UNIVERSITY OF IOANNINA

Σχολή

Πληροφορικής και Τηλεπικοινωνίων Άρτας

Πτυχιακή Εργασία

« Χειριστής χαρακτήρα πολλαπλών χρήσεων σε Unity»

Πέτρος Έξαρχος

Επιβλέπων καθηγητής : Χρήστος Γκόγκος

Άρτα , .. , 2021

ΠΡΟΛΟΓΟΣ

Το Interactive Entertainment System ή απλούστερα τα ηλεκτρονικά παιχνίδια είναι ένα σύγχρονο media που εμφανίστηκε τις τελευταίες 5 δεκαετίες και εξελίσσετε από τότε με ραγδαίους ρυθμούς. Η μοναδική του ιδιότητα να έχει αλληλεπίδραση με τον χρήστη σε αντίθεση με τα υπόλοιπα media, που επιτρέπουν μόνο τον ρόλο του παρατηρητή/ακροατή (Κινηματογράφος, μουσική), το έχουν καταστήσει επίκεντρο ενδιαφέροντος στον τομέα της τέχνης αλλά και της επιστήμης και συγκεκριμένα της επιστήμης της πληροφορικής .

Η παρούσα πτυχιακή επιχειρεί να διερευνήσει το media αυτό, από την σκοπιά της επιστήμης της πληροφορικής και να κάνει επίδειξη θεωρητικών μοντέλων του προγραμματισμού στην πράξη. Η πτυχιακή αποτελείτε από έξι ενότητες, την εισαγωγή στο πλαίσιο των γραφικών των ηλεκτρονικών παιχνιδιών, τα physics στα ηλεκτρονικά παιχνίδια, το περιβάλλον ανάπτυξης εφαρμογών της Unity, την σχεδιαστική λογική που ακολουθήθηκε στο προγραμματιστικό κομμάτι της πτυχιακής, τη δομή του και τέλος τα σημαντικά μέρη του προγράμματος.

UNIVERSITY OF IOANNINA

Σχολή

Πληροφορικής και Τηλεπικοινωνίων Άρτας

Πτυχιακή Εργασία

« Χειριστής χαρακτήρα πολλαπλών χρήσεων σε Unity»

Πέτρος Έξαρχος

Επιβλέπων καθηγητής : Χρήστος Γκόγκος

Άρτα , .. , 2021

ΕΥΧΑΡΙΣΤΙΕΣ

Επιθυμώ να εκφράσω όλο μου το σεβασμό και την ευγνωμοσύνη στον επιβλέπον καθηγητή Χρήστο Γκόγκο, ο οποίος με καθοδήγησε καθ' όλη τη διάρκεια εκπόνησης της πτυχιακής μου εργασίας. Επίσης, ευχαριστώ θερμά όλους τους καθηγητές του Πανεπιστημίου Ιωαννίνων του Πληροφορικής και Τηλεπικοινωνιών Άρτας, για τις πολύτιμες γνώσεις που μου παρείχαν κατά τη διάρκεια της φοίτησής μου στο προπτυχιακό πρόγραμμα.

Περίληψη

Η εργασία αυτή πραγματεύεται τον σχεδιασμό και την ανάπτυξη ενός module, για το περιβάλλον της μηχανής Unity, το οποίο έχει κατασκευαστεί με την γλώσσα C# και την χρήση των native βιβλιοθηκών της εν λόγω μηχανής. Ο σκοπός του module είναι η εύκολη και φιλική προς τον χρήστη εγκατάσταση του, σε οποιοδήποτε τύπου project της Unity, καθώς και η χρήση του, ως χειριστή χαρακτήρα, με πλήθος δυνατοτήτων προσαρμογής.
Αρχικά, γίνεται αναφορά στα γραφικά των ηλεκτρονικών παιχνιδιών και τις βασικές έννοιες και ορολογίες που αφορούν αυτό το πεδίο. Έπειτα γίνεται αναφορά στον τομέα των physics των video games και την χρησιμότητά τους, στην ανάπτυξη ηλεκτρονικών παιχνιδιών. Στη συνέχεια, γίνεται αναφορά στο σχεδιαστικό και προγραμματιστικό περιβάλλον της Unity, επισημαίνοντας τα χαρακτηριστικά και τις δυνατότητες της μηχανής. Ακολουθεί εισαγωγή στην σχεδιαστική λογική του module, αναλύοντας τις προγραμματιστικές τεχνικές που χρησιμοποιήθηκαν και στην συνέχεια αναλύονται τα δομικά μέλη του module. Τέλος, γίνετε λεπτομερής αναφορά σε κύρια και ουσιώδη κομμάτια κώδικα και τις λειτουργείες που εκπληρώνουν.

Λέξεις – Κλειδιά: Module, γραφικά, Physics, Unity, C#, δομή, σχεδιαστική λογική

Abstract

This thesis regards the design and development of a module for the Unity game development engine, constructed in C# language and utilized the native libraries of the engine. The purpose of this API is easy and user-friendly installation, in any type of Unity project, as well as usage as a character controller, with various adaptation possibilities.
Initially, a reference is made to video graphics along with the standard concepts and terminologies. Afterwards, a reference is made to the physics of video games and its usefulness in video game development. Next, a reference is made to Unity's design and programing environment, pointing out the features and possibilities of the engine. Afterwards, there is an introduction on the design logic of the module, analyzing its programming technics that were used and next up an analysis on the structure components of the module. Next up, a detailed reference is made to major and essential parts of code and the operations accomplished by them.

Keywords: Module, graphics, Physics, Unity, C#, structure, design logic

Περιεχόμενα

Contents

1 Ιστορική εξέλιξη των γραφικών ηλεκτρονικών παιχνιδιών

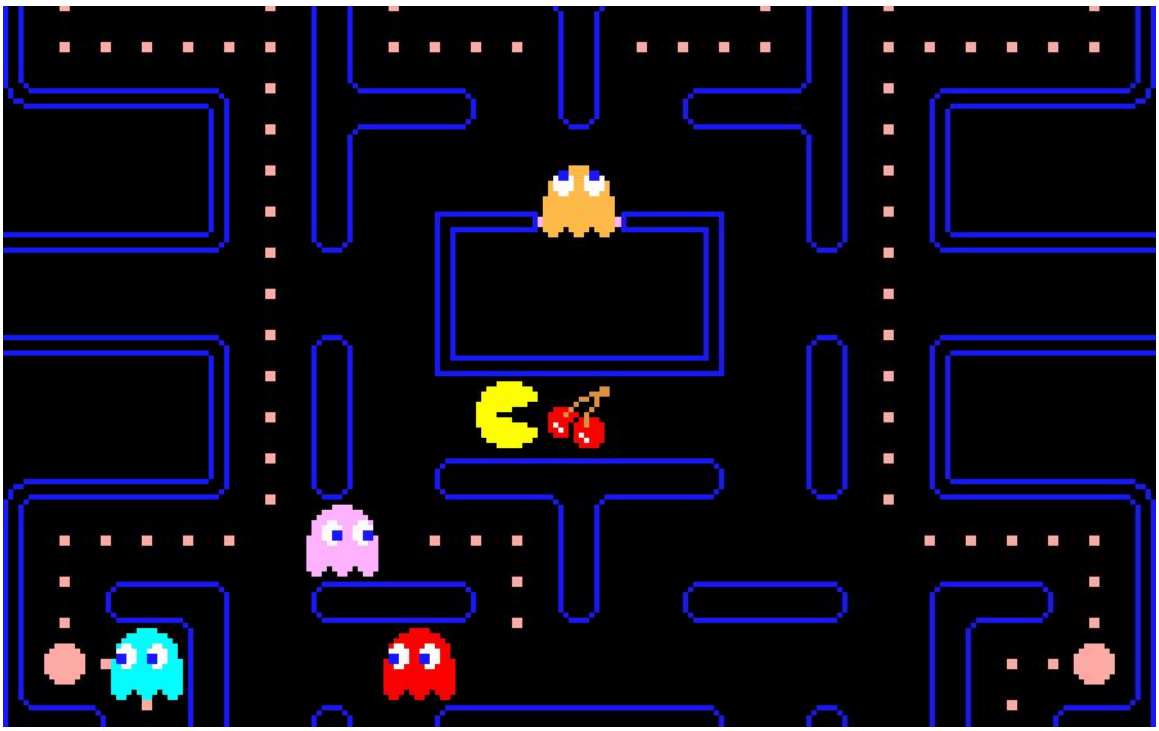
Η απεικόνιση των γραφικών στοιχείων των ηλεκτρονικών παιχνιδιών, αποτελεί το σήμα κατατεθέν της εξέλιξης της τεχνολογίας τους. Από το πρώτο αρχαϊκό παιχνίδι συσκευής καθολικού σωλήνα των Thomas T. Goldsmith Jr και Estle Ray Mann, στην εμφάνιση των Arcade μηχανών και παιχνιδοκονσολων και την ανάδειξη και εξέλιξη των τρισδιάστατων γραφικών, τα γραφικά έχουν κάνει ένα μεγάλο τεχνολογικό άλμα στην διάρκεια της ιστορίας των ηλεκτρονικών παιχνιδιών. Τα γραφικά έχουν πληθώρα ορόσημων της εξέλιξης τους, σε διάφορες χρονικές περιόδους, που έχουν επηρεάσει τον τρόπο που κατασκευάζονται τα ηλεκτρονικά παιχνίδια αλλά και τον τρόπο που βιώνονται, σε παρόμοιο βαθμό με την εξέλιξη του gameplay design.

Εικόνα 1.1 - το παιχνίδι tennis for two σε παλμογράφο

Στα πρώιμα χρόνια της εμφάνισης των video games, η παρουσία τους τόσο στο κοινό όσο και στην αγορά, ήταν μία τεχνολογική καινοτομία, που εξελισσόταν ραγδαία με κάθε βήμα, σε μία εποχή, όπου η μετακίνηση ενός pixel στην οθόνη της τηλεόρασης ήταν κατόρθωμα. Παιχνίδια όπως το Pong, θεωρούνταν απίστευτα τεχνολογικά επιτεύγματα, που έγιναν μία από τις μεγαλύτερες επιτυχίες της βιομηχανίας ηλεκτρονικών παιχνιδιών. Στο πέρασμα του χρόνου, όταν τα video games, εξασφάλισαν την θέση τους στην αγορά, η ανάγκη για εξέλιξη του hardware και των γραφικών έγινε επίκεντρο ενδιαφέροντος. Το Galaxian του

1979 ήταν το πρώτο επιτυχημένο video game που χρησιμοποιούσε RGB χρώματα και έθεσε τον πήχη για μεταγενέστερους τίτλους, όπως το Pac Man.

1.1 Δισδιάστατα γραφικά



Εικόνα 1.2 - το παιχνίδι Pacman

Οι μεθοδολογίες rendering της εποχής, ήταν το Rastering και τα Vector. Το Rastering, προέρχεται από την λατινική λέξη rastrum που σημαίνει οργώνω και μέχρι και σήμερα αποτελεί τον επικρατές τρόπο rendering. Κατά την μεθοδολογία αυτή, το σύστημα ταχύτατα σαρώνει κάθε γραμμή της οθόνης σχηματίζοντας ένα πλέγμα στην πορεία, και σειρά παρά σειρά ζωγραφίζετε μία εικόνα στην οθόνη. Αντιθέτως, τα Vector επηρεάζουν απευθείας την δέσμη ηλεκτρονίων, ώστε να σχηματίσουν εικόνα, ακριβώς όπως ένας παλμογράφος. Η διαφορά μεταξύ των δύο rendering μεθόδων, είναι πως τα Vectors είναι πολύ πιο αποδοτικά αλλά έχουν πολύ περιορισμένες δυνατότητες σε σχέση με το Rastering, εξού και ο λόγος που δεν επικράτησαν. Προκειμένου να αξιοποιηθεί στο μέγιστο βαθμό το διαθέσιμο hardware της εποχής, εφαρμόζονταν διάφορα οπτικά κόλπα, με σκοπό να δίνουν μια καλύτερη αίσθηση της εικόνας και των κινούμενων στοιχείων στην οθόνη. Μία από τις πρώτες τεχνικές που χρησιμοποιήθηκαν, ήταν το Scrolling, όπου μια μεγάλη εικόνα μεγεθύνονταν στην οθόνη και μετακινούνταν, δίνοντας την αίσθηση της κίνησης στο χώρο.

Η τεχνική αυτή μείωνε τις απαιτήσεις των γραφικών αλλά αύξανε τις απαιτήσεις της μνήμης καθώς μετακινούνταν τεράστια κομμάτια μνήμης. Μία εξίσου έξυπνη μέθοδος να αποδοθεί η αίσθηση του βάθους, ήταν η αυξομείωση μεγεθών των sprites δίνοντας την αίσθηση της εγγύτητας, καθώς και το parallax scrolling που μετακινούσε το σκηνικό στο background με μικρότερη ταχύτητα από αυτήν που μετακινούνταν η εικόνα που βρισκόταν τα sprites, δίνοντας μεγαλύτερη την αίσθηση του βάθους. Μία από τις πιο τεχνολογικά ανεπτυγμένες μεθόδους δημιουργίας sprites, αλλά εξαιρετικά χρονοβόρα, ήταν το rotoscoping, όπου τα sprites του παιχνιδιού δημιουργούνταν από την επεξεργασία live action video, καρέ καρέ, δίνοντας αληθοφανής αποτελέσματα στην κίνηση.

Εικόνα 1.3 – Rotoscoping για δημιουργία κινούμενων sprites

Επιπλέων αξιοσημείωτες αναφορές αποτελούν τα digitized sprites, όπου κανονικές φωτογραφίες συρρικνώνονταν και χρησιμοποιούνταν ως sprites καθώς και το full motion video, όπου ένα παιχνίδι χρησιμοποιούσε κομμάτια από καταγεγραμμένα video clips σε συνδυασμό με sprites.

1.2 Τρισδιάστατα γραφικά

Τα τρισδιάστατα παιχνίδια κάνουν την εμφάνισή τους ήδη από το 1980, χωρίς μεγάλη επιτυχία λόγω των περιορισμών του hardware της εποχής, αδυνατώντας να ανταγωνιστούν στην αγορά. Τα πρώτα τρισδιάστατα games χρησιμοποιούσαν wireframes για την αναπαράσταση πολυγώνων, έχοντας εξαιρετικά λιτή οπτική εμφάνιση. Το επόμενο βήμα ήταν ο χρωματισμός των wireframe polygons με flat shading, από το video game I Robot του 1983, ένα εξίσου απλό εφέ αλλά δύσκολο να εφαρμοστεί επιτυχώς. Τα τρισδιάστατα παιχνίδια στο τέλος της δεκαετίας θα αρχίσουν να κυριαρχούν στην αγορά, λόγο της βελτίωσης του hardware των υπολογιστών και παιχνιδοκονσόλων. Τόσο από τα τρισδιάστατα όσο και από τα δισδιάστατα, δεν έλειψαν τα έξυπνα κόλπα με σκοπό να μην επιβαρύνεται οι επιδόσεις των παιχνιδιών, με ένα από τα πιο διάσημα να είναι η χρήση δισδιάστατων sprites με πολλές πλευρές. Σε άλλες τεχνικές χρησιμοποιούνταν πολύγωνα μόνο για συγκεκριμένες οντότητες ενώ το περιβάλλον ήταν μία bitmap φωτογραφία που άλλαζε ανάλογα με το που έδειχνε η κάμερα. Ένα τεράστιο άλμα θα γίνει με την τεχνολογία raycasting όπου κάνει render μόνο ότι είναι στο πεδίο της οθόνης κάνοντας εξαιρετικά αποδοτική την οπτικοποίηση τρισδιάστατων χώρων, μία τεχνική που θα κορυφωθεί με την μηχανή id tech 1 του John Carmack, όπου κατασκευάστηκε το Doom το 1993. Τα τρισδιάστατα παιχνίδια με πραγματικό texture map θα αρχίσουν να αναδεικνύονται μετά το 1995 και θα φέρουν μαζί τους την πέμπτη γενιά παιχνιδοκονσόλων με το playstation της Sony και το Nintendo 64 της Nintendo, να κυριαρχούν στην αγορά. Μία αξιοσημείωτη αναφορά στην εξέλιξη των τρισδιάστατων video games ήταν τα voxels ή volumetric pixels, που ήταν ένας διαφορετικός και πιο αποδοτικός τρόπος κατασκευής πολυγώνων, με την χρήση τρισδιάστατων pixel ή blocks, που αποδείχτηκε αχρείαστος λόγω της εξέλιξης του hardware και την ανάδυση του 3D acceleration. Το 3D acceleration ήταν μία μέθοδος υποστήριξης του επεξεργαστή από επιπρόσθετο hardware. Μια από τις πρώτες κάρτες που εκπλήρωναν αυτό τον σκοπό ήταν η 3dfx's Voodoo. Προς το τέλη του 2000 το 3D acceleration έγινε αναπόσπαστο κομμάτι των video games, γεγονός που ισχύει μέχρι και σήμερα. Από αυτό το σημείο και μετά η εξέλιξη τον γραφικών εστίασε σε πολλούς τομείς και πεδία, όπως την δυναμική αναπαράσταση του φωτός, το anti aliasing, bloom, chromatic aberration και άλλες μεθόδους καλυτέρευσης της τρισδιάστατης εικόνας. Η βελτιστοποίηση των γραφικών συνεχίζεται μέχρι και σήμερα με εξαιρετικά αληθοφανή αποτελέσματα, και καινούργιες τεχνολογίες να εμφανίζονται σε τακτά χρονικά διαστήματα, όπως το RTX της Nvidia το 2018.

2 Physics στα ηλεκτρονικά παιχνίδια

Τα αντικείμενα στον πραγματικό κόσμο λειτουργούν βάση των κανόνων της φυσικής, όπως η αδράνεια, η βαρύτητα, οι νόμοι του Νεύτωνα, κτλ. Με το ίδιο σκεπτικό, θα ήταν αναμφίβολα ένα μεγάλο εγχείρημα, τα αντικείμενα των ηλεκτρονικών παιχνιδιών να λειτουργούν με τον ίδιο τρόπο. Τα γραφικά, όπως αναφέρθηκε σε προηγούμενο κεφάλαιο, έχουν εξελιχθεί σε προηγμένο στάδιο που πλησιάζει όλο και περισσότερο το ρεαλιστικό. Η προσομοίωση των φυσικών νόμων στα ηλεκτρονικά παιχνίδια όμως, δεν έχει κάνει αντίστοιχη τεχνολογική πρόοδο. Μέχρι και σήμερα δεν υπάρχει κάποιο video game που να προσομοιώνει με πραγματική ακρίβεια τους νόμους της φυσικής σε μοριακό επίπεδο. Παρόλο που δύνανται να κατασκευαστεί ένα τέτοιο παιχνίδι, διότι οι νόμοι της φυσικής είναι γνωστοί και επιβεβαιωμένοι με μαθηματικούς κανόνες, υπάρχουν δύο βασικοί λόγοι που δεν υφίστανται μία τέτοια εφαρμογή. Ο πρώτος είναι πως δεν υπάρχει αρκετή επεξεργαστική ισχύς για να πετύχει μία τέτοια ενέργεια σε πραγματικό χρόνο, από κανένα υπολογιστικό σύστημα. Ο δεύτερος είναι πως τα video games προσομοιώνονται κατά προτίμηση αντιρεαλιστικά, με σκοπό το παιχνίδια να είναι ενδιαφέρων και ψυχαγωγικά, απαραίτητα χαρακτηριστικά για την επιτυχία μιας τέτοιας εφαρμογής. Τα χαρακτηριστικά αυτά θα περιορίζονταν κάτω από ένα πραγματικό σύστημα φυσικών κανόνων.

2.1 Εφαρμογή των Physics

Τα physics στα ηλεκτρονικά παιχνίδια, ως ορολογία, περιγράφουν τον τρόπο με τον οποίο μεταβάλλονται τα αντικείμενα στον δισδιάστατο ή τρισδιάστατο χώρο και ενδέχεται να διαφέρουν σημαντικά σε διαφορετικούς τύπους παιχνιδιών αλλά και από παιχνίδι σε παιχνίδι της ίδιας κατηγορίας. Στο παρελθόν, λόγω των περιορισμών του hardware, η υλοποίηση των physics στηρίζονταν σε έτοιμα animations δίνοντας την ψευδαίσθηση ενός συστήματος φυσικών νόμων. Πολλές φορές, προκειμένου να είναι πιο αληθοφανή η προσομοίωση, υπήρχαν περισσότερα από ένα animations για κάθε γεγονός, που είτε επιλέγονταν τυχαία ή κάτω από ορισμένες συνθήκες. Με την εξέλιξη της τεχνολογίας τα video games άρχισαν να στηρίζονται όλο και περισσότερο σε συστήματα από φυσικούς κανόνες και λιγότερο σε έτοιμα animation για την προσομοίωση. Τα συστήματα αυτά καθόριζαν με ποιον τρόπο αντιδρούν τα αντικείμενα στον χώρο από εξωτερικές δυνάμεις και από αλληλεπιδράσεις μεταξύ τους. Ο τρόπος λειτουργίας των συστημάτων αυτών περιλαμβάνει συνδυασμούς από υπολογισμούς ταχύτητας, θέσης και ασκούμενων δυνάμεων στα σημεία επαφής αντικειμένων, και στην συνέχεια, την πυροδότηση των αντιδράσεων τους, βάση ιδιοτήτων τους, όπως το βάρος τους. Η διαδικασία συμβαίνει επαναλαμβανόμενα σε πραγματικό χρόνο και δεν είναι προκαθορισμένα τα αποτελέσματα που θα προκύψουν, και σε συνδυασμό με την δυνατότητα που του παίκτη να επεμβαίνει ενεργά στο σύστημα αποτελεί τον πιό αληθοφανές τρόπο προσομοίωσης μέχρι και σήμερα.

2.2 Τεχνολογίες των Physics

Ragdolls: Το Ragdoll είναι ένα σύνολο από στερεά σώματα, συνδεδεμένα μεταξύ τους με τρόπο ώστε να λειτουργούν ως ένα σώμα. Η πιο ευρέως διαδεδομένη χρήση των ragdolls είναι για τους ανθρώπινους χαρακτήρες, όπου τα συνδετικά μέλη του ragdoll αναπαριστούν τα κόκαλα. Το κάθε μέλος μπορεί να κινείται και να περιστρέφεται με συγκεκριμένο τρόπο όταν του ασκείται κάποια δύναμη, επηρεάζοντας και τα υπόλοιπα μέλη στην πορεία. Αυτό αποδίδει μία αληθοφανής αναπαράσταση της κίνησης σωμάτων.

Inverse Kinematics και Procedural Animation: Δύο τεχνικές που αξιοποιούν τα ragdolls είναι το inverse kinematics και το procedural Animation. Το inverse kinematics είναι ένα post-processing effect όπου τροποποιεί ένα ragdoll animation ώστε να προσαρμόζεται κατάλληλα στον χώρο μετά την περάτωση του animation. Το procedural animation είναι μία μέθοδος που εξαλείφει την ανάγκη για έτοιμα ragdoll animations και τα παράγει locomotion κατά την εκτέλεση, βάση προκαθορισμένων κανόνων.

Particle Effects: Ένα ανεξάρτητο παράρτημα των physics ενός παιχνιδιού, υπεύθυνο για την δημιουργία γραφικών εφέ όπως : εκρήξεις, φωτιά, νερό, χιόνι βροχή, κίνηση μαλλιών κ.τ.λ. Τα εφέ αυτά είναι πάρα πολύ δύσκολο να προσομοιωθούν σε πραγματικό χρόνο από τον renderer λόγο των πάρα πολλών σωματιδίων εν κινήσει ταυτόχρονα, και για τον λόγο αυτό υλοποιούνται από το physics system και συγκεκριμένα από το particle physics system του παιχνιδιού που αξιοποιεί πόρους του CPU ένατη του GPU για να τα οπτικοποιήσει.

3 Unity Engine

Μέχρι και πριν δύο δεκαετίες, η κατασκευή τρισδιάστατων αλλά και διδιάστατων ηλεκτρονικών παιχνιδιών αποτελούσε μία εξαιρετικά πολύπλοκη διαδικασία και απαιτούσε κατά κύριο λόγο ομάδες τουλάχιστον δέκα ατόμων με βαθιές γνώσεις στον τομέα που θα αναλάμβαναν να εργαστούν. Τώρα πλέον με την εξέλιξη της τεχνολογίας και την αύξηση του ενδιαφέροντος στο συγκεκριμένο media είναι δυνατόν, το ίδιο έργο να παραχθεί ακόμη και από ένα άτομο. Το έργο αυτό επιτυγχάνεται με τη χρήση λογισμικού, ειδικό για την ανάπτυξη ηλεκτρονικών παιχνιδιών, τα επονομαζόμενα game engines. Ο ορισμός προϋπάρχει και σε παλιότερα στάδια εξέλιξης της βιομηχανίας ηλεκτρονικών παιχνιδιών αλλά η εμφάνισή τους γίνεται στο ευρύ κοινό μετά το 2000 ως προϊόντα ή ελεύθερο λογισμικό, με σκοπό να διευκολύνει και να επιταχύνει την διαδικασία ανάπτυξης. η χρήση τους γίνεται τόσο από μικρά studio και ελεύθερους επαγγελματίες όσο και από μεγάλες εταιρείες.

3.1 Game Engines

Υπάρχει μεγάλη γκάμα από διαθέσιμα Game Engines με διαφορετικές δυνατότητες και για διαφορετικούς σκοπούς, παραδείγματος χάρη εξειδίκευση σε έναν συγκεκριμένο τύπο game, σε συγκεκριμένους τύπους γραφικών κ.τ.λ. Μερικές από τις πιο διάσημες είναι μεταξύ άλλων :

- **Unity Engine** και **Unreal Engine 4**: Οι πιο ευρέως αναγνωρισμένες δωρεάν μηχανές γενικής χρήσης και διαπλατφορμικών δυνατοτήτων, με την Unreal να εστιάζει περισσότερο στις δυνατότητες των γραφικών και την Unity στις αποδόσεις.
- **Game Maker Studio** : Εστιάζει στην εξορθολόγηση της διαδικασίας ανάπτυξης και το Pixel Art Style, έχει περιορισμένες 3D δυνατότητες.
- **RPG Maker** : Εξειδικεύεται στην ανάπτυξη JRPG games.
- **Source Engine** : Εξειδικεύεται στην ανάπτυξη FPS games, to modularity και τα physics.
- **ForgeLight engine** : Εξειδικεύεται στην ανάπτυξη MMO Games.
- **Ego** : Εξειδικεύεται στην ανάπτυξη Racing Games.

3.2 Χαρακτηριστικά της μηχανής Unity

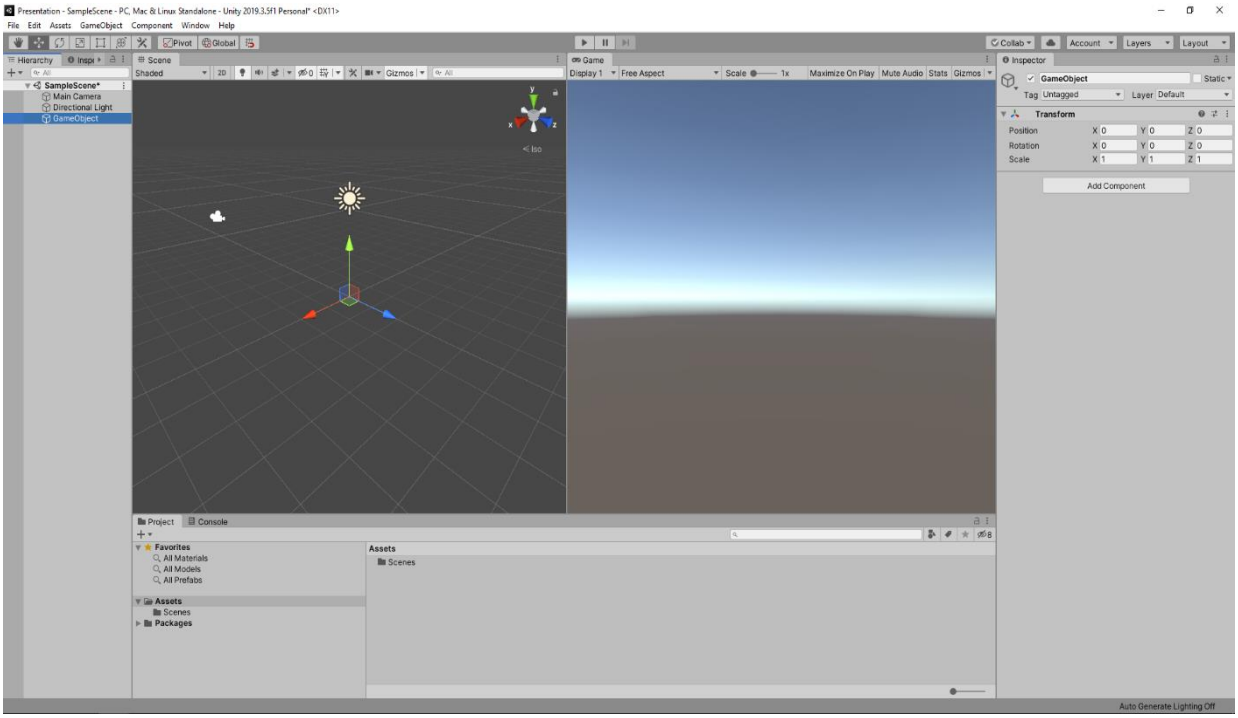
Στην παρούσα εργασία χρησιμοποιείται η μηχανή Unity που αποτελεί μία από τις πιο διάσημες δωρεάν μηχανές. Το Unity είναι ένα διαπλατφορμικό game engine που έχει αναπτυχθεί από την εταιρία Unity Technologies. Το λογισμικό μολονότι δεν είναι ανοιχτού κώδικα, παρέχεται δωρεάν και είναι υπό συνεχή ανάπτυξη. Η μηχανή μπορεί να χρησιμοποιηθεί για κατασκευή τρισδιάστατων αλλά και δισδιάστατων παιχνιδιών καθώς επίσης μπορεί να υποστηρίξει εικονική πραγματικότητα (virtual reality) αλλά και επαυξημένη πραγματικότητα (augmented reality). Όσον αφορά τους τύπους παιχνιδιών που μπορούν να υλοποιηθούν, περιορίζεται μόνο από το hardware της προοριζόμενης πλατφόρμας. Είναι σημαντικό να επισημανθεί ότι το λογισμικό χρησιμοποιείται και έξω από τον πλαίσιο των video games καθώς χρησιμοποιείται επαγγελματικά και από δημιουργούς κινουμένων σχεδίων, αρχιτέκτονες και μηχανικούς.

Η ανάπτυξη εφαρμογής γίνεται μέσω των εργαλείων του editor της Unity, παράλληλα με τη χρήση ενός code editor, με προεπιλεγμένο το Visual Studio Community της Microsoft. Η κατασκευή κώδικα και η χρήση των εργαλείων του editor, καθώς και η ενσωμάτωση των έτοιμων modules της Unity, στηρίζονται πάνω σε συγκεκριμένες, προκατασκευασμένες οντότητες της μηχανής. Αυτό γίνεται προκειμένου να απλοποιείται η διαδικασία κατασκευής, χρήσης και εφαρμογής προγραμματιστικών λύσεων σε γραφικό περιβάλλον, και να αποφεύγετε η ανάπτυξη από το μηδέν, χωρίς να απαγορεύεται μία τέτοια επιλογή.

Εικόνα 3.1 – Το Logo της Unity Technologies

3.3 Σχεδιαστικό περιβάλλον της Unity

Το σχεδιαστικό περιβάλλον της Unity ή αλλιώς Editor, λειτουργεί με την χρήση δυναμικά προσαρμόσιμων παραθύρων που εκτελούν διαφορετικές λειτουργίες. Τα πιο σημαντικά παράθυρα είναι μεταξύ άλλων :

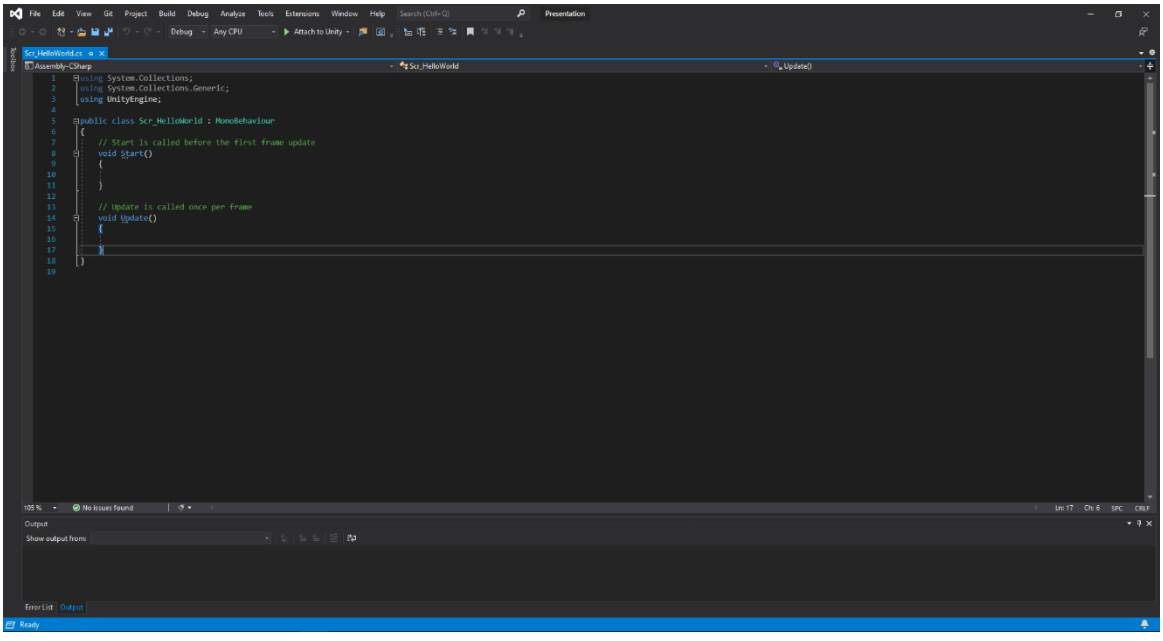


Εικόνα 3.2 – Το περιβάλλον ανάπτυξης παιχνιδιών της Unity

- Scene: Προσφέρει οπτική αναπαράσταση του τρισδιάστατου ή δισδιάστατου χώρου της φορτωμένης σκηνής και προσθαφαίρεση ή μετακίνηση gameobjects σε αυτή, με λειτουργία drag & drop.
- Hierarchy: Εμφανίζει σε λίστα όλα τα gameobjects καθώς και τα παιδία Gameobjects των gameobjects της φορτωμένης σκηνής
- Inspector: Προσθαφαιρεί και τροποποιεί τα δομικά μέλη των gameobjects της σκηνής ή των prefab αντικειμένων.
- Project: Διαχειρίζεται τους φακέλους και τα πηγαία αρχεία του project.
- Game: Στο παράθυρο αυτό γίνεται η δοκιμαστική εκτέλεση της εφαρμογής. Οπτικοποιεί οτιδήποτε γίνεται render κατά την.
- Console: Εμφανίζει μηνύματα για debugging, warnings και errors πού προκύπτουν από τον editor (π.χ. compile errors) ή από την εκτέλεση της εφαρμογής (π.χ. null reference exception

3.4 Προγραμματιστικό περιβάλλον της Unity

Η ανάπτυξη κώδικα στην Unity, γίνεται σε C# με την χρήση οποιουδήποτε προγράμματος. Στην συγκεκριμένη εργασία χρησιμοποιείται το Default IDE της Unity, το Visual Studio Community της Microsoft. Ο λόγος που επιλέγεται το συγκεκριμένο Development Environment, είναι το Unity Development Kit Addon, που προσφέρει έξτρα πλεονεκτήματα όπως, live syntax check και live debugging. Η ανάπτυξη κώδικα στην Unity δεν ακολουθεί την λογική του δομημένου προγραμματισμού αλλά στηρίζεται στην δημιουργία script classes τα οποία, μέσω events, εκτελούν κώδικα σε συγκεκριμένες χρονικές περιόδους κάθε frame και απαιτείται να βρίσκονται στην σκηνή ως μέλος ενός gameobject προκειμένου να λειτουργήσουν, δηλαδή, με όρους της Unity, να είναι components. Προκειμένου ένα script να αξιοποιηθεί ως component ενός gameobject της σκηνής, απαιτεί να έχει το class Monobehaviour ως superclass. Το εν λόγω superclass επιτρέπει την κλήση των sequenced events της Unity, για εκτέλεση κώδικα στα threads της εφαρμογής και την προσάρτηση του script σε οποιοδήποτε gameobject. Σε κάθε άλλη περίπτωση, δύναται να χρησιμοποιηθούν όλες οι native βιβλιοθήκες και μέθοδοι της C#, με ορισμένους περιορισμούς. Η πιο αξιοσημείωτη χρήση μεθόδων της C# στην Unity, είναι τα coroutines που επιτρέπουν την δημιουργία ενός νήματος για εκτέλεση κώδικα ασύγχρονα ή συγχρονισμένα με τα sequenced events της Unity καθώς και οι default τύποι delegates, Action και Func για δημιουργία event.



Εικόνα 3.3 – Το περιβάλλον Visual Studio Community 2019

3.5 Gameobject Entity

Μία από τις πιο βασικές οντότητες της Unity, αποτελεί το gameobject. Το gameobject είναι ένα ειδικό class, μέσω του οποίου γίνεται η δημιουργία οποιουδήποτε αντικειμένου στον χώρο. Τα αντικείμενα αυτού του class αποτελούνται από ένα δυναμικό σύνολο components, με αναπόσπαστο προαπαιτούμενο, το component transform, το οποίο αναπαριστά με τρία διανύσματα την θέση, τον προσανατολισμό και την κλίμακα μεγέθους του αντικειμένου στο χώρο. Τα components , native της unity ή κατασκευασμένα σε C#, καθορίζουν την συμπεριφορά του αντικειμένου που τα εμπεριέχει. Η προσθαφαίρεση τους γίνεται δυναμικά και επιτρέπεται να γίνει και κατά την εκτέλεση της εφαρμογής. Η λειτουργία και η σκοπιά ενός component δεν περιορίζεται με κάποιο τρόπο, διότι λειτουργεί αυτόνομα με μοναδική εξάρτηση το gameobject που εμπεριέχεται. Ένα αντικείμενο gameobject δεν μπορεί να έχει ως component ένα άλλο gameobject. Ειδική συμπεριφορά παρουσιάζουν μεταξύ τους σχετικά με την μέθοδο της σύνθεσης, καθώς ένα child Gameobject σε ένα άλλο gameobject, αλλάζει τον τρόπο αναπαράστασης του στον χώρο, διότι το παιδί αντικείμενο χάνει την απόλυτη αναπαράσταση του στον χώρο και μετατρέπεται σε σχετική με το parent gameobject. Ένα gameobject μπορεί να έχει άπειρα children αλλά μόνο ένα parent.

Δύο εξίσου σημαντικά χαρακτηριστικά σχετικά με το gameobject είναι :

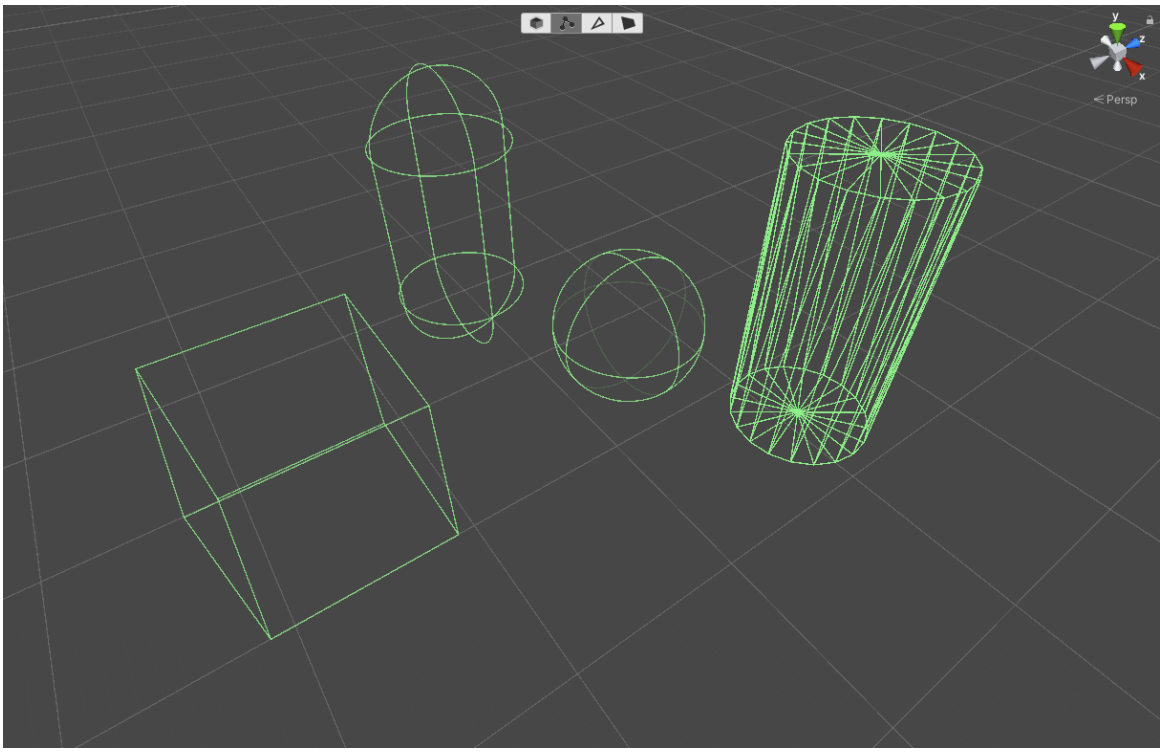
1): Αποτελεί απορρέων class του UnityEngine.Object, που έχει παρόμοια συμπεριφορά με το class Object της C#, συνεπώς οποιαδήποτε μεταβλητή τύπου του προ αναφέρων class ή των component του, αποτελούν αυτομάτως αναφορά.

2): Οτιδήποτε προορίζεται να εκτελεστεί σε πραγματικό χρόνο, στο κυρίως thread της εφαρμογής, πρέπει απαραίτητα να βρίσκετε ως component σε κάποιο αντικείμενο τύπου gameobject, εξαιρούμενων ειδικών περιπτώσεων static class και μεθόδων περιορισμένων δυνατοτήτων.

Συνοψίζοντας, επισημαίνεται πως το gameobject φέρει τον ρόλο του container και όχι του superclass. Η αναντικατάστατη χρησιμότητα του βασίζεται στην ικανότητα του να υπάγεται σε πολυμορφισμό με ανορθόδοξο τρόπο, καθώς δύνανται να υλοποιηθεί ως αυτόνομο αυθαίρετο αντικείμενο και τα components του να σχηματίσουν τις ιδιότητές του ακόμα και κατά την εκτέλεση.

3.6 Colliders και Collision

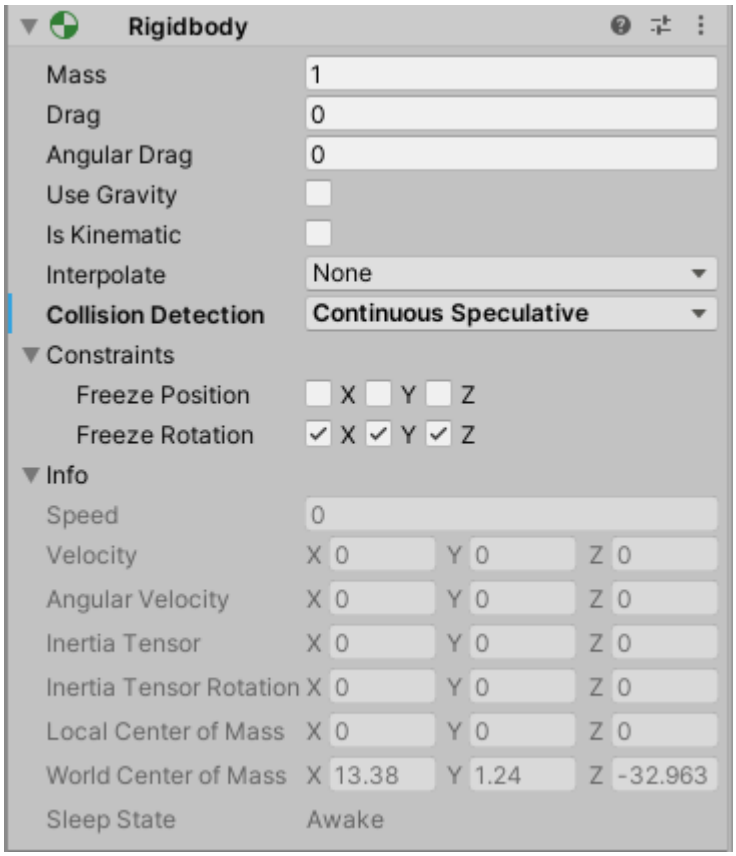
Προκειμένου να αναπαρασταθεί μία οντότητα στον χώρο, απαιτείται η χρήση ενός collider component. Τα colliders είναι ειδικά components, που ορίζουν τα τρισδιάστατα ή δισδιάστατα όρια των αντικειμένων που προσαρτώνται. Τα όρια αυτά χρησιμοποιούνται για υπολογισμούς επαφών μεταξύ άλλων colliders, κάνοντας τον κύριο σκοπό του component την αλληλεπίδραση στον χώρο. Τα colliders μπορούν να πάρουν ποικίλα σχήματα primitive τύπου της Unity, όπως κύβους, κυλίνδρους, σφαίρες και κάψουλες με τροποποιήσιμα χαρακτηριστικά. Μία ειδική κατηγορία collider αποτελεί το mesh collider που λαμβάνει το σχήμα του τρισδιάστατου μοντέλου, mesh, που είναι παράλληλα προσαρτημένο στο gameobject. Χρησιμοποιείται σε ειδικές περιπτώσεις λόγω της δυσκολίας των υπολογισμών που ενδέχεται να προκύψουν από ένα πολύπλοκο mesh.



Εικόνα 3.4 – Διαφορετικοί τύποι από Colliders

3.7 Rigidbody

Προκειμένου ένα αντικείμενο με collider να αξιοποιήσει το physics system της Unity, απαιτείται το component rigidbody να είναι συνδεδεμένο στο gameobject. Σε κάθε άλλη περίπτωση το collider, μεταχειρίζεται από την εφαρμογή ως ακίνητο σώμα και μία πιθανή μετακίνησή του από κάποιο script, δεν λαμβάνεται υπόψη του physics system. Το rigidbody αποδίδει ιδιότητες όπως μάζα, τριβή και βαρύτητα στο αντικείμενο που είναι συνδεδεμένο. Η πιο σημαντική ιδιότητα που παρέχεται από το rigidbody είναι η άσκηση δυνάμεων στο σώμα του αντικειμένου και η αλληλεπίδραση με άλλα σώματα που έχουν rigidbody, παραδείγματος χάρη, η μεταφορά δυνάμεων κατά την σύγκρουση. Οποιοσδήποτε υπολογισμός αλλά και εφαρμογή αποτελέσματος σχετικά με μεταφορά σωμάτων που έχουν rigidbody component, γίνεται στο physics event της Unity, που ονομάζεται FixedUpdate. Στην παρούσα εργασία, όσον αφορά το rigidbody, χρησιμοποιείται μόνο η άσκηση δυνάμεων και η δημιουργία επαφών, καθώς όλες οι άλλες λειτουργίες προσομοιώνονται ανεξάρτητα του component.



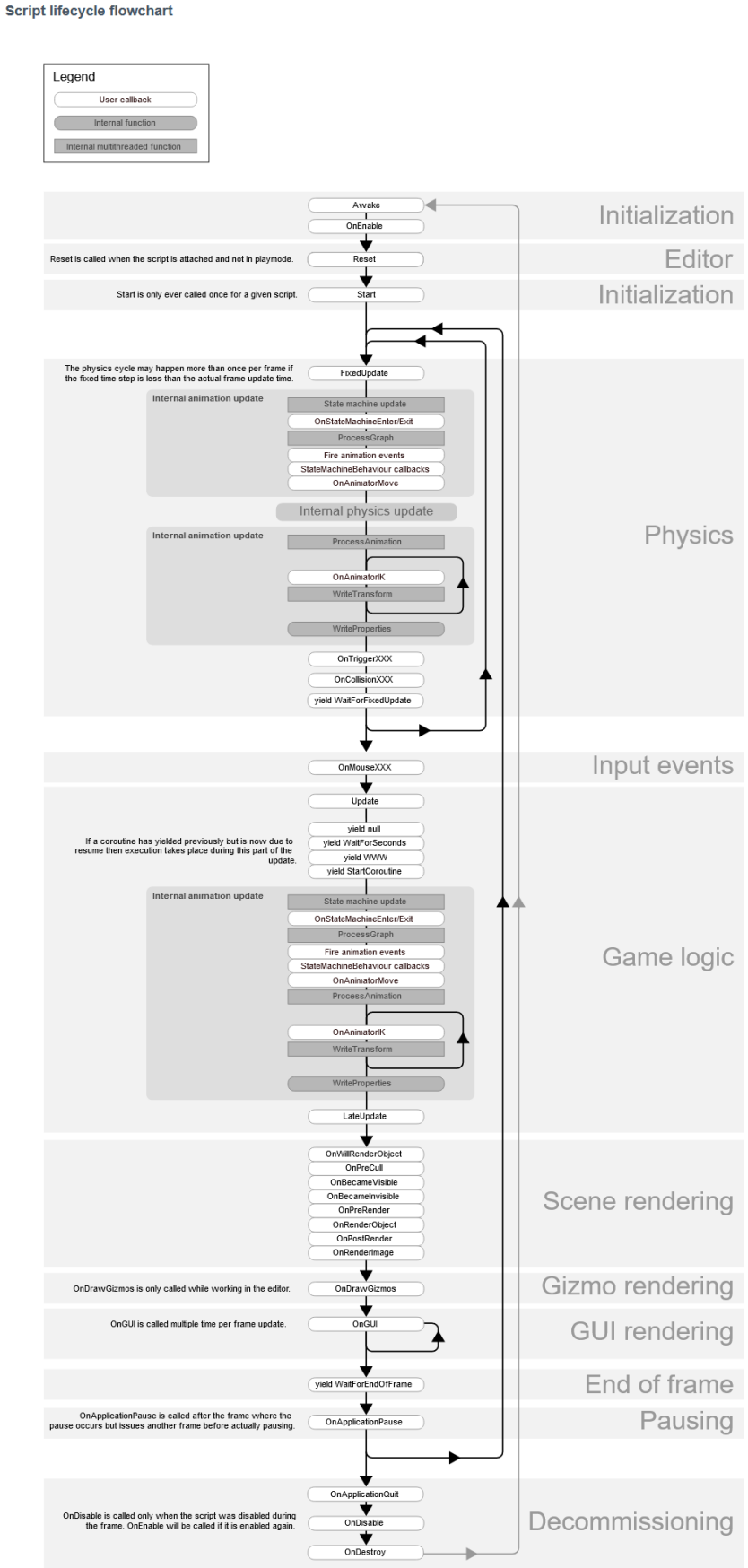
Εικόνα 3.5 – Το Rigidbody Component της Unity

3.8 Layer Masks (General)

Οι μάσκες επιπέδου χρησιμοποιούνται για να διαχωρίσουν ποια Colliders μπορούν να έχουν επαφή μεταξύ τους και ποια θα αγνοούν το ένα το άλλο. Κάθε Gameobject έχει ως προεπιλογή την μάσκα Default. Προκειμένου ένα gameobject να αλλάξει σε μία άλλη μάσκα, πρέπει μέσω του παραθύρου Inspector στον Editor ή προγραμματιστικά, να δηλώσει μία ήδη κατασκευασμένη ή να φτιάξει καινούργια. Το ποια μάσκα επικοινωνεί με ποια άλλη, ορίζεται μέσω του collision matrix, που βρίσκεται στο παράθυρο Project Settings του Editor. Είναι σημαντικό να προστεθεί, πως μία ιεραρχία από gameobjects μπορεί να αξιοποιεί διαφορετικές μάσκες, χωρίς να επηρεάζει το ένα το άλλο, με την προϋπόθεση ότι υπάρχουν διαφορετικά collider για την κάθε μάσκα. Οι μάσκες επιπέδου λειτουργούν και εκτός του collision detection, ως φίλτρα για τις κάμερες της σκηνής, και με ίδια λογική, μία μάσκα καθορίζει τι θα γίνεται rendering στο πεδίο όρασης της κάμερας που χρησιμοποιεί την μάσκα.

3.9 Order of execution for event functions

Οποιαδήποτε ενέργεια λαμβάνει χώρα κατά την εκτέλεση μιας εφαρμογής κατασκευασμένη σε Unity, η χρονική στιγμή εκτέλεσης της τοποθετείται εντός ενός event timeline, μίας προκαθορισμένης, από την Unity, ακολουθίας γεγονότων. Η ακολουθία αυτή συμπεριλαμβάνει και ταξινομεί όλα τα είδη λειτουργιών που υπάρχουν και όπου ποικίλουν από το rendering τον γραφικών στοιχείων στην οθόνη, μέχρι την επεξεργασία των input από τον χρήστη. Η πλειονότητα των events μπορεί να αξιοποιηθεί από τα scripts, ώστε να ορίζει ο προγραμματιστής σε ποια χρονική στιγμή μιας επανάληψης της ακολουθίας θα εκτελεστεί ο κώδικας. Για όλες τις υπόλοιπες λειτουργίες των integrated components και μερικές ειδικές περιπτώσεις, όπως εντολές πού κάνουν χρήση του physics system της Unity, η θέση τους στο timeline προσδιορίζεται αυτόματα. Είναι σημαντικό να προστεθεί πως η κλήση ενός event της ακολουθείας, γίνεται σε όλα τα scripts και αντικείμενα που το χρησιμοποιούν, πριν κληθεί το επόμενο event στην σειρά της ακολουθίας με τον ίδια λογική. Τα πιο σημαντικά events της ακολουθίας, αποτελούν το Update και το FixedUpdate, τα οποία είναι τα κυρίως νήματα της εφαρμογής. Είναι σημαντικό να επισημανθεί πως τα νήματα αυτά, δεν τρέχουν παράλληλα, αλλά σε σειρά εναλλάξ και με διαφορετική συμπεριφορά, κυρίως ως προς τον αριθμό επαναλήψεων και το χρονικό περιθώριο εκτέλεσης.



Εικόνα 3.6 – Διάγραμμα ροής των events της Unity

3.10 Update

Το Update event αποτελεί την καρδιά της εφαρμογής και η λογική εκτέλεσής του είναι να κάνει όσο το δυνατόν περισσότερες προσπελάσεις σε σύντομο χρονικό διάστημα. Συνεπώς ο αριθμός επαναλήψεων του Update δεν είναι σταθερός και αλλάζει δυναμικά σε κάθε frame, διότι προσαρμόζεται ανάλογα με το φόρτο που έχει και την διαθέσιμη επεξεργαστική ισχύς που παρέχεται, ώστε να αποφεύγει τελείως να καθυστερήσει την εκτέλεση της εφαρμογής, αποδίδοντας παράλληλα τον μέγιστο αριθμό επαναλήψεων. Η σωστή αξιοποίηση του Update προϋποθέτει πως οι λειτουργίες που εκτελούνται είναι ανεξάρτητες του χρόνου έναρξης και ολοκλήρωσης, καθώς όπως εξηγήθηκε, δεν είναι σταθερός. Παρόλα αυτά η Unity παρέχει μία μεταβλητή που υπολογίζει σε millisecond την διαφορά χρόνου εκτέλεσης μεταξύ της παρούσας και προηγούμενης επανάληψης του Update, το time.deltaTime. Αυτό δίνει την δυνατότητα να μπορούν να προσαρμόζονται οι υπολογισμοί μέσα στο Update ανάλογα με αυτή την τιμή αυτή και να δίνουν ίδια αποτελέσματα σε σταθερό χρονικό διάστημα. Μία από τις βέλτιστες πρακτικές αυτής της τεχνικής, είναι η υλοποίηση του animation του παιχνιδιού εντός του Update, το οποίο κατά προτίμηση είναι επιθυμητό να τρέξει όσο το δυνατόν περισσότερες φορές το δευτερόλεπτο, για να αποδώσει την κίνηση εξομαλυμένα και σε σταθερό χρόνο, αλλά και σταθερή μεταβολή των keyframes του animation στον χρόνο αυτό. Το time.deltaTime δεν έχει τέλεια ακρίβεια και πολλές φορές έχει απώλειες που προκύπτουν από πληθώρα προβλημάτων, με την πιο σύνηθες να είναι τα garbage collection spikes. Συνεπώς δεν πρέπει να χρησιμοποιείται για λειτουργίες που προσδοκάτε απόλυτη συνέπεια και ντετερμινισμός, καθώς σε ένα σοβαρό framerate spike, το Update θα κάνει skip ένα οι περισσότερα frames ώστε να μην επιβαρυνθεί περαιτέρω η εφαρμογή.

3.11 Fixed Update

Το FixedUpdate λειτουργεί αντίθετα από το Update. Η λογική του βασίζεται στην εκτέλεση λειτουργιών σε συγκεκριμένο χρονικό διάστημα. Η μεταβλητή Fixed Timestep καθορίζει σε millisecond, το χρονικό διάστημα μέχρι την επόμενη επανάληψη. Η αύξηση της τιμής του αντιστοιχεί σε λιγότερους υπολογισμούς σε συγκεκριμένο χρόνο, ενώ η μείωση του αντιστοιχεί σε περισσότερους . Το FixedUpdate είναι το πιο συνεπής νήμα που παρέχεται από το Engine, όχι μόνο για την σταθερότητα του στον χρόνο, αλλά και για την ιδιότητά του να εκτελεί κώδικα κάτω από οποιεσδήποτε συνθήκες. Το γεγονός αυτό είναι βολικό για τον προγραμματιστή, καθώς γνωρίζει πως ότι κληθεί εντός του FixedUpdate, θα ολοκληρωθεί επιτυχώς η εκτέλεσή του. Το χαρακτηριστικό αυτό όμως μπορεί να έχει αρνητική επίδραση, εάν το σύνολο των υπολογισμών που γίνονται στο FixedUpdate δεν ολοκληρώνονται μέσα στο χρονικό περιθώριο που προσδιορίζεται από το Fixed Timestep. Σε αυτήν την περίπτωση η εφαρμογή θα "παγώσει" οποιαδήποτε άλλη λειτουργία, μέχρι να ολοκληρωθούν οι υπολογισμοί. Είναι εξαιρετικά σημαντικό να επισημανθεί ότι, μολονότι οι ενέργειες του FixedUpdate γίνονται με σταθερή διαφορά, δεν μπορεί να εκτελέσει ποτέ κώδικα σε συντομότερο χρόνο από το Fixed Timestep και συνεπώς δεν είναι το κατάλληλο για ταχύς υπολογισμούς.

3.12 Αξιοσημείωτα Events

Άλλα σημαντικά events στην ακολουθία είναι μεταξύ άλλων :

- **Awake** : Καλείται μία φορά στην ζωή του αντικείμενο κατά την δημιουργία.
- **Start** : Καλείται μία φορά στην ζωή του αντικείμενο κατά την δημιουργία και το Script πρέπει να είναι ενεργοποιημένο.
- **OnEnable** : Καλείται κάθε φορά που το αντικείμενο ενεργοποιείται.
- **OnDisable** : Καλείται κάθε φορά που το αντικείμενο απενεργοποιείται.
- **OnDestroy** : Καλείται όταν το αντικείμενο καταστραφεί
- **OnCollisionEnter** : Καλείται μετά από κάθε FixedUpdate, όταν το αντικείμενο κάνει επαφή με άλλα αντικείμενα για πρώτη φορά. Απαιτείται τα αντικείμενα να έχουν Collider στην ιεραρχία του Gameobject που είναι προσαρτημένα.
- **OnCollisionStay** : Καλείται μετά από κάθε FixedUpdate, για κάθε αντικείμενο που διατηρεί επαφή με το αντικείμενο. Απαιτείται τα αντικείμενα να έχουν Collider στην ιεραρχία του Gameobject που είναι προσαρτημένα.
- **OncollisionExit** : Καλείται μετά από κάθε FixedUpdate, όταν το αντικείμενο χάσει επαφή με άλλα αντικείμενα. Απαιτείται τα αντικείμενα να έχουν Collider στην ιεραρχία του Gameobject που είναι προσαρτημένα.
- **yield WaitForFixedUpdate** : Καλείται μετά την εκτέλεση του FixedUpdate. Μπορεί να χρησιμοποιηθεί μόνο μέσα σε Coroutine.
- **OnDrawGizmos** : Καλείται μόνο στον Editor της Unity και χρησιμοποιείται γιά visual debugging.
- **OnInputXXX** : Καλείται όταν δοθεί Input από πληκτρολόγιο, ποντίκι ή gamepad.

3.13 Δημιουργία νέου Project

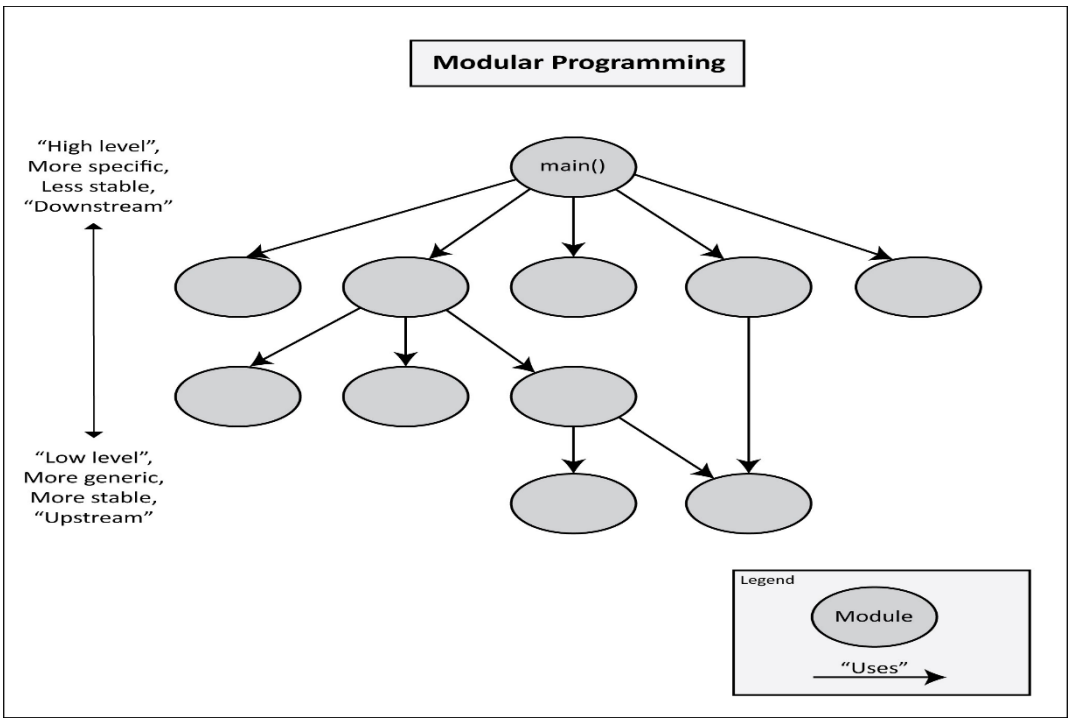
Ως ένα απλό παράδειγμα λειτουργίας της μηχανής θα δημιουργηθεί ένα Project, με την έκδοση της Unity που κατασκευάστηκε το module και θα εκτυπώνει το μήνυμα Hello World, στην κονσόλα αποσφαλμάτωσης. Αρχικά θα δημιουργηθεί ένα νέο Project με την χρήση του Unity Hub Project Manager. Όταν ανοίξει το περιβάλλον του Editor θα δημιουργηθεί ένα καινούργιο gameobject, πατώντας δεξί κλικ στο κενό πλαίσιο του παραθύρου Hierarchy και κάνοντας κλικ create empty στην λίστα που θα ανοίξει. Έπειτα θα δοθεί το όνομα HelloWorldGOBJ, κάνοντας δεξί κλικ στο μόλις δημιουργημένο gameobject και πατωντας Rename. Στην συνέχεια θα δημιουργηθεί ένα καινούργιο C# script πατώντας στο παράθυρο Project δεξί κλικ->create->C# Script και θα δοθεί το όνομα Scr_HelloWorld. Κάνοντας διπλό αριστερό κλικ πάνω στο script θα γίνει εκκίνηση του Visual Studio Community για επεξεργασία του script. Εντός του προκατασκευασμένου Function Start(), θα προστεθεί η γραμμή κώδικα : Debug.Log (“Hello World from gameobject : ”+gameobject.name); και θα αποθηκευτεί το script πατώντας Ctrl+S. Επιστρέφοντας στον Editor, με την ολοκλήρωση της αυτόματης μεταγλώττιση του script που τροποποιήθηκε, θα γίνει πρόσθεση του script στο HelloWorldGOBJ, σέρνοντας το script από το παράθυρο του Project στο gameobject στο παράθυρο του Hierarchy και θα γίνει εκκίνηση του προγράμματος πατώντας το κουμπί Play στο παράθυρο Game. Στο παράθυρο αποσφαλμάτωσης θα εμφανιστεί το μήνυμα : Hello World from gameobject : HelloWorldGOBJ.

4 Σχεδιασμός

Η παρούσα πτυχιακή με τη χρήση των εργαλείων της Unity και την προγραμματιστική γλώσσα C#, επιδιώκει την υλοποίηση ενός module για το περιβάλλον της Unity. Ο σκοπός αυτού του module είναι η εύκολη και γρήγορη ενσωμάτωση του σε οποιοδήποτε τύπου project, καθώς και η ευέλικτη χρήση και παραμετροποίηση του για εξυπηρέτηση μεγάλου εύρους λειτουργιών χωρίς να επιβαρύνεται η απόδοση. Τα πλεονεκτικά χαρακτηριστικά αυτά, προϋποθέτουν την χρήση εννοιών και την εφαρμογή τεχνικών και μεθόδων που αναφερθήκαμε σε προηγούμενα κεφάλαια.

4.1 Διαίρει και βασίλευε (Custom type modular design)

Προκειμένου ένα module να αντικαταστήσει μεγάλο εύρος μεμονωμένων προγραμματιστικών λύσεων, χρειάζεται να μπορεί να ενεργοποιεί, να απενεργοποιεί και να προσθαφαιρεί λειτουργίες και ιδιότητες δυναμικά, ούτως ώστε να προσαρμόζεται στις ανάγκες του development και του design.



Εικόνα 4.1 – Παράδειγμα ροής αρθρωτού προγράμματος

Αυτό επιτυγχάνεται με την χρήση της σχεδιαστικής μεθόδου του αρθρωτού προγραμματισμού, δηλαδή την αποσύνθεση ενός προγράμματος σε μικρότερα αυτόνομα υποπρογράμματα που θα εξυπηρετούν σε συνεργασία μεταξύ τους, τον σκοπό του αρχικού προγράμματος. Κάθε υποπρόγραμμα ή αλλιώς component στο περιβάλλον της Unity, εξυπηρετεί μια λειτουργία χωρίς να έχει επαφή με τα άλλα υποπρογράμματα άμεσα και ανταλλάσσει δεδομένα μέσω ενός κεντρικού υποπρογράμματος-διαχειριστή που έχει ως σκοπό την ομαλή λειτουργία μεταξύ των άλλων υποπρογραμμάτων. Αυτό συμβάλλει για δύο λόγους:

- Ο πρώτος είναι πως η απαλοιφή ή ανακατασκευή ενός υποπρογράμματος δεν θα προκαλέσει δυσλειτουργία σε όλα τα υπόλοιπα, καθώς κανένα υποπρόγραμμα δεν έχει αναφορά στα υπόλοιπα
- Ο δεύτερος λόγος είναι ότι οποιαδήποτε αλλαγή στον τρόπο που επικοινωνούν τα υποπρογράμματα μεταξύ τους θα γίνει μόνο μέσω του

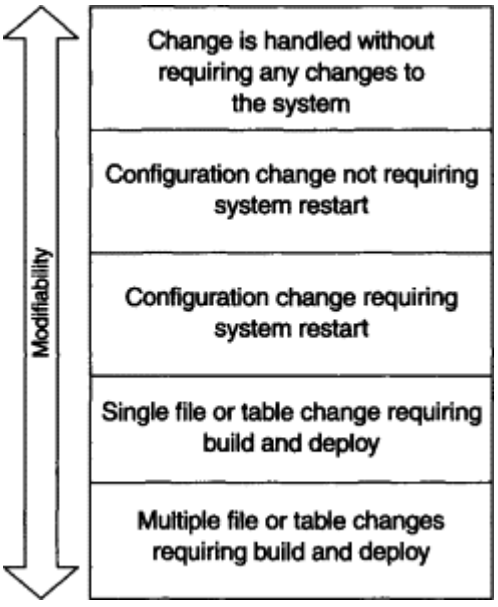
υποπρογράμματος-διαχειριστή αποφεύγοντας με αυτόν τον τρόπο χρονοτριβή με αλλαγές στα υπόλοιπα.

Τα υποπρογράμματα σημειωτέων δεν έχουν αναφορά ούτε στο υποπρόγραμμα-διαχειριστή αλλά μόνο το ανάστροφο, δηλαδή ο διαχειριστής "ακούει" οποιαδήποτε ενέργεια γίνεται στα υποπρογράμματα και μεταβιβάζει δεδομένα μεταξύ αυτών.
Σημαντικό είναι να αναφερθεί πως όταν δεν είναι δυνατή αυτή η μονόδρομη επικοινωνία, παραδείγματος χάρη όταν γίνεται εισαγωγή δεδομένων από το πληκτρολόγιο, ο βέλτιστος τρόπος υλοποίησης στα πλαίσια του αρθρωτού προγραμματισμού, είναι η χρήση διεπαφών, που παρόλο που δεν καθιστούν το υποπρόγραμμα απόλυτα αυτόνομο, του επιτρέπουν να λειτουργεί δυναμικά.

4.2 Μέγιστος διαθέσημος βαθμός τροποποίησης

Η επόμενη σχεδιαστική μέθοδος που εφαρμόζεται, είναι η αυξημένη τροποποιητικότητα και αφορά κυρίως τα υποπρογράμματα. Η αύξηση του βαθμού τροποποίησης ενός προγράμματος στοχεύει στην επιτέλεση διαφορετικών ενεργειών, μέσω της αλλαγής των τιμών των πεδίων του προγράμματος η /και την κλήση συναρτήσεων αυτού, με διαφορετικές τιμές στις παραμέτρους.

Αυτό προϋποθέτει ότι το πρόγραμμα αποτελείται από ένα σύνολο μεθόδων, των οποίων οι τιμές που καθορίζουν το αποτέλεσμα, πεδία ή παράμετροι, είναι τροποποιήσιμες μεταβλητές, που μπορούν να αλλάξουν τιμή, είτε κατά βούληση του προγραμματιστή (πεδία μόνο) ή από άλλα προγράμματα που έχουν αναφορά σε αυτό. Η δόμηση ενός προγράμματος κατά αυτόν τον τρόπο, δηλαδή να επιλύει διαφόρου τύπου προβλήματα μέσω τροποποιήσιμων πεδίων και συναρτήσεων χωρίς να χρειάζεται περαιτέρω αποσύνθεση σε μικρότερα προγράμματα, αυξάνει την πολυχρησιμότητα του προγράμματος και αποτελεί μία εξαιρετική τακτική για εύκολη αποσφαλμάτωση και γρηγορότερη καινοτομία, με την προϋπόθεση ότι δεν εφαρμόζεται παραεκτεταμένα καθώς αυξάνει παράλληλα την πολυπλοκότητα του. Η σωστή εφαρμογή της εν λόγω σχεδιαστική μεθόδου βασίζεται στην ισορροπία, ούτος ώστε να μην επιδρά περισσότερο αρνητικά παρότι θετικά στην εφαρμογή.



Εικόνα 4.2 – Παράδειγμα αυξομείωσης του modifiability

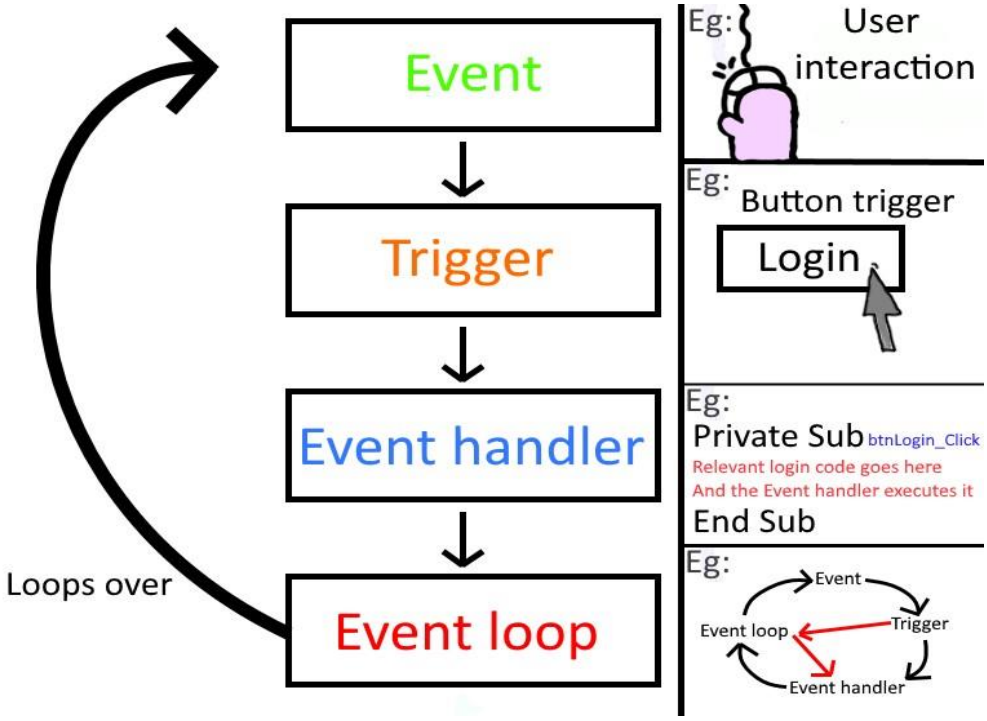
Ο βαθμός τροποποίησης του προγράμματος προσαρμόζεται βάση κάποιων κριτηρίων:

- ✓ Πόσο συχνά εκτιμάται ότι πρέπει να γίνουν αλλαγές στο πρόγραμμα.
- ✓ Σε τι βάθος γίνονται αυτές οι αλλαγές.
- ✓ Ποιος θα κάνει την αλλαγή (προγραμματιστής, σχεδιαστής γραφικών, σχεδιαστής level).
- ✓ Πόσο αρθρωτά ανεπτυγμένο είναι το πρόγραμμα.
- ✓ Πόσο συχνά παρουσιάζονται οντότητες τροποποιήσιμων μελών του προγράμματος.
- ✓ Πόσο αυξάνεται η επαναχρησιμότητα των τροποποιήσιμων μελών.

4.3 Event Oriented Programming

Τα προγράμματα με γραφικό περιβάλλον και συγκεκριμένα τα Video Games τρέχουν πάνω σε νήματα που εκτελούν χιλιάδες υπολογισμούς το δευτερόλεπτο. Η χρήση αυτών των νημάτων είναι απαραίτητη προϋπόθεση για την υλοποίηση των προγραμματιστικών λύσεων που θα επιτυγχάνουν τους προοριζόμενους στόχους ενός τέτοιου προγράμματος. Προκειμένου μία προγραμματιστική λύση να αξιοποιήσει με βέλτιστο τρόπο τους διαθέσιμους πόρους, δεν αρκείται μόνο στο πώς θα υλοποιηθεί, αλλά και στο πότε θα χρησιμοποιηθεί από το πρόγραμμα, με σκοπό να αποφεύγεται η αχρείαστη σπατάλη πόρων. Το Event Oriented Programming αποτελεί μία αναπτυξιακή μέθοδο, όπου ο έλεγχος της ροής του προγράμματος καθορίζεται από την ύπαρξη γεγονότων. Η ανάπτυξη πολύπλοκων προγραμμάτων κατά αυτή την λογική εξασφαλίζει βελτιωμένη απόδοση, καθώς οι περισσότερες ενέργειες εκτελούνται μόνο όταν χρειαστούν, έναντι του να εκτελούνται ή να ελέγχουν πότε πρέπει εκτελεστούν με κάθε προσπέλαση του νήματος. Η αρχιτεκτονική του του Event Oriented Programming βασίζεται σε δύο τύπους μεθόδων:

- Τους Event Handlers : δηλαδή τις μεθόδους που διαχειρίζονται τα γεγονότα και εκτελούνται στην κυρίως ακολουθία.
- Τους Event Listeners : δηλαδή τις μεθόδους που έχουν κάνει εγγραφή σε κάποιον Event Handler και εκτελούνται όταν εκτελεστεί αυτός.



Εικόνα 4.3 – Διάγραμμα λειτουργείας ενός event

Οι Event Handlers μπορούν να δέχονται παραμέτρους με την προϋπόθεση ότι οι Event Listeners που έχουν κάνει subscribe σε αυτούς δέχονται ίδιο αριθμό και τύπο παραμέτρων. Αυτό επιτρέπει πιο δυναμική λειτουργία μεταξύ Event Handlers και Listeners καθώς οι Handlers κατά την κλήση τους θα περάσουν τις τιμές με τις οποίες κλήθηκαν στους εγγεγραμμένους Listeners. Η κλήση ενός Event Handler από έναν Event Listener που κλήθηκε αυτόματα από άλλον Event Handler, δεν απαγορεύεται δίνοντας την δυνατότητα δημιουργίας μιας αλυσίδας γεγονότων. Το Event Oriented Programming δεν προσφέρει μονάχα καλύτερες επιδόσεις, αλλά και καλύτερη δυναμική επικοινωνία μεταξύ χρήστη και προγράμματος, καθώς επίσης έχει αντίστοιχα προτερήματα με τον αρθρωτό προγραμματισμό, λόγω της ευελιξίας προσθαφαίρεσης Listeners σε Handlers, ακόμη και κατά την εκτέλεση του προγράμματος.

5 Δομή

Η δομή του module που κατασκευάστηκε ακολουθεί τις αναπτυξιακές μεθόδους που αναλύθηκαν και αποσκοπεί στην χρήση του ως χειριστή χαρακτήρα πολλαπλών χρήσεων Το module στο μεγαλύτερο κομμάτι αποτελείται από τρία Unity Components γραμμένα σε C#. Τα Components αυτά είναι τα: Scr_Sync, Scr_Physics, Scr_Mechanics. Όλα τα components έχουν ως βάση το class MonoBehaviour της Unity που δίνει την δυνατότητα προσάρτησης ενός script σε gameobject και ουσιαστικά το μετατρέπει σε Unity component. Για την προσομοίωση των physics χρησιμοποιούνται συγκεκριμένες λειτουργίες από Built-In Components της Unity, καθώς και μία ιεραρχία από gameObjects στην οποία είναι μοιρασμένα αυτά τα components για λόγους που θα αναλυθούν παρακάτω. Συνεπώς τα τρία scripts σε συνδυασμό με το component rigidbody και τα τρία components capsule collider σχηματίζουν το gameobject εν ονόματι Character_Controller με την εξής ιεραρχία:

-Character_Controller: Transform, Rigidbody, Scr_Sync, Scr_Physics, Scr_Mechanics
    -CollisionObjs : (none)
        -CharacterCollider : Capsule Collider
        -WolrdCollider : Capsule Collider
        -ForceFieldCollider : Capsule Collider

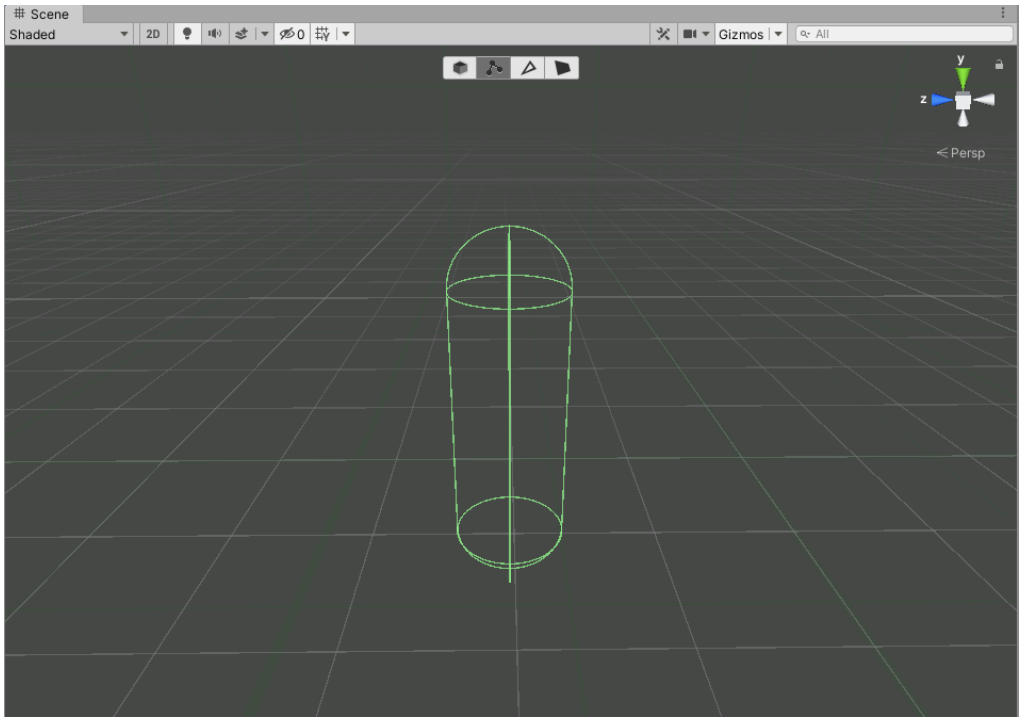
Είναι σημαντικό να αναφερθεί πως το Transform Component είναι παρών σε κάθε gameobject της ιεραρχίας αλλά χρησιμοποιείτε μόνο το πρώτο, και πως η χρήση του gameobject CollisionObjs χρησιμοποιείται μόνο για grouping χωρίς άλλη πρακτική χρήση. Επίσης το module σαν οντότητα υλοποιείται ως το σύνολο των προκατασκευασμένων και μη components αλλά και του Gameobject Character_Controller με την εν λόγω ιεραρχία.

5.1 Ιεραρχία Gameobject

Η Ιεραρχία που ακολουθείται στα gameobjects παιδιά του gameobject Character_Controller εξυπηρετεί έμμεσα πληθώρα λειτουργιών για το module. Το κεντρικό gameobject περιέχει όλα τα components που έχουν κατασκευαστεί στην C# μαζί με τα components της Unity Rigidbody και Transform. Τα τρία gameobjects παιδιά περιέχουν ένα Capsule Collider το καθένα τα οποία εξυπηρετούν διαφορετικούς σκοπούς. Το κάθε Capsule Collider ανήκει σε διαφορετική μάσκα επιπέδου. Οι μάσκες αυτές έχουν οριστεί να δημιουργούν επαφές, συγκρούσεις και ανταλλαγές δυνάμεων με συγκεκριμένες μάσκες ενώ να αγνοεί τις υπόλοιπες.

5.2 CharacterCollider

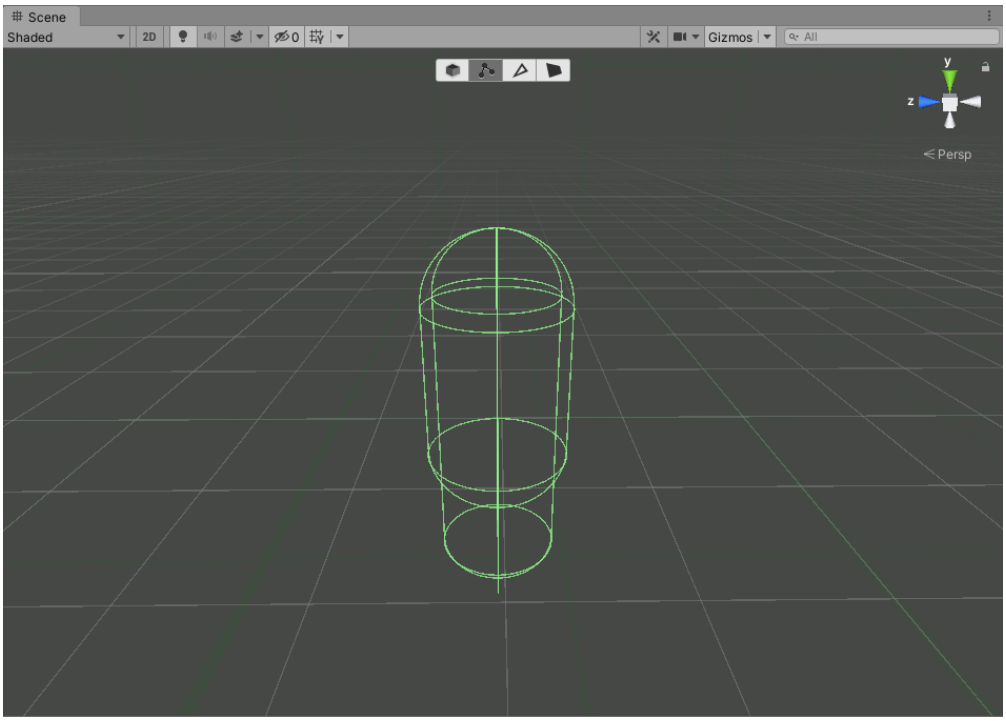
Το πρώτο σε σειρά capsule ανήκει στο gameobject CharacterCollider και έχει την μάσκα επιπέδου Character η οποία δεν κάνει επαφή με τίποτα εκτός από meshes με την μάσκα Force_Field. Το collider λειτουργεί κατά κύριο λόγω ως placeholder για τα άλλα δύο Collider καθώς επίσης συνεισφέρει έμμεσα στην λειτουργικότητα του Force_Field Capsule. Η αναπαράσταση του CharacterCollider capsule αποδίδει τις προδιαγραφές του χαρακτήρα στον χώρο, του οποίου τα μεγέθη ύψους και πάχους της κάψουλας μπορούν να χρησιμοποιηθούν ως όρια για την οπτικοποίηση του χαρακτήρα με την χρήση τρισδιάστατων κινούμενων μοντέλων.



Εικόνα 5.1 – Το wireframe της κάψουλας Character Collider

5.3 WorldOnlyCollider

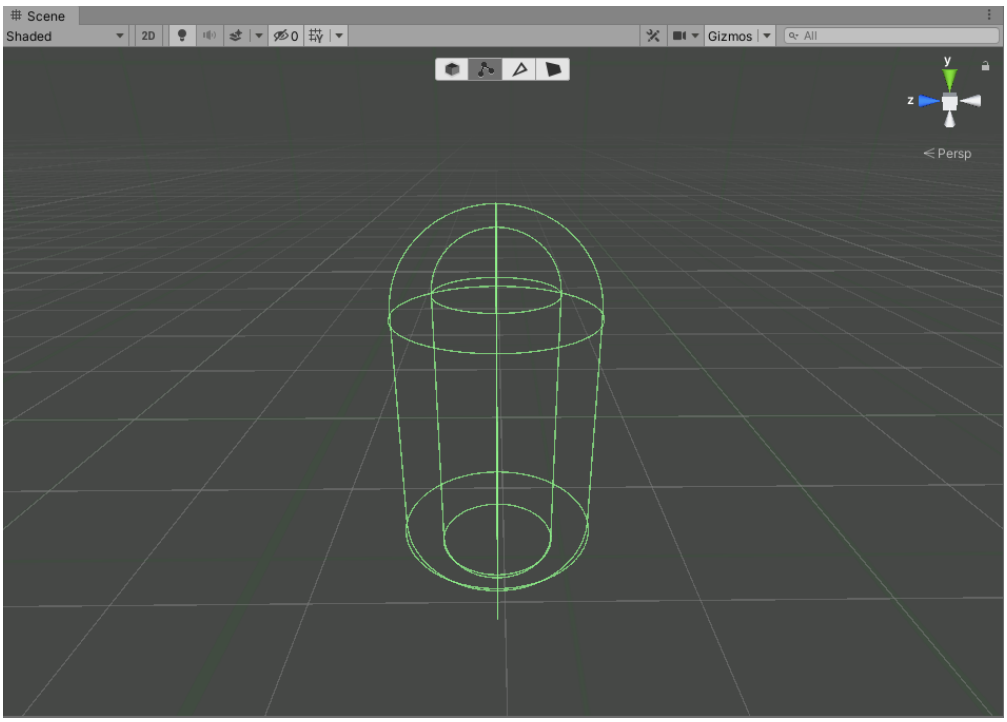
Επόμενο collider είναι το WorldOnlyCollider που χρησιμοποιεί την μάσκα World Only η οποία βλέπει μόνο αντικείμενα της μάσκας Default. Το ύψος και το πλάτος του υπολογίζεται από το script Scr_Physics κατά την δημιουργία του script και ορίζεται ως ένα θετικό ποσοστό που καθορίζει το πόσο σχετικά μεγαλύτερο είναι το πλάτος του από το character collider και ένα θετικό ποσοστό που καθορίζει το ύψος του σε σχέση με το character collider. Τα ποσοστιαία μεγέθη που μπορεί να πάρει ο Collider περιορίζονται ως προς το πλάτος καθώς δεν μπορεί να είναι μικρότερο του μηδενός και ως προς το ύψος, καθώς δεν πρέπει να είναι το ίδιο ή μεγαλύτερο με το character collider διότι απενεργοποιεί λειτουργίες των physics λόγο της επικάλυψης που θα προκύψει. Η λειτουργία του WorldOnlyCollider είναι η δημιουργία επαφών και συγκρούσεων μεταξύ αυτού και ως επι των πλείστων στατικών, αντικειμένων όπως τοίχους, οροφές, εδαφικές εγκαταστάσεις και γενικά colliders απλής γεωμετρίας.



Εικόνα 5.2 – Το wireframe της κάψουλας World Collider με επαυξημένη περιφέρεια και μειωμένο ύψος σε σχέση με το Character Collider

5.4 ForceFieldCollider

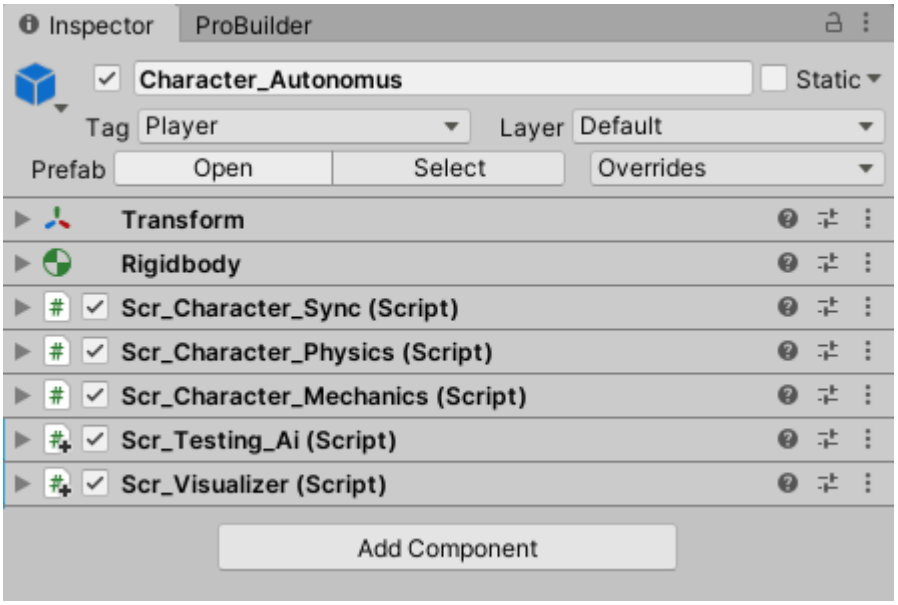
Τέλος το ForceFieldCollider ανήκει στην μάσκα Force_Field η οποία κάνει επαφές μόνο με την μάσκα character. Το ύψος και το πλάτος του ορίζεται ως το συνάθροισμα του ύψους και το πλάτους του character collider με μία δεκαδική τιμή, πεδίο της Scr_Physics. Ο σκοπός του είναι να εξαλείφει δυνάμεις κρούσης που προέρχονται από άλλους Character controllers και αυτό επιτυγχάνεται περικλείοντας τον Character Collider με τον Force_Field Collider οποίος θα έχει αυξημένη περιφέρεια κατά συγκεκριμένο βαθμό. Συνεπώς δύο Character Controllers με την ίδια περιφέρεια στον Force_Field Collider θα διασχίζονται στο ίδιο σημείο, καθώς η μάσκα Force_Field δεν έχει επαφή με colliders της ίδιας μάσκας, αλλά θα σταματάει πάνω στον Character Collider και στους δύο controllers ανεξαρτήτως του μέτρου δύναμης κρούσης. Αντιθέτως εάν ένας εκ των δύο έχει διαφορετική περιφέρεια στον Force_Field, αυτός με τον μεγαλύτερο βαθμό θα σπρώχνει τον άλλο με τον μικρότερο χωρίς να είναι δυνατό το αντίθετο.



Εικόνα 5.3 – Το wireframe της κάψουλας Force Collider με επαυξημένη περιφέρεια και ύψος σε σχέση με το Character Collider

5.5 C# Components

Τα τρία Scripts που καθορίζουν το μεγαλύτερο μέρος της συμπεριφοράς του module επικοινωνούν μεταξύ του μέσω της μεθόδου διαίρει και βασίλευε με το script εγκέφαλο Scr_Sync στο ρόλο του διαύλου επικοινωνίας. Το Scr_Sync έχει αναφορά στα άλλα δύο scripts, Scr_Physics και Scr_Mechanics, προκειμένου να αντλεί και να μεταβιβάζει δεδομένα από και σε αυτά. Τα δύο scripts αυτά λειτουργούν τελείως αυτόνομα με μοναδικό περιορισμό το Scr_Physics που χρειάζεται αναφορά σε ένα rigidbody Component ώστε να είναι αποτελεσματικό. Αυτό προϋποθέτει ότι η επικοινωνία με το Scr_Sync δεν είναι αμφίδρομη και τόσο η άντληση των δεδομένων όσο και η παραλαβή είναι διαδικασίες αυστηρά ελεγχόμενες από τα ίδια τα scripts. Σε αυτό το σημείο είναι σημαντικό να αναλυθεί τί και πώς αναλαμβάνει να διαχειριστεί το κάθε script καθώς και την συνεισφορά του στην ολοκλήρωση και την επιτυχία του module.



Εικόνα 5.4 – Τα components στο main gameobject

5.6 Scr_Physics

Το Scr_Physics έχει το ρόλο του διαχειριστή των φυσικών κανόνων που υπάγεται ο χειριστής χαρακτήρα. Αυτό σημαίνει ότι η λειτουργία του βασίζεται πάνω στην συνεχή και αδιάκοπη χρήση ντετερμινιστικών μεθόδων, που υπολογίζουν την επόμενη κατάσταση του χαρακτήρα στον χώρο. Η λογική αυτή προϋποθέτει ότι κανένας εξωτερικός παράγοντας δεν θα μπορεί να καλεί αυτές τις μεθόδους ή να αλλάζει τα αποτελέσματά τους άμεσα άλλα μόνο να τα μεταβάλει μέσω των διαθεσίμων δημοσίων πεδίων προς τροποποίηση. Οι μέθοδοι που υπολογίζουν αυτά τα αποτελέσματα, αντλούν τα δεδομένα προς επεξεργασία, από τα διαθέσιμους "αισθητήρες" τρισδιάστατου χώρου του module. Οι αισθητήρες αυτοί δημιουργούνται από το script βάση των προδιαγραφών των τριών colliders, τους οποίους αξιοποιούν μόνο για τον προσδιορισμό της θέσης τους στον χώρο και υπολογισμό επαφών και συγκρούσεων. Όλες οι άλλες λειτουργίες των αισθητήρων υλοποιούνται και χρησιμοποιούνται από το ίδιο το Script. Το Scr_Physics αποτελεί την σπονδυλική στήλη του module καθώς ελέγχει το πιο σημαντικό κομμάτι της συμπεριφοράς του και συνεπώς το πιο πολύπλοκο και απαιτητικό σε πόρους λόγω της των πολύπλοκων μαθηματικών υπολογισμών και της ανάγκης να εκτελείται σε κάθε physics frame. Ως προς τον βαθμό τροποποίησης, δίνεται η δυνατότητα ενεργοποίησης και απενεργοποίησης εκτέλεσης ορισμένων μεθόδων και η μεταβολή συντελεστών για διαφορετική συμπεριφορά.

5.7 Scr_Mechanics

Το Scr_Mechanics διαχειρίζεται ποικίλες ιδιότητες του Character_Controller. Η δομή του κατασκευάστηκε έτσι ώστε να υποστηρίζει λειτουργίες που είναι πιο κοντά στην λογική του παιχνιδιού παρά των τεχνικών λειτουργιών του. Παραδείγματος χάρη ορίζει πόσες φορές μπορεί να κάνει άλμα ο χαρακτήρας στον αέρα ή πόσο γρήγορα μπορεί να τρέξει αλλά δεν εκτελεί αυτές τις ενέργειες. Η λειτουργικότητά του βασίζεται κατά κύριο λόγο σε πεδία και μεθόδους επιστροφής τιμών, αξιοποιώντας παράλληλα μεθόδους επεξεργασίας δεδομένων οι οποίες εκτελούνται μόνο κατά ζήτηση ή κατά την πυροδότηση κάποιου γεγονότος στο οποίο είναι συνδεδεμένοι. Είναι σημαντικό να προστεθεί πως το εν λόγω script είναι αποκλειστικά event oriented και κάνει εκτεταμένη χρήση των delegates προκειμένου να υποστηρίξει αποδοτικά το module και να είναι πιο εύκολα τροποποιήσιμο ή και αντικαταστάσιμο. Η δομή αυτή ακολουθείται για τον λόγο ότι Scr_Mechanics, καθώς αποτελεί τον διαχειριστή της ευρύτερης έννοιας του χαρακτήρα και της λογικής του παιχνιδιού, επεκτείνεται και έξω από τον σκοπό του module, καθώς διαχειρίζεται και επεξεργάζεται δεδομένα και καταστάσεις, έμμεσα συνδεδεμένα ή και άσχετα με τον χειριστή χαρακτήρα. Συνεπώς, η μόνο κατά ζήτηση μεθοδολογία υλοποίησης των άμεσα συνδεδεμένων λειτουργιών, μειώνουν την πολυπλοκότητα και βελτιώνουν την συνοχή του module, χωρίς να χάνει την αυτονομία του το script.

5.8 Scr_Sync

Το Scr_Sync εμπεριέχει τον κώδικα επικοινωνίας μεταξύ των scripts του module και εξωτερικών παραγόντων με το module. Η δομή του συγκεκριμένου script περιλαμβάνει δύο βασικούς κανόνες. Ο πρώτος είναι ότι δεν μπορεί να περιέχει κανένα πεδίο, δημόσιο ή ιδιωτικό, καθώς ο σκοπός του είναι να μεταβιβάζει δεδομένα μεταξύ των scripts και όχι να αποθηκεύει τιμές αυτών με οποιονδήποτε τρόπο. Η μεθοδολογία αυτή υποστηρίζεται από το γεγονός ότι το εν λόγω script αποτελεί το τελευταίο γρανάζι της ομαλής λειτουργίας του module καθώς υλοποιεί την λογική που θα υποστηρίξει, όσων το δυνατό περισσότερο, αποκέντρωση λειτουργιών από ένα κεντρικό script και καθιστά το κάθε συνδεδεμένο script αυτόνομο στο πεδίο που αναλαμβάνει να καλύψει, παράγοντας παράλληλα το επιθυμητό αποτέλεσμα μέσω μη αναφοροποιημένης συνεργασίας με τα υπόλοιπα. Ο τρόπος που επιτυγχάνεται αυτό είναι η αναπτυξιακή μέθοδος event oriented programming που αναλύθηκε, καθώς όλα τα cross communication μεταξύ των Scripts αποτελούνε events τα οποία ο Scr_Sync συνδέει μεταξύ τους. Ο δεύτερος κανόνας προϋποθέτει πως όλες οι επικοινωνίες με εξωτερικά Script, εισερχόμενες και εξερχόμενες, επιτυγχάνονται μέσω διεπαφών. Με τον τρόπο αυτό απλουστεύεται η διαδικασία με την οποία ένα script εκτός του module κάνει τροποποίηση σε αυτό και ελέγχεται σε τι βαθμό μπορεί να το τροποποιήσει χωρίς να χρειάζεται να έχει αναφορά στο κάθε Component που θέλει να κάνει αλλαγή. Συμπερασματικά, είναι σημαντικό επισημανθεί πως οι κλήσεις μεθόδων που γίνονται στο Scr_Sync είναι αποκλειστικά τα υλοποιημένα interfaces που έχει, καθώς όλες οι άλλες μέθοδοι αποτελούνε events που αφορούν τα υπόλοιπα scripts και το Scr_Sync κάνει μόνο την σύνδεση μεταξύ Handlers και Listeners αυτών.

5.9 Κατηγοριοποίηση δομικών μελών

Ταξινομώντας τα δομικά μέρη που αναλύθηκαν προκύπτουν οι εξής έξι κατηγορίες:

- Οι οντότητες της σκηνής: δηλαδή το κυρίως gameobject prefab και η ιεραρχία από gameobjects παιδιά του.
- Οι οντότητες επεξεργασίας και αποθήκευσης δεδομένων: δηλαδή τα components του κάθε gameobject.
- Ο διαχειριστής των physics: δηλαδή το script component Scr_Physics.
- Ο διαχειριστής του Game Logic ή Game Mechanics: δηλαδή το Script Component Scr_Mechanics
- Ο συνδετήρας όλων των components και δίαυλο επικοινωνίας άλλων οντοτήτων με το module: δηλαδή το δηλαδή το Script Component Scr_Sync
- Οι εξωτερικοί παράγοντες: δηλαδή scripts που έχουν αναφορά στο Scr_Sync ή Components που αλλάζουν καταστάσεις ή πυροδοτούν γεγονότα του module.

6 Λειτουργικότητα

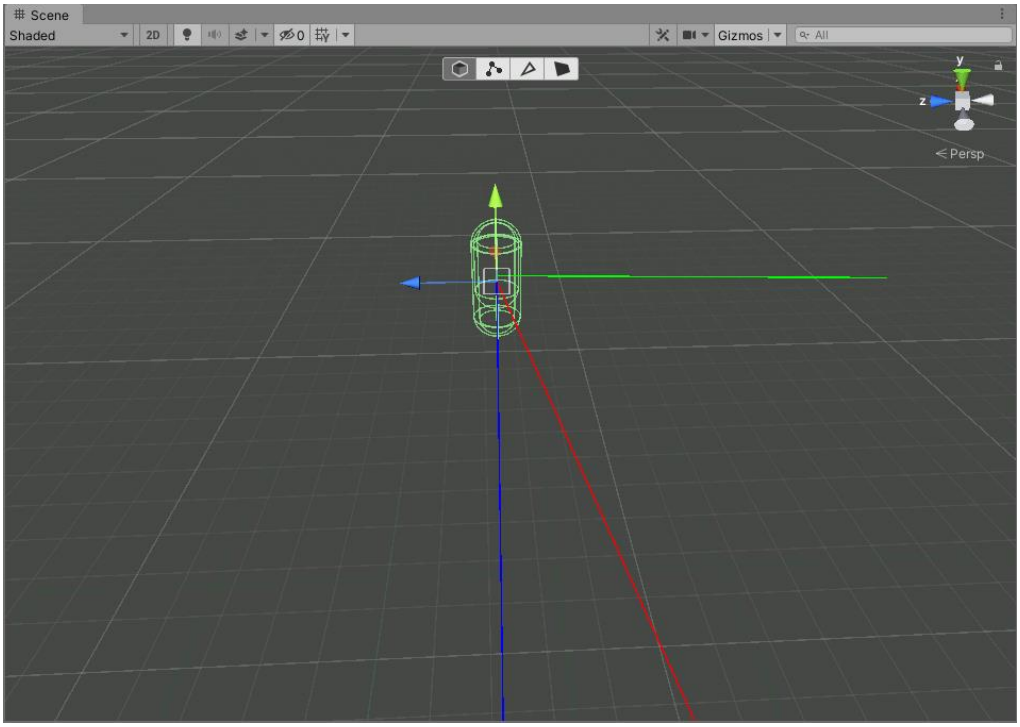
Ως χειριστής χαρακτήρα στην ανάπτυξη video παιχνιδιών, ορίζεται το κομμάτι ή τα κομμάτια κώδικα υπεύθυνα για την μετάβαση του χαρακτήρα στον τρισδιάστατο ή δισδιάστατο χώρο και την αλληλεπίδρασή του, κατά τις μεταβάσεις αυτές, με άλλους χαρακτήρες και οντότητες. Ουσιαστικά, το κομμάτι ή κομμάτια αυτά καθορίζουν τους "φυσικούς νόμους" κάτω από τους οποίους υπάγεται ο χαρακτήρας. Μολονότι η εν λόγω ορολογία δεν γίνεται απόλυτα σαφής ονοματικά, λόγο της φύσης των video games να προσομοιώνουν τους φυσικούς νόμους κατά επιλογή αντιρεαλιστικά, σε προγραμματιστικό επίπεδο καθορίζεται σε ακέραιο βαθμό ως προς τον τρόπο δόμησης και λειτουργίας της, βάση των επιθυμητών αποτελεσμάτων. Είναι σημαντικό να αναφερθεί πως τα συνθετικά του όρου υπάρχουν επίσης ως ορολογίες με δική τους σημασία που συνδέεται άμεσα με τον χειριστή χαρακτήρα. Ο χειριστής, σε προγραμματιστικό επίπεδο είναι σχεδόν πάντα αναπόσπαστο κομμάτι του χειριστή χαρακτήρα και προσδιορίζει πως και σε τι βαθμό ελέγχεται ο χαρακτήρας από τον παίκτη, ενώ ο χαρακτήρας παραπέμπει σε αυθαίρετη έννοια που σε προγραμματιστικό επίπεδο ορίζεται μόνο ονοματικά καθώς χρησιμοποιείται μόνο για ταυτοποίηση ή διαφοροποίηση μηχανισμών σχετικών με τον χειριστή χαρακτήρα και άλλων οντοτήτων. Συνεπώς, λαμβάνοντας υπόψη τους εν λόγω ορισμούς και τις ιδιότητές τους, η κατασκευή ενός χειριστή χαρακτήρα πολλαπλών χρήσεων στοχεύει να προσφέρει προγραμματιστική λύση για μεγάλο εύρος παιχνιδιών με διαφορετικούς χαρακτήρες και φυσικούς κανόνες καθώς επίσης και προσαρμόσιμο βαθμό ελέγχου για τον παίκτη.

6.1 Σύστημα ανάθεσης τρισδιάστατων προδιαγραφών

Η ανάθεση των τρισδιάστατων ορίων του controller ή με άλλα λόγια του μεγέθους του, αποτελεί μία δυναμική διαδικασία που εξαρτάται από τρεις απόλυτες δεκαδικές τιμές και δύο ποσοστιαίες τιμές που εξαρτώνται άμεσα από τις απόλυτες τιμές. Οι μέθοδοι που διαχειρίζονται τις αναθέσεις ποικίλουν ανάλογα με τις μεταβλητές που αλλάζουν, με πιο σημαντική μέθοδο AddTo_CharacterCollider_Ratio() η οποία αυξάνει ή μειώνει κατά ένα ποσοστό την αναλογία ύψους και πλάτους του Character Collider. Κάθε μέθοδος που τροποποιεί τις απόλυτες ή ποσοστιαίες τιμές τρέχει στο τέλος της εκτέλεσης της, την μέθοδο Adjust_Adjustible_Variables(). Η μέθοδος αυτή εκτός από το ότι εφαρμόζει τις αλλαγές των μεταβλητών στα αντίστοιχα colliders, φροντίζει να προσαρμοστούν όλες οι εξαρτώμενες από τα collider λειτουργίες, στις καινούργιες τιμές.

6.2 Τριδιανισματική ταχύτητα

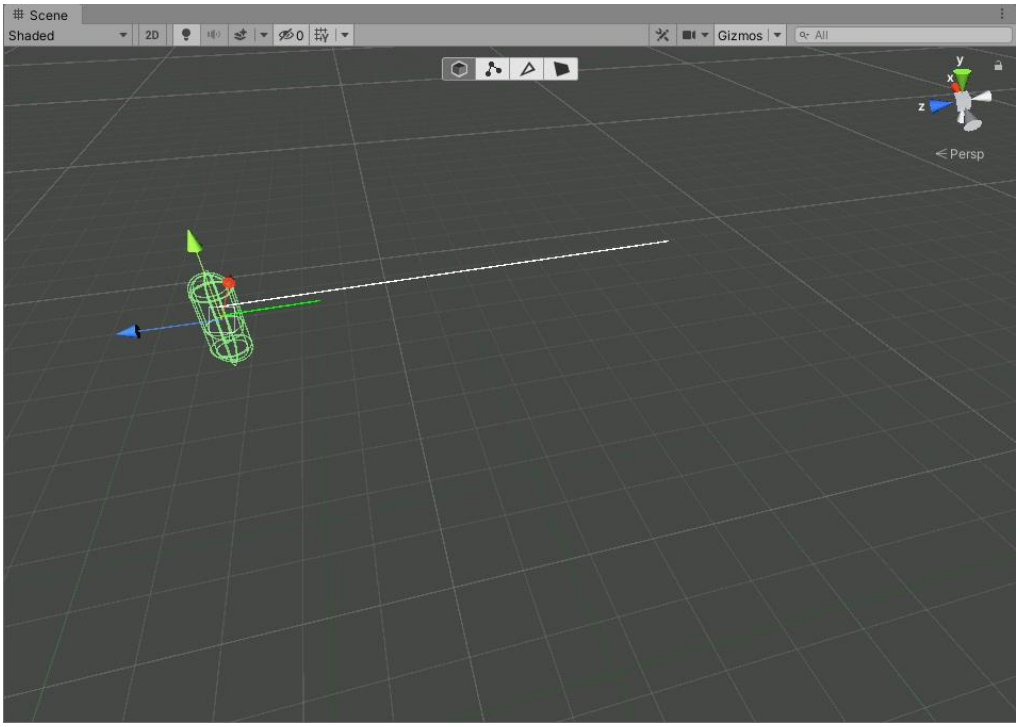
Το σύστημα μεταβολής της ταχύτητας του Character_Controller βασίζεται πάνω σε τρία διανύσματα, το move_velocity, το force_velocity και το inertia_velocity. Σε κάθε physics frame το velocity του rigidbody της επόμενης κατάστασης υπολογίζεται ως το συνάθροισμα των τριών διανυσμάτων. Το κάθε διάνυσμα αποτελεί ένα ιδιωτικό πεδίο τύπου Vectror3 που αποθηκεύει το αποτέλεσμα επεξεργασίας μιας ή περισσοτέρων μεθόδων και εξυπηρετεί έναν συγκεκριμένο τομέα της ακολουθίας υπολογισμού των physics του Character_Controller.



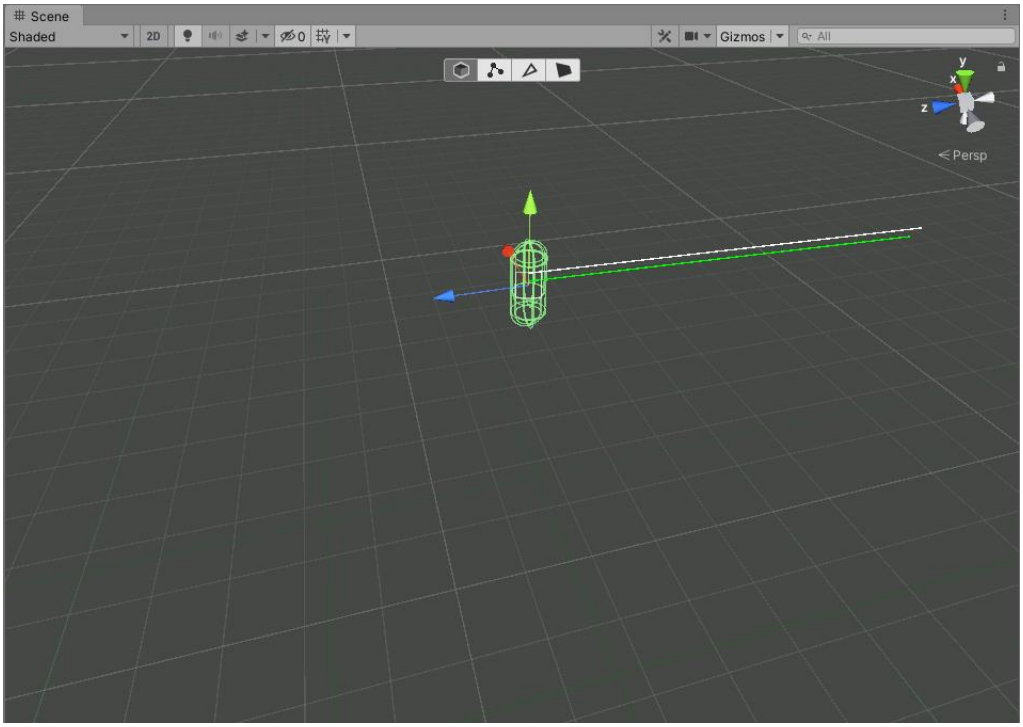
Εικόνα 6.1 – Το διάνυσμα ForceVel (μπλε) και το διάνυσμα MoveVel (πράσινο) συναθροίζονται και εκχωρούνται στο rigidbody.velocity (κόκκινο)

6.3 move_velocity

Το πεδίο διάνυσμα move_velocity αποθηκεύει την ταχύτητα κίνησης του Character_Controller και συγκεκριμένα την κίνηση που προκύπτει από τον ίδιο τον Character_Controller. Με ποιο απλά λόγια, επεξεργάζεται το πού προσπαθεί να κινηθεί ο ίδιος ο Character_Controller ασκώντας μόνος του την δύναμη ώθησης. Η Ενέργεια πραγματοποιείται έπειτα από εντολή του Scr_Sync στον οποίο έχει δοθεί εντολή από κάποιο εξωτερικό script Input Handler ή AI. Η μεταβολή του διανύσματος δεν γίνεται με ανάθεση τιμής στο ίδιο το πεδίο αλλά μέσω του δευτερεύων διανύσματος, target_move_velocity, στο οποίο ο Scr_Sync αναθέτει την επιθυμητή κατεύθυνση και ταχύτητα κίνησης και το Scr_Physics επιχειρεί να κάνει ισοβαθμήσει το πεδίο move_velocity μετακινώντας το, προς το επιθυμητό διάνυσμα, με σταθερό βήμα μεταβολής. Το βήμα μεταβολής αποτελεί δημόσιο πεδίο και αναπαριστά ποσοστό μεταβολής σε κάθε frame.



Εικόνα 6.2 – Το διάνυσμα TargetMoveVel (άσπρο) και το διάνυσμα MoveVel (πράσινο) στο πρώτο physics frame μετά την τροποποίηση του MoveVel



Εικόνα 6.3 – Το διάνυσμα TargetMoveVel (άσπρο) και το διάνυσμα MoveVel (πράσινο) στο εικοστό πέμπτο physics frame μετά την τροποποίηση του MoveVel

Η ανάθεση του target_move_velocity μπορεί να γίνει με δύο μεθόδους, την Set_Target_Move_Velocity και την Set_Target_Move_Velocity_Relative. Η πρώτη μέθοδος αναθέτει τιμές μέσω δύο ορισμάτων, του new_t_dir στο οποίο δέχεται ένα διάνυσμα, και του new_mag στο οποίο δέχεται το μήκος ή στην συγκεκριμένη περίπτωση την ταχύτητα. Ο λόγος που δέχεται το διάνυσμα χωριστά με το μήκος είναι ότι η συνάρτηση δέχεται απόλυτες τιμές κατεύθυνσης και τις μετατρέπει σε σχετικές με την κατεύθυνση και τον προσανατολισμό του Character_Controller. Συνεπώς το διάνυσμα πρέπει να πρώτα να γίνει normalize και μετά να ανακατευθυνθεί. Αυτό είναι αναγκαίο λόγω της πολύπλοκης μαθηματικής διαδικασίας ανακατεύθυνσης, η οποία επηρεάζεται από το μήκος όταν δεν είναι μοναδιαίο. Το μήκος πολλαπλασιάζεται έπειτα με το ανακατευθυνόμενο διάνυσμα και προκύπτει η επιθυμητή ταχύτητα. Η δεύτερη μέθοδος ακολουθεί την ίδια διαδικασία, με την μόνη διαφορά ότι δέχεται άλλα δύο μοναδιαία διανύσματα τα οποία χρησιμοποιεί για να κάνει ανακατεύθυνση. Τα διανύσματα αυτά είναι το πάνω και το μπροστά, και μέσω της μαθηματικής εξίσωσης το Cross Product, βγαίνει το διάνυσμα δεξιά ή αριστερά το οποίο χρειάζεται για να γίνει η ανακατεύθυνση. Το move_velocity μπορεί να λάβει επιθυμητές τιμές μέσω του target_move_velocity χωρίς να μεσολαβήσει κάποιο sanity check ενδιάμεσα. Όταν κάποια μέθοδος της ακολουθίας

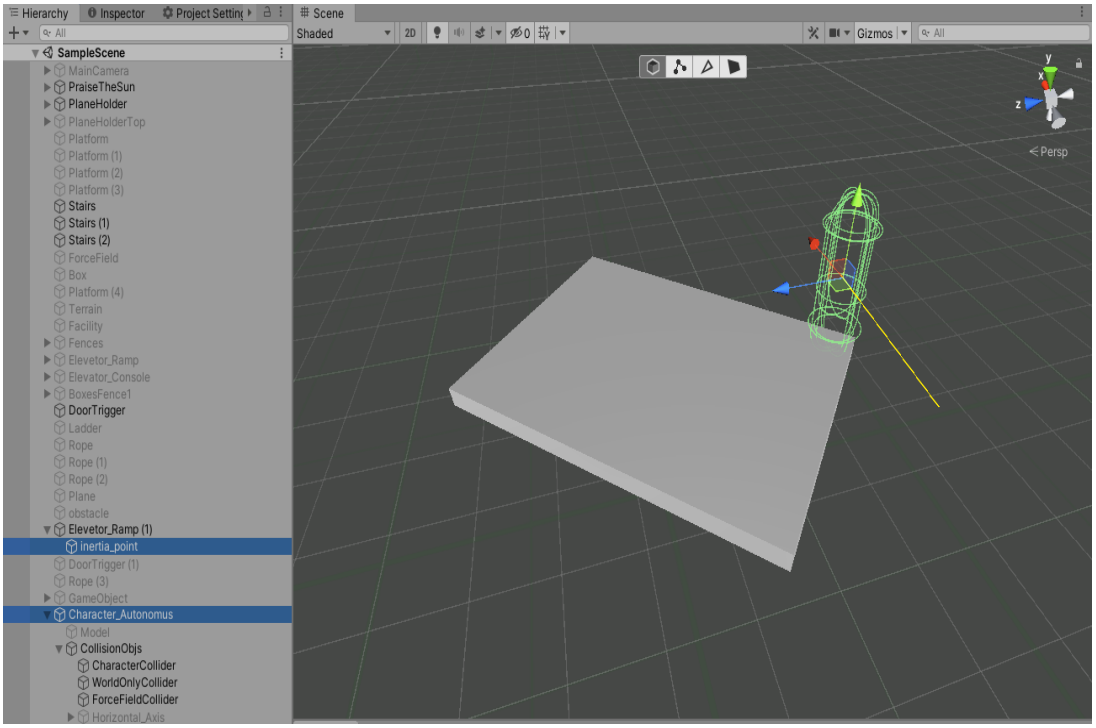
χρειάζεται να τροποποιήσει το move_velocity ακολουθεί την ίδια λογική. Δεδομένου ότι η μέθοδος προηγείται του υπολογισμού της τιμής του move_velocity τα αποτελέσματα θα εφαρμοστούν στο ίδιο frame.

6.4 force_velocity

Το πεδίο διάνυσμα force_velocity αποθηκεύει το διάνυσμα όλων των ασκούμενων δυνάμεων στον Character_Controller, εξαιρουμένου της κατά επιλογήν κίνησης και της αδράνειας. Το διάνυσμα μεταβάλει την τιμή του, είτε με ανάθεση ή με πρόσθεση διανύσματος στο ίδιο, μέσω της συνάρτησης Apply_Force_Vector που καλύπτει και τις δύο περιπτώσεις με την προαιρετική χρήση της παραμέτρου willOveride. Ο υπολογισμός της τιμής του διανύσματος σε κάθε frame προκύπτει μέσω της χρήσης ενός δευτερεύων διανύσματος το οποίο ονομάζεται constant_Force και αποθηκεύει τις σταθερές δυνάμεις που ασκούνται στον Character_Controller, παραδείγματος χάρη, τη βαρύτητα. Ο υπολογισμός γίνεται με τον ίδιο ακριβώς τρόπο που γίνεται μεταξύ του move_Velocity και του target_move_velocity με την μόνη διαφορά ότι το βήμα αναπαρηστά δεκαδικό και όχι ποσοστό. Όταν στο controller δεν ασκείται κάποια σταθερά δύναμη, δηλαδή το constant_Force force είναι 0, τότε force_velocity κινείται με σταθερό βήμα προς το μηδέν. Επομένως η τιμή του βήματος προσομοιώνει την επιτάχυνση ή επιβράδυνση των δυνάμεων που ασκούνται και με την δυνατότητα να αλλάζει τιμές κατά την εκτέλεση, μπορεί να προσομοιώσει διαφορετικές συνθήκες όπως τριβή εδάφους, αντίσταση του αέρα, επιτάχυνση της βαρύτητας κ.τ.λ. Σημειωτέων, το constant_Force μπορεί να λάβει τιμές επίσης, δεδομένου ότι το callback γίνεται επαναλαμβανόμενα εντός του physics frame διότι το constant_Force μηδενίζεται σε κάθε επανάληψη.

6.5 inertia_velocity

Το inertia_Velocity αποθηκεύει το διάνυσμα της αδράνειας του Character_Controller δηλαδή την δύναμη που ασκείται όταν ο controller αποτελεί μέρος μιας άλλης κινούμενης οντότητας. Ένα απλό παράδειγμα της χρήσης inertia_Velocity είναι ο Character_Controller να βρίσκεται επιβάτης σε κάποιο όχημα όπως ένα τρένο ή ένα ελικόπτερο. Ο υπολογισμός της δύναμης είναι διαφορετικός από τα άλλα δύο διανύσματα και προκύπτει με την χρήση δημιουργίας και μεταφοράς κληρονομικότητας gameobject στη σκηνή. Όταν μία οντότητα ζητήσει μέσω του Scr_Sync να γίνει ο Character_Controller μεταφερόμενο σώμα από αυτήν την οντότητα, το Scr_Physics δημιουργεί ένα καινούργιο άδειο gameobject ως παιδί του Character_Controller, και του αναθέτει, με σχετικές συντεταγμένες την τιμή μηδέν. Έπειτα μεταφέρει την κληρονομικότητα του άδειου gameobject στην οντότητα που έκανε το request χωρίς να αλλάξει τις συντεταγμένες του. Με την κίνηση της οντότητας τώρα, το άδειο gameobject θα μετακινηθεί μαζί του. Το διάνυσμα της απόστασης μεταξύ της θέσης δημιουργίας και της επόμενης θέσης θα ανατεθεί στο inertia_Velocity, μετά από προσαρμογή μήκους ως προς την μονάδα μέτρησης απόστασης σε physics scale του project που προκύπτει από το διάνυσμα διά το Fixed Timestep.



Εικόνα 6.4 – Το διάνυσμα InertiaVel επηρεάζεται από την περιστρεφόμενη ράμπα δημιουργώντας κεντρομόλο ταχύτητα.

Η επαναδημιουργία του αντικειμένου δεν είναι απαραίτητη σε κάθε frame, μόνο η μεταφορά του η οποία γίνεται στο PostFixedUpdate δηλαδή μόλις ολοκληρωθεί το physics frame για αποφυγή λαθών. Η καταστροφή του γίνεται με το πέρας της διαδικασίας. Η ενεργοποίηση ή απενεργοποίηση ενός τέτοιου γεγονότος, δηλαδή η δημιουργία διανύσματος αδράνειας από ξένο κινούμενο σώμα, μπορεί να γίνει και από το Scr_Physics αυτόνομα, αρκεί να υπάρχει η αναφορά στην οντότητα και η λογική εμπλοκής/απεμπλοκής.

6.6 OnFixedUpdateEnd

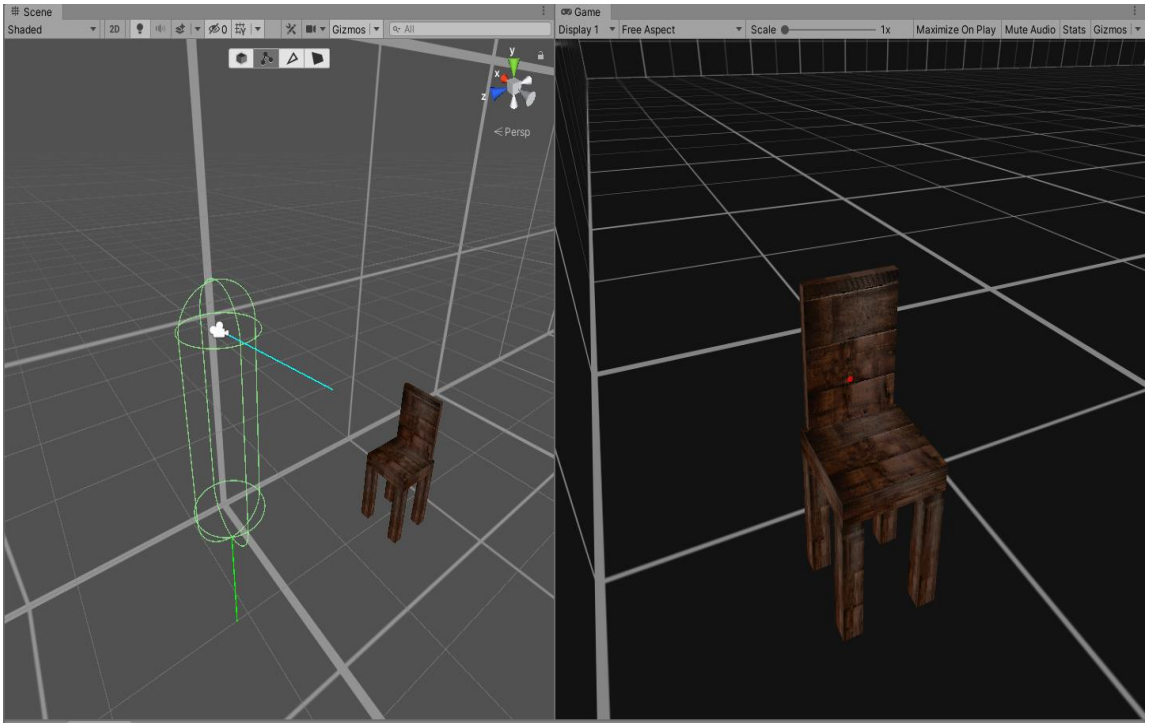
Η μεταβολή των διανυσμάτων δυνάμεων καθώς και άλλες λειτουργίες αποτελούν διαδικασίες που παράγουν έργο, χωρίς να λαμβάνουν τα άμεσα αποτελέσματα για έλεγχο εγκυρότητας αυτών. Αυτό συμβαίνει διότι το feedback από τους διαθέσιμους σένσορες του χώρου δεν παρέχει τα αποτελέσματα στον χρόνο που χρειάζονται για να γίνουν έλεγχοι εγκυρότητας. Αυτό είναι ένας περιορισμός από το physics engine της Unity το οποίο δεν επιτρέπει να γίνουν collision checks ή άλλες σχετικές με τα physics ενέργειες έξω από το physics frame. Μολονότι δύνανται να καθοριστεί η σειρά που μπορούν να γίνουν τα γεγονότα των physics, δεν δύνανται να γίνουν εκπρόθεσμα κάποιων hardcoded γεγονότων όπως οι αλγόριθμοι collision check της Unity και η διαδικασία αλλαγής της θέσης των rigidbody των gameobjects. Μία ενδιάμεση λύση προσφέρεται με την χρήση των coroutines της C# που επιτρέπουν να τρέξει ένα νήμα σε συγκεκριμένο χρόνο. Με την μέθοδο OnFixedUpdateEnd τύπου επιστροφής coroutine και την εντολή yield return new WaitForFixedUpdate() εκτελείται ένα κομμάτι κώδικα αμέσως μετά το τέλος του physics frame και αμέσως πριν τα κανονικά frames. Παρότι δεν αναιρούνται οι προηγούμενοι περιορισμοί με αυτήν την μέθοδο, γίνεται δυνατός ο έλεγχος των άμεσων αποτελεσμάτων του physics frame και διορθώσεις των συντελεστών υπολογισμού. Μία μέθοδος διόρθωσης που τρέχει μέσα στην OnFixedUpdateEnd είναι Vector_Correction_Sequence ή οποία ελέγχει το άθροισμα και την κατεύθυνση των μηκών των τριών διανυσμάτων δυνάμεων με το πραγματικό διάνυσμα κίνησης και προσαρμόζει τα τρία διανύσματα στις αντίστοιχες λογικές τιμές αν ανιχνευτεί μεγάλη απόκλιση

6.7 Προηγμένο σύστημα κατεύθυνσης και προσανατολισμού

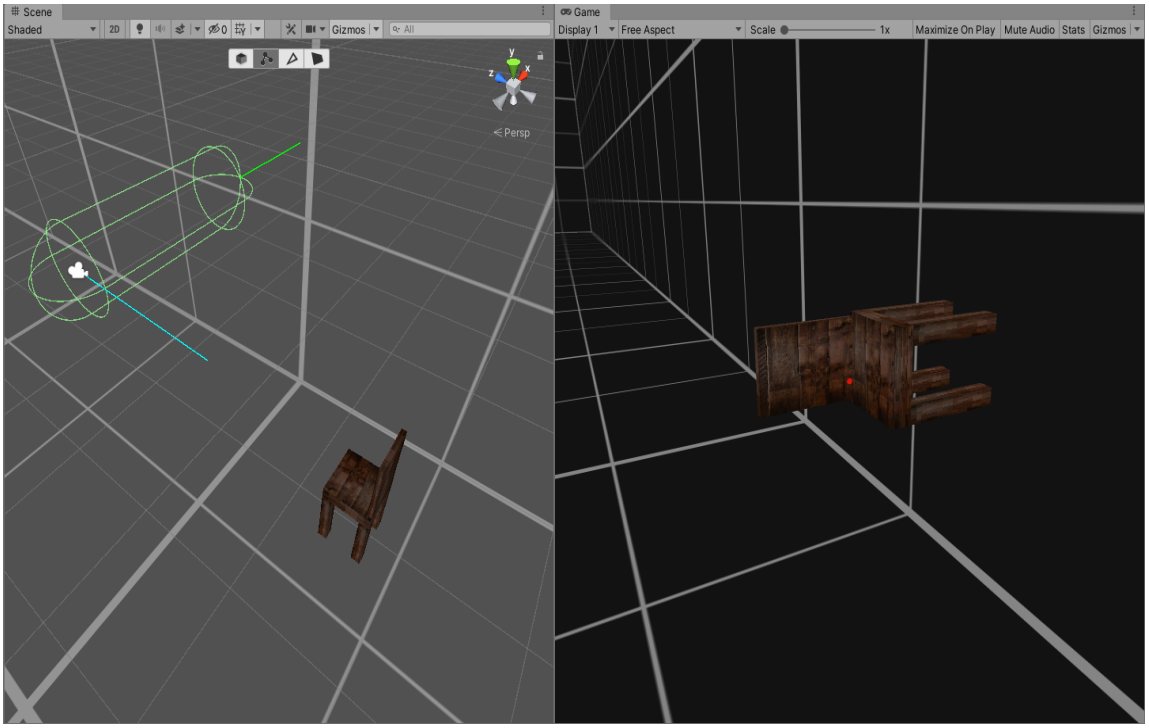
Στον controller, Το διάνυσμα θέσης δηλαδή η κατεύθυνσή του, και το διάνυσμα περιστροφής, δηλαδή ο προσανατολισμός του, είναι δεδομένα που ελέγχονται αποκλειστικά από το Scr_Physics. Το Scr_Physics διαχειρίζεται το διάνυσμα περιστροφής έμμεσα με την χρήση τρισδιάστατου διανύσματος Euler αντί για το Default τετραδιάστατο Quaternion της Unity. Το γεγονός αυτό επιτρέπει μία πιο εύκολη κατανόηση του προσανατολισμού του διανύσματος από τον προγραμματιστή και την αποφυγή του τριγωνομετρικού προβλήματος Gimbal Lock. Το διάνυσμα περιστροφής εξαρτάται από δύο πεδία που δεν αλληλοεπιδρούν μεταξύ τους, αλλά εξυπηρετούν διαφορετικές λειτουργίες. Το πρώτο πεδίο ονομάζεται Eul_Rot και αποτελεί την νοητή κατεύθυνση προσανατολισμού του controller. Νοητή χαρακτηρίζεται διότη δεν επηρεάζει πραγματικά το transform.rotation αλλά λειτουργεί ακριβώς με τον ίδιο τρόπο. Αυτό γίνεται για δύο λόγους :

- Ο πρώτος είναι πως, καθώς ο controller είναι συμμετρικός ως προς τον Y, δεν χρειάζεται να περιστρέφεται πραγματικά αλλά μονάχα να είναι γνωστό το που “βλέπει” ώστε να εκτελούνται οι σχετικές λειτουργίες αντίστοιχα με αυτό, όπως κίνηση και ο προσανατολισμός της κάμερας όταν βρίσκεται σε προοπτική πρώτου και τρίτου προσώπου.
- Ο δεύτερος είναι, λιγότερο απαιτητικοί υπολογισμοί των physics καθώς οι μεταβολές στα physics ως προς την περιστροφή είναι πιο απαιτητικοί από τις μεταβολές της θέσης

Το δεύτερο πεδίο ονομάζεται frGravity_Direction και δείχνει την επιθυμητή κατεύθυνση της βαρύτητας. Διάνυσμα αυτό, εκτός από το ότι εξυπηρετεί τον επονομαζόμενο σκοπό του, καθορίζει επίσης και το προσανατολισμό του controller στον χώρο. Ο υπολογισμός αυτός γίνεται μέσω της μεθόδου Orientation_Ctrl(), η οποία ορίζει το διάνυσμα UP του Transform του controller, ως το αντίστροφο του διανύσματος frGravity_Direction, περιστρέφοντας έτσι τον Controller, ώστε η κατεύθυνση του προς τα επάνω να είναι αντίρροπη με την κατεύθυνση της βαρύτητάς του. Το μήκος του διανύσματος δεν έχει επίδραση.



Εικόνα 6.5 – Το διάσημα GravityDir (πράσινο) και το διάνισμα LookRot (κυανό) με default τιμή στο GravityDir (0 , -1 , 0)



Εικόνα 6.6 – Το διάσημα GravityDir (πράσινο) και το διάνισμα LookRot (κυανό) με καινούρια τιμή στο GravityDir (-1, 0 , 0)

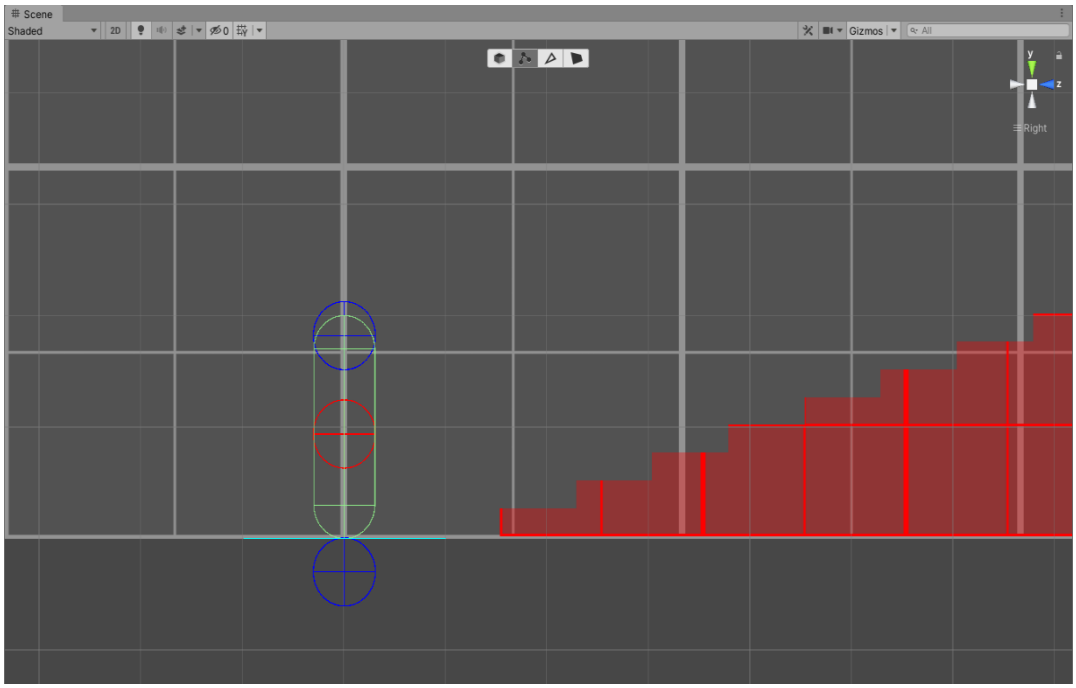
6.8 Check Casting

check casting ονομάζεται το σύνολο λειτουργιών που χρησιμοποιούν εντολές της βιβλιοθήκης Physics.XCast. Οι εντολές αυτές παρέχουν δυνατότητες χωρικού ελέγχου του τρισδιάστατου χώρου. Η λογική τους βασίζεται, στην δημιουργία ενός σώματος το οποίο κινείται προς μία συγκεκριμένη κατεύθυνση και απόσταση, και προσπαθεί να συγκρουστεί με colliders που θα βρει στο πέρασμά του, επιστρέφοντας το πρώτο που θα βρει. Ανάλογα με το είδος του casting, χρησιμοποιείται διαφορετικό σχήμα με τον περιορισμό ότι πρέπει να είναι ένα από τα primitive σχήματα της Unity. Ένα επιπλέων χαρακτηριστικό είναι πως δίνεται η δυνατότητα να διανύσει όλη την απόσταση και να επιστρέψει όλα τα Colliders στο πέρασμά του σε πίνακα, με τον περιορισμό ότι δεν θα είναι με την σειρά που τα συνάντησε στον πίνακα επιστροφής. Όλες οι Physics.XCast, ανεξάρτητα από το που θα κληθούν, ξεκινούν και τελειώνουν εντός ενός FixedUpdate loop.

Στο Module της εργασίας γίνεται κατά κύριο λόγο η χρήση του physics.raycast που τραβάει μία ευθεία γραμμή προς μία κατεύθυνση και το physics.spherecast που εκπέμπει μία σφαίρα προς μία κατεύθυνση. Η πιο βασική λειτουργία που επιτελείται με την εντολή physics.spherecast, είναι το Ground Checking, δηλαδή ο έλεγχος της ύπαρξης ή μη εδάφους κάτω από τον χαρακτήρα. Η διαδικασία ελέγχου γίνεται :

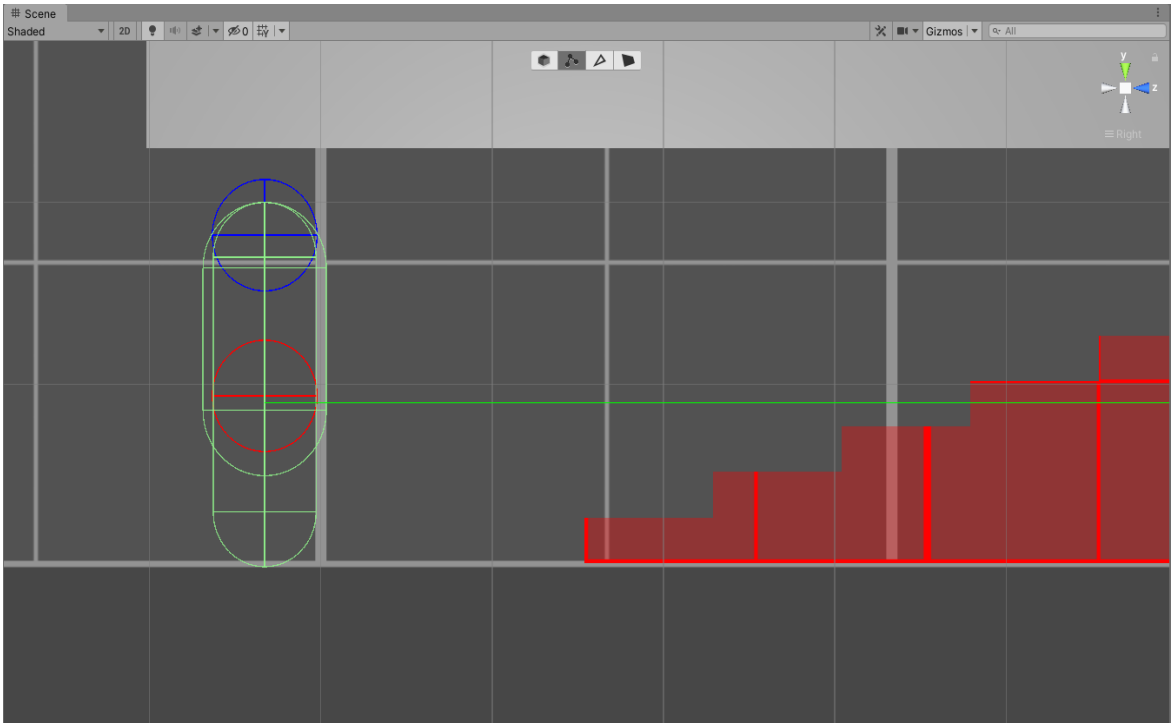
- δίνοντας την θέση που ξεκινάει η σφαίρα για να κινηθεί, ως το κέντρο του World Collider προσαυξημένο ως προς τον Y κατά ένα ποσοστό,
- την περιφέρειά της, ως την περιφέρεια του World Collider μειωμένη κατά ένα ποσοστό,
- την κατεύθυνση της, ως την κατεύθυνση της βαρύτητας του Controller,
- την απόσταση που θα διανύσει, ως το επιθυμητό βήμα που μπορεί να κατέβει πριν ενεργοποιηθεί η λογική της βαρύτητας
- και τέλος την μάσκα που μπορεί να κάνει επαφή με άλλα Colliders

Εάν σε οποιαδήποτε περίπτωση δεν επιστρέψει κάτι συνάρτηση, δηλαδή το out RaycastHit info είναι null, τότε ενεργοποιείται αυτόματα η λογική της βαρύτητας. Ειδάλλως, έπειτα από έλεγχο εγκυρότητας του αντικειμένου εδάφους, η μεταβλητή που προσδιορίζει την κατάσταση εδαφική επαφής του controller, Is_Grounder, αλλάζει σε true.

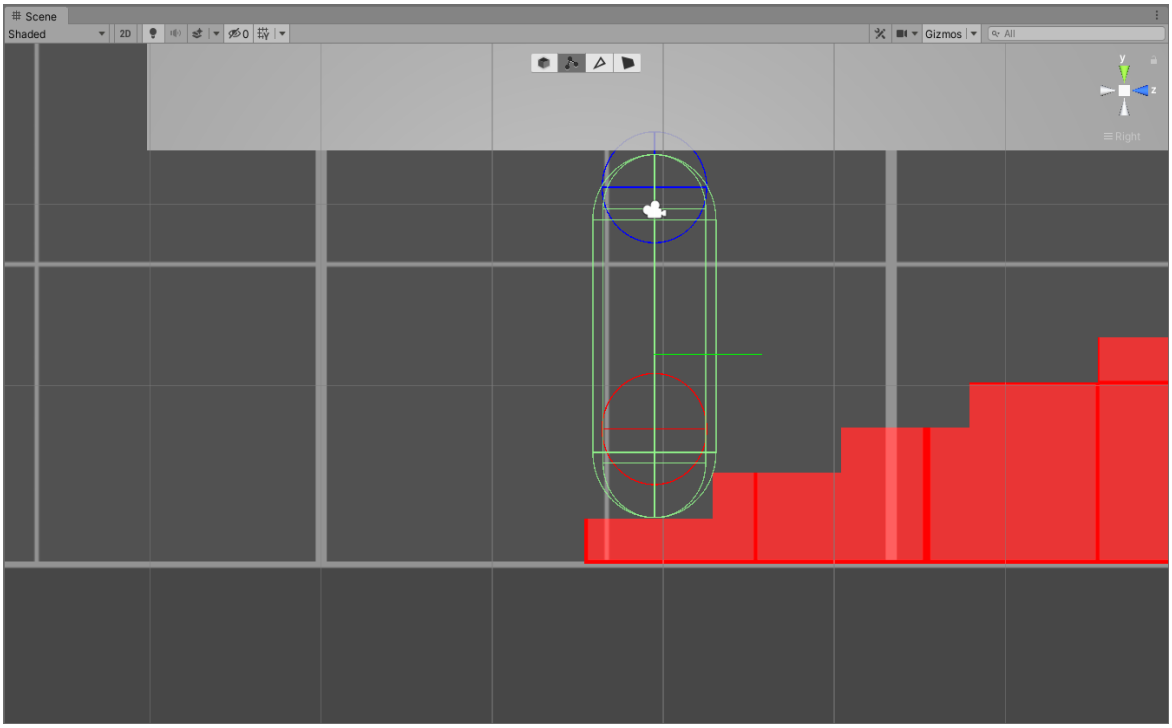


Εικόνα 6.7 – Η θέση έναρξης του casting (κόκκινο) και οι τελικές θέσεις που θα εκλέξει στο πέρασμα του (μπλε)

Με την ίδια λογική λειτουργεί το Top Checking το οποίο απαιτήται λόγο της ιδιαιτερότητας να αιωρείται ο Controller ώστε να επιτυγχάνεται σωστά το Step Offset. Αν ανιχνευτεί σώμα σε απόσταση ίση με την απόσταση που μπορεί να διανύσει το Ground Check, δηλαδή την απόσταση βήματος προς τα κάτω, τότε αυτομάτως το ύψος του WorldCollider γίνεται ίσο με το ύψος το CharaterCollider, αποτρέποντας στην πορεία μία ασυνήθιστη διασταύρωση μεταξύ controller και αντικειμένου.



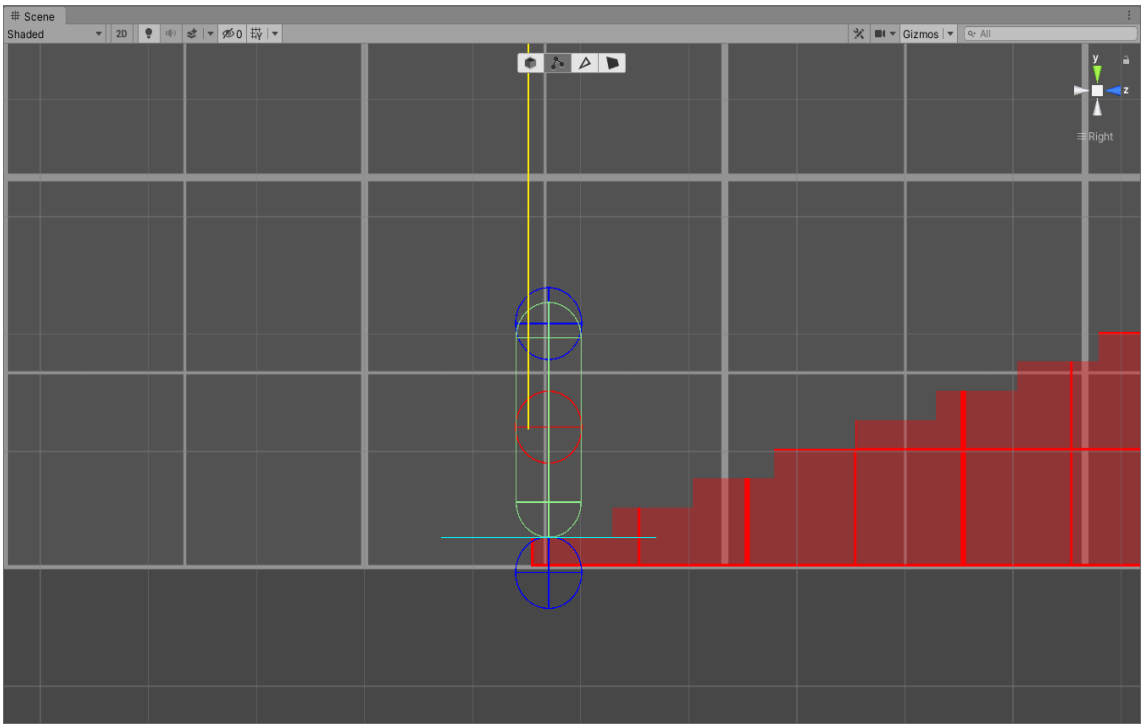
Εικόνα 6.8 – Το ύψος του world collider πριν το top check επιστρέψει true



Εικόνα 6.9 – Το ύψος του world collider μετά το top check επιστρέψει true

6.9 Step Offset Logic

Η λειτουργία Step Offset αναφέρεται στην τεχνητή αλλαγή της θέσης ύψους του controller, δηλαδή την αλλαγή της τιμής του άξονα Y, σε σχετικές συντεταγμένες με το ίδιο. Η λειτουργία που επιτελεί, είναι η ομαλή μετάβαση σε ψηλότερο ή χαμηλότερο επίπεδο χωρίς να επιδράτε το συνολικό velocity του συστήματος κίνησης του contoller και συγκεκριμένα το move velocity. Η λογική του Step Offset είναι να αλλάζει με ακαριαία δύναμη την θέση του controller μέσω του διανύσματος inertia_velocity, προς τα πάνω όταν ανιχνευτεί διαφορά ύψους μικρότερη της διαφοράς ύψους του Character Collider και του World Collider, αποτρέποντας έτσι μία πιθανή σύγκρουση με το έδαφος και αλλαγή της ταχύτητας κίνησης. Με τον ίδιο τρόπο, και μόνο όταν το Is_Grounded είναι ήδη true, λειτουργεί προς τα κάτω όταν ανιχνεύεται διαφορά μεγαλύτερη της διαφοράς ύψους του Character Collider και του World Collider, αποφεύγοντας την άσκοπη εκτέλεση του διανύσματος βαρύτητας και διατηρώντας την κατάσταση Is_Grounder σε true, διότι έχει επαληθευτεί η ύπαρξη εδάφους εκ τον προτέρων από το Ground Check. Το Step Offset τρέχει μέσα στον Check Casting κώδικα αμέσως μετά την επαλήθευση του Is_Grounded.



Εικόνα 6.10 – Το διάνυσμα inertiaVel «εκτινάσσει» για ένα physics frame τον controller, φέρνοντας τον στο ύψος του επιπέδου που υπολογιστικέ από το StepOffset

6.10 Action Events και Action Listeners (Internal Communication Logic)

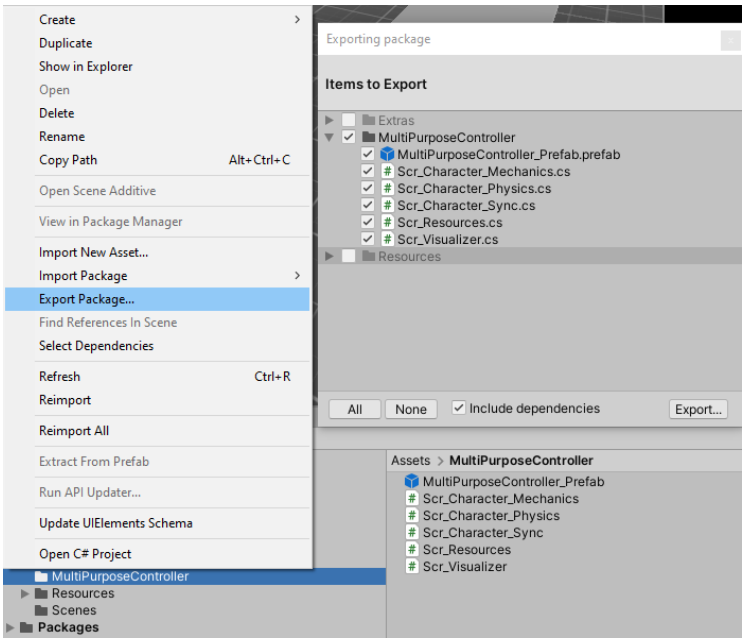
Όπως αναφέρθηκε σε προηγούμενη ενότητα, προκειμένου να επικοινωνήσουν τα scripts Scr_Physics και Scr_Mechanics μεταξύ τους χωρίς να έχουν αναφορά το ένα στο άλλο, απαιτείται η υλοποίηση δημοσίων event handlers και event listeners τα οποία, το Scr_Sync κατά την έναρξη της λειτουργίας του θα αντιστοιχίσει. Τα events αυτά υλοποιούνται με την χρήση δύο προ κατασκευασμένων delegate της C#, Το System.Action και το System.Func. Και τα δύο delegates λειτουργούν με τον ίδιο τρόπο, με την μόνη διαφορά, ότι το System.Func επιστρέφει τιμή από τις περιεχόμενες συναρτήσεις. Συνεπώς, το System.Action χρησιμοποιείται για να πυροδοτήσει ή να στείλει τιμές σε μεθόδους άλλων Scripts, ενώ το System.Func χρησιμοποιείται για να πάρει τιμές από μεθόδους άλλων Scripts. Προκειμένου να διατηρηθεί η αυτονομία των scripts και να αποφευχθούν τα errors, απαιτείται τα System.Func delegates που υλοποιούνται, να επιστρέφουν μια Default τιμή σε περίπτωση που είναι null. Ένα παράδειγμα της χρήσης των System.Action, για χρήση μεθόδων του Scr_Physics από το Scr_Mechanics, είναι το Delegate ActionDelegate_On_Found_Ground_Gobj που παρέχεται από το Scr_Physics και ενεργοποιείται κατά την επαφή του controller με το έδαφος. Το εν λόγω Delegate δέχεται ως όρισμα μία παράμετρο αναφοράς σε gameobject αντικείμενο, που στην συγκεκριμένη περίπτωση είναι το gameobject που ανιχνεύεται από το Ground Check. Προκειμένου να αξιοποιηθεί το delegate από το Scr_Mechanics, η μέθοδος Set_Ground_Gobj προστίθεται στις αναφορές μεθόδων του ActionDelegate_On_Found_Ground_Gobj, από το Scr_Sync που έχει αναφορά και στο delegate και στην μέθοδο. Κατά την πυροδότηση του γεγονότος, η παράμετρος gameobject με την οποία θα κληθεί, θα περαστεί και στο Set_Ground_Gobj. Άλλη μία περίπτωση χρήσης του System.Action σε συνδυασμό με το System.Func είναι το Delegate ActionDelegate_On_Change_Posture, στο οποίο, αντίστροφα τώρα, "ακούει" μία μέθοδος του Scr_Physics που αλλάζει το ύψος του controller. Εν μέσω εκτέλεσης της μεθόδου που πυροδοτεί το Action Delegate, πυροδοτείται ένα Func Delegate με τύπο επιστροφής boolean το οποίο επιχειρεί, μέσω των συνδεδεμένων σε αυτό μεθόδων, να ελέγξει εάν δύνανται η αλλαγή ύψους του controller. Εάν μια σχετική μέθοδος ελέγχου του Scr_Physics έχει συνδεθεί με το delegate, θα επιστρέψει τιμή από αυτήν, ειδάλλως, εάν το delegate είναι null θα επιστρέφει πάντα true..

6.11 Interfaces του Scr_Sync (External Communication Logic)

Τα Scr_Sync υλοποιεί πληθώρα interfaces τα οποία χρησιμοποιούνται για επικοινωνία του module με άλλες οντότητες. Τα ποιο σημαντικά interfaces του Scr_Sync, είναι το ICtrl_Able και το ICamera_Able. Το ICtrl_Able προσδίδει την δυνατότητα απευθείας ελέγχου στο module και οι μέθοδοί του αναλαμβάνουν να υλοποιήσουν την λογική του βαθμού ελέγχου που επιτρέπεται από ένα script, με αναφορά στο εν λόγω interface. Βέλτιστο παράδειγμα στην συγκεκριμένη περίπτωση αποτελεί ένα input manager, το οποίο έχει αναφορά στο ICtrl_Able του module, και μέσω inputs επιθυμεί να μετακινήσει τον controller. Το interface μέσω της μεθόδου Receive_MoveDir, θα λάβει ένα μοναδιαίο διάνυσμα και θα το στείλει στο Scr_Physics να επεξεργαστεί το "αίτημα" μετακίνησης. Με τον ίδιο τρόπο αλλάζει η περιστροφή του controller μέσω Receive_RotDir και άλλες ενέργειες από την μέθοδο Receive_Action_Input παραδείγματος χάρη να κάνει άλμα. Η χρήση του ICtrl_Able δεν προορίζεται μόνο για χρήση από input handlers, αλλά δύνανται να χρησιμοποιηθεί και από ένα AI Script. Το δεύτερο σημαντικό interface είναι το ICamera_Able και υλοποιεί τον κώδικα που ελέγχει την κάμερα. Με τον τρόπο αυτό μία κάμερα μπορεί να συνδεθεί και να αποσυνδεθεί στο module μόνο με την χρήση του interface. Το ICamera_Able δουλεύει διαφορετικά από το ICtrl_Able με την λογική πως κάθε παράμετρος των μεθόδων του είναι μία αναφορά μεταβλητής και ουσιαστικά επιστρέφει δεδομένα στο καλών αντικείμενο.

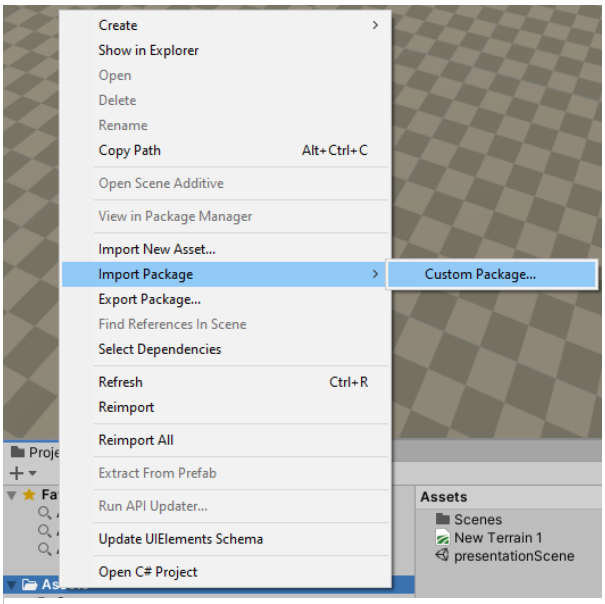
6.12 Εξαγωγή Module ως Unity Asset Package και εγκατάσταση σε νέο Project

Το module δύνανται να γίνει export και import σε νέο project της Unity με την χρήση του asset packaging της Unity, το οποίο επιτρέπει την εξαγωγή resources ενός project όπως textures, scripts και prefabs μαζί με τις ρυθμίσεις τους, σε μορφή .unitypackage.



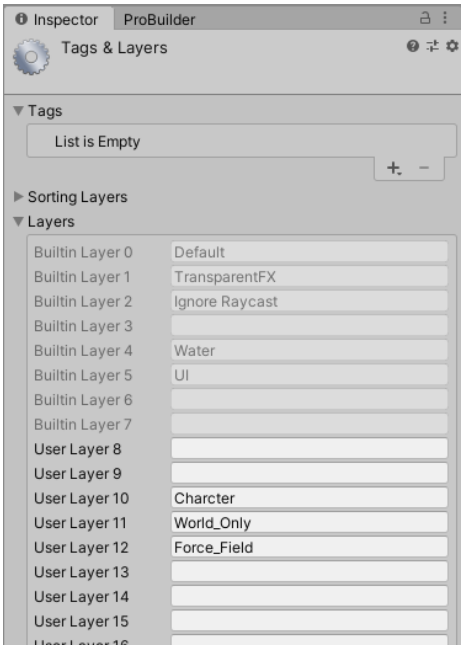
Εικόνα 6.11 – Εξαγωγή του module από φάκελο ως Asset Package

Το αρχείο αυτό έπειτα, μπορεί εγκατασταθεί σε οποιοδήποτε project με την προϋπόθεση ότι η version της Unity στην οποία εγκαθιστάτε, αναγνωρίζει τις εκδώσεις των assets και τους ορισμούς των scripts.



Εικόνα 6.12 – Εισαγωγή του module σε νέο project της Unity από Asset Package

Στην περίπτωση του module της εργασίας είναι ανάγκη να γίνουν μερικές έξτρα τροποποιήσεις μετά την εγκατάσταση, ώστε το module να λειτουργήσει σωστά. Οι τροποποιήσεις αυτές αφορούν τις μάσκες οι οποίες δεν μεταφέρονται με το module. Στο νέο project, αφού γίνει import του module, πατώντας δεξί κλικ πάνω δεξιά στο κουμπί Layers, θα εμφανιστεί μία λίστα με όλα τα Layers του project. Στις θέσεις 10, 11 και 12 πρέπει να προστεθούν τα layers : Character, World_Only και Force_Field με την αντίστοιχη σειρά.



Εικόνα 6.13 – Η απαιτούμενες μάσκες του module 10 , 11 και 12

Έπειτα, πρέπει να οριστούν οι σχέσεις των τριών αυτών layer μεταξύ τους και με τα υπόλοιπα layers, τροποποιώντας τον πίνακα collision_matrix στο παράθυρο project settings, σύμφωνα με τις παρακάτω πίνακα.

Εικόνα 6.14 – Το collision matrix του module

Πηγές

A Brief History of Graphics - YouTube

A Brief History of Physics in Video Games - Roblox Blog

Why Arent Real-World Physics Equations Used In Video Games? - YouTube

How Do Developers Implement Physics In Video Games? (Part 1) - YouTube

How Do Developers Implement Physics In Video Games? (Part 2) - YouTube

How To Write Large Programs. Techniques for programming in the… | by Oleg Alexander | Medium

Delegates - C# Programming Guide | Microsoft Docs

Unity - Manual: Order of Execution for Event Functions

What is Event-driven Programming?

Event Driven Programming in C#. Changing from a procedure-driven to… | by Jordan Lee | Level Up Coding

Java Web Services Architecture | ScienceDirect

Modifiability - an overview | ScienceDirect Topics

«Δηλώνω ρητά ότι, σύμφωνα με το άρθρο 8 του Ν.1599/1986, η παρούσα εργασία αποτελεί αποκλειστικά προϊόν προσωπικής μου εργασίας, δεν προσβάλλει κάθε μορφής δικαιώματα διανοητικής ιδιοκτησίας, προσωπικότητας και προσωπικών δεδομένων τρίτων, δεν περιέχει έργα/εισφορές τρίτων για τα οποία απαιτείται άδεια των δημιουργών/δικαιούχων και δεν είναι προϊόν μερικής ή ολικής αντιγραφής, οι πηγές δε που χρησιμοποιήθηκαν περιορίζονται στις βιβλιογραφικές αναφορές και μόνον και πληρούν τους κανόνες της επιστημονικής παράθεσης».

Έξαρχος Πέτρος