανάπτυξη, όπως γραφικά, ήχο, φυσική και λειτουργίες AI. [22]

2.3 Τι είναι το Cross-platform

Στο computing, το λογισμικό πολλαπλών πλατφορμών (Cross-Platform ή multi-platform software ή platform-independent software) είναι ένα λογισμικό υπολογιστή που υλοποιείται σε πολλαπλές πλατφόρες υπολογιστών. Το λογισμικό πολλαπλών πλατφορμών μπορει να χωριστεί σε δύο μέρη. Το ένα μέρος αναπτύσσει τον κώδικα και τον μεταφράζει σε μια εφαρμογή που υποστηρίζεται από το άλλο μέρος συνήθως σε μεγαλύτερο αριθμό πλατφορμών. Για παράδειγμα μια εφαρμογή πολλαπλών πλατφορμών μπορεί να εκτελεστεί σε Microsoft Windows στην αρχιτεκτονική x86, σε Linux στην αρχιτεκτονική x86, και σε macOS στα συστήματα PowerPC ή x86-based Apple Macintosh systems. Τα προγράμματα πολλαπλών πλατφορμών μπορούν να λειτουργούν σε όλες τις υπάρχουσες πλατφόρμες ή σε δύο μόνο πλατφόρμες.

Ένα λογισμικό πολλαπλών πλατφόρμων είναι ένας όρος που μπορεί να εφαρμοστεί και σε βιντεοπαιχνίδια που κυκλοφορούν σε μια σειρά από κονσόλες βιντεοπαιχνιδιών, εξειδικευμένους υπολογιστές αφιερωμένους στο έργο της αναπαραγωγής παιχνιδιών. Ένα παράδειγμα είναι το βιντεοπαιχνίδι FIFA που έχει κυκλοφορήσει σε διάφορες πλατφόρμες παιχνιδιών όπως: PS4, Xbox One, υπολογιστές και κινητές συσκευές. [23]

2.4 Application Programming Interface (API)

Στον προγραμματισμό υπολογιστών τα Application Programming Interface (API) είναι ένα σύνολο ορισμών υπορουτίνας (subroutines definitions), πρωτοκόλλων και εργαλείων για την κατασκευή εφαρμογών λογισμικού (software). [24]

2.4.1 Τι είναι τα API

Σε γενικές γραμμές πρόκειτε για ένα σύνολο καθορισμένων μεθόδων επικοινωνίας μεταξύ διαφόρων στοιχείων λογισμικού. Ένα καλό API διευκολύνει την ανάπτυξη ενός προγράμματος υπολογιστή παρέχοντας όλα τα δομικά στοιχεία τα οποία στην συνέχεια συνδέονται από τον προγραμματιστή. Ένα API μπορεί να είναι για ένα web-based σύστημα, λειτουργικό σύστημα, σύστημα βάσης δεδομένων, computer hardware ή

βιβλιοθήκη λογισμικού. Τα API συνήθως σχετίζονται με μια βιβλιοθήκη λογισμικού. Το API περιγράφει και ορίζει την αναμενόμενη συμπεριφορά(μια προδιαγραφή) ενώ η βιβλιοθήκη είναι μία πραγματική εφαρμογή αυτού του συνόλου κανόνων.
Η μηχανή της Unity συνδέεται με τα ακόλουθα γραφικά API (Application Programming Interface): Direct3D σε Windows και Xbox One, OpenGL σε Linux, macOS και Windows, OpenGL ES σε Android και iOS, WebGL για το web και για ιδιόκτητα API στις κονσόλες βιντεοπαιχνιδιών.

2.4.2 Σκοπός των API

Ακριβώς όπως ένα γραφικό περιβάλλον διευκολύνει τους ανθρώπους να χρησιμοποιούν προγράμματα, τα API διευκολύνουν τους προγραμματιστές να χρησιμοποιούν συγκεκριμένες τεχνολογίες για την δημιουργία εφαρμογών. Με την μεθόδο της αφαίρεσης (abstraction),όπου είναι μια τεχνική στον προγραμματισμό για την διευθέτηση της πολυπλοκότητας των συστημάτων υπολογιστών [25]. Ο προγραμματιστής λειτουργεί με μια εξιδανικευμένη διασύνδεση (συνήθως καθορισμένη) και μπορεί να προσθέσει επιπλέον επίπεδα λειτουργικότητας που διαφορετικά θα ήταν πολύ περίπλοκο για να τα χειριστεί. Εκθέτοντας μόνο objects ή δράσεις που χρειάζεται ο προγραμματιστής, ένα API απλοποιεί τον προγραμματισμό. [24]

2.4.3 Επεξήγηση των API που στοχεύχει η Unity

- **Direct3D** είναι ένα API για τα Microsoft Windows.Το Direct3D χρησιμοποιεί hardware acceleration έαν είναι διαθέσιμο στην κάρτα γραφικών, επιτρέποντας το hardware acceleration ολόκληρου του αγωγού τρισδιάστατης απεικόνισης ή ακόμα και μόνο μερική επιτάχυνση. Το Direct3D εκθέτει τις προηγμένες δυνατότητες γραφικών του τρισδιάστατου hardware γραφικών. Η Microsoft προσπαθεί να ενημερώνει συνεχώς το Direct3D για την υποστήριξη της ποιο σύγχρονης τεχνολογίας που είναι διαθέσιμη σε τρισδιάστατες κάρτες γραφικών. [26]
- **Open graphics Library (OpenGL)** είναι μία cross-language, cross-platform API για rendering σε δισδιάστατα και τρισδιάστατα διανυσματικά γραφικά. Το API συνήθως χρησιμοποιείται για να αλληλεπιδράσει με μία μονάδα επεξεργασίας γραφικών (GPU), για να πετύχει hardware accelerated rendering. Η προδιαγραφή OpenGL περιγράφει ένα αφηρημένο API για την σχεδίαση δισδιάστατων και τρισδιάστατων γραφικών. Παρόλο που είναι δυνατό το API να υλοποιηθεί εξ ολοκλήρου στο

λογισμικό, έχει σχεδιαστεί για να υλοποιείται ως επί το πλείστον ή εξ ολοκλήρου στο hardware. [27]

- **OpenGL for Embedded systems (OpenGL ES)** είναι ένα υποσύνολο της OpenGL απεικόνισης γραφικών υπολογιστών για rendering δισδιάστατων και τρισδιάστατων γραφικών όπως αυτα που χρησιμοποιούνται από τα βιντεοπαιχνίδια, συνήθως η επιτάχυνση hardware χρησιμοποιεί μία μονάδα επεξεργασίας γραφικών (GPU). Είναι σχεδισμένο για ενσωματωμένα συστήματα, όπως smartphones, tablet, κονσόλες βιντεοπαιχνιδιών. [28]
- **Web Graphics Library (WebGL)** είναι ένα API JavaScript για rendering διαδραστικών δισδιάστατων και τρισδιάστατων γραφικών σε οποιοδήποτε συμβατό πρόγραμμα περιήγησης web browser χωρίς τη χρήση plug-ins. Το WebGL ενσωματώνεται πλήρως σε όλα τα πρότυπα web του web browser, επιτρέποντας την επιτάχυνση της GPU με την χρήση της φυσικής και της επεξεργασίας εικόνας και των αποτελεσμάτων ως μέρος του καμβά της ιστοσελίδας. [29]

Στην συνέχεια αφού έγινε η εξήγηση για το τι είναι η Unity Game Engine, τις δυνατότητες και τι μπορεί να προσφέρει στον προγραμματιστή θα αναφερθούμε στην ιστορία της Unity.

2.5 Ιστορία της Unity και η ανάπτυξη της

Όπως αναφέρθηκε παραπάνω η πρώτη ανακοίνωση έγινε το 2005 στο παγκόσμιο συνέδριο προγραμματιστών της Apple. Στο εμπορικό σόου της Apple WWDC (Worldwide Developers Conference) του 2006, η Apple Inc βράβευσε την Unity με το δεύτερο βραβείο Apple Design στην κατηγορία καλύτερης χρήσης γραφικών του λειτουργικού Mac OS X, ένα μόλις χρόνο μέτα την κυκλοφορία της Unity στην ίδια εμπορική έκθεση. Η Unity Technologies λέει ότι είναι η πρώτη φορά που έχει επιλεγεί ένα εργαλείο σχεδιασμού παιχνιδιών για αυτό το βραβείο. Μια έρευνα το Μάιο του 2012 από το περιοδικό Game Developer ανέδειξε την Unity ως την κορυφαία μηχανή παιχνιδιών για κινητές πλατφόρμες. Τον Ιούλιο του 2014 η Unity κέρδισε το βραβείο “Best Engine” στο Ηνωμένο Βασίλειο στα ετήσια βραβεία Develop Industry Excellence Awards. [21]

2.5.1 Licensing

Μέτα την κυκλοφορία της Unity 5, η Unity Technologies έγινε δέκτης από κάποιες αρνητικές κριτικές για το μεγάλο όγκο των κακοσχεδιασμένων παιχνιδιών που κυκλοφόρησαν γρήγορα στην πλατφόρμα διανομής Steam από άπειρους προγραμματιστές. Αύτο προκλήθηκε περισσότερο εξαιτίας μια μεγαλής αλλαγής όπου η έκδοση της μηχανή παιχνιδιών έκλεινε το χάσμα ανάμεσα σε ένα μεγάλο studio και έναν personal user. Η μηχανή ερχόταν με δύο εκδοχές την pro και την personal. Το personal license να περιορίζεται σε άτομα και στούντιο λιγότερα με λιγότερα από 100.000 δολάρια έσοδα ή χρηματοδότηση. [21]

Τον Δεκέμβριο του 2016, η Unity Technologies ανακοίνωσε ότι θα αλλάξει το σύστημα αρίθμησης εκδόσεων για την Unity από τα αναγνωριστικά με βάση την ακολουθία έως το έτος έκδοσης για να ευθυγραμμίσει την έκδοση με τον πιο συχνό ρυθμό απελευθέρωσης τους.

Αυτή την στιγμή υπάρχουν τέσσερα licenses για το πως μπορεί κανείς να χρησιμοποιεί το Unity Engine για νόμιμη χρήση.

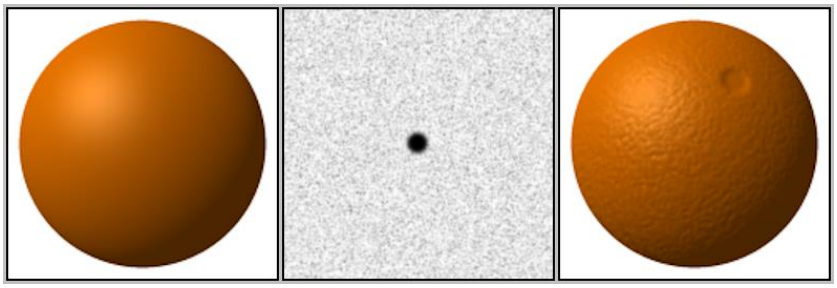
License Name	All Engine Features and platforms	Multiplayer	Revenue Capacity	Premium Support	Access to Source Code	Price
Personal	Yes	20 CCUs	100,000$	No	No	Free
Plus	Yes	50 CCUs	200.000$	No	No	35$/monthly
Pro	Yes	200 CCUs	Unlimited	Yes	No	125$/monthly
Enterprise	Yes	Custom	Unlimited	Yes	Yes	Negotiated Pricing

Πίνακας 2.1 Οι διαφορετικές licenses και τα κριτήρια τους για το Unity.

Τέλος όπως και στο προηγούμενο κεφάλαιο θα γίνει η επεξήγηση ορισμένων συμαντικών όρων, όπου η απόδοση τους δεν είναι εφικτή με μία λέξη ή φράση στα ελληνικά.

2.6 Επεξήγηση συμαντικών όρων

- **Aliasing** είναι η περιγραφή μιας κατάστασης στην οποία μια θέση δεδομένων στην μνήμη μπορούμε να έχουμε πρόσβαση μέσω διαφορετικών συμβολικών ονομάτων στο πρόγραμμα. Έτσι, η τροποποίηση των δεδομένων μέσω ενός ονόματος τροποποιεί σιωπηρά τις τιμές που σχετίζονται όλα τα aliased ονόματα, τα οποία μπορεί να μην αναμένονται από τον προγραμματιστή. Ως αποτέλεσμα το aliasing καθίστα ιδιαίτερα δύσκολη την κατανόηση, την ανάλυση και τη βελτιστοποίηση των προγραμμάτων. [30]
- **Ambient occlusion** είναι μια τεχνική σκίασης και rendering που χρησιμοποιείται για τον υπολογισμό του τρόπου με τον οποίο εκτίθεται κάθε σημείο σε μια σκηνή στον περιβάλλοντα φωτισμό. [31]
- **Bump Mapping** είναι μια τεχνική στα γραφικά του υπολογιστή για την προσομοίωση εξογκωμάτων και ρυτίδων στην επιφάνεια ενός αντικειμένου. Αυτό επιτυγχάνεται με την διατάραξη των κανονικών επιφανειών του αντικειμένου και τη χρήση του διαταραγμένου κανονικού κατά τη διάρκεια των υπολογισμών φωτισμού. Το αποτέλεσμα είναι μια προφανώς ανώμαλη επιφάνεια και όχι μια λεία επιφάνεια, αν και η επιφάνεια του αντικειμένουν δεν αλλάζει. [32]



Εικόνα 2.2 Η σφαίρα στα αριστερά είναι χωρίς bump mapping και η σφαίρα στα δεξιά με bump mapping https://en.wikipedia.org/wiki/File:Bump-map-demo-full.png

- **CCU – concurrent users** είναι ο αριθμός των ταυτόχρονων χρηστών για έναν πόρο σε μια τοποθεσία, όπου η τοποθεσία είναι ένα δίκτυο υπολογιστών ή ένας μόνο υπολογιστής, αναφέρεται στον συνολικό αριθμό των ατόμων που χρησιμοποιούν τον πόρο μέσα σε μια προκαθορισμένη χρονική περίοδο. Ο πόρος μπορεί για παράδειγμα να είναι ένα πρόγραμμα υπολογιστή, ένα αρχείο ή ο υπολογιστής στο σύνολο του. [33]
- **Mipmaps** στα γραφικά υπολογιστή είναι προ-υπολογισμένες βελτιστοποιημένες ακολουθίες εικόνων, όπου κάθε μία από τις οποίες είναι μια προοδευτικά χαμηλότερης ανάλυσης αναπαράσταση της ίδιας εικόνας. Σκοπός τους είναι να αυξήσουν την ταχύτατα εμφάνισης και να μειώσουν τα aliasing artifacts. [34]

- **Parallax mapping** είναι μια βελτίωση της bump mapping ή τον κανονικών τεχνικών mapping που εφαρμόζονται στις υφές σε τρισδιάστατα rendering όπως τα βιντεοπαιχνίδια. Για τον τελικό χρήστη, αυτό σημαίνει ότι οι υφές όπως οι τοίχοι από πέτρα θα έχουν περισσότερο εμφανές βάθος και επομένως μεγαλύτερο ρεαλισμό με λιγότερη επιρροή στην απόδοση της προσομοίωσης. [35]
- **Plug-in (ή plugin, add-in, add-on, addon, extension)** είναι ένα στοιχείο λογισμικού που προσθέτει μια συγκεκριμένη λειτουργία σε ένα υπάρχον πρόγραμμα υπολογιστή. Όταν ένα πρόγραμμα υποστηρίζει plug-ins, επιτρέπει το customization. Τα συνηθισμένα παραδείγματα είναι οι προσθήκες που χρησιμοποιούνται στα web browsers για την προσθήκη νέων λειτουργιών, όπως μηχανών αναζήτηση, σαρωτές ιών ή την δυνατότητα χρήσης ενός νέου τύπου αρχείου, όπως ενός νέου τύπου βίντεο. [36]
- **Reflection mapping** είναι μία αποτελεσματική τεχνική φωτισμού με βάση την εικόνα για τν προσέγγιση της εμφάνισης μιας αντανακλαστικής επιφάνειας με τη βοήθεια μιας προ-συσσωρευμένης εικόνας υφής. Η υφή χρησιμοποιείται για την αποθήκευση της εικόνας του απομακρυσμένου περιβάλλοντος που περιβάλλει το τεκμηριωμένο αντικείμενο. [37]
- **Rendering** είναι η αυτόματη διαδικασία δημιουργίας μιας φωτορεαλιστικής ή μη φωτορεαλιστικής εικόνας από ένα μοντέλο δισδιάστατο ή τρισδιάστατο (ή μοντέλα που συλλογικά θα μπορούσαν να ονομαστούν scene file) μέσω προγραμμάτων ηλεκτρονικών υπολογιστών. Επίσης τα αποτελέσματα της εμφάνισης ενός τέτοιου μοντέλου μπορούν να ονομαστούν render. [38]
- **Scene File** είναι ένα αρχείο σκηνής όπου περιέχει αντικείμενα με αυστηρά καθορισμένη δομή δεδομένων ή γλώσσας. Θα περιέχει γεωμετρία, υφή, φωτισμό, viewpoint και πληροφορίες σκίασης ως περιγραφή της εικονικής σκηνής. Τα δεδομένα που περιέχονται στο αρχείο σκηνής διαβιβάζονται στην συνέχεια σε ένα πρόγραμμα επεξεργασίας που πρόκειται να επεξεργαστεί και να εξάγεται σε μια ψηφιακή εικόνα ή αρχείο εικόνων.
- **Screen space ambient occlusion (SSAO)** είναι μια τεχνική γραφικών ηλεκτρονικών υπολογιστών για την αποτελεσματική προσέγγιση της φαινόμενης απόφραξης του περιβάλλοντος σε πραγματικό χρόνο. [39]
- **Script** είναι προγράμματα γραμμένα για ένα ειδικό περιβάλλον χρόνου εκτέλεσης (run-time) που αυτοματοποιούν την εκτέλεση εργασιών που θα μπορούσαν εναλλακτικά να εκτελεστούν έας προς ένα από έναν άνθρωπο. [40]

- **Shadow mapping ή Shadow projection** είναι μια διαδικασία με την οποία προστίθενται σκιές σε τρισδιάστατα γραφικά υπολογιστή. Οι σκιές δημιουργούνται ελέγχοντας αν ένα pixel είναι ορατό απο την πηγή φωτός, συγκρίνοντας το pixel με έναν z-buffer ή μία εικόνα βάθους της προβολής της φωτεινής πηγής, που αποθηκεύεται με την μορφή υφής. [41]
- **Texture compression** είναι μια εξειδικευμένη μορφή συμπίεσης εικόνων που έχει σχεδιαστεί για την αποθήκευση texture maps σε συστήματα rendering για τρισδιάστατα γραφικά. Σε αντίθεση με τους συμβατικούς αλγόριθμους συμπίεσης εικόνας, οι αλγόριθμοι συμπίεσης υφής βελτιστοποιούνται για τυχαία πρόσβαση. [42]
- **Texture mapping** είναι μια μέθοδος για τον καθορισμό λεπτομέρειας υψηλής συχνότητας, surface texture ή πληροφορίες χρώματος σε ένα γραφικό ή τρισδιάστατο μοντέλο που παράγεται από υπολογιστή. [43]
- **Z-buffering** επίσης γνωστή ως buffering βάθους, είναι η διαχείριση συντεταγμένων βάθους εικόνας σε τρισδιάστατα γραφικά, που συνήθως πραγματοποιούνται στο hardware και μερικές φορές στο λογισμικό. Πρόκειται για μια λύση στο πρόβλημα της απόφασης για το ποια στοιχεία μιας rendered σκηνής είναι ορατά και ποια είναι κρυμμένα. [44]

Κεφάλαιο 3 Βασικά Εργαλεία για την ανάπτυξη ενός παιχνιδιού στην Unity Engine

Για την ανάπτυξη των βιντεοπαιχνιδιών σε αυτή την εργασία χρησιμοποιήθηκαν δύο βασικά εργαλεία

1. Για την δημιουργία του visual content και της τελικής εφαρμογής χρησιμοποιήθηκε το περιβάλλον της Unity engine (Κεφάλαιο 2).
2. για την δημιουργία των scripts που ελέγχουν την ροή και την αλληλεπίδραση των GameObjects χρησιμοποιήθηκε η C# (Κεφάλαιο 1) σε περιβάλλον Visual Studio.

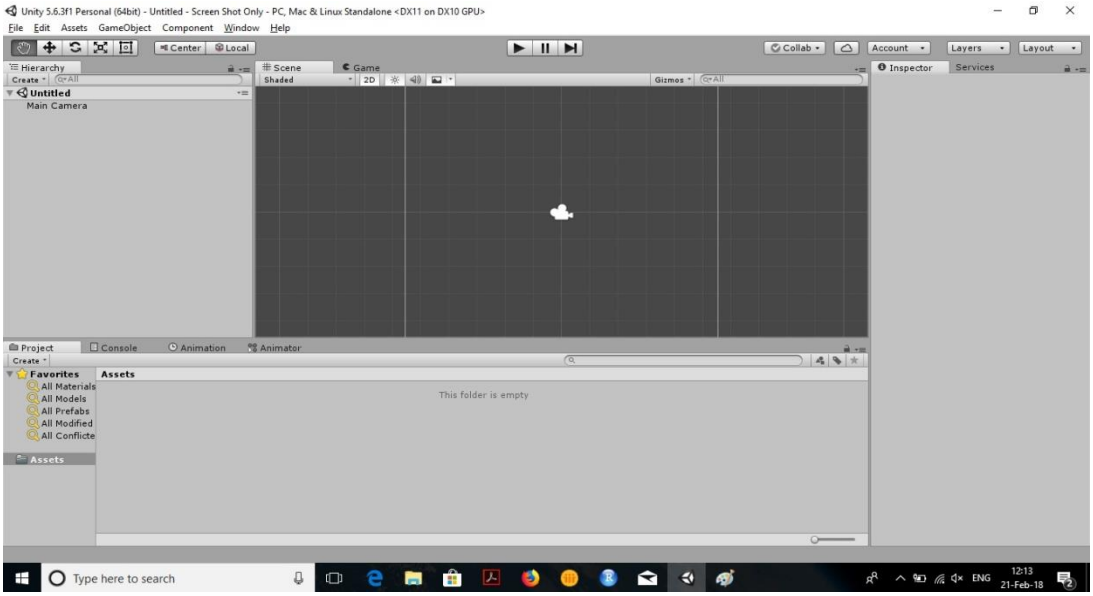
Στο πρώτο μέρος αυτού του κεφάλαιο θα παρουσιαστεί το interface της Unity και θα γίνει η ανάλυση τον βασικών εργαλείων. Στο δεύτερο μέρος του κεφαλαίου θα παρουσιαστούν τα βασικά εργαλεία του Visual Studio.

Πριν συνεχίσουμε την ανάλυση του interface της Unity και τι δυνατότητες μας προσφέρει θα γίνει αναφορά πρώτα στο τι είναι τα GameObjects. Το GameObject είναι η πιο σημαντική ένοια του Unity Editor.

Κάθε αντικείμενο στο παιχνίδι είναι ένα GameObject, από χαρακτήρες και συλλεκτικά αντικείμενα σε φώτα, κάμερες και ειδικά εφέ. Ωστόσο ένα GameObject δεν μπορεί να κάνει τίποτα μόνο του,πρέπει να του δώσουμε ιδιότητες προτού να μπορέσει να γίνει ένας χαρακτήρας, ένα περιβάλλον ή ένα ειδικό αποτέλεσμα. Για να δώσουμε στο GameObject τις ιδιότητες που χρειάζεται για να γίνει ένα φως, ένα δέντρο ή μία κάμερα, πρέπει να του προσθέσουμε components σε αυτό. Ανάλογα με το είδος του αντικειμένου που θέλουμε να δημιουργήσουμε προσθέτουμε διαφορετικούς συνδυασμούς components σε ένα GameObject. [45] Στο **3.6 The Inspector Window** γίνεται περαιτέρω αναφορά στα GameObjects.

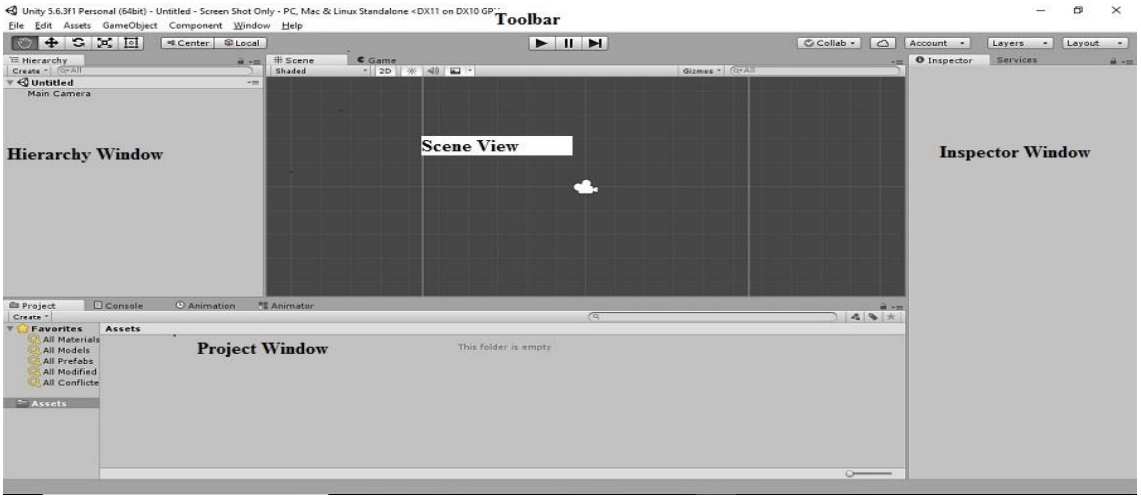
3.1 Interface της Unity

Σε αυτή την εννότητα θα αναλυθεί το Interface της Unity.



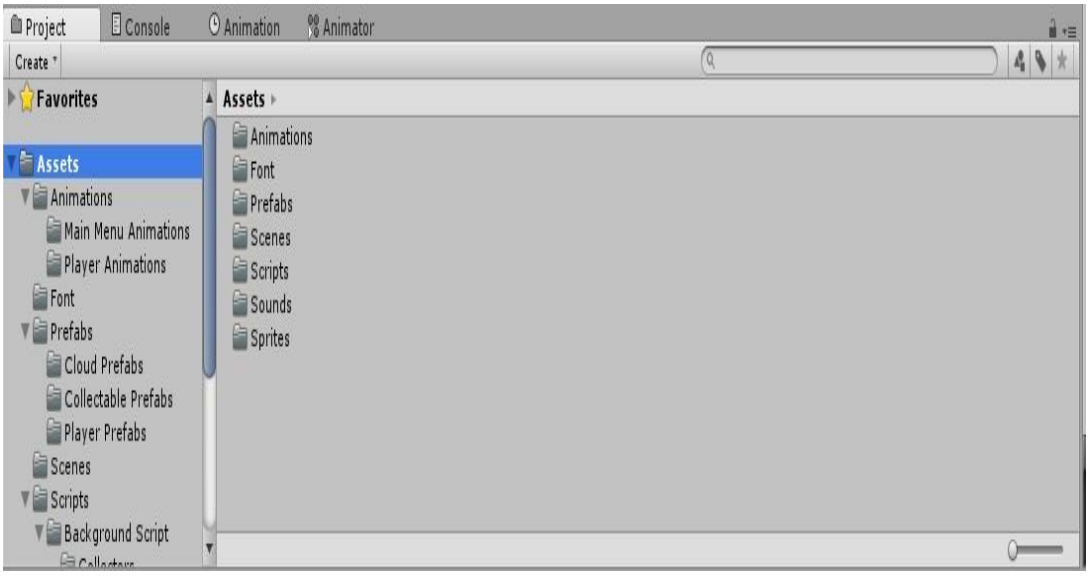
Εικόνα 3-1 Default Interface της Unity έκδοση 5.6.3

Από την εικόνα 3-1 που δείχνει την default κατάσταση της Unity όταν ανοίγουμε ένα καινούριο project μπορούμαι να χωρίσουμε το interface σε 5 βασίκα μέρη όπως μπορούμε να διακρίνουμε απο την εικόνα 3-2: toolbar, hierarchy window, scene view, inspector window, project window. [46]



Εικόνα 3-2 Interface της Unity

3.2 Project Window



Εικόνα 3-3 Project Window

Στο **Project Window**, μπορούμε να αποκτήσουμε πρόσβαση και να διαχειριστούμε τα στοιχεία (assets) που ανήκουν στο project μας. [46] Το αριστερό πλαίσιο εμφανίζει τη δόμη του φακέλου του project ως ιεραχική λίστα. Όταν επιλέγεται ένας φάκελος από τη λίστα, τα περιεχόμενα του θα εμφανιστούν στον πίνακα δεξιά. Τα μεμονωμένα assets εμφανίζονται στο δεξί πλαίσιο ως εικονίδα που υποδεικνύουν τον τύπο τους (script, material, sub folder κ.λπ.). [47]

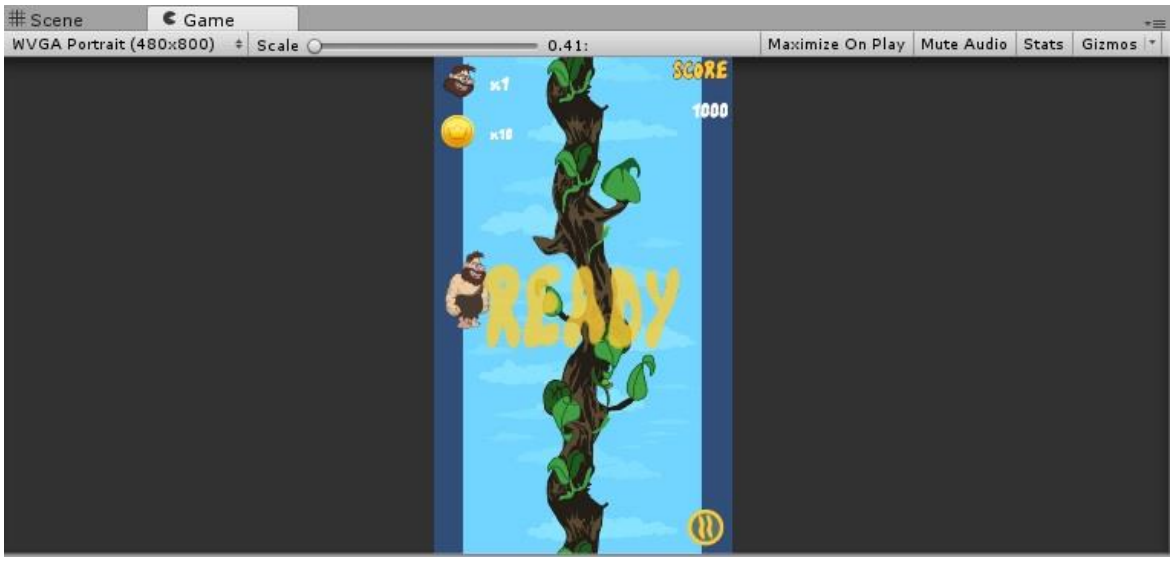
3.3 The Scene View

Εικόνα 3-4 Scene View

Η Scene View είναι η διαδραστική προβολή στον κόσμο που δημιουργούμε. Θα χρησιμοποιήσουμε την Scene View για να επιλέξουμε και να τοποθετήσουμε τοπία, χαρακτήρες, κάμερες, φώτα, και όλους τους άλλους τύπους Game Object. Η δυνατότητα επιλογής, χειρισμού και τροποποίησης αντικειμένων στην προβολή σκηνής είναι μερικές από τις πρώτες δεξιότητες που πρέπει να μάθει κάποιος για να αρχίσει να εργάζεται στην Unity.

Πέρα από το πλαίσιο της Scene View υπάρχει και η γραμμή ελέγχου για την Scene View όπου μας επιτρέπει να επιλέξουμε διάφορες επιλογές για την προβολή της σκηνής και επίσης να ελέγχετε αν είναι ενεργοποιημένος ο φωτισμός και ο ήχος. Αυτά τα στοιχεία ελέγχου επηρεάζουν μόνο την Scene View κατά την ανάπτυξη και δεν επηρεάζουν το ολοκληρωμένο παιχνίδι. [48]

3.4 The Game View



Εικόνα 3-5 Game View

Το Game View είναι rendered από την κάμερα του παιχνιδιού. Είναι αντιπροσωπευτικό του τελικού δημοσιευμένου παιχνιδιού. Θα χρειαστεί να χρησιμοποιηθούν μία ή περισσότερες κάμερες για να ελέγχουμε τι πραγματικά βλέπει ο παίκτης όταν παίζει το παιχνίδι. [49]

Από την εικόνα 3-6 παρατηρούμε ότι οπώς και στην Scene View υπάρχει μία γραμμή ελέγχου για την Game View και θα αναλύσουμε τα ποιο βασικά από αυτά. [49]

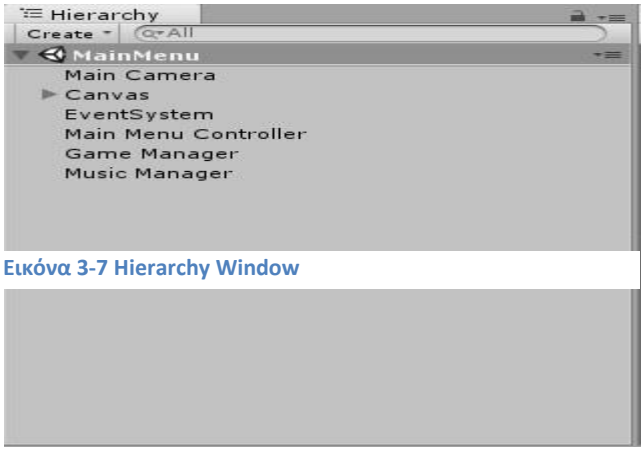
Εικόνα 3-6 Game View Control Bar

a) Το πρώτο κουμπί από τα αριστερά είναι το **Aspect** και επιτρέπει να επιλέξουμε διαφορετικές τιμές για να ελέγξουμε πως θα εμφανίζεται το παιχνίδι σε οθόνες με διαφορετικούς λόγους διαστάσεων. Αυτή η ρύθμιση έχει οριστεί ως Free Aspect απο προεπιλογή.

b) Το δευτέρο κουμπί είναι το **Scale Slider**, κάνοντας scroll προς τα δεξιά κάνουμε μεγέθυνση και μπορούμε να εξετάσουμε λεπτομερέστερα τις περιοχές της Game Screen. Μας επιτρέπει επίσης να κάνουμε σμίκρυνση για να δούμε ολόκληρη την οθόνη όπου η ανάλυση της συσκευής είναι υψηλότερη από το μέγεθος του παραθύρου Game View. Επίσης μπορούμε να χρησιμοποιήσουμε το scroll wheel και το μεσαίο πλήκτρο, για να το κάνουμε αυτό το παιχνίδι θα πρέπει να βρίσκεται σε κατάσταση Paused ή Stopped.

c) Το επόμενο κουμπί είναι το **Maximize on Play** χρησιμοποιούμε αυτό το πλήκτρο για να μεγιστοποιήσουμε το Game View (100% του παραθύρου Editor) για προεπισκόπηση πλήρους οθόνης (full screen) κατά την είσοδο στην λειτουργία Play Mode.

d) Δεξιά απο το Mute Audio έχουμε το **Stats button** κάνοντας click σε αυτή την επιλογή μεταβαίνουμε στην επικάλυψη στατιστικών στοιχείων, η οποία περιέχει Rendering Statistics για τον ήχο και τα γραφικά του παιχνιδιού. Αυτό είναι πολύ χρήσιμο για την παρακολούθηση της απόδοσης του παιχνιδιού ένω βρισκόμαστε στο Play Mode.



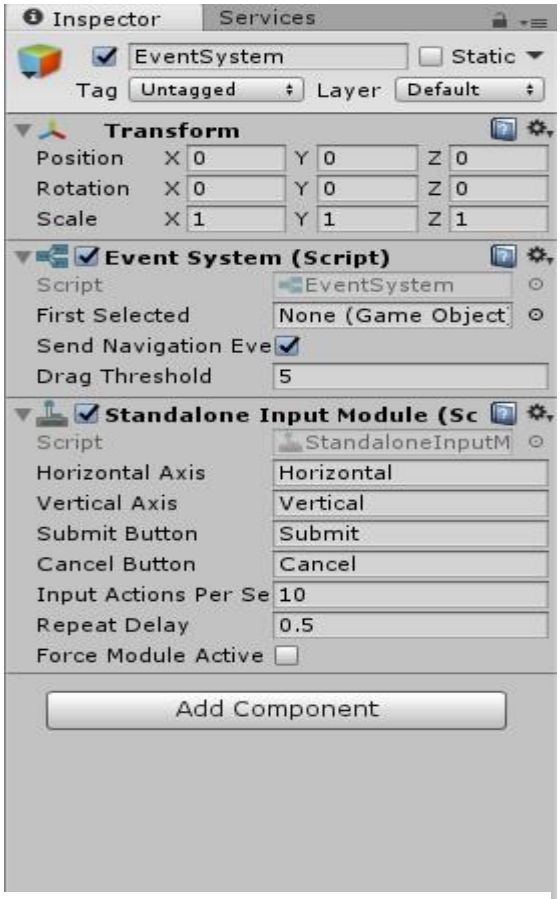
Εικόνα 3-7 Hierarchy Window

3.5 Παράθυρο Ιεραρχίας (Hierarchy Window)

Το παράθυρο Ιεραρχίας περιέχει μια λίστα με κάθε GameObject στην τρέχουσα σκηνή. Ορισμένες από αυτές είναι άμεσες εμφανίσεις αρχείων Asset (οπώς τρισδιάστατα μοντέλα) και άλλα

είναι περιπτώσεις Prefabs, τα οποία είναι προσαρμοσμένα αντικείμενα που αποτελούν το μεγαλύτερο μέρος του παιχνιδιού. Καθώς προστίθενται και αφαιρούνται αντικείμενα στην σκηνή, θα εμφανιστούν και θα εξαφανιστούν και από την Ιεραρχία. Από προεπιλογή, τα αντικείμενα (objects) παρατίθενται στο παράθυρο Ιεραρχίας με την σειρά που έχουν δημιουργηθεί. Μπορούμε να ταξινομήσουμε εκ νέου τα αντικείμενα σύροντας τα προς τα πάνω ή προς τα κάτω ή κάνοντας "child" ή "parent" objects. Η Unity χρησιμοποιεί μια έννοια που ονομάζεται **Parenting.** Όταν δημιουργείται μία ομάδα αντικειμένων, το ανώτατο αντικείμενο ή σκηνή ονομάζεται "parent object" και όλα τα αντικείμενα που ομαδοποιούνται κάτω αυτό ονομάζονται "child objects" ή "**child**". Μπορούμαι επίσης να δημιουργήσουμε nested parent-child objects. Για να κάνουμε οποιαδήποτε αντικείμενο "child" ενός άλλου, μπορούμε να κάνουμε drag and drop το επιδιωκόμενο child αντικείμενο στο επιδιωκόμενο parent αντικείμενο στην Ιεραρχία. [50]

3.6 The Inspector Window or Inspector



Εικόνα 3-8 Inspector Window or Inspector

Τα project του Unity Editor αποτελούνται από πολλαπλά GameObjects που περιέχουν scripts, ήχους, πλέγματα (Meshes) και αλλά γραφικά στοιχεία όπως τα φώτα. Το Inspector Window εμφανίζει λεπτομερείς πληροφορίες σχετικά με τον τρέχον επιλεγμένο GameObject, συμπεριλαμβανόμενων όλων των συνημμένων στοιχείων (components) και των ιδιοτήτων τους και μας επιτρέπει να τροποποιήσουμε την λειτουργικότητα των GameObjects στην σκηνή μας. [51]

Χρησιμοποιώντας τον Inspector για να προβάλουμε και να επεξεργαστούμε τις ιδιότητες και τις ρυθμίσεις σχεδόν όλων των στοιχείων του Unity Editor, συμπεριλαμβανομένων των στοιχείων του παιχνιδιού όπως

τα GameObjects, τα Assets και τα υλικών (Materials), καθώς και τις ρυθμίσεις και τις προιτμήσεις του Editor.

Όταν επιλέγουμε ένα GameObject είτε από την Hierarchy Window ή το Scene View, ο Inspector εμφανίζει τις ιδιότητες όλων των components και των υλικών του συγκεκριμένου GameObject. Χρησιμοποιώντας τον Inspector για να επεξεργαστούμε τις ρυθμίσεις αυτών των components και των υλικών. Στην εικόνα 3-9 έχουμε επιλεγμένο ένα GameObject με το όνομα Player. Όταν τα GameObjects έχουν προσαρμοσμένα custom script components ο Inspector εμφανίζει τις δημόσιες μεταβλητές (public variables) του συγκεκριμένου script. Μπορούμε να επεξεργαστούμε αυτές τις μεταβλητές ως ρυθμίσεις με τον ίδιο τρόπο που μπορούμε να επεξεργαστούμε τις ρυθμίσεις των ενσωματωμένων στοιχείων του Editor. Αυτό σημαίνει ότι μπορρούμε να ορίσουμε εύκολα τις παραμέτρους και τις προεπιλεγμένες τιμές στα scrips χωρίς να τροποποιήσουμε τον κώδικα μας.

Εικόνα 3-9 Inspector Window του GameObject Player

Εικόνα 3-10 Inspector Window for Tags&Layers

Ο Inspector επίσης μας δίνει την δυνατότητα να φτιάξουμε και να ορίσουμε για τα GameObjects μας στο παιχνίδι tags και layers. Το tag είναι μια λέξη αναφοράς που μπορούμε να αντιστοιχίσουμε σε ένα ή περισσότερα GameObjects. Τα tags μας βοηθάνε να εντοπίσουμε GameObjects για scripting σκοπούς.

Εξασφαλίζουν ότι δεν χρειάζεται να προσθέσουμε χειροκίνητα GameObjects στις

εκτεθειμένες ιδιότητες ενός script χρησιμοποιώντας drag and drop, εξοικονομώνας έτσι χρόνο όταν χρησιμοποιείτε ο ίδιος script κώδικας σε πολλά GameObjects. Τα tags είναι επίσης χρήσιμα για ενεργοποιήσεις ελέγχου Collider scripts. Πρέπει να υπολογίσουν αν ο παίκτης αλληλεπιδρά για παράδειγμα με έναν εχθρό ή ένα collectable. [51]

Τα Collider components καθορίζουν το σχήμα ένος αντικειμένου για τους σκοπούς των φυσικών συγκρούσεων. Ένας collider ο οποίος είναι αόρατος δεν χρειάζεται να είναι ακριβώς το ίδιο σχήμα με το πλέγμα του αντικειμένου και στην πραγματικότητα μία τραχιά προσέγγιση είναι συχνά ποιο αποτελεσματική και αδιαίρετη στο gameplay. Με προσεκτική τοποθέτηση και μεγέθυνση σύνθετοι Colliders μπορούν συχνά να προσεγγίσουν το σχήμα ενός αντικειμένου αρκετά καλά διατηρώντας ταυτόχρονα μία χαμηλή χρήση του επεξεργαστή μας. [52]

Με την αναφορά του Inspector Window κλείνει το πρώτο μέρος του κεφαλαίου που ήταν για την παρουσίαση του inteface της Unity. Ο σκοπός της επόμενης ενότητας είναι να γίνει μία αναφορά του προγράμματος που χρησιμοποιήθηκε για να γραφτεί ο κώδικας του παιχνιδιού το Microsoft Visul Studio.

3.7 Microsoft Visual Studio

Το Microsoft Visual Studio είνα ένα ολοκληρωμένο περιβάλλον ανάπτυξης (intergrated development environment) (IDE) από την Microsoft. Χρησιμοποιείται για την ανάπτυξη προγραμμάτων ηλεκτρονικών υπολογιστών καθώ και για web sites, web apps, web services και εφαρμογές κινητών. Μπορεί να παράγει native code και managed code. Το Visual Studio περιλαμβάνει ένα πρόγραμμα επεξεργασίας κώδικα το οποίο υποστηρίζει το IntelliSense καθώς και το code refactoring. Το Visual Studio υποστηρίζει 36 διαφορετικές γλώσσες προγραμματισμού και επιτρέπει στον code editor και debugger να υποστηρίζει σε διαφορετικούς βαθμούς σχεδόν οποιαδήποτε γλώσσα προγραμματισμού, υπό την προϋπόθεση ότι υπάρχει για την συγκεκριμένη γλώσσα υπερσία υποστήριξεις. Οι ενσωματωμένες γλώσσες που περιλαμβάνει είναι C, C++, C++/CLI, Visual Basic .NET, C#, F#, JavaScript, TypeScript, XML, XSLT, HTML και CSS. Όμως υπάρχει υποστήριξη για άλλες γλώσσες προγραμματισμού όπως, Python, Ruby, Node.js είναι κάποιες μεταξύ άλλων, η υποστήριξη τους είναι διαθέσιμη μέσω Plug-ins. Τέλος η πιο βασική και απλή έκδοση του Visual Studio, the Community Edition διατίθεται δωρεάν. [53]

3.8 Βασικά χαρακτηριστικά του Visual Studio

Παρακάτω θα παρουσιαστούν κάποια βασικά χαρακτηριστίκα της Visual Studio. Η ενότητα αυτή δεν έχει σκοπό την πλήρης ανάλυση των δυνατοτήτων της Visual Studio, αλλά την αναφορά και ανάλυση των πιο βασικών χαρακτηριστικών της.

3.8.1 Code Editor

Όπως κάθε άλλο IDE, περιλαμβάνει έναν code editor ο οποίος υποστηρίζει την επισήμανση του συντακτικού και την ολοκλήρωση κώδικα χρησιμοποιώντας το IntelliSense για μεταβλητές, λειτουργίες (functions), μεθόδους, βρόχους και LINQ queries. Το IntelliSense είναι μία εφαρμογή ολοκλήρωσης κώδικα από την Microsoft, περισσότερο γνωστή από την χρήση στο Visual Studio, επίσης υποστηρίζεται για τις γλώσσες που περιλαμβάνονται καθώς και για το XML και για CCS και JavaScript. Οι προτάσεις αυτόματης συμπλήρωσης εμφανίζονται σε ένα πεδίο χωρίς λίστες δίπλα στο παράθυρο του code editor, κοντά στον editing cursor. Από το Visual Studio 2008 και μετά, οι προτάσεις αυτόματης συμπλήρωσης μπορούν να γίνουν προσωρινά ημιδιαφανείς για να βλέπουμε τον κώδικα χωρίς να εμποδίζεται απο τις προτάσεις. [53]

Ένας source code editor είναι ένα πρόγραμμα επεξεργασίας κειμένου που έχει σχεδιαστεί ειδικά για την επεργασία του πηγαίου κώδικα (source code) των προγραμμάτων υπολογιστή από τους προγραμματιστές. Μπορεί να είναι μια αυτόνομη εφαρμογή ή μπορεί να ενσωματτωθεί σε ένα IDE ή σε πρόγραμμα περιήγησης ιστού (web browser). Οι Source code editors είναι το πιο βασικό εργαλείο προγραμματισμού, καθώς η θεμελιώδης δουλειά των προγραμματιστών είναι να γράφουν και να επεξεργάζονται τον πηγαίο κώδικα. [54]

Το Visual Studio διαθέτει background compilation. Καθώς ο κώδικας γράφεται το Visual Studio συνθέτει στο παρασκήνιο για να παρέχει πληροφορίες σχετικά με τα σφάλματα σύνταξης και compilation errors, τα οποία επισημαίνονται με κόκκινη κυματιστή υπογράμμιση. Οι προειδοποιήσεις επισημαίνονται με μια πράσινη υπογράμμιση. Το background compilation **δεν δημιουργεί εκτελέσιμο κώδικα**, καθώς απαιτεί διαφορετικό compiler από αυτόν που χρησιμοποιείται για τη δημιουργία εκτελέσιμου κώδικα. Το background compilation αρχικά εισήχθη με τη Microsoft Visual Basic αλλά έχει πλέον επεκταθεί για όλες τις γλώσσες που περιλαμβάνονται στο Visual Studio. [53]

Η Visual Basic είναι μια γλώσσα προγραμματισμού που βασίζεται σε γεγονότα τρίτης γενιάς (third-generation event-driven) και είναι ένα IDE από την Microsoft για το Component Object Model(COM) το πρώτο μοντέλο προγραμματισμού και κυκλοφόρησε το 1991 και δηλώθηκε ως legacy κατά την διάρκεια του 2008. Στον προγραμματισμό, legacy system είναι μια παλαιά μέθοδος, τεχνολογία, σύστημα ηλεκτρονικού υπολογιστή ή πρόγραμμα. Αυτό μπορεί επίσης να σημαίνει ότι το σύστημα είναι ξεπερασμένο ή χρειάζεται αντικατάσταση. [55] [56]

3.8.2 Debugger

Το Visual Studio περιλαμβάνει ένα πρόγραμμα εντοπισμού σφαλμάτων που λειτουργεί τόσο ως εντοπισμός σφαλμάτων σε source-level όσο και ως εργαλείο εντοπισμού σφαλμάτων σε επίπεδο μηχανής (machine-level). Λειτουργεί τόσο με τον managed code όσο και με τον native code και μπορεί να χρησιμοποιηθεί για την ανίχνευση εφαρμογών γραμμένες σε οποιαδήποτε γλώσσα υποστηρίζεται απο το Visual Studio. Επιπλέον, μπορεί επίσης να επισυνάπτεται σε τρέχουσες διαδικασίες και να παρακολουθεί και να εντοπίζει σφάλματα σε αυτές τις διαδικασίες. Εάν ένας πηγαίος κώδικας είναι διαθέσιμος για running process, εμφανίζει τον κώδικα καθώς εκτελείται. Εάν δεν είναι διαθέσιμος μπορεί να εμφανιστεί η αποσυναρμολόγηση(dissasembly). Το Visual Studio Debugger μπορεί επίσης να δημιουργήσει memory dumps καθώς και να τις φορτώσει αργότερα για debugging.

Ο Debugger επιτρέπει τον καθορισμό σημείων διακοπής(breakpoints) και watches. Τα breakpoints επιτρέπουν την προσωρινή διακοπή της εκτέλεσης του κώδικα σε μια συγκεκριμένη θέση, ενώ τα watches παρακολουθούν τις τιμές των μεταβλητών κατά την εξέλιξη της εκτέλεσης. Τα breakpoint μπορούν να είναι υπό όρους, δηλαδή να ενεργοποιούνται όταν πληρούνται οι προϋποθέσεις. Ο κώδικας μπορεί να μετακινηθεί, δηλαδή να τρέξει μία γραμμή του πηγαίου κώδικα τη φορά. Μπορεί είτε να εισαγάγει λειτουργίες για να εντοπίσει σφάλματα στο εσωτερικό του, είτε να περάσει επάνω του, δηλαδή η εκτέλεση ενός function body δεν είναι διαθέσιμη για χειροκίνητη επιθεώρηση. [53]

3.9 Visual Studio Tools και αλληλεπίδραση με Unity

Το Visual Studio επιτρέπει στους προγραμματιστές να γράφουν επεκτάσεις για επεκτείνουν τις δυνατότητες του. Οι επεκτάσεις έρχονται με τη μορφή μακροεντολών (macros), add-ins, και πακέτων. [53]

- Οι **μακροεντολές** αντιπροσωπεύουν επαναλαμβανόμενα καθήκοντα και δράσεις που οι προγρμαματιστές μπορούν να καταγράψουν προγραμματικά για αποθήκευση αναπαραγωγή και διανομή. Οι μακροεντολές ωστόσο δεν μπορούν να εφαρμόσουν νέες εντολές ή να δημιουργήσουν παράθυρα εραγαλείων.
- Τα **add-ins** παρέχουν πρόσβασηστο μοντέλο αντικειμένου του Visual Studio και μπορούν να αλληλεπιδρούν με τα εργαλεία IDE. Τα Add-ins μπορούν να χρησιμοποιηθούν για την εφαρμογή νέων λειτουργιών και μπορούν να προσθέσουν νέα παράθυρα εργαλείων. Τα πρόσθετα συν΄δεονται στο IDE μέσω COM και μπορούν να δημιουργηθούν σε όλες τις COM-γλώσσες.
- Τα **πακέτα** δημιουργούνται χρησιμοποιώντας το Visual Studio SDK και παρέχουν το υψηλότερο επίπεδο επεκτασιμότητας. Μπορούν να δημιουργήσουν σχεδιαστές και άλλα εργαλεία καθώς και να ενσωματώσουν άλλες γλώσσες προγραμματισμού.

Ενώ ο Unity Editor είναι ιδανικός για να φτιάξουμε ένα παιχνίδι δεν μπορεί να γράψει τον κώδικα μας. Με το Visual Studio Tools για την Unity μπορούμε να χρησιμοποιήσουμε τον editor, τον debugger και τις άλλες δυνατότητες Visual Studio για να δημιουργήσουμε scripts για ένα project για την Unity χρησιμοποιώντας C#. Αλλά τα εργαλεία Visual Studio για την Unity έχουν επίσης βαθιά ενσωμάτωση (deep integration) με τον Unity Editor, ωστέ να ξοδεύετε λιγότερος χρόνος για να κάνουμε απλές εργασίες, να παρέχει βελτιώσεις παραγωγικότητας για την Unity. [57]

Η ενσωμάτωση του Visual Studio με την Unity μας επιτρέπει τα εξής: [58]

- Η μονάδα Unity Project Explorer εμφανίζει ολόκληρο το project της Unity μέσα στο Visual Studio χρησιμοποιώντας την ίδια ιεραρχία που εμφανίζεται στο Unity Editor.
- Η ενσωμάτωση της Unity console εμφανίζει την έξοδο από το παράθυρο console Unity απευθείας στο παράθυρο σφάλματος του Visual Studio.
- Μπορούμε να ξεκινήσουμε το debugging του παιχνιδιού μας από το Visual Studio χωρίς να επιστρέψουμε στην Unity.

Εκτός από την καθιερωμένη παραγωγικότητα του Visua Studio για το γράψιμο κώδικα σε C#, τα εργαλεία για την Unity παρέχουν πρόσθετες δυνατότητες παραγωγικότητας. [58]

- Το χρωματισμό στην σύνταξη για την γλώσσα ShaderLab της Unity μας βοηθά να εντοπίζουμε λάθη στα shaders μας πριν γίνουν σφάλματα.
- Ο οδηγός MonoBehavior μας δίνει την δυνατότητα να περιηγηθούμε σε μία λίστα με τις συμπεριφορές της Unity και να δημιουργήσουμε κώδικα boilerplate για συμπεριφορές που μπορεί να μην γνωρίζουμαι.
- Πρόσβαση σε documents της Unity από το Visual Studio

Τέλος θα ακολουθήσει η ενότητα για την επεξήγηση ορισμένων συμαντικών όρων, όπου η απόδοση τους δεν είναι εφικτή με μία λέξη ή φράση στα ελληνικά.

3.10 Επεξήγηση συμαντικών όρων

- **Component Object Model (COM)** είναι ένα πρότυπο δυαδικής διασύνδεσης για τα λογισμικά components που εισήγαγε η Microsoft to 1993. Χρησιμοποιείται για να επιτρέψει τη δμιουργία αντικειμένων επικοινωνίας μεταξύ διαδικασιών σε ένα ευρύ φάσμα γλωσσών προγραμματισμού. Η ουσία του COM είναι ένας γλωσσικά ουδέτερος τρόπος υλοποίησης αντικειμένων που μπορούν να χρησιμοποιηθούν σε διαφορετικά περιβάλλοντα από εκείνα στα οποία δημιουργήθηκαν. [59]
- **Machine Code or Native Code** είναι ένα σύνολο εντολών που εκτελούνται απευθείας από την CPU. [60]
- **Managed Code** είναι κώδικας προγράμματος που απαιτεί και θα εκτελείται μόνο υπό την διαχείριση μιας εικονικής μηχανής CLR συνήθως του .NET Framework. [61]
- **Memory Dump** στον προγραμματισμό αποτελείται από την καταγεγραμμένη κατάσταση της μνήμης εργασίας ενός προγράμματος υπολογιστή σε μία συγκεκριμένη χρονική στιγμή, γενικά όταν το πρόγραμμα έχει καταρρεύσει(crashed) ή έχει τερματιστεί ασυνήθιστα. [62]
- **Prefabs** Το prefab που μας επιτρέπει να αποθηκεύσουμε ένα GameObject με όλα τα components και τις ιδιότητες του. Το prefab λειτουργεί ως πρότυπο από το οποίο μπορείτε να δημιουργήσετε νέες εμφανίσεις αντικειμένων στην σκηνή. Οποιεσδήποτε επεξεργασίες πραγματοποιούνται σε prefab asset αντανακλώνται άμεσως σε όλες τις παραστάσεις που παράγονται από αυτό, αλλά μπορούμε επίσης να αντικαταστήσουμε τα componets και τις ρυθμίσεις για κάθε διαφορετικό αντίγραφο του GameObject ξεχωριστά. [63]

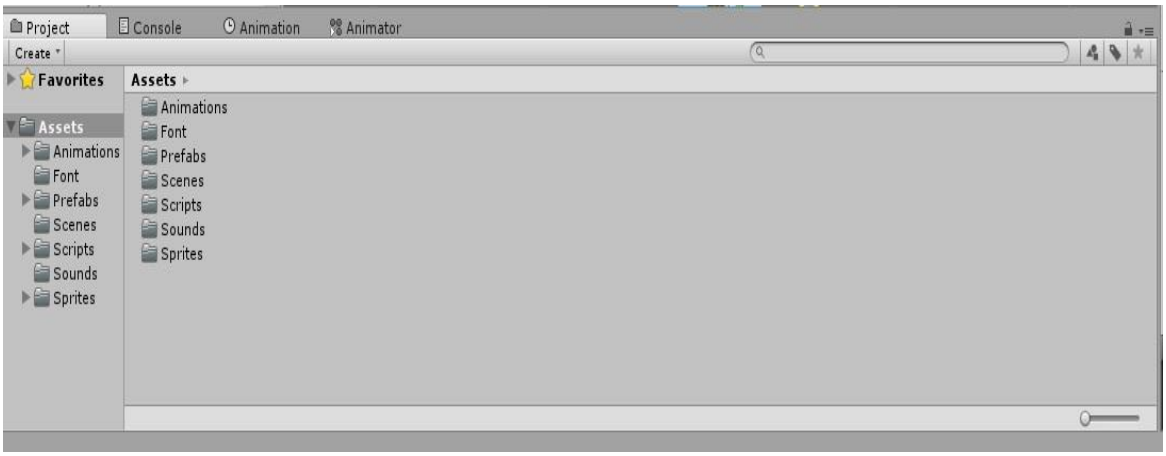
- **Rendering Statistics** Για την Game View ύπαρχει ένα κουμπί stats button όταν πατήσουμε το κουμπί αυτό ένα παράθυρο επικάλυψης εμφανίζεται. Το παράθυρο αυτό εμφανίζει rendering statistics σε πραγματικό χρόνο, τα οποία είναι χρήσιμα για την βελτιστοποίηση της απόδοσης. [64]
- **Software Develepment Kit (SDK)** Ένα SDK ή devkit είναι συνήθως ένα σύνολο εργαλείων ανάπτυξης λογισμικού που επιτρέπει τη δημιουργία εφαργμογών για ένα συγκεκριμένο πακέτο λογισμικού, λογισμικό framework, hardware platform, σύστημα υπολογιστή, κονσόλα βιντεοπαιχνιδιών, λειτουργικό σύστημα ή παρόμοια πλατφόρμα ανάπτυξης. Για να εμπλουτίσουν εφαρμογές προηγμένες λειτουργίες, διαφημίσεις, ειδοποιήσεις push και πολλά άλλα, οι περισσότεροι προγραμματιστές εφαρμογών εφαρμόζουν συγκεκριμένα κιτ ανάπτυξης λογισμικού. [65]
- **XML Documents** στον προγραμματισμό είναι μια γλώσσα σήμανσης που καθορίζει ένα σύνολο κανόνων για την κωδικοποίηση εγγράφων σε μορφή που είναι αναγνώσιμη τόσο από τον άνθρωπο όσο και αναγνώσιμη από μηχανές. [66]

Κεφάλαιο 4 Δημιουργία Βιντεοπαιχνιδιών στην Unity

Σε αυτό το κεφάλαιο θα αναλυθούν οι κώδικες των βιντεοπαιχνιδιών που δημιουργήθηκαν στην διάρκεια αυτής της πτυχιακής εργασίας με την χρήση της Unity και σε γλώσσα C#. Επιπλέον θα γίνει περαιτέρω ανάλυση πως μπορούμε να προσθέσουμε και να δημιουργήσουμε ένα animation μέσω του animation window της Unity.

4.1 Διαχείρηση των Assets

Είναι σημαντικό να λάβουμε υπόψη μας όταν κανούμε ένα project αυτού του μεγέθους, ότι η διαχείρηση των Assets του project είναι εξαιρετικά σημαντική για λόγους παραγωγικότητας. Ως Assets ορίζουμε τα αρχεία Scripts, τα Animations, Scenes κτλ που ανήκουν στο Project. Αυτό επιτυχάνεται χωρίζοντας τα Assets σε φακέλους. Στην εικόνα 4-1 βλέπουμε στο Project Window ότι έχουμε χωρίσει το Project μας σε 7 βασικούς φακέλους: **Animations, Font, Prefabs, Scenes, Scripts, Sounds, Sprites.**



Εικόνα 4-1 Project Window απο το παιχνίδι Jack The Giant

- Στον φάκελο **Animations** έχουμε όλα τα amimations μας για το project.
- Στο φάκελο **Font** έχουμε την γραμματοσειρά που χρησιμοποιούμε για το κείμενο που εμφανίζεται στον παίκτη
- Στον φάκελο **Prefabs** έχουμε όλα τα prefab των GameObject.
- Στον φάκελο **Scenes** περιέχονται όλες οι διαφορετικές σκηνές που έχουμε για το παιχνίδι π.χ Main Menu, Option Menu
- Στον φάκελο **Scripts** περιέχονται όλα τα scripts του παιχνιδιού. Αυτός ο φάκελος είναι ο συμαντικότερος για το project καθώς περιέχει των κώδικα για κάθε Object.

- Στον φάκελο **Sounds** περιέχονται όλοι οι ήχοι που χρησιμοποιούμε στο παιχνίδι
- Στον φάκελο **Sprites** περιέχονται όλες οι εικονές που θα χρησιμοποιηθούν στο παιχνίδι και είναι ο δεύτερο συμαντικότερος για το project.

Αν και είναι φαινομενικά ευκολότερο να τοποθετήσουμε όλα τα scripts σε ένα μεγάλο script, στην πραγματικότητα αυτή η πρακτική είναι προβληματική. Είναι πιο παραγωγικό να ομαδοποιήσουμε τα scripts ανά GameObject γιατί έτσι κάποια αλλαγή στο GameObject αυτό μπορεί να επηρεάσει μόνο ένα script και αποφεύγονται λάθη κατά το editing των scripts.

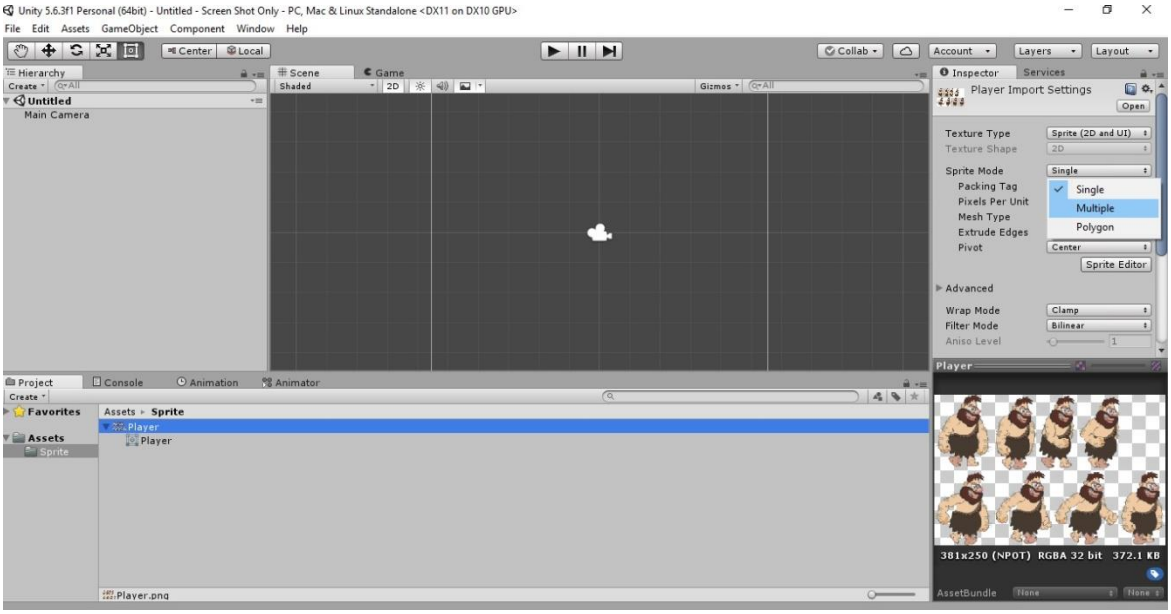
Στην επόμενη ενότητα θα αναφερθεί ο τρόπος με τον οποίο μπορούμε να χωρίσουμε μια εικόνα σε πολλά sprites και να δημιουργήσουμε ένα animation με αυτά.

4.2 Χωρισμός sprite και δημιουργία animation

Θα πρέπει να αναγνωριστεί εδώ η τεράστια συμβολή του περιβάλλοντος Unity ως ένα εργαλείο (τύπου Ελβετικού σουγιά) που επιτρέπει σε ένα και μόνο περιβάλλον τον διαχωρισμό σε πολλά διαφορετικά sprites και την δημιουργία animations με την χρήση Animator Window, Animation Window, Sprite Editor και όλα να συνεργάζονται άμεσα (seamless integration) μεταξύ τους.

Ο Sprite Editor της Unity είναι ένα ισχυρό εργαλείο το οποίο μας διευκολύνει για την εξαγωγή στοιχείων από μια σύνθετη εικόνα. Συχνά είναι πιο βολικό να συνδυάζουμε πολλά σχετικά γραφικά μαζί σε μια ενιαία εικόνα.

4.2.1 Διαδικασία εξαγωγής Sprite από μια εικόνα



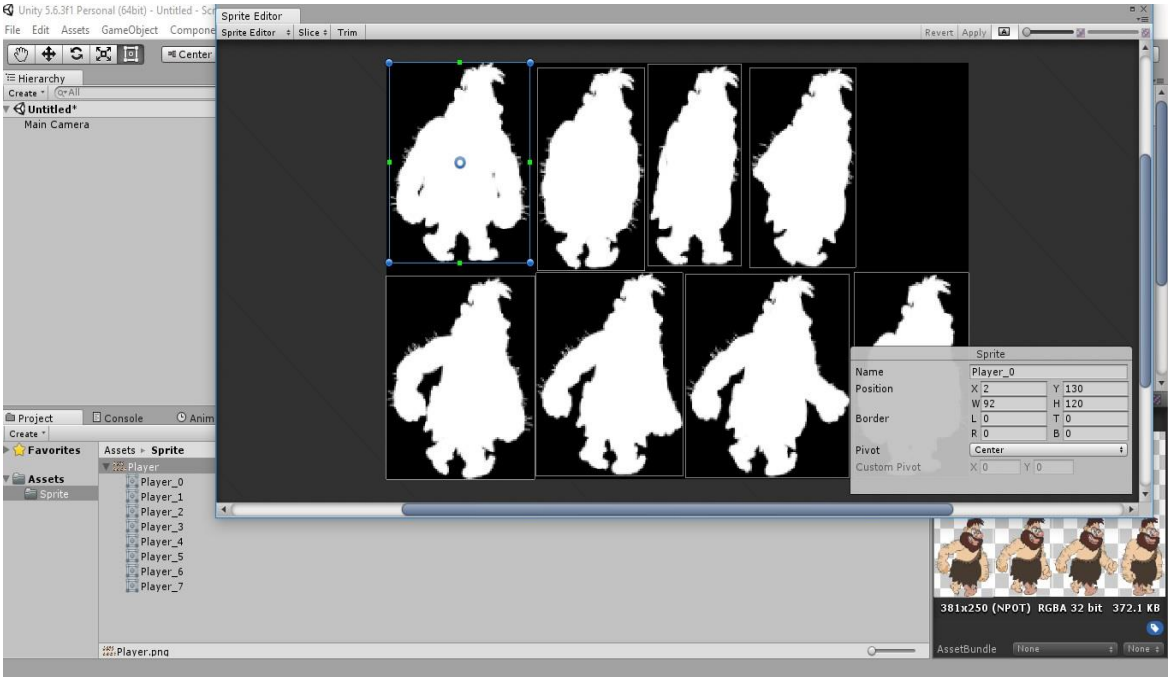
Εικόνα 4-2 Βήμα Πρώτο

Στην εικόνα 4-2 έχουμε το **πρώτο βήμα**. Στο Inspector Window τα πρώτα πράγματα που **πρέπει να προσέξουμε** πριν ανοίξουμε τον Sprite Editor είναι τα εξής. Το **Texture Type** της εικόνας να είναι **Sprite (2D and UI)** και το **Sprite Mode** πρέπει από single να το αλλάξουμε σε **Multiple**.

Εικόνα 4-3 Βήμα Δεύτερο Sprite Editor Window

Στην είκονα 4-3 βλέπουμε το παράθυρο του Sprite Editor. Σε αυτό το βήμα πατώντας το drop-down κουμπί Slice βλέπουμε τις επιλογές που εμφανίζονται. Στο Type

διαλέγουμε την επιλογή Automatic και πατάμε Slice για να χωρίσει ο Sprite Editor την εικόνα σε πολλά sprites. Επίσης μπορούμε να χωρίσουμε την εικόνα σε sprites και χειροκίνητα, η αλήθεια όμως είναι πως πολύ σπάνια θα χρειαστεί επειδή ο Sprite Editor της Unity είναι εξαιρετικός στο να ξεχωρίζει αυτόματα τα sprite από μία εικόνα.

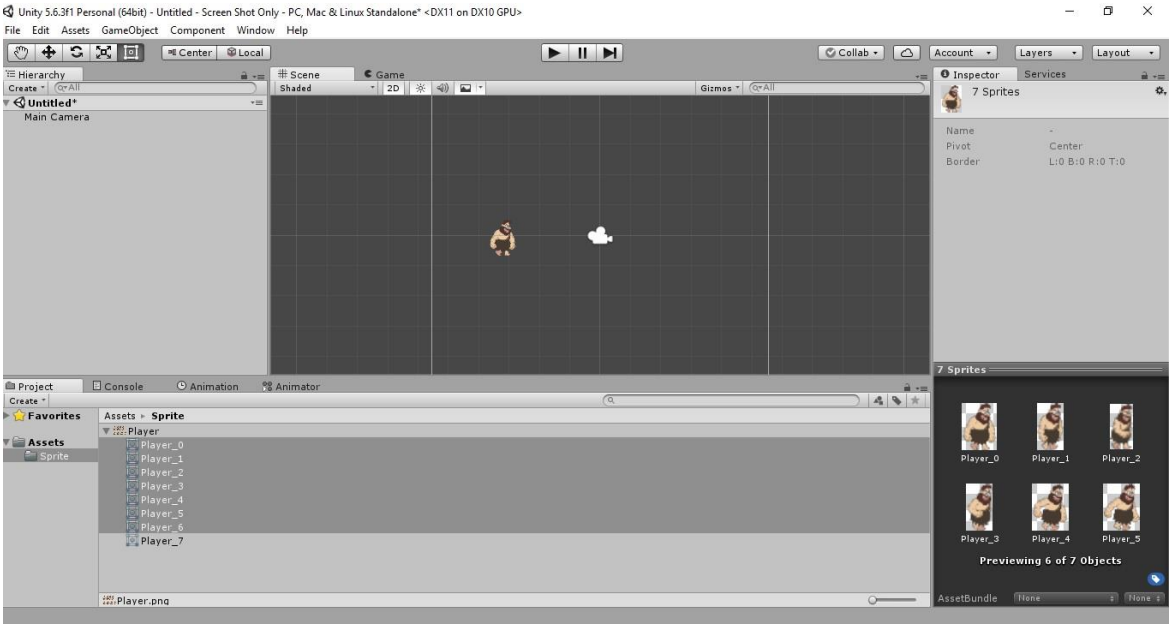


Εικόνα 4-4 Βήμα Τρίτο Sprite Editor Window που δειχνει το τελικο αποτέλεσμα

Στην εικόνα 4-4 βλέπουμε το τελικό αποτέλεσμα. Ο Sprite Editor έχει χωρίσει την εικόνα σε οκτώ sprites. Μπορούμε να ξεχωρίσουμε τα διαφορετικά sprites καθώς διακρίνουμε το οριοθετημένο πλαίσιο για το κάθε sprite.

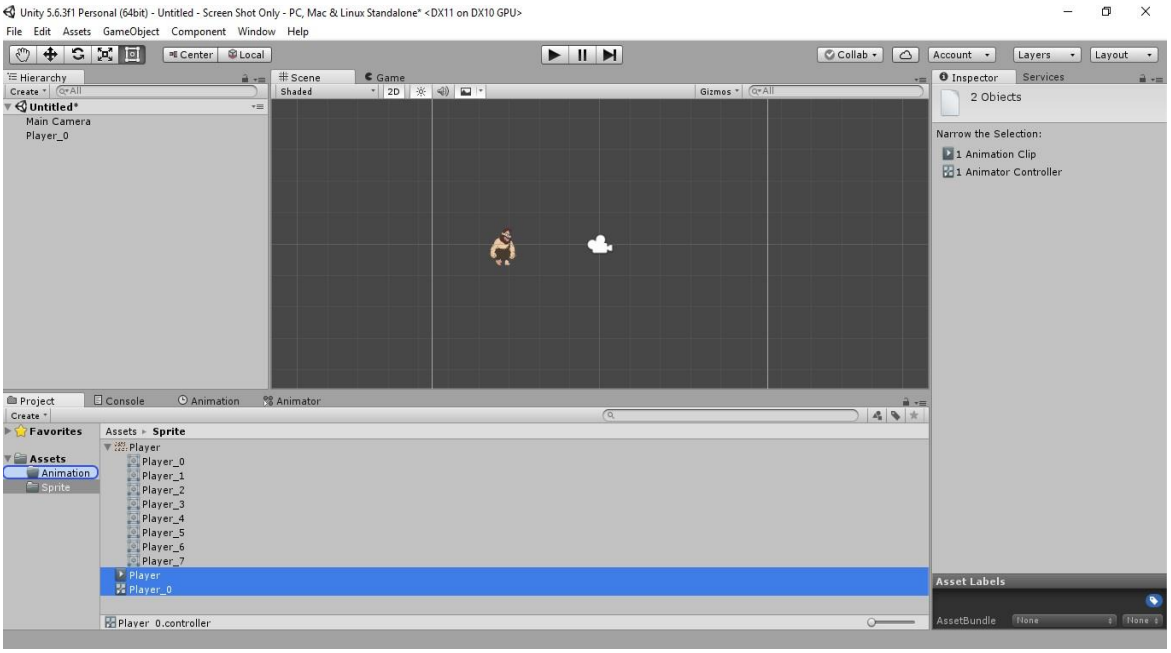
Παρακάτω θα χρησιμοποιήσουμε τα sprites που δημιουργήσαμε προηγουμένος για να φτιάξουμε ένα animation.

4.2.2 Δημιουργία Animation

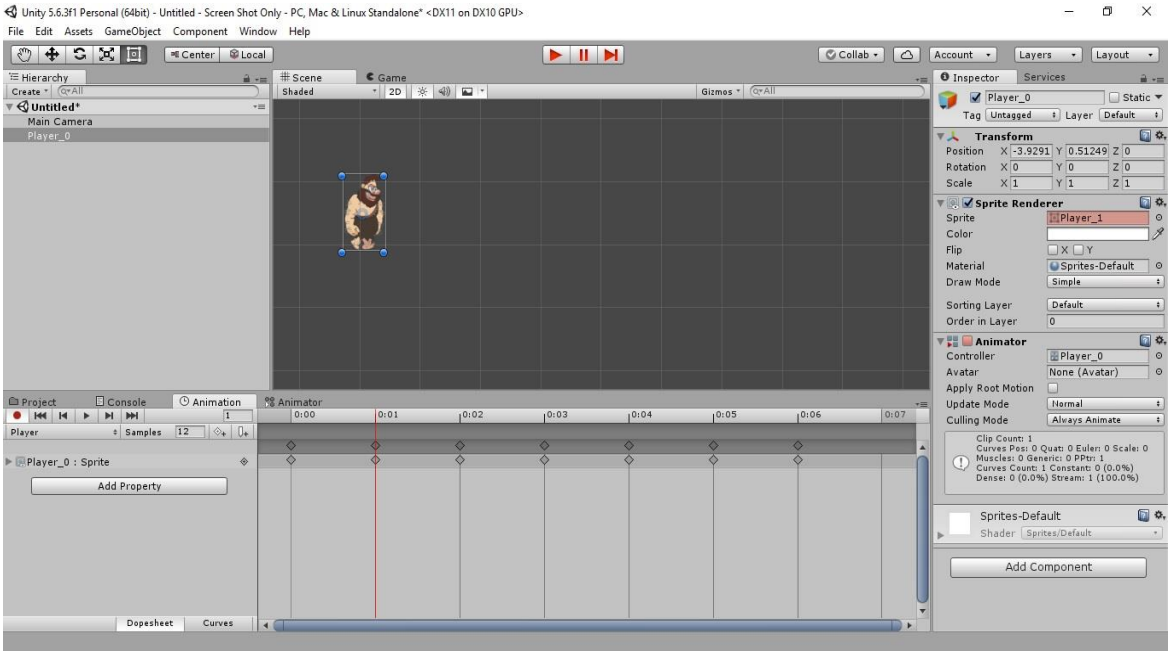


Εικόνα 4-5 Βήμα Πρώτο

Για το πρώτο βήμα επιλέγουμε τα sprites που θέλουμε να έχει το animation μας και τα τοποθετούμε στο Scene View. Σαν αποτέλεσμα αυτής της ενέργειας μας ο Editor δημιούργησε ένα GameObject με όνομα Player_0 που περιέχει όλα τα sprites που επιλέξαμε. Στην συνέχεια στον φάκελο των Assets φτιάχνουμε ένα φάκελο με όνομα Animations και τοποθετούμε εκεί το animation που δημιουργήθηκε.

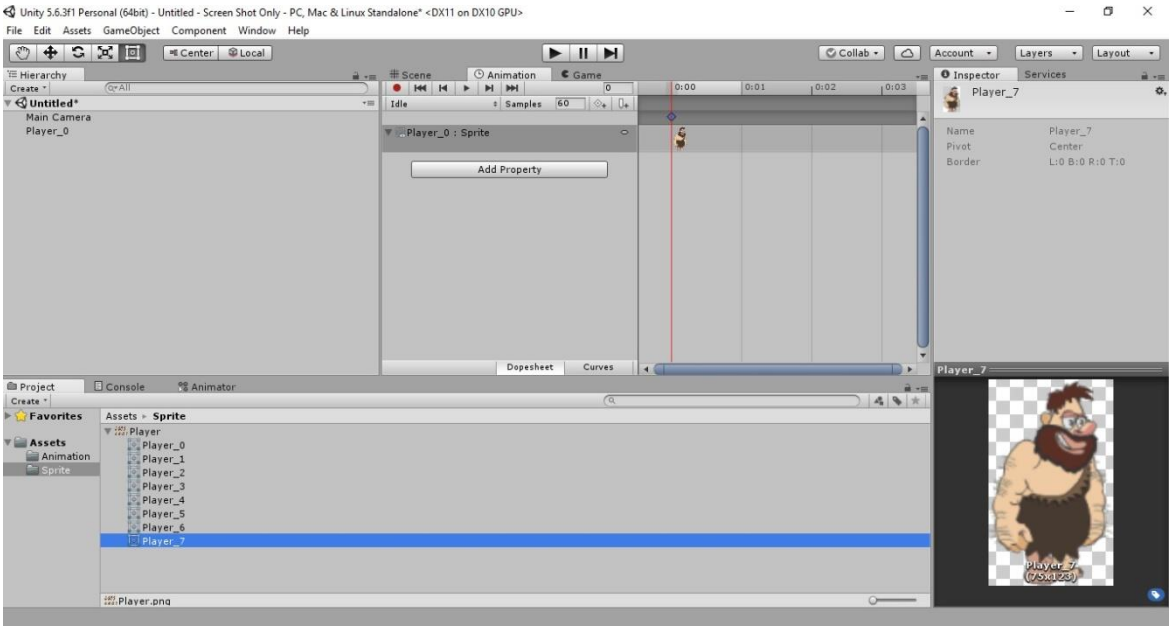


Εικόνα 4-6 Αποτέλεσμα του πρώτου βήματος



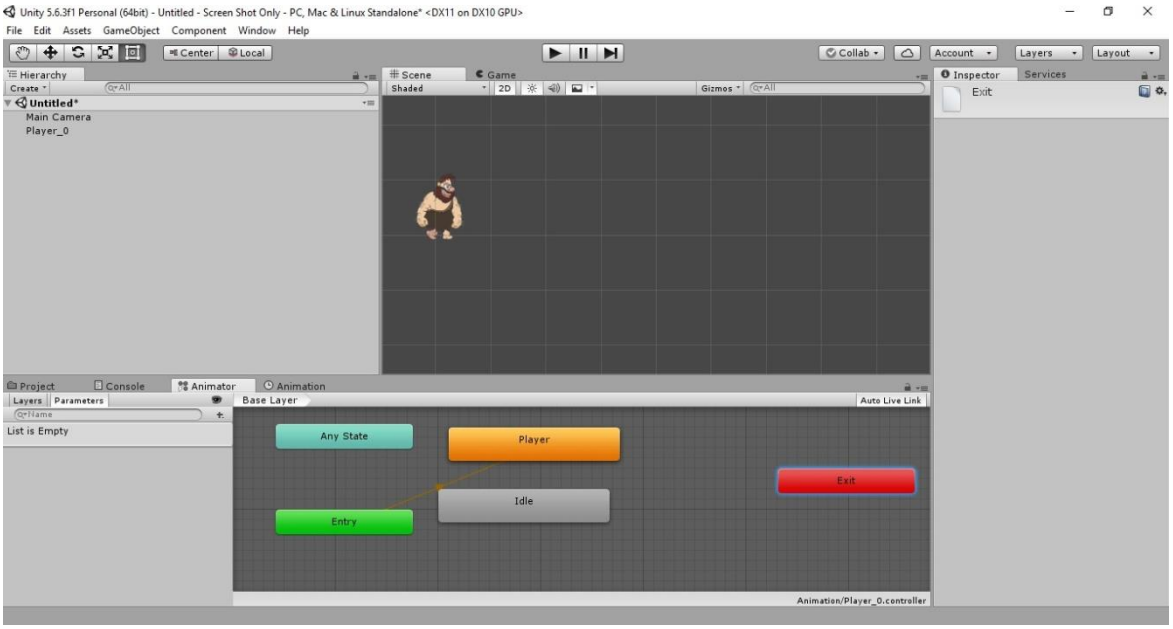
Εικόνα 4-7 Animation Window

Την εικόνα 4-7 επιλέγοντας το GameObject που δημιουργήθηκε και πηγαίνοντας στον Animation Window βλέπουμε το animation που έχει δημιουργηθεί απο την Unity. Η Unity από μόνη της τοποθετεί ένα sprite ανά δευτερόλεπτο. Επιλέγοντας το Play Button μπορούμε να δούμε τον χαρακτήρα να κουνιέται σαν να περπατάει.

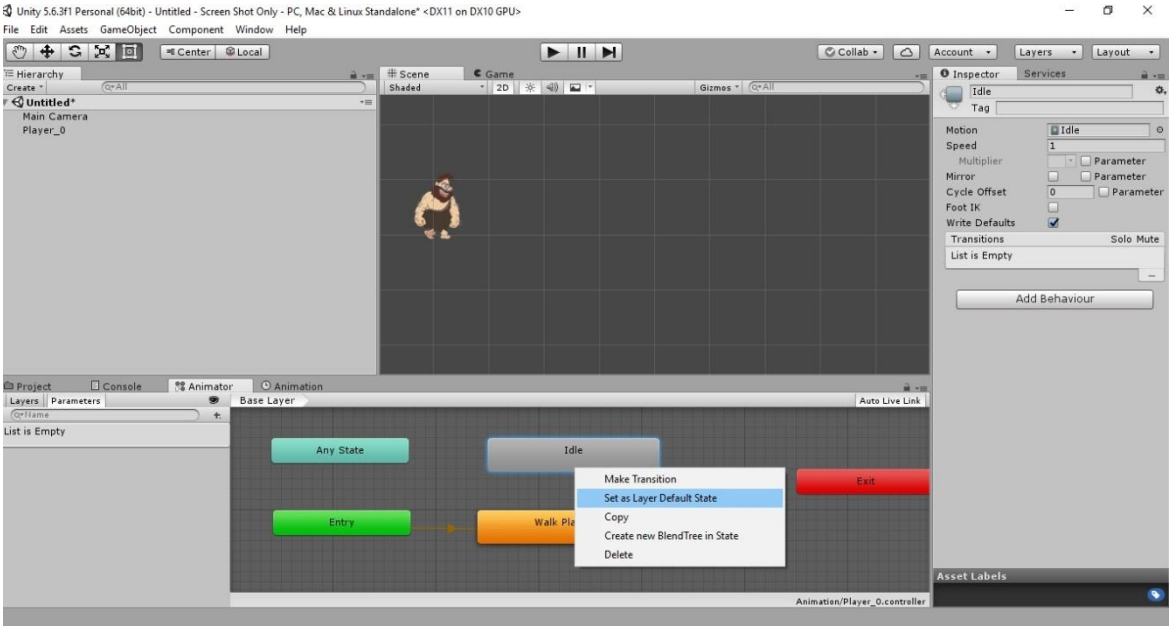


Εικόνα 4-8 Βήμα Δεύτερο

Για το **δεύτερο βήμα** τοποθετούμε το Animation Window στο Scene View και επιλέγουμε το sprite με το όνομα Player_7 και το τοποθετούμε στο Animatio Window. Με αυτόν τον τρόπο δημιουργούμε την idle κατάσταση του animation μας.



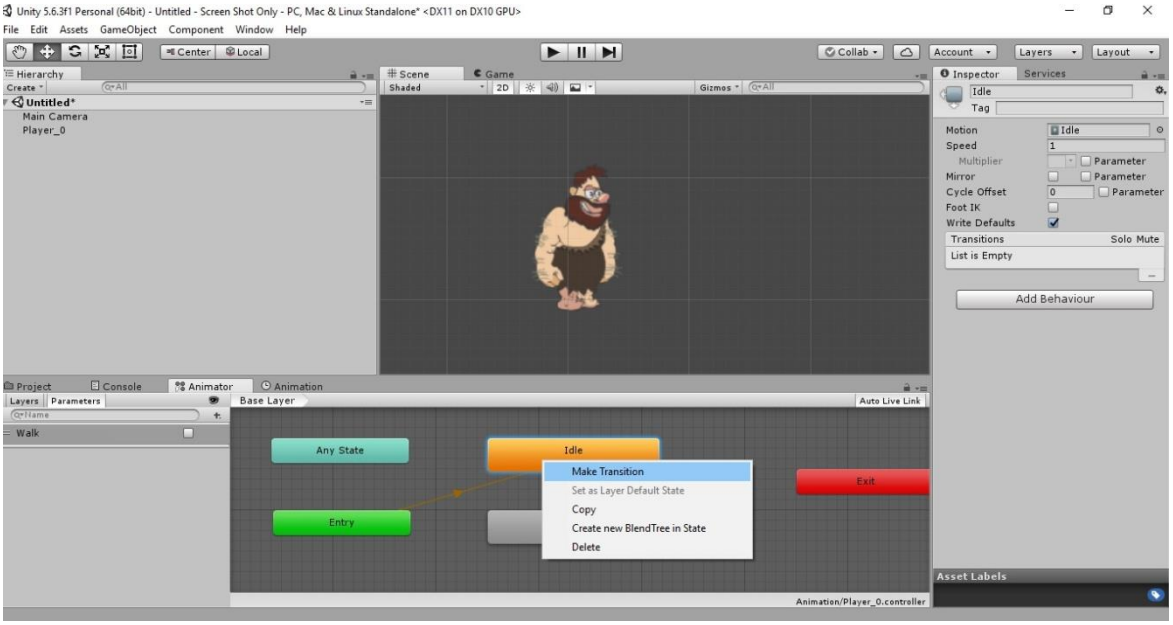
Εικόνα 4-9 Animator Window μέτα το βήμα 2.



Εικόνα 4-10 Βήμα Τρίτο Animator Window

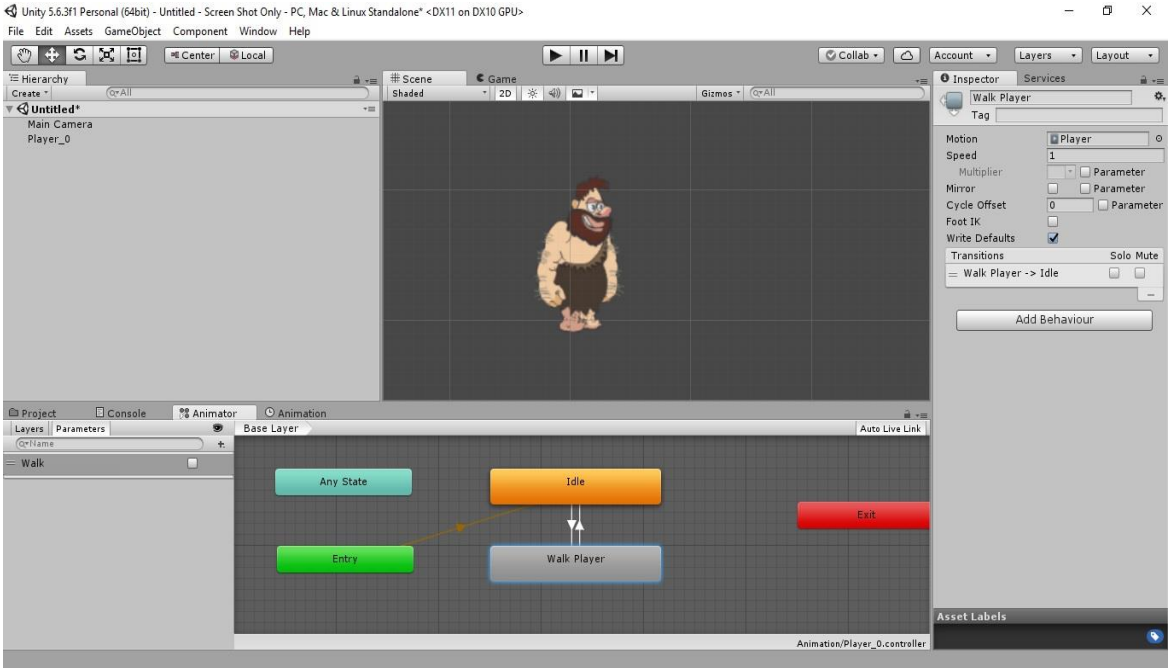
Όταν αρχίζουμε το παιχνίδι η default κατάσταση του animation είναι το walking animation. Η default κατάσταση ξεχωρίζει με πορτοκαλί χρώμα. Ωστόσο συνήθως όταν ξεκινάει το παιχνίδι δεν θέλουμε να αρχίζει το animation χωρίς ο παίκτης να κινείται. Εδώ χρησιμεύει το Idle animation που δημιουργήθηκε στο προηγούμενο βήμα. Για να γίνει

αυτό στο animator window κάνουμε δεξί click στο Idle και επιλέγουμεε Set as Layed Default State (**βήμα τρίτο.**)



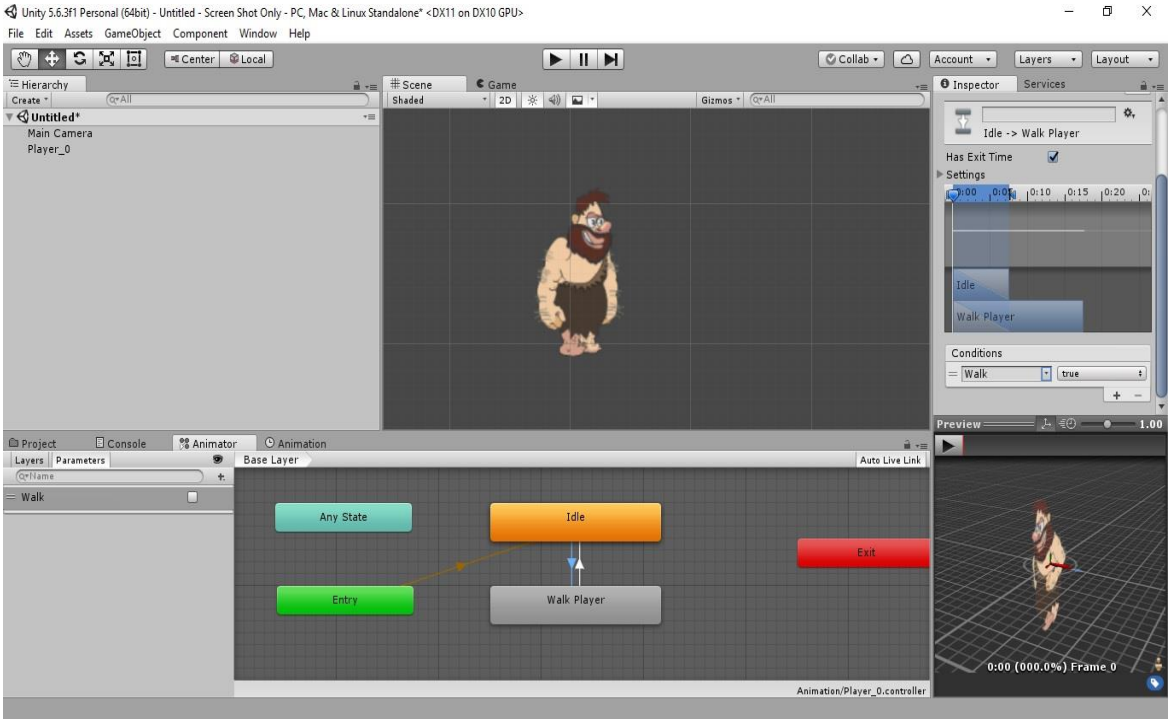
Εικόνα 4-11 Βήμα Τέταρτο

Στο **τέταρτο βήμα** δημιουργούμε μία παράμετρο με το όνομα Walk. Την παράμετρο Walk θα την χρειαστούμε για να ορίσουμε πότε υπάρχει μετάβαση από την κατάσταση idle στο walk animation. Για να φτιάξουμε μία μετάβαση μεταξύ δύο καταστάσεων κάνουμε δεξί click και πρώτα στην μία κατάσταση και επιλέγουμε Make Transition, αυτή η επιλογή εμφανίζει ένα βελάκι για ενώσουμε την μία κατάσταση με την άλλη.

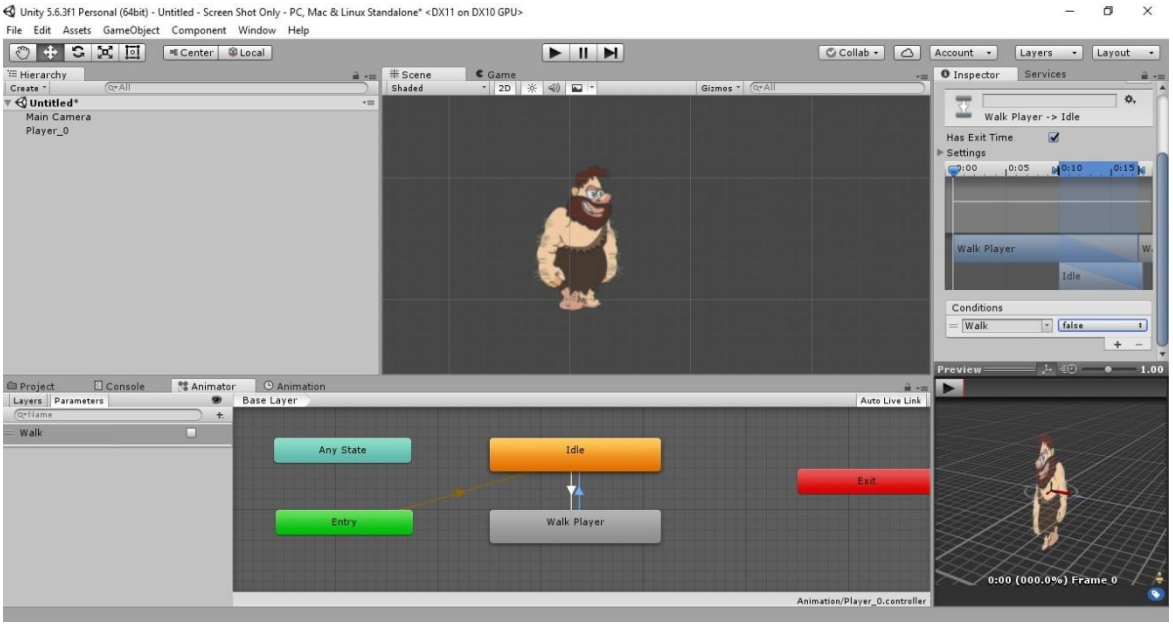


Εικόνα 4-12 Αποτέλεσμα του τέταρτου βήματος

Τέλος στο **πέμπτο βήμα** θα βάλουμε παράμετρους στις μεταβάσεις μεταξύ των καταστάσεων για να γνωρίζει το animation μέσω του κώδικα πότε πρέπει να παίζει το walking animation και πότε να μένει idle. Επιλέγοντας το βελάκι της μετάβασεις απο το idle στο walking animation στο Inspector window όπως βλέπουμε στην εικόνα 4-13. Στα conditions για να γίνει η μετάβαση θα τοποθετήσουμε την παράμετρο walk που δημιουργήσαμε προηγουμένος και θα επιλέξουμε **true**. Στην εικόνα 4-14 θα κάνουμε την ιδια διαδικασία μόνο που θα επιλέξουμε αυτή την φορά **false** επειδή η μετάβαση είναι τώρα από το walking animation στην idle κατάσταση.



Εικόνα 4-13 Βήμα Πέμπτο Inspector Window του Transition από την idle κατάσταση στο walk animation



Εικόνα 4-14 Βήμα Πέμπτο Inspector Window του Transition από το walk animation στην idle κατάσταση

Στις ενότητες που ακολουθούν θα γίνει εξήγηση του κώδικα για τα παιχνίδια που αναπτύχθηκαν για αυτή την εργασία του Jack The Giant και Flappy Bird.

4.3 Components που χρησιμοποιήθηκαν

- **Rigidbody 2D** είναι ένα component που τοποθετεί το GameObject υπό τον έλεγχο του physics engine της Unity. [67]
- **Box Collider 2D** είναι ένα collider component για χρήση με 2D physics. [68]
- **Button (Script)** είναι ένα απλό κουμπί που πρέπει να πατήσουμε για να ενεργοποιήσουμε ένα συμβάν. [69]

4.4 Game Jack The Giant: Κώδικας και επεξήγησή του

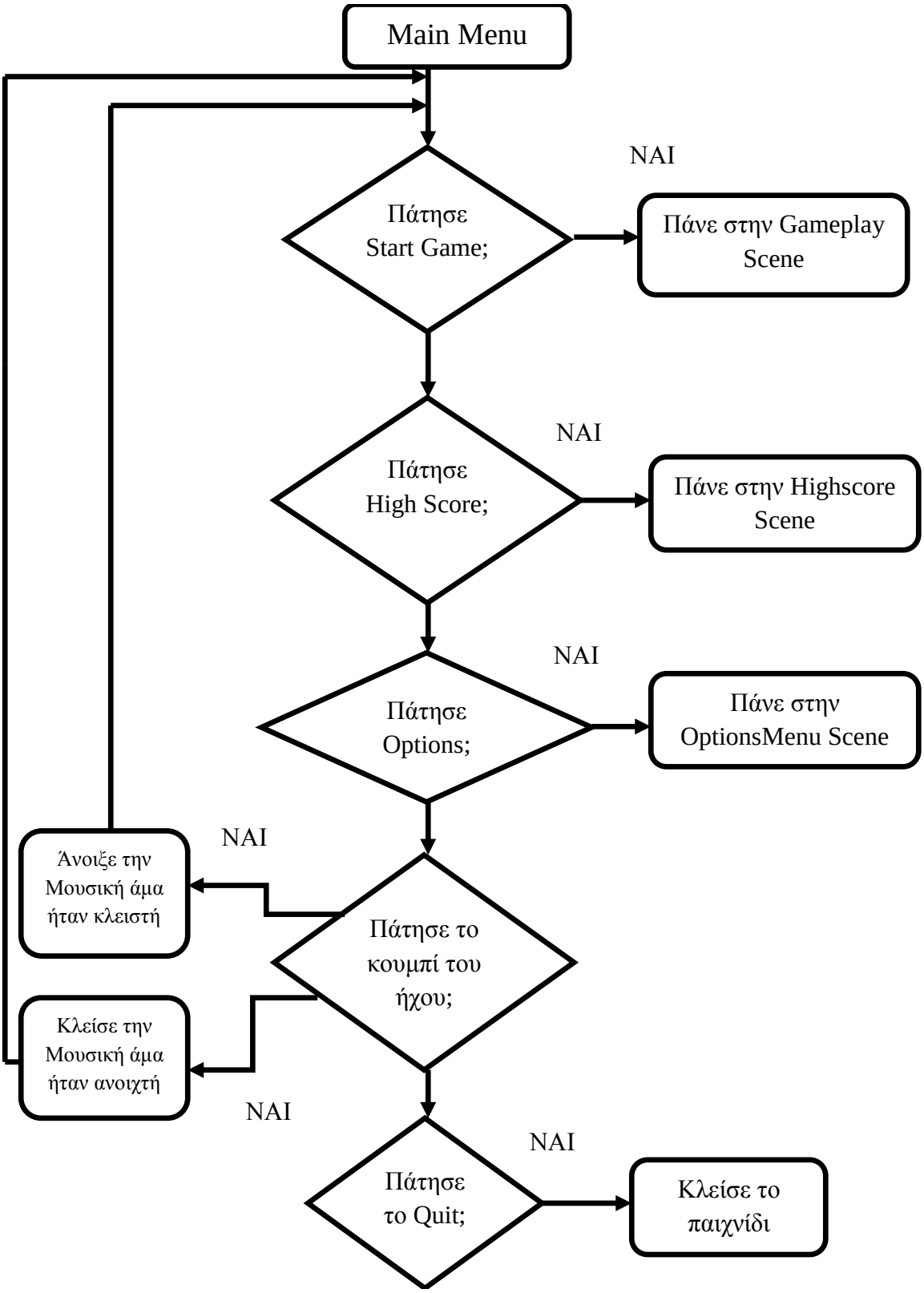
Ο σκοπός του παιχνιδιού Jack The Giant είναι ο παίκτης να πηδάει από σύννεφο σε σύννεφο και να μαζεύει νομίσματα ένω η ταχύτητα που κατέβενει η κάμερα αυξάνεται.

Ο κώδικας για το παιχνίδι Jack the Giant θα παρουσιαστεί σε διαγράμματα ροής, για αυτόν τον λόγο ο κώδικας θα μπει σε appendix. Στο project ο κώδικας χωρίζεται σε πολλά διαφορετικά script, για αυτό τον λογό το καθέ script βρίσκεται σε έναν φάκελο που περιέχει scripts για τον ίδιο σκοπό ή για το ίδιο GameObject.

4.4.1 Διάγραμμα Ροής Main Menu Scene

Το παρακάτω διάγραμμα ροής δείχνει τον τρόπο που λειτουργεί το main menu scene. Για να έχουμε αυτό το αποτέλεσμα χρησιμοποιούνται τα παρακάτω scripts:

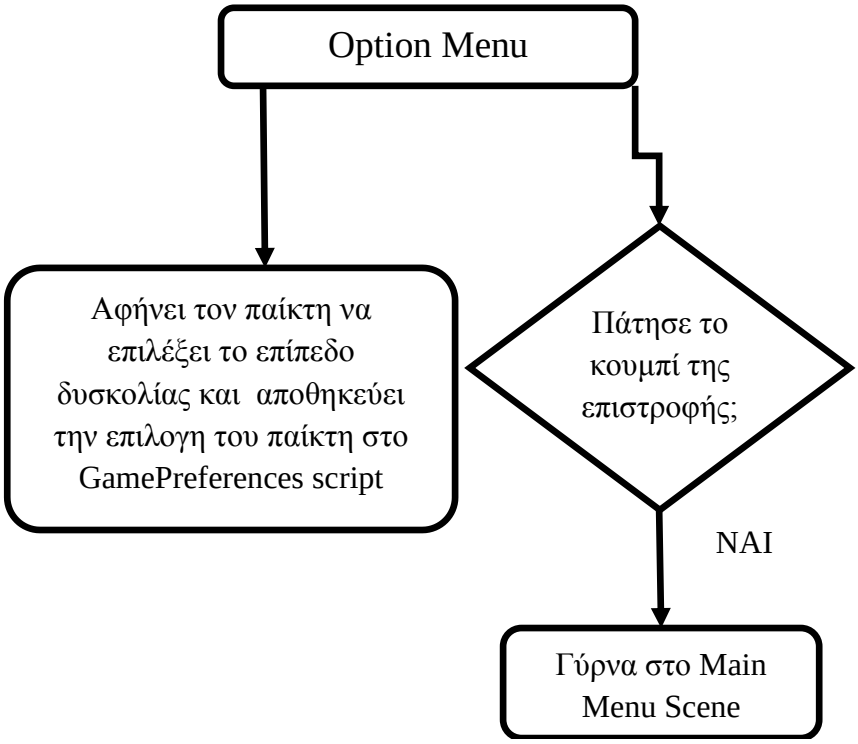
- Το MainMenuController script που επιτρέπει τον παίκτη να αλληλεπιδρά με τα κουμπιά που ύπαρχουν στο Main Menu και να τους μεταφέρει στην ανάλογη σκηνή.
- Το GameManager script μας ακολουθάει σε όλα τα scenes του παιχνιδιού και το υπεύθυνο script που κρατάει συμαντικά στοιχεία για το παιχνίδι όπως ποιο είναι το score, πόσες ζωές έχει ο παίκτης, ποιος είναι ο βαθμός δυσκολίας του παιχνιδιού.
- Το GamePreferences script άμα **δεν είναι** η πρώτη φόρα που ανοίγουμε το παιχνίδι έχει αποθηκευμένα τις προτιμήσεις του παίκτη.
- MusicController script ενεργοποιεί ή απενεργοποιεί την μουσική του παιχνιδιού όταν πατιέται το κουμπί του ήχου.


Main Menu
Πάτησε Start Game;
ΝΑΙ
Πάνε στην Gameplay Scene
Πάτησε High Score;
ΝΑΙ
Πάνε στην Highscore Scene
Πάτησε Options;
ΝΑΙ
Πάνε στην OptionsMenu Scene
Πάτησε το κουμπί του ήχου;
ΝΑΙ
Άνοιξε την Μουσική άμα ήταν κλειστή
Κλείσε την Μουσική άμα ήταν ανοιχτή
Πάτησε το Quit;
ΝΑΙ
ΝΑΙ
Κλείσε το παιχνίδι

4.4.2 Διάγραμμα ροής Option Menu Scene

Το Option Menu scene χρησιμοποιεί τα παρακάτω scripts:

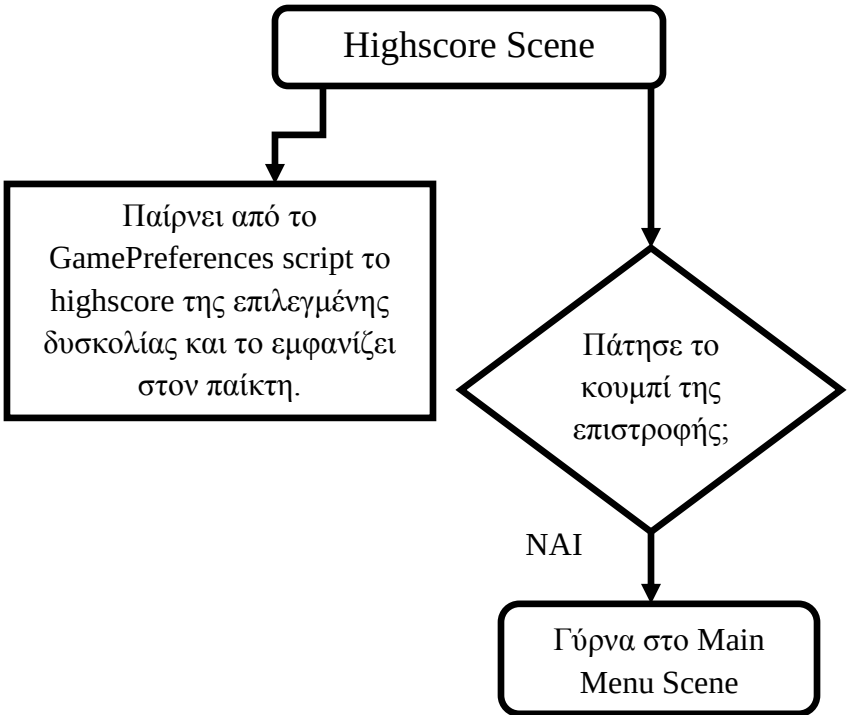
- Το OptionsController script που βρίσκει ποιο επίπεδο επέλεξε ο παίκτης.
- Το GamePreferences script όπου αποθηκεύει το επίπεδο που διάλεξε ο παίκτης.



4.4.3 Διάγραμμα Ροής Highscore Scene

Το Highscore scene χρησιμοποιεί τα παρακάτω scripts:

- Το HighscoreController script δείχνει το highscore του παίκτη ανάλογα σε ποιο difficulty έχει επιλέξει να παίζει.
- Το GamePrefernces script οπού σε αυτό βρίσκονται αποθηκευμένα όλα τα Highscore του παίκτη.

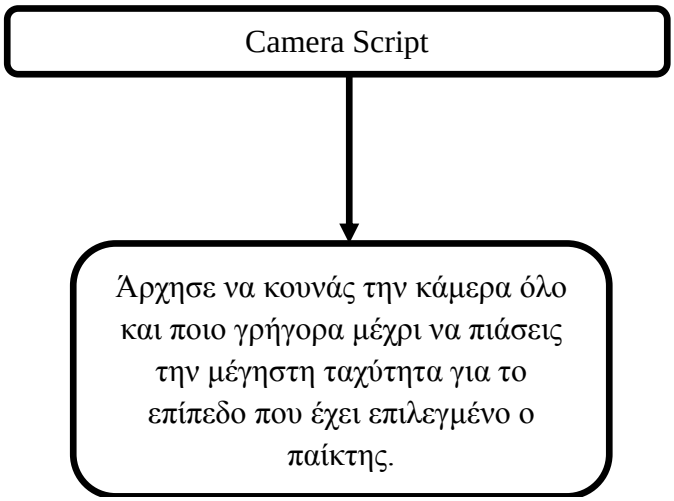


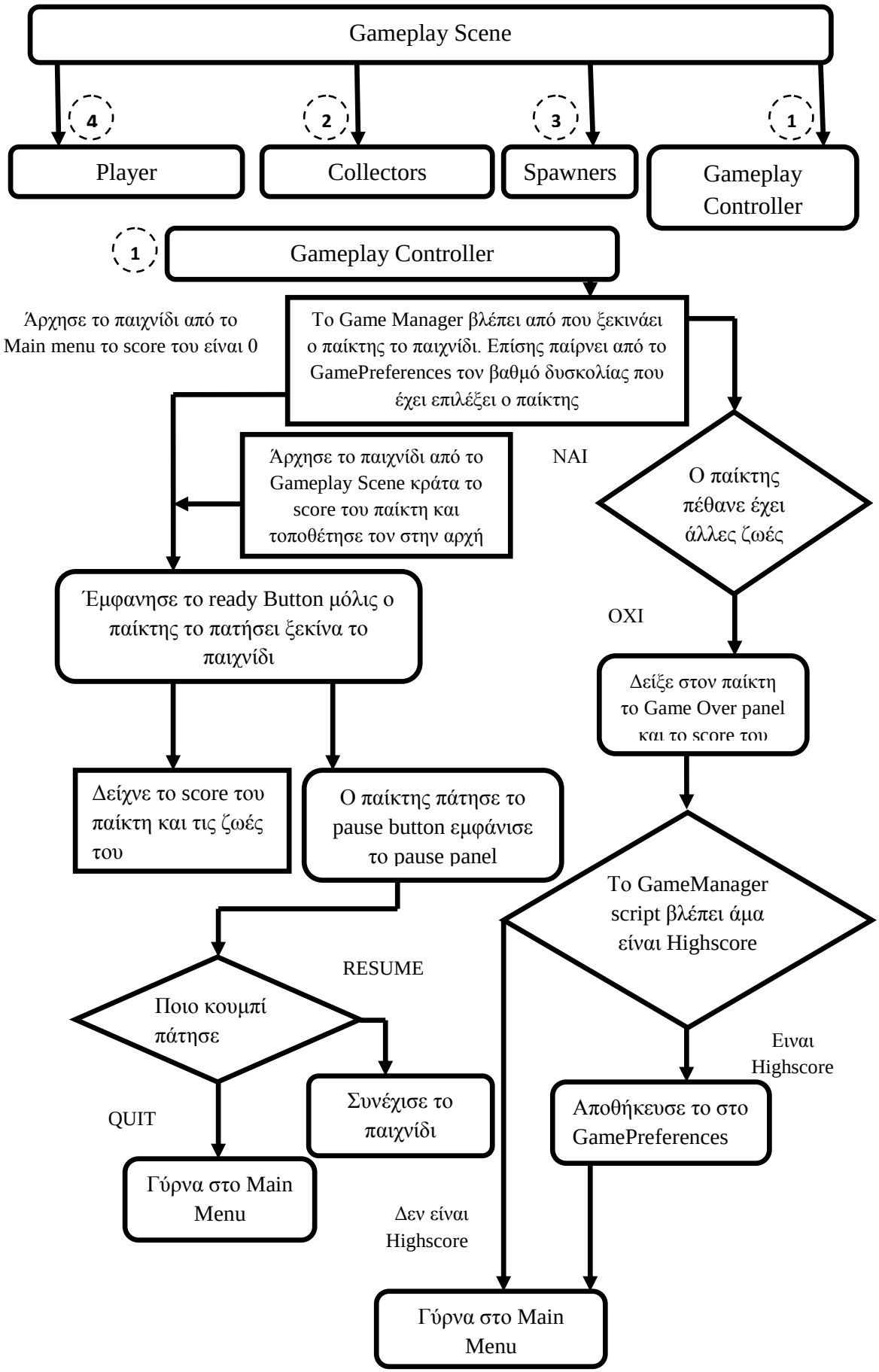
4.4.4 Διάγραμμα ροής Gameplay Scene

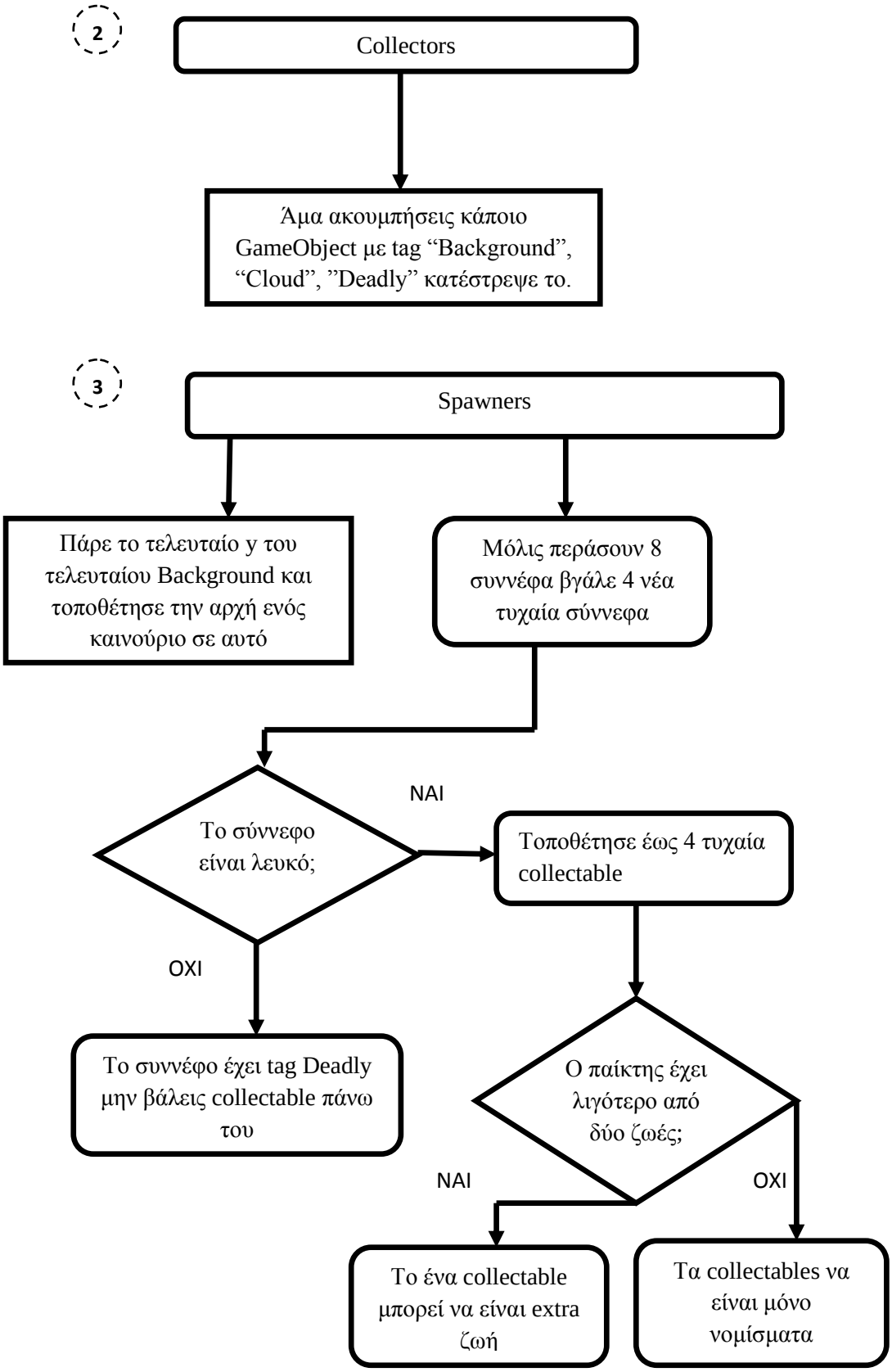
Το Gameplay Scene χρησιμοποιεί τα παρακάτω script:

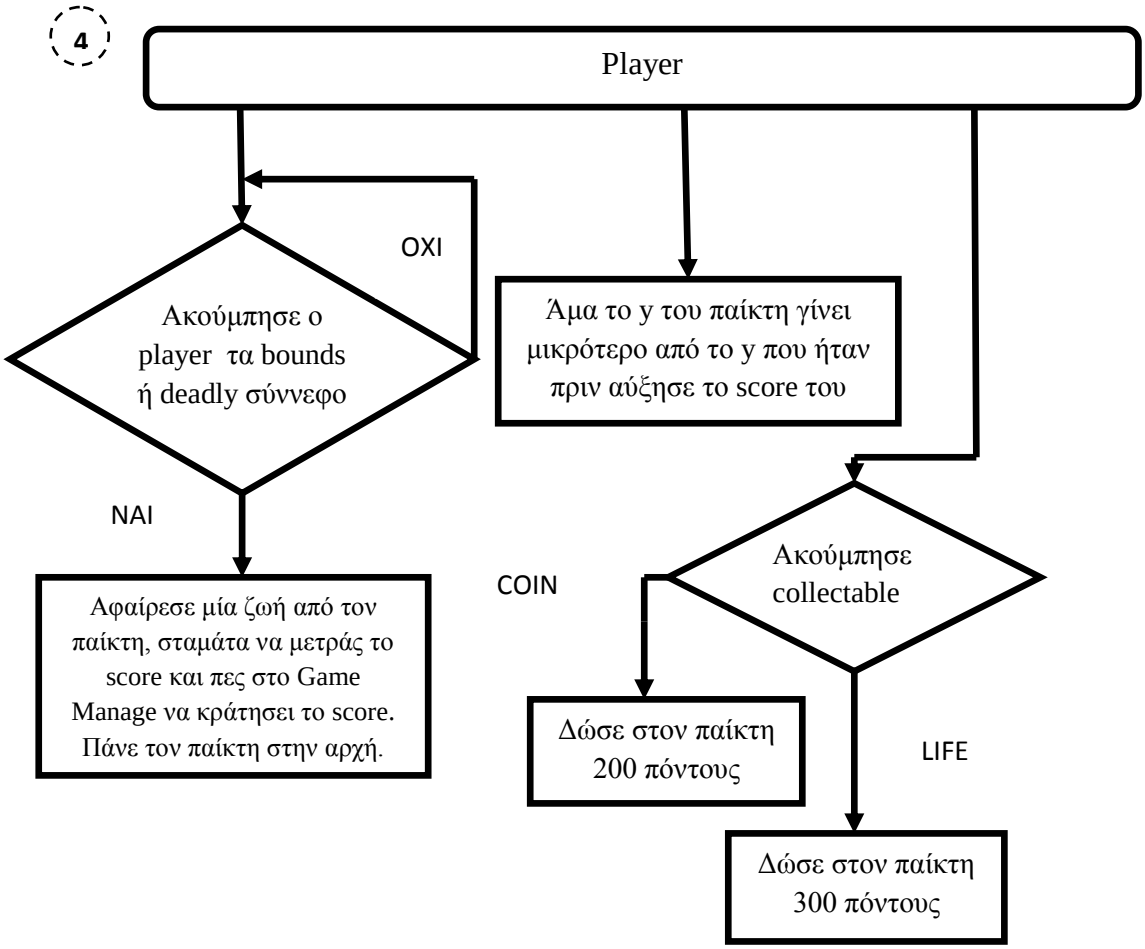
- Το GameplayController script μας δίνει την δυνατότητα να έχουμε text μέσα στο παιχνίδι για να δείχνουμε στον παίκτη πόσες ζωές έχει, το score του, πόσα νομίσματα έχει μαζέψει. Επίσης μας δίνει την δυνατότητα να κάνουμε pause το παιχνίδι.
- Το GameManager script μας ακολουθάει σε όλα τα scenes του παιχνιδιού και το υπεύθυνο script που κρατάει συμαντικά στοιχεία για το παιχνίδι όπως ποιο είναι το score, πόσες ζωές έχει ο παίκτης, ποιος είναι ο βαθμός δυσκολίας του παιχνιδιού.
- Το GamePreferences script για να μας πει πιο είναι το επίπεδο δυσκολίας και να αποθηκεύσει το highscore του παίκτη.
- Τα δύο collectors scripts, BGCollector script και CloudCollector script. Όπου καταστρέφουν το background και τα σύννεφα που βγαίνουν έξω από την κάμερα.
- Τα δύο spawners scripts, BGSpawner script και CloudSpawner script. Όπου δημιουργούν νέο background, σύννεφα και collectables.
- Το camera script δίνουμε κίνηση στην κάμερα μας. Στο ίδιο script επίσης μπορούμε να ορίσουμε τις μέγιστες ταχύτητες ανάλογα την δυσκολία που έχει επιλέξει ο παίκτης.

- Τα Player scripts που είναι το Player script δίνει την δυνατότητα στον player να κουνιέται με το πληκτρολόγιο. Το PlayerBounds script δεν επιτρέπει τον player να βγαίνει έξω από την οθόνη του παιχνιδιού. Το PlayerMoveJoystick script που μαζί με το Joystick script χρησιμοποιούντε για να μπορέσουμε να κουνάμε τον παίκτη με το ποντίκι ή από την οθόνη ενός smartphone άμα παίζουμε το παιχνίδι στο κινητό. Το PlayerScore script μετράει το score που έχει ο παίκτης ενώ παίζει το παιχνίδι καθώς και όταν χάνει μία ζωή ο παίκτης να τον τοποθετή ξανά στην αρχική του θέση και άμα δεν έχει άλλες ζωές τότε τερματίζει το παιχνίδι.




Gameplay Scene
4
2
3
1
Player
Collectors
Spawners
Gameplay Controller
1
Gameplay Controller
Άρχησε το παιχνίδι από το Main menu το score του είναι 0
Το Game Manager βλέπει από που ξεκινάει ο παίκτης το παιχνίδι. Επίσης παίρνει από το GamePreferences τον βαθμό δυσκολίας που έχει επιλέξει ο παίκτης
Άρχησε το παιχνίδι από το Gameplay Scene κράτα το score του παίκτη και τοποθέτησε τον στην αρχή
ΝΑΙ
Ο παίκτης πέθανε έχει άλλες ζωές
Έμφανησε το ready Button μόλις ο παίκτης το πατήσει ξεκίνα το παιχνίδι
ΟΧΙ
Δείξε στον παίκτη το Game Over panel και το score του
Δείχνε το score του παίκτη και τις ζωές του
Ο παίκτης πάτησε το pause button εμφάνισε το pause panel
Το GameManager script βλέπει άμα είναι Highscore
RESUME
Ποιο κουμπί πάτησε
Ειναι Highscore
Συνέχισε το παιχνίδι
Αποθήκευσε το στο GamePreferences
QUIT
Γύρνα στο Main Menu
Δεν είναι Highscore
Γύρνα στο Main Menu


2
Collectors
Άμα ακουμπήσεις κάποιο GameObject με tag “Background”, “Cloud”, ”Deadly” κατέστρεψε το.
3
Spawners
Πάρε το τελευταίο y του τελευταίου Background και τοποθέτησε την αρχή ενός καινούριο σε αυτό
Μόλις περάσουν 8 συννέφα βγάλε 4 νέα τυχαία σύννεφα
Το σύννεφο είναι λευκό;
ΝΑΙ
Τοποθέτησε έως 4 τυχαία collectable
ΟΧΙ
Το συννέφο έχει tag Deadly μην βάλεις collectable πάνω του
Ο παίκτης έχει λιγότερο από δύο ζωές;
ΝΑΙ
ΟΧΙ
Το ένα collectable μπορεί να είναι extra ζωή
Τα collectables να είναι μόνο νομίσματα


4
Player
ΟΧΙ
Ακούμπησε ο player τα bounds ή deadly σύννεφο
Άμα το y του παίκτη γίνει μικρότερο από το y που ήταν πριν αύξησε το score του
ΝΑΙ
Αφαίρεσε μία ζωή από τον παίκτη, σταμάτα να μετράς το score και πες στο Game Manage να κράτησει το score. Πάνε τον παίκτη στην αρχή.
COIN
Ακούμπησε collectable
Δώσε στον παίκτη 200 πόντους
LIFE
Δώσε στον παίκτη 300 πόντους

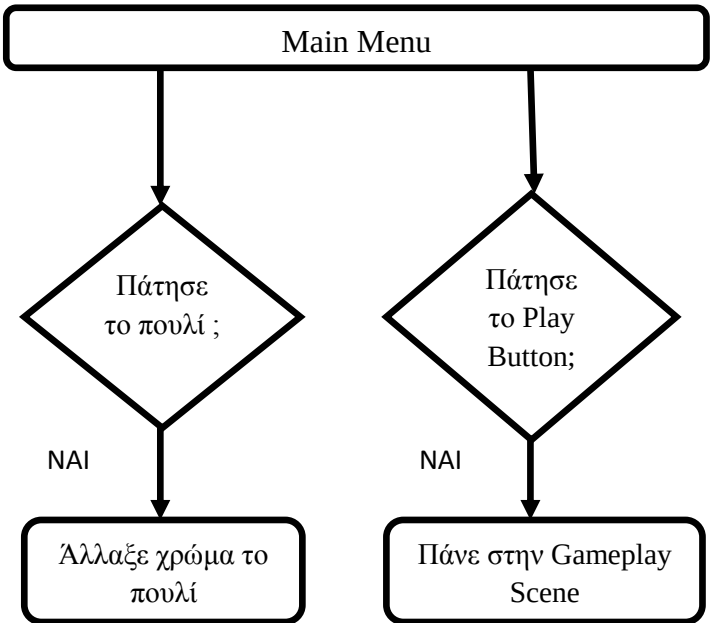
4.5 Game Flappy Bird: Κώδικας και επεξήγησή του

Ο σκοπός του παιχνιδιού Flappy Bird είναι ο παίκτης να κάνει το πουλί να πετάει ανάμεσα από τα εμπόδια ενώ αυτά εμφανίζονται διαδοχικά και σε τυχαίο ύψος. Όπως και με το παιχνίδι Jack The Giant έτσι και εδώ θα χαρήσουμε τον κώδικα σε διαγράμματα ροής και ο κώδικας θα μπει σε appendix

4.5.1 Διάγραμμα Ροής Main Menu scene

Το διάγραμμα ροής του Main Menu χρησιμοποιεί τα παρακάτω scripts:

- Το MenuController script μας δίνει την δυνατότητα να αλλάζουμε το χρώμα του πουλιού και να έχουμε αλληλεπίδραση με τα κουμπιά στο Main Menu
- Το GameController script είναι υπεύθυνο για να αποθηκεύει τις προτιμήσεις του παίκτη καθώς και το high score του.

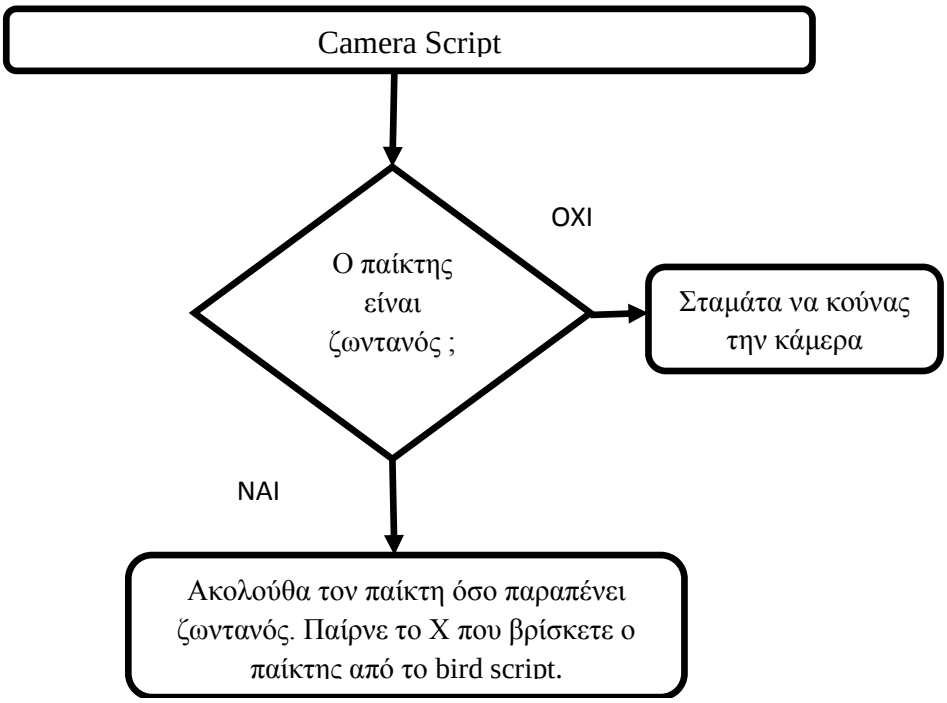


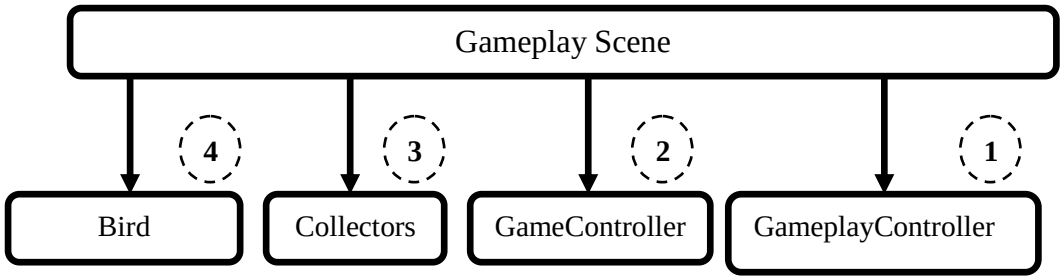
4.5.2 Διάγραμμα Ροής Gameplay scene

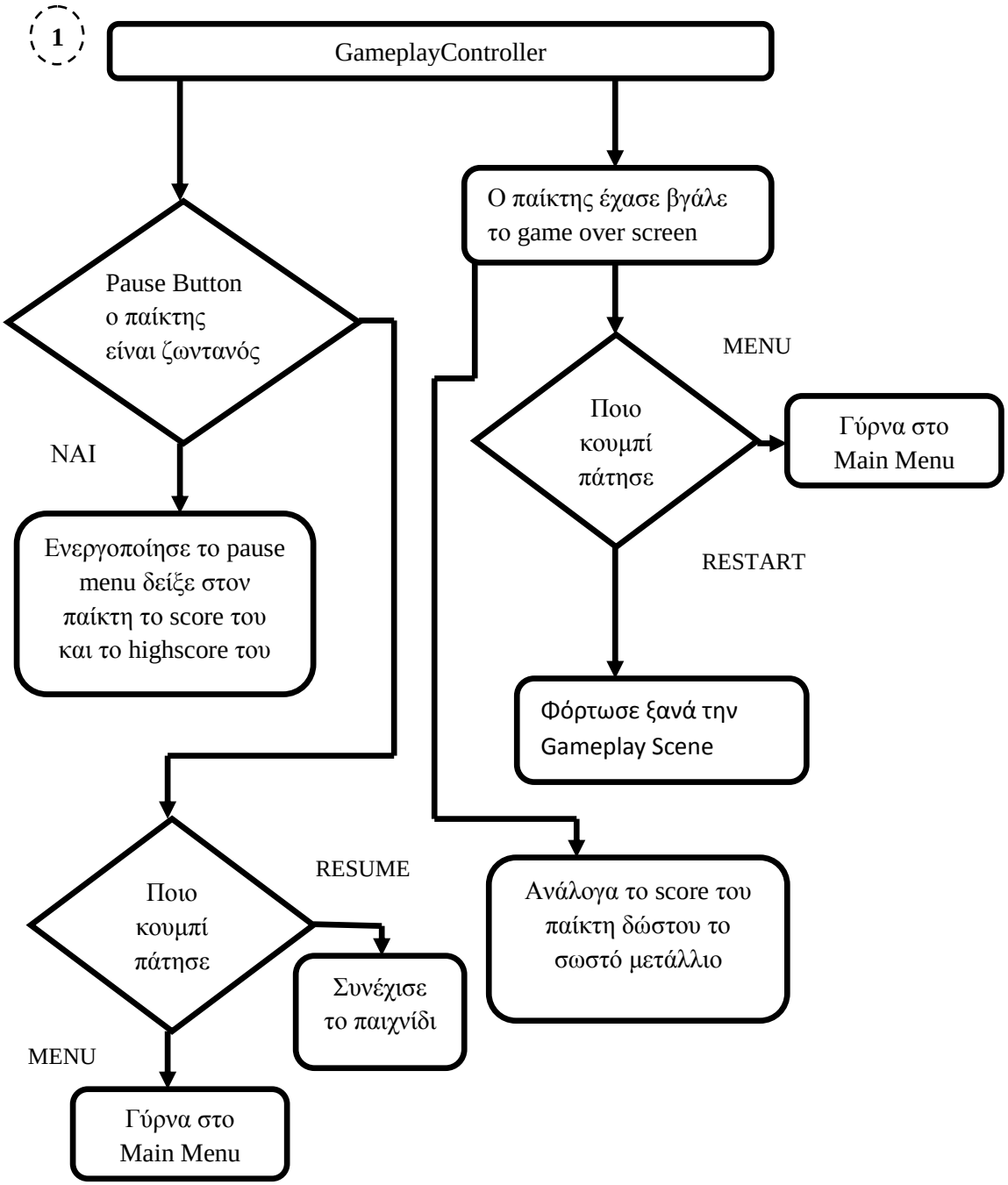
Το διάγραμμα ροής του Gameplay scene χρησιμοποιεί τα παρακάτω scripts:

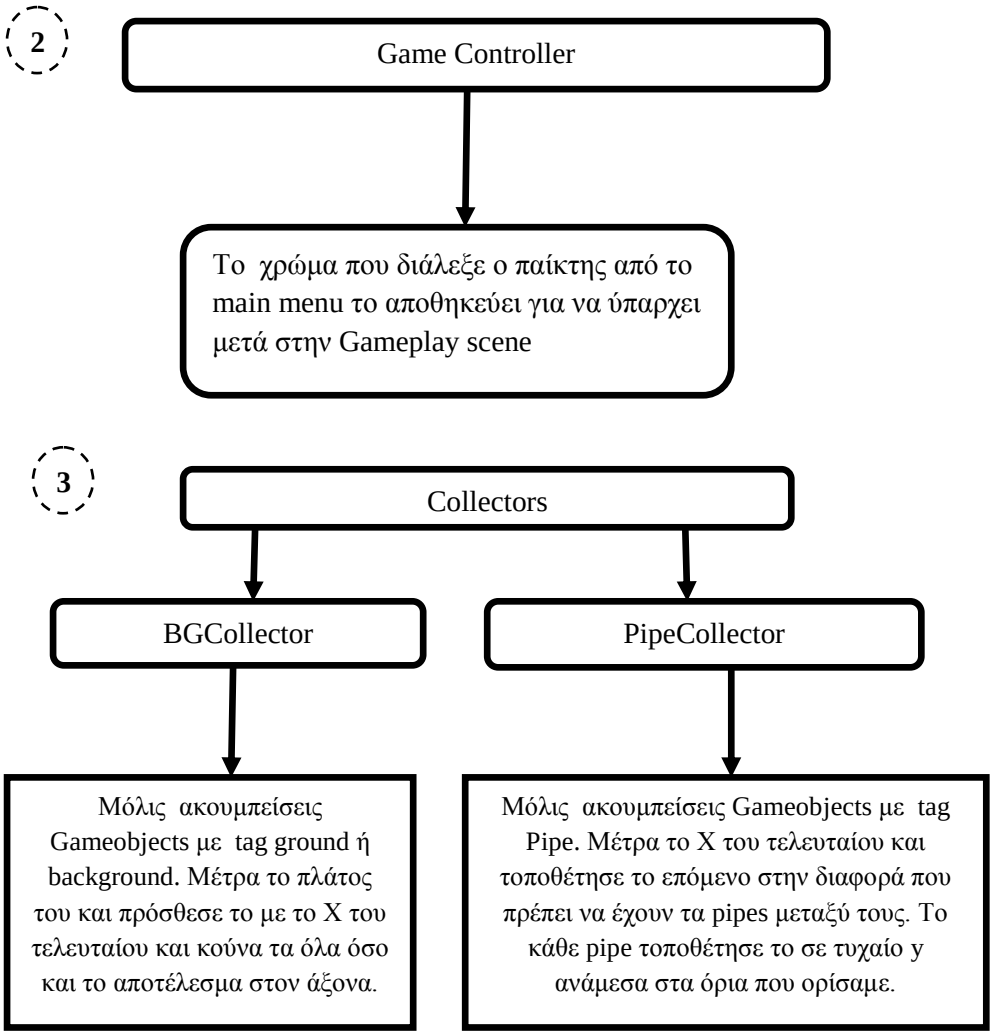
- Το GameplayController script είναι υπεύθυνο για το pause menu του παιχνιδιού, να δίνει ανάλογα το score του παίκτη το σωστό μετάλλιο, δείχνει το score που έκανε και το highscore στο game over screen.

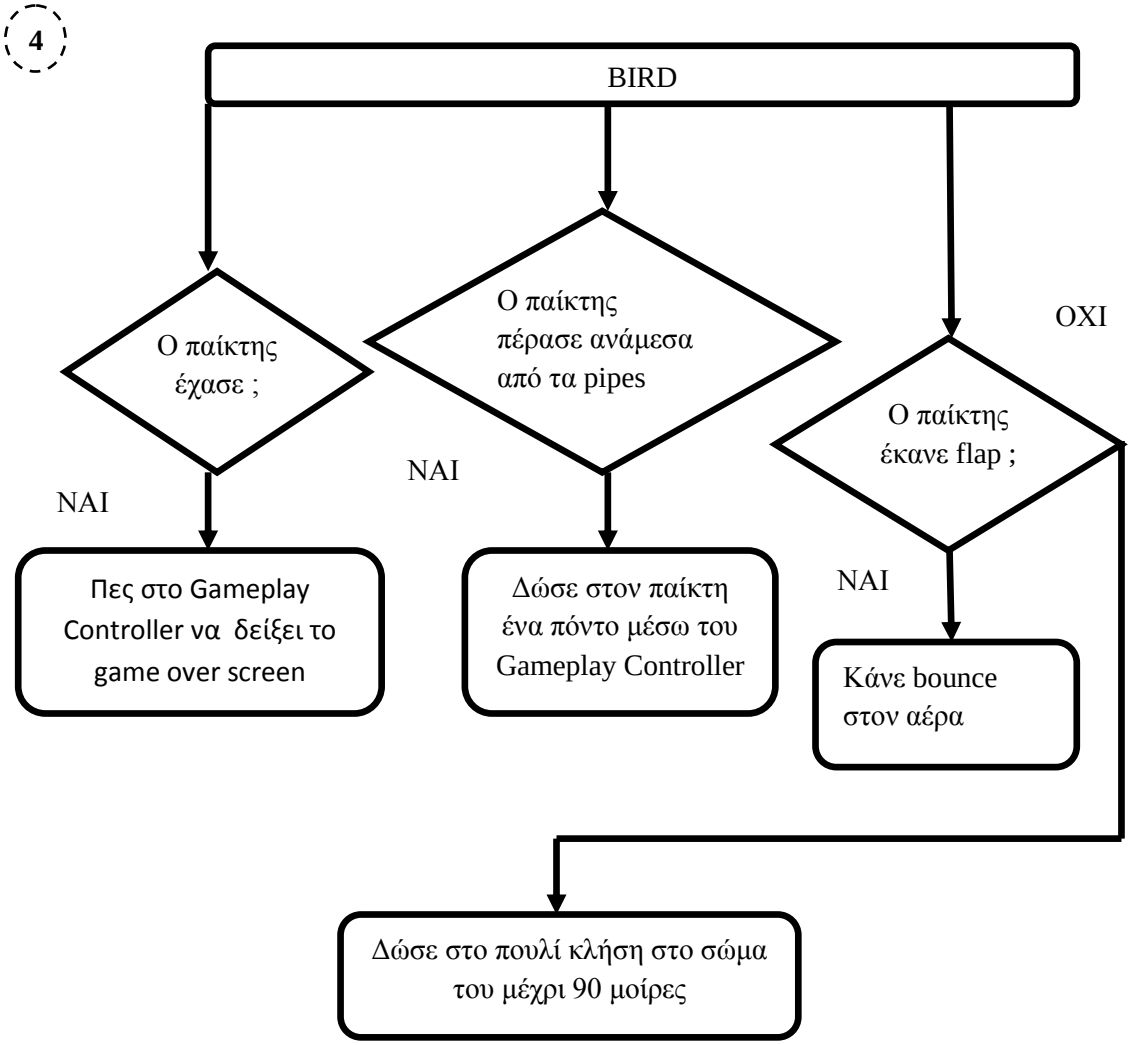
- Το GameController script είναι υπεύθυνο για να αποθηκεύει τις προτιμήσεις του παίκτη καθώς και το highscore του.
- Τα Collectors scripts που είναι το BGCollector script και το PipeCollector script. Το BGCollector script είναι υπεύθυνο να μαζεύει το background και το ground και να ξανατοποθετή καινούρια, για να δώσει στον παίκτη την ψευδαίσθηση ενός άπειρου background. Το PipeCollector είναι υπεύθυνο για παρόμεια δουλεία αλλά για τα εμπόδια που εμφανίζονται για να περνάει ενδιάμεσα ο παίκτης. Επίσης τοποθετή τα εμπόδια σε τυχαίο ύψος για να μην υπάρχει μονοτονία και επαναληπτικότητα στα εμπόδια που εμφανίζονται.
- Το Bird script δίνει την δυνατότητα στο παίκτη να κάνει το πουλί να πετά. Επίσης είναι το υπεύθυνο script για ξέρει το πρόγραμμα πότε ακούμπησε ο παίκτης το έδαφος ή ένα εμπόδιο για να του βγάλει το game over screen.
- Το camera script είναι υπεύθυνο να κουνάει την κάμερα για όσο ο παίκτης παραμένει ζωντανός.




Gameplay Scene
4
3
2
1
Bird
Collectors
GameController
GameplayController


1
GameplayController
Pause Button
ο παίκτης
είναι ζωντανός
ΝΑΙ
Ενεργοποίησε το pause
menu δείξε στον
παίκτη το score του
και το highscore του
Ο παίκτης έχασε βγάλε
το game over screen
MENU
Ποιο
κουμπί
πάτησε
Γύρνα στο
Main Menu
RESTART
Φόρτωσε ξανά την
Gameplay Scene
Ποιο
κουμπί
πάτησε
RESUME
Συνέχισε
το παιχνίδι
MENU
Γύρνα στο
Main Menu
Ανάλογα το score του
παίκτη δώστου το
σωστό μετάλλιο


2
Game Controller
Το χρώμα που διάλεξε ο παίκτης από το main menu το αποθηκεύει για να ύπαρχει μετά στην Gameplay scene
3
Collectors
BGCollector
PipeCollector
Μόλις ακουμπείσεις Gameobjects με tag ground ή background. Μέτρα το πλάτος του και πρόσθεσε το με το X του τελευταίου και κούνα τα όλα όσο και το αποτέλεσμα στον άξονα.
Μόλις ακουμπείσεις Gameobjects με tag Pipe. Μέτρα το X του τελευταίου και τοποθέτησε το επόμενο στην διαφορά που πρέπει να έχουν τα pipes μεταξύ τους. Το κάθε pipe τοποθέτησε το σε τυχαίο y ανάμεσα στα όρια που ορίσαμε.


4
BIRD
Ο παίκτης έχασε ;
Ο παίκτης πέρασε ανάμεσα από τα pipes
Ο παίκτης έκανε flap ;
ΟΧΙ
ΝΑΙ
ΝΑΙ
ΝΑΙ
Πες στο Gameplay Controller να δείξει το game over screen
Δώσε στον παίκτη ένα πόντο μέσω του Gameplay Controller
Κάνε bounce στον αέρα
Δώσε στο πουλί κλήση στο σώμα του μέχρι 90 μοίρες

Επίλογος

Για την δημιουργία βιντεοπαιχνιδιών δυό πολύ βασικοί παράμετροι είναι η φορητότητα και η παραγωγικότητα. Για την παραγωγικότητα απαιτούνται εργαλεία που συνεργάζονται άψογα μεταξύ τους στην δημιουργία και ανάπτυξη του παιχνιδιού.

Στην εργασία αυτή δείξαμε πως η Unity μπορεί να προσφέρει στους προγραμματιστές ολά τα απαραίτητα εργαλεία για να δημιουργήσουν τα βιντεοπαιχνίδια τους καθώς και να τα δημοσιεύσουν σε πολλές διαφορετικές πλατφόρμες χωρίς να χρειαστεί να γίνουν μεγάλες αλλαγές στον κώδικα πχ σε smartphones με android ή iOS λογισμικό. Όλες οι αλληλεπιδράσεις των στοιχείων, η αλληλεπίδραση με τον παίκτη κτλ γίνεται μέσω της C# και για την ανάπτυξη των scripts χρησιμοποιείτε το Visual Studio καθώς διαθέτει. Το interface της Unity μπορεί να διαμορφωθεί πλήρως ανάλογα τις προτιμήσεις του προγραμματιστή, με αυτό τον τρόπο το interface της Unity μπορεί να έρθει σε μια πιο οικεία μορφή στον προγραμματιστή άμα έχει εμπειρία με άλλα game engine.

Μερικές ιδέες για περαιτέρω βελτιώσεις στα δύο βιντεοπαιχνίδια μπορεί να είναι:

- Εισαγωγή third-party plugins για να δώσουμε την δυνατότητα στον παίκτη να μοιράζεται το score στα social media.
- Ένας πίνακας βαθμολογίας οπού ο παίκτης να μπορεί να βλέπει το score των φίλων του σε σχέση με το δικό του.
- Δημιουργία νέων collectable με διαφορετικές ιδιότητες.
- Δημιουργία νέων γραφικών (skins) για το πως φαίνεται ο παίκτης μέσα στο παιχνίδι, διαφορετικά background για να σπάει την μονοτονία.
- Εισαγωγή διάφορων εορταστικών εκδηλώσεων με special rewards για τον παίκτη.
- Εισαγωγή διαφημίσεων ή ακομά και skins για τον παίκτη επί πληρωμή για να βγάλουμε κάποιο κέρδος από τα παιχνίδια αυτά.

Για κάποιον που θέλει να αρχίσει χωρίς καμία εμπειρία την ανάπτυξη παιχνιδιών η Unity ένα πολυεργαλείο που πρέπει να λάβει σοβαρά υπόψη του. Η Unity λόγω του ότι μας προσφέρει όλες τις δυνατότητες της με την δωρεάν έκδωση είναι ένα πολύ ελκυστικό εργαλείο για τους νέους προγραμματιστές. Επίσης διαθέτει ένα μεγάλο community, που μπορεί κανείς να βρει σε forums και να ανατρέξει σε αυτό για να βρει λύση στο πρόβλημα του. Τέλος η Unity ως ένα πολυεργαλείο τύπου ελβετικού σουγιά προσφέρει πολλές

δυνατότητες για την ανάπτυξη παιχνιδιών και εξαιτίας του μεγάλου αριθμού νέων προγραμματιστών, υπάχει μεγάλος αριθμός παιχνιδιών που έχουν φτιαχτεί με την χρήση της Unity. Αυτό μας οδηγεί στο συμπέρασμα ότι η Unity Game Engine είναι μια μηχανή που εμπνέει τους προγραμματιστές να δημιουργούν ακόμα και την περίοδο που έρχονται σε αρχική επαφή με το εργαλείο αυτό χωρίς να διαθέτουν τις απαραίτητες γνώσεις.

Πιστεύουμε και ελπίζουμε η πτυχιακή αυτή εργασία να εμπνεύσει περισσότερους επίδοξους προγραμματιστές να ασπαστούν αυτά τα εργαλεία και να δημιουργήσουν νέες πλατφόρμες και βιντεοπαιχνίδια για διασκέδαση ή/και μάθηση.

APPENDIX A Jack the giant Code

Σε αυτό το παράρτημα είναι όλα τα script για το παιχνίδι Jack the giant.

BGScaler script

```
using System.Collections;
using System.Collections.Generic;
using UnityEngine;
//ο σκοπος αυτού του script είναι να κάνει scale την εικόνα του background
//ανάλογα την οθόνη που τρέχει το παιχνίδι. Είναι ένας απλός τρόπος για να
//κάνουμε scale χωρίς ωστόσο να λειτουργεί πάντα σε μεγάλες οθόνες
public class BGScaler : MonoBehaviour {
	void Start () {
		SpriteRenderer sr = GetComponent <SpriteRenderer> ();
		Vector3 tempScale = transform.localScale;
		float width = sr.sprite.bounds.size.x;
		float worldHeight = Camera.main.orthographicSize * 2.0f;
		float worldWidth = worldHeight / Screen.height * Screen.width;
		tempScale.x = worldWidth / width;
		transform.localScale = tempScale;
	}
}//BGScaler
```

BGCollector script

```
using System.Collections;
using System.Collections.Generic;
using UnityEngine;
//ο σκοπός αυτού του script είναι όταν ο BGCollector ακουμπάει τα Game Object
//με tag Background να τα καταστρέφει
public class BGCollector : MonoBehaviour {
	void OnTriggerEnter2D(Collider2D target){
		if (target.tag == "Background") {
			target.gameObject.SetActive (false);
		}
	}
}// BGCollector
```

BGSpawner script

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

public class BGSpawner : MonoBehaviour {
	private GameObject[] backgrounds;
	private float lastY;
	void Start () {
		GetBackgroundsAndSetLastY ();
	}
//ο σκοπός αυτής της function είναι να βρεί στον άξονα Y
//την τιμή Y που έχει το τελευταίο Game Object με tag Background
	void GetBackgroundsAndSetLastY () {
		backgrounds = GameObject.FindGameObjectsWithTag ("Background");
		lastY = backgrounds [0].transform.position.y;
		for (int i = 1; i < backgrounds.Length; i++) {
			if (lastY > backgrounds [i].transform.position.y)
				lastY = backgrounds [i].transform.position.y;
```

```
                }
        }
//όταν ο BGSpawner ακουμπάει το Background με την τιμή Y που πήρε
//η προηγούμενη function τοποθετή καινούρια Backgrounds που αρχίζουν
//απο την τιμή Y
        void OnTriggerEnter2D(Collider2D target){
                if (target.transform.position.y == lastY) {
                        Vector3 temp = target.transform.position;
                        float height = ((BoxCollider2D)target).size.y;
                        for (int i = 0; i < backgrounds.Length; i++) {
                                if (!backgrounds [i].activeInHierarchy) {
                                        temp.y -= height;
                                        lastY = temp.y;
                                        backgrounds [i].transform.position = temp;
                                        backgrounds [i].SetActive (true);
                                }
                        }
                }
        }
}//BGSpawner
```

Camera script

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

public class CameraScript : MonoBehaviour {

        private float speed = 1f;
        private float accelaration = 0.2f;
        private float maxSpeed = 3.2f;

        private float easySpeed = 2.7f;
        private float mediumSpeed= 3.7f;
        private float hardSpeed = 4.2f;

        [HideInInspector]
        public bool moveCamera;
//θέτη την μέγιστη ταχύτητα που μπορεί να κινηθεί η κάμερα
//ανάλογα με την δυσκολία
        void Start () {
                if (GamePreferences.GetEasyDifficulty () == 1) {
                        maxSpeed = easySpeed;
                }
                if (GamePreferences.GetMediumDifficulty () == 1) {
                        maxSpeed = mediumSpeed;
                }
                if (GamePreferences.GetHardDifficulty () == 1) {
                        maxSpeed = hardSpeed;
                }
                moveCamera = true;
        }
        void Update () {//καλήτε κάθε 1 frame να κουνάει την κάμερα
                if (moveCamera) {
                        MoveCamera ();
                }
        }
//η παρακάτω function είναι υπεύθυνη για την μετακίνηση της κάμερας
        void MoveCamera(){
                Vector3 temp = transform.position;
                float oldY = temp.y;
```

```
                float newY = temp.y - (speed * Time.deltaTime);
                temp.y = Mathf.Clamp (temp.y, oldY, newY);
                transform.position = temp;
                speed += accelaration * Time.deltaTime;
                if (speed > maxSpeed)
                        speed = maxSpeed;
        }
}//Camera Script
```

CloudCollector script

```
using System.Collections;
using System.Collections.Generic;
using UnityEngine;
//ο σκοπός αυτού του script είναι όταν ο CloudCollector ακουμπάει τα Game Object
//με tag Cloud ή Deadly να τα καταστρέφει
public class CloudCollector : MonoBehaviour {

        void OnTriggerEnter2D(Collider2D target){
                if (target.tag == "Cloud" || target.tag == "Deadly") {
                        target.gameObject.SetActive (false);
                }
        }
} //CloudCollector
```

CloudSpawner script

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

public class CloudSpawner : MonoBehaviour {
        [SerializeField]
        private GameObject[] clouds;
        private float distanceBetweenClouds = 3f;
        private float minX,maxX;
        private float lastCloudPositionY;
        private float controlX;
        [SerializeField]
        private GameObject[] collectables;
        private GameObject player;
//στο void Awake function είναι όλες οι functions που θέλουμε να ξεκινάνε
//μόλις αρχίζει το παιχνίδι
        void Awake(){
                controlX = 0f;
                SetMinandMaxX ();
                CreateClouds ();
                player = GameObject.Find ("Player");
                for (int i = 0; i < collectables.Length; i++) {
                        collectables [i].SetActive (false);
                }
        }
        void Start(){//η void Start ξεκινάει μετά την void Awake
                PositionThePlayer ();
        }
//θέτη όρια για της αποσταστάσεις που πρέπει να έχουν τα σύννεφα
//μεταξύ τους στον άξονα X
void SetMinandMaxX(){
        Vector3 bounds = Camera.main.ScreenToWorldPoint (new Vector3 (Screen.width,
        Screen.height, 0));
                maxX = bounds.x - 0.5f;
                minX = -bounds.x + 0.5f;
        }
```

```
void shuffle(GameObject[] arrayToShuffle){//randomize the game obj spawn
        for(int i=0;i<arrayToShuffle.Length;i++){
                GameObject temp = arrayToShuffle [i];
                int random = Random.Range (i, arrayToShuffle.Length);
                arrayToShuffle [i] = arrayToShuffle [random];
                arrayToShuffle [random] = temp;
        }
}
void CreateClouds(){ //making new clouds
        shuffle (clouds);
        float positionY = 0f;
        for(int i = 0; i < clouds.Length; i++) {
                Vector3 temp = clouds [i].transform.position;
                temp.y = positionY;
                temp.x = Random.Range (minX, maxX);
                if (controlX == 0) { //control X axis
                        temp.x = Random.Range (0.0f, maxX);
                        controlX = 1;
                } else if (controlX == 1) {
                        temp.x = Random.Range (0.0f, minX);
                        controlX = 2;
                } else if (controlX == 2) {
                        temp.x = Random.Range (1.0f, maxX);
                        controlX = 3;
                } else if (controlX == 3) {
                        temp.x = Random.Range (-1.0f, minX);
                        controlX = 0;
                }

                lastCloudPositionY = positionY;
                clouds [i].transform.position = temp;
                positionY -= distanceBetweenClouds;
        }
}
void PositionThePlayer(){
GameObject[] darkClouds = GameObject.FindGameObjectsWithTag ("Deadly");
GameObject[] cloudsInGame = GameObject.FindGameObjectsWithTag ("Cloud");
for (int i = 0; i < darkClouds.Length; i++) { // change the position
                                              //of an obj with another
if (darkClouds [i].transform.position.y == 0f) {
                Vector3 t = darkClouds [i].transform.position;
        darkClouds [i].transform.position = new Vector3 (
                        cloudsInGame [0].transform.position.x,
                        cloudsInGame [0].transform.position.y
                        cloudsInGame [0].transform.position.z);
                        cloudsInGame [0].transform.position = t;
                }
        }
        Vector3 temp = cloudsInGame [0].transform.position;
        for (int i = 1;i<cloudsInGame.Length;i++){
                if(temp.y< cloudsInGame[i].transform.position.y){
                temp= cloudsInGame[i].transform.position;
                }
        }
        temp.y += 0.8f;
        player.transform.position = temp;
void OnTriggerEnter2D(Collider2D target){ //create new clouds
        if (target.tag == "Cloud" || target.tag == "Deadly") {
        if(target.transform.position.y == lastCloudPositionY){
                shuffle (clouds);
                shuffle (collectables);
        Vector3 temp = target.transform.position;
```

```
            for (int i = 0; i < clouds.Length; i++) {
                if (!clouds [i].activeInHierarchy) {
                if (controlX == 0) {
                temp.x = Random.Range (0.0f, maxX);
                controlX = 1;
                } else if (controlX == 1) {
                temp.x = Random.Range (0.0f, minX);
                controlX = 2;
                } else if (controlX == 2) {
                        temp.x = Random.Range (1.0f, maxX);
                        controlX = 3;
                } else if (controlX == 3) {
                        temp.x = Random.Range (-1.0f, minX);
                        controlX = 0;
                }
                        temp.y -= distanceBetweenClouds;
                        lastCloudPositionY = temp.y;
                        clouds [i].transform.position = temp;
                        clouds [i].SetActive (true);

int random = Random.Range (0, collectables.Length); //Βλεπει αμα εχουμε ασπρο
                                //συννεφο και τοποθετη απο πανω ενα collectable
        if (clouds [i].tag != "Deadly") {
        if (!collectables [random].activeInHierarchy) {
                Vector3 temp2 = clouds [i].transform.position;
                temp2.y += 0.7f;

if (collectables [random].tag == "Life") {
if (PlayerScore.lifeCount < 2) {   //Δεν αφηνει τον player να εχει πανω απο 2life
                        collectables [random].transform.position = temp2;
                        collectables [random].SetActive (true);
            }
                        } else {
                        collectables [random].transform.position = temp2;
                        collectables [random].SetActive (true);
                                                }
                                        }
                                }



                                }
                        }
                }
            }
        }
}// Cloudspawner
```

Collectable script

```
using System.Collections;
using System.Collections.Generic;
using UnityEngine;
//ο σκοπός αυτού του script είναι να καταστρέφει τα collectables που μπορεί
//να πάρει ο παίκτης όταν ο παίκτης τα ακουμπάει
public class CollectableScript : MonoBehaviour {

	void OnEnable(){
		Invoke ("DestroyCollectable",6f);
	}

	void DestroyCollectable(){// destroy a coin or a life when player touch it
		gameObject.SetActive(false);
	}

}// CollectableScript
```

GameManager script

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

public class GameManager : MonoBehaviour {
	public static GameManager instance;
	[HideInInspector]
	public bool gameStartedFromMainMenu,gameRestarAfterDeath;
	[HideInInspector]
	public int score, coinScore, lifeScore;
	void Awake () {
		MakeSingleton ();
	}
	void Start(){
		InitializeVariables ();
	}
	void OnEnable(){ //sub
		SceneManager.sceneLoaded += LevelFinishedLoading;
	}
	void OnDisable(){ //unsub
		SceneManager.sceneLoaded -= LevelFinishedLoading;
	}
	void LevelFinishedLoading(Scene scene,LoadSceneMode mode){//finding what
								//scene we are
		if(scene.name =="Gameplay"){
			if(gameRestarAfterDeath){
				GameplayController.instance.SetScore (score);
				GameplayController.instance.SetCoinScore (coinScore);
				GameplayController.instance.SetLifeScore (lifeScore);

				PlayerScore.scoreCount = score;
				PlayerScore.coinCount = coinScore;
				PlayerScore.lifeCount = lifeScore;
			} else if (gameStartedFromMainMenu) {
					PlayerScore.coinCount = 0;
					PlayerScore.scoreCount = 0;
					PlayerScore.lifeCount = 2;

					GameplayController.instance.SetScore (0);
```

```
                                GameplayController.instance.SetCoinScore (0);
                                GameplayController.instance.SetLifeScore (2);
                        }
                }
        }
        void InitializeVariables(){//Saving the initialize preferences of player
                if (!PlayerPrefs.HasKey ("Game Initialized")) {
                        GamePreferences.SetEasyDifficulty (0);
                        GamePreferences.SetEasyDifficultyCoinScore (0);
                        GamePreferences.SetEasyDifficultyHighScore (0);

                        GamePreferences.SetMediumDifficulty (1);
                        GamePreferences.SetMediumDifficultyCoinScore (0);
                        GamePreferences.SetMediumDifficultyHighScore (0);

                        GamePreferences.SetHardDifficulty (0);
                        GamePreferences.SetHardDifficultyCoinScore (0);
                        GamePreferences.SetHardDifficultyHighScore (0);

                        GamePreferences.SetMusicState (0);

                        PlayerPrefs.SetInt ("Game Initialized", 123);
                }
        }
        void MakeSingleton () {
                if (instance != null) {
                        Destroy (gameObject);
                } else {
                        instance = this;
                        DontDestroyOnLoad (gameObject);
                }
        }
        public void CheckGameStatus(int score,int coinScore,int lifeScore){
if (lifeScore < 0) {
        if (GamePreferences.GetEasyDifficulty () == 1) {//Βλεπει τι difficulty
                                                //εχουμε και αναλογα βαζει το highscore
        int highScore = GamePreferences.GetEasyDifficultyHighScore ();
        int coinHighScore = GamePreferences.GetEasyDifficultyCoinScore ();
        if (highScore < score)
                GamePreferences.SetEasyDifficultyHighScore (score);
        if (coinHighScore < coinScore)
                GamePreferences.SetEasyDifficultyCoinScore (coinScore);
                        }
if (GamePreferences.GetMediumDifficulty () == 1) {//Βλεπει τι difficulty εχουμε και
                                                        //αναλογα βαζει το highscore
        int highScore = GamePreferences.GetMediumDifficultyHighScore ();
        int coinHighScore = GamePreferences.GetMediumDifficultyCoinScore ();
        if (highScore < score)
                GamePreferences.SetMediumDifficultyHighScore (score);
        if (coinHighScore < coinScore)
                GamePreferences.SetMediumDifficultyCoinScore (coinScore);
                        }
if (GamePreferences.GetHardDifficulty () == 1) {//Βλεπει τι difficulty εχουμε και
                                                        //αναλογα βαζει το highscore
        int highScore = GamePreferences.GetHardDifficultyHighScore ();
        int coinHighScore = GamePreferences.GetHardDifficultyCoinScore ();
        if (highScore < score)
        GamePreferences.SetHardDifficultyHighScore (score);
        if (coinHighScore < coinScore)
        GamePreferences.SetHardDifficultyCoinScore (coinScore);
        }
        gameStartedFromMainMenu = false;
```

```
        gameRestarAfterDeath = false;
        GameplayController.instance.GameOverShowPanel (score, coinScore);
                } else {
                        this.score = score;
                        this.coinScore = coinScore;
                        this.lifeScore = lifeScore;
                        gameStartedFromMainMenu = false;
                        gameRestarAfterDeath = true;
                        GameplayController.instance.PlayerDiedRestartGame ();
                }
        }
}//GameManager
```

GameplayController script

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

public class GameplayController : MonoBehaviour {
        public static GameplayController instance;//make it possible to have access
                                                  //in the class variables
[SerializeField]
private Text scoreText, coinText, lifeText, gameOverScoreText, gameOverCoinText;
[SerializeField]
private GameObject pausePanel, gameOverPanel;
[SerializeField]
private GameObject readyButton;
        void Awake () {
                MakeInstance ();
        }
        void Start(){
                Time.timeScale = 0f;
        }
        void MakeInstance () {
                if (instance == null) {
                        instance = this;
                }
        }
        public void GameOverShowPanel(int finalScore,int finalCoinScore){
                gameOverPanel.SetActive (true);
                gameOverScoreText.text = finalScore.ToString ();
                gameOverCoinText.text = finalCoinScore.ToString ();
                StartCoroutine (GameOverLoadMainMenu ());
        }
        IEnumerator GameOverLoadMainMenu(){
                yield return new WaitForSeconds (5f);
                SceneManager.LoadScene ("MainMenu");
        }
        public void PlayerDiedRestartGame(){
                StartCoroutine (PlayerDiedRestart ());
        }
        IEnumerator PlayerDiedRestart(){
                yield return new WaitForSeconds (1f);
                SceneManager.LoadScene ("Gameplay");
        }
        public void SetScore(int score){
                scoreText.text = "x" + score;
        }
```

```
        public void SetCoinScore(int cointScore){
                coinText.text = "x" + cointScore;
        }
        public void SetLifeScore(int lifeScore){
                lifeText.text = "x" + lifeScore;
        }
        public void PauseGame(){ //pause the game
                Time.timeScale = 0f;
                pausePanel.SetActive (true);
        }
        public void ResumeGame(){//resume the game
                Time.timeScale = 1f;
                pausePanel.SetActive (false);
        }
        public void QuitGame(){ // Quit the game to the main menu
                Time.timeScale = 1f;
                SceneManager.LoadScene ("MainMenu");
        }
        public void StartGame(){// Preparing the player to start the game
                Time.timeScale = 1f;
                readyButton.SetActive (false);
        }
}//GameplayController
```

HighscoreController script

```
using System.Collections;
using System.Collections.Generic;
using UnityEngine.SceneManagement;
using UnityEngine;
using UnityEngine.UI;
//ο σκοπός αυτού του script είναι όταν πάμε στην highscore οθόνη
//ανάλογα το επίπεδο δυσκολίας που έχουμε να μας δείχνει το highscore
public class HighscoreController : MonoBehaviour {
        [SerializeField]
        private Text scoreText, coinText ;
        void Start () {
        SetScoreBasedOnDifficulty ();
        }
        void SetScore(int score,int coinScore){
                scoreText.text = score.ToString ();
                coinText.text = coinScore.ToString ();
        }
void SetScoreBasedOnDifficulty(){
if (GamePreferences.GetEasyDifficulty () == 1) {
        SetScore (GamePreferences.GetEasyDifficultyHighScore(),
        GamePreferences.GetEasyDifficultyCoinScore ());
                }
if (GamePreferences.GetMediumDifficulty () == 1) {
        SetScore (GamePreferences.GetMediumDifficultyHighScore (),
                  GamePreferences.GetMediumDifficultyCoinScore ());
                }
if (GamePreferences.GetHardDifficulty () == 1) {
        SetScore (GamePreferences.GetHardDifficultyHighScore (),
                  GamePreferences.GetHardDifficultyCoinScore ());
                }

        }
        public void LoadMainMenu() {
                SceneManager.LoadScene("MainMenu");
        }
}//HighscoreController
```

MainMenuController script

```
using System.Collections;
using System.Collections.Generic;
using UnityEngine.SceneManagement;
using UnityEngine;
using UnityEngine.UI;
public class MainMenuController : MonoBehaviour {
	[SerializeField]
	private Button musicBtn;
	[SerializeField]
	private Sprite[] musicIcons;
	void Start () {
		CheckToPlayTheMusic ();
	}
	void CheckToPlayTheMusic(){
		if (GamePreferences.GetMusicState () == 1) {
			MusicController.instance.Playmusic (true);
			musicBtn.image.sprite = musicIcons [1];
		} else {
			MusicController.instance.Playmusic (false);
			musicBtn.image.sprite = musicIcons [0];
		}
	}
	public void StartGame(){
		GameManager.instance.gameStartedFromMainMenu = true;
		SceneManager.LoadScene ("Gameplay");
	}
	public void HighscoreMenu(){
	SceneManager.LoadScene ("Highscore");
	}
	public void OptionsMenu(){
		SceneManager.LoadScene ("OptionsMenu");
	}
	 public void QuitGame(){
		Application.Quit();
	}
	public void MusicButton(){
		if (GamePreferences.GetMusicState () == 0) {
			GamePreferences.SetMusicState (1);
			MusicController.instance.Playmusic (true);
			musicBtn.image.sprite = musicIcons [1];
		} else if(GamePreferences.GetMusicState()==1) {
			GamePreferences.SetMusicState (0);
			MusicController.instance.Playmusic(false);
			musicBtn.image.sprite = musicIcons[0];
		}
	}
}// MainMenuController
```

MusicController script

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

public class MusicController : MonoBehaviour {
	public static MusicController instance;
	private AudioSource audioSource;
```

```
        void Awake () {
                MakeSingleton ();
                audioSource = GetComponent<AudioSource> ();
        }
        void MakeSingleton(){
                if (instance != null) {
                        Destroy (gameObject);
                }else {
                        instance = this;
                        DontDestroyOnLoad (gameObject);
                }
        }

        public void Playmusic(bool play){
        if (play) {
                if (!audioSource.isPlaying) {
                        audioSource.Play ();
                        }
        }else {
                if (audioSource.isPlaying) {
                        audioSource.Stop ();
                                }
                        }
                }
}//MusicController
```

OptionsController script

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

public class OptionsController : MonoBehaviour {
        [SerializeField]
        private GameObject easySign,mediumSign,hardSign;
        void Start () {
                SetTheDifficulty ();
        }
        void SetInitialDifficulty(string difficulty){
                switch (difficulty) {
                case "easy":
                        mediumSign.SetActive (false);
                        hardSign.SetActive (false);
                        break;
                case"medium":
                        easySign.SetActive (false);
                        hardSign.SetActive (false);
                        break;
                case"hard":
                        easySign.SetActive (false);
                        mediumSign.SetActive (false);
                        break;
                }
        }
        void SetTheDifficulty(){ //Βρισκει ποια δυσκολια επελεξε ο player
                if (GamePreferences.GetEasyDifficulty () == 1) {
                        SetInitialDifficulty ("easy");
                }
                if (GamePreferences.GetMediumDifficulty () == 1) {
                        SetInitialDifficulty ("medium");
                }
```

```
            if (GamePreferences.GetHardDifficulty () == 1) {
                SetInitialDifficulty ("hard");
            }
        }
        public void EasyDifficulty(){
            GamePreferences.SetEasyDifficulty (1);
            GamePreferences.SetMediumDifficulty (0);
            GamePreferences.SetHardDifficulty (0);

            easySign.SetActive (true);
            mediumSign.SetActive (false);
            hardSign.SetActive (false);
        }
        public void MediumDifficulty(){
            GamePreferences.SetEasyDifficulty (0);
            GamePreferences.SetMediumDifficulty (1);
            GamePreferences.SetHardDifficulty (0);

            easySign.SetActive (false);
            mediumSign.SetActive (true);
            hardSign.SetActive (false);
        }
        public void HardDifficulty(){
            GamePreferences.SetEasyDifficulty (0);
            GamePreferences.SetMediumDifficulty (0);
            GamePreferences.SetHardDifficulty (1);

            easySign.SetActive (false);
            mediumSign.SetActive (false);
            hardSign.SetActive (true);
        }
        public void LoadMainMenu(){
            SceneManager.LoadScene("MainMenu");
        }
}//OptionsController
```

GamePreferences script

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

public static class GamePreferences {
public static string EasyDifficulty = "EasyDifficulty";
public static string MediumDifficulty = "MediumDifficulty";
public static string HardDifficulty = "HardDifficulty";
public static string EasyDifficultyHighScore = "EasyDifficultyHighScore";
public static string MediumDifficultyHighScore = "MediumDifficultyHighScore";
public static string HardDifficultyHighScore = "HardDifficultyHighScore";
public static string EasyDifficultyCoinScore = "EasyDifficultyCoinScore";
public static string MediumDifficultyCoinScore = "MediumDifficultyCoinScore";
        public static string HardDifficultyCoinScore = "HardDifficultyCoinScore";
        public static string IsMusicOn = "IsMusicOn";
        //NOTE θα χρησιμοποιήσουμε ακέραιους αριθμούς για να αναπαραστήσουμε τις
        //μεταβλητές boolean
        //0-false,1-true
        public static int GetMusicState(){//music
        return PlayerPrefs.GetInt (GamePreferences.IsMusicOn);
        }
        public static void SetMusicState(int state){
        PlayerPrefs.SetInt (GamePreferences.IsMusicOn, state);
        }
```

```
//easy mode
public static void SetEasyDifficulty(int difficulty){
PlayerPrefs.SetInt (GamePreferences.EasyDifficulty, difficulty);
}
public static int GetEasyDifficulty(){
return PlayerPrefs.GetInt (GamePreferences.EasyDifficulty);
}
//medium mode
public static void SetMediumDifficulty(int difficulty){
PlayerPrefs.SetInt (GamePreferences.MediumDifficulty, difficulty);
}
public static int GetMediumDifficulty(){
return PlayerPrefs.GetInt (GamePreferences.MediumDifficulty);
}
//hard mode
public static void SetHardDifficulty(int difficulty){
PlayerPrefs.SetInt (GamePreferences.HardDifficulty, difficulty);
}
public static int GetHardDifficulty(){
return PlayerPrefs.GetInt (GamePreferences.HardDifficulty);
}

//easy mode highscore
public static void SetEasyDifficultyHighScore(int difficulty){
PlayerPrefs.SetInt (GamePreferences.EasyDifficultyHighScore, difficulty);
}
public static int GetEasyDifficultyHighScore(){
return PlayerPrefs.GetInt (GamePreferences.EasyDifficultyHighScore);
}
//medium mode highscore
public static void SetMediumDifficultyHighScore(int difficulty){
PlayerPrefs.SetInt (GamePreferences.MediumDifficultyHighScore, difficulty);
}
public static int GetMediumDifficultyHighScore(){
return PlayerPrefs.GetInt (GamePreferences.MediumDifficultyHighScore);
}
//hard mode highscore
public static void SetHardDifficultyHighScore(int difficulty){
PlayerPrefs.SetInt (GamePreferences.HardDifficultyHighScore, difficulty);
}
public static int GetHardDifficultyHighScore(){
return PlayerPrefs.GetInt (GamePreferences.HardDifficultyHighScore);
}

//easy mode coin score
public static void SetEasyDifficultyCoinScore(int difficulty){
PlayerPrefs.SetInt (GamePreferences.EasyDifficultyCoinScore, difficulty);
}
public static int GetEasyDifficultyCoinScore(){
return PlayerPrefs.GetInt (GamePreferences.EasyDifficultyCoinScore);
}
//medium mode coin score
public static void SetMediumDifficultyCoinScore(int difficulty){
PlayerPrefs.SetInt (GamePreferences.MediumDifficultyCoinScore, difficulty);
}
public static int GetMediumDifficultyCoinScore(){
return PlayerPrefs.GetInt (GamePreferences.MediumDifficultyCoinScore);
}
//hard mode coin score
public static void SetHardDifficultyCoinScore(int difficulty){
PlayerPrefs.SetInt (GamePreferences.HardDifficultyCoinScore, difficulty);
}
```

```
	public static int GetHardDifficultyCoinScore(){
	return PlayerPrefs.GetInt (GamePreferences.HardDifficultyCoinScore);
	}
}// Game Preferences
```

Joystick script

```
using System.Collections;
using System.Collections.Generic;
using UnityEngine;
using UnityEngine.EventSystems;
//ο σκοπός αυτού του script είναι να δώσουμε ιδιότητες στα κουμπιά
//που έχουμε βάλει για να κουνάμε τον παίχτη χωρίς την χρήση του keyboard
public class Joystick : MonoBehaviour,IPointerUpHandler,IPointerDownHandler {
	private PlayerMoveJoystick playerMove;
	void Start(){
	playerMove = GameObject.Find ("Player").GetComponent<PlayerMoveJoystick> ();
	}
	public void OnPointerDown(PointerEventData data){
		if (gameObject.name == "Left") {
			playerMove.SetMoveLeft (true);
		} else {
			playerMove.SetMoveLeft (false);
		}
	}
	public void OnPointerUp(PointerEventData data){
		playerMove.StopMoving ();
	}
}//Joystick
```

Player script

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

public class Player : MonoBehaviour {
	public float speed = 8f, maxVelocity = 4f;
	private Rigidbody2D myBody;
	private Animator anim;
	void Awake(){
		myBody = GetComponent<Rigidbody2D> ();
		anim = GetComponent<Animator> ();
	}
	void Start () {

	}
	void FixedUpdate () {
		PlayerMoveKeyboard ();
	}
void PlayerMoveKeyboard(){ //κωδικας για να κουνάμε τον player με το keyboard
		float forceX = 0f;
		float vel = Mathf.Abs (myBody.velocity.x);
		float h = Input.GetAxisRaw ("Horizontal");
		//με το κουμπί a ή το αριστερό βέλος πάει αριστερά
		//με το κουμπί d ή το δεξί βέλο πάει δεξιά
		if (h > 0) {
			if (vel < maxVelocity)
			forceX = speed;
			Vector3 temp = transform.localScale;
```

```
                        temp.x = 1.3f;
                        transform.localScale = temp;
                        anim.SetBool ("PlayerWalk", true);
                } else if (h < 0) {
                        if (vel < maxVelocity)
                        forceX = -speed;
                        Vector3 temp = transform.localScale;
                        temp.x = -1.3f;
                        transform.localScale = temp;
                        anim.SetBool ("PlayerWalk", true);
                } else {
                        anim.SetBool ("PlayerWalk", false);
                }
                myBody.AddForce (new Vector2(forceX,0));
        }
}//Player script
```

PlayerBounds script

```
using System.Collections;
using System.Collections.Generic;
using UnityEngine;
//ο σκοπός αυτού του script είναι να αποτρέψουμε τον παίχτη
//να έχει την δυνατότητα να βγει απο την οθόνη
public class PlayerBounds : MonoBehaviour {
        private float minX, maxX;
        void Start () {
                SetMinAndMaxX ();
        }
        void Update () {
                if (transform.position.x < minX) {
                        Vector3 temp = transform.position;
                        temp.x = minX;
                        transform.position = temp;
                }
                if (transform.position.x > maxX) {
                        Vector3 temp = transform.position;
                        temp.x = maxX;
                        transform.position = temp;
                }
        }
        void SetMinAndMaxX(){
                Vector3 bounds = Camera.main.ScreenToWorldPoint (new Vector3
                (Screen.width, Screen.height, 0));
        maxX = bounds.x-0.5f;
        minX = -bounds.x+0.5f;
        }
}//PlayerBounds
```

PlayerMovejoystick script

```
using System.Collections;
using System.Collections.Generic;
using UnityEngine;
//ο σκοπός αυτού του script είναι να δώσουμε ιδιότητες στο παίχτη
//ανάλογα προς τα που κινείται να παίζει το σωστό animation
public class PlayerMoveJoystick : MonoBehaviour {
        public float speed = 8f, maxVelocity = 4f;
        private Rigidbody2D myBody;
        private Animator anim;
```

```
        void Awake(){
                myBody = GetComponent<Rigidbody2D> ();
                anim = GetComponent<Animator> ();

        }
        private bool moveLeft,moveRight;
        void Update () {
                if (moveLeft) {
                        MoveLeft ();
                }
                if (moveRight) {
                        MoveRight ();
                }
        }
        public void SetMoveLeft(bool moveLeft){
                this.moveLeft = moveLeft;
                this.moveRight = !moveLeft;
        }
        public void StopMoving(){
                moveLeft = moveRight = false;
                anim.SetBool ("PlayerWalk", false);
        }
        void MoveLeft(){
                float forceX = 0f;
                float vel = Mathf.Abs (myBody.velocity.x);
                if (vel < maxVelocity)
                        forceX = -speed;
                Vector3 temp = transform.localScale;
                temp.x = -1.3f;
                transform.localScale = temp;
                anim.SetBool ("PlayerWalk", true);
                myBody.AddForce (new Vector2 (forceX, 0));
        }
        void MoveRight(){
                float forceX = 0f;
                float vel = Mathf.Abs (myBody.velocity.x);
                if (vel < maxVelocity)
                        forceX = speed;
                Vector3 temp = transform.localScale;
                temp.x = 1.3f;
                transform.localScale = temp;
                anim.SetBool ("PlayerWalk", true);

                myBody.AddForce (new Vector2 (forceX, 0));
        }
}//PlayerMoveJoystick
```

PlayerScore script

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

public class PlayerScore : MonoBehaviour {
        [SerializeField]
        private AudioClip coinClip,lifeClip;
        private CameraScript cameraScript;
        private Vector3 previousPosition;
        private bool countScore;
        public static int scoreCount;
        public static int lifeCount;
```

```
        public static int coinCount;
        void Awake(){
                cameraScript = Camera.main.GetComponent<CameraScript> ();
        }
        void Start () {
                previousPosition = transform.position;
                countScore = true;
        }


        void Update () {
                CountScore ();
        }
        void CountScore(){//Βλεπει αμα ο παικτης κουνιετε και προσθετη ποντους
                if (countScore) {
                        if (transform.position.y < previousPosition.y) {
                        scoreCount++;
                        }
                        previousPosition = transform.position;
                        GameplayController.instance.SetScore (scoreCount);
                }
        }
        void OnTriggerEnter2D(Collider2D target){ //function for the coin obj
                if (target.tag == "Coin") {
                        coinCount++;
                        scoreCount += 200;
                        GameplayController.instance.SetScore (scoreCount);
                        GameplayController.instance.SetCoinScore (coinCount);
                        AudioSource.PlayClipAtPoint (coinClip, transform.position);
                        target.gameObject.SetActive (false);
                }
        if (target.tag == "Life") {//function for the life obj
                        lifeCount++;
                        scoreCount += 300;
                        GameplayController.instance.SetScore (scoreCount);
                        GameplayController.instance.SetLifeScore (lifeCount);
                        AudioSource.PlayClipAtPoint (lifeClip, transform.position);
                        target.gameObject.SetActive (false);
                }
        //function to see if the player is out of bounds or touching a dark cloud
        if (target.tag == "Bounds"|| target.tag=="Deadly") {
        cameraScript.moveCamera = false;
        countScore = false;
        transform.position = new Vector3 (500,500,0);
        lifeCount--;  //if player dies reposition the player
        GameManager.instance.CheckGameStatus (scoreCount, coinCount, lifeCount);
                }
        }
}//PlayerScore
```

APPENDIX B Flappy Bird Code

Σε αυτό το παράρτημα είναι όλα τα script για το παιχνίδι Flappy Bird.

Bird script

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

public class BirdScript : MonoBehaviour {
    public static BirdScript instance;
    [SerializeField]
    private Rigidbody2D myRigidBody;
    [SerializeField]
    private Animator anim;
    private float forwardSpeed = 3f, bounceSpeed = 4f;
    private bool didFlap;
    public bool isAlive;
    private Button flapButton;
    [SerializeField]
    private AudioSource audioSource;
    [SerializeField]
    private AudioClip flapClick, pointClip, diedClip;
    public int score;
    void Awake(){
        if (instance == null) {
            instance = this;
        }
        isAlive = true;
        score = 0;
flapButton = GameObject.FindGameObjectWithTag("FlapButton").GetComponent<Button>();
flapButton.onClick.AddListener(() => FlapTheBird());
        SetCameraX ();
    }
    void Start () {

       }
    void FixedUpdate () {//it's called every 2 or 3 frames not every frame
        if (isAlive) {
            Vector3 temp = transform.position;
            temp.x += forwardSpeed * Time.deltaTime;
            transform.position = temp;
            if (didFlap){
                didFlap = false;
                myRigidBody.velocity = new Vector2(0, bounceSpeed);
                audioSource.PlayOneShot(flapClick);
                anim.SetTrigger("Flap");
            }
            if (myRigidBody.velocity.y >= 0){
                transform.rotation = Quaternion.Euler(0, 0, 0);
            }else {
                float angle = 0;
angle = Mathf.Lerp(0, -90, -myRigidBody.velocity.y / 7);
//παιρνει τιμες απο 0 μεχρι -90 και ο τριτος αριθμος ειναι ο χρονος που δινουμαι
//για παει απο το 0 στο-90
         transform.rotation = Quaternion.Euler(0, 0, angle);
            }
        }
       }
```

```
    void SetCameraX() {//κανει το offset της καμερας
CameraScript.offsetX=(Camera.main.transform.position.x-transform.position.x)-1f;
    }
    public float GetPositionX() {
        return transform.position.x;
    }
    public void FlapTheBird() {
        didFlap = true;
    }
    void OnCollisionEnter2D(Collision2D target) {
    if(target.gameObject.tag == "Ground" || target.gameObject.tag == "Pipe"){
//κώδικας για να σκοτώνει τον παίχτη όταν ακουμπά GameObjects με tags ground or
//pipe
            if (isAlive) {
                isAlive = false;
                anim.SetTrigger("Bird Died");
                audioSource.PlayOneShot(diedClip);
                GameplayController.instance.PlayerDiedShowScore(score);
            }
        }
    }
    void OnTriggerEnter2D(Collider2D target){ //κώδικας για να αυξάνει το score του
                                              //παίχτη όταν παιρνάει ανάμεσα από τους σωλύνες
        if (target.tag == "PipeHolder") {
            score++;
            GameplayController.instance.SetScore(score);
            audioSource.PlayOneShot(pointClip);
        }
    }
}//BirdScript
```

Camera script

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

public class CameraScript : MonoBehaviour {

    public static float offsetX;
       void Start () {

       }
       void Update () {//οσο ο παίχτης είναι ζωντανός κούνα την κάμερα
        if (BirdScript.instance != null) {
            if (BirdScript.instance.isAlive) {
                MoveTheCamera();
            }
        }
       }
    void MoveTheCamera() {//κουνά την κάμερα στον άξονα X ανάλογα το X του παίκτη
        Vector3 temp = transform.position;
        temp.x = BirdScript.instance.GetPositionX() + offsetX;
        transform.position = temp;
    }
}//Camera Script
```

BGCollector script

```
using System.Collections;
using System.Collections.Generic;
using UnityEngine;
//ο σκοπός αυτού του script είναι οταν ακουμπάει ο BGCollector ta Game Obejct
//με tag Background ή ground να παίρνει απο το τελευταιο background ή ground
//που υπάρχει το X τους και να τοποθετη ένα καινουριο background ή ground
//που θα αρχίζει από το X αυτό
public class BGCollector : MonoBehaviour {
    private GameObject[] backgrounds;
    private GameObject[] grounds;
    private float lastBGX;
    private float lastGroundX;
       void Awake () {
        backgrounds = GameObject.FindGameObjectsWithTag("Background");
        grounds = GameObject.FindGameObjectsWithTag("Ground");
        lastBGX = backgrounds[0].transform.position.x;
        lastGroundX = grounds[0].transform.position.x;
        for (int i = 1; i < backgrounds.Length; i++) {
            if (lastBGX < backgrounds[i].transform.position.x) {
                lastBGX = backgrounds[i].transform.position.x;
            }
        }
        for (int i = 1; i < grounds.Length; i++){
            if (lastGroundX < grounds[i].transform.position.x){
                lastGroundX = grounds[i].transform.position.x;
            }
        }
    }
       void OnTriggerEnter2D(Collider2D target) {
        if (target.tag == "Background")
        {
            Vector3 temp = target.transform.position;
            float width = ((BoxCollider2D)target).size.x;
            temp.x = lastBGX + width;
            target.transform.position = temp;
            lastBGX = temp.x;
        }
        else if (target.tag == "Ground") {
            Vector3 temp = target.transform.position;
            float width = ((BoxCollider2D)target).size.x;
            temp.x = lastGroundX + width;
            target.transform.position = temp;
            lastGroundX = temp.x;
        }
       }
}// BG Collector
```

PipeCollector script

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

public class PipeCollector : MonoBehaviour {

    private GameObject[] pipeHolders;
    private float distance = 3f;
    private float lastPipesX;
    private float pipeMin = -1.92f;
    private float pipeMax = 2.43f;
```

```
    private void Awake(){
        pipeHolders = GameObject.FindGameObjectsWithTag("PipeHolder");
       //παιρνει ολα τα obj me tag PipeHolder
        for (int i = 0; i < pipeHolders.Length; i++) {
            Vector3 temp = pipeHolders[i].transform.position;
            temp.y = Random.Range(pipeMin, pipeMax);
       //περνει έναν αριθμο απο το pipeMin μεχρι το pipeMax
            pipeHolders[i].transform.position = temp;
        }
        lastPipesX = pipeHolders[0].transform.position.x;
        for (int i = 1; i < pipeHolders.Length; i++) {
            if (lastPipesX < pipeHolders[i].transform.position.x) {
                lastPipesX = pipeHolders[i].transform.position.x;

            }
        }

    }
    private void OnTriggerEnter2D(Collider2D target){
        if (target.tag == "PipeHolder") {
            Vector3 temp = target.transform.position;
            temp.x = lastPipesX + distance;
            temp.y = Random.Range(pipeMin, pipeMax);
            target.transform.position = temp;
            lastPipesX = temp.x;

        }
    }
}//PipeCollector
```

GameController script

```
using System.Collections;
using System.Collections.Generic;
using UnityEngine;
using UnityEngine.UI;
public class GameController : MonoBehaviour{
    public static GameController instance;
   //const ειναι μεταβλητη που δεν αλλαζει
    private const string HIGH_SCORE = "High Score";
    private const string SELECTED_BIRD = "Selected Bird";
    private const string GREEN_BIRD = "Green Bird";
    private const string RED_BIRD = "Red Bird";
    void Awake(){
        MakeSingleton();
        IsTheGameStartedForTheFirstTime();
        //PlayerPrefs.DeleteAll();
    }
    // Use this for initialization
    void Start(){

    }
    void MakeSingleton(){//καταστρεφει το αρχειο αμα υπαρχει ειδη μια φορα και
    if (instance != null){//αμα δεν υπαρχει δημιουργει ενα καινουριο αρχειο
            Destroy(gameObject);
        }else{
            instance = this;
```

```
            DontDestroyOnLoad(gameObject);
        }
    }
    void IsTheGameStartedForTheFirstTime(){
        if (!PlayerPrefs.HasKey("IsTheGameStartedForTheFirstTime")){
            PlayerPrefs.SetInt(HIGH_SCORE, 0);
            PlayerPrefs.SetInt(SELECTED_BIRD, 0);
            PlayerPrefs.SetInt(GREEN_BIRD, 1);
            PlayerPrefs.SetInt(RED_BIRD, 1);
            PlayerPrefs.SetInt("IsTheGameStartedForTheFirstTime", 0);
        }
    }
    public void SetHighscore(int score){
        PlayerPrefs.SetInt(HIGH_SCORE, score);
    }
    public int GetHighscore()    {
        return PlayerPrefs.GetInt(HIGH_SCORE);
    }
    public void SetSelectedBird(int selectedBird){
        PlayerPrefs.SetInt(SELECTED_BIRD, selectedBird);
    }
    public int GetSelectedBird(){
        return PlayerPrefs.GetInt(SELECTED_BIRD);
    }
    public void UnlockGreenBird(){
        PlayerPrefs.SetInt(GREEN_BIRD, 1);
    }
    public int IsGreenBirdUnlocked(){
        return PlayerPrefs.GetInt(GREEN_BIRD);
    }
    public void UnlockRedbird(){
        PlayerPrefs.SetInt(RED_BIRD, 1);
    }
    public int IsRedBirdUnlocked(){
        return PlayerPrefs.GetInt(RED_BIRD);
    }
}//Game Controller
```

GameplayController script

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

public class GameplayController : MonoBehaviour {

    public static GameplayController instance;

    [SerializeField]
    private Text scoreText, endScore, bestScore, gameOverText;
    [SerializeField]
    private Button restartGameButton, instructionsButton;
    [SerializeField]
    private GameObject pausePanel;
    [SerializeField]
    private GameObject[] birds;
    [SerializeField]
    private Sprite[] medals;
    [SerializeField]
    private Image medalImage;
    void Awake () {
```

```
        MakeInstance();
        Time.timeScale = 0f;
       }
    void MakeInstance() {
        if (instance == null) {
            instance = this;
        }
    }
    public void PauseGame() {
        if (BirdScript.instance != null) {
        if (BirdScript.instance.isAlive) {
pausePanel.SetActive(true);
gameOverText.gameObject.SetActive(false);
endScore.text = "" + BirdScript.instance.score;
bestScore.text = "" + GameController.instance.GetHighscore();
Time.timeScale = 0f;
restartGameButton.onClick.RemoveAllListeners();//βγαζουμε ολες τις ιδιωτητες που
       //ειχε ποιο πριν,αμα ειχε δυο ιδιωτητες το κουμπι θα εκτελουσε 2 functions
restartGameButton.onClick.AddListener(() => ResumeGame());
       //προσθετουμε την ιδιωτητα να κανει resume to game
            }
        }
    }
    public void GoToMenuButton() {
        //SceneFader.instance.FadeIn("MainMenu");
        SceneManager.LoadScene("MainMenu");
    }
    public void ResumeGame() {
        pausePanel.SetActive(false);
        Time.timeScale = 1f;
    }
    public void RestartGame() {
        SceneManager.LoadScene("Gameplay");
    }
    public void PlayGame() {
        scoreText.gameObject.SetActive(true);
        birds[GameController.instance.GetSelectedBird()].SetActive(true);
        instructionsButton.gameObject.SetActive(false);
        Time.timeScale = 1f;
    }
    public void SetScore(int score) {
        scoreText.text = "" + score;
    }
    public void PlayerDiedShowScore(int score) {
        pausePanel.SetActive(true);
        gameOverText.gameObject.SetActive(true);
        scoreText.gameObject.SetActive(false);
        endScore.text = "" + score;
        if (score > GameController.instance.GetHighscore()) {
            GameController.instance.SetHighscore(score);
        }
        bestScore.text = "" + GameController.instance.GetHighscore();

        if (score <= 20) {
            medalImage.sprite = medals[0];
        }else if(score > 20 && score < 40) {
            medalImage.sprite = medals[1];
            if (GameController.instance.IsGreenBirdUnlocked() == 0) {
                GameController.instance.UnlockGreenBird();
            }
        }
        else {
```

```
            medalImage.sprite = medals[2];
            if (GameController.instance.IsGreenBirdUnlocked() == 0) {
                GameController.instance.UnlockGreenBird();
            }
            if (GameController.instance.IsRedBirdUnlocked() == 0) {
                GameController.instance.UnlockRedbird();
            }
        }
        restartGameButton.onClick.RemoveAllListeners();
        restartGameButton.onClick.AddListener(() => RestartGame());
    }
}//GameplayController
```

MenuController script

```
using System.Collections;
using System.Collections.Generic;
using UnityEngine;
using UnityEngine.SceneManagement;
public class MenuController : MonoBehaviour {
    public static MenuController instance;
    [SerializeField]
    private GameObject[] birds;
    private bool isGreenBirdUnlocked, isRedBirdUnlocked;
    void Awake(){
        MakeInstance();
    }
    // Use this for initialization
    void Start () {
        birds[GameController.instance.GetSelectedBird()].SetActive(true);
        CheckIfBirdsAreUnlocked();
       }
    void MakeInstance() {
        if (instance == null) {
            instance = this;
        }
    }
    void CheckIfBirdsAreUnlocked() {
        if (GameController.instance.IsRedBirdUnlocked() == 1) {
            isRedBirdUnlocked = true;
        }
        if (GameController.instance.IsGreenBirdUnlocked() == 1){
            isGreenBirdUnlocked = true;
        }
    }
    public void PlayGame() {
        //SceneFader.instance.FadeIn("Gameplay");
        SceneManager.LoadScene("Gameplay");
    }
    public void ChangeBird() {
        if (GameController.instance.GetSelectedBird() == 0){
            if (isGreenBirdUnlocked){
                birds[0].SetActive(false);
                GameController.instance.SetSelectedBird(1);
                birds[GameController.instance.GetSelectedBird()].SetActive(true);
            }
        }else if (GameController.instance.GetSelectedBird() == 1) {
            if (isRedBirdUnlocked) {
                birds[1].SetActive(false);
                GameController.instance.SetSelectedBird(2);
                birds[GameController.instance.GetSelectedBird()].SetActive(true);
            }
```

```
            else {
                birds[1].SetActive(false);
                GameController.instance.SetSelectedBird(0);
                birds[GameController.instance.GetSelectedBird()].SetActive(true);
            }
        }else if(GameController.instance.GetSelectedBird() == 2) {
            birds[2].SetActive(false);
            GameController.instance.SetSelectedBird(0);
            birds[GameController.instance.GetSelectedBird()].SetActive(true);
        }
    }
}//Menu Controller
```

Μεταφρασμένη, στην Ελληνική γλώσσα βιβλιογραφία

Bibliography

[1] Wikipedia, "C Sharp (programming language)-Wikipedia," [Online]. Available: https://en.wikipedia.org/wiki/C_Sharp_(programming_language). [Accessed 26 02 2018].

[2] Wikipedia, "Professional Developers Conference-Wikipedia," [Online]. Available: https://en.wikipedia.org/wiki/Professional_Developers_Conference. [Accessed 09 02 2018].

[3] Wikipedia, "Metaprogramming-Wikipedia," [Online]. Available: https://en.wikipedia.org/wiki/Metaprogramming. [Accessed 15 02 2018].

[4] Wikipedia, "Common Language Runtime-Wikipedia," [Online]. Available: https://en.wikipedia.org/wiki/Common_Language_Runtime. [Accessed 11 02 2018].

[5] Wikipedia, ".NET Framework-Wikipedia," [Online]. Available: https://en.wikipedia.org/wiki/.NET_Framework. [Accessed 05 03 2018].

[6] Wikipedia, "ADO.NET-Wikipedia," [Online]. Available: https://en.wikipedia.org/wiki/ADO.NET. [Accessed 26 02 2018].

[7] Wikipedia, "Anonymous type-Wikipedia," [Online]. Available: https://en.wikipedia.org/wiki/Anonymous_type. [Accessed 13 02 2018].

[8] Wikipedia, "Boilerplate Code-Wikipedia," [Online]. Available: https://en.wikipedia.org/wiki/Boilerplate_code. [Accessed 13 02 2018].

[9] Wikipedia, "Common Language Infrastructure-Wikipedia," [Online]. Available: https://en.wikipedia.org/wiki/Common_Language_Infrastructure. [Accessed 26 02 2018].

[10] Wikipedia, "Declarative programming-Wikipedia," [Online]. Available: https://en.wikipedia.org/wiki/Declarative_programming. [Accessed 14 02 2018].

[11] Wikipedia, "Extension method-Wikipedia," [Online]. Available: https://en.wikipedia.org/wiki/Extension_method. [Accessed 13 02 2018].

[12] Wikipedia, "Functional programming-Wikipedia," [Online]. Available: https://en.wikipedia.org/wiki/Functional_programming. [Accessed 26 02 2018].

[13] Wikipedia, "Imperative programming-Wikipedia," [Online]. Available: https://en.wikipedia.org/wiki/Imperative_programming. [Accessed 14 02 2018].

[14] Wikipedia, "Generic programming-Wikipedia," [Online]. Available: https://en.wikipedia.org/wiki/Generic_programming. [Accessed 13 02 2018].

[15] Wikipedia, "Language Integrated Query-Wikipedia," [Online]. Available: https://en.wikipedia.org/wiki/Language_Integrated_Query. [Accessed 14 02 2018].

[16] Wikipedia, "Programming paradigm-Wikipedia," [Online]. Available: https://en.wikipedia.org/wiki/Programming_paradigm. [Accessed 14 02 2018].

[17] Wikipedia, "Object-oriented programming-Wikipedia," [Online]. Available: https://en.wikipedia.org/wiki/Object-oriented_programming. [Accessed 14 02 2018].

[18] Wikipedia, "Reification (compiter science)-Wikipedia," [Online]. Available: https://en.wikipedia.org/wiki/Reification_(computer_science). [Accessed 13 02 2018].

[19] Wikipedia, "Strong and weak typing-Wikipedia," [Online]. Available: https://en.wikipedia.org/wiki/Strong_and_weak_typing. [Accessed 14 02 2018].

[20] Wikipedia, "Type inference-Wikipedia," [Online]. Available: https://en.wikipedia.org/wiki/Type_inference. [Accessed 13 02 2018].

[21] Wikipedia, "Unity (game engine)-Wikipedia," [Online]. Available: https://en.wikipedia.org/wiki/Unity_(game_engine). [Accessed 20 02 2018].

[22] Wikipedia, "Game Engine-Wikipedia," [Online]. Available: https://en.wikipedia.org/wiki/Game_engine. [Accessed 20 02 2018].

[23] Wikipedia, "Cross-platform- Wikipedia," [Online]. Available: https://en.wikipedia.org/wiki/Cross-platform. [Accessed 19 02 2018].

[24] Wikipedia, "Application programming interface-Wikipedia," [Online]. Available: https://en.wikipedia.org/wiki/Application_programming_interface. [Accessed 19 02 2018].

[25] Wikipedia, "Abstraction (software engineering)-Wikipedia," [Online]. Available: https://en.wikipedia.org/wiki/Abstraction_(software_engineering). [Accessed 19 02 2018].

[26] Wikipedia, "Direct3D-Wikipedia," [Online]. Available: https://en.wikipedia.org/wiki/Direct3D. [Accessed 20 02 2018].

[27] Wikipedia, "OpenGL-Wikipedia," [Online]. Available: https://en.wikipedia.org/wiki/OpenGL. [Accessed 20 02 2018].

[28] Wikipedia, "OpenGL ES-Wikipedia," [Online]. Available: https://en.wikipedia.org/wiki/OpenGL_ES. [Accessed 20 02 2018].

[29] Wikipedia, "WebGL-Wikipedia," [Online]. Available: https://en.wikipedia.org/wiki/WebGL.

] [Accessed 20 02 2018].

[30 Wikipedia, "Aliasing (computing)-Wikipedia," [Online]. Available:
] https://en.wikipedia.org/wiki/Aliasing_(computing). [Accessed 20 02 2018].

[31 Wikipedia, "Ambient occlusion-Wikipedia," [Online]. Available:
] https://en.wikipedia.org/wiki/Ambient_occlusion. [Accessed 20 02 2018].

[32 Wikipedia, "Bump mapping-Wikipedia," [Online]. Available:
] https://en.wikipedia.org/wiki/Bump_mapping. [Accessed 20 02 2018].

[33 Wikipedia, "Concurrent User-Wikipedia," [Online]. Available:
] https://en.wikipedia.org/wiki/Concurrent_user. [Accessed 19 02 2018].

[34 Wikipedia, "Mipmap-Wikipedia," [Online]. Available: https://en.wikipedia.org/wiki/Mipmap.
] [Accessed 19 02 2018].

[35 Wikipedia, "Parallax mapping-Wikipedia," [Online]. Available:
] https://en.wikipedia.org/wiki/Parallax_mapping. [Accessed 20 02 2018].

[36 Wikipedia, "Plug-in (computing)-Wikipedia," [Online]. Available:
] https://en.wikipedia.org/wiki/Plug-in_(computing). [Accessed 20 02 2018].

[37 Wikipedia, "Reflection mapping-Wikipedia," [Online]. Available:
] https://en.wikipedia.org/wiki/Reflection_mapping. [Accessed 20 02 2018].

[38 Wikipedia, "Rendering (computer graphics)-Wikipedia," [Online]. Available:
] https://en.wikipedia.org/wiki/Rendering_(computer_graphics). [Accessed 20 02 2018].

[39 Wikipedia, "Screen space ambient occlusion-Wikipedia," [Online]. Available:
] https://en.wikipedia.org/wiki/Screen_space_ambient_occlusion. [Accessed 20 02 2018].

[40 Wikipedia, "Scripting language-Wikipedia," [Online]. Available:
] https://en.wikipedia.org/wiki/Scripting_language. [Accessed 19 02 2018].

[41 Wikipedia, "Shadow mapping-Wikipedia," [Online]. Available:
] https://en.wikipedia.org/wiki/Shadow_mapping. [Accessed 20 02 2018].

[42 Wikipedia, "Texture compression-Wikipedia," [Online]. Available:
] https://en.wikipedia.org/wiki/Texture_compression. [Accessed 19 02 2018].

[43 Wikipedia, "Texture mapping-Wikipedia," [Online]. Available:
] https://en.wikipedia.org/wiki/Texture_mapping. [Accessed 20 02 2018].

[44 Wikipedia, "Z-buffering- Wikipedia," [Online]. Available: https://en.wikipedia.org/wiki/Z-
] buffering. [Accessed 20 02 2018].

[45] Unity, "Unity-Manual: GameObejcts," [Online]. Available: https://docs.unity3d.com/Manual/GameObjects.html. [Accessed 23 02 2018].

[46] Unity, "Unity-Manual: Learning the interface," [Online]. Available: https://docs.unity3d.com/Manual/LearningtheInterface.html?_ga=2.12653298.457138558.1519213296-397481912.1498134592. [Accessed 21 02 2018].

[47] Unity, "Unity-Manual: The Project Window," [Online]. Available: https://docs.unity3d.com/Manual/ProjectView.html. [Accessed 26 02 2018].

[48] Unity, "Unity-Manual: The Scene View," [Online]. Available: https://docs.unity3d.com/Manual/UsingTheSceneView.html. [Accessed 26 02 2018].

[49] Unity, "Unity-Manual: The Game View," [Online]. Available: https://docs.unity3d.com/Manual/GameView.html. [Accessed 23 02 2018].

[50] Unity, "Unity-Manual: The Hieararchy window," [Online]. Available: https://docs.unity3d.com/Manual/Hierarchy.html. [Accessed 22 02 2018].

[51] Unity, "Unity-Manual: The Inspector window," [Online]. Available: https://docs.unity3d.com/Manual/UsingTheInspector.html. [Accessed 22 02 2018].

[52] Unity, "Unity-Manual: Colliders," [Online]. Available: https://docs.unity3d.com/Manual/CollidersOverview.html. [Accessed 23 02 2018].

[53] Wikipedia, "Microsoft Visual Studio-Wikipedia," [Online]. Available: https://en.wikipedia.org/wiki/Microsoft_Visual_Studio. [Accessed 26 02 2018].

[54] Wikipedia, "Source code editor-Wikipedia," [Online]. Available: https://en.wikipedia.org/wiki/Source_code_editor. [Accessed 26 02 2018].

[55] Wikipedia, "Visual Basic-Wikipedia," [Online]. Available: https://en.wikipedia.org/wiki/Visual_Basic. [Accessed 26 02 2018].

[56] Wikipedia, "Legacy System-Wikipedia," [Online]. Available: https://en.wikipedia.org/wiki/Legacy_system. [Accessed 26 02 2018].

[57] Microsoft, "Visual Studio Tools for Unity| Microsoft Docs," [Online]. Available: https://docs.microsoft.com/en-us/visualstudio/cross-platform/visual-studio-tools-for-unity. [Accessed 26 02 2018].

[58] Microsoft, "Overview of Visual Studio Tools for Unity| Microsoft," [Online]. Available: https://docs.microsoft.com/en-us/visualstudio/cross-platform/overview-of-visual-studio-tools-for-unity. [Accessed 26 02 2018].

[59] Wikipedia, "Componet Object Model-Wikipedia," [Online]. Available:

] https://en.wikipedia.org/wiki/Component_Object_Model. [Accessed 26 02 2018].

[60 Wikipedia, "Machine code-Wikipedia," [Online]. Available:
] https://en.wikipedia.org/wiki/Machine_code. [Accessed 26 02 2018].

[61 Wikipedia, "Managed code-Wikipedia," [Online]. Available:
] https://en.wikipedia.org/wiki/Managed_code. [Accessed 26 02 2018].

[62 Wikipedia, "Core dump-Wikipedia," [Online]. Available:
] https://en.wikipedia.org/wiki/Core_dump. [Accessed 26 02 2018].

[63 Unity, "Unity-Manual: Prefabs," [Online]. Available:
] https://docs.unity3d.com/Manual/Prefabs.html?_ga=2.96526173.359323104.1519727054-397481912.1498134592. [Accessed 27 02 2018].

[64 Unity, "Unity-Manual: Rendering Statistics Window," [Online]. Available:
] https://docs.unity3d.com/Manual/RenderingStatistics.html. [Accessed 26 02 2018].

[65 Wikipedia, "Software development kit-Wikipedia," [Online]. Available:
] https://en.wikipedia.org/wiki/Software_development_kit. [Accessed 26 02 2018].

[66 Wikipedia, "XML-Wikipedia," [Online]. Available: https://en.wikipedia.org/wiki/XML.
] [Accessed 26 02 2018].

[67 Unity, "Unity-Manual: Rigidbody 2D," [Online]. Available:
] https://docs.unity3d.com/Manual/class-Rigidbody2D.html. [Accessed 06 03 2018].

[68 Unity, "Unity-Manual: Box Collider 2D," [Online]. Available:
] https://docs.unity3d.com/Manual/class-BoxCollider2D.html. [Accessed 06 03 2018].

[69 Unity, "Unity-Scripting API: Button," [Online]. Available:
] https://docs.unity3d.com/ScriptReference/UI.Button.html. [Accessed 06 03 2018].