Τεχνολογικό Εκπαιδευτικό Ίδρυμα Κρήτης

Σχολή Τεχνολογικών Εφαρμογών
Τμήμα Μηχανικών Πληροφορικής

Πτυχιακή Εργασία

Τίτλος : Ανίχνευση κίνησης με αισθητήρες που προσαρμόζονται στο σώμα

Μπατάκης Νικόλαος Τουόμο (3600)

Επιβλέπων καθηγητής : Μαλάμος Αθανάσιος

Ηράκλειο
2016

Description

This thesis titled “Motion Detecting with Wearable Sensors” has as purpose of showing the capabilities of wearable sensors. The project is a 2d platformer game made in the Unity game engine. After some thinking I decided to go with an imitation of the classical Super Mario Bros for the sole reason of being able to use many of the Myo armbands capabilities. In the game, there will be the possibility of choosing between different accounts / players, save progress of level completion and be on the high scores table with the use of your personal account. The wearable sensor used is the Myo armband by Thalmic Labs, Myo armband is a gesture recognition device worn on the forearm, which allows the user to control technology wirelessly using hand motions. Myo uses Electromyographic sensors to read muscle activity. With the use of gestures, accelerometer and its rotation the user will be able to play the game completely with the use of the Myo armband.

Περιγραφή

Η εργασία αυτή με τίτλο « Ανίχνευση κίνησης με αισθητήρες που προσαρμόζονται στο σώμα » έχει σαν στόχο την επίδειξη των δυνατοτήτων των αισθητήρων σώματος. Στην εργασία αυτή θα δημιουργήσω ένα 2D platformer παιχνίδι με την χρήση της μηχανής κατασκευής παιχνιδιών Unity. Θα εξηγήσουμε το Unity και την χρήση του. Θα μιλήσουμε για το Myo, τα δεδομένα που προσφέρει, τους τρόπους που χρησιμοποιείται και γιατί διαφέρει από τους άλλους αισθητήρες κίνησης. Έπειτα θα γίνει μια σύντομη παρουσίαση των σκηνών του παιχνιδιού για να καταλάβει ο αναγνώστης πως λειτουργεί. Στο τέλος γίνεται η ανάλυση και επεξήγηση όλων των σκηνών και των πιο σημαντικών κομματιών κώδικα που έχουμε δημιουργήσει για την ευκολότερη κατανόηση της δημιουργίας του παιχνιδιού όπως επίσης και η επεξήγηση του κώδικα που παρέχει το SDK του Myo.

Περιεχόμενα

Πίνακας Εικόνων

Πίνακας Κωδίκων

1 Εισαγωγή

Η πτυχιακή είναι μια ευκαιρία για να ασχοληθεί ο φοιτητής με πράγματα που τον ενδιαφέρουν και ίσως του επιφέρουν βοήθεια στην εύρεση εργασίας. Η πτυχιακή αυτή υλοποιήθηκε για την κατανόηση των αισθητήρων κίνησης και των τρόπο χρήσης τους.

1.1 Περίληψη

Η εργασία αυτή με τίτλο «Ανίχνευση κίνησης με αισθητήρες που προσαρμόζονται στο σώμα » έχει σαν στόχο την επίδειξη των δυνατοτήτων των αισθητήρων σώματος. Στην εργασία αυτή θα δημιουργήσω ένα 2D platformer παιχνίδι με την χρήση της μηχανής κατασκευής παιχνιδιών Unity. Η επιλογή της δημιουργίας ενός δισδιάστατου παιχνιδιού με βάση το Super Mario Bros έγινε έπειτα από την εξακρίβωση των δυνατοτήτων του Myo armband και κατά πόσο θα ήταν δυνατόν να ανταπεξέλθει σε αυτά που ζητούνται. Στο παιχνίδι αυτό υπάρχει η δυνατότητα επιλογής μεταξύ χρηστών, αποθήκευσης του επιπέδου που έχει φτάσει ο χρήστης και η παρουσίαση των μέγιστων βαθμολογιών. Ο αισθητήρας σώματος που θα χρησιμοποιηθεί είναι το περιβραχιόνιο Myo της Thalmic Labs, το Myo είναι μια συσκευή αναγνώρισης χειρονομιών το οποίο φοριέται στο πήχη (ή αντιβράχιο) και επιτρέπει στο χρήστη να ελέγχει διάφορες συσκευές ή εφαρμογές ασύρματα χρησιμοποιώντας τις κινήσεις του χεριού. Το Myo χρησιμοποιεί ηλεκτρομυογραφικούς αισθητήρες για να διαβάσει την δραστηριότητα των μυών. Με την χρήση χειρονομιών, του επιταχυνσιόμετρου και την περιστροφή του Myo, ο χρήστης θα είναι σε θέση να παίξει το παιχνίδι μόνο με το Myo το οποίο ήταν ένας από τους στόχους.

1.2 Κίνητρο για τη Διεξαγωγή της Εργασίας

Στις αρχές του 2013 είδα την καμπάνια της εταιρίας Thalmic Labs για το Myo στο www.kickstarter.com το οποίο είναι μια παγκόσμια πλατφόρμα χρηματοδότησης από το κοινό. Αμέσως αποφάσισα ότι θα με ενδιέφερε να εξερευνήσω αυτόν τον τομέα. Έπειτα από 1 χρόνο χρήσης του Myo αποφάσισα ότι θα ήθελα να δημιουργήσω μια εφαρμογή η οποίο θα χρησιμοποιεί το Myo. Έπειτα από συζητήσεις με τον κ. Μαλάμο καταλήξαμε στην δημιουργία ενός παιχνιδιού για την ερεύνα άλλου ενός τομέα που ήθελα να ερευνήσω.

1.3 Σκοπός και στόχοι της Εργασίας

Σκοπός ήταν να συμπεριληφθεί το Myo στην δημιουργία ενός παιχνιδιού και ταυτόχρονα η σχεδίαση ενός ολοκληρωμένου παιχνιδιού το οποίο θα χρησιμοποιεί το Myo αντί για τα περιφερειακά του ηλεκτρονικού υπολογιστή.

1.4 Δομή της Εργασίας

Τα κεφάλαια 2 Unity και 3 Thalmic Myo θα βοηθήσουν τον αναγνώστη να καταλάβει τα αντικείμενα που χρησιμοποιήσαμε για την δημιουργία της εφαρμογής αυτής. Στο 4^{ο} κεφάλαιο γίνεται μια επεξήγηση στην χρησιμότητα κάθε σκηνής και τον τρόπο που αντιδράει στις εντολές του χρήστη. Το 5^{ο} κεφάλαιο αποτελεί το βασικό κομμάτι αυτής της εργασίας το οποίο δείχνει αναλυτικά ένα μεγάλο μέρος του κώδικα που χρησιμοποιείται και εξηγεί πως και γιατί γίνεται η χρήση αυτών.

2 Το Πρόγραμμα Unity

Το Unity είναι μια μηχανή κατασκευής cross-platform παιχνιδιών που αναπτύχθηκε απο την εταιρία Unity Technologies και χρησιμοποιείται για την δημιουργία παιχνιδιών για υπολογιστές, κονσόλες, κινητά και ιστοσελίδες. Το Unity έχει ενσωματωμένο IDE. Το IDE σημαίνει Integrated Development Environment που μεταφράζεται ως Ολοκληρωμένο Περιβάλλον Ανάπτυξης : Το Unity είναι η ένωση της μηχανής παιχνιδιού που αφηνει τον χρήστη να τρέξει το παιχνίδι που έχει δημιουργήσει, μιας εφαρμογής που όλα τα «κομμάτια» του παιχνιδιού τοποθετούνται μαζί για την δημιουργία του παιχνιδιού και του επεξεργαστή κώδικα που παρέχει το Unity , MonoDevelop το οποίο δεν θα χρησιμοποιήσω λόγω συνήθειας.

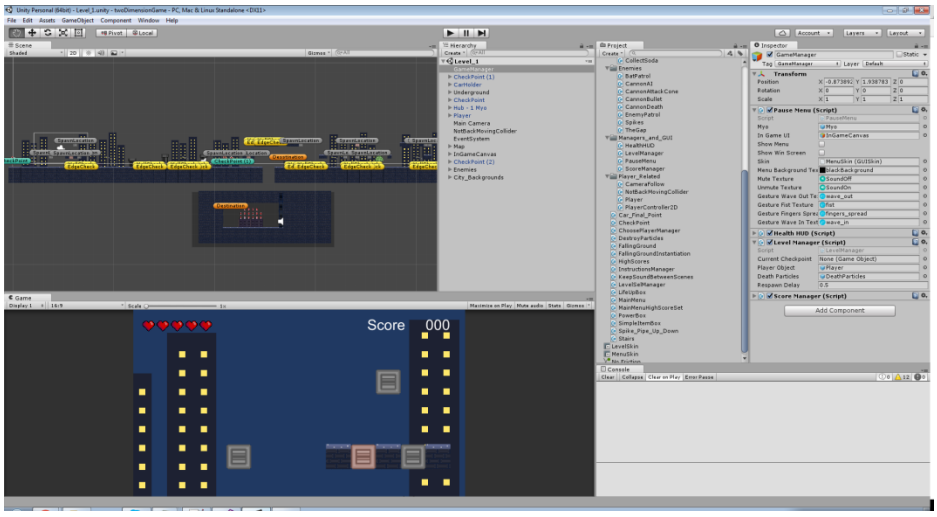
Εικόνα 1 : Λογότυπο Unity

2.1 Δομή του Unity

Θα γίνει μια σύντομη αναφορά στον Editor του Unity και στο MonoDevelop, τα οποία αποτελούν βασικά στοιχεία της συγκεκριμένης μηχανής παιχνιδιών. Επίσης θα αναφέρουμε το Mecanim.

2.1.1 Ο Unity Editor

Ο Unity Editor αποτελεί το βασικό μέρος του Unity, αφού με τη χρήση αυτού δημιουργούνται τα παιχνίδια. Με την εισαγωγή των assets του παιχνιδιού μπορούν να κατασκευαστούν διάφορες σκηνές όπως και θα αναλύσουμε στο 5^{ο} κεφάλαιο. Συνδυάζοντας την σκηνή με τον κώδικα που θα δημιουργήσουμε μας δίνει το τελικό παιχνίδι.



Εικόνα 2 : Unity Editor

2.1.2 Το MonoDevelop

Το MonoDevelop είναι ένα πρόγραμμα το οποίο εξυπηρετεί τον προγραμματιστή στην δημιουργία κώδικα για την σωστή εκτέλεση του παιχνιδιού. Το Unity μαζί με το MonoDevelop υποστηρίζουν τρεις γλώσσες προγραμματισμού για την δημιουργία παιχνιδιών.

- C# (C Sharp) : Αντικειμενοστραφής γλώσσα προγραμματισμού, την οποία και θα χρησιμοποιήσω για την δημιουργία αυτού του παιχνιδιού.
- JavaScript : Διερμηνευμένη γλώσσα προγραμματισμού επηρεασμένη από τη C.
- Boo : Επηρεασμένη από την Python, η Boo είναι μια στατική γενικού σκοπού γλώσσα προγραμματισμού.

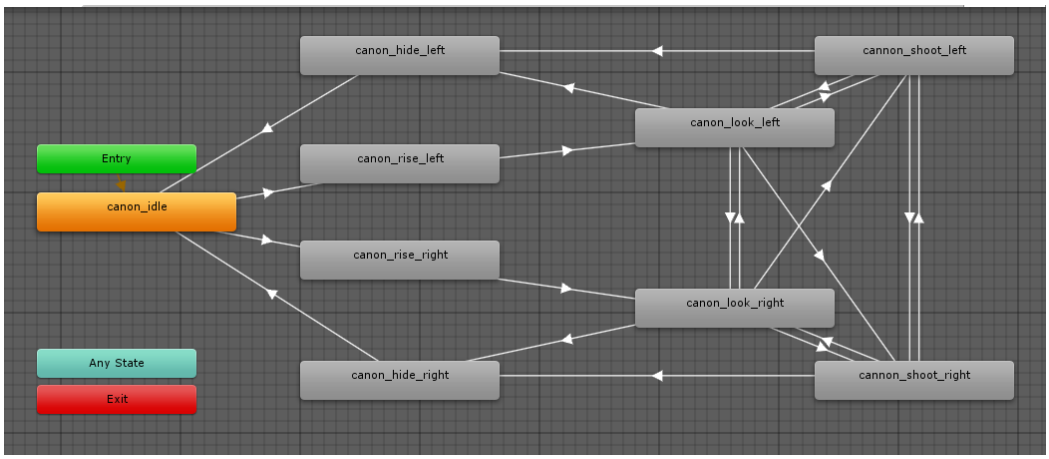
Το MonoDevelop έχει παρόμοια χαρακτηριστικά με το Visual Studio το οποίο θα χρησιμοποιήσω για λόγους συνήθειας.

Εικόνα 3 : Λογότυπο MonoDevelop

2.1.3 Το Mecanim

Το Unity έχει ένα εξελιγμένο σύστημα animation το οποίο παρέχει :

- Εύκολη ρύθμιση των animations σε ανθρωποειδές χαρακτήρες και όχι μόνο.
- Την ικανότητα να εφαρμόσει animations από ένα αντικείμενο σε ένα άλλο.
- Διαχείριση πολύπλοκων αλληλεπιδράσεων μεταξύ animations με την χρήση ενός οπτικού εργαλείου το οποίο και θα χρησιμοποιήσω.
- Η δημιουργία κίνησης μέσω animation σε διάφορα μέλη του σώματος με χρήση διαφορετικής λογικής για το καθένα.



Εικόνα 4 : Το Mecanim

3 Το Περιβραχιόνιο Myo

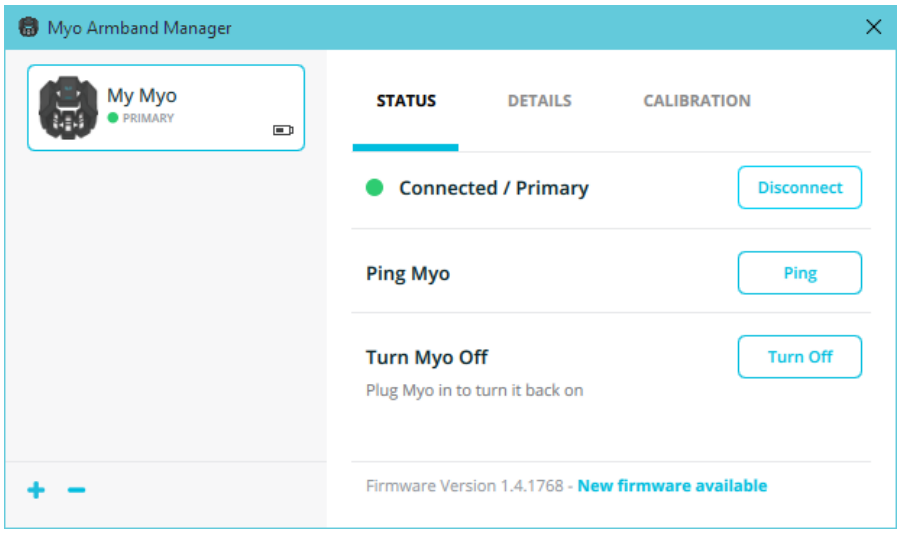
Στις αρχές του 2013 το Myo έκανε την εμφάνιση του στο κοινό μέσω της καμπάνιας της εταιρίας Thalmic Labs στο www.kickstarter.com. Το Myo είναι μια συσκευή αναγνώρισης χειρονομιών το οποίο φοριέται στο πήχη (ή αντιβράχιο) και επιτρέπει στο χρήστη να ελέγχει διάφορες συσκευές ή εφαρμογές ασύρματα χρησιμοποιώντας τις κινήσεις του χεριού.

.

Εικόνα 5 : Περιβραχιόνιο Myo

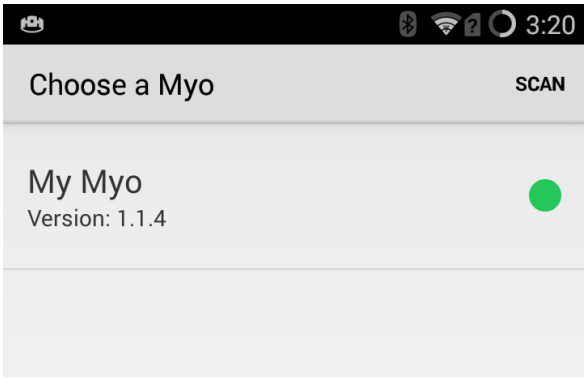
3.1 Ενεργοποίηση

Για την χρήση του Myo σε Windows ή Mac η διαδικασία που πρέπει να ακολουθήσει ο χρήστης παρέχεται στο software που χρειάζεται να κατεβάσει για την χρήση του με τον υπολογιστή. Μετά την εγκατάσταση του MyoConnect ο χρήστης συνδέει το Bluetooth dongle και το Myo στον υπολογιστή, ενημερώνει το λογισμικό του Myo και έπειτα είναι έτοιμο να φορεθεί. Μόλις ο χρήστης το τοποθετήσει στο χέρι του χρειάζεται να κάνει μια βαθμονόμηση (calibration) για την σωστή λειτουργία του.



Εικόνα 6 : Πρόγραμμα Myo Connect

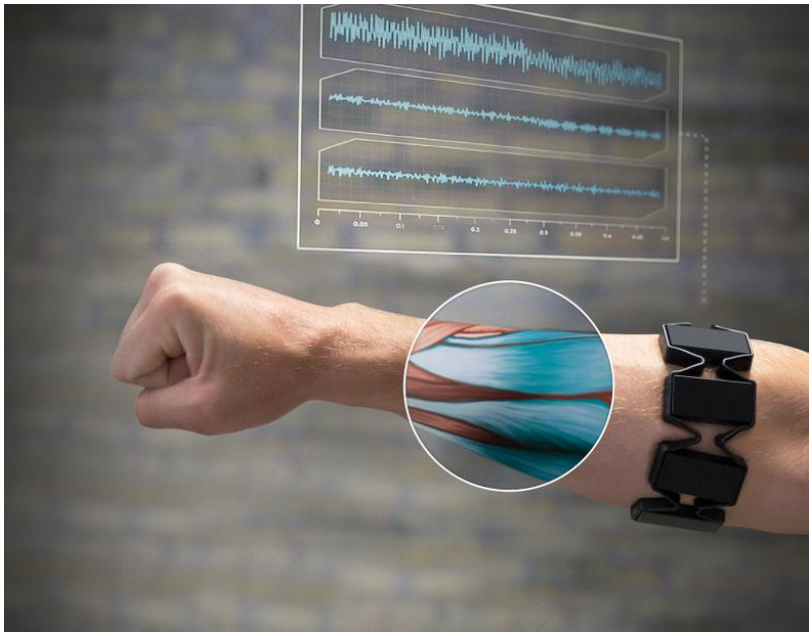
Για την χρήση του σε IOS ή Android η διαδικασία είναι πολύ πιο απλή εφόσον όλη η δουλειά γίνεται στην εφαρμογή με το οποίο θα δουλεύει το Myo. Άρα ο χρήστης χρειάζεται να τοποθετήσει το Myo στο χέρι του, να ενεργοποιήσει το Bluetooth στην κινητή συσκευή του και μέσω της εφαρμογής να συνδεθεί στο Myo.



Εικόνα 7 : Σύνδεση Myo σε Android

3.2 Τρόπος λειτουργίας

Το Myo αποτελείται από 8 blocks όπως φαίνεται στην **εικόνα 5**, κάθε ένα από αυτά περιέχει ηλεκτρομυογραφικούς αισθητήρες ιατρικού επιπέδου για να διαβάζει την δραστηριότητα των μυών. Επίσης χρησιμοποιεί three-axis gyroscope, three-axis accelerometer και three-axis magnetometer για να μπορεί να ανιχνεύει την κίνηση σε οποιαδήποτε κατεύθυνση. Η μυϊκή δραστηριότητα και η αναγνώριση χειρονομιών χειρίζεται από ένα ενσωματωμένο ARM Cortex M4 επεξεργαστή που επικοινωνεί με τις συσκευές μέσω Bluetooth. Για συσκευές που δεν διαθέτουν την λειτουργία Bluetooth από μόνες τους, παρέχεται ένα dongle Bluetooth που συνδέεται σε οποιαδήποτε θύρα USB.



Εικόνα 8 : Ηλεκτρομυογραφικοί Αισθητήρες

3.3 Διαφορές με άλλους αισθητήρες

Οι περισσότεροι αισθητήρες αναγνώρισης χειρονομιών ανιχνεύουν την κίνηση μέσω μιας κάμερας το οποίο μπορεί να δημιουργήσει πρόβλημα όταν έχουμε χαμηλό φωτισμό, η απόσταση δεν είναι σωστή ή υπάρχουν εμπόδια μεταξύ εμάς και της κάμερας. Παίρνοντας πληροφορία κατευθείαν από τους μυς του χεριού αντί μέσω μιας κάμερας το Myo ξεπερνάει όλα αυτά τα προβλήματα.

3.4 Τρόποι χρήσεις

Η Thalmic Labs με το που ξεκίνησαν τις παραδόσεις των Myo άνοιξαν και το Myo Market το οποίο είναι μια ιστοσελίδα με δωρεάν εφαρμογές που έχουν δημιουργήσει αυτοί και έπειτα με τον καιρό μπορούσαν και οι προγραμματιστές να μοιραστούν τις δίκες τους εφαρμογές. Το Myo στην αρχή μπορούσε να χρησιμοποιηθεί για παρουσιάσεις και multimedia. Με το πέρασμα του χρόνου μπορεί κάνεις να χρησιμοποιήσει το Myo για τον έλεγχο του ποντικιού, ρομπότ, drones και διάφορα παιχνίδια. Τον τελευταίο χρόνο η Thalmic Labs έχει κάνει μια σημαντική προσπάθεια να βάλει το Myo σε χρήση για πιο σημαντικά θέματα όπως ο έλεγχος πρόσθετων μελών σώματος, στην αποκωδικοποίηση της νοηματικής γλώσσας και η ιατρική απεικόνιση για χειρούργους.

Εικόνα 9 : Myo και Τεχνητά Μέλη Σώματος

3.5 Τύποι δεδομένων

Το Myo παρέχει 2 ειδών δεδομένα σε μια εφαρμογή.

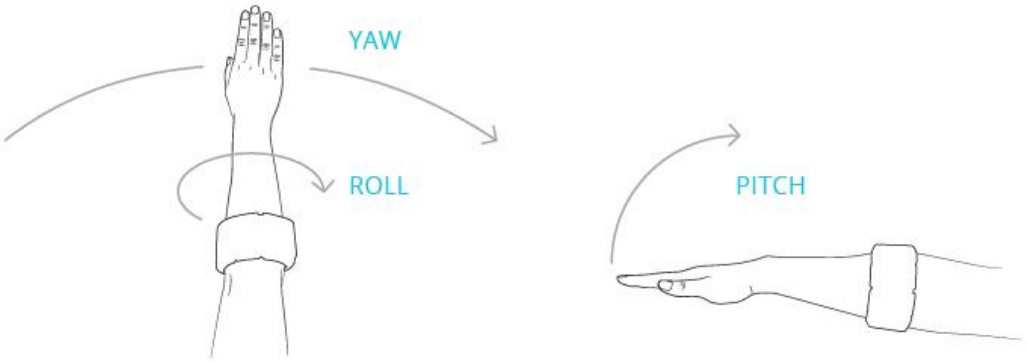
- Spatial Data (δεδομένα χώρου)
- Gestural Data (δεδομένα χειρονομιών)

Τα Spatial Data ενημερώνουν την εφαρμογή για τον προσανατολισμό και την κίνηση του χεριού. Αυτά τα δεδομένα προέρχονται από το board 9-αξόνων αδρανειακής μονάδας μέτρησης (inertial measurement unit IMU). Το SDK του Myo παρέχει 3 είδη spatial data :

1. Δεδομένα προσανατολισμού (orientation data), τα οποία δείχνουν με ποιο τρόπο το Myo είναι περιστραμένο.
2. Διανυσματικά δεδομένα επιτάχυνσης (Acceleration vector data) τα οποία αντιπροσωπεύουν την επιτάχυνση που δέχεται το Myo οποιαδήποτε χρονική στιγμή.
3. Δεδομένα γωνιακής ταχύτητας (Angular Velocity data) ,τα οποία παρέχονται από το γυροσκόπιο.

Τα Gestural Data δείχνουν στην εφαρμογή τι κάνει ο χρήστης με τα χέρια του. Το SDK του Myo παρέχει δεδομένα χειρονομιών με την μορφή μιας από τις προκαθορισμένες χειρονομίες, οι οποίες αντιπροσωπεύουν μια ιδιαίτερη στάση του χεριού του χρήστη. Τα δεδομένα αυτά παρέχονται τα τους ηλεκτρομυογραφικούς αισθητήρες.

Το Myo SDK υποστηρίζει Windows, Mac, IOS, Android αλλά υπάρχουν αρκετά εργαλεία φτιαγμένα από την κοινότητα του Myo Developers Forum. Επίσης το Myo SDK περιέχει ένα Unity package για τις εκδόσεις των Windows και Mac.



Εικόνα 10 : Roll, Pitch, Yaw

Το Myo αυτή την στιγμή παρέχει 5 προκαθορισμένες χειρονομίες, οι προγραμματιστές έχουν την δυνατότητα να συνδυάσουν τις 5 αυτές χειρονομίες με τα δεδομένα που παίρνουν από το board 9-αξόνων αδρανειακής μονάδας μέτρησης. Οπότε θα μπορούσαμε να βάλουμε το πρόγραμμα μας να αντιδρά αλλιώς για την απλή χειρονομία της γροθιάς και αλλιώς αν ο χρήστης έκανε γροθιά και ταυτόχρονα είχε γυρίσει το χέρι του τουλάχιστον 90 μοίρες προς δεξιά. Άρα έχουμε την δυνατότητα να χρησιμοποιήσουμε πολλές παραπάνω από τις 5 προκαθορισμένες χειρονομίες.

Εικόνα 11 : Χειρονομίες του Myo

3.6 Συμπεράσματα

Το Myo κατάφερε να ξεπεράσει τις προσδοκίες μου όταν το πρωτοπήρα στα χέρια μου, είναι εύκολο στην χρήση και υπάρχει μεγάλη γκάμα εφαρμογών που συνεχώς μεγαλώνει. Το design του είναι ελαστικό με αποτέλεσμα να μπορεί να φορεθεί από άτομα με οποιοδήποτε μεγέθους αντιβραχίου. Ανταποκρίνεται γρήγορα και είναι συμβατό με αρκετά λογισμικά όπως προαναφέρθηκε. Τα αρνητικά του είναι ότι έχει περιορισμένο εύρος χειρονομιών το οποίο βέβαια μπορεί να ξεπεραστεί και ότι η σωστή βαθμονόμηση μπορεί να πάρει αρκετή ώρα μέχρι να επιτευχθεί που ακόμα και τότε υπάρχει περίπτωση να μην είναι τέλεια.

PROS

- Γρήγοροι Χρόνοι Αντίδρασης
- Μεγάλη Γκάμα Εφαρμογών
- Εύκολο στην Χρήση
- Ελαστικό design

CONS

- Περιορισμένος Αριθμός Χειρονομιών
- Αρκετός χρόνος για σωστή βαθμονόμηση

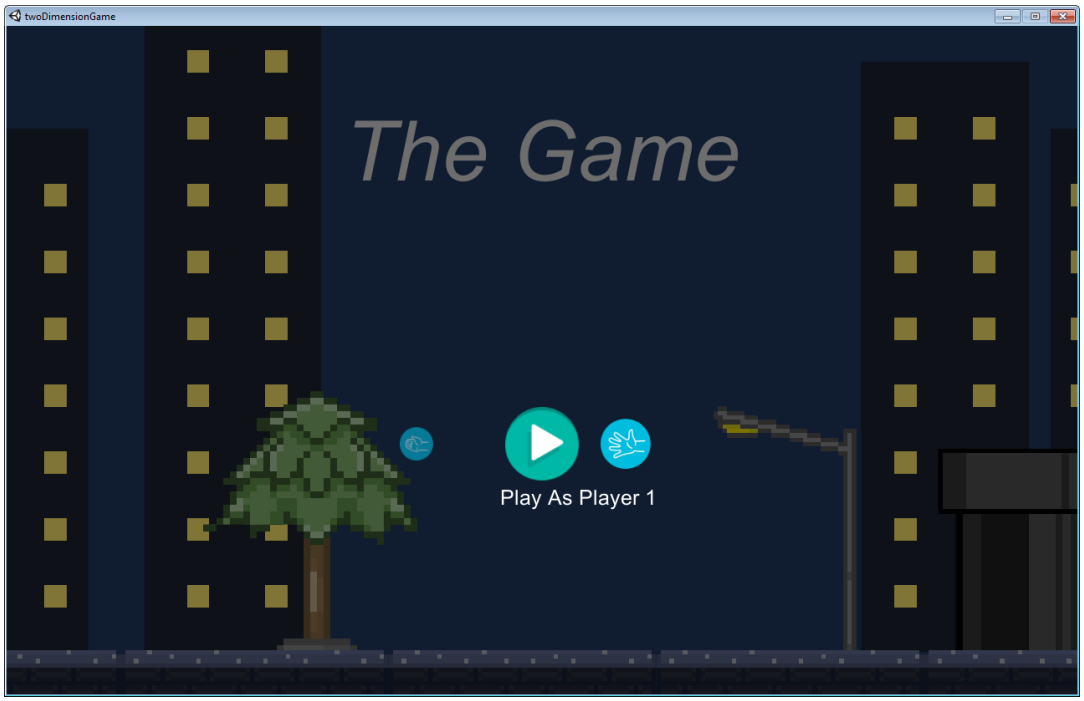
4 Επισκόπηση Παιχνιδιού

Σε αυτό το κεφάλαιο θα αναλύσουμε τις σκηνές του παιχνιδιού όσον αφορά την χρήση τους, για τον κώδικα θα γίνει επεξήγηση στο επόμενο κεφάλαιο. Οι σκηνές είναι οι εξής :

- Επιλογή Παίκτη
- Κεντρικό Μενού
- Μέγιστες Βαθμολογίες
- Οδηγίες
- Επιλογή Επιπέδου
- Επίπεδα 1 - 5

4.1 Σκηνή : Επιλογή Παίκτη

Σε αυτή την σκηνή ο χρήστης έχει την επιλογή να διαλέξει τον λογαριασμό στον οποίο θα παίζει. Σε περίπτωση που δεν έχει υπάρξει προηγούμενος παίκτης τότε η σκηνή αυτή παραλείπετε και δημιουργείται αυτόματα ο "Player 1", από εκεί και έπειτα σε κάθε επανέναρξη του παιχνιδιού εμφανίζεται στην οθόνη η επιλογή να παίξεις ως ένας από τους υπάρχων παίκτες ή να δημιουργήσεις ένα νέο λογαριασμό.



Εικόνα 12 : Σκηνή Επιλογής Παίκτη

Όπως φαίνεται και στην **εικόνα 12**, ο χρήστης με την χρήση των χειρονομιών Wave_In & Wave_Out μπορεί να αλλάξει μεταξύ λογαριασμών. Όταν τελειώσουν οι χρήστες στην λίστα εμφανίζεται σαν τελευταία επιλογή η δημιουργία ενός νέου λογαριασμού. Όταν ο χρήστης έχει διαλέξει τον λογαριασμό που επιθυμεί με την χειρονομία Fingers_Spread μεταφέρεται στην επόμενη σκηνή το "Κεντρικό Μενού".

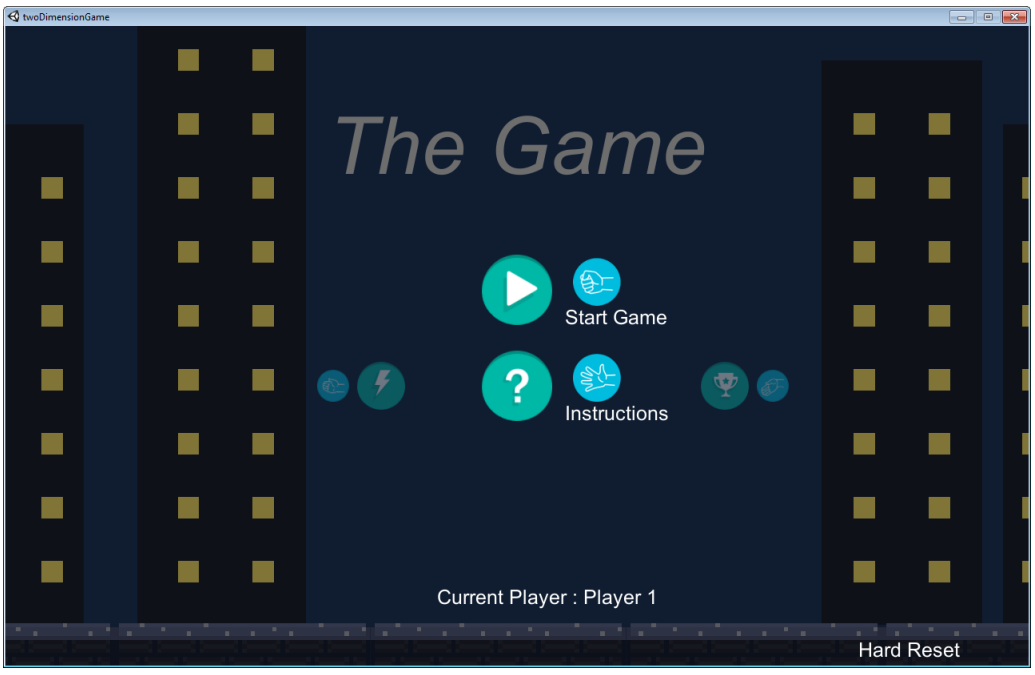
4.2 Σκηνή : Κεντρικό Μενού

Το Menu αποτελείται από 7 επιλογές :

- Choose Player : επιστροφή στην πρώτη σελίδα για την επιλογή λογαριασμού
- Instructions : μια σελίδα με οδηγίες για το πως δουλεύει το παιχνίδι.
- Exit : έξοδος από το παιχνίδι
- Levels : μια σελίδα που επιτρέπει στον χρήστη να διαλέξει πια πίστα θα παίξει, για να μπορέσει να διαλέξει μια πίστα σημαίνει ότι πρέπει να έχει κερδίσει την προηγούμενη.
- High Scores : μια σελίδα με τα 5 κορυφαία score όλων των χρηστών
- Start Game : ξεκινάει το $1^{ο}$ επίπεδο του παιχνιδιού.
- Hard Reset : διαγράφει από την μνήμη του παιχνιδιού όλα τα δεδομένα

Η επιλογή του "Start Game" μπορεί να επιλεχθεί με την χειρονομία Fist, ενώ μπορεί να κάνει εναλλαγή όλων των άλλων επιλογών εκτός του "Hard Reset" με τις χειρονομίες Wave_In & Wave_Out και να την επιλέξει με το Fingers_Spread. Όπως προαναφέρθηκε στο $2^{ο}$ κεφάλαιο το τέλειο calibration δεν είναι εύκολο να επιτευχθεί, οπότε για λόγους ασφαλείας για να μην ενεργοποιηθεί η επιλογή "Hard Reset" κατά λάθος έχει παραληφθεί από τις επιλογές του Myo.

Στο κάτω μέρος της οθόνης φαίνεται ποιος χρήστης είναι ενεργός σε αυτή την σύνδεση. Στην **εικόνα 13** φαίνεται ότι ενεργός σε αυτή την σκηνή είναι ο "Player 1".

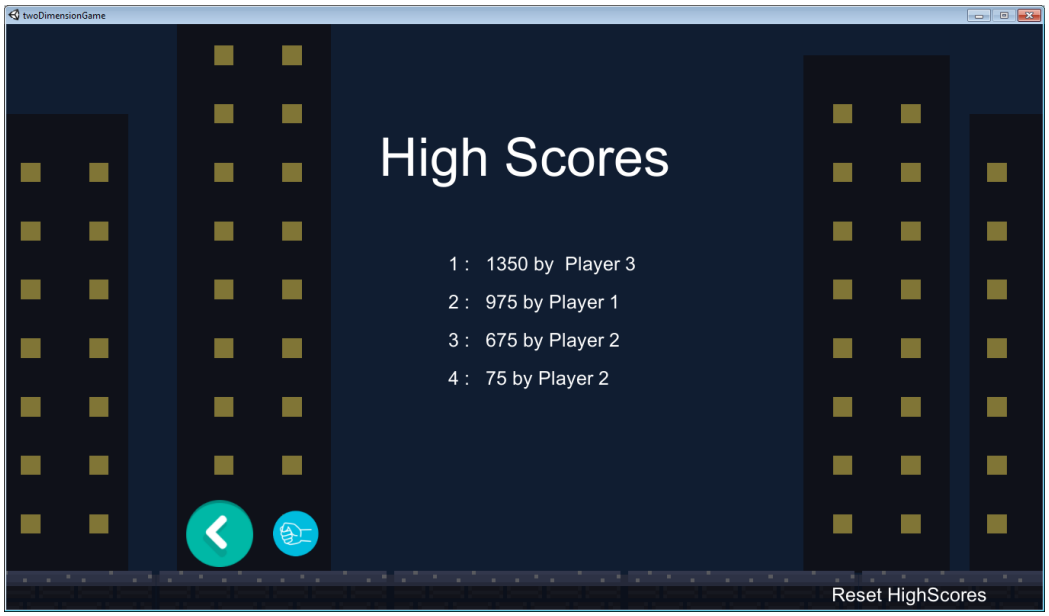


Εικόνα 13 : Σκηνή Κεντρικό Μενού

4.3 Σκηνή : Μέγιστες Βαθμολογίες

Σε αυτή την σελίδα εμφανίζονται οι 5 κορυφαίες βαθμολογίες που έχουν επιτευχθεί από τους χρήστες. Σε περίπτωση που δεν υπάρχουν αρκετά απλά εμφανίζονται όσα υπάρχουν. Στην σελίδα επίσης υπάρχουν 2 κουμπιά, ένα επιστρέφει τον χρήστη στο "Κεντρικό Μενού" και άλλο ένα το οποίο διαγράφει τις βαθμολογίες που υπάρχουν στην

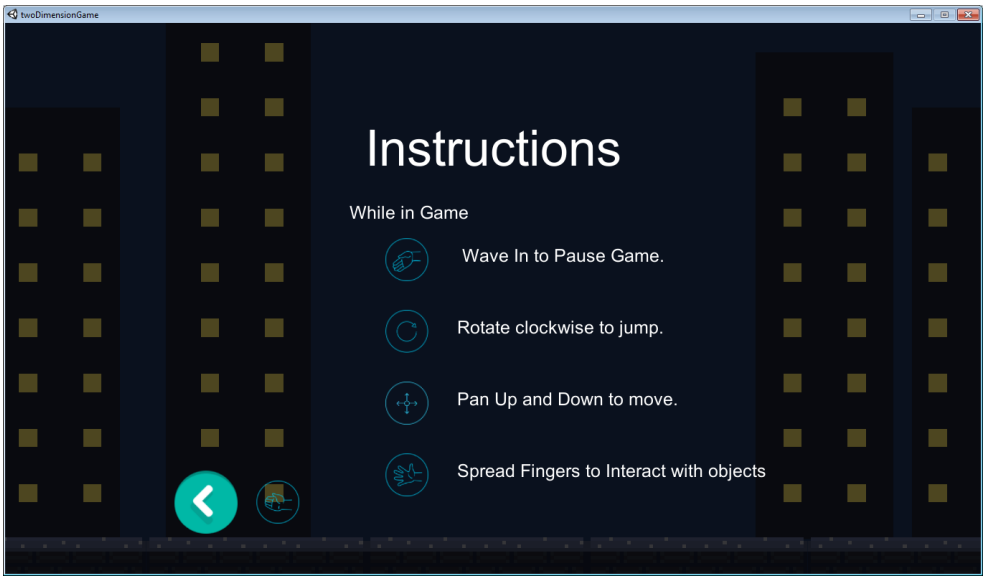
λίστα. Το κουμπί αυτό έχει την ίδια λογική με το κουμπί “Hard Reset” που βρίσκεται στο “Κεντρικό Μενού”. Για να περαστεί η βαθμολογία ενός χρήστη στην λίστα πρέπει να τελειώσει το παιχνίδι που παίζει, δηλαδή πρέπει να φτάσει στον τερματισμό όλων των πιστών ή να χάσει όλες τις ζωές του.



Εικόνα 14 : Σκηνή Μέγιστες Βαθμολογίες

4.4 Σκηνή : Οδηγίες

Σε αυτή την σελίδα ο χρήστης μπορεί να δει πως μπορεί να μετακινεί τον χαρακτήρα εντός παιχνιδιού. Για την μετακίνηση του χαρακτήρα μπρος-πίσω ο χρήστης πρέπει να αλλάζει το pitch όπως φαίνεται στην **εικόνα 10**, άρα με το κατέβασμα του χεριού προς τα κάτω ο χαρακτήρας πάει μπροστά και με το ανέβασμα του χεριού προς τα πίσω ο χαρακτήρας πηγαίνει προς τα πίσω. Με την περιστροφή του χεριού δεξιόστροφα, ο χαρακτήρας πηδάει στον αέρα ενώ με την χρήση της χειρονομίας Fingers_Spread ο χαρακτήρας κάνει χρήση των σωλήνων. Επίσης με την χειρονομία Wave_In ενεργοποιείται το “Μενού Παύσης” το οποίο θα αναλύσουμε σε επόμενο κεφάλαιο.



Εικόνα 15 : Σκηνή Οδηγίες

4.5 Σκηνή : Επιλογή Επιπέδου

Σε αυτή την σκηνή ο χρήστης με την χρήση των Wave_In & Wave_Out μπορεί να μετακινείται μεταξύ των επιπέδων που έχει ξεκλειδώσει. Για να εκκινήσει το επίπεδο που επιθυμεί πρέπει πρώτα να τοποθετήσει τον χαρακτήρα κάτω από το κουτί με τον αριθμό και έπειτα με την χειρονομία Fist μεταφέρεται στο επίπεδο αυτό. Για να φύγει από αυτήν την σκηνή μπορεί με την χειρονομία Fingers_Spread να επιστρέψει στο "Κεντρικό Μενού". Τα επίπεδα που δεν έχουν ξεκλειδωθεί ακόμα έχουν ένα λουκέτο μπροστά και δεν είναι προσβάσιμα.

Εικόνα 16 : Σκηνή Επιλογή Επιπέδου

4.6 Επίπεδα 1 - 5

Στα επίπεδα 1 έως 5 ο χρήστης έχει σαν σκοπό να τερματίσει τις πίστες με τη μεγαλύτερη δυνατόν βαθμολογία. Η βαθμολογία περνάει από το 1 επίπεδο στο άλλο για να υπάρχει μια τελική βαθμολογία στο τέλος του παιχνιδιού. Για το πως μετακινείται ο χαρακτήρας έχει αναφερθεί στο **κεφάλαιο 4.4** με τις οδηγίες. Στα επίπεδα αυτά υπάρχουν διάφορα αντικείμενα που μπορεί να συλλέξει ο χαρακτήρας τα οποία έχουν και άλλη επίδραση. Αυτά είναι :

- Pizza : προσθέτει μια ζωή στον χαρακτήρα.
- Soda : απλά προσθέτει πόντους.
- Milk : κάνει τον χαρακτήρα μεγαλύτερο σε μέγεθος και αν τυχόν χτυπηθεί από κάποιον αντίπαλο τον επιστρέφει στο αρχικό μέγεθος και για 1 δευτερόλεπτο δεν μπορεί να πεθάνει.
- Alcohol : ο χαρακτήρας παίρνει ένα πράσινο χρώμα και για τα επόμενα 10 δευτερόλεπτα δεν μπορεί να χτυπηθεί από κάποιον αντίπαλο και με το άγγιγμα ο αντίπαλος πεθαίνει.

Όλα τα αντικείμενα προσθέτουν πόντους στην βαθμολογία του χρήστη η οποία φαίνεται στην **εικόνα 17** στην πάνω δεξιά γωνία.

Στην πάνω αριστερή γωνία φαίνονται πόσες ζωές έχουν απομείνει στον χρήστη. Στην **εικόνα 17** επίσης μπορούμε να δούμε 4 αντιπάλους Android που είναι ένα από τα 5 πράγματα που μπορούν να σκοτώσουν τον χαρακτήρα. Αυτά είναι :

- Ρομπότ Android : υπάρχουν τα απλά και τα "Red" τα οποία χρειάζεται να χτυπηθούν 2 φορές για να πεθάνουν.
- Καρφιά : υπάρχουν 2 ειδών καρφιά, αυτά που βγαίνουν από ορισμένους σωλήνες και τα αυτά που περιστρέφονται συνεχώς σε ένα σταθερό σημείο.
- Νυχτερίδες : έχουν την ίδια λογική με τα Ρομπότ Android με την μόνη διαφορά ότι αντί να μετακινούνται στο έδαφος είναι στον αέρα.
- Κανόνι : όταν ο χαρακτήρας πλησιάσει αρκετά αρχίζει και πυροβολεί τον χρήστη, για να πεθάνει χρειάζεται να χτυπηθεί 3 φορές.
- Κενό : αν ο χαρακτήρας πέσει στο κενό είναι φυσιολογικό να πεθαίνει.

Όταν ο χαρακτήρας μας πεθάνει επιστρέφει στο τελευταίο CheckPoint από το οποίο έχει περάσει.

Εικόνα 17 : Παίζοντας το Επίπεδο 1

Όπως προαναφέρθηκε στις οδηγίες ο χρήστης μπορεί να κάνει παύση του παιχνιδιού. Όταν γίνει αυτό σταματάνε τα πάντα και εμφανίζεται ένα Menu με τις παρακάτω επιλογές :

- Resume : επιστρέφει στο παιχνίδι
- Try Again : ξαναρχίζει το επίπεδο από την αρχή.
- Quit : επιστρέφει στο "Κεντρικό Μενού"
- Change Audio : η σίγαση και η κατάργηση της σίγασης του ήχου.

Σε περίπτωση που ο χρήστης τερματίσει το επίπεδο τότε εμφανίζεται ένα παρόμοιο μενού το οποίο εμφανίζει στον χρήστη το επίπεδο το οποίο τερμάτισε, την βαθμολογία που έχει και τις επιλογές :

- Continue : μεταφέρει τον χαρακτήρα στο επόμενο επίπεδο.
- Quit : επιστρέφει στο "Κεντρικό Μενού"

- Change Audio : η σίγαση και η κατάργηση της σίγασης του ήχου.

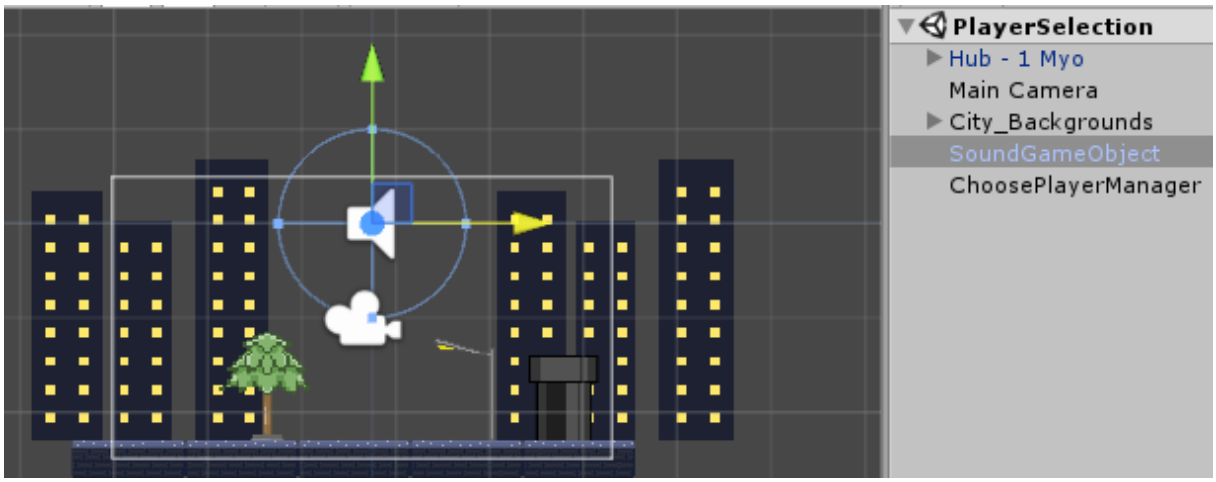
5 Ανάλυση Παιχνιδιού και Κώδικα

Σε αυτό το κεφάλαιο αρχικά θα αναλύσουμε των κώδικα για όλες τις σκηνές εκτός των επιπέδων και έπειτα θα αναλύσουμε όλα τα αντικείμενα που βρίσκονται στα 5 επίπεδα του παιχνιδιού. Επίσης θα γίνει επεξήγηση του κώδικα που παρέχεται από το Myo Unity Package και του τρόπου χρήσης του.

5.1 Ανάλυση Σκηνών

5.1.1 Επιλογή Παίκτη

Σε αυτή την σκηνή υπάρχουν μερικά αντικείμενα στον "φάκελο" City_Backgrounds



τα οποία είναι τοποθετημένα για να μην είναι κενή η σκηνή. Επίσης μπορούμε να δούμε ότι εκτός τα αντικείμενα του background υπάρχουν το Hub – 1 Myo , η κάμερα, το SoundGameObject και το ChoosePlayerManager που περιέχει το βασικό script της σκηνής. Για το Hub – 1 Myo θα γίνει αναφορά στο **5.6 Myo Analysis**.

```
static KeepSoundBetweenScenes instance = null;
public static KeepSoundBetweenScenes Instance
{
    get { return instance; }
}
void Awake()
{
if (instance != null && instance != this)
{
    Destroy(this.gameObject); return;
 }else{
    instance = this;
 }
    DontDestroyOnLoad(this.gameObject);
}
void OnApplicationQuit()
{
   Application.CancelQuit();
   System.Diagnostics.Process.GetCurrentProcess().Kill();
}
```


Κώδικας 1 : Διατήρηση Ήχου Μεταξύ Σκηνών

Το SoundGameObject περιέχει 2 components τα οποία είναι ένα αρχείο ήχου και ένα script το οποίο διατηρεί το αντικείμενο αυτό ενεργό στις αλλαγές των σκηνών για την διατήρηση της μουσικής και για την αποφυγή crash της εφαρμογής κατά την έξοδο από αυτή.

Στον κώδικα δίπλα φαίνεται ότι μόλις εκκινήσει το script ελέγχουμε για την ύπαρξη ενός στιγμιότυπού του αντικειμένου μας και αν υπάρχει καταστρέφουμε το παρόν αντικείμενο, αλλιώς διατηρούμε αυτό σαν στιγμιότυπο. Με τον κώδικα DontDestroyOnLoad() διατηρούμε το αντικείμενο αυτό ενεργό μεταξύ των σκηνών. Επίσης έχω προσθέσει την function OnApplicationQuit() η οποία ενεργοποιείται όταν ο χρήστης κάνει έξοδο από την εφαρμογή. Το κομμάτι κώδικα αυτό καταργεί την έξοδο που έχει ζητήσει ο χρήστης και καταστρέφει την τωρινή διεργασία. Ο λόγος που το χρησιμοποιώ είναι ότι όταν ο χρήστης προσπαθήσει να κάνει κανονικά έξοδο το πρόγραμμα κολλούσε και δεν αντιδρούσε με αποτέλεσμα ο μοναδικός τρόπος για να κλείσει το πρόγραμμα ήταν μέσω του "Task Manager" του υπολογιστή.

```
if (PlayerPrefs.GetInt("NumOfPlayers") > 0)
{
currentPlayerNum = PlayerPrefs.GetInt("NumOfPlayers") + 1;
}else{
currentPlayerNum = 1;
CreateNewAndgoToMainMenu();
}
```

Κώδικας 2 : Επιλογή Χαρακτήρα, Start

Στον **κώδικα 2** φαίνεται ένα κομμάτι της Start() διεργασίας το οποίο κάνει έλεγχο στον ακέραιο "NumOfPlayers" ο οποίος είναι αποθηκευμένος στην μνήμη του υπολογιστή λόγω του "PlayerPrefs", αν ο ακέραιος είναι μεγαλύτερος του μηδενός τότε σημαίνει ότι υπάρχουν ήδη παίκτες οπότε βάζουμε στον τωρινό χρήστη τον αριθμό του συνόλου των παικτών + 1. Στην περίπτωση που δεν υπάρχει άλλος παίκτης δίνουμε στον χρήστη τον αριθμό 1 και εκκινούμε την διεργασία CreateNewAndgoToMainMenu() .

```
void CreateNewAndgoToMainMenu()
    {
        if (PlayerPrefs.GetInt("NumOfPlayers") > 0)
            PlayerPrefs.SetInt("NumOfPlayers", currentPlayerNum);
        else
            PlayerPrefs.SetInt("NumOfPlayers", 1);


        PlayerPrefs.SetInt("CurrentPlayer", currentPlayerNum);

        ThalmicHub.Destroy(ThalmicHub.instance);
SceneManager.LoadScene("MainMenu");
    }
```

Κώδικας 3 : Δημιουργία Παίκτη

Ο **κώδικας 3** κάνει τον ίδιο έλεγχο του **κώδικα 2** και αποθηκεύει τον νέο αριθμό παικτών στην μεταβλητή "NumOfPlayers". Έπειτα στην μεταβλητή "CurrentPlayer" τοποθετεί τον αριθμό του τωρινού παίκτη, καταστρέφουμε το στιγμιότυπο του ThalmicHub

αφού σε κάθε σκηνή υπάρχει ένα, θα εξηγήσουμε τον λόγω που γίνεται αυτό στο κεφάλαιο **5.6 Myo Analysis,** και τελικά με την χρήση του SceneManager μεταφερόμαστε στην σκηνή του "Κεντρικού Μενού".

```
void Update ()
    {
        ThalmicMyo thalmicMyo = myo.GetComponent<ThalmicMyo>(); // set myo

        if (thalmicMyo.pose != _lastPose)
        {
            _lastPose = thalmicMyo.pose;
            if (thalmicMyo.pose == Pose.WaveOut)
                if (counter <= PlayerPrefs.GetInt("NumOfPlayers"))
                    counter++;

            if (thalmicMyo.pose == Pose.WaveIn)
                if (counter > 1)
                    counter--;

        }
        if (thalmicMyo.pose == Pose.FingersSpread)
            PlayerChosenAndGoToMainMenu();
    }
```

Κώδικας 4 : Επιλογή Παίκτη , Update()

Στον **κώδικα 4** βλέπουμε την διεργασία Update η οποία εκτελείται συνέχεια, σε αυτή δηλώνουμε ότι το thalmicMyo είναι το script που βρίσκεται στο αντικείμενο myo της σκηνής μας και έπειτα κάνουμε έλεγχο για τις χειρονομίες που κάνει ο χρήστης. Κάνουμε έλεγχο για το αν ο χρήστης έχει χρησιμοποιήσει νέα χειρονομία, αν έχει αλλάξει τότε στην περίπτωση της χειρονομίας WaveOut προσθέτουμε 1 μονάδα στον counter ενώ στην περίπτωση της WaveIn αφαιρούμε. Αν ο χρήστης κάνει την χειρονομία FingersSpread τότε εκτελούμε την διεργασία PlayerChosenAndGoToMainMenu.

```
void PlayerChosenAndGoToMainMenu()
    {
        PlayerPrefs.SetInt("CurrentPlayer", counter);
        ThalmicHub.Destroy(ThalmicHub.instance);
        SceneManager.LoadScene("MainMenu");
    }
```

Κώδικας 5 : Παικτής Επιλέχθηκε

Η διεργασία αυτή αλλάζει την μεταβλητή "CurrentPlayer" στον αριθμό που βρίσκεται στον counter ο οποίος αλλάζει με βάση τον **κώδικα 4** της **Update()** και έπειτα όπως η διεργασία στον **κώδικα 3** καταστρέφουμε το στιγμιότυπο του ThalmicHub και πηγαίνουμε στο MainMenu.

Στον **κώδικα 6** εμφανίζεται η OnGUI() που διαχειρίζεται το γραφικό περιβάλλον. Έχω δημιουργήσει ένα skin το οποίο είναι μια συλλογή από GUIStyles που μπορούν να εφαρμοστούν στα αντικείμενα του GUI. Τοποθετώ στο background μια μαύρη εικόνα που

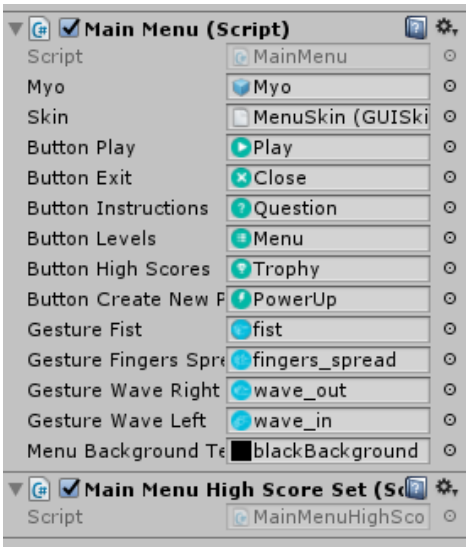
παίρνει όλο το μέγεθος της οθόνης και της χαμηλώνω το alpha ώστε να φαίνονται τα στοιχεία που βρίσκονται από πίσω. Έπειτα τοποθετώ τον τίτλο του παιχνιδιού και δημιουργώ μερικά ορθογώνια στα οποία θα τοποθετήσω γραφικά. Όταν ο counter γίνει μεγαλύτερος από το σύνολο των παικτών, τότε εμφανίζεται στην οθόνη ένα κουμπί που ενεργοποιεί την διεργασία του **κώδικα 3**. Σε αντίθετη περίπτωση ενεργοποιείται η διεργασία του **κώδικα 5.**

```
void OnGUI()
    {
        GUI.skin = skin;
        GUI.color = new Color(1, 1, 1, 0.5f);   // Lower Alpha to 0.5
        GUI.DrawTexture(new Rect(0, 0, Screen.width, Screen.height),
menuBackgroundTexture);
        GUI.color = Color.white; // Get Alpha back up

        GUI.Label(new Rect(Screen.width / 2 - 250, Screen.height / 2 - 350, 500,
200), "The Game");   // GAME TITLE

        Rect secondRowLeftRect = new Rect(Screen.width / 2 - 50, Screen.height / 2
+ 50, 100, 100);
        Rect secondRowTextRect = new Rect(Screen.width / 2 - 50, Screen.height / 2
+ 150, 100, 50);
        Rect secondRowGestureRect = new Rect(Screen.width / 2 + 70, Screen.height /
2 + 70, 60, 60);

        Rect leftGestureRect = new Rect(Screen.width / 2 - 170, Screen.height / 2 +
80, 40, 40);
        Rect rightGestureRect = new Rect(Screen.width / 2 + 220, Screen.height / 2
+ 80, 40, 40);

            if(counter != PlayerPrefs.GetInt("NumOfPlayers") + 1)
            {
                if (GUI.Button(secondRowLeftRect, selectPlayerTexture))
                    PlayerChosenAndGoToMainMenu();
                GUI.Label(secondRowTextRect, "Play As Player " + counter,
"MenuItems");
                GUI.DrawTexture(secondRowGestureRect, gestureFingersSpread);
            }
            else
            {
                if (GUI.Button(secondRowLeftRect, selectPlayerTexture))
                    CreateNewAndgoToMainMenu();

                GUI.Label(secondRowTextRect, "Create Player " +
currentPlayerNum.ToString(), "MenuItems");
                GUI.DrawTexture(secondRowGestureRect, gestureFingersSpread);
            }

        GUI.color = new Color32(255, 255, 255, 150);    // Lower Opacity
        if(counter != 1)
            GUI.DrawTexture(rightGestureRect, gestureWaveIn);

      if(counter != PlayerPrefs.GetInt("NumOfPlayers") + 1 && currentPlayerNum > 1)
            GUI.DrawTexture(leftGestureRect, gestureWaveOut);

        GUI.color = new Color32(255, 255, 255, 255);    // Max opacity
    }
```

Κώδικας 6 : Γραφικό Περιβάλλον

5.1.2 Κεντρικό Μενού

Αυτή η σκηνή αποτελείται από το background, την κάμερα, το Hub – 1 Myo και το MainMenuManager το οποίο έχει τον κώδικα MainMenu και MainMenuHighScoreSet τα οποία χειρίζονται το Κεντρικό Μενού και την τοποθέτηση της νέας βαθμολογίας στην σκήνη των βαθμολογιών αντίστοιχα.



Κώδικας 7 : Κεντρικό Μενού, Update ( )

Εικόνα 19 : Επιθεώρηση Διαχειριστή Κεντρικού Μενού

```
if (thalmicMyo.pose == Pose.Fist)
                NewGame();
            if (thalmicMyo.pose == Pose.FingersSpread)
            {
                switch (menuCounter)
                {
                    case 0: LevelSelectionPage(); break;

                    case 1: HighScoresPage(); break;

                    case 2: InstructionsPage(); break;

                    case 3: CreateNewPlayerPage(); break;

                    case 4: ExitGame(); break;

                    default: ExitGame(); break;
                }
            }
```

Ο κώδικας του MainMenu δουλεύει με τον ίδιο τρόπο που δουλεύει και ο κώδικας της σκηνής επιλογής χρήστη, εδώ πάλι με ένα counter κάνουμε περιστροφή των αντικειμένων του μενού με την χρήση των WaveIn & WaveOut και με την χρήση του FingersSpread γίνεται η επιλογή για να τρέξει η διεργασία. Σε περίπτωση που ο χρήστης θέλει να εκκινήσει νέο παιχνίδι υπάρχει μόνιμα η επιλογή στην οθόνη για ευκολία.

Οι διεργασίες του **κώδικα 7** μεταφέρουν τον χρήστη στις σελίδες που μπορεί να φανεί και από τους τίτλους των διεργασιών. Επίσης υπάρχει και η επιλογή του HardReset η οποία μηδενίζει όλες τις μεταβλητές που αποθηκεύονται στην μνήμη και μεταφέρει τον χρήστη στην σελίδα επιλογής χρήστη η οποία βέβαια λόγω μη ύπαρξης άλλου παίκτη μας επιστρέφει πάλι στο menu αλλά ως χρήστη 1.

5.1.3 Μέγιστες Βαθμολογίες και Οδηγίες

Η σκηνή Μέγιστες Βαθμολογίες παίρνει τα “HighScoreX” (X = αριθμός) δεδομένα από το PlayerPrefs και αν είναι μεγαλύτερο του μηδενός τότε το εμφανίζει στην οθόνη σε μια λίστα με μέγιστο 5 βαθμών. Επίσης υπάρχουν 2 κουμπιά από το γραφικό περιβάλλον (GUI) από τα οποία το ένα επιστρέφει τον χρήστη στο menu με την χρήση της χειρονομίας Fist και το άλλο με το πάτημα του διαγράφει τους βαθμούς από την λίστα.

Η σκηνή Οδηγίες αποτελείται μόνο από γραφικά τα οποία απλά εμφανίζουν οδηγίες και εικόνες στην οθόνη. Επίσης υπάρχει ένα κουμπί που επιστρέφει τον χρήστη στο κεντρικό μενού με το πάτημα του ή με την χρήση της χειρονομίας WaveOut.

5.1.4 Επιλογή Επιπέδου

Αυτή η σκηνή αποτελείται από το Hub, τα σκηνικά στο background , το γραφικό κομμάτι του χαρακτήρα που περιέχει τον κώδικα της σκηνής και 5 αντικείμενα με τους αριθμούς των επιπέδων, επίσης κάθε αντικείμενο από αυτά περιέχει ένα αντικείμενο κλειδαριά για τον συμβολισμό ότι το επίπεδο είναι κλειδωμένο.

```
public GameObject[] locks;
public bool[] levelUnlocked;

void Start()
    {
        for (int i = 1; i <=5; i++)
        {
            if(PlayerPrefs.GetInt("Player" + PlayerPrefs.GetInt("CurrentPlayer") +
"Level") < i)
                levelUnlocked[i-1] = false;
            else
                levelUnlocked[i-1] = true;

            if (levelUnlocked[i-1])
                locks[i-1].SetActive(false);

        }
        positionSelector = PlayerPrefs.GetInt("Player" +
PlayerPrefs.GetInt("CurrentPlayer") + "Level") - 1;
        transform.position = locks[positionSelector].transform.position + new
Vector3(0, distanceBelow, 0);
    }
```

Κώδικας 8 : Διαχειριστής Επιλογής Επιπέδου

Στον **κώδικα 8** αρχικοποιούμε 2 πίνακες τους οποίους στον inspector του Unity βάζουμε 5 θέσεις στον καθένα, στον πρώτο βάζουμε τα λουκέτα και τον 2^{ο} τον αφήνουμε κενό διότι αλλάζει μέσα από τον κώδικα. Στην συνέχεια σε μια loop συγκρίνουμε τον αριθμό του τελευταίου επιπέδου που έχει τερματίσει ο χρήστης με τον αριθμό της loop για να μπορέσουμε να τοποθετήσουμε τα λουκέτα μπροστά από τα επίπεδα.

Σε ένα μετρητή τοποθετούμε τον αριθμό του επιπέδου που έχει τερματίσει ο χρήστης μείον μια μονάδα. Ο λόγος που αφαιρούμε μια μονάδα είναι ότι οι πίνακες που χρησιμοποιούμε ξεκινάνε από το 0 ενώ τα επίπεδα ξεκινάνε από το 1. Στην συνέχεια τοποθετούμε τον χαρακτήρα στην θέση του λουκέτου που βρίσκεται ο μετρητής και προσθέτουμε μια απόσταση στον κατακόρυφο άξονα για να τοποθετηθεί λίγο πιο κάτω από το λουκέτο και όχι πάνω του.

```
transform.position = Vector3.MoveTowards(transform.position,
locks[positionSelector].transform.position + new Vector3(0, distanceBelow, 0),
moveSpeed*Time.deltaTime);
```

Κώδικας 9 : Μετακίνηση προς νέο προορισμό

Η διεργασία Update δουλεύει με τον ίδιο τρόπο με τις 2 προηγούμενες, δηλαδή χρησιμοποιώντας ένα μετρητή, επίσης έχει και τον **κώδικα 9** μέσα από τον οποίο κάνει τον

χαρακτήρα να μετακινηθεί προς το επόμενο σημείο. Με βάση την αλλαγή του positionSelector, η διεργασία MoveTowards που παίρνει δεδομένα **(τωρινές συντεταγμένες , συντεταγμένες στόχου, βηματισμός)** μεταφέρει τον χαρακτήρα μας προς την επόμενη τοποθεσία που του έχουμε ζητήσει με το βήμα που του έχουμε βάλει.

5.2 Ανάλυση Αντικειμένων

Σε αυτό το κομμάτι θα αναλύσουμε όλα τα αντικείμενα με τα οποία αλληλοεπιδρά ο χαρακτήρας.

5.2.1 Συλλέξιμα Αντικείμενα

Αυτά είναι τα αντικείμενα τα οποία μπορεί να συλλέξει ο χαρακτήρας εάν περάσει από πάνω τους. Κάθε ένα από αυτά έχει και άλλο αποτέλεσμα.

- Alcohol : Αλλάζει τον χαρακτήρα σε πράσινο για 10 δευτερόλεπτα και σε περίπτωση που χτυπήσει αντίπαλο τον σκοτώνει απλά με το πέρασμα του.
- Milk : Μεγαλώνει το μέγεθος του χαρακτήρα και του δίνει διπλή ευκαιρία, δηλαδή αν τον χτυπήσει αντίπαλος δεν πεθαίνει αμέσως όπως θα γινόταν σε άλλη περίπτωση αλλά τον αφήνει να συνεχίσει το παιχνίδι.
- Pizza : προσθέτει 1 ζωή στον χαρακτήρα.
- Soda : απλά προσθέτει πόντους στον χαρακτήρα

Εικόνα 20 : Συλλέξιμα Αντικείμενα

```
    AudioSource source;
    bool taken = false;

    void Start()
    {
        source = gameObject.GetComponent<AudioSource>();
    }

    void Update()
    {
        if (taken)
        {
            if (!source.isPlaying)
                Destroy(this.gameObject);
        }
    }

    void OnTriggerEnter2D(Collider2D col)
    {
        if (col.transform.CompareTag("Player"))
        {
            source.Play();
            Player.playerStatus = 2;
            ScoreManager.AddPoints(75);
            taken = true;
        }
    }
```

Κώδικας 10 : Αντικείμενο Alcohol

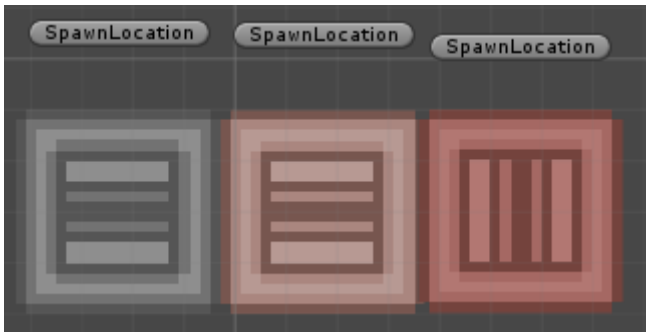
Στον **κώδικα 10** μπορούμε να δούμε ότι γίνεται η αρχικοποίηση του στοιχείου source το οποίο είναι ένα αρχείο ήχου που έχει προστεθεί στα components του Alcohol. Σε περίπτωση που ο χρήστης ακουμπήσει το αντικείμενο αυτό τότε, παίζουμε το αρχείο ήχου, αλλάζουμε το Status του χαρακτήρα σε 2 (θα εξηγηθεί στο **κεφάλαιο 5.5**) προσθέτουμε πόντους μέσω το ScoreManager (θα εξηγηθεί στο **κεφάλαιο 5.4**). Ο λόγος που χρησιμοποιώ την μεταβλητή taken είναι ότι αν κατέστρεφα κατευθείαν το αντικείμενο, ο ήχος που περιέχει δεν θα έπαιζε ποτέ, άρα μέσα στην update κάνω ένα έλεγχο για το πότε έχει σταματήσει να παίζει ο ήχος για να καταστρέψω το αντικείμενο. Τον ίδιο κώδικα χρησιμοποιώ και για το Milk με την διαφορά ότι αλλάζω το Status σε 1. Για τα αντικείμενα Pizza και Soda που δεν περιέχουν ήχο είναι πιο απλά τα πράγματα αφού χρησιμοποιούν μόνο το κομμάτι του OnTriggerEnter2D. Για την Pizza χρησιμοποιούμε επιπλέον την διεργασία AddLife() (θα εξηγηθεί στο **κεφάλαιο 5.5**).

```
void OnTriggerEnter2D(Collider2D col)
    {
        if (col.transform.CompareTag("Player"))
        {
            ScoreManager.AddPoints(100);
            col.GetComponent<Player>().AddLife();
            Destroy(gameObject);
        }
    }
```

Κώδικας 11 : Αντικείμενο Pizza

5.2.2 Κουτιά και Κέρματα

Υπάρχουν 3 είδη κουτιών, ένα για τα κέρματα, ένα για τις ζωές που δίνει την Pizza και ένα δυνάμεων που βγάζει Alcohol ή Milk ανάλογα το Status που έχει ο χαρακτήρας.



Εικόνα 21 : Κουτιά

```
    void OnCollisionEnter2D(Collision2D collision)
    {
        if (collision.gameObject.CompareTag("Player"))
        {
            bool hitOncePerJump = true;

            foreach (ContactPoint2D point in collision.contacts)
            {
                if (point.normal.y >= 0.5f)
                {
                    if (hitOncePerJump)
                    {
                        if (Player.playerStatus == 0)
                            Instantiate(Milk, SpawnLocation.transform.position,
Quaternion.identity);
                        else if (Player.playerStatus >= 1)
                            Instantiate(Alcohol, SpawnLocation.transform.position,
Quaternion.identity);

                        Destroy(this);
                    }
                }
                hitOncePerJump = false;
            }
            hitOncePerJump = true;
        }
    }
```

Κώδικας 12 : Λογική Κουτιών, Κουτί Δυνάμεων

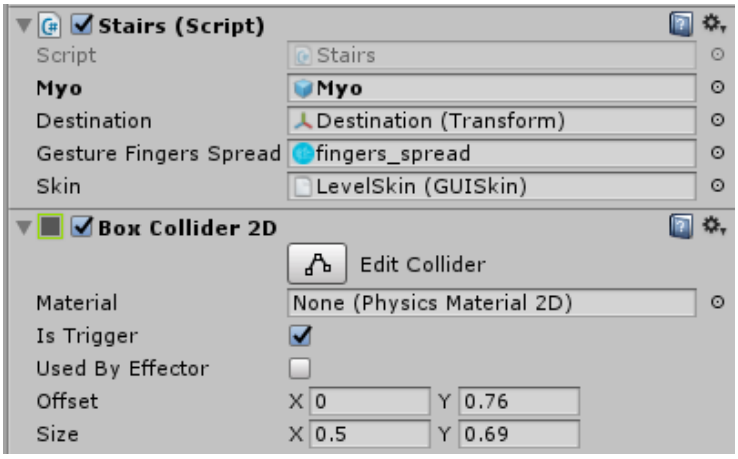
Στον **κώδικα 12** σε περίπτωση που κάτι έχει κάνει επαφή με το κουτί κάνουμε ένα έλεγχο να δούμε αν είναι ο χαρακτήρας μας, σε περίπτωση που είναι τότε παίρνουμε όλα τα σημεία που υπήρχε επαφή, αν ο χαρακτήρας ακούμπησε το κουτί από κάτω (αυτό φαίνεται από το **point.normal.y > 0.5f**) τότε ανάλογα με το Status του χαρακτήρα δημιουργούμε ένα από τα αντικείμενα Milk ή Alcohol στο σημείο του SpawnLocation το οποίο έχουμε επιλέξει και μπορεί να φανεί στην **εικόνα 21**. Μόλις γίνει η διαδικασία αυτή καταστρέφουμε τον κώδικα για να μην μπορεί ο χρήστης να πάρει αλλά αντικείμενα από το συγκεκριμένο κουτί. Το κουτί με τα κέρματα διαφέρει σε αυτό το κομμάτι διότι έχουμε βάλει ένα μετρητή για να μπορεί ο

χρήστης να πάρει περισσότερα κέρματα από το κουτί. Επίσης χρησιμοποιούμε μια μεταβλητή bool λόγω ότι ο χρήστης όταν χτυπάει το κουτί συνήθως έχει 2 σημεία επαφής με αποτέλεσμα να δημιουργούνται 2 αντικείμενα, παρότι ο χρήστης θα πάρει το effect του αντικειμένου μια φορά, θα πάρει τους διπλάσιους πόντους που δεν είναι κάτι που θέλουμε.

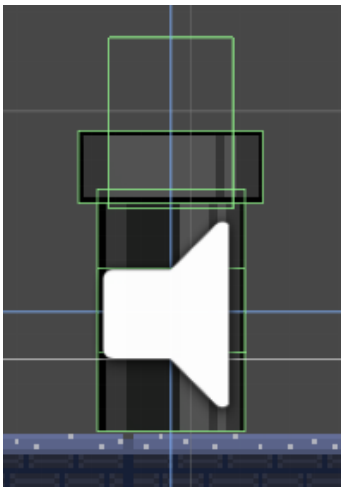
Το κέρμα εμφανίζεται στην σκηνή όπως προαναφέρθηκε, αποτελείται από 2 κομμάτια, 1^{ο} το CoinHolder το οποίο έχει τον ήχο και ένα κομμάτι κώδικα το οποίο καταστρέφει το αντικείμενο μετά από 2 δευτερόλεπτα και το 2^{ο} αντικείμενο είναι το ίδιο το κέρμα το οποίο έχει ένα animation το οποίο κάνει fade out μέχρι να εξαφανιστεί. Το animation και ο ήχος ξεκινάνε αμέσως μόλις δημιουργηθεί το κέρμα οπότε δεν χρειάζεται να γίνει η διαδικασία αυτή μέσω κώδικα.

5.2.3 Σωλήνες και Σκάλες

Οι σωλήνες και οι σκάλες έχουν ακριβώς τον ίδιο κώδικα. Όταν ο χαρακτήρας μπει στο εσωτερικό ενός TriggerCollider τότε αλλάζουμε την μεταβλητή canTravel σε αληθής και όσο ο χρήστης είναι μέσα στα όρια αν χρησιμοποιήσει την χειρονομία FingersSpread τότε μεταφέρετε στην τοποθεσία που έχουμε προκαθορίσει. Επίσης μόλις μπει ο χρήστης στα όρια εμφανίζουμε στην οθόνη μέσω του GUI το εικονίδιο της χειρονομίας για να ενημερώσουμε τον χρήστη ότι μπορεί να χρησιμοποιήσει την σκάλα ή τον σωλήνα στον οποίο βρίσκεται. Μόλις ο χρήστης κάνει την χειρονομία ακούγεται ένας ήχος ο οποίος ενεργοποιείται μέσω του κώδικα.



Εικόνα 22 : Επιθεωρητής Σωλήνων και Σκάλας



Εικόνα 23 : Σωλήνας στο Unity

5.2.4 Τερματισμός (Αυτοκίνητο)

Μόλις ο χρήστης φτάσει στον τερματισμό του επιπέδου ενεργοποιούνται δυο διεργασίες, η $1^{η}$ αποθηκεύει τα δεδομένα ώστε να περάσουνε στο επόμενο επίπεδο και η $2^{η}$ χρησιμοποιείται για να εμφανίσει το μενού τερματισμού. Χρησιμοποιούμε την διαδικασία της StartCoroutine η οποία μας δίνει την δυνατότητα να κάνουμε παύση της διεργασίας με την χρήση του **yield.** Σε αυτή την περίπτωση εξαφανίζουμε τον χαρακτήρα, ξεκινάμε το animation και τον ήχο που έχουμε βάλει στο αυτοκίνητο και έπειτα από 2 δευτερόλεπτα εμφανίζουμε το μενού τερματισμού (θα εξηγηθεί στο **κεφάλαιο 5.4**). Η διεργασία SaveScore εκτός από το να αποθηκεύει την βαθμολογία, κρατάει και τις ζωές που έχουν απομείνει στον χρήστη και ελέγχει το επίπεδο που έχει φτάσει ο χρήστης ώστε να αποθηκεύσει το υψηλότερο επίπεδο που έχει φτάσει για να μπορεί να εμφανιστεί στην σκηνή επιλογής επίπεδων.

```
void OnTriggerEnter2D(Collider2D col)
    {
        if(col.CompareTag("Player"))
        {
            SaveScore();
            StartCoroutine(LevelCompletedCo());
        }
    }

    public IEnumerator LevelCompletedCo()
    {
        player.SetActive(false);
        gameObject.GetComponent<Animation>().Play();
        source.Play();
        yield return new WaitForSeconds(2);
        pauseMenu.showMenu = true;
        pauseMenu.showWinScreen = true;
    }

    void SaveScore()
    {
        PlayerPrefs.SetInt("PlayerLifes",
player.GetComponent<Player>().currentPlayerLifes);
        PlayerPrefs.SetInt("PlayerScore", ScoreManager.score);

        if(PlayerPrefs.GetInt("Player" + PlayerPrefs.GetInt("CurrentPlayer") +
"Level") < currentLevel + 1 && currentLevel != 5)
            PlayerPrefs.SetInt("Player" + PlayerPrefs.GetInt("CurrentPlayer") +
"Level" , currentLevel + 1);
    }
```

Κώδικας 13 : Τερματισμός (Αυτοκίνητο)

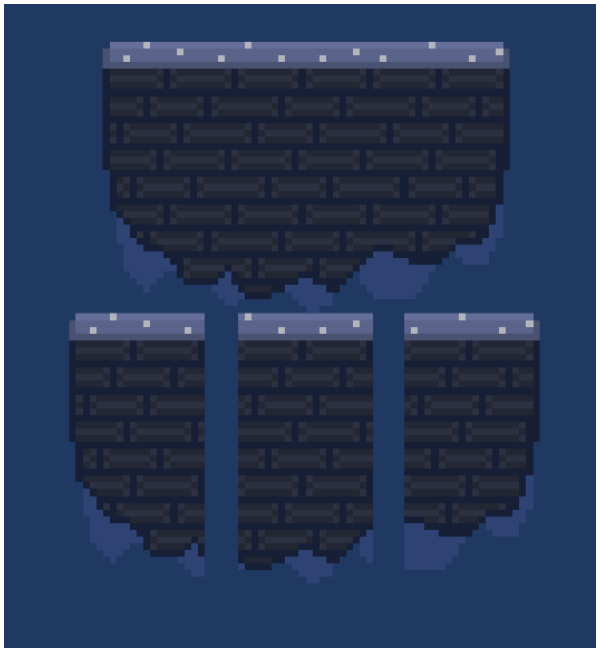
5.2.5 Σημεία Ελέγχου

Τα CheckPoints έχουν ένα TriggerCollider, όταν ο χρήστης μπει στα όρια του τότε αλλάζει το CurrentCheckPoint στον LevelManager σε αυτό το οποίο μόλις περάσαμε ώστε σε περίπτωση θανάτου ο παίκτης να ξεκινήσει από αυτό το σημείο αντί για την αρχή του επιπέδου.

Εικόνα 24 : Επιθεώρηση Διαχειριστή Επιπέδου

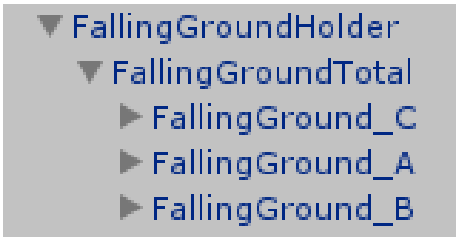
5.2.6 Πτώση Εδάφους

Το FallingGround αποτελείται από 3 κομμάτια τα οποία είναι τοποθετημένα μαζί σε ένα αντικείμενο για να μετακινούνται σαν σύνολο, έπειτα αυτό τοποθετείται σε ένα άλλο αντικείμενο "Holder" για την χρήση του Instantiate. Σε περίπτωση που δεν υπήρχε το αντικείμενο "Holder", το αντικείμενο FallingGroundTotal θα δημιουργούταν στις συντεταγμένες που ζητούσαμε σύμφωνα με τις παγκόσμιες συντεταγμένες αντί να πάρει υπόψιν τις συντεταγμένες του αντικειμένου "Holder" που έχει τον ρόλο του **parent.**



Εικόνα 25 : Πτώση Εδάφους

Στην **εικόνα 25** φαίνεται ο χωρισμός των επιμέρους κομματιών, όλα τα κομμάτια έχουν τον **κώδικα 14** και ο Holder έχει τον **κώδικα 15** που κάνει instantiate το TotalFallingGround στο σημείο του Holder.



Εικόνα 26 : Ιεραρχία στα αντικείμενα Πτώσης Εδάφους

Σε κάθε κομμάτι του FallingGround έχει προστεθεί RigidBody2D το οποίο προσθέτει φυσική στο αντικείμενο. Άρα προσθέτοντας αυτή την επιλογή έχουμε βαρύτητα στο αντικείμενο για να μπορεί να πέσει. Για να μην πέσει με την έναρξη του παιχνιδιού έχουμε ενεργοποιήσει την επιλογή isKinematic η οποία κάνει παύση των δυνάμεων φυσικής μέχρι να αλλάξει.

Σε περίπτωση που ο χρήστης ακουμπήσει τον Collider του αντικειμένου τότε ενεργοποιείται η Coroutine στον **κώδικα 14**. Σε αυτήν καθυστερούμε για 0.5 δευτερόλεπτα, απενεργοποιούμε το isKinematic και κάνουμε τον Collider του αντικειμένου Trigger ώστε να μην σταματήσει όταν έρθει σε επαφή με κάποιο άλλο αντικείμενο. Στην Update του αντικειμένου έχει προστεθεί ένας έλεγχος ώστε όταν το αντικείμενο φτάσει ενα συγκεκριμένο ύψος να καταστραφεί.

```
IEnumerator FallCo()
    {
        yield return new WaitForSeconds(fallDelay);
        rigidbody.isKinematic = false;
        GetComponent<BoxCollider2D>().isTrigger = true;
        yield return 0;
    }
```

Κώδικας 14 : Coroutine του Εδάφους που πέφτει

```
void Start ()
    {
        player = GameObject.FindGameObjectWithTag("Player").GetComponent<Player>();
        tempLifeCount = player.maxPlayerLifes;
    }


       void Update ()
    {
           if(tempLifeCount > player.currentPlayerLifes)
        {
            Instantiate(FallingGround, SpawnLocation.transform.position,
Quaternion.identity);
            tempLifeCount = player.currentPlayerLifes;
        }
      }


            tempLifeCount = player.currentPlayerLifes;
        }
      }
```

Κώδικας 15 : Instantiation του εδάφους που πέφτει

Στον **κώδικα 15** κατά την έναρξη του παιχνιδιού δηλώνουμε σε μια μεταβλητή tempLifeCount οτι ο χρήστης εχει 5 ζωές, αυτή η δήλωση μπορει να είναι ψευδής αλλά δεν μας δημιουργεί πρόβλημα. Στην Update κάνουμε ένα έλεγχο, αν η μεταβλητη αυτή είναι μεγαλύτερη απο τις ζωές του χρήστη τότε εκτελούμε την εντολή Instantiate, τοποθετούμε το FallingGroundTotal στην τοποθεσία που του έχουμε ορίσει και κάνουμε την μεταβλητή tempLifeCount ίση με τις ζωές του χρήστη. Ο λόγος που δεν μας ενοχλεί είναι οτι στην

χειρότερη περίπτωση θα έχουμε μέγιστο 4 επιπλέον κλώνους του αντικειμένου που δεν είναι αρκετοί ωστε να υπερφορτώσει τον υπολογιστή.

5.3 Αντίπαλοι

Σε αυτό το κεφάλαιο θα αναλύσουμε την λογική των αντιπάλων και αντικειμένων που μπορούν να σκοτώσουν τον χαρακτήρα.

5.3.1 Το Κενό

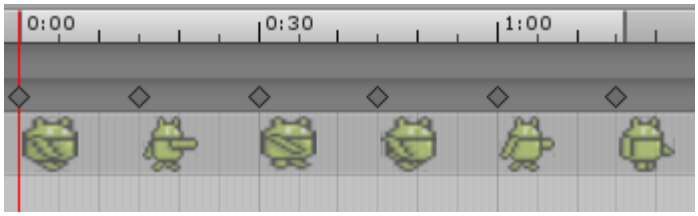
Το κενό τοποθετείται στα κενά που υπάρχουν στην πίστα από τα οποία ο χρήστης πρέπει να πηδήξει. Περιέχει ένα TriggerCollider και στον κώδικα ελέγχουμε αν ο χρήστης εισέλθει στα όρια του τότε αλλάζουμε το Status του σε 0 και εκτελούμε την διεργασία Die() που βρίσκεται στον **κώδικα 30** του χρήστη και θα αναλύσουμε στο **κεφάλαιο 5.5.**

5.3.2 Τα Καρφιά

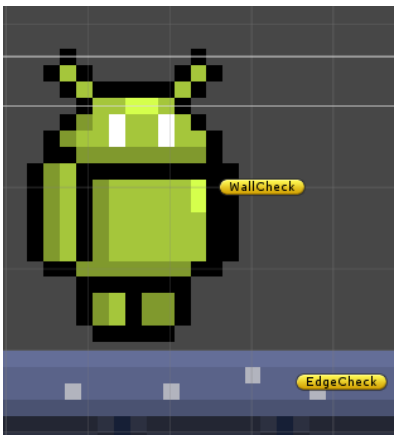
Υπάρχουν 2 ειδών καρφιά, τα περιστρεφόμενα και αυτά που είναι μέσα στους σωλήνες. Χρησιμοποιούν τον ίδιο κώδικα αλλά διαφέρουν στο animation το οποίο χρησιμοποιούν, επίσης τα καρφιά σωλήνα έχουν ένα επιπλέον κώδικα ο οποίος χρησιμοποιείται για να υπάρχει η καθυστέρηση ώστε τα καρφιά να μην βγαίνουν αμέσως μόλις τελειώσει το animation.

5.3.3 Ρομπότ Android

Υπάρχουν 2 ειδών Ρομπότ Android τα πράσινα και τα κόκκινα, η μοναδική διαφορά που έχουν είναι ότι τα κόκκινα χρειάζεται να χτυπηθούν δύο φορές για να σκοτωθούν ενώ τα πράσινα μία. Τα Android έχουν δημιουργηθεί με την χρήση έτοιμων sprite. Με την τοποθέτηση των sprite αυτών στο animation δημιουργήσαμε την κίνηση των μελών του σώματος του Android όπως φαίνεται στην **εικόνα 26**. Έπειτα δημιουργώντας τον **κώδικα 16** και με την χρήση 2 υπό αντικειμένων του Android καταφέρνουμε να έχουμε κίνηση και σε περίπτωση σύγκρουσης με κάποιο αντικείμενο εδάφους ή με την εύρεση κενού στο έδαφος επιστρέφει προς τα πίσω.



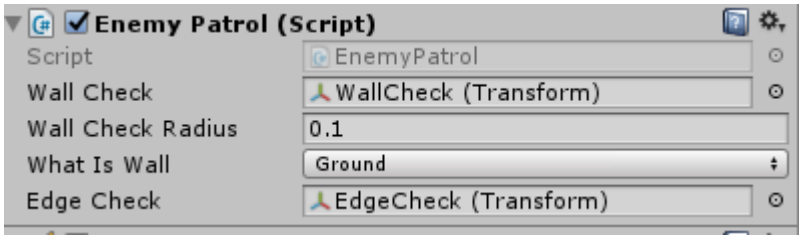
Εικόνα 27 : Animation του Android



Εικόνα 28 : Android στο Unity

```
void FixedUpdate()
    {
        hitWall = Physics2D.OverlapCircle(wallCheck.position, wallCheckRadius,
whatIsWall);
        foundEdge = Physics2D.OverlapCircle(edgeCheck.position, wallCheckRadius,
whatIsWall);

        if (hitWall || !foundEdge)
            movingRight = !movingRight;
    }

    void Update()
    {
        Vector3 currentRotation = Transform.eulerAngles;
            if (movingRight)
            {
                RigidBody2D.velocity = new Vector2(moveSpeed,
RigidBody2D.velocity.y);
                currentRotation.y = 0;
            }
            else
            {
                RigidBody2D.velocity = new Vector2(-moveSpeed,
RigidBody2D.velocity.y);
                currentRotation.y = 180;
            }
Transform.eulerAngles = currentRotation;
      . . . . . .
    }
```

Κώδικας 16 : Περιπολία Android



Εικόνα 29 : Επιθεωρητής του Enemy Patrol

Συνδυάζοντας τον **κώδικα 16** και τις **εικόνες 27 & 29** και την βοήθεια του Unity.docs χρησιμοποιούμε την εντολή

Physics2D.OverlapCircle(point, radius, layerMask) οπού

Point : center of the circle

Radius : Radius of the circle

LayerMask : Filter to check objects only on specific layers

η οποία μας επιστρέφει μεταβλητή bool άρα μεταφράζοντας τον κώδικα αυτό σε λόγια μπορούμε να πούμε ότι κάνουμε ένα έλεγχο για το πότε το αντικείμενο WallCheck το οποίο έχει ακτίνα 0.1 ακουμπήσει πάνω σε αντικείμενο το οποίο έχει layer Ground. Το ίδιο γίνεται και για το κενό. Έπειτα κάνουμε ένα επιπλέον έλεγχο για το πότε είναι αληθή ή ψευδή αυτά και σε περίπτωση που ένα από τα δυο ισχύει τότε αλλάζουμε την φορά που περπατάει μέσω

του κώδικα που ακολουθεί στο Update. Δίνουμε νέα ταχύτητα στο αντικείμενο μας μετακινώντας το σταδιακά προς την κατεύθυνση που έχουμε. Επίσης γυρίζουμε το αντικείμενο κατά 180^{o} ώστε να κοιτάζει προς την άλλη κατεύθυνση.

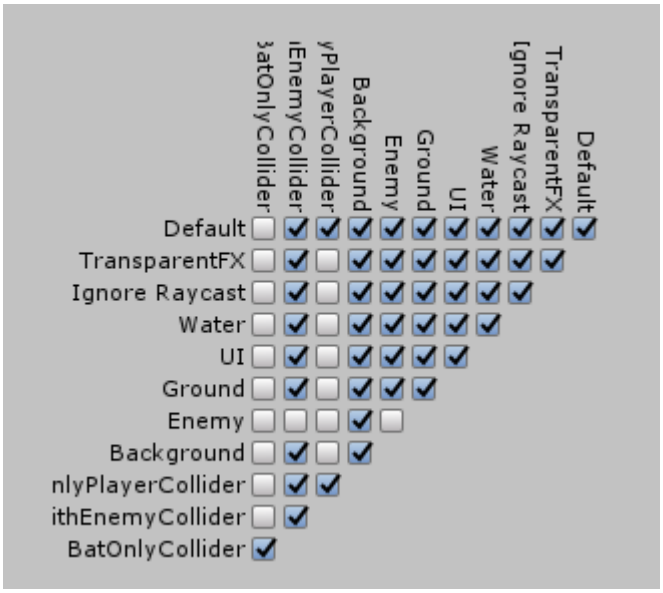
```
void OnCollisionEnter2D(Collision2D collision)
    {
        if (collision.gameObject.CompareTag("Player"))
        {
            bool killOnce = true;
            foreach (ContactPoint2D point in collision.contacts)
            {
                if(Player.playerStatus != 2)
                {
                    if (point.normal.y <= -0.5f)
                     {
                        player.BounceOnEnemyKilled();
                        if (this.tag == "EnemyDouble")
                        {
                            Renderer.color = new Color(1f, 1f, 1f, 1f);
                            if (timesHit >= 1)
                            {
                              ScoreManager.AddPoints(175);
                              source.Play();
                              this.GetComponent<BoxCollider2D>().enabled = false;
                              this.GetComponent<SpriteRenderer>().enabled = false;
                              enemyKilled = true;
                             }
                            timesHit++;
                        }
                        else
                        {
                            ScoreManager.AddPoints(100);
                            source.Play();
                            this.GetComponent<BoxCollider2D>().enabled = false;
                            this.GetComponent<SpriteRenderer>().enabled = false;
                            enemyKilled = true;
                        }
                    }
                    else
                    {
                        if (killOnce)
                        {
                            player.Die();
                            killOnce = false;
                        }
                    }
                }
                else    // If Player.status is 2 --> Player is invincible
                {
                    if (this.tag == "EnemyDouble")
                        ScoreManager.AddPoints(175);
                    else
                        ScoreManager.AddPoints(100);
                    source.Play();
                    this.GetComponent<BoxCollider2D>().enabled = false;
                    this.GetComponent<SpriteRenderer>().enabled = false;
                    enemyKilled = true;
                }
            }
            killOnce = true;
        }
    }
```

Κώδικας 17 : Θάνατος και Επίθεση του Android

Σ οποιήσει στα κουτια με την οιαφορα οτι τωρα θελουμε να γινεται κατι οταν χτυπαμε απο πανω. Στην περίπτωση που ο χρήστης έχει Status 2 τότε με το άγγιγμα σκοτώνεται ο αντίπαλος, παίρνουμε πόντους και κρύβουμε το αντικείμενο του αντιπάλου έως ότου τελειώσει ο ήχος για να μπορέσουμε να το καταστρέψουμε. Στην περίπτωση που ο χρήστης δεν είναι στο Status 2 τότε αν ο χαρακτήρας χτυπήσει τον αντίπαλο από πάνω τότε αν ο αντίπαλος έχει ετικέτα "EnemyDouble" αλλάζουμε το χρώμα του σε πράσινο και στην δεύτερη φορά τον σκοτώνουμε, αν πάλι είναι πράσινος τότε σκοτώνεται με την πρώτη φορά. Σε περίπτωση που η επαφή δεν είναι από πάνω τότε ενεργοποιούμε την διεργασία Die() του Player. Όπως με τα κουτιά έτσι και εδώ λόγω ότι υπάρχουν 2 σημεία επαφής χρησιμοποιούμε μια μεταβλητή bool για την αποφυγή του προβλήματος.

5.3.4 Νυχτερίδες

Οι νυχτερίδες χρησιμοποιούν ακριβώς τον ίδιο κώδικα με τα Android με την μόνη διαφορά ότι έχουν τοποθετηθεί 2 Colliders οι οποίοι είναι ρυθμισμένοι έτσι ώστε να μπορούν να αντιδρούν μόνο με τις νυχτερίδες και είναι τοποθετημένοι δεξιά και αριστερά από την νυχτερίδα για να ρυθμίσουμε την διαδρομή που θα κάνει περιπολία η νυχτερίδα. Όπως και στο Android με την χρήση sprites εχει δημιουργηθεί ενα animation το οποίο μετακινεί τα φτερά της νυχτερίδας για να δώσει την ψευδαίσθηση της κίνησης.

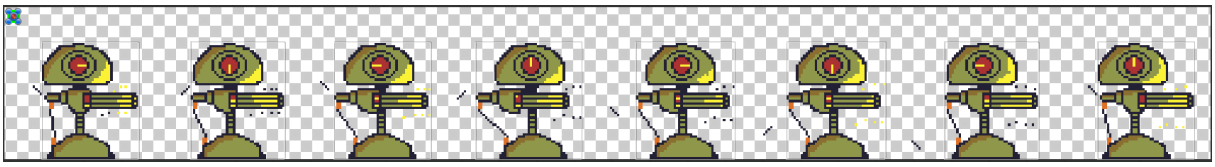


Εικόνα 30 : Ρυθμίσεις Physics2D

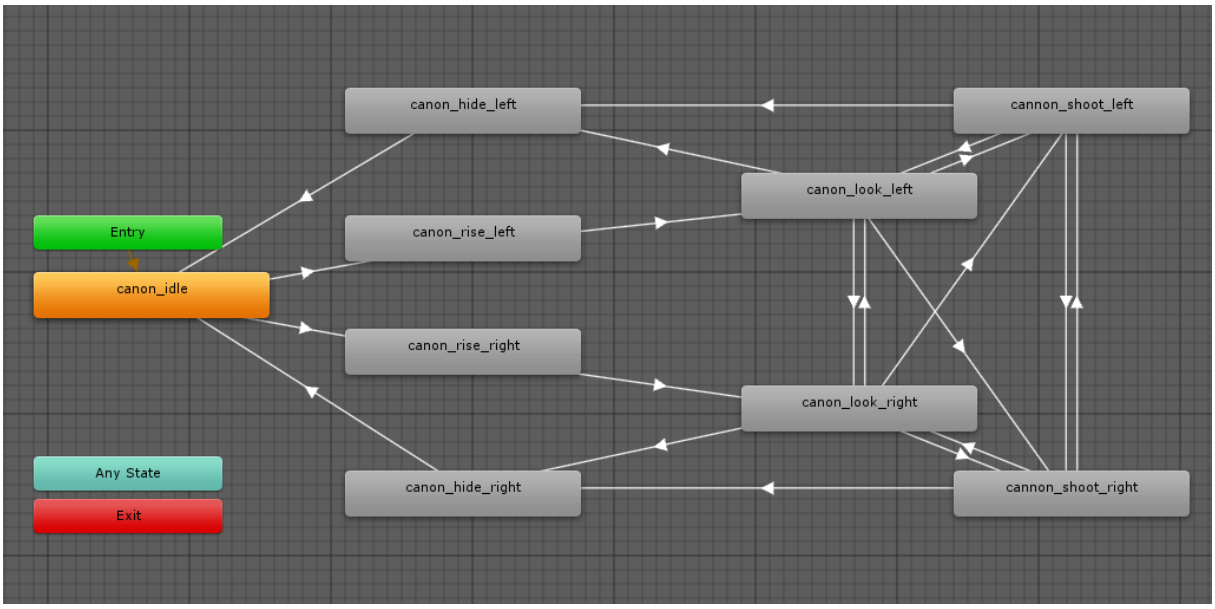
5.3.5 Κανόνι

Το κανόνι είναι ο ισχυρότερος αντίπαλος του παιχνιδιού και ο πιο περίπλοκος. Αποτελείται από τον CannonHolder που κρατάει όλα τα αντικείμενα σε ένα και χρησιμοποιείται για την χρήση των animations, μετά είναι το ίδιο το κανόνι το οποίο έχει τον κώδικα και αυτό περιέχει 3 αντικείμενα, 2 colliders για να ελέγχουμε την είσοδο του

χαρακτήρα μέσα στα όρια και το σημείο από όπου φεύγουν οι σφαίρες. Ξεκινώντας από τα απλά πράγματα, για να μπορέσει να σκοτωθεί το κανόνι χρειάζεται να χτυπηθεί 3 φορές από πάνω και χρησιμοποιεί τον ίδιο κώδικα που χρησιμοποιούμε και για τα Android και τις νυχτερίδες. Η σφαίρα χρησιμοποιεί ένα timer ο οποίος καταστρέφει το αντικείμενο μετά από 3 δευτερόλεπτα. Αν η σφαίρα έρθει σε επαφή με τον χρήστη τότε εκτελείται η διεργασία Die() του Player. Με την χρήση της **εικόνας 30** έχω δημιουργήσει 8 animation τα οποία είναι χωρισμένα σε shoot, hide, look, rise για δεξιά και αριστερά.



Εικόνα 31 : Sprites Κανονιού



Εικόνα 32 : Animator Κανονιού

Χρησιμοποιώντας τον animator δημιούργησα την λογική των animations που φαίνεται στην **εικόνα 31** με 3 μεταβλητές bool οι οποίες είναι οι **risen, lookRight** και **inRange**. Τα βέλη που φαίνονται στην **εικόνα 32** ονομάζονται AnimatorTransitions όπου χρησιμοποιώντας τις μεταβλητές που αναφέραμε μπορούμε να πηγαίνουμε από το ένα animation στο άλλο. Για παράδειγμα στην **εικόνα 32** για να μπορέσουμε να πάμε από την κατάσταση cannon_idle στο cannon_rise_right χρειάζεται να είναι αληθής οι μεταβλητές risen & lookRight, λόγω ότι οι παράμετροι που χρειάζονται για να περάσουμε στην επόμενη κατάσταση παραμένουν οι ίδιες μόλις ολοκληρωθεί το animation του cannon_rise_right πηγαίνουμε στην κατάσταση cannon_look_right όπου παραμένουμε έως ότου γίνει κάποια αλλαγή στις μεταβλητές.

```
animator.SetBool("risen", risen);
animator.SetBool("lookRight", lookRight);
 animator.SetBool("inRange", inRange);
```

Κώδικας 18 : CannonAI, ορισμός Animator

Στον **κώδικα 18** κάνουμε την σύνδεση αυτών των μεταβλητών με 3 μεταβλητές που θα μπορούμε να χρησιμοποιούμε μέσω του κώδικα για την πιο εύκολη χρήση τους.

Όπως προαναφέραμε το κανόνι έχει σαν υποαντικείμενα 2 TriggerColliders οι οποίοι ελέγχουν αν ο χρήστης έχει μπει στα όρια της επίθεσης.

```
void OnTriggerStay2D(Collider2D col)
    {
        if (col.CompareTag("Player"))
        {
            cannonAI.inRange = true;
            cannonAI.Attack(true);
        }
    }

    void OnTriggerExit2D(Collider2D col)
    {
        if (col.CompareTag("Player"))
            cannonAI.inRange = false;
    }
```

Κώδικας 19 : Όρια Ενεργοποιήσης Κανονιού

Σε περίπτωση που ο χαρακτήρας μπει σε αυτά τα όρια αλλάζουμε την μεταβλητή inRange του **κώδικα 18** και εκτελούμε την διεργασία Attack του **κώδικα 20.**

```
void Update()
    {
       ....
       RangeCheck();
        if(target.position.x > transform.position.x)
            lookRight = true;
        else
            lookRight = false;
    }
    void RangeCheck()
     {
        distance = Vector3.Distance(transform.position, target.position);
            if(distance < riseRange)
                risen = true;
            if(distance > riseRange)
                risen = false;
    }
    public void Attack(bool attackPermit)
    {
        bulletTimer += Time.deltaTime;
        if(bulletTimer >= shootInterval)
        {
            Vector2 direction = target.transform.position - transform.position;
            direction.Normalize();
            GameObject bulletClone;
            bulletClone = Instantiate(bullet, shootPoint.transform.position,
shootPoint.transform.rotation) as GameObject;
            bulletClone.GetComponent<Rigidbody2D>().velocity = direction *
bulletSpeed;
            bulletTimer = 0;
        }
    }
```

Κώδικας 20 : Λογική Κανονιού

Στον **κώδικα 20** πρώτα κάνουμε τον έλεγχο RangeCheck ο οποίος ελέγχει την απόσταση του χαρακτήρα από το κανόνι και αν είμαστε μέσα στις 4 μονάδες μέτρησης που έχουμε ορίσει τότε αλλάζουμε την μεταβλητή risen σε αληθή. Έπειτα κάνουμε ξανά έλεγχο την τοποθεσία του χαρακτήρα (target) ως προς την τοποθεσία του κανονιού για να δούμε προς ποια μεριά θα κοιτάζει το κανόνι. Τέλος μέσω του **κώδικα 19** ενεργοποιείται η διεργασία Attack η οποία κάθε 1 δευτερόλεπτο κάνει Instantiate μια σφαίρα με σημείο εκκίνησης τις συντεταγμένες του shootPoint και της δίνει ταχύτητα ίση με την κατεύθυνση που βρίσκεται ο χρήστης επι μια καθορισμένη μεταβλητή bulletSpeed για να είναι λίγο πιο γρήγορη η ταχύτητα της σφαίρας. Έπειτα μηδενίζεται ο μετρητής και η διαδικασία επαναλαμβάνετε.

5.4 Διαχείριση Παιχνιδιού

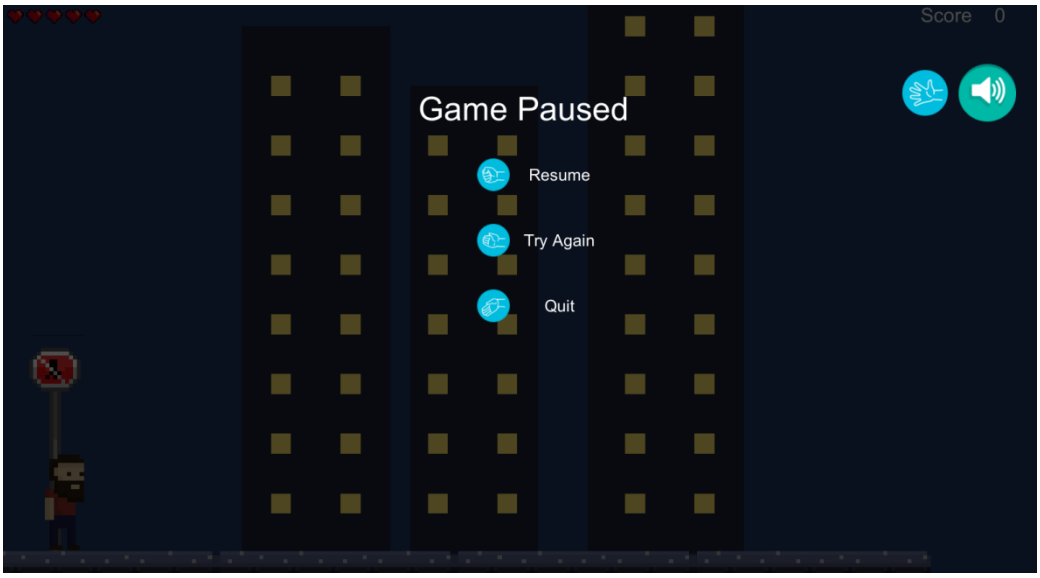
Σε αυτό το κεφάλαιο θα αναλύσουμε την λογική του GameManager ο οποίος έχει 4 διαφορετικά scripts για την σωστή λειτουργία του κάθε επιπέδου.

5.4.1 Μενού Παύσης

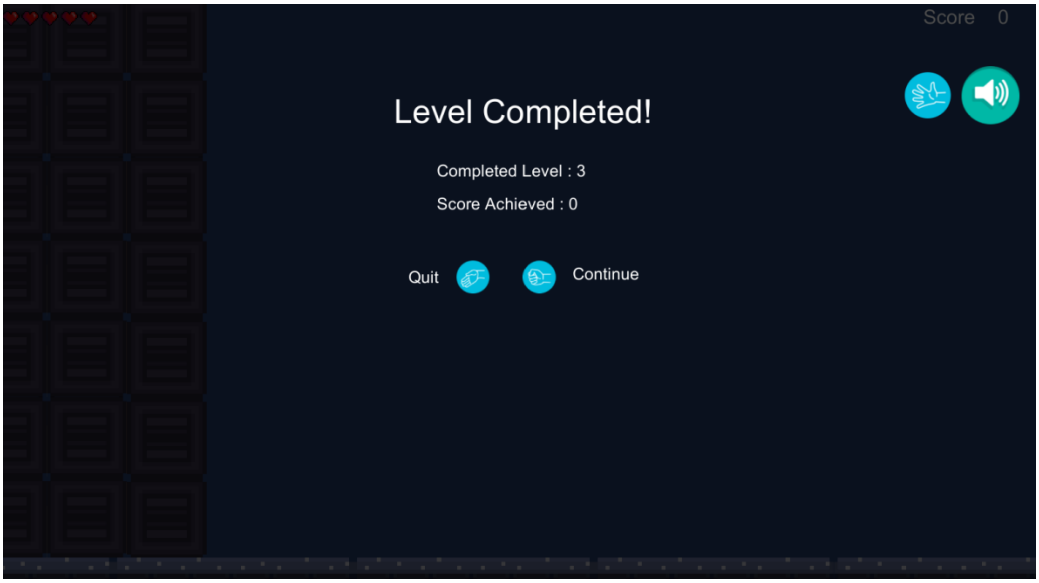
Το μενού παύσης χρησιμεύει για την εμφάνιση ενός μενού κατά την διάρκεια που παίζει ο χρήστης και θέλει να κάνει μια παύση στο παιχνίδι. Ενώ παίζει ο χρήστης χρησιμοποιώντας την χειρονομία WaveIn, μηδενίζουμε την κλίμακα με την οποία περνάει ο χρόνος με αποτέλεσμα όλα τα αντικείμενα τις σκηνής να σταματήσουν να κινούνται. Αλλάζουμε τις μεταβλητές showMenu & showPauseMenu bool σε αληθή με αποτέλεσμα να εμφανιστούν στην σκηνή τα γραφικά του GUI. Εμφανίζονται διάφορα κουμπιά τα οποία χρησιμεύουν για την εκκίνηση διεργασιών με την χρήση χειρονομιών που εμφανίζονται δίπλα στα κουμπιά :

- **TryAgain()** : ξανακάνει την κλίμακα χρόνου ίση με 1, καταστρέφει το instance του Hub και εκκινεί το επίπεδο το οποίο βρισκόμαστε από την αρχή.
- **LoadNextLevel() :** ξανακάνει την κλίμακα χρόνου ίση με 1, καταστρέφει το instance του Hub και εκκινεί το επόμενο επίπεδο εκτός και αν δεν υπάρχει άλλο που μας επιστρέφει στο αρχικό μενού. Εμφανίζεται μόνο αν ο παίκτης τερματίσει το επίπεδο.
- **GoToMainMenu() :** ξανακάνει την κλίμακα χρόνου ίση με 1, καταστρέφει το instance του Hub, κάνουμε reset τις μεταβλητές της βαθμολογίας και ζωών του παίκτη ώστε σε έναρξη νέου παιχνιδιού από την σκηνή Επιλογή Επιπέδου να ξεκινάει ο παίκτης από την αρχή και μας επιστρέφει στο αρχικό μενού.
- **ResumeGame() :** ξανακάνει την κλίμακα χρόνου ίση με 1, αλλάζει τις μεταβλητές bool σε ψευδή και επιστρέφουμε στο παιχνίδι από το σημείο που είχαμε μείνει.

Σε περίπτωση που ο χρήστης τερματίσει το επίπεδο εμφανίζονται η βαθμολογία και το επίπεδο που τερμάτισε. Στο πάνω δεξιά μέρος του μενού εμφανίζεται και η επιλογή να κάνουμε σίγαση και απενεργοποίηση της σίγασης των ήχων του παιχνιδιού. Τα κουμπιά αυτά φαίνονται στις **εικόνες 33 & 34.**



Εικόνα 33 : Παυση Παιχνιδιού



Εικόνα 34 : Τερματισμός Επιπέδου

5.4.2 Προβολή Ζωών Παίκτη

Έχοντας δημιουργήσει 5 εικονίδια με 5,4,3,2,1 καρδιές και χρησιμοποιώντας τις ζωές που έχει ο χρήστης εμφανίζουμε στην οθόνη το κατάλληλο εικονίδιο με τις αντίστοιχες καρδιές. Το αντικείμενο HeartUI στον **κώδικα 21** είναι ένα αντικείμενο του καμβά το οποίο εμφανίζεται πάντα στην οθόνη. *HUD = Heads Up Display

```
HeartUI.sprite = HeartSprites[player.currentPlayerLifes];
```

Κώδικας 21 : Προβολή Ζωών Παίκτη

5.4.3 Διαχειριστής Βαθμολογίας

Σε αυτό τον κώδικα διατηρούμε και εμφανίζουμε την βαθμολογία στην οθόνη. Κάνουμε ένα έλεγχο να δούμε αν ο χρήστης έχει ήδη κάποια βαθμολογία αποθηκευμένη από προηγούμενο επίπεδο για να εμφανίσουμε αυτή αλλιώς ξεκινάμε με μηδενική βαθμολογία. Έπειτα τοποθετούμε την βαθμολογία σε ένα αντικείμενο του καμβά που έχουμε τοποθετήσει ο οποίος εμφανίζεται πάντα στην οθόνη. Επιπλέον έχει δημιουργηθεί μια στατική διεργασία για την αύξηση της βαθμολογίας από οποιοδήποτε μέρος χωρίς την αρχικοποίηση και δήλωση των μεταβλητών αυτών κάθε φορά που χρειάζεται σε κάποιο νέο κώδικα.

5.4.4 Διαχειριστής Επιπέδου

Σε αυτό τον κώδικα διατηρούμε το τωρινό CheckPoint για να μπορούμε να κάνουμε respawn τον χρήστη σε περίπτωση θανάτου και κάνουμε την διαδικασία του Respawn.

```
public IEnumerator RespawnPlayerDelayCo()
    {
        Instantiate(deathParticles, player.transform.position,
player.transform.rotation);
        playerObject.SetActive(false);

        yield return new WaitForSeconds(respawnDelay);

        playerObject.SetActive(true);
        player.transform.position = currentCheckpoint.transform.position;
    }
```

Κώδικας 22 : Respawn Χαρακτήρα

Η διαδικασία αυτή ξεκινάει με την απενεργοποίηση του χαρακτήρα και το Instantiate των DeathParticles στο σημείο που πέθανε ο χρήστης. Τα DeathParticles είναι ένα σύστημα σωματιδίων το οποίο έχουμε δημιουργήσει μέσα στο Unity και με τις κατάλληλες ρυθμίσεις φαίνεται σαν να πεθαίνει ο χαρακτήρας. Έπειτα περιμένουμε 0.5 δευτερόλεπτα και ενεργοποιούμε τον χαρακτήρα και τον μεταφέρουμε στην τοποθεσία του τωρινού CheckPoint. Τα DeathParticles έχουν και ένα κώδικα ο οποίος κάνει τα Particles να καταστρέφονται όταν τελειώσει το animation τους.

5.4.5 Όριο Οπίσθιας Κίνησης

Έχουμε δημιουργήσει ένα Collider ο οποίος ακολουθεί και διατηρεί μια μέγιστη απόσταση των 5 μονάδων από τον χαρακτήρα μας.

```
void RangeCheck()
    {
        playersX = targetPlayer.position.x;
        collidersX = transform.position.x;
        distance = playersX - collidersX;

        if (distance > 5 || distance < 0)
            transform.position = new Vector3(targetPlayer.position.x - 5,
targetPlayer.position.y, targetPlayer.position.z);
    }
```

Κώδικας 23 : Όριο Οπίσθιας Κίνησης

Όπως φαίνεται και στον **κώδικα 23** , διαχέουμε συνεχώς τον άξονα των X του χαρακτήρα και του Collider, σε περίπτωση που η απόσταση τους είναι μεγαλύτερη από 5 μονάδες τότε αλλάζουμε τις συντεταγμένες του Collider ώστε να είναι ίση με αυτή του χαρακτήρα μας μείον 5 μονάδες για τον άξονα των X. Ο λόγος που έχουμε βάλει στον έλεγχο αν η απόσταση είναι μικρότερη του μηδενός είναι ότι αν ο χαρακτήρας μας πεθάνει επιστρέφει στο κοντινότερο checkPoint με αποτέλεσμα να εμφανιστεί ο Collider μπροστά του, οπότε για να μην μπλοκάρει την διαδρομή του χαρακτήρα τον επανατοποθετούμε πίσω μας.

5.4.6 Κάμερα

Στην κάμερα έχει τοποθετηθεί κώδικας για να ακολουθεί τον χαρακτήρα με μια ελαφριά καθυστέρηση για να υπάρχει η αίσθηση ότι η κάμερα ακολουθάει τον χαρακτήρα αντί να είναι κολλημένη πάνω του. Επίσης υπάρχει ένα κομμάτι κώδικα το οποίο βάζει όρια μέχρι ποιες συντεταγμένες μπορεί να φτάσει.

```
float posX = Mathf.SmoothDamp(
                    transform.position.x,
                    player.transform.position.x,
                    ref velocity.x,
                    smoothTimeX);

float posY = Mathf.SmoothDamp(transform.position.y, player.transform.position.y,
ref velocity.y, smoothTimeY);

transform.position = new Vector3(posX + 0.5f, posY - 0.1f, transform.position.z);
```

Κώδικας 24 : Κάμερα Ακολουθάει τον Χαρακτήρα

Στον **κώδικα 24** χρησιμοποιούμε την
SmoothDamp(float current, float target, ref float currentVelocity, float smoothTime)

Current : τωρινή τοποθεσία

Target : τοποθεσία που θέλουμε να πάμε

currentVel : τωρινή ταχύτητα

smoothTime : στο περίπου ο χρόνος που θα χρειαστεί για να φτάσουμε τις συντεταγμένες που θέλουμε.

Έπειτα με την χρήση των posX & posY μπορούμε να αλλάξουμε τις συντεταγμένες της κάμερας. Ο λόγος που προσθέτουμε 0.5f στον άξονα των X είναι για να έχουμε τον χαρακτήρα στην αριστερή πλευρά της οθόνης αντί να είναι στο κέντρο. Το ίδιο κάνουμε και για τον άξονα Y ώστε να βρισκόμαστε λίγο πιο χαμηλά από το κέντρο.

Στον **κώδικα 25** παρακάτω χρησιμοποιούμε την Clamp(float value, float min, float max); η οποία τοποθετεί μια τιμή μεταξύ ενός μέγιστου και ενός ελάχιστου ορίου. Σε αυτή την περίπτωση το ελάχιστο όριο θα είναι ένας Collider στην αρχή της πίστας ή ο Collider που ακολουθεί τον χαρακτήρα. Το μέγιστο όριο είναι το τέλος του επιπέδου.

```
if(minCameraPos.x > backWardsMovingCollider.position.x)
        {
            if(player.transform.position.y < -1)
            {
                transform.position = new Vector3(
                    Mathf.Clamp(transform.position.x, minCameraPos.x,
maxCameraPos.x),
                    Mathf.Clamp(transform.position.y, minCameraPos.y,
maxCameraPos.y),
                    Mathf.Clamp(transform.position.z, minCameraPos.z,
maxCameraPos.z));
            }
            else
            {
                transform.position = new Vector3(
                    Mathf.Clamp(transform.position.x, minCameraPos.x,
maxCameraPos.x),
                    Mathf.Clamp(transform.position.y, 1, maxCameraPos.y),
                    Mathf.Clamp(transform.position.z, minCameraPos.z,
maxCameraPos.z));
            }

        }
        else
        {
            if (player.transform.position.y < -1)
            {
                transform.position = new Vector3(
                    Mathf.Clamp(transform.position.x,
backWardsMovingCollider.position.x + 2.8f, maxCameraPos.x),
                    Mathf.Clamp(transform.position.y, minCameraPos.y,
maxCameraPos.y),
                    Mathf.Clamp(transform.position.z, minCameraPos.z,
maxCameraPos.z));
            }
            else
            {
                transform.position = new Vector3(
                    Mathf.Clamp(transform.position.x,
backWardsMovingCollider.position.x + 2.8f, maxCameraPos.x),
                    Mathf.Clamp(transform.position.y, 1, maxCameraPos.y),
                    Mathf.Clamp(transform.position.z, minCameraPos.z,
maxCameraPos.z));
            }
        }
```

Κώδικας 25 : Όρια Κάμερας

5.5 Χαρακτήρας

Σε αυτό το κεφάλαιο θα αναλύσουμε την λογική του χαρακτήρα, για το πως κινείται και όλες τις διεργασίες που περιέχει.

5.5.1 Κώδικας Παίκτη

Αυτός ο κώδικας ασχολείται με το Status του χρήστη, τον θάνατο του και επίσης περιέχει την αναπήδηση όταν ο χρήστης σκοτώσει έναν αντίπαλο.

```
playerStatus = 0;

if (currentLevel != 1)
            currentPlayerLifes = PlayerPrefs.GetInt("PlayerLifes");
else
            currentPlayerLifes = maxPlayerLifes;
```

Κώδικας 26 : Παίκτης , Start ( )

Στον **κώδικα 26** ξεκινάμε τον χαρακτήρα από το 0 Status που είναι η βασική μορφή και επίσης κάνουμε ένα έλεγχο για το επίπεδο που βρισκόμαστε για να δώσουμε στον χρήστη τις ζωές που του έχουν απομείνει αν δεν βρίσκεται στο $1^{ο}$ επίπεδο.

```
       void Update ()
    {
        if (currentPlayerLifes > maxPlayerLifes)
            currentPlayerLifes = maxPlayerLifes;
        if (currentPlayerLifes <= 0)
            GameOver();

        Status();
    }
```

Κώδικας 27 : Παίκτης , Update ( )

Στον **κώδικα 27** ελέγχουμε συνεχώς για τις ζωές που έχει ο χρήστης ώστε να μην μπορεί να έχει πάνω από το όριο και σε περίπτωση που φτάσει στο 0 να τελειώνει το παιχνίδι. Επίσης εκτελούμε την διεργασία Status η οποία ελέγχει το Status του χαρακτήρα. Όλη η διαδικασία με τις διαφορές ανάλογα του Status του χαρακτήρα και την επίδραση τους στους αντίπαλους έχει τοποθετηθεί στον κώδικα των αντιπάλων οπότε εδώ τοποθετείται ο κώδικας για τις εμφανισιακές διαφορές για να ξεχωρίζει ο χρήστης σε τι κατάσταση βρίσκεται ο χαρακτήρας.

Όπως φαίνεται στον **κώδικα 28** από κάτω ανάλογα το Status αλλάζουμε το μέγεθος και το χρώμα του χαρακτήρα. Το κανονικό μέγεθος του χαρακτήρα είναι το 0.7 * 0.7 units και το χρώμα είναι αυτά του αρχικού sprite. Σε περίπτωση που ο χρήστης πάρει το Milk τότε το Status του αλλάζει σε 1 οπότε αλλάζουμε το μέγεθος του σε 1 * 1 units με αποτέλεσμα ο χαρακτήρας να είναι μεγαλύτερος. Το Status 2 έχει την διάφορα ότι δεν είναι μόνιμο άρα

έχουμε τοποθετήσει ένα μετρητή ο οποίος στα 9 δευτερόλεπτα ειδοποιεί τον χρήστη ότι τελειώνει η επίδραση και στα 10 δευτερόλεπτα επιστρέφει τον χρήστη στο Status 0. Επίσης στο Status 2 ο χαρακτήρας έχει το φυσιολογικό μέγεθος αλλά του έχουμε αλλάξει το χρώμα σε πράσινο για να ξεχωρίζει.

```
void Status()
    {
        if(playerStatus == 0)
        {
            if(playerController.moveHorizontal == 1)
                this.transform.localScale = new Vector3(0.7f, 0.7f, 0.7f);
            else if(playerController.moveHorizontal == -1)
                this.transform.localScale = new Vector3(-0.7f, 0.7f, 0.7f);

            playerGraphics.GetComponent<SpriteRenderer>().color = normalColor;
        }

        if(playerStatus == 1)
        {
            if (playerController.moveHorizontal == 1)
                this.transform.localScale = new Vector3(1f, 1f, 1f);
            else if (playerController.moveHorizontal == -1)
                this.transform.localScale = new Vector3(-1f, 1f, 1f);

            playerGraphics.GetComponent<SpriteRenderer>().color = normalColor;
        }

        if(playerStatus == 2)
        {
            invincibilityTimer += Time.deltaTime;

            if(invincibilityTimer > 9)
            {
                AudioSource source = GetComponent<AudioSource>();
                source.clip = invincibilityEndsAudio;
                source.volume = 0.5f;
                source.Play();
            }

            if(invincibilityTimer >= 10)
            {
                invincibilityTimer = 0;
                playerStatus = 0;
            }

            if (playerController.moveHorizontal == 1)
                this.transform.localScale = new Vector3(0.7f, 0.7f, 0.7f);
            else if (playerController.moveHorizontal == -1)
                this.transform.localScale = new Vector3(-0.7f, 0.7f, 0.7f);

            playerGraphics.GetComponent<SpriteRenderer>().color = alcoholColor;
        }
    }
```

Κώδικας 28 : Κατάσταση Παίκτη

```
public void AddLife()
    {
        currentPlayerLifes += 1;
    }
```

Κώδικας 29 : Προσθήκη Ζωής

Στον **κώδικα 29** προσθέτουμε μια ζωή στον χρήστη, η διεργασία αυτή χρησιμοποιείται στον **κώδικα 11** της Pizza.

```
public void Die()
    {
        if (thalmicMyo.isPaired)
            thalmicMyo.Vibrate(Thalmic.Myo.VibrationType.Short);

        if (playerStatus <= 0)
        {
            playerStatus = 0;
            levelManager.RespawnPlayer();
            currentPlayerLifes -= 1;
        }
        else if(playerStatus == 1)
        {
            playerStatus = 0;
            StartCoroutine(PlayerHitWhileBigCo());
        }
    }
```

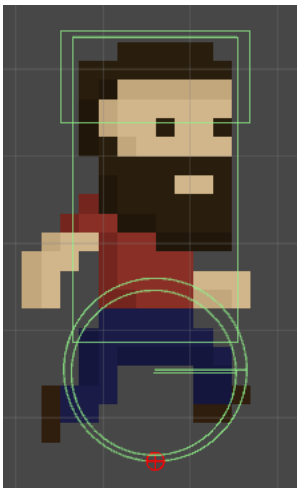
Κώδικας 30 : Θάνατος Παίκτη

Ο **κώδικας 30** εκτελείται από όλους τους αντίπαλους όταν χτυπάνε τον χρήστη, σε αυτόν ελεούμε το Status του χρήστη και ανάλογα εκτελούμε τις αντίστοιχες εντολές. Για Status 0 εκτελούμε την διεργασία του **κώδικα 22** και αφαιρούμε μια ζωή από τον χρήστη. Σε περίπτωση που ο χρήστης έχει Status 1 τότε εκτελούμε την Coroutine του **κώδικα 31.**

```
public IEnumerator PlayerHitWhileBigCo()
    {
        this.GetComponent<CircleCollider2D>().enabled = false;
        this.GetComponent<BoxCollider2D>().enabled = false;
        playerGraphics.GetComponent<Animation>().Play("player_damaged");

        yield return new WaitForSeconds(1);

        this.GetComponent<CircleCollider2D>().enabled = true;
        this.GetComponent<BoxCollider2D>().enabled = true;
    }
```

Κώδικας 31 : Χτύπημα Παίκτη σε Κατάσταση 1

Ο χαρακτήρας αποτελείται από 4 Colliders. Οι 2 από αυτούς αντιδρούν σε χτύπημα του αντίπαλου ενώ οι άλλοι είναι για να μπορεί ο χαρακτήρας να συνεχίσει να κινείται έπειτα από την διαδικασία του **κώδικα 31.** Οπότε συνδυάζοντας τον **κώδικα 31** και την **εικόνα 34,** εκτελείται η παρακάτω διαδικασία απενεργοποιούμε τους 2 collider που αντιδρούν με τους αντιπάλους, εκτελούμε το animation το οποίο κάνει τον χαρακτήρα μας να κάνει ένα γρήγορο fade_out, fade_in μερικές φορές για 1 δευτερόλεπτο περίπου και έπειτα ενεργοποιούμε ξανά τους Colliders ώστε να μπορεί ο χρήστης να χτυπηθεί από τους αντίπαλους.



Εικόνα 35 : Χαρακτήρας

```
public void GameOver()
    {
        PlayerPrefs.SetInt("GameOverScore", ScoreManager.score);
        PlayerPrefs.SetInt("PlayerScore", 0);
        PlayerPrefs.SetInt("PlayerLifes", 5);
        ThalmicHub.Destroy(ThalmicHub.instance);
        SceneManager.LoadScene("MainMenu");
    }
```

Κώδικας 32 : Τέλος Παιχνιδιού

Η διεργασία GameOver έχει σαν σκοπό να αποθήκευση την βαθμολογία που πέτυχε ο χαρακτήρας ώστε να τοποθετηθεί στην λίστα βαθμολογίων, να κάνει reset τις τιμές της βαθμολογίας εντός παιχνιδιού και τις ζωές και έπειτα να επιστρέψει τον χρήστη στο βασικό menu.

```
public void BounceOnEnemyKilled()
    {
        Vector2 velocity = rb2D.velocity;
        velocity.y = jumpVelocity;
        rb2D.velocity = velocity;
    }
```

Κώδικας 33 : Αναπήδηση στον θάνατο Αντιπάλου

Η διεργασία BounceOnEnemyKilled έχει σαν σκοπό να σπρώχνει τον χαρακτήρα στον αέρα ώστε να μην πέφτει κατευθείαν ο χαρακτήρας στο πάτωμα όταν σκοτώσει ένα αντίπαλο. Η διαδικασία αυτή γίνεται με την χρήση του **κώδικα 33** ο οποίος παίρνει την ταχύτητα του χαρακτήρα και την αλλάζουμε σε μια σταθερά που έχουμε καθορίσει και την ξαναδίνουμε στον χαρακτήρα με αποτέλεσμα να πετάγεται ο χαρακτήρας μας στον αέρα.

5.5.2 Διαχειριστής Κίνησης Παίκτη

Αυτός ο κώδικας χρησιμοποιείται για την κίνηση και τα άλματα του χαρακτήρα.

```
        if (Player.playerStatus != 1)
        {
            movingRightScale = new Vector3(0.7f, 0.7f, 1f);
            movingLeftScale = new Vector3(-0.7f, 0.7f, 1f);
        }
        else
        {
            movingRightScale = new Vector3(1f, 1f, 1f);
            movingLeftScale = new Vector3(-1f, 1f, 1f);
        }

        float x = myo.transform.rotation.eulerAngles.x;

        if ((20 < x && x < 180) || (Input.GetAxisRaw("Horizontal") == 1))
        {
            moveHorizontal = 1;
            transform.localScale = movingRightScale;
            playerGraphics.GetComponent<Animator>().SetBool("isMoving", true);
        }
        else if ((180 < x && x < 345) || (Input.GetAxisRaw("Horizontal") == -1))
        {
            moveHorizontal = -1;
            transform.localScale = movingLeftScale;
            playerGraphics.GetComponent<Animator>().SetBool("isMoving", true);
        }
        else
        {
            playerGraphics.GetComponent<Animator>().SetBool("isMoving", false);
            moveHorizontal = 0;
        }

        Vector2 moveDir = new Vector2(moveHorizontal * moveSpeed, rb2D.velocity.y);
             rb2D.velocity = moveDir;
```

Κώδικας 34 : Κίνηση Παίκτη

Ανάλογα την φορά που έχει ο χρήστης έχουμε και τον άξονα τον X αντεστραμμένο ώστε ο χαρακτήρας να κοιτάζει προς την μεριά που πρέπει. Επίσης ανάλογα το Status του αλλάζει και η τιμή των αξόνων X και Y. Έπειτα παίρνουμε την τιμή του **myo.transform.rotation.eulerAngles.x** η οποία είναι το pitch που φαίνεται στην **εικόνα 10.** Έπειτα κάνουμε ένα έλεγχο για τα όρια μέσα στα οποία βρίσκεται η τιμή αυτή και ανάλογα διαλέγουμε την φορά που κοιτάζει και κινείται ο χρήστης και εκτελούμε το animation της κίνησης, αν ο χρήστης είναι στην νεκρή περιοχή των τιμών που έχουμε βάλει στον έλεγχο σταματάμε το animation και μηδενίζουμε την τιμή κίνησης. Έπειτα δίνουμε την κίνηση στον

χαρακτήρα μας με την χρήση του rigidbody2D το οποίο όπως έχουμε εξηγήσει δίνει την δυνατότητα να προσθέσουμε δυνάμεις φυσικής στο αντικείμενο μας.

```
if (thalmicMyo.isPaired)
        {
        if ((Input.GetKeyDown(KeyCode.Space) || thalmicMyo.accelerometer.x < 0.2f)
&& isGrounded && jumpReleased)
            {
                wantsToJump = true;
                jumpReleased = false;
            }
        }

        if(thalmicMyo.accelerometer.x > 0.2f)
        {
            jumpReleased = true;
        }
```

Κώδικας 35 : Άλμα Παίκτη, κομμάτι 1

Για να μπορέσει ο χρήστης να πηδήξει πρέπει να περιστρέψει το χέρι του προς τα δεξιά περίπου 90^{o}. Με την χρήση της εντολής **thalmicMyo.accelerometer.x** κάνουμε έλεγχο του Myo με βάση την δύναμη της βαρύτητας, σε περίπτωση που φτάσει την τιμή που θέλουμε τότε αλλάζουμε τις τιμές των μεταβλητών wantsToJump σε αληθή για να γίνει το άλμα και το JumpReleased σε ψευδή ώστε να μην μπορούν να γίνουν συνεχόμενα άλματα. Ο λόγος που δεν παίρνουμε την τιμή του **myo.transform.rotation.eulerAngles.z** που ισοδυναμεί στο roll της **εικόνας 10** είναι ότι ο χρήστης μπορεί να φορέσει το Myo περιστραμένο αλλιώς κάθε φορά.

```
void FixedUpdate()
    {
        ThalmicMyo thalmicMyo = myo.GetComponent<ThalmicMyo>();

        isGrounded = Physics2D.OverlapCircle(groundPoint.position, radius,
groundMask);

        if (wantsToJump)
        {
            rb2D.AddForce(Vector2.up *jumpHeight);
            AudioSource source = GetComponent<AudioSource>();
            source.volume = 0.4f;
            source.clip = jumpAudioClip;
            source.Play();

            wantsToJump = false;
        }

    }
```

Κώδικας 36 : Άλμα Παίκτη, κομμάτι 2

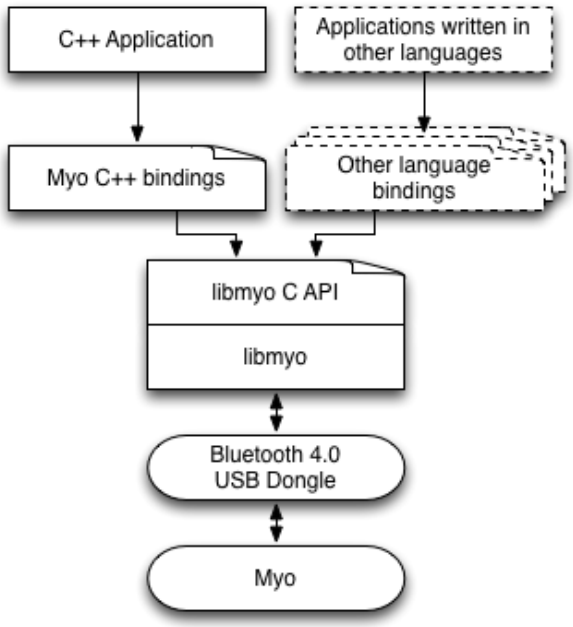
Εδώ κάνουμε ένα έλεγχο για το αν ο χρήστης ακουμπάει στο έδαφος, με τον ίδιο τρόπο που κάναμε έλεγχο για τους αντίπαλους ώστε να αλλάξουν φορά στον **κώδικα 16**. Έπειτα κάνουμε άλλο ένα έλεγχο, εάν έχει δοθεί η εντολή για άλμα τότε δίνουμε μια δύναμη στον άξονα των Y και εκτελούμε ένα αρχείο ήχου για το άλμα. Έπειτα γυρνάμε την τιμή του wantsToJump σε ψευδή ώστε ο χρήστης να μπορεί να ξαναδώσει την εντολή για άλμα.

5.6 Ανάλυση του Myo

Σε αυτό το κεφάλαιο θα αναλύσουμε τον κώδικα που μας παρέχει το SDK του Myo και το Unity Package το οποίο χρησιμοποιούμε.

5.6.1 Βιβλιοθήκη libmyo

Στην βάση του Myo SDK βρίσκεται η βιβλιοθήκη libmyo η οποία επιτρέπει στις εφαρμογές να δουλέψουν με το Myo. Όλες οι λειτουργίες της βιβλιοθήκης είναι διαθέσιμες για χρήση μέσω ενός απλού API (application programming interface) σε C. Οι εφαρμογές δεν δουλεύουν με απευθείας χρήση του API αλλά χρησιμοποιούν μια δεσμευμένη γλώσσα που αντιστοιχεί στην γλώσσα προγραμματισμού που χρησιμοποιεί η εφαρμογή. Η **εικόνα 36** εμφανίζει ένα διάγραμμα με τον τρόπο που συνδέεται μια εφαρμογή με το Myo.



Εικόνα 36 : Από την Εφαρμογή στο Myo

5.6.2 Το αντικείμενο Thalmic Hub

Το hub μπορεί να θεωρηθεί ως η είσοδος στο SDK. Το hub παρέχει ένα τρόπο σύνδεσης με ένα ή περισσότερα Myo. Σε έναν υπολογιστή, η εφαρμογή MyoConnect λειτουργεί σαν το κεντρικό hub και επιτρέπει στις εφαρμογές να συνδεθούν με το Myo και φροντίζει να γίνει η σύνδεση.

Σύμφωνα με τον κώδικα που παρέχεται από το Myo SDK **ThalmicHub.cs** θα μπορούσαμε να έχουμε ένα αντικείμενο Hub 1 - Myo στην πρώτη σκηνή και έπειτα να μεταφέρεται στις υπόλοιπες σκηνές με τον ίδιο τρόπο που μεταφέρουμε και τον ήχο στον **κώδικα 1,** το πρόβλημα με αυτή την εκδοχή είναι ότι κάθε σκηνή έχει αντικείμενα τα οποία πρέπει να συνδεθούν με το αντικείμενο Myo, θα μπορούσε να γίνει αλλά θα χρειαζόταν επιπλέον δουλειά η οποία μπορεί να αποφευχθεί με τον τρόπο που χρησιμοποιούμε τώρα ο οποίος είναι η καταστροφή του αντικειμένου Hub σε κάθε αλλαγή σκηνής και η δημιουργία νέου σε κάθε έναρξη μιας σκηνής.

5.6.3 Το αντικείμενο Thalmic Myo

Το αντικείμενο αυτό αντιπροσωπεύει το Myo. Μέσα στον κώδικα **ThalmicMyo.cs** εμφανίζονται όλα τα δεδομένα που μπορούμε να πάρουμε μέσω του Myo. Χρησιμοποιώντας αυτό το αντικείμενο καταφέρνουμε να πάρουμε τα Gestural και Spatial δεδομένα για να τα χρησιμοποιήσουμε στην εφαρμογή μας.

Η αντιστοίχιση του αντικειμένου με τα υπόλοιπα αντικείμενα τις εφαρμογής γίνεται με τον **κώδικα 37** παρακάτω.

```
public GameObject myo = null;
. . . . .

void Update()
    {
        ThalmicMyo thalmicMyo = myo.GetComponent<ThalmicMyo>();
       . . . . .
```

Κώδικας 37 : Σύνδεση του Myo

Έπειτα μπορούμε να πάρουμε ότι δεδομένα μας προσφέρει το myo, από τα οποία φαίνεται ένα μέρος τους στον **κώδικα 38.**

```
// True if and only if Myo has detected that it is on an arm.
    public bool armSynced;

    // Returns true if and only if Myo is unlocked.
    public bool unlocked;

    // The current arm that Myo is being worn on. An arm of Unknown means that Myo
is unable to detect the arm
    // (e.g. because it's not currently being worn).
    public Arm arm;

    // The current direction of Myo's +x axis relative to the user's arm. A
xDirection of Unknown means that Myo is
    // unable to detect the direction (e.g. because it's not currently being worn).
    public XDirection xDirection;

    // The current pose detected by Myo. A pose of Unknown means that Myo is unable
to detect the pose (e.g. because
    // it's not currently being worn).
    public Pose pose = Pose.Unknown;

    // Myo's current accelerometer reading, representing the acceleration due to
force on the Myo armband in units of
    // g (roughly 9.8 m/s^2) and following Unity coordinate system conventions.
    public Vector3 accelerometer;

    // Myo's current gyroscope reading, representing the angular velocity about
each of Myo's axes in degrees/second
    // following Unity coordinate system conventions.
```

Κώδικας 38 : Κομμάτι του ThalmicMyo.cs

6 Πιθανές Επεκτάσεις

Το μέγεθος των επίπεδων του παιχνιδιού αυτή την στιγμή καθιστούν το παιχνίδι σαν demo και όχι ένα ολοκληρωμένο παιχνίδι οπότε θα ξεκινούσαμε με την δημιουργία νέων επίπεδων και πιθανόν νέα στοιχεία που θα κάνουν το παιχνίδι πιο ενδιαφέρον. Ένα άλλο κομμάτι το οποίο ήταν μέσα στα σχέδια μου να γίνει αλλά τελικά παρέλειψα ήταν η δυνατότητα ο χρήστης όταν δημιουργεί τον λογαριασμό του να μπορεί να του δώσει ένα όνομα 3 γραμμάτων. Ο λόγος που παραλείφθηκε ήταν ότι είναι μια διαδικασία χρονοβόρα η οποία δεν θα εμφάνιζε κάτι διαφορετικό σε θέμα κώδικα που να δείχνει ότι έχω γνώσεις και νέων πράγματων. Για να μπορούσε ο χρήστης να δώσει όνομα, θα υπήρχε μια σελίδα ακόμα η οποία με την χρήση της εναλλαγής μεταξύ των γραμμάτων όπως εναλλάσσουμε τα αντικείμενα του menu ο χρήστης θα επέλεγε το όνομα, έπειτα αυτό θα αποθηκευόταν σε μια string μεταβλητή στα PlayerPrefs για να διατηρείται το όνομα.

Η βασικότερη επέκταση που θα μπορούσε να γίνει είναι η επέκταση του παιχνιδιού και σε άλλες πλατφόρμες. Ήταν στα σχέδια να γίνει το παιχνίδι σε android αλλά λόγω έλλειψης χρόνου και ότι η Thalmic Labs δεν υποστηρίζει άμεσα το Unity στο SDK της αποφασίσαμε να γίνει σε Windows.

7 Συμπεράσματα

Ξεκινώντας με το game development μπορώ να πω ότι ήταν μια ευχάριστη διαδικασία μέσω της οποίας απέκτησα γνώσεις, όμως δεν είναι κάτι το οποίο με βλέπω να συνεχίζω μιας και υπάρχουν αλλά αντικείμενα με τα οποία έχω ασχοληθεί που θεωρώ ότι είναι πιο ενδιαφέρον και με περισσότερες πιθανότητες εύρεσης εργασίας. Έχοντας χρησιμοποιήσει μόνο το Unity δεν μπορώ να το συγκρίνω με άλλες μηχανές δημιουργίας παιχνιδιών αλλά είμαι ικανοποιημένος με τις δυνατότητες που σου δίνει και το community του το οποίο είναι αρκετά μεγάλο με αποτέλεσμα να υπάρχουν απαντήσεις για όλα τα ερωτήματα που μπορεί να εμφανιστούν στην πορεία δημιουργίας του παιχνιδιού.

Μιλώντας γενικά για αισθητήρες κινήσεις θεωρώ ότι είναι ένας τομέας με μέλλον, ποιος δεν θα ήθελε να χειρίζεται την τεχνολογία χωρίς την χρήση περιφερειακών συσκευών? Το Myo είναι ένα βήμα πιο κοντά σε αυτό μιας και δεν δουλεύει με κάμερα αλλά απευθείας πάνω στον χρήστη. Το Myo είναι μια καταπληκτική συσκευή η οποία όπως έχουμε περιγράψει όσο περνάει ο καιρός χρησιμοποιείται και σε νέους τομείς για να βοηθήσουν τον άνθρωπο. Το μόνο πρόβλημα που έχει είναι ότι έχουν περάσει 3 χρονιά από όταν κυκλοφόρησε και ακόμα δεν είναι σίγουρο ότι οι αισθητήρες του θα πιάσουν όλες τις βασικές χειρονομίες. Με το Myo έχω σκοπό να ασχοληθώ ξανά αλλά σε άλλους τομείς όπως εφαρμογές Android ή με την χρήση του σε εφαρμογές με σκοπό την διευκόλυνση του χρήστη να επιτύχει ότι κάνει. Αυτό μπορεί να είναι shortcuts σε εφαρμογές ή ακόμα και η δημιουργία scripts τα οποία να στέλνουν εντολές σε smart συσκευές όπως τηλεοράσεις.

Βιβλιογραφία

- https://unity3d.com/unity
- https://en.wikipedia.org/wiki/Unity_(game_engine)
- https://en.wikipedia.org/wiki/Myo_armband
- https://support.getmyo.com/hc/en-us
- https://www.thalmic.com/
- UnitySpeedTutorials https://www.youtube.com Channel
- Brackeys https://www.youtube.com Channel
- http://opengameart.org/ Assets like enemies and sounds
- http://answers.unity3d.com/
- https://developer.thalmic.com/docs/api_reference/platform/the-sdk.html
- https://www.assetstore.unity3d.com/
- http://www.mariouniverse.com/

Credits

- Android by DezrasDragons (http://opengameart.org/users/dezrasdragons)
- Bat by MoikMellah (http://opengameart.org/content/bat-32x32)
- City Tileset by software_atelier (http://opengameart.org/content/city-pixel-tileset)
- Music by mrpoly (http://opengameart.org/content/menu-music)
- Sound Effects by SubSpaceAudio (http://opengameart.org/content/512-sound-effects-8-bit-style)
- Character by UnitySpeedTutorials Youtube channel
- Collectables by Clint Bellanger (http://opengameart.org/content/recycle-items-set)

- Cannon by psygnosys (http://opengameart.org/content/cannon-gun-sprite-animated)
- Car by chasersgaming (http://opengameart.org/content/2d-car-sprite-2)
- Rest Images from Google Search or Unity Asset Store

Κώδικας

CoinDestroy.cs

```
public class CoinDestroy : MonoBehaviour {

    float destroyTimer;

        void Update ()
    {
        destroyTimer += Time.deltaTime;

        if(destroyTimer > 2)
            Destroy(gameObject);
   }
}
```

CollectAlcohol.cs

```
public class CollectAlcohol : MonoBehaviour
{
    AudioSource source;
    bool taken = false;

    void Start()
    {
        source = gameObject.GetComponent<AudioSource>();
    }
    void Update()
    {
        if (taken)
        {
            if (!source.isPlaying)
                Destroy(this.gameObject);   // When audio has stopped , destroy GameObject
        }
    }
    void OnTriggerEnter2D(Collider2D col)
    {
```

```
        if (col.transform.CompareTag("Player"))
        {
            source.Play();
            Player.playerStatus = 2;
            ScoreManager.AddPoints(75);
            taken = true;
        }
    }
}
```

CollectMilk.cs

```
public class CollectMilk : MonoBehaviour
{
    AudioSource source;
    bool taken = false;

    void Start()
    {
        source = gameObject.GetComponent<AudioSource>();
    }

    void Update()
    {
        if (taken) {
            if (!source.isPlaying)
                Destroy(this.gameObject);
            }
    }

    void OnTriggerEnter2D(Collider2D col)
    {
        if (col.transform.CompareTag("Player"))
        {
            source.Play();
            Player.playerStatus = 1;
            ScoreManager.AddPoints(50);
            taken = true;
        }
    }
}
```

CollectPizza.cs

```
public class CollectPizza : MonoBehaviour
```

```
{
    void OnTriggerEnter2D(Collider2D col)
    {
        if (col.transform.CompareTag("Player"))
        {
            ScoreManager.AddPoints(100);
            col.GetComponent<Player>().AddLife();
            Destroy(gameObject);
        }
    }
}
```

CollectSoda.cs

```
public class CollectSoda : MonoBehaviour {

    void OnTriggerEnter2D(Collider2D col)
    {
        if (col.transform.CompareTag("Player"))
        {
            ScoreManager.AddPoints(25);
            Destroy(gameObject);
        }
    }
}
```

BatPatrol.cs

```
public class BatPatrol : MonoBehaviour
{
        float moveSpeed = 1;
        bool movingRight;
        bool enemyKilled = false;

        Rigidbody2D RigidBody2D;
        Transform Transform;
        Player player;
        AudioSource source;

    public Transform colliderCheck;
    public float colliderCheckRadius;
    public LayerMask whatIsCollider;
        bool hitCollider;

    void Start()
```

```
    {
        Transform = this.transform;
        RigidBody2D = this.GetComponent<Rigidbody2D>();
        source = GetComponent<AudioSource>();
        player = GameObject.FindGameObjectWithTag("Player").GetComponent<Player>();
    }
    void FixedUpdate()
    {
        hitCollider = Physics2D.OverlapCircle(colliderCheck.position, colliderCheckRadius,
whatIsCollider);
            if (hitCollider)
                movingRight = !movingRight;
    }

    void Update()
    {
        Vector3 currentRotation = Transform.eulerAngles;
            if (movingRight)
            {
                RigidBody2D.velocity = new Vector2(moveSpeed, RigidBody2D.velocity.y);
                currentRotation.y = 180;
            }
            else
            {
                RigidBody2D.velocity = new Vector2(-moveSpeed, RigidBody2D.velocity.y);
                currentRotation.y = 0;
            }
        Transform.eulerAngles = currentRotation;

            if (enemyKilled)
            {
                if (!source.isPlaying)
                    Destroy(this.gameObject);
            }
    }

    void OnCollisionEnter2D(Collision2D collision)
    {
        if (collision.gameObject.CompareTag("Player"))
        {
            bool killOnce = true;
            foreach (ContactPoint2D point in collision.contacts)
            {
                if (Player.playerStatus != 2)
```

```
            {
                if (point.normal.y <= -0.5f)
                {
                    player.BounceOnEnemyKilled();
                    ScoreManager.AddPoints(125);
                    source.Play();
                    this.GetComponent<BoxCollider2D>().enabled = false;
                    this.GetComponent<SpriteRenderer>().enabled = false;
                    enemyKilled = true;
                }
                else
                {
                    if (killOnce)
                    {
                        player.Die();
                        killOnce = false;
                    }
                }
            }
            else
            {
                ScoreManager.AddPoints(125);
                source.Play();
                this.GetComponent<BoxCollider2D>().enabled = false;
                this.GetComponent<SpriteRenderer>().enabled = false;
                enemyKilled = true;
            }
        }
        killOnce = true;
    }
  }
}
```

CannonAI.cs

```
public class CannonAI : MonoBehaviour
{
    float distance;
    public float riseRange;
    public float shootInterval;
    public float bulletSpeed;
    public float bulletTimer;
    public bool risen = false;
    public bool lookRight = false;
```

```
public bool inRange = false;
public GameObject bullet;
public Transform target;
public Animator animator;
public Transform shootPoint;

void Start()
{
   animator = gameObject.GetComponent<Animator>();
}

void Update()
{
   animator.SetBool("risen", risen);
   animator.SetBool("lookRight", lookRight);
   animator.SetBool("inRange", inRange);

   RangeCheck();

   if(target.position.x > transform.position.x)
      lookRight = true;
   else
      lookRight = false;
}

void RangeCheck()
{
   distance = Vector3.Distance(transform.position, target.position);
      if(distance < riseRange)
         risen = true;
      if(distance > riseRange)
         risen = false;
}

public void Attack(bool attackPermit)
{
   print(attackPermit);
   bulletTimer += Time.deltaTime;
   if(bulletTimer >= shootInterval)
   {
      Vector2 direction = target.transform.position - transform.position;
      direction.Normalize();
      GameObject bulletClone;
```

```
            bulletClone = Instantiate(bullet, shootPoint.transform.position,
shootPoint.transform.rotation) as GameObject;
            bulletClone.GetComponent<Rigidbody2D>().velocity = direction;
            bulletTimer = 0;
        }
    }
}
```

CannonAttackCone.cs

```
public class CannonAttackCone : MonoBehaviour {

    public CannonAI cannonAI;

     void Start()
    {
        cannonAI = gameObject.GetComponentInParent<CannonAI>();
    }

    void OnTriggerStay2D(Collider2D col)
    {
        if (col.CompareTag("Player"))
        {
            cannonAI.inRange = true;
            cannonAI.Attack(true);
        }
    }
    void OnTriggerExit2D(Collider2D col)
    {
        if (col.CompareTag("Player"))
            cannonAI.inRange = false;
    }
}
```

CannonBullet.cs

```
public class CannonBullet : MonoBehaviour {

    Player player;
    float destroyTimer;

    void Start()
    {
        player = GameObject.FindGameObjectWithTag("Player").GetComponent<Player>();
```

```
    }

    void Update()
    {
        destroyTimer += Time.deltaTime;
            if(destroyTimer > 3)
                Destroy(gameObject);
    }

    void OnTriggerEnter2D(Collider2D col)
    {
        if (col.CompareTag("Player"))
            player.Die();
    }
}
```

CannonDeath.cs

```
public class CannonDeath : MonoBehaviour {

    Player player;
    int timesHit;
    AudioSource source;
    bool enemyKilled = false;

    void Start()
    {
        player = GameObject.FindGameObjectWithTag("Player").GetComponent<Player>();
        source = GetComponent<AudioSource>();
    }

    void Update()
    {
        if (enemyKilled)
        {
            if (!source.isPlaying)
                Destroy(this.gameObject);
        }
    }

    void OnCollisionEnter2D(Collision2D col)
    {
        if (col.gameObject.CompareTag("Player"))
        {
```

```
        foreach (ContactPoint2D point in col.contacts)
        {
            if (point.normal.y <= -0.5f)
            {
                player.BounceOnEnemyKilled();
                    if (timesHit >= 2)
                    {
                        ScoreManager.AddPoints(300);
                        source.Play();
                        this.GetComponent<BoxCollider2D>().enabled = false;
                        this.GetComponent<SpriteRenderer>().enabled = false;
                        enemyKilled = true;
                    }
                timesHit++;
            }
        }
    }
}
```

EnemyPatrol.cs

```
public class EnemyPatrol : MonoBehaviour {

    float moveSpeed = 1;     // enemys speed
    bool movingRight;
    int timesHit;
    bool enemyKilled = false;

    AudioSource source;
    Rigidbody2D RigidBody2D;
    Transform Transform;
    SpriteRenderer Renderer;
    Player player;

    public Transform wallCheck;
    public float wallCheckRadius;
    public LayerMask whatIsWall;
        bool hitWall;
        bool foundEdge;
    public Transform edgeCheck;

    void Start()
    {
```

```
        Transform = this.transform;
        RigidBody2D = this.GetComponent<Rigidbody2D>();
        Renderer = gameObject.GetComponent<SpriteRenderer>();
        source = GetComponent<AudioSource>();
        player = GameObject.FindGameObjectWithTag("Player").GetComponent<Player>();
    }

    void FixedUpdate()
    {
        hitWall = Physics2D.OverlapCircle(wallCheck.position, wallCheckRadius, whatIsWall);
        foundEdge = Physics2D.OverlapCircle(edgeCheck.position, wallCheckRadius,
whatIsWall);

        if (hitWall || !foundEdge)
            movingRight = !movingRight;
    }

    void Update()
    {
        Vector3 currentRotation = Transform.eulerAngles;
            if (movingRight)
            {
                RigidBody2D.velocity = new Vector2(moveSpeed, RigidBody2D.velocity.y);
                currentRotation.y = 0;
            }
            else
            {
                RigidBody2D.velocity = new Vector2(-moveSpeed, RigidBody2D.velocity.y);
                currentRotation.y = 180;
            }
        Transform.eulerAngles = currentRotation;
            if (enemyKilled)
            {
                if (!source.isPlaying)
                    Destroy(this.gameObject);
            }
    }

    void OnCollisionEnter2D(Collision2D collision)
    {
        if (collision.gameObject.CompareTag("Player"))
        {
            bool killOnce = true;
```

```
foreach (ContactPoint2D point in collision.contacts)
{
   Debug.DrawLine(point.point, point.point + point.normal, Color.red, 10);
   if(Player.playerStatus != 2)
   {
      if (point.normal.y <= -0.5f)    // By looking Debug.Log(point.normal);
      {
         player.BounceOnEnemyKilled();
         if (this.tag == "EnemyDouble")
         {
            Renderer.color = new Color(1f, 1f, 1f, 1f);
               if (timesHit >= 1)
               {
                  ScoreManager.AddPoints(175);
                  source.Play();
                  this.GetComponent<BoxCollider2D>().enabled = false;
                  this.GetComponent<SpriteRenderer>().enabled = false;
                  enemyKilled = true;
               }
            timesHit++;
         }
         else
         {
            ScoreManager.AddPoints(100);
            source.Play();
            this.GetComponent<BoxCollider2D>().enabled = false;
            this.GetComponent<SpriteRenderer>().enabled = false;
            enemyKilled = true;
         }

      }
      else
      {
         if (killOnce)
         {
            player.Die();
            killOnce = false;
         }
      }
   }
   else    // If Player.status is 2 --> Player is invincible
   {
      if (this.tag == "EnemyDouble")
         ScoreManager.AddPoints(175);
```

```
            else
                ScoreManager.AddPoints(100);
            source.Play();
            this.GetComponent<BoxCollider2D>().enabled = false;
            this.GetComponent<SpriteRenderer>().enabled = false;
            enemyKilled = true;
          }
        }
        killOnce = true;
      }
    }
}
```

Spikes.cs

```
public class Spikes : MonoBehaviour {

    Player player;

        void Start ()
    {
        player = GameObject.FindGameObjectWithTag("Player").GetComponent<Player>();
        }

    void OnTriggerEnter2D(Collider2D col)
    {
        if (col.CompareTag("Player"))
        player.Die();
    }
}
```

Spike_Pipe_Up_Down.cs

```
public class Spike_Pipe_Up_Down : MonoBehaviour {

    public float delayTime;

    void Start()
    {
        StartCoroutine(AnimateSpikesCo());
    }

    IEnumerator AnimateSpikesCo()
    {
        while (true)
        {
```

```
            GetComponent<Animation>().Play();
            yield return new WaitForSeconds(delayTime);
        }
    }
}
```

TheGap.cs

```
public class TheGap : MonoBehaviour {

    Player player;

    void Start()
    {
        player = GameObject.FindGameObjectWithTag("Player").GetComponent<Player>();
    }

    void OnTriggerEnter2D(Collider2D col)
    {
        if (col.CompareTag("Player"))
        {
            Player.playerStatus = 0;
            player.Die();
        }
    }
}
```

ScoreManager.cs

```
public class ScoreManager : MonoBehaviour {

    public static int score;
    public GameObject textObject;
    Text text;

    void Start()
    {
        textObject = GameObject.FindGameObjectWithTag("Score");
        text = textObject.GetComponent<Text>();

        if(PlayerPrefs.GetInt("PlayerScore") <= 0)
            score = 0;
        else
            score = PlayerPrefs.GetInt("PlayerScore");
    }
```

```
    void Update()
    {
        if (score <= 0)
            score = 0;

        text.text = score.ToString();
    }

    public static void AddPoints(int points)
    {
        score += points;
    }
}
```

HealthHUD.cs

```
public class HealthHUD : MonoBehaviour
{
    public Sprite[] HeartSprites;
    public Image HeartUI;
    Player player;

    void Start()
    {
        player = GameObject.FindGameObjectWithTag("Player").GetComponent<Player>();
    }

    void Update()
    {
        HeartUI.sprite = HeartSprites[player.currentPlayerLifes];
    }
}
```

LevelManager.cs

```
public class LevelManager : MonoBehaviour {

    Player player;
    PlayerController2D playerController;

    public GameObject currentCheckpoint;
    public GameObject playerObject;
    public GameObject deathParticles;
```

```
    public float respawnDelay;
    int currentLevel;

    void Start ()
    {
        player = GameObject.FindGameObjectWithTag("Player").GetComponent<Player>();
        playerController =
GameObject.FindGameObjectWithTag("Player").GetComponent<PlayerController2D>();
        currentLevel = SceneManager.GetActiveScene().buildIndex;

        if(currentLevel == 1)
        {
            PlayerPrefs.SetInt("PlayerScore", 0);
            PlayerPrefs.SetInt("PlayerLifes", 5);
        }
    }

    public void RespawnPlayer()
    {
        StartCoroutine("RespawnPlayerDelayCo");
    }

    public IEnumerator RespawnPlayerDelayCo()
    {
        Instantiate(deathParticles, player.transform.position, player.transform.rotation);
        playerObject.SetActive(false);

        yield return new WaitForSeconds(respawnDelay);

        playerObject.SetActive(true);
        player.transform.position = currentCheckpoint.transform.position;
    }
}
```

PauseMenu.cs

```
public class PauseMenu : MonoBehaviour {

    public GameObject myo = null;
    public GameObject inGameUI;

    public bool showMenu = false;  //show Menu
    public bool showWinScreen = false; // if win , what is shown in Menu
    bool showPauseScreen = false; // if paused what is shown in Menu
```

```
Pose _lastPose = Pose.Unknown;
int currentLevel;
int currentScore;

GameObject camera;
GameObject player;

// GUI Skin
public GUISkin skin;
public Texture menuBackgroundTexture;
public Texture muteTexture;
public Texture unmuteTexture;
private Texture currentAudioTexture;
public Texture gestureWaveOutTexture;
public Texture gestureFistTexture;
public Texture gestureFingersSpreadTexture;
public Texture gestureWaveInTexture;

void Start()
{
   inGameUI.SetActive(true);
   currentLevel = SceneManager.GetActiveScene().buildIndex;
   player = GameObject.FindGameObjectWithTag("Player");
   camera = GameObject.FindGameObjectWithTag("MainCamera");

   if (camera.GetComponent<AudioListener>().enabled)
      currentAudioTexture = unmuteTexture;
   else
      currentAudioTexture = muteTexture;
}

void Update()
{
   currentScore = ScoreManager.score;
   ThalmicMyo thalmicMyo = myo.GetComponent<ThalmicMyo>();

   if (showMenu)
      player.GetComponent<PlayerController2D>().enabled = false;
   else
      player.GetComponent<PlayerController2D>().enabled = true;


   if (thalmicMyo.pose != _lastPose)
```

```
{
    _lastPose = thalmicMyo.pose;    // if lastPose has changed get new one

    if (showMenu)
    {
        if (thalmicMyo.pose == Pose.Fist)
        {
            if (showWinScreen)
                LoadNextLevel();
            else
                resumeGame();
        }

        if (thalmicMyo.pose == Pose.WaveIn)
            goToMainMenu();

        if (thalmicMyo.pose == Pose.WaveOut)
        {
            if (showPauseScreen)
                tryAgain();
        }

        if (thalmicMyo.pose == Pose.FingersSpread)
            changeAudioState();

    }
    else
    {
        if (thalmicMyo.pose == Pose.WaveIn)
        {
            Time.timeScale = 0f;    // Stop World time for everything that is moving
            showMenu = true;
            showPauseScreen = true;
        }
    }
}

if (showMenu)
{
    if (showPauseScreen)
    {
        if (Input.GetKeyDown(KeyCode.Escape))
            resumeGame();
    }
```

```
        }
        else
        {
            if (Input.GetKeyDown(KeyCode.Escape))
            {
                Time.timeScale = 0f;
                showMenu = true;
                showPauseScreen = true;
            }
        }
    }

    void OnGUI()
    {
        if (showMenu)
        {
            GUI.skin = skin;
            GUI.color = new Color(1, 1, 1, 0.7f);   // Lower Alpha to 0.7
            GUI.DrawTexture(new Rect(0, 0, Screen.width, Screen.height),
menuBackgroundTexture);
            GUI.color = Color.white; // Get Alpha back up

            if (showWinScreen)
            {
                GUI.Label(new Rect(Screen.width / 2 - 150, 150, 300, 50), "Level Completed!",
"InGameMenuTitles");

                GUI.Label(new Rect(Screen.width / 2 - 130, 300, 300, 30), "Score Achieved : " +
currentScore.ToString(), "MenuItems");
                GUI.Label(new Rect(Screen.width / 2 - 130, 250, 300, 30), "Completed Level : " +
currentLevel, "MenuItems");


                if (GUI.Button(new Rect(Screen.width / 2 + 50, 400, 150, 40), "Continue"))
                    LoadNextLevel();
                GUI.DrawTexture(new Rect(Screen.width / 2, 400, 50, 50), gestureFistTexture);

                if (GUI.Button(new Rect(Screen.width / 2 - 200, 400, 100, 50), "Quit"))
                    goToMainMenu();
                GUI.DrawTexture(new Rect(Screen.width / 2 - 100, 400, 50, 50),
gestureWaveInTexture);
            }
            else
            {
```

```
            GUI.Label(new Rect(Screen.width / 2 - 150, 150, 300, 50), "Game Paused",
"InGameMenuTitles");

            if (GUI.Button(new Rect(Screen.width / 2 - 20, 250, 150, 50), "Resume"))
                resumeGame();
            GUI.DrawTexture(new Rect(Screen.width / 2 - 70, 250, 50, 50),
gestureFistTexture);

            if (GUI.Button(new Rect(Screen.width / 2 - 20, 350, 150, 50), "Try Again"))
                tryAgain();
            GUI.DrawTexture(new Rect(Screen.width / 2 - 70, 350, 50, 50),
gestureWaveOutTexture);



            if (GUI.Button(new Rect(Screen.width / 2 - 20, 450, 150, 50), "Quit"))
                goToMainMenu();
            GUI.DrawTexture(new Rect(Screen.width / 2 - 70, 450, 50, 50),
gestureWaveInTexture);
        }

           if (GUI.Button(new Rect(Screen.width - 150, 100, 100, 100), currentAudioTexture))
                changeAudioState();
            GUI.DrawTexture(new Rect(Screen.width - 230, 115, 70, 70),
gestureFingersSpreadTexture);
    }
  }

  void LoadNextLevel()
  {
    Time.timeScale = 1f;    // Start World Time so everything can move
    ThalmicHub.Destroy(ThalmicHub.instance);    // Destroy Thalmic Hub since every level
has its own and can't have 2 at the same time
    if (currentLevel == 5)

player.GetComponent<Player>().GameOver();//SceneManager.LoadScene("MainMenu");
    else
      SceneManager.LoadScene(currentLevel + 1);
  }

  void tryAgain()
  {
    Time.timeScale = 1f;
    ThalmicHub.Destroy(ThalmicHub.instance);
    SceneManager.LoadScene(currentLevel);
```

```
    }

    void changeAudioState()
    {
        if (camera.GetComponent<AudioListener>().enabled)
        {
            camera.GetComponent<AudioListener>().enabled = false;
            currentAudioTexture = unmuteTexture;
        }
        else
        {
            camera.GetComponent<AudioListener>().enabled = true;
            currentAudioTexture = muteTexture;
        }
    }

    void goToMainMenu()
    {
        Time.timeScale = 1f;
        ThalmicHub.Destroy(ThalmicHub.instance);
        PlayerPrefs.SetInt("PlayerScore", 0);
        PlayerPrefs.SetInt("PlayerLifes", 5);
        SceneManager.LoadScene("MainMenu");
    }

    void resumeGame()
    {
        Time.timeScale = 1f;
        showPauseScreen = false;
        showMenu = false;
    }
}
```

PlayerController.cs

```
public class PlayerController2D : MonoBehaviour {

    public GameObject myo = null;

    Pose _lastPose = Pose.Unknown;

    public int moveSpeed;
        public int jumpHeight;
    public GameObject playerGraphics;
```

```
// stuff needed for Jumping
        public Transform groundPoint;
        public float radius;
        public LayerMask groundMask;
        public bool isGrounded;
public AudioClip jumpAudioClip;
bool wantsToJump = false;
bool jumpReleased = false;

Rigidbody2D rb2D;
public float moveHorizontal;

Vector3 movingRightScale;
Vector3 movingLeftScale;
void Start ()
{
                rb2D = GetComponent<Rigidbody2D>();
}

void Update ()
{
   ThalmicMyo thalmicMyo = myo.GetComponent<ThalmicMyo>();

   if (Player.playerStatus != 1)
   {
      movingRightScale = new Vector3(0.7f, 0.7f, 1f);
      movingLeftScale = new Vector3(-0.7f, 0.7f, 1f);
   }
   else
   {
      movingRightScale = new Vector3(1f, 1f, 1f);
      movingLeftScale = new Vector3(-1f, 1f, 1f);
   }

   float x = myo.transform.rotation.eulerAngles.x;

   if ((20 < x && x < 180) || (Input.GetAxisRaw("Horizontal") == 1))
   {
      moveHorizontal = 1;
      transform.localScale = movingRightScale;
      playerGraphics.GetComponent<Animator>().SetBool("isMoving", true);
   }
   else if ((180 < x && x < 345) || (Input.GetAxisRaw("Horizontal") == -1))
```

```
        {
            moveHorizontal = -1;
            transform.localScale = movingLeftScale;
            playerGraphics.GetComponent<Animator>().SetBool("isMoving", true);
        }
        else
        {
            playerGraphics.GetComponent<Animator>().SetBool("isMoving", false);
            moveHorizontal = 0;
        }

        Vector2 moveDir = new Vector2(moveHorizontal * moveSpeed, rb2D.velocity.y);
                    rb2D.velocity = moveDir;

        if (thalmicMyo.isPaired)
        {
            if ((Input.GetKeyDown(KeyCode.Space) || thalmicMyo.accelerometer.x < 0.2f) &&
isGrounded && jumpReleased)
            {
                wantsToJump = true;
                jumpReleased = false;
            }
        }

            if (Input.GetKeyDown(KeyCode.Space) && isGrounded)
            {
                wantsToJump = true;
            }


        if(thalmicMyo.accelerometer.x > 0.2f)
        {
            jumpReleased = true;
        }

    }

    void FixedUpdate()
    {
        ThalmicMyo thalmicMyo = myo.GetComponent<ThalmicMyo>();

        isGrounded = Physics2D.OverlapCircle(groundPoint.position, radius, groundMask);

        if (wantsToJump)
```

```
    {
        rb2D.AddForce(Vector2.up *jumpHeight);
        AudioSource source = GetComponent<AudioSource>();
        source.volume = 0.4f;
        source.clip = jumpAudioClip;
        source.Play();

        wantsToJump = false;
    }
  }

        void OnDrawGizmos()
  {
                Gizmos.color = Color.red;
                Gizmos.DrawWireSphere(groundPoint.position, radius);
        }
}
```

PlayerController.cs

```
public class CameraFollow : MonoBehaviour {

  Vector2 velocity;

  public float smoothTimeY;
  public float smoothTimeX;

  public GameObject player;
  public PlayerController2D playerController;

  public bool bounds;
  public Vector3 minCameraPos;
  public Vector3 maxCameraPos;

  public Transform backWardsMovingCollider;

        void Start ()
  {
    player = GameObject.FindGameObjectWithTag("Player");
    playerController =
GameObject.FindGameObjectWithTag("Player").GetComponent<PlayerController2D>();
  }

  void FixedUpdate()
```

```
  {
     float posX = Mathf.SmoothDamp(transform.position.x, player.transform.position.x, ref
velocity.x, smoothTimeX);
     float posY = Mathf.SmoothDamp(transform.position.y, player.transform.position.y, ref
velocity.y, smoothTimeY);

     transform.position = new Vector3(posX + 0.5f, posY - 0.1f, transform.position.z);

     if(minCameraPos.x > backWardsMovingCollider.position.x)
     {
        if(player.transform.position.y < -1)
        {
           transform.position = new Vector3(
              Mathf.Clamp(transform.position.x, minCameraPos.x, maxCameraPos.x),
              Mathf.Clamp(transform.position.y, minCameraPos.y, maxCameraPos.y),
              Mathf.Clamp(transform.position.z, minCameraPos.z, maxCameraPos.z));
        }
        else
        {
           transform.position = new Vector3(
              Mathf.Clamp(transform.position.x, minCameraPos.x, maxCameraPos.x),
              Mathf.Clamp(transform.position.y, 1, maxCameraPos.y),
              Mathf.Clamp(transform.position.z, minCameraPos.z, maxCameraPos.z));
        }

     }
     else
     {
        if (player.transform.position.y < -1)
        {
           transform.position = new Vector3(
              Mathf.Clamp(transform.position.x, backWardsMovingCollider.position.x + 2.8f,
maxCameraPos.x),
              Mathf.Clamp(transform.position.y, minCameraPos.y, maxCameraPos.y),
              Mathf.Clamp(transform.position.z, minCameraPos.z, maxCameraPos.z));
        }
        else
        {
           transform.position = new Vector3(
              Mathf.Clamp(transform.position.x, backWardsMovingCollider.position.x + 2.8f,
maxCameraPos.x),
              Mathf.Clamp(transform.position.y, 1, maxCameraPos.y),
              Mathf.Clamp(transform.position.z, minCameraPos.z, maxCameraPos.z));
        }
```

```
        }
    }
}
```

NotBackMovingCollider.cs

```
public class NotBackMovingCollider : MonoBehaviour {

    public float distance;
    float collidersX;
    float playersX;
    public Transform targetPlayer;

    void Update()
    {
        RangeCheck();
    }
    void RangeCheck()
    {
        playersX = targetPlayer.position.x;
        collidersX = transform.position.x;
        distance = playersX - collidersX;

        if (distance > 5 || distance < 0)
            transform.position = new Vector3(targetPlayer.position.x - 5, targetPlayer.position.y,
targetPlayer.position.z);
    }
}
```

Player.cs

```
public class Player : MonoBehaviour {

    public GameObject myo = null;

    public int currentPlayerLifes;
    public int maxPlayerLifes;

    int currentLevel;

    Rigidbody2D rb2D;

    public GameObject playerGraphics;
    PlayerController2D playerController;
    public LevelManager levelManager;
```

```
    public int jumpVelocity;

    public static int playerStatus;

    public Color normalColor;   // Normal State
    public Color alcoholColor;  // Invincibility State

    float invincibilityTimer;
    public AudioClip invincibilityEndsAudio;

    ThalmicMyo thalmicMyo;

    void Start()
    {
        playerStatus = 0;
        thalmicMyo = myo.GetComponent<ThalmicMyo>();
        rb2D = GetComponent<Rigidbody2D>();
        currentLevel = SceneManager.GetActiveScene().buildIndex;
        levelManager = FindObjectOfType<LevelManager>();
        playerController =
GameObject.FindGameObjectWithTag("Player").GetComponent<PlayerController2D>();

        if (currentLevel != 1)
            currentPlayerLifes = PlayerPrefs.GetInt("PlayerLifes");
        else
            currentPlayerLifes = maxPlayerLifes;
    }

            void Update ()
    {
        if (currentPlayerLifes > maxPlayerLifes)
            currentPlayerLifes = maxPlayerLifes;
        if (currentPlayerLifes <= 0)
            GameOver();

        Status();
    }

    void Status()
    {
        if(playerStatus == 0)
        {
            if(playerController.moveHorizontal == 1)
```

```
            this.transform.localScale = new Vector3(0.7f, 0.7f, 0.7f);
        else if(playerController.moveHorizontal == -1)
            this.transform.localScale = new Vector3(-0.7f, 0.7f, 0.7f);

        playerGraphics.GetComponent<SpriteRenderer>().color = normalColor;
    }

    if(playerStatus == 1)
    {
        if (playerController.moveHorizontal == 1)
            this.transform.localScale = new Vector3(1f, 1f, 1f);
        else if (playerController.moveHorizontal == -1)
            this.transform.localScale = new Vector3(-1f, 1f, 1f);

        playerGraphics.GetComponent<SpriteRenderer>().color = normalColor;
    }

    if(playerStatus == 2)
    {
        invincibilityTimer += Time.deltaTime;

        if(invincibilityTimer > 9)
        {
            AudioSource source = GetComponent<AudioSource>();
            source.clip = invincibilityEndsAudio;
            source.volume = 0.5f;
            source.Play();
        }

        if(invincibilityTimer >= 10)
        {
            invincibilityTimer = 0;
            playerStatus = 0;
        }

        if (playerController.moveHorizontal == 1)
            this.transform.localScale = new Vector3(0.7f, 0.7f, 0.7f);
        else if (playerController.moveHorizontal == -1)
            this.transform.localScale = new Vector3(-0.7f, 0.7f, 0.7f);

        playerGraphics.GetComponent<SpriteRenderer>().color = alcoholColor;
    }
}
```

```
public void Die()
{
    if (thalmicMyo.isPaired)
        thalmicMyo.Vibrate(Thalmic.Myo.VibrationType.Short);

    if (playerStatus <= 0)
    {
        playerStatus = 0;
        levelManager.RespawnPlayer();
        currentPlayerLifes -= 1;
    }
    else if(playerStatus == 1)
    {
        playerStatus = 0;
        StartCoroutine(PlayerHitWhileBigCo());
    }
}

public IEnumerator PlayerHitWhileBigCo()
{
    this.GetComponent<CircleCollider2D>().enabled = false;
    this.GetComponent<BoxCollider2D>().enabled = false;
    playerGraphics.GetComponent<Animation>().Play("player_damaged");

    yield return new WaitForSeconds(1);

    this.GetComponent<CircleCollider2D>().enabled = true;
    this.GetComponent<BoxCollider2D>().enabled = true;
}

public void AddLife()
{
    currentPlayerLifes += 1;
}

public void GameOver()
{
    PlayerPrefs.SetInt("GameOverScore", ScoreManager.score);
    PlayerPrefs.SetInt("PlayerScore", 0);
    PlayerPrefs.SetInt("PlayerLifes", 5);
    ThalmicHub.Destroy(ThalmicHub.instance);
    SceneManager.LoadScene("MainMenu");
}
```

```
    public void BounceOnEnemyKilled()
    {
        Vector2 velocity = rb2D.velocity;
        velocity.y = jumpVelocity;
        rb2D.velocity = velocity;
    }
}
```

KeepSoundBetweenScenes.cs

```
public class KeepSoundBetweenScenes : MonoBehaviour
{
    static KeepSoundBetweenScenes instance = null;
    public static KeepSoundBetweenScenes Instance
    {
        get { return instance; }
    }

    void Awake()
    {
        if (instance != null && instance != this)
        {
            Destroy(this.gameObject);
            return;
        }
        else {
            instance = this;
        }
        DontDestroyOnLoad(this.gameObject);
    }

    void OnApplicationQuit()
    {
        Application.CancelQuit();
        System.Diagnostics.Process.GetCurrentProcess().Kill();
    }
}
```

Car_Final_Point.cs

```
public class Car_Final_Point : MonoBehaviour {

    public GameObject player;

    PlayerController2D playerController;
```

```
int currentLevel;
AudioSource source;
PauseMenu pauseMenu;

    void Start ()
{
   playerController =
GameObject.FindGameObjectWithTag("Player").GetComponent<PlayerController2D>();
   currentLevel = SceneManager.GetActiveScene().buildIndex;
   source = GetComponent<AudioSource>();
   pauseMenu =
GameObject.FindGameObjectWithTag("GameManager").GetComponent<PauseMenu>();
}

void OnTriggerEnter2D(Collider2D col)
{
   if(col.CompareTag("Player"))
   {
      SaveScore();
      StartCoroutine(LevelCompletedCo());
   }
}

public IEnumerator LevelCompletedCo()
{
   player.SetActive(false);
   gameObject.GetComponent<Animation>().Play();
   source.Play();
   yield return new WaitForSeconds(2);
   pauseMenu.showMenu = true;
   pauseMenu.showWinScreen = true;
}

void SaveScore()
{
   PlayerPrefs.SetInt("PlayerLifes", player.GetComponent<Player>().currentPlayerLifes);
   PlayerPrefs.SetInt("PlayerScore", ScoreManager.score);  // Score saved for level

   if(PlayerPrefs.GetInt("Player" + PlayerPrefs.GetInt("CurrentPlayer") + "Level") <
currentLevel + 1 && currentLevel != 5)
      PlayerPrefs.SetInt("Player" + PlayerPrefs.GetInt("CurrentPlayer") + "Level" ,
currentLevel + 1);    // Since Finish is reached, we add 1 to current level, so it will be
unlocked on Level Selection
}
```

```
}
```

CheckPoint.cs

```
public class CheckPoint : MonoBehaviour {

    public LevelManager levelManager;

        void Start ()
    {
        levelManager = FindObjectOfType<LevelManager>();
        }

    void OnTriggerEnter2D(Collider2D col)
    {
        if (col.CompareTag("Player"))
            levelManager.currentCheckpoint = gameObject;
    }
}
```

ChoosePlayerManager.cs

```
public class ChoosePlayerManager : MonoBehaviour {

    public GameObject myo = null;
    Pose _lastPose = Pose.Unknown;

    public GUISkin skin;
    public Texture selectPlayerTexture;
    public Texture gestureFist;
    public Texture gestureFingersSpread;
    public Texture gestureWaveOut;
    public Texture gestureWaveIn;
    public Texture menuBackgroundTexture;

    int currentPlayerNum;
    int counter = 1;

    void Start ()
    {
        PlayerPrefs.SetInt("GameOverScore", 0);

        if (PlayerPrefs.GetInt("NumOfPlayers") > 0)
        {
            currentPlayerNum = PlayerPrefs.GetInt("NumOfPlayers") + 1;
        }
```

```
    else
    {
        currentPlayerNum = 1;
        CreateNewAndgoToMainMenu();
    }
}

    void Update ()
{
    ThalmicMyo thalmicMyo = myo.GetComponent<ThalmicMyo>(); // set myo

    if (thalmicMyo.pose != _lastPose)
    {
        _lastPose = thalmicMyo.pose;    // if lastPose has changed get new one

        if (thalmicMyo.pose == Pose.WaveOut)
        {
            if (counter <= PlayerPrefs.GetInt("NumOfPlayers"))
                counter++;
        }

        if (thalmicMyo.pose == Pose.WaveIn)
        {
            if (counter > 1)
                counter--;
        }
    }

    if (Input.GetKeyDown(KeyCode.RightArrow))
    {
        if (counter <= PlayerPrefs.GetInt("NumOfPlayers"))
            counter++;
    }

    if (Input.GetKeyDown(KeyCode.LeftArrow))
    {
        if (counter > 1)
            counter--;
    }

    if (thalmicMyo.pose == Pose.FingersSpread)
        PlayerChosenAndGoToMainMenu();

}
```

```
   void OnGUI()
   {
      GUI.skin = skin;
      GUI.color = new Color(1, 1, 1, 0.5f);   // Lower Alpha to 0.5
      GUI.DrawTexture(new Rect(0, 0, Screen.width, Screen.height),
menuBackgroundTexture);
      GUI.color = Color.white; // Get Alpha back up

      GUI.Label(new Rect(Screen.width / 2 - 250, Screen.height / 2 - 350, 500, 200), "The
Game");

      Rect secondRowLeftRect = new Rect(Screen.width / 2 - 50, Screen.height / 2 + 50, 100,
100);
      Rect secondRowTextRect = new Rect(Screen.width / 2 - 50, Screen.height / 2 + 150, 100,
50);
      Rect secondRowGestureRect = new Rect(Screen.width / 2 + 70, Screen.height / 2 + 70,
60, 60);

      Rect leftGestureRect = new Rect(Screen.width / 2 - 170, Screen.height / 2 + 80, 40, 40);
      Rect rightGestureRect = new Rect(Screen.width / 2 + 220, Screen.height / 2 + 80, 40,
40);

      if(counter != PlayerPrefs.GetInt("NumOfPlayers") + 1)
      {
         if (GUI.Button(secondRowLeftRect, selectPlayerTexture))
            PlayerChosenAndGoToMainMenu();

         GUI.Label(secondRowTextRect, "Play As Player " + counter, "MenuItems");
         GUI.DrawTexture(secondRowGestureRect, gestureFingersSpread);
      }
      else
      {
         if (GUI.Button(secondRowLeftRect, selectPlayerTexture))
            CreateNewAndgoToMainMenu();

         GUI.Label(secondRowTextRect, "Create Player " + currentPlayerNum.ToString(),
"MenuItems");
         GUI.DrawTexture(secondRowGestureRect, gestureFingersSpread);
      }

      GUI.color = new Color32(255, 255, 255, 150);    // Lower Opacity
      if(counter != 1)
         GUI.DrawTexture(rightGestureRect, gestureWaveIn);
```

```
        if(counter != PlayerPrefs.GetInt("NumOfPlayers") + 1 && currentPlayerNum > 1)
            GUI.DrawTexture(leftGestureRect, gestureWaveOut);

        GUI.color = new Color32(255, 255, 255, 255);    // Max opacity
    }

    void CreateNewAndgoToMainMenu()
    {
        if (PlayerPrefs.GetInt("NumOfPlayers") > 0)
            PlayerPrefs.SetInt("NumOfPlayers", currentPlayerNum);
        else
            PlayerPrefs.SetInt("NumOfPlayers", 1);

        PlayerPrefs.SetInt("CurrentPlayer", currentPlayerNum);

        ThalmicHub.Destroy(ThalmicHub.instance);    // Destroy Thalmic Hub since every level
has its own and can't have 2 at the same time
        SceneManager.LoadScene("MainMenu");     // Load Main Menu Scene
    }
    void PlayerChosenAndGoToMainMenu()
    {
        PlayerPrefs.SetInt("CurrentPlayer", counter);

        ThalmicHub.Destroy(ThalmicHub.instance);    // Destroy Thalmic Hub since every level
has its own and can't have 2 at the same time
        SceneManager.LoadScene("MainMenu");     // Load Main Menu Scene
    }
}
```

DestroyParticles.cs

```
public class DestroyParticles : MonoBehaviour {

    private ParticleSystem thisParticleSystem;

    void Start()
    {
        thisParticleSystem = GetComponent<ParticleSystem>();
    }

    void Update ()
    {
        if (thisParticleSystem.isPlaying)
```

```
            return;
        Destroy(gameObject);
            }
}
```

FallingGround.cs

```
public class FallingGround : MonoBehaviour {

    Rigidbody2D rigidbody;
    public float fallDelay;

            void Start ()
    {
        rigidbody = GetComponent<Rigidbody2D>();
            }

    void Update()
    {
        if(transform.position.y < -30)
            Destroy(this.gameObject);
    }

    void OnCollisionEnter2D(Collision2D col)
    {
        if (col.transform.CompareTag("Player"))
            StartCoroutine(FallCo());
    }

    IEnumerator FallCo()
    {
        yield return new WaitForSeconds(fallDelay);
        rigidbody.isKinematic = false;
        GetComponent<BoxCollider2D>().isTrigger = true;
        yield return 0;
    }
}
```

FallingGroundInstantiation.cs

```
public class FallingGroundInstantiation : MonoBehaviour {

    public GameObject FallingGround;
    public GameObject SpawnLocation;
    Player player;
```

```
    int tempLifeCount;

    void Start ()
    {
        player = GameObject.FindGameObjectWithTag("Player").GetComponent<Player>();
        tempLifeCount = player.maxPlayerLifes;
    }

     void Update ()
    {
            if(tempLifeCount > player.currentPlayerLifes)
        {
            Instantiate(FallingGround, SpawnLocation.transform.position, Quaternion.identity);
            tempLifeCount = player.currentPlayerLifes;
        }
            }
}
```

HighScores.cs

```
public class HighScores : MonoBehaviour
{
    public GameObject myo = null;
    public GUISkin skin;
    public Texture buttonReturn;
    public Texture buttonExit;
    public Texture gestureFist;
    public Texture gestureFingersSpread;
    public Texture gestureWaveOut;
    public Texture menuBackgroundTexture;

    void OnGUI()
    {
        GUI.skin = skin;

        GUI.color = new Color(1, 1, 1, 0.5f);   // Lower Alpha to 0.5
        GUI.DrawTexture(new Rect(0, 0, Screen.width, Screen.height),
menuBackgroundTexture);
        GUI.color = Color.white; // Get Alpha back up

        Rect highScoresTitleRect = new Rect(Screen.width / 2 - 200, 100, 400, 150);
        Rect highScoresRect = new Rect(Screen.width / 2 - 400, 100, 800, 800);

        GUI.Label(highScoresTitleRect, "High Scores", "HighScores");
```

```
    //Best 5 scores
    if(PlayerPrefs.GetInt("HighScore1") != 0){
        GUI.Label(new Rect(highScoresRect.x + 300, highScoresRect.y + 200, 150, 30), "1 : 
" + PlayerPrefs.GetInt("HighScore1") + " by  Player " + 
PlayerPrefs.GetString("HighScore1Player"), "MenuItems");
            if (PlayerPrefs.GetInt("HighScore2") != 0)
            {
                GUI.Label(new Rect(highScoresRect.x + 300, highScoresRect.y + 250, 150, 30), 
"2 :   " + PlayerPrefs.GetInt("HighScore2") + " by Player " + 
PlayerPrefs.GetString("HighScore2Player"), "MenuItems");
                    if (PlayerPrefs.GetInt("HighScore3") != 0)
                    {
                        GUI.Label(new Rect(highScoresRect.x + 300, highScoresRect.y + 300, 150, 
30), "3 :   " + PlayerPrefs.GetInt("HighScore3") + " by Player " + 
PlayerPrefs.GetString("HighScore3Player"), "MenuItems");
                            if (PlayerPrefs.GetInt("HighScore4") != 0)
                            {
                                GUI.Label(new Rect(highScoresRect.x + 300, highScoresRect.y + 350, 
150, 30), "4 :   " + PlayerPrefs.GetInt("HighScore4") + " by Player " + 
PlayerPrefs.GetString("HighScore4Player"), "MenuItems");
                                    if (PlayerPrefs.GetInt("HighScore5") != 0)
                                    {
                                    GUI.Label(new Rect(highScoresRect.x + 300, highScoresRect.y + 
400, 150, 30), "5 :   " + PlayerPrefs.GetInt("HighScore5") + " by Player " + 
PlayerPrefs.GetString("HighScore5Player"), "MenuItems");
                                    }
                            }
                    }
            }
    }

    if (GUI.Button(new Rect(highScoresRect.x - 50, Screen.height - 150, 100, 100), 
buttonReturn))   // Return menu button
        returnToMenu();
    GUI.DrawTexture(new Rect(highScoresRect.x + 70, Screen.height - 130, 60, 60), 
gestureFist);    // Gesture fist icon next to play button

    if (GUI.Button(new Rect(Screen.width - 300, Screen.height - 40, 250, 40), "Reset 
HighScores"))
        ResetScores();
  }

  void Update()
  {
```

```
        ThalmicMyo thalmicMyo = myo.GetComponent<ThalmicMyo>(); // set myo

        if (thalmicMyo.pose == Pose.Fist)
            returnToMenu();
    }

    void returnToMenu()
    {
        ThalmicHub.Destroy(ThalmicHub.instance);    // destroy myo instance
        SceneManager.LoadScene("MainMenu");  // load 1st level
    }

    void ResetScores()
    {
        PlayerPrefs.SetInt("HighScore1", 0);
        PlayerPrefs.SetInt("HighScore2", 0);
        PlayerPrefs.SetInt("HighScore3", 0);
        PlayerPrefs.SetInt("HighScore4", 0);
        PlayerPrefs.SetInt("HighScore5", 0);
        PlayerPrefs.SetString("HighScore1Player", "");
        PlayerPrefs.SetString("HighScore2Player", "");
        PlayerPrefs.SetString("HighScore3Player", "");
        PlayerPrefs.SetString("HighScore4Player", "");
        PlayerPrefs.SetString("HighScore5Player", "");
        PlayerPrefs.SetInt("GameOverScore", 0);
    }
}
```

InstructionsManager.cs

```
public class InstructionsManager : MonoBehaviour
{
    public GameObject myo = null;

    public GUISkin skin;
    public Texture buttonReturn;
    public Texture gestureWaveOut;
    public Texture gestureFingersSpread;
    public Texture gestureFist;
    public Texture gestureWaveIn;
    public Texture gesturePan;
    public Texture gestureRotate;
    public Texture menuBackgroundTexture;
```

```
    void OnGUI()
    {
        GUI.skin = skin;
        GUI.color = new Color(1, 1, 1, 0.7f);   // Lower Alpha to 0.7
        GUI.DrawTexture(new Rect(0, 0, Screen.width, Screen.height),
menuBackgroundTexture);
        GUI.color = Color.white; // Get Alpha back up

        Rect highScoresTitleRect = new Rect(Screen.width / 2 - 200, 100, 400, 150);
        Rect highScoresRect = new Rect(Screen.width / 2 - 400, 100, 800, 800);

        GUI.Label(highScoresTitleRect, "Instructions", "HighScores");
        GUI.Label(new Rect(highScoresRect.x + 200, highScoresRect.y + 150, 300, 30), "While
in Game", "MenuItems");

        GUI.DrawTexture(new Rect(highScoresRect.x + 250, highScoresRect.y + 200, 60, 60),
gestureWaveIn);
        GUI.Label(new Rect(highScoresRect.x + 350, highScoresRect.y + 210, 150, 30), " Wave
In to Pause Game.", "MenuItems");

        GUI.DrawTexture(new Rect(highScoresRect.x + 250, highScoresRect.y + 300, 60, 60),
gestureRotate);
        GUI.Label(new Rect(highScoresRect.x + 350, highScoresRect.y + 310, 150, 30), "Rotate
clockwise to jump.", "MenuItems");

        GUI.DrawTexture(new Rect(highScoresRect.x + 250, highScoresRect.y + 400, 60, 60),
gesturePan);
        GUI.Label(new Rect(highScoresRect.x + 350, highScoresRect.y + 410, 150, 30), "Pan
Up and Down to move.", "MenuItems");

        GUI.DrawTexture(new Rect(highScoresRect.x + 250, highScoresRect.y + 500, 60, 60),
gestureFingersSpread);
        GUI.Label(new Rect(highScoresRect.x + 350, highScoresRect.y + 510, 150, 30),
"Spread Fingers to Interact with objects ", "MenuItems");

        //Buttons
        if (GUI.Button(new Rect(highScoresRect.x - 50, Screen.height - 150, 100, 100),
buttonReturn))    // Return menu button
            returnToMenu();
        GUI.DrawTexture(new Rect(highScoresRect.x + 70, Screen.height - 130, 60, 60),
gestureWaveOut);    // Gesture fist icon next to play button
    }

    void Update()
    {
```

```
        ThalmicMyo thalmicMyo = myo.GetComponent<ThalmicMyo>(); // set myo

        if (thalmicMyo.pose == Pose.WaveOut)
            returnToMenu();
    }

    void returnToMenu()
    {
        ThalmicHub.Destroy(ThalmicHub.instance);    // destroy myo instance
        SceneManager.LoadScene("MainMenu");  // load 1st level
    }
}
```

LevelSelManager.cs

```
public class LevelSelManager : MonoBehaviour
{
    public GameObject myo;      // Setting myo
    Pose _lastPose = Pose.Unknown;

    public GameObject[] locks;
    public bool[] levelUnlocked;

    public int positionSelector;

    float moveSpeed = 2;
    float distanceBelow = -0.74f;

    void Start()
    {
        for (int i = 1; i <=5; i++)
        {
            if(PlayerPrefs.GetInt("Player" + PlayerPrefs.GetInt("CurrentPlayer") + "Level") < i)
                levelUnlocked[i-1] = false;
            else
                levelUnlocked[i-1] = true;


            if (levelUnlocked[i-1])
                locks[i-1].SetActive(false);

        }
        positionSelector = PlayerPrefs.GetInt("Player" + PlayerPrefs.GetInt("CurrentPlayer")
+ "Level") - 1;
```

```
        transform.position = locks[positionSelector].transform.position + new Vector3(0,
distanceBelow, 0);
    }

    void Update()
    {
        ThalmicMyo thalmicMyo = myo.GetComponent<ThalmicMyo>();     // Set myo

        transform.position = Vector3.MoveTowards(transform.position,
locks[positionSelector].transform.position + new Vector3(0, distanceBelow, 0),
moveSpeed*Time.deltaTime);

        if (thalmicMyo.pose != _lastPose)
        {
            _lastPose = thalmicMyo.pose;    // if lastPose has changed get new one

            if (thalmicMyo.pose == Pose.WaveOut)
            {
                if (positionSelector < 4)
                {
                    if (levelUnlocked[positionSelector + 1])
                        positionSelector += 1;
                }
            }
            if (thalmicMyo.pose == Pose.WaveIn)
            {
                if (positionSelector > 0)
                    positionSelector -= 1;
            }
        }

        if (Input.GetKeyDown(KeyCode.RightArrow))
        {
            if(positionSelector < 4)
            {
                if (levelUnlocked[positionSelector + 1])
                    positionSelector += 1;
            }
        }

        if (Input.GetKeyDown(KeyCode.LeftArrow))
        {
            if (positionSelector > 0)
                positionSelector -= 1;
```

```
        }

        if(transform.position == locks[positionSelector].transform.position + new Vector3(0, distanceBelow, 0))
        {
            if (Input.GetKeyDown(KeyCode.Space) || thalmicMyo.pose == Pose.Fist)
            {
                if (levelUnlocked[positionSelector])
                {
                    ThalmicHub.Destroy(ThalmicHub.instance);    // destroy myo instance
                    SceneManager.LoadScene(positionSelector + 1);
                }
            }
        }
    }

    public GUISkin skin;

    public Texture gestureFistTexture;
    public Texture gestureFingersSpreadTexture;
    public Texture exitButtonTexture;


    void OnGUI()
    {
        //Pause Button while in Game
        if (GUI.Button(new Rect(Screen.width - 150, 50, 75, 75), exitButtonTexture, "PauseButton"))
        {
            goToMainMenu();
        }
        GUI.DrawTexture(new Rect(Screen.width - 60, 60, 50, 50), gestureFingersSpreadTexture);

        GUI.skin = skin;
        // Middle of screen Box for Menu
        Rect menuScreenRect = new Rect(Screen.width / 2 - 125, Screen.height / 2 - 250, 260, 400);

        if (transform.position == locks[positionSelector].transform.position + new Vector3(0, distanceBelow, 0))
        {
            if (levelUnlocked[positionSelector])
            {
```

```
            GUI.DrawTexture(new Rect(Screen.width / 2 - 60, 150, 120, 120),
gestureFistTexture);
        }
      }
   }

   void goToMainMenu()
   {
      ThalmicHub.Destroy(ThalmicHub.instance);
SceneManager.LoadScene("MainMenu");
   }
}
```

MainMenu.cs

```
public class MainMenu : MonoBehaviour
{
   public GameObject myo = null;
   Pose _lastPose = Pose.Unknown;

   public GUISkin skin;
   public Texture buttonPlay;
   public Texture buttonExit;
   public Texture buttonInstructions;
   public Texture buttonLevels;
   public Texture buttonHighScores;
   public Texture buttonCreateNewPlayer;
   public Texture gestureFist;
   public Texture gestureFingersSpread;
   public Texture gestureWaveRight;
   public Texture gestureWaveLeft;
   public Texture menuBackgroundTexture;

   int menuCounter = 0;

   void OnGUI()
   {
      GUI.skin = skin;
      GUI.color = new Color(1, 1, 1, 0.5f);   // Lower Alpha to 0.5
      GUI.DrawTexture(new Rect(0, 0, Screen.width, Screen.height),
menuBackgroundTexture);
      GUI.color = Color.white; // Get Alpha back up
```

```
    GUI.Label(new Rect(Screen.width / 2 - 250, Screen.height / 2 - 350, 500, 200), "The
Game");

    Rect firstRowLeftRect = new Rect(Screen.width / 2 - 50, Screen.height / 2 - 120, 100,
100);
    Rect firstRowTextRect = new Rect(Screen.width / 2 + 60, Screen.height / 2 - 50, 100,
50);
    Rect firstRowGestureRect = new Rect(Screen.width / 2 + 70, Screen.height / 2 - 110, 60,
60);

    Rect secondRowLeftRect = new Rect(Screen.width / 2 - 50, Screen.height / 2, 100, 100);
    Rect secondRowMidRect = new Rect(Screen.width / 2 + 60, Screen.height / 2 + 70, 100,
50);
    Rect secondRowRightRect = new Rect(Screen.width / 2 + 70, Screen.height / 2 + 10, 60,
60);

    Rect leftIconRect = new Rect(Screen.width / 2 - 200, Screen.height / 2 + 20, 60, 60);
    Rect leftGestureRect = new Rect(Screen.width / 2 - 250, Screen.height / 2 + 30, 40, 40);

    Rect rightIconRect = new Rect(Screen.width / 2 + 230, Screen.height / 2 + 20, 60, 60);
    Rect rightGestureRect = new Rect(Screen.width / 2 + 300, Screen.height / 2 + 30, 40,
40);



      if (GUI.Button(firstRowLeftRect, buttonPlay))
        NewGame();
      GUI.Label(firstRowTextRect, "Start Game", "MenuItems");


    if (menuCounter == 0) // levelSelection
    {
      if (GUI.Button(secondRowLeftRect, buttonLevels))    // Play button
        LevelSelectionPage();
      GUI.Label(secondRowMidRect, "Levels", "MenuItems");

      // Lower Opacity
      GUI.color = new Color32(255, 255, 255, 100);
      //left
      GUI.DrawTexture(rightIconRect, buttonExit);
      //right
      GUI.DrawTexture(leftIconRect, buttonHighScores);
      // Max opacity
      GUI.color = new Color32(255, 255, 255, 255);
```

```
}
else if (menuCounter == 1) // highScores
{
   if (GUI.Button(secondRowLeftRect, buttonHighScores)) // Exit game button
      HighScoresPage();

   GUI.Label(secondRowMidRect, "High Scores", "MenuItems");

   // Lower Opacity
   GUI.color = new Color32(255, 255, 255, 100);
   //left
   GUI.DrawTexture(rightIconRect, buttonLevels);
   //right
   GUI.DrawTexture(leftIconRect, buttonInstructions);
   // Max opacity
   GUI.color = new Color32(255, 255, 255, 255);

}
else if (menuCounter == 2) // Instructions
{
   if (GUI.Button(secondRowLeftRect, buttonInstructions)) // Instructions Button
   {
      InstructionsPage();
   }
   GUI.Label(secondRowMidRect, "Instructions", "MenuItems");

   // Lower Opacity
   GUI.color = new Color32(255, 255, 255, 100);
   //left
   GUI.DrawTexture(rightIconRect, buttonHighScores);
   //right
   GUI.DrawTexture(leftIconRect, buttonCreateNewPlayer);
   // Max opacity
   GUI.color = new Color32(255, 255, 255, 255);

}
else if (menuCounter == 3)
{
   if (GUI.Button(secondRowLeftRect, buttonCreateNewPlayer)) // Instructions Button
   {
      CreateNewPlayerPage();
   }
   GUI.Label(secondRowMidRect, "Choose Player", "MenuItems");
```

```
            // Lower Opacity
            GUI.color = new Color32(255, 255, 255, 100);
            //left
            GUI.DrawTexture(rightIconRect, buttonInstructions);
            //right
            GUI.DrawTexture(leftIconRect, buttonExit);
            // Max opacity
            GUI.color = new Color32(255, 255, 255, 255);
        }
        else    // exit
        {
            if (GUI.Button(secondRowLeftRect, buttonExit)) // Exit game button
            {
                ExitGame();
            }
            GUI.Label(secondRowMidRect, "Exit", "MenuItems");

            // Lower Opacity
            GUI.color = new Color32(255, 255, 255, 100);
            //left
            GUI.DrawTexture(rightIconRect, buttonCreateNewPlayer);
            //right
            GUI.DrawTexture(leftIconRect, buttonLevels);
            // Max opacity
            GUI.color = new Color32(255, 255, 255, 255);
         }

        GUI.DrawTexture(firstRowGestureRect, gestureFist);   // Gesture fist for continue / play
        GUI.DrawTexture(secondRowRightRect, gestureFingersSpread);    // Gesture
fingers_spread for menu options

        GUI.color = new Color32(255, 255, 255, 100);    // Lower Opacity
        GUI.DrawTexture(rightGestureRect, gestureWaveLeft);
        GUI.DrawTexture(leftGestureRect, gestureWaveRight);
        GUI.color = new Color32(255, 255, 255, 255);    // Max opacity

        GUI.Label(new Rect(Screen.width / 2 - 100, Screen.height - 100, 200, 50), "Current
Player : Player " + PlayerPrefs.GetInt("CurrentPlayer").ToString(), "MenuItems");

        if(GUI.Button(new Rect(Screen.width - 250, Screen.height - 40, 200, 40), "Hard Reset"))
            HardReset();
    }
```

```
void Update()
{
    ThalmicMyo thalmicMyo = myo.GetComponent<ThalmicMyo>(); // set myo

    if (thalmicMyo.pose != _lastPose)
    {
        _lastPose = thalmicMyo.pose;    // if lastPose has changed get new one


        if (thalmicMyo.pose == Pose.Fist)
            NewGame();

        if (thalmicMyo.pose == Pose.FingersSpread)
        {
            switch (menuCounter)
            {
                case 0: LevelSelectionPage(); break;
                case 1: HighScoresPage(); break;
                case 2: InstructionsPage(); break;
                case 3: CreateNewPlayerPage(); break;
                case 4: ExitGame(); break;
                default: ExitGame(); break;
            }
        }
        if (thalmicMyo.pose == Pose.WaveOut || Input.GetKeyDown(KeyCode.RightArrow))
        {
            if (menuCounter < 4)
                menuCounter++;
            else
                menuCounter = 0;
        }

        if (thalmicMyo.pose == Pose.WaveIn || Input.GetKeyDown(KeyCode.LeftArrow))
        {
            if (menuCounter > 0)
                menuCounter--;
            else
                menuCounter = 4;
        }
    }

    if (Input.GetKeyDown(KeyCode.RightArrow))
    {
        if (menuCounter < 4)
```

```
            menuCounter++;
        else
            menuCounter = 0;
    }
    if (Input.GetKeyDown(KeyCode.LeftArrow))
    {
        if (menuCounter > 0)
            menuCounter--;
        else
            menuCounter = 4;
    }
}

void NewGame()
{
    ThalmicHub.Destroy(ThalmicHub.instance);
    SceneManager.LoadScene(1);
}
void LevelSelectionPage()
{
    ThalmicHub.Destroy(ThalmicHub.instance);
    SceneManager.LoadScene("LevelSelection");
}

void HighScoresPage()
{
    ThalmicHub.Destroy(ThalmicHub.instance);
    SceneManager.LoadScene("HighScores");
}

void ExitGame()
{
    Application.Quit();
}

void InstructionsPage()
{
    ThalmicHub.Destroy(ThalmicHub.instance);
    SceneManager.LoadScene("Instructions");
}

void CreateNewPlayerPage()
{
    ThalmicHub.Destroy(ThalmicHub.instance);
```

```
        SceneManager.LoadScene("PlayerSelection");
    }

    void HardReset()
    {
        PlayerPrefs.SetInt("HighScore1", 0);
        PlayerPrefs.SetInt("HighScore2", 0);
        PlayerPrefs.SetInt("HighScore3", 0);
        PlayerPrefs.SetInt("HighScore4", 0);
        PlayerPrefs.SetInt("HighScore5", 0);
        PlayerPrefs.SetString("HighScore1Player", "");
        PlayerPrefs.SetString("HighScore2Player", "");
        PlayerPrefs.SetString("HighScore3Player", "");
        PlayerPrefs.SetString("HighScore4Player", "");
        PlayerPrefs.SetString("HighScore5Player", "");
        PlayerPrefs.SetInt("GameOverScore", 0);

        for (int i = 0; i< PlayerPrefs.GetInt("NumOfPlayers"); i++)
        {
            PlayerPrefs.SetInt("Player" + i + "Level", 0);
        }

        PlayerPrefs.SetInt("CurrentPlayer", 1);
        PlayerPrefs.SetInt("NumOfPlayers", 0);

        ThalmicHub.Destroy(ThalmicHub.instance);
        SceneManager.LoadScene("ChoosePlayer");
    }
}
```

MainMenuHighScoreSet.cs

```
public class MainMenuHighScoreSet : MonoBehaviour {

    bool highScoreOrdered = false;

        void Start ()
    {
        // Unlock Level 1 if not unlocked
        if(PlayerPrefs.GetInt("Player" + PlayerPrefs.GetInt("CurrentPlayer") + "Level") == 0)
            PlayerPrefs.SetInt("Player" + PlayerPrefs.GetInt("CurrentPlayer") + "Level", 1);

        int score = PlayerPrefs.GetInt("GameOverScore");
        string currentPlayer = PlayerPrefs.GetInt("CurrentPlayer").ToString();
```

```
        if (score > PlayerPrefs.GetInt("HighScore5"))
        {
            PlayerPrefs.SetInt("HighScore5", score);
            PlayerPrefs.SetString("HighScore5Player", currentPlayer);
        }

        if (score > PlayerPrefs.GetInt("HighScore4"))
        {
            PlayerPrefs.SetInt("HighScore5", PlayerPrefs.GetInt("HighScore4"));
            PlayerPrefs.SetInt("HighScore4", score);

            PlayerPrefs.SetString("HighScore5Player",
PlayerPrefs.GetString("HighScore4Player"));
            PlayerPrefs.SetString("HighScore4Player", currentPlayer);
        }

        if (score > PlayerPrefs.GetInt("HighScore3"))
        {
            PlayerPrefs.SetInt("HighScore4", PlayerPrefs.GetInt("HighScore3"));
            PlayerPrefs.SetInt("HighScore3", score);

            PlayerPrefs.SetString("HighScore4Player",
PlayerPrefs.GetString("HighScore3Player"));
            PlayerPrefs.SetString("HighScore3Player", currentPlayer);
        }

        if (score > PlayerPrefs.GetInt("HighScore2"))
        {
            PlayerPrefs.SetInt("HighScore3", PlayerPrefs.GetInt("HighScore2"));
            PlayerPrefs.SetInt("HighScore2", score);

            PlayerPrefs.SetString("HighScore3Player",
PlayerPrefs.GetString("HighScore2Player"));
            PlayerPrefs.SetString("HighScore2Player", currentPlayer);
        }

        if (score > PlayerPrefs.GetInt("HighScore1"))
        {
            PlayerPrefs.SetInt("HighScore2", PlayerPrefs.GetInt("HighScore1"));
            PlayerPrefs.SetInt("HighScore1", score);

            PlayerPrefs.SetString("HighScore2Player",
PlayerPrefs.GetString("HighScore1Player"));
```

```
            PlayerPrefs.SetString("HighScore1Player", currentPlayer);
        }

        PlayerPrefs.SetInt("GameOverScore", 0);
    }
}
```

Stairs.cs

```
public class Stairs : MonoBehaviour {

    public GameObject myo = null;
    Pose _lastPose = Pose.Unknown;

    GameObject player;
    public Transform destination;

    bool canTravel = false;

    public Texture gestureFingersSpread;
    public GUISkin skin;
    AudioSource source;

    void Start()
    {
        player = GameObject.FindGameObjectWithTag("Player");
        source = GetComponent<AudioSource>();
    }

    void Update()
    {
        ThalmicMyo thalmicMyo = myo.GetComponent<ThalmicMyo>();

        if (thalmicMyo.pose != _lastPose)
        {
            _lastPose = thalmicMyo.pose;

            if (canTravel)
            {
                if (thalmicMyo.pose == Pose.FingersSpread)
                {
                    source.Play();
                    player.transform.position = new Vector2(destination.position.x,
destination.position.y);
```

```
            }
        }
    }

    if (canTravel)
    {
        if (Input.GetKeyDown(KeyCode.LeftControl))
        {
            source.Play();
            player.transform.position = new Vector2(destination.position.x,
destination.position.y);
        }
    }
}

void OnTriggerEnter2D(Collider2D col)
{
    if (col.CompareTag("Player"))
        canTravel = true;
}

void OnTriggerExit2D(Collider2D col)
{
    if (col.CompareTag("Player"))
        canTravel = false;
}

void OnGUI()
{
    GUI.skin = skin;

    if (canTravel)
    {
        GUI.DrawTexture(new Rect(Screen.width / 2 - 50, Screen.height / 2 - 100, 50, 50),
gestureFingersSpread);
        GUI.Label(new Rect(Screen.width / 2 + 10, Screen.height / 2 - 90, 300, 50), "to
travel", "StairsInfo");
    }
}
}
```

LifeUpBox.cs

```
public class LifeUpBox : MonoBehaviour
```

```
{
    public GameObject Pizza;     //  Life Up Mushroom
    public GameObject SpawnLocation;

    void OnCollisionEnter2D(Collision2D collision)
    {
        if (collision.gameObject.CompareTag("Player"))
        {
            bool hitOncePerJump = true;
            foreach (ContactPoint2D point in collision.contacts)
            {
                if (point.normal.y >= 0.5f)
                {
                    if (hitOncePerJump)
                    {
                        Instantiate(Pizza, SpawnLocation.transform.position, Quaternion.identity);
                        Destroy(this);
                    }
                }
                hitOncePerJump = false;
            }
            hitOncePerJump = true;
        }
    }
}
```

PowerBox.cs

```
public class PowerBox : MonoBehaviour
{
    public GameObject Milk;     //  Magic Mushroom
    public GameObject Alcohol;    // Fireflower
    public GameObject SpawnLocation;

    void OnCollisionEnter2D(Collision2D collision)
    {
        if (collision.gameObject.CompareTag("Player"))
        {
            bool hitOncePerJump = true;

            foreach (ContactPoint2D point in collision.contacts)
            {
                if (point.normal.y >= 0.5f)
                {
```

```
            if (hitOncePerJump)
            {
                if (Player.playerStatus == 0)
                    Instantiate(Milk, SpawnLocation.transform.position, Quaternion.identity);
                else if (Player.playerStatus >= 1)
                    Instantiate(Alcohol, SpawnLocation.transform.position,
Quaternion.identity);

                Destroy(this);
            }
        }
        hitOncePerJump = false;
    }
    hitOncePerJump = true;
  }
 }
}
```

SimpleItemBox.cs

```
public class SimpleItemBox : MonoBehaviour {

  public GameObject Coin;    //  Coin
  public GameObject SpawnLocation;
  public int timesToTake;
  int timesTaken = 0;

  void OnCollisionEnter2D(Collision2D collision)
  {
    if (collision.gameObject.CompareTag("Player"))
    {
      bool hitOncePerJump = true;

      foreach (ContactPoint2D point in collision.contacts)
      {
        if (point.normal.y >= 0.5f)
        {
          if (hitOncePerJump)
          {
            if(timesTaken < timesToTake)
            {
              Instantiate(Coin, SpawnLocation.transform.position, Quaternion.identity);
              ScoreManager.AddPoints(25);
              timesTaken++;
```

```
                }
                else
                {
                    Destroy(this);
                }
            }
        }
        hitOncePerJump = false;
    }
    hitOncePerJump = true;
  }
 }
}
```