Τεχνολογικό Εκπαιδευτικό
Ίδρυμα Κρήτης

Σχολή Τεχνολογικών Εφαρμογών
Τμήμα Μηχανικών Πληροφορικής

Πτυχιακή Εργασία

Ανάπτυξη Εφαρμογής για την Εισαγωγή Δεδομένων από NUI Devices στην Πλατφόρμα Unity

Σφενδύλης Εμμανουήλ (ΑΜ: 1065)

Επιβλέπων καθηγητής : Βιδάκης Νικόλαος
Επιτροπή Αξιολόγησης :
Ημερομηνία παρουσίασης:

Ευχαριστίες

Στους γονείς μου οι οποίοι με στηρίζουν όλα αυτά τα χρόνια και στον καθηγητή μου κ.Βιδάκη Νικόλαο ο οποίος επιμελήθηκε αυτής της εργασίας.

Abstract

This work deals with the development of an application for the interaction between Microsoft's natural user interaction sensor, Kinect and Unity's game engine web player.

At the same time there is a reference to the most commonly used natural user interaction sensors that are currently available on the market as well as the most commonly used game engines.

We will also see the already existing ways available on the internet regarding the interaction of the Kinect sensor with Unity game engine and of course our own approach.

Σύνοψη

Η εργασία αυτή ασχολείται με τη δημιουργία μιας εφαρμογής για την αλληλεπίδραση του αισθητήρα φυσικής αλληλεπίδρασης χρήστη της Microsoft, Kinect με τον Web Player της μηχανής παιχνιδιών Unity.

Παράλληλα γίνεται μια αναφορά στους πιο διαδεδομένους αισθητήρες φυσικής αλληλεπίδρασης χρήστη που υπάρχουν αυτή τη στιγμή στην αγορά όπως επίσης και στις πιο διαδεδομένες μηχανές παιχνιδιών.

Θα δούμε επίσης τους ήδη διαθέσιμους τρόπους που υπάρχουν στο διαδίκτυο όσον αφορά τη σύνδεσή του Kinect με το Unity και φυσικά τη δική μας προσέγγιση.

Πίνακας Περιεχομένων

Πίνακας Εικόνων

Λίστα Μπλοκ Κώδικα

Εισαγωγή

Οι υπολογιστές υπάρχουν γύρω μας για περισσότερο από μισό αιώνα, και όμως ο τρόπος που οι περισσότεροι άνθρωποι αλληλεπιδρούν μαζί τους δεν έχει αλλάξει κατά πολύ. Τα πληκτρολόγια που οι περισσότεροι χρησιμοποιούμε έχουν εξελιχθεί από τις γραφομηχανές, μια τεχνολογία που χρονολογείται σχεδόν 150 χρόνια.

Παρ'όλα αυτά στις μέρες μας δεν χρειάζεται να κοιτάξουμε πολύ μακριά για να αντιληφθούμε ότι η τεχνολογία γίνεται κάτι όλο και πιο φυσικό και διαισθητικό. Σε μια τυπική ημέρα, πολλοί άνθρωποι χρησιμοποιούν την αφή ή την ομιλία για να αλληλεπιδρούν με τεχνολογικές συσκευές, τα κινητά τους τηλέφωνα, τα ATM, ακόμα και τα αυτοκίνητά τους. Οι υπολογιστές έχουν πλέον γίνει προσιτοί σε όλους και πιο εύχρηστοι χάρη σε πιο φυσικούς τρόπους για να αλληλεπιδράσουν οι άνθρωποι με αυτούς.

1.1 Αλληλεπίδραση Ανθρώπου-Υπολογιστή

Η αλληλεπίδραση ανθρώπου-υπολογιστή αποτελεί έναν από τους πιο σημαντικούς τομείς της πληροφορικής και της ανάπτυξης πληροφοριακών συστημάτων.

Είναι ο τομέας της πληροφορικής που ασχολείται με τις αρχές που διέπουν την επικοινωνία του ανθρώπου και του υπολογιστή και καθορίζουν τις αρχές σχεδιασμού, ανάπτυξης και αξιολόγησης των συστημάτων διεπαφής με τους χρήστες (user interfaces). Το σύστημα διεπαφής χρηστών είναι ένα από τα σημαντικότερα τμήματα των σύγχρονων συστημάτων λογισμικού είτε αυτά βρίσκονται σε μεμονωμένους υπολογιστές ή κινητά τηλέφωνα, είτε σε δίκτυα ή στο διαδίκτυο. Κατά συνέπεια ένα μεγάλο μέρος της προσπάθειας ανάπτυξης λογισμικού αφιερώνεται στην ανάπτυξη εύχρηστων συστημάτων διεπαφής χρηστών.

Η αλλελεπίδραση ανθρώπου-υπολογιστή ξεκίνησε με την χρήση του πληκτρολογίου για την είσοδο δεδομένων και προχώρησε ένα βήμα παραπέρα με την εφεύρεση του υπολογιστικού ποντικιού στα μέσα της δεκαετίας του '60. Το υπολογιστικό ποντίκι θα μπορούσαμε να πούμε ότι αποτελεί την αρχή της αποτύπωσης της φυσικής κίνησης του ανθρώπου στα τεχνολογικά μέσα, αφού ακολουθεί την κίνηση του χεριού και την αποτυπώνει στην οθόνη με την χρήση ενός κέρσορα. Η εισαγωγή δεδομένων έμεινε για πολλά χρόνια στάσιμη και επικεντρωμένη στην χρήση του ποντικιού και του πληκτρολογίου, έπειτα εξελίχθηκε με τις οθόνες αφής που προσέφεραν, πέρα από ένα εικονικό πληκτρολόγιο και την αντικατάσταση του ποντικιού με τα δάχτυλα ή την γραφίδα, ανίχνευση κινήσεων του χεριού (για παράδειγμα για την χρήση της εστίασης ή του scroll bar), καθώς και αισθητήρες που αντιλαμβάνονταν τη κλίση που έχουν οι κινητές συσκευές ως προς το έδαφος. Οι οθόνες αφής επέφεραν και αξιοσμείωτες αλλαγές στον τομέα της επεξεργασίας εικόνας.

Η φυσική αλληλεπίδραση χρήστη (Natural User Interaction – NUI) είναι η απόλυτη καινοτομία της τεχνολογίας που έρχεται στις μέρες μας να αναβαθμίσει την εμπειρία του χρήστη όσον αφορά την αλληλεπίδραση του με κάποια συσκευή. Είναι μία υποδειγματική αλλαγή που μας επιτρέπει να πάμε ένα βήμα παραπέρα από την απλή χειροκίνητη εισαγωγή δεδομένων και με την χρήση της οποίας η επικοινωνία με τις τεχνολογικές συσκευές γίνεται πλέον με απόλυτα φυσικό τρόπο.

1.2 Δομή Εργασίας

Στην εργασία αυτή θα μιλήσουμε αρχικά για τη φυσική αλληλεπίδραση χρήστη και τους πιο διαδεδομένους αισθητήρες φυσικής αλληλεπίδρασης χρήστη που υπάρχουν στην αγορά. Πιο συγκεκριμένα θα αναφερθούμε στο Kinect της Microsoft, το οποίο αποτελεί και αντικείμενο αυτής της εργασίας, το Leap Motion της ομώνυμης εταιρίας και το Myo Armband της ThalmicLabs.

Συνεχίζοντας θα αναφερθούμε στις μηχανές παιχνιδιών και ειδικότερα στη Unity, η οποία αποτελεί επίσης αντικείμενο της εργασίας αυτής, την Unreal Engine και την CryEngine, οι οποίες είναι από τις πιο δημοφιλείς μηχανές παιχνιδιών που υπάρχουν αυτή τη στιγμή.

Προχωρώντας θα μιλήσουμε λίγο για την αρχιτεκτονική λογισμικού και θα δούμε παράλληλα την αρχιτεκτονική του Kinect for Windows SDK το οποίο και αποτέλεσε βασικό εργαλείο της εργασίας αυτής.

Το επόμενο κεφάλαιο είναι αφιερωμένο στους ήδη υπάρχον τρόπους που υπάρχουν στο διαδίκτυο σχετικά με την σύνδεση του Kinect με το Unity.

Αυτοί είναι το KinectWrapper Package for Unity3D, το Kinect with MS-SDK By RF Solutions και το Zigfu ZDK.

Τέλος θα δούμε την δική μας υλοποίηση και τα στάδια τα οποία ακολουθήσαμε κατά τη διάρκειά της υλοποίησης καθώς επίσης και τα όποια συμπεράσματα αποκομίσαμε από αυτή.

State Of The Art

2.1 Natural User Interaction (NUI)

Ο όρος φυσική αλληλεπίδραση χρήστη δεν σχετίζεται με το πληκτρολόγιο ή το υπολογιστικό ποντίκι, αλλά έρχεται να αντικαταστήσει αυτές τις συσκευές ως ενδιάμεσες, με τις φυσικές λειτουργίες του ανθρώπου. Όταν αναφερόμαστε στις φυσικές λειτουργίες του ανθρώπου εννοούμε την αφή, την κίνηση, την ομιλία, ακόμα και την σκέψη. Η NUI κάνει τις φυσικές λειτουργίες να αλληλεπιδρούν άμεσα με τις ηλεκτρονικές συσκευές, κάτι που επιτυγχάνεται με την χρήση των κατάλληλων συσκευών και αισθητήρων, καθώς και την χρήση κατάλληλων συναρτήσεων που ερμηνεύουν τις φυσικές λειτουργίες ώστε να τις αποτυπώσουν στο εικονικό περιβάλλον.

Η φυσική αλληλεπίδραση χρήστη επιτυγχάνεται μέσω ενός αόρατου και συνήθως πολύπλοκου περιβάλλοντος διεπαφής, με το οποίο ο χρήστης εξοικειώνεται γρήγορα καθώς αυτό αντιπροσωπεύει τις ανθρώπινες ικανότητες, με άλλα λόγια όλοι οι χρήστες είναι έμπειροι για τον απλούστατο λόγο ότι είναι άνθρωποι. Η αλληλεπίδραση του χρήστη με το περιβάλλον διεπαφής είναι συνεχώς επιτυχημένη και σχεδόν απαλλαγμένη από μηνύματα λάθους, πράγμα που κάνει την εμπειρία πιο ευχάριστη και ξεκούραστη.

Η χρήση της εφαρμόζεται συχνά σε παιχνίδια ή σε περιβάλλοντα εκμάθησης με χρήση παιχνιδιών (χωρίς η εφαρμογή της να περιορίζεται μόνο σε αυτά), αποτυπώνοντας την φυσική λειτουργία της ανθρώπινης κίνησης και δίνοντας την υποκειμενική αίσθηση της παρουσίας του χρήστη στο γραφικό περιβάλλον. Κατά συνέπεια μιλάμε για μια τεχνική που αναβαθμίζει την ψυχαγωγική δραστηριότητα χωρίς να κουράζει τον χρήστη με την εκμάθηση του περιβάλλοντος και φέρνει μεγάλες ανατροπές στον τομέα του σχεδιασμού του περιβάλλοντος του χρήστη.

Για να φτάσουμε στο σημείο να έχουμε φυσικές διεπαφές χρήστη πολλοί μελετητές έχουν αναλύσει τις ανθρώπινες λειτουργίες σε δυαδική κωδικοποίηση με στόχο να ανιχνεύεται η ταχύτητα της κίνησης, η δύναμη, η ένταση και άλλοι παράγοντες που βοηθούν στην καλύτερη κατανόηση των λειτουργικών ικανοτήτων από τα μέσα τεχνολογίας.

Η φυσική αλληλεπίδραση χρηστών αποτελούσε πάντα ένα όραμα της εξέλιξης της τεχνολογίας. Μια σειρά παραγόντων όμως καθυστέρησε τόσο πολύ την εμφάνιση της. Κάποιοι από τους παράγοντες αυτούς ήταν η ανεπαρκής υπολογιστική ισχύς (οι αισθητήρες NUI έχουν μεγάλες απαιτήσεις υπολογιστικής ισχύος), το υψηλό κόστος και η έλλειψη σε πλατφόρμες και βιβλιοθήκες που να υποστηρίζουν τον προγραμματισμό τους.

Ολοένα και περισσότεροι μελετητές απασχολούνται πλέον με το αντικείμενο της φυσικής αλληλεπίδρασης χρήστη. Το αντικείμενο αυτό είναι αναμφισβήτητα το μέλλον της τεχνολογίας και τα τελευταία χρόνια προσπαθεί να ασχοληθεί, πέρα από την αποτύπωση της ομιλίας, της κίνησης και της σκέψης, και με την απεικόνιση των συναισθημάτων. Είναι ο κλάδος που ενώνει τις επιστήμες της Πληροφορικής, της Ηλεκτρονικής, της Φυσικής αλλά και της Ιατρικής και της Ψυχολογίας καθοριστικά.

2.2 Natural User Interaction Sensors

Οι αισθητήρες φυσικής αλληλεπίδρασης χρήστη ανιχνεύουν την συμπεριφορά των χρηστών, την ομιλία ή ακόμα και τους ίδιους τους χρήστες και τους δίνουν την δυνατότητα να αλληλεπιδρούν με τους υπολογιστές ή τις κονσόλες παιχνιδιών, χωρίς να χρησιμοποιήσουν το πληκτρολόγιο, το ποντίκι ή άλλα χειριστήρια. Χάρις σε αυτές τις συσκευές οι χρήστες μπορούν να ελέγξουν τις υπολογιστικές συσκευές χωρίς να χρειαστεί να τις αγγίξουν ή να έχουν άμεση επαφή με τα χέρια τους, για αυτό αποκαλούνται και «touchless».

Οι αισθητήρες αυτοί επιτυγχάνουν την ανίχνευση χειρονομιών, την αναγνώριση χρήστη, την ανίχνευση ματιού ή προσώπου, την αναγνώριση ομιλίας και άλλα, με στόχο την κατανόηση των συμπεριφορών ώστε να δώσουν στους χρήστες την ευκαιρία να αλληλεπιδράσουν με τις υπολογιστικές συσκευές πιο φυσικά και αποτελεσματικά. Χρησιμοποιούν μικρόφωνα, κάμερες, υπέρυθρες, laser και μικροτσίπ για την ανίχνευση συμπεριφορών και κινήσεων και τις κάνουν κατανοητές προς τα μηχανήματα εισάγοντας διακριτές τιμές (0 και 1) ή ανιχνεύοντας μεγέθη όπως η δύναμη και η ταχύτητα. Η κίνηση μπορεί να ανιχνευθεί από τον ήχο με αισθητήρες ήχου, από την αδιαφάνεια με οπτικούς και υπέρυθρους αισθητήρες και επεξεργαστές εικόνας και βίντεο, από τον γεωμαγνητισμό με μαγνητικούς αισθητήρες και μαγνητόμετρα, από την αντανάκλαση της μεταφερόμενης ενέργειας με υπέρυθρα laser ραντάρ, αισθητήρες υπερήχων και αισθητήρες ραντάρ μικροκυμάτων, και από την ηλεκτρομαγνητική επαγωγή και τους κραδασμούς.

Μοναδικά χαρακτηριστικά που έχουν οι αισθητήρες που προορίζονται για την φυσική αλληλεπίδραση χρήστη είναι τα εξής:

- Υψηλό εύρος ζώνης: Διαθέτουν κάμερες που μπορούν να μεταδώσουν πολλά gigabits ανα δευτερόλεπτο(πάνω από 3Gb/s)
- Πολλαπλοί αισθητήρες: Διαθέτουν πολλαπλές κάμερες, πολλαπλά μικρόφωνα, γυροσκόπιο, επιταχυνσιόμετρο, μαγνητόμετρο, θερμόμετρο, GPS και άλλους αισθητήρες.
- Χαμηλή λανθάνουσα κατάσταση: Ο στόχος είναι να εισάγουν τα δεδομένα των ανθρώπινων κινήσεων και λειτουργιών σε πραγματικό χρόνο.
- Συνεχής Κύκλος: Οι αισθητήρες NUI πρέπει να είναι ενεργοί όλη την ώρα.

2.2.1 Microsoft Kinect

Το Kinect είναι ένα περιφερειακό αναγνώρισης κίνησης σχεδιασμένο από την Microsoft για της κονσόλες παιχνιδιών Xbox 360 και Xbox One και για προσωπικούς υπολογιστές με λειτουργικό σύστημα Windows 7 ή 8. Επιτρέπει στους χρήστες να ελέγχουν και να αλληλεπιδρούν με τις κονσόλες τους ή τον υπολογιστή τους χωρίς την χρήση κάποιου χειριστηρίου, μέσω της φυσικής διεπαφής χρήστη με την χρήση χειρονομιών και προφορικών εντολών.

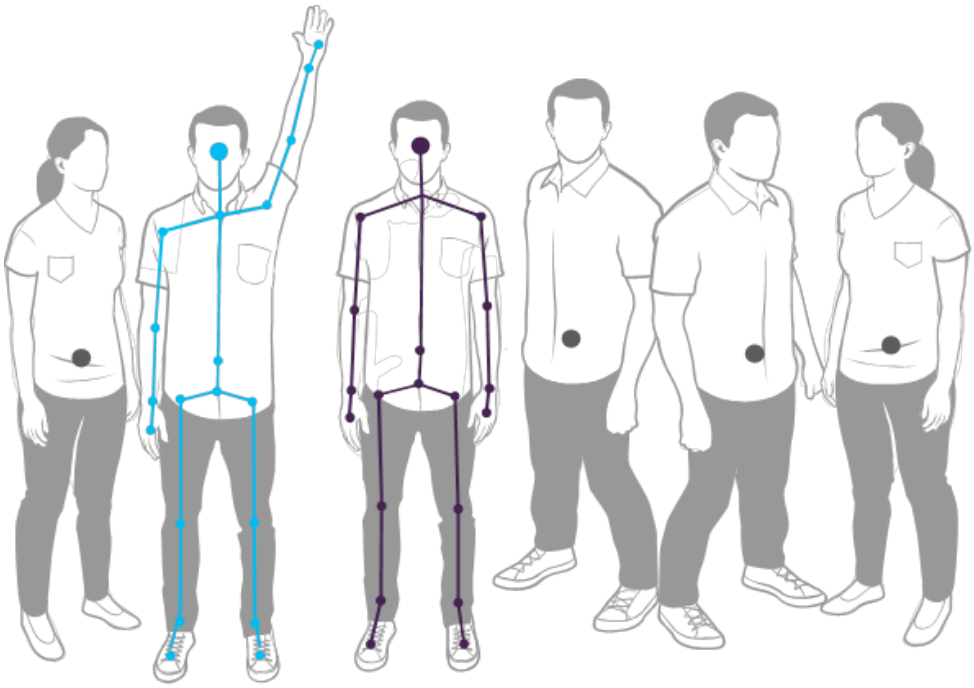
Εικόνα 2.1: Kinect sensor

Κυκλοφόρησε για πρώτη φορά τον Νοέμβριο του 2010 για τις κονσόλες Xbox, με στόχο να διευρύνει το κοινό τους, ενώ το 2012 εισήχθη στην αγορά το Kinect για Windows. Τον Ιούνιο του 2011 κυκλοφόρησε το Kinect SDK, που επιτρέπει στους developers να γράφουν Kinecting εφαρμογές, εφαρμογές οι οποίες χρησιμοποιούν το Kinect δηλαδή, σε C++, C#, ή Visual Basic.

Το Kinect βασίζεται σε τεχνολογίες λογισμικού που αναπτύσσονται από την θυγατρική εταιρεία της Microsoft, Rare και την τεχνολογία της ισραηλινής PrimeSense πάνω στις κάμερες και τις συσκευές λήψης βίντεο. Η PrimeSense ανέπτυξε μία τεχνολογία που μπορεί να αντιληφθεί χειρονομίες και καθιστά δυνατό τον έλεγχο της συσκευής με την χρήση μιας υπέρυθρης κάμερας, δύο φακών και ενός ειδικού μικροτσίπ για την παρακολούθηση της κίνησης των αντικειμένων και των ατόμων σε τρεις διαστάσεις.

Ο αισθητήρας Kinect είναι μια οριζόντια μπάρα που συνδέεται με μία βάση με μηχανοκίνητο άξονα περιστροφής και τοποθετείται κατά μήκος της οθόνης προβολής στο πάνω ή στο κάτω μέρος. Η συσκευή διαθέτει μια κάμερα RGB, έναν αισθητήρα βάθους(που είναι συνδυασμός δύο φακών), και μια συστοιχία μικροφώνων για την αναγνώριση φωνής. Αυτά σε συνδυασμό με το λογισμικό που περιλαμβάνει το Microsoft Kinect SDK, πλήρη καταγραφή της κίνησης του σώματος σε τρεις διαστάσεις, αναγνώριση προσώπου και δυνατότητες αναγνώρισης φωνής. Η συστοιχία μικροφώνων του Kinect δίνει την δυνατότητα στο Xbox 360 να εντοπίζει την ακουστική πηγή και κάνει εφικτή την καταστολή θορύβου του περιβάλλοντος. Ο αισθητήρας βάθους αποτελείτε από έναν υπέρυθρο προβολέα laser σε συνδυασμό με έναν αισθητήρα CMOS, ο οποίος καταγράφει δεδομένα τρισδιάστατου βίντεο κάτω από οποιεσδήποτε συνθήκες φωτισμού. Η απόσταση ανίχνευσης του αισθητήρα βάθους είναι ρυθμιζόμενη και το λογισμικό του Kinect είναι σε θέση να βαθμονομήσει αυτόματα τον αισθητήρα με βάση το φυσικό περιβάλλον του χρήστη, ανεξάρτητα από την παρουσία άλλων αντικειμένων στον χώρο.

Η συσκευή μπορεί να αναγνωρίσει έως 6 άτομα που βρίσκονται στο περιβάλλον ταυτόχρονα μεταξύ των οποίων δύο ενεργούς παίκτες των οποίων αναλύει την κίνηση με δυνατότητα αντίληψης 20 κινήσεων ανά παίκτη. Οι κινήσεις των υπόλοιπων παικτών δεν επηρεάζουν το σύστημα κίνησης ακόμα και αν κινούνται στην γωνία ανίχνευσης.



2.2.2 Leap Motion

Η Leap Motion είναι μια αμερικάνικη εταιρεία που κατασκευάζει τον Leap Motion controller. Ο Leap Motion controller είναι μια συσκευή αισθητήρα φυσικής αλληλεπίδρασης χρήστη, ανάλογη με το υπολογιστικό ποντίκι, που υποστηρίζει την είσοδο δεδομένων στον υπολογιστή χωρίς άγγιγμα ή επαφή, δηλαδή απελευθερώνει τα χέρια του χρήστη.

Η τεχνολογία Leap Motion αναπτύχθηκε για πρώτη φορά το 2008, από τον David Holz, ο οποίος ίδρυσε την εταιρεία σε συνεργασία με τον παιδικό του φίλο Micheal Buckward, χάρις σε μία ισχυρή επένδυση. Το 2010 για πρώτη φορά η εταιρεία έβγαλε ανακοίνωση για το πρώτο της προϊόν που αρχικά ονομαζόταν «The Leap». Στο πλαίσιο της ανάπτυξης εφαρμογών για το προϊόν, κυκλοφόρησε περίπου 12.000 συσκευές για τους προγραμματιστές που ενδιαφέρονταν να δημιουργήσουν εφαρμογές για το προϊόν. Σύντομα οι πωλήσεις εκτοξεύτηκαν και η εταιρεία κυκλοφόρησε την δεύτερη έκδοση λογισμικού για προγραμματιστές το 2014. Η εταιρεία συνεργάζεται με την ASUS και την HP για να ενσωματώσει την τεχνολογία της σε μιά νέα γενιά υπολογιστών, ακόμα και σε πληκτρολόγια.

Εικόνα 2.2: Leap Motion controller

Ο Leap Motion controller είναι μία περιφερειακή συσκευή USB μικρή σε μέγεθος που έχει σχεδιαστεί για να τοποθετείτε σε μια φυσική επιφάνεια εργασίας, στραμμένη προς τα πάνω. Χρησιμοποιεί δύο μονοχρωματικές IR κάμερες και τρεις υπέρυθρες LED. Η περιοχή που καλύπτει η ανίχνευση της συσκευής είναι ημισφαιρική και φτάνει το ένα μέτρο. Τα LED δημιουργούν ένα 3D μοτίβο από κουκκίδες φωτός IR και οι κάμερες παράγουν σχεδόν 300 καρέ ανα δευτερόλεπτο, τα οποία αποστέλλονται μέσω ενός καλωδίου USB στον υπολογιστή που είναι συνδεδεμένη η συσκευή. Εκεί αναλύονται από το λογισμικό του ελεγκτή Leap Motion με χρήση πολύπλοκων μαθηματικών μεθόδων, οι οποίες δεν έχουν αποκαλυφθεί από την εταιρεία.

Το μικρότερο μέγεθος της συσκευής Leap Motion και η μεγαλύτερη ανάλυση είναι αυτά που την διαφοροποιούν από το Microsoft Kinect. Το Kinect θεωρείται καλύτερο για παρακολούθηση ολόκληρου του σώματος σε μεγαλύτερο χώρο ενώ το Leap Motion controller είναι πολύ αποτελεσματικό στην παρακολούθηση των κινήσεων των χεριών, ακόμα και των δαχτύλων. Η ανίχνευση κινήσεων γίνεται με ακρίβεια εκατοστού του χιλιοστού. Η συσκευή είναι συμβατή με τα λειτουργικά συστήματα Windows και OS X.

Η συσκευή αυτή δεν έχει καταφέρει ακόμα να αντικαταστήσει το ποντίκι ή το πληκτρολόγιο, αλλά χρησιμοποιείται συνδυαστικά με αυτές. Η τεχνολογία του όπως ισχυρίζονται τα στελέχη της εταιρείας μπορεί πέρα από την βιομηχανία του gaming να εφαρμοστεί και σε άλλους τομείς, όπως στην χειρουργική, την αρχιτεκτονική, την μηχανική και τον σχεδιασμό.

Οι χρήστες της συσκευής μπορούν να προμηθευτούν εφαρμογές από την σελίδα της Leap Motion https://apps.leapmotion.com/ που αυτή την στιγμή απαριθμεί 219 εφαρμογές εκ των οποίων οι 93 παρέχονται δωρεάν.

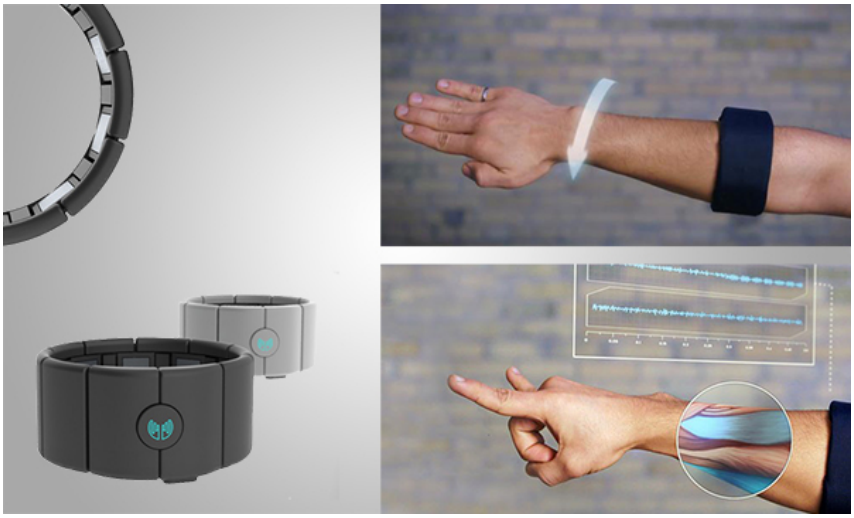
Αξιοσημείωτες είναι οι αξιολογήσεις για το Leap Motion οι οποίες ενώ επιβραβεύουν την τεχνολογία του, είναι συγκρατημένες, καθώς η συσκευή δεν αντικαθιστά το ποντίκι και το πληκτρολόγιο και περιορίζεται ακόμα στην χρήση συγκεκριμένων εφαρμογών. Επομένως η χρησιμότητά του είναι ακόμα αμφιλεγόμενη.

2.2.3 ThalmicLabs Myo Armband

Η ThalmicLabs είναι μια νεοσύστατη εταιρεία με έδρα το Βατερλώ. Ιδρύθηκε το 2012 από τρεις απόφοιτους της Μηχανοτρονικής του Πανεπιστημίου του Βατερλώ, τον Stephen Lake, τον Matthew Bailey και τον Aaron Grant. Η εταιρεία έχει αναπτυχθεί σε μια παγκόσμια ομάδα της οποίας ηγούνται επιστήμονες, μηχανικοί και σχεδιαστές και στόχος της είναι να αλλάξει τον τρόπο με τον οποίο ο άνθρωπος αλληλεπιδρά με τις ηλεκτρονιές συσκευές. Με τα προϊόντα της προσπαθεί να συνδέσει τον πραγματικό με τον ψηφιακό κόσμο. Το προϊόν στο οποίο βασίζεται η ίδρυση της είναι το Myo Armband.

Εικόνα 2.3: Myo Armband

Το Myo είναι ένα καινοτόμο περιβραχιόνιο ελέγχου χειρονομιών, που χρησιμοποιεί το Bluetooth 4.0 για την επικοινωνία του με τις συσκευές που είναι συνδεδεμένο. Δίνει στους χρήστες τον έλεγχο των παρουσιάσεων, των βίντεο, του περιεχομένου, των παιχνιδιών και πολλών άλλων χωρίς αυτοί να χρειαστεί να αγγίξουν οθόνες, πληκτρολόγια ή ποντίκια, απελευθερώνοντας τα χέρια. Το slogan του προϊόντος είναι: «Οι μύες σας μιλάνε, το Myo Armband ακούει».



Οι αισθητήρες που χρησιμοποιεί είναι αισθητήρες μυών ανοξείδωτου χάλυβα EMG εγκεκριμένοι ιατρικά και ένας αισθητήρας κίνησης, που είναι μία υψηλής ευαισθησίας αδρανιακή μονάδα μέτρησης 9 αξόνων που περιλαμβάνει τριαξονικό γυροσκόπιο, τριαξονικό επιταχυνσιόμετρο και τριαξονικό μαγνητόμετρο. Οι αισθητήρες ανιχνεύουν μικρές κινήσεις των μυών, περιστροφές του πήχη και ακόμα και ερεθίσματα όπως οι χειρονομίες. Οι ανιχνεύσεις μεταφέρονται στην υπολογιστική πηγή ως δεδομένα εισόδου μέσω Bluetooth και μεταφράζονται στην γλώσσα του υπολογιστή με το λογισμικό της εταιρίας. Είναι εντυπωσιακό το γεγονός ό,τι μπορεί να ανιχνεύσει κινήσεις πριν προλάβει ο χρήστης να κάνει την κίνηση, λόγω του ότι αντιλαμβάνεται τις κινήσεις μέσω των μυών.

Διαθέτει έναν επεξεργαστή Arm Cordex 4, επαναφορτιζόμενες μπαταρίες ιόντων λιθίου που κρατάνε για μία μέρα μετά από πλήρη φόρτιση. Επιδρά στον χρήστη με δονήσεις μικρής, μεσαίας και μεγάλης διάρκειας. Για την σύνδεση με την ηλεκτρονική συσκευή χρησιμοποιείτε ένας Bluetooth Adapter που συνδέεται μέσω USB.

Τα συμβατά λειτουργικά συστήματα με το Myo Armband είναι:

- Windows 7
- Windows 8

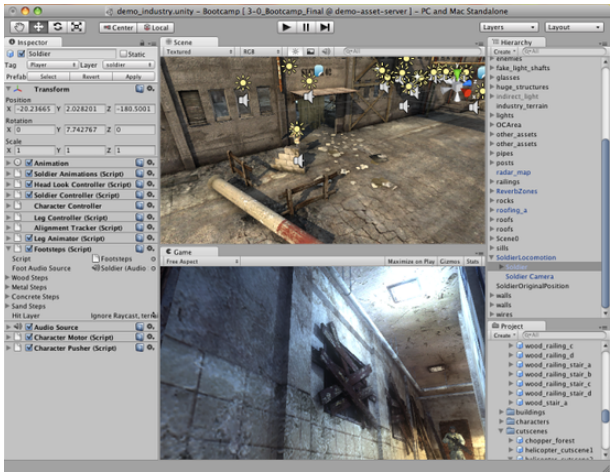
- OS X 10.8 (Mountain Lion)
- iOS έκδοση 7.0 και πάνω
- Android 4.3 (Jelly Bean) και πάνω.

Βασική προϋπόθεση των συσκευών με τις οποίες συνδέεται το Myo είναι να διαθέτουν Bluetooth 4.0.

Η εταιρία ετοιμάζει ένα ηλεκτρονικό κατάστημα από όπου οι χρήστες θα προμηθεύονται εφαρμογές για το Myo.

2.3 Game Engines

Μία μηχανή παιχνιδιών είναι ένα σύστημα λογισμικού σχεδιασμένο για την δημιουργία και την ανάπτυξη βιντεοπαιχνιδιών. Οι προγραμματιστές χρησιμοποιούν τις μηχανές παιχνιδιών για να δημιουργήσουν βιντεοπαιχνίδια για κονσόλες παιχνιδιών, κινητά τηλέφωνα ή υπολογιστές. Θα μπορούσαμε να ορίσουμε την μηχανή παιχνιδιών ως την οργάνωση του κώδικα που επιτρέπει των διαχωρισμό των γενικών λειτουργιών ενός παιχνιδιού από τις πιο εξειδικευμένες λειτουργίες. Είναι σχεδιασμένες για να λειτουργούν σε λειτουργικά συστήματα όπως τα Windows, το Linux και το OS X. Η λειτουργικότητα του πυρήνα που παρέχεται τυπικά από μια παιχνιδομηχανή περιλαμβάνει μια μηχανή φωτοαπόδοσης για δισδιάστατα ή τρισδιάστατα γραφικά, που στα αγγλικά αποκαλείται "renderer", μία φυσική μηχανή ή εντοπισμού συγκρούσεων (με ανίχνευση σύγκρουσεων και απόκριση συγκρούσεων), ήχο, scripting, κινούμενα γραφικά σχέδια, τεχνητή νοημοσύνη, δικτύωση, streaming, διαχείριση μνήμης, threading, υποστήριξη localization, και έναν γράφο σκηνής. Μία μηχανή παιχνιδιού μπορεί να χρησιμοποιηθεί για να δημιουργηθούν πολλά διαφορετικά παιχνίδια ή να διευκολύνει ένα παιχνίδι να υποστηριχθεί από διαφορετικές πλατφόρμες.



Εικόνα 2.4: Unity editor

Ο ήχος είναι ένα αναπόσπαστο κομμάτι των μηχανών παιχνιδιών. Οι μηχανές παιχνιδιών προσφέρουν ηχητικά εφέ, αυξομειώσεις της έντασης του ήχου και αντήχηση, γεγονός που κάνει τα παιχνίδια πιο ρεαλιστικά και έτσι ο ήχος των παιχνιδιών δεν βασίζεται μόνο στις ρυθμίσεις των ηχείων. Χάρις στην δικτύωση μπορούν να δημιουργηθούν online παιχνίδια με πολλαπλούς χρήστες και γίνεται εφικτή η επικοινωνία από διαφορετικούς servers και υπολογιστές. Με τον όρο φυσική μηχανή, αναφερόμαστε σε έναν μηχανισμό που κάνει τα παιχνίδια να μοιάζουν ρεαλιστικά. Κάποιες μηχανές φυσικής που συνεργάζονται με τις σύγχρονες παιχνιδομηχανές είναι η Havok, η Box2d και η PhysX και χειρίζονται πολύπλοκα μαθηματικά για την ρεαλιστική προσομοίωση του πραγματικού κόσμου. Μία άλλη σημαντική παροχή των μηχανών παιχνιδιών είναι τα σενάρια που διαθέτουν και

συνδέονται με τα αντικείμενα στον κόσμο του παιχνιδιού, έτσι ελέγχονται οι συμπεριφορές των αντικειμένων απαλλάσσοντας τον προγραμματιστή από αυτή την διαδικασία. Η χρήση της τεχνητής νοημοσύνης αποτελεί ένα μεγάλο πλεονέκτημα στην χρήση σεναρίων και κάνει την δουλειά του προγραμματιστή πολύ πιο γρήγορη.

Ένα από τα μεγαλύτερα πλεονεκτήματα της χρήσης μηχανών παιχνιδιών είναι ότι προσφέρονται εργαλεία έτσι ώστε ο χρήστης να μην χρειάζεται να ξεκινήσει από το χαμηλότερο επίπεδο δημιουργίας γραφικών για την υλοποίηση των εφαρμογών του. Έτσι επιτυγχάνεται μείωση της πολυπλοκότητας, του χρόνου και του κόστους κατασκευής. Οι μηχανές παρέχουν συνήθως τεχνικές για να μπορούν τα παιχνίδια να τρέξουν και σε άλλες πλατφόρμες (κονσόλες, υπολογιστές ή κινητά) με λίγες ακόμα γραμμές κώδικα αν όχι καθόλου. Είναι επεκτάσιμες και αντικαταστάσιμες αφού επιτρέπουν την επέκταση και την αντικατάσταση των συστημάτων τους με πιο εξειδικευμένα όπως τις μηχανές φυσικής που προαναφέραμε, το FMOD για τον ήχο, ή το Scaleform για το UI και το βίντεο.

Πέρα από την κατασκευή παιχνιδιών οι μηχανές αυτές χρησιμοποιούνται και για παραγωγή λογισμικών άλλων ειδών με γραφικές απαιτήσεις, όπως είναι οι επιδείξεις marketing, οι αρχιτεκτονικές οπτικοποιήσεις, οι εκπαιδευτικές εξομοιώσεις και τα περιβάλλοντα μοντελοποίησης.

Πολλές παιχνιδομηχανές παρέχουν μόνο ικανότητες rendering 3D πραγματικού χρόνου, αυτές ονομάζονται μηχανές γραφικών, μηχανές rendering ή μηχανές 3D. Το rendering είναι η διαδικασία δημιουργίας από ένα δισδιάστατο ή τρισδιάστατο μοντέλο με την βοήθεια προγραμμάτων ηλεκτρονικών υπολογιστών.

Συνήθως οι μηχανές παιχνιδιών κατασκευάζονται πάνω σε ένα API γραφικών, όπως τα Direct3d ή OpenGL. Χρησιμοποιούν βιβλιοθήκες χαμηλού επιπέδου όπως DirectX, SDL και OpenAL ώστε να παρέχεται πρόσβαση ανεξάρτητη από το υλικό σε άλλο υλικό υπολογιστή, όπως συσκευές εισόδου, κάρτες δικτύου και κάρτες ήχου.

Ο όρος μηχανή παιχνιδιών ήρθε στην επιφάνεια στα μέσα της δεκαετίας του '90, ειδικά για παιχνίδια FPS (First-person Shooters. Τα παιχνίδια Doom και Quake είχαν τόση μεγάλη δημοτικότητα που άλλοι δημιουργοί, αντί να ξεκινήσουν την κατασκευή παιχνιδιών από το μηδέν, πήραν άδεια για την χρήση των κεντρικών κομματιών του λογισμικού τους, και σχεδίασαν τα δικά τους γραφικά, χαρακτήρες και όπλα για το περιεχόμενου των παιχνιδιών τους.

Οι σύγχρονες μηχανές παιχνιδιών θεωρούνται (και όχι άδικα) από τις πολυπλοκότερες εφαρμογές για το γράψιμο κώδικα. Χαρακτηρίζονται ως τελειοποιημένα συστήματα που εγγυούνται μια ακριβή και ελεγχόμενη εμπειρία χρήστη. Πλέον μια τυπική ομάδα ανάπτυξης εφαρμογών παιχνιδιών αποτελείται από περισσότερους σχεδιαστές παρά προγραμματιστές.

Οι περισσότερες σύγχρονες μηχανές παιχνιδιών κατασκευάζονται σε γλώσσες προγραμματισμού υψηλού επιπέδου όπως η java, η C# .NET και η Python.

2.3.1 Unity

Το Unity είναι μία από τις πιο γνωστές μηχανές παιχνιδιών, ανεξαρτήτου πλατφόρμας (cross-platform), δημιουργημένη από την εταιρία Unity Technologies. Η Unity Technologies ιδρύθηκε το 2004 από τον David Helgason, τον Nicholas Francis και τον Joachim Ante στην Κοπεγχάγη της Δανίας. Οι τρεις του αναγνώρισαν την σημαντικότητα της ύπαρξης παιχνιδομηχανών και αποφάσισαν να κατασκευάσουν μια παιχνιδομηχανή που να παρέχεται σε προσιτή τιμή. Το 2008 με την άνοδο του iPhone το Unity ήταν η πρώτη εφαρμογή που υποστήριξε την δημιουργία εφαρμογών για την πλατφόρμα εξ ολοκλήρου. Πλέον περισσότερα από το 50% των παιχνιδιών για κινητά τηλέφωνα Android και iPhone είναι κατασκευασμένα με το Unity.

Εικόνα 2.5: Unity logo

Το Unity χρησιμοποιείται για την δημιουργία βιντεοπαιχνιδιών για ιστότοπους, ηλεκτρονικούς υπολογιστές, κονσόλες παιχνιδιών και για κινητά τηλέφωνα. Κυκλοφόρησε αρχικά για τα λειτουργικά συστήματα Mac OS και έπειτα επεκτάθηκε σε πάνω από 15 πλατφόρμες. Προσφέρει την δυνατότητα στους δημιουργούς παιχνιδιών να επιλέξουν αν το παιχνίδι τους θα δημοσιευτεί σε κινητές συσκευές, προγράμματα περιήγησης στο διαδίκτυο, ηλεκτρονικούς υπολογιστές ή κονσόλες παιχνιδιών.

Ο όρος cross-platform (ανεξαρτήτου πλατφόρμας) δηλώνει ότι τα παιχνίδια που παράγονται με την Unity μπορούν να κυκλοφορήσουν σε πολλές πλατφόρμες. Κάποιες από αυτές οι εξής:

- BlackBerry 10
- Windows Phone 8
- Windows
- Linux (κυρίως το Ubuntu)
- Android
- iOS
- iOS X
- Ιστότοπους και Facebook με την εγκατάσταση του Unity Web Player
- Adobe Flash
- PlayStation 3
- PlayStation 4
- PlayStation Vita
- Xbox 360
- Xbox One
- Wii U
- Και Wii

Το Unity περιλαμβάνει έναν διακομιστή, και την μηχανή φυσικής της Nvidia PhysX. Για την καλύτερη υποστήριξη των γραφικών στα παιχνίδια της χρειάζεται τα ακόλουθα APIs: Direct3D για το λειτουργικό σύστημα Windows και Xbox, την OpenGL για Mac, Windows και Linux, OpenGL ES για Android και iOS και άλλα ιδιόκτητα APIs για τις κονσόλες βιντεοπαιχνιδιών.

Ένα από τα δυνατά πλεονεκτήματα του Unity είναι ότι παρέχει δυνατότητα συμπίεσης και αλλαγής της ανάλυσης των αρχείων εικόνας και γραφικών γενικότερα, ανάλογα με τις απαιτήσεις της κάθε πλατφόρμας. Για τον προγραμματισμό με Unity μπορούν να χρησιμοποιηθούν οι γλώσσες C#, JavaScript και Boo η οποία μοιάζει με την γλώσσα προγραμματισμού Python. Παρέχει υποστήριξη για ump mapping, reflection mapping, parallax mapping, screen space ambient occlusion (SSAO), dynamic shadows με shadow maps, render-to-texture και full-screen postprocessing effects. Υποστηρίζει στοιχεία και μορφές αρχείων από 3ds Max, Maya, Softimage, Blender, Modo, ZBrush, Cinema 4D, Cheetah3D, Adobe Photoshop και Adobe Fireworks.

Ας σημειώσουμε ότι το Unity είναι το προεπιλεγμένο SDK (Software Development Kit) για τη κονσόλα Wii U της Nintendo. Το Unity παρέχεται είτε δωρεάν, είτε με μικρό αντίτιμο στην Unity Pro έκδοση του.

2.3.2 Unreal Engine 4

Η Unreal Engine είναι μια παιχνιδομηχανή που αναπτύχθηκε από την Epic Games και παρουσιάστηκε για πρώτη φορά το 1998 στο πρώτο FPS (first-person shooter) παιχνίδι Unreal. Η Epic Games είναι αμερικάνικη εταιρία ανάπτυξης βιντεοπαιχνιδιών που ιδρύθηκε το 1991 από τον Tim Sweeney στο Rockville της πολιτείας Maryland, με το όνομα Epic MegaGames. Η τρέχουσα έκδοση της Unreal Engine (Unreal Engine 4) είναι σχεδιασμένη για το Microsoft DirectX (για Windows, Xbox One και Windows RT), για OpenGL (για Linux, OS X, PlayStation 4, iOS, Android και Windows XP) και για Javascript/WebGL (για HTML5 web browsers).

Εικόνα 2.6: Unreal Engine 4 logo

Τον Αύγουστο του 2005 ο αντιπρόεδρος της Epic Games αποκάλυψε ότι η Unreal Engine 4 αναπτυσσόταν από το 2003. Μέχρι τα μέσα του 2008 η ανάπτυξη της πλατφόρμας έγινε αποκλειστικά από τον Tim Sweeney, που εκτός από ιδρυτής της εταιρίας είναι και τεχνικός διευθυντής. Η μηχανή στοχεύει στην ανάπτυξη παιχνιδιών για κονσόλες βιντεοπαιχνιδιών όγδοης γενιάς, υπολογιστές και συσκευές Android όπως ανακοινώθηκε τον Ιανουάριο του 2014. Ο Mark Rein είπε ότι οι άνθρωποι θα συγκλονιστούν όταν δουν την Unreal Engine 4. Η 4η έκδοση της παιχνιδομηχανής παρουσιάστηκε για πρώτη φορά σε περιορισμένο κοινό, στο συνέδριο με τίτλο "Game Developers Conference" του 2012.

Ένα από τα κύρια χαρακτηριστικά της Unreal Engine 4 είναι το real-time global illumination (GI). Είναι η υλοποίηση της παραδοχής ότι το φως αντανακλάται από επιφάνεια σε επιφάνεια. Κάθε επιφάνεια που φωτίζεται λειτουργεί σαν πηγη φωτός. Οι απαιτήσεις για την υλοποιήση αυτή είναι τεράστιες και μέχρι σήμερα οι παιχνιδομηχανές χρησιμοποιούσαν static global illumination. Οι προγραμματιστές έπρεπε να τρέξουν νέους υπολογισμούς κάθε φορά που άλλαζε ένα μέρος της πίστας. Το real-time Global Illumination είναι απαλλαγμένο από τα pre-computed lightmaps που

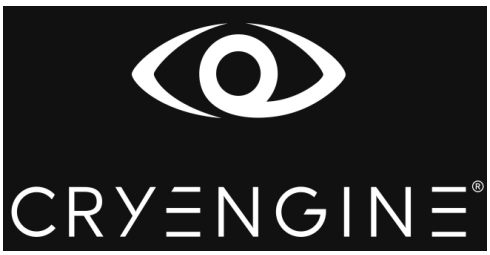
υπήρχαν στις παραδοσιακές παιχνιδομηχανές και ωφελεί και τους παίκτες και τους προγραμματιστές. Τους πρώτους επειδή προσφέρει ποιοτικό φωτισμό χωρίς σκοτεινά σημεία και τους δεύτερους επειδή τους απαλλάζει από τον χαμένο χρόνο στο pre-computation, επομένως προσφέρει και χαμηλότερο κόστος.

Άλλα χαρακτηριστικά της UE4 είναι η υψηλού επιπέδου μηχανή φυσικής PhysX, το Extreme tessellation, το εργαλείο scripting kismet 2 που καθορίζει τις σχέσεις ανάμεσα στα αντικείμενα στον κόσμο του παιχνιδιού και το Subsurface Scattering. Η UE4 περιλαμβάνει επίσης νέα προγραματιστικά χαρακτηριστικά για να μειώσει τον χρόνο επανάληψης και επιτρέπει την ενημέρωση του κώδικα C++ ενώ η μηχανή είναι σε λειτουργία. Το Blueprint Visual Scripting επιτρέπει στον καθένα να δημιουργήσει γρήγορα πρωτότυπα περιεχόμενα χωρίς να γράψει γραμμή κώδικα.

Την Unreal Engine 4 μπορεί να την χρησιμοποιήσει ο καθένας καταβάλλοντας μηνιαία συνδρομή 19 δολαρίων συν 5% από τα έσοδα που προκύπτουν από τυχόν εμπορικά προϊόντα με την χρήση της αν αυτά έχουν ξεπεράσει τα 3000 δολάρια. Τον Σεπτέμβριο του 2014, η Epic κυκλοφόρησε την UE4 δωρεάν σε σχολεία και πανεπιστήμια. Οι υπεύθυνοι της Epic Games θεωρούν ότι με αυτή την κίνηση δίνεται η δυνατότητα στους μαθητές να κάνουν χρήση των εξελιγμένων εργαλείων που τους δίνει η Unreal Engine 4 και να γνωρίσουν έτσι την τέχνη του game development στην πιο αγνή του μορφή.

2.3.3 CryENGINE

Η CryEngine είναι μία παιχνιδομηχανή που κατασκευάστηκε από την γερμανοτουρκική εταιρία Crytek. Η Crytek ιδρύθηκε το 1999, από τους αδερφούς Cevat, Avni και Faruk Yerli με έδρα την Φρανκφούρτη της Γερμανίας. Η εταιρία είναι περισσότερο γνωστή για το παιχνίδι Far Cry το οποίο προβάλλει και την παιχνιδομηχανή CryEngine.



Εικόνα 2.7: CryEngine logo

Η CryEngine είναι μία ακόμα cross-platform μηχανή που υποστηρίζει την ανάπτυξη παιχνιδιών για Windows, Linux, Xbox 360, Xbox One, PlayStation 3, PlayStation 4, Wii U, iOS και Android, ανάλογα με τις εκδόσεις της. Ένα από τα σύγχρονα χαρακτηριστικά της μηχανής είναι το Live Create, που επιτρέπει στους developers να δουλεύουν σε μία πλατφόρμα αλλά να βλέπουν τα αποτελέσματα για όλες τις πλατφόρμες. Επίσης παρέχει πολύ δυνατά εργαλεία για δημιουργία αντικειμένων, όπως δρόμοι, ποτάμια, δέντρα και οχήματα κάνοντας την ανάπτυξη παιχνιδιών πιο γρήγορη. Υποστηρίζει όπως και η Unreal Engine 4 υψηλής ποιότητας γραφικά χάρις στην τεχνολογία real-time global illumination και περιλαμβάνει μία αναβαθμισμένη μηχανή φυσικής multi-threaded που μπορεί να εφαρμοστεί σχεδόν σε οποιοδήποτε αντικείμενο και να δώσει ρεαλιστική μορφή σε φυσικά φαινόμενα, εκρήξεις, συγκρούσεις και άλλα. Παρέχει επίσης στους προγραμματιστές εργαλεία ανάλυσης απόδοσης.

Η πρώτη έκδοση της μηχανής παιχνιδιών CryEngine, αναπτύχθηκε για να αναδείξει τις δυνατότητες της Nvidia μέσω του Far Cry. Όταν κυκλοφόρησαν οι κάρτες γραφικών για 3.0 pixel και vertex shaders, η εταιρία κυκλοφόρησε την έκδοση 1.2 που χρησιμοποιεί μερικές δυνατότητες για καλύτερα γραφικά. Στην πορεία στην έκδοση 1.3 πρόσθεσε την υποστήριξη HDR φωτισμού. Έπειτα δημιουργήθηκε η CryEngine 2 για το παιχνίδι Crysis, με την οποία κατασκευάστηκαν και άλλα παιχνίδια όπως το Blue Mars, το The Day και το Entropia Universe. Οι εκδόσεις 1 και 2 υποστήριζαν την ανάπτυξη παιχνιδιών για τα Windows. Η CryEngine 3 παρουσιάστηκε για πρώτη

φορά το 2009 και για πρώτη φορά υποστήριξε την ανάπτυξη παιχνιδιών και για κονσόλες, πιο συγκεκριμένα για το Xbox 360, το PlayStation 3, το Wii U. Στην πιο πρόσφατη έκδοση της η CryEngine 4th generation (2013) περιλαμβάνει στις πλατφόρμες που υποστηρίζει το PlayStation 4, το Xbox One, το Wii U και τα Linux, ενώ το 2014 κατασκευάστηκε με αυτήν το The Collectable για Android και iOS.

Οι προγραμματιστές μπορούν να χρησιμοποιήσουν το CryEngine SDK που αρχικά ονομαζόταν Sandbox Editor, το οποίο παρέχεται δωρεάν για μη εμπορική χρήση και τους δίνει την δυνατότητα να πειραματιστούν σε αυτό. Ένα μεγάλο πλεονέκτημα της παιχνιδομηχανής είναι ότι η Crytek προσφέρει ένα επίσημο forum όπου οι προγραμματιστές μπορούν να έχουν εξειδικευμένη τεχνική υποστήριξη, να προβάλλουν τις δημιουργίες τους ή να δουν και να χρησιμοποιήσουν δημιουργίες άλλων.

2.4 Software Architecture

Αρχιτεκτονική Λογισμικού αποκαλείται ο υψηλού επιπέδου σχεδιασμός λογισμικού. Είναι το πρώτο βήμα για το σχεδιασμό μετά την ανάλυση όλων των απαιτήσεων του λογισμικού. Περιγράφει τη γενική στατική δομή του συστήματος, τα στοιχεία από τα οποία αποτελείται, τις αλληλεπιδράσεις μεταξύ τους καθώς και τα πρότυπα και τους περιορισμούς που καθοδηγούν την σύνθεση αυτών των στοιχείων. Δεν επηρεάζει μόνο τις λειτουργικές απαιτήσεις αλλά και τις μη λειτουργικές όπως είναι η επεκτασιμότητα, η φορητότητα και η δυνατότητα συντήρησης.

Στόχος της αρχιτεκτονικής λογισμικού είναι να βοηθήσει στην καλύτερη κατανόηση του συστήματος, να γίνει κατανοητή η αρχιτεκτονική του και να οργανωθεί το έργο. Έτσι αναλαμβάνει τον διαχωρισμό και την κατάτμηση της εφαρμογής στα συστατικά δομικά στοιχεία από τα οποία αποτελείται το σύστημα και καθορίζει τον σκελετό της εφαρμογής. Με αυτόν τον τρόπο η ανάπτυξη των δομικών μονάδων μπορεί να γίνει ανεξάρτητα ως έναν βαθμό από μια ομάδα μηχανικών λογισμικού. Η αρχιτεκτονική προδιαγράφει επίσης σημαντικές ιδιότητες του συστήματος, όπως είναι η απόδοση, η ασφάλεια, η ευρωστία και η ευκολία αναβάθμισης.

Πιο συγκεκριμένα η αρχιτεκτονική λογισμικού ασχολείται με την οργάνωση του συστήματος ως σύνθεση εξαρτημάτων, τις καθολικές δομές ελέγχου, τα πρωτόκολλα επικοινωνίας, συγχρονισμού και πρόσβασης σε αποθηκευμένα δεδομένα, την ευθυγράμμιση των λειτουργιών με τα στοιχεία του σχεδιασμού, τη σύνθεση των στοιχείων του σχεδίου, τη φυσική υλοποίηση του συστήματος, την απόδοση και την ανταπόκριση σε αυξανόμενες απαιτήσεις, τις δυνατότητες εξέλιξης και την επιλογή μεταξύ εναλλακτικών σχεδίων. Γενικότερα η αρχιτεκτονική περιγράφει το σύστημα ως σύνολο υπολογιστικών εξαρτημάτων και συνδέσεων μεταξύ αυτών.

Για τον καλύτερο σχεδιασμό συστημάτων χρησιμοποιούνται συγκεκριμένα μοτίβα και πρότυπα, που είναι δοκιμασμένα. Αυτά αποτελούν και ένα κοινό λεξικό για τους μηχανικούς λογισμικού που συμμετέχουν στο έργο και χαρακτηρίζονται και ως μέσο επικοινωνίας.

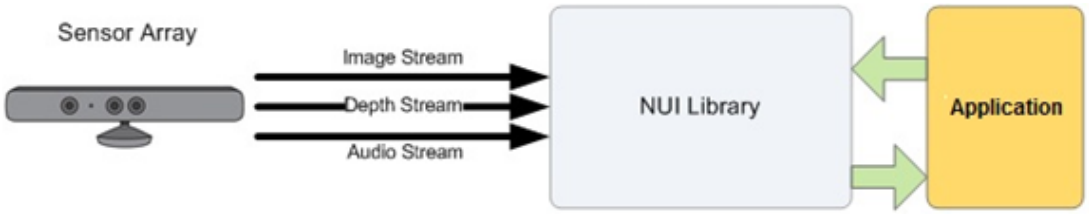
Η ποιότητα της αρχιτεκτονικής κρίνεται και αξιολογείται από μερικές παραμέτρους. Μία από αυτές είναι η τεκμηρίωση, η αρχιτεκτονική πρέπει να επιθεωρηθεί από όλους τους εταίρους, και πρέπει να περιγράφεται σωστά και με κατανοητό τρόπο.

Επίσης πρέπει να είναι καλά ορισμένα τα τμήματα από τα οποία αποτελείται και να είναι όσο το δυνατόν ανεξάρτητα μεταξύ τους και να μπορούν να υλοποιηθούν παράλληλα. Επιπλέον η αρχιτεκτονική πρέπει να έχει ελάχιστη δυνατή εξάρτηση από το υλικό.

2.4.1 Kinect for Windows Architecture

Το Kinect for Windows SDK λειτουργεί ως μια διεπαφή μεταξύ του Kinect και της εφαρμογής μας και παρέχει μια εκλεπτυσμένη βιβλιοθήκη λογισμικού και εργαλεία τα οποία βοηθούν τους προγραμματιστές να χρησιμοποιήσουν την φυσική είσοδο του Kinect.

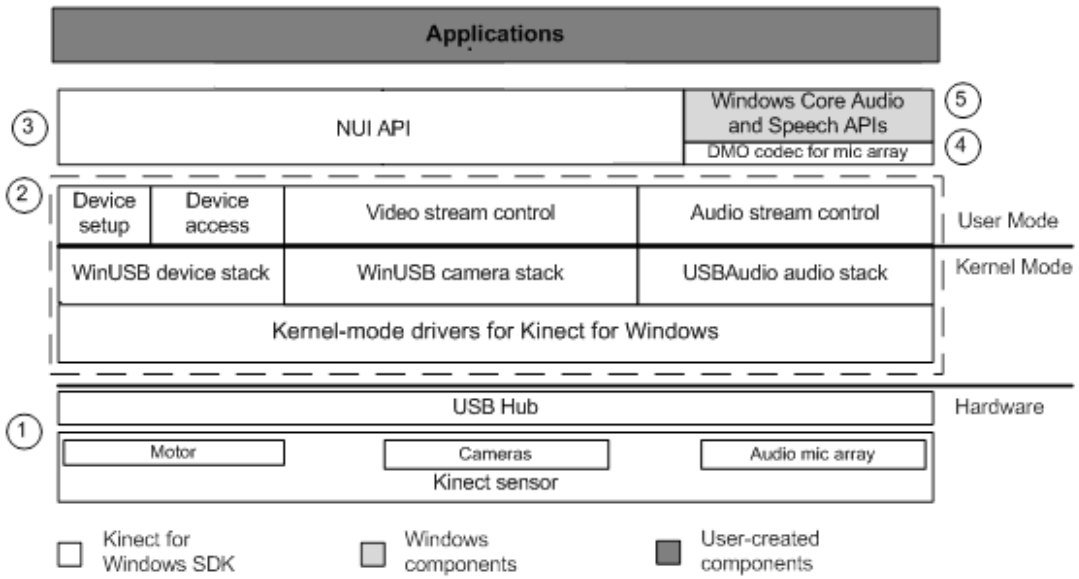
Το Kinect και η βιβλιοθήκη λογισμικού αλληλεπιδρούν με κάποια εξωτερική εφαρμογή, όπως φαίνεται παρακάτω:



Εικόνα 2.8: Kinect for Windows Hardware and Software Interaction with an Application

Kinect for Windows SDK Architecture

Πιο αναλυτικά η αρχιτεκτονική του SDK έχει ως εξής



Εικόνα 2.9: Kinect for Windows SDK Architecture

1. Kinect hardware - Τα στοιχεία του υλικού, συμπεριλαμβανομένου του αισθητήρα Kinect και του διανομέα USB μέσω του οποίου ο αισθητήρας Kinect είναι συνδεδεμένος με τον υπολογιστή.

2. Kinect drivers - Τα προγράμματα οδήγησης των Windows για το Kinect, που εγκαθίστανται ως μέρος της διαδικασίας εγκατάστασης του SDK. Τα προγράμματα οδήγησης των Windows για το Kinect υποστηρίζουν:

   - Τη συστοιχία μικροφώνων του Kinect ως συσκευή ήχου λειτουργίας πυρήνα στην οποία μπορούμε να έχουμε πρόσβαση μέσω των τυπικών API ήχου των Windows.
   - Λειτουργίες ελέγχου ήχου και βίντεο συνεχούς ροής για ήχο και βίντεο (χρώμα, βάθος και σκελετός).
   - Λειτουργίες απαρίθμησης συσκευών που επιτρέπουν στην εφαρμογή μας να χρησιμοποιεί περισσότερα του ενός Kinect.

3. Audio and Video Components

   - Kinect natural user interface για την παρακολούθηση του σκελετού, του ήχου, και την απεικόνιση της εικόνας και του βάθους.

4. DirectX Media Object (DMO) για συστοιχία μικροφώνων beamforming και εντοπισμό της πηγής ήχου.

5. Windows 7 τυπικά API - Τα API ήχου, ομιλίας και πολυμέσων των Windows 7, όπως περιγράφονται στο Windows 7 SDK και Microsoft Speech SDK. Αυτά τα API είναι επίσης διαθέσιμα για desktop εφαρμογές στα Windows 8.

Όπως βλέπουμε στο σχεδιάγραμμα τα data streams μεταδίδονται στον υπολογιστή μας μέσω USB hub.

Όταν χρειαστεί να αποκτήσουμε πρόσβαση στον αισθητήρα, η εφαρμογή μας στέλνει μια κλήση API στον οδηγό και ο οδηγός ελέγχει την πρόσβαση στα δεδομένα του αισθητήρα.

Το NUI API συλλέγει τα ανεπεξέργαστα δεδομένα και τα παρουσιάζει στις εφαρμογές ως ροες δεδομένων (data streams).

Η ανάπτυξη Kinect-enabled εφαρμογών είναι ουσιαστικά η ίδια με την ανάπτυξη οποιονδήποτε άλλων εφαρμογών των Windows, με τη διαφορά ότι το Kinect SDK παρέχει υποστήριξη για τα χαρακτηριστικά του Kinect, συμπεριλαμβανομένων των εικόνων χρώματος και βάθους, την είσοδο ήχου, και των σκελετικών δεδομένων.

Παρακάτω βλέπουμε μερικά παραδείγματα χρηστικότητας που μπορούν να αποκτήσουν οι εφαρμογές μας χρησιμοποιώντας τη λειτουργικότητα που υποστηρίζεται από το SDK:

- Αναγνώριση και παρακολούθηση της μετακίνησης ανθρώπων μέσω της ανίχνευση σκελετού (skeletal tracking).
- Καθορισμός της απόστασης μεταξύ ενός αντικειμένου και της κάμερας του αισθητήρα χρησιμοποιώντας τα δεδομένα βάθους (depth data).
- Σύλληψη ήχου χρησιμοποιώντας noise and echo cancellation και εντοπισμός της θέσης της πηγής του ήχου.
- Ανάπτυξη εφαρμογών που ενεργοποιούνται με τη φωνή (voice-activated) προγραμματίζοντας μία γραμματική για χρήση με μια μηχανή αναγνώρισης ομιλίας (speech recognition engine).

Περισσότερες πληροφορίες για το Kinect for Windows SDK μπορούμε να δούμε στην επίσημη ιστοσελίδα του Microsoft Developer Network. https://msdn.microsoft.com/en-us/library/hh855347.aspx

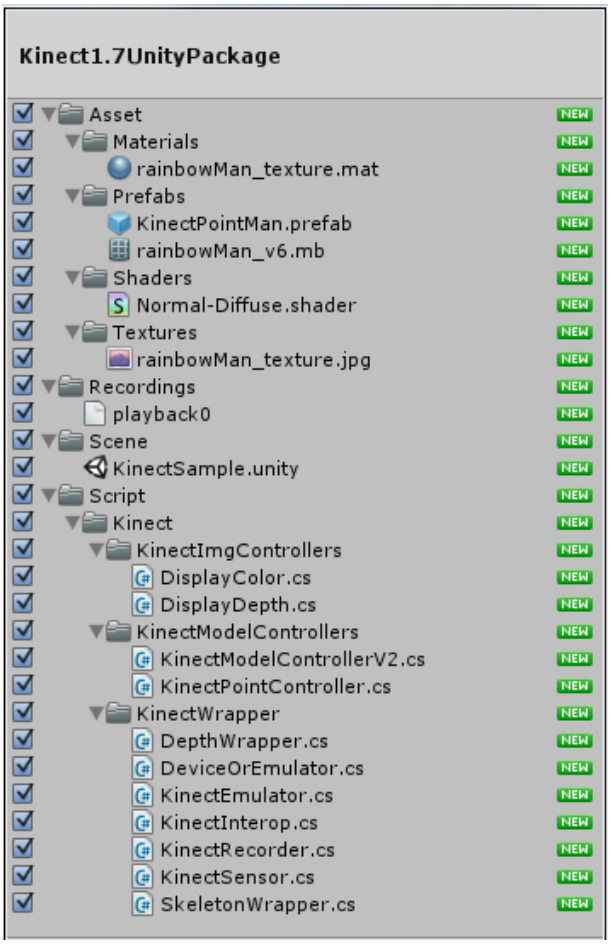
Kinect Vs Unity3D

Όσον αφορά τη σύνδεση του Kinect v1 με το Unity τρεις είναι οι επιλογές που έχουν στη διάθεση τους οι προγραμματιστές και τις οποίες θα δούμε στην συνέχεια παρακάτω.

3.1 Kinect Wrapper Package for Unity3D - Kinect1.7UnityPackage

Το Kinect Wrapper Package for Unity3D μπορεί κάποιος να το προμηθευτεί δωρεάν από την σελίδα ETC Internal Unity3D Wiki και πιο συγκεκριμένα από το link http://wiki.etc.cmu.edu/unity3d/index.php/Microsoft_Kinect_-_Microsoft_SDK.

Στην εικόνα 3.1 μπορούμε να δούμε τα περιεχόμενα του package για το Unity.

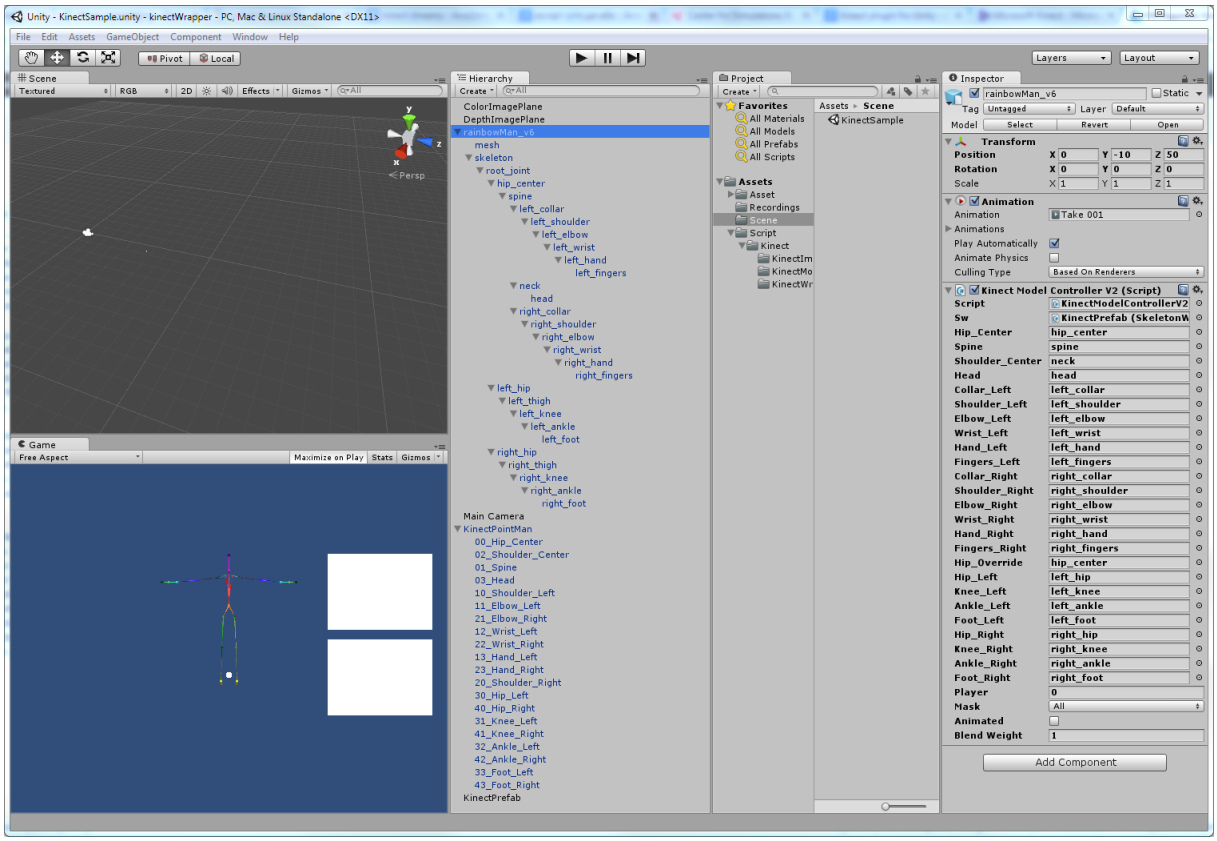


Εικόνα 3.1: Τα περιεχόμενα του Kinect1.7UnityPackage

Το Kinect Wrapper Package χρησιμοποιεί το Kinect for Windows SDK και το Kinect for Windows Developer Toolkit. Περιέχει τα απαραίτητα scripts καθώς και μια σκηνή του Unity

την KinectSample στην οποία γίνεται αναπαράσταση των skeleton, color και depth streams και την οποία μπορούν να χρησιμοποιήσουν οι προγραμματιστές ως παράδειγμα για τις δικές τους εφαρμογές.

Στην εικόνα 3.2 μπορούμε να δούμε την σκηνή KinectSample στον editor του Unity.



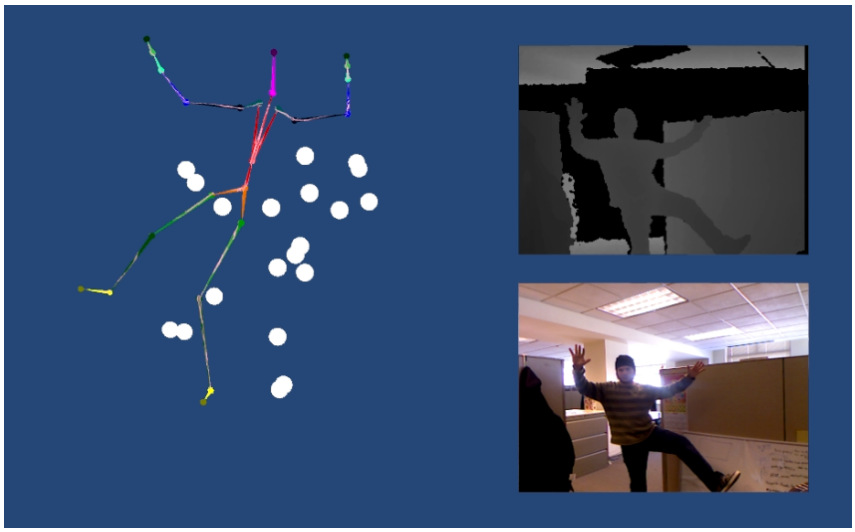
Εικόνα 3.2: Η σκηνή KinectSample όπως φαίνεται στον editor του Unity

Ρίχνοντας μια πιο προσεκτική ματιά στην ιεραρχία μέσα στη σκηνή KinectSample παρατηρούμε δύο συγκεκριμένα game objects τα rainbowMan_v6 και KinectPointMan.

Σε αυτά τα game objects είναι συνδεδεμένα τα δύο scripts τα οποία είναι υπεύθυνα ώστε να καταγράφουν και να μιμούνται τις κινήσεις του χρήστη, τα KinectPointController και KinectModelControllerV2. Και τα δύο αυτά scripts χρειάζονται μια αναφορά (reference) στον skeleton wrapper για να λειτουργήσουν.

Το KinectPrefab καταγράφει τα δεδομένα κίνησης σε πραγματικό χρόνο και στέλνει τις πληροφορίες στα controller scripts. Τα controller scripts αντιστοιχίζουν αυτά τα δεδομένα σε game objects και μιμούνται τη σχετική τοποθέτηση τους σε ένα 3D περιβάλλον.

Στην εικόνα 3.3 μπορούμε να δούμε το αποτέλεσμα της σκηνής KinectSample.



Εικόνα 3.3: Η σκηνή KinectSample

Η κύρια διαφορά μεταξύ των KinectPointController και KinectModelControllerV2 είναι το πώς αναφέρονται σε κάθε συγκεκριμένο μέρος του σώματος. Το KinectPointController χρησιμοποιεί μεμονωμένα game objects. Στη σκηνή KinectSample χρησιμοποιούνται σφαίρες για την αναπαράσταση των σημείων του σκελετού. Στην πράξη αυτά τα αντικείμενα θα μπορούσε να αντικατασταθούν με ένα απλό μοντέλο, το χέρι ή ένα αντικείμενο δείκτη που ο χρήστης μπορεί να ελέγχει και να αλληλεπιδρά με αυτό.

3.2 Kinect with MS-SDK By RF Solutions

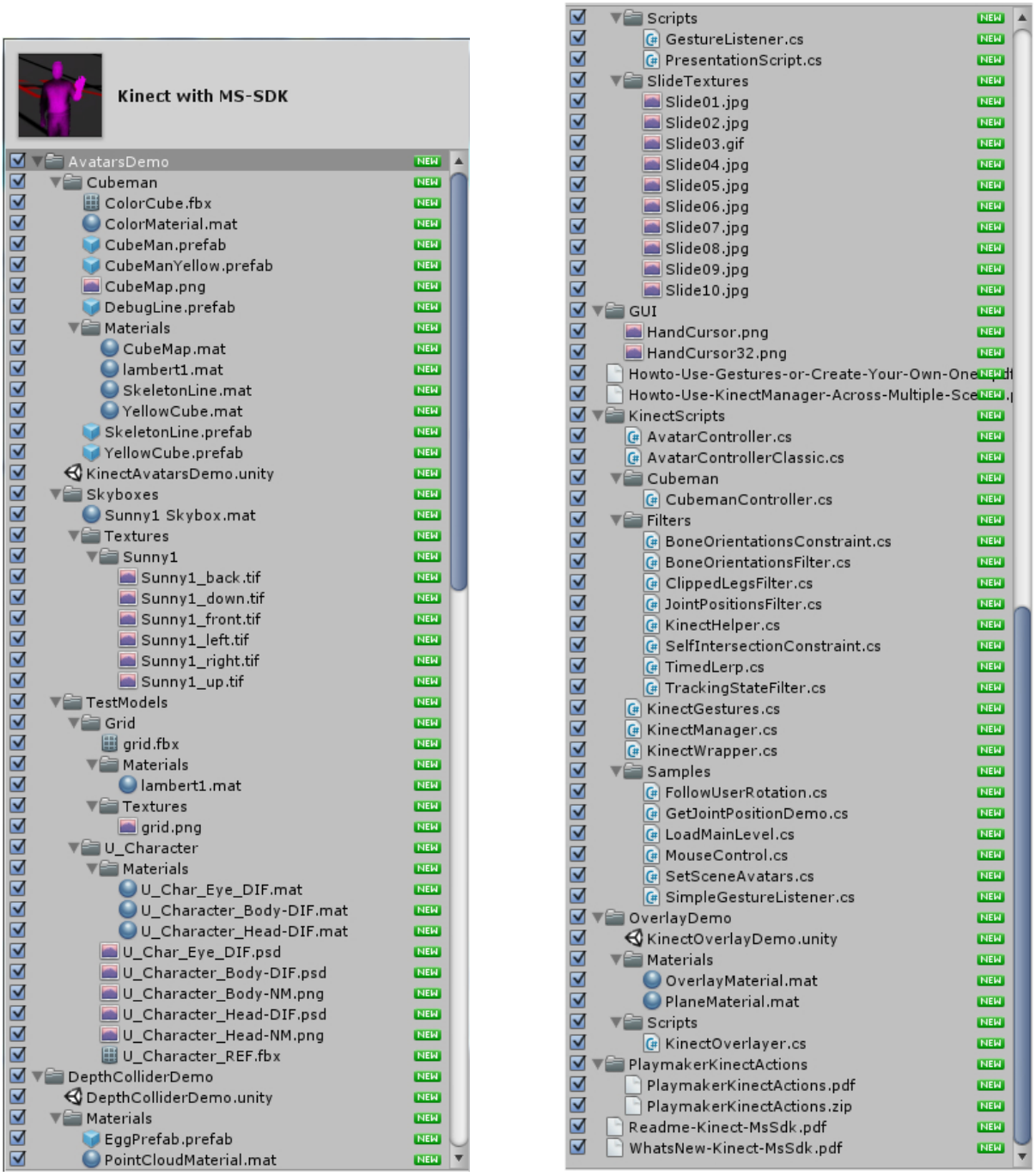
Μία δεύτερη λύση η οποία είναι διαθέσιμη στο διαδίκτυο είναι το Kinect with MS-SDK του Rumen Filkov (RF Solutions) το οποίο μπορεί να προμηθευτεί όποιος ενδιαφέρεται επίσης δωρεάν από το Asset Store του Unity και το εξής link https://www.assetstore.unity3d.com/en/#!/content/7747 και του οποίου τα περιεχόμενα μπορούμε να δούμε στην εικόνα 3.4.

Ο Rumen έχει εκδώσει και ένα συμπληρωματικό package το KinectExtras with MsSDK το οποίο είναι διαθέσιμο επίσης μέσω του Asset Store από το παρακάτω λινκ https://www.assetstore.unity3d.com/en/#!/content/10492 και το οποίο είναι μια επέκταση του Kinect with MS-SDK package. Διατίθεται με το αντίτιμο των $10 και παρέχει επιπλέον παραδείγματα και demos με βάση το Kinect SDK που δεν σχετίζονται άμεσα με τη μεταφορά της ανθρώπινης κίνησης σε ψηφιακούς χαρακτήρες.

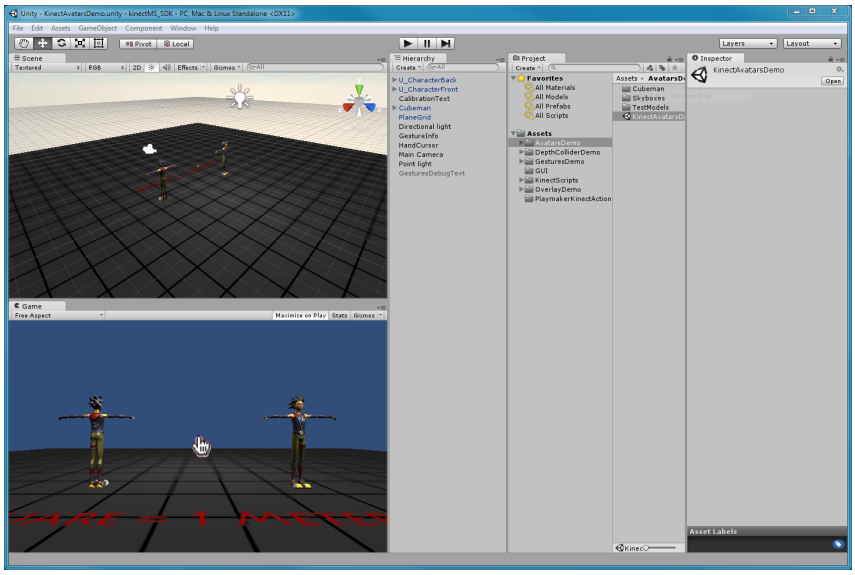
Το package αποτελείται από το Kinect Interaction demo το οποίο παρουσιάζει το hand cursor control καθώς και τις grip και release χειρονομίες, παράδειγμα αναγνώρισης ομιλίας, παράδειγμα face-tracking και demo αφαίρεσης φόντου. Είναι δυνατόν να ενσωματωθούν όλα τα χαρακτηριστικά σε ένα project, δηλαδή το KinectExtras και KinectManager (από το Kinect with MS-SDK package) λειτουργούν παράλληλα.

Το package είναι συμβατό με το Kinect v1 μόνο, υποστηρίζει 32-bit και 64-bit εκδόσεις και μπορεί να λειτουργήσει στη Pro και Free έκδοση του Unity.

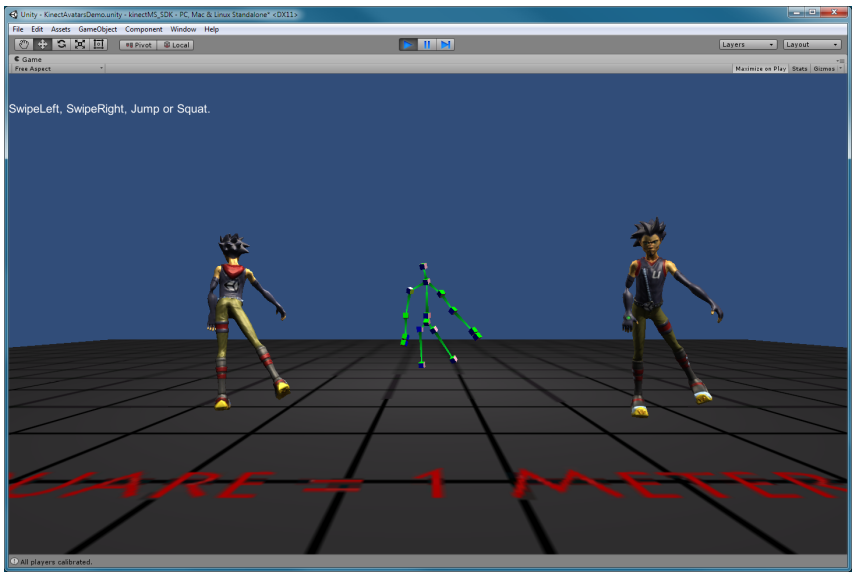
Παρέχεται δωρεάν σε σχολεία, πανεπιστήμια, φοιτητές και εκπαιδευτικούς έπειτα από επικοινωνία με τον Rumen στο e-mail του.



Εικόνα 3.4: Τα περιεχόμενα του package Kinect with MS-SDK



Εικόνα 3.5: Η σκηνή KinectAvatarsDemo που περιλαμβάνεται στο Package



Εικόνα 3.6: Η σκηνή KinectAvatarsDemo που περιλαμβάνεται στο Package

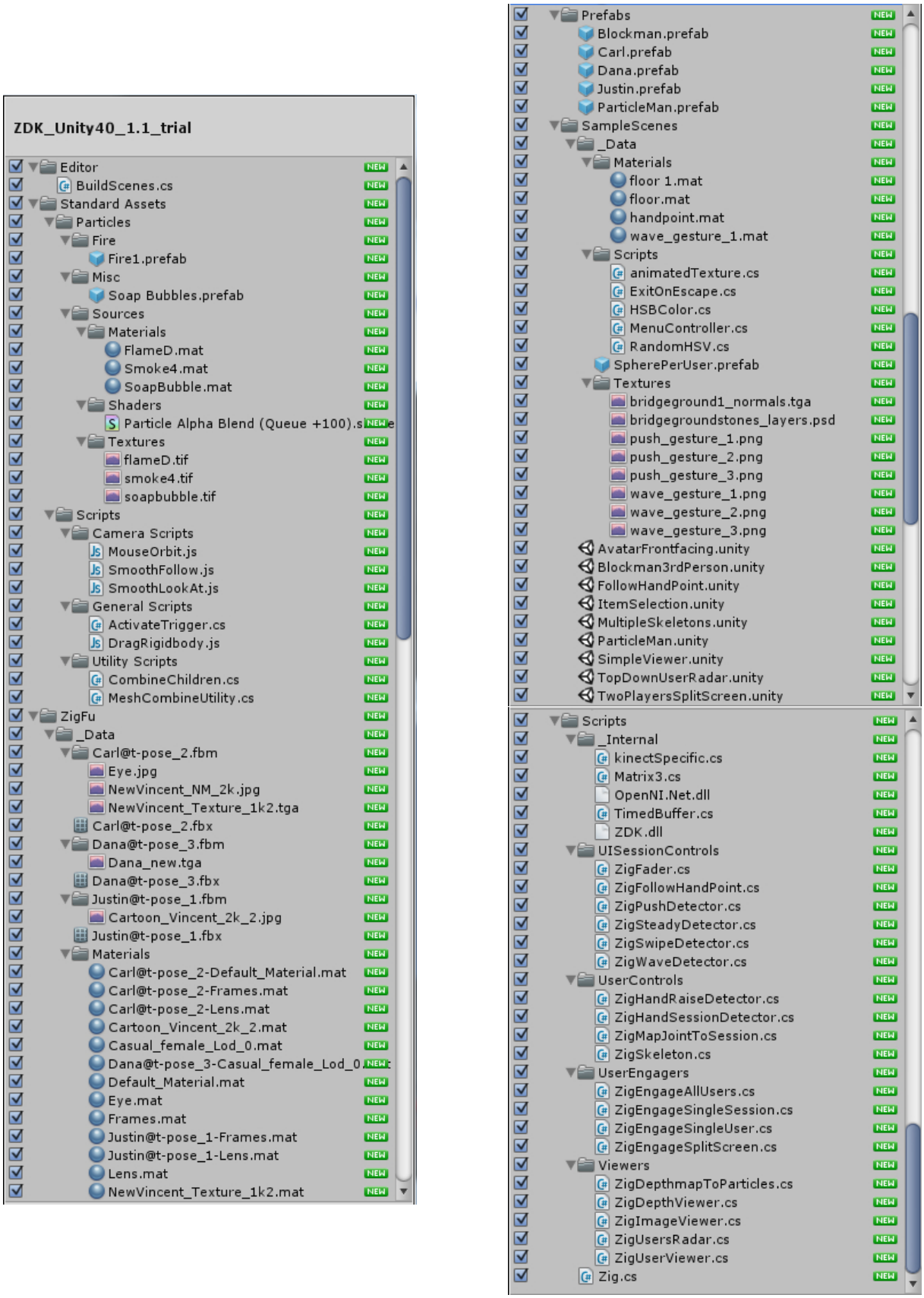
3.3 Zigfu ZDK for Unity3D

Η τρίτη και τελευταία λύση που θα παρουσιάσουμε είναι το ZigFu ZDK το οποίο μπορεί να προμηθευτεί όποιος ενδιαφέρεται από την ιστοσελίδα της εταιρείας http://zigfu.com

Το ZigFu δίνει στους προγραμματιστές τη δυνατότητα να αναπτύξουν εφαρμογές φυσικής διεπαφής σε μερικά μόνο λεπτά σύμφωνα με τον Amir Hirsch συν-ιδρυτή της εταιρείας. Παρέχει τα απαραίτητα εργαλεία για τη σύνδεση του Kinect με το Unity3D δίνοντας παράλληλα στους προγραμματιστές αρκετές εφαρμογές παραδείγματα ώστε να βασιστούν και να ξεκινήσουν τη δική τους δουλειά.

Χρειάζεται περίπου ένα λεπτό για να μετατρέψει κάποιος έναν πλοηγό για μία λίστα στοιχείων σε εφαρμογή ελεγχόμενη με κίνηση σύμφωνα με τον Hirsch. Για παράδειγμα, μπορούμε να μετατρέψουμε ένα κατάλογο περιήγησης ή μία παρουσίαση διαφανειών σε λίγα μόνο λεπτά.

Το ZigFu χρησιμοποιεί το OpenNI framework για την φυσική αλληλεπίδραση, το οποίο παρέχει ένα αφηρημένο framework για τη χρήση και παρακολούθηση του σκελετού του χεριού.



Εικόνα 3.7: Τα περιεχόμενα της trial έκδοσης του ZDK για το Unity3D

Το OpenNI προσπαθεί ουσιαστικά να βεβαιωθεί ότι το hardware που είναι εστιασμένο στην παρακολούθηση χειρονομιών αλλά και το middleware και το λογισμικό, όλα αυτά θα λειτουργούν με τα ίδια πρότυπα και θα λειτουργούν καλά μαζί. Το framework ανοιχτού κώδικα προσφέρει στο OpenNI ένα API για τη φυσική αλληλεπίδραση με το υλικό και το λογισμικό.

Εκτός από το OpenNI το ZigFu χρησιμοποιεί επίσης και το PrimeSense NITE, ένα middleware που επιτρέπει στους υπολογιστές να αντιληφθούν ένα τρισδιάστατο κόσμο και μια τεχνολογία που έχει χρησιμοποιηθεί εκτενώς σε Kinect hacking εφαρμογές.

Φυσικά το ZigFu υποστηρίζει και το Kinect for Windows SDK.

Χρησιμοποιώντας το ZigFu σύμφωνα και πάλι με τον Hirsch χρειάζονται περίπου δύο λεπτά click-and-drag δουλειάς ώστε ένα ελεγχόμενο με κίνηση avatar να οργανωθεί και να λειτουργήσει. Η δουλειά που χρειάζεται να γίνει είναι απλώς η σύνδεση των ώμων, των αγκώνων, των γονάτων και των ισχίων του 3D χαρακτήρα με την έξοδο των δεδομένων από το module παρακολούθησης του σκελετού. Το γεγονός του να έχουμε μια μηχανή παιχνιδιών όπως το Unity σε συνδυασμό με τον έλεγχο του avatar μας δίνει δυνατότητες που επεκτείνονται κατά πολύ από εκεί και πέρα.

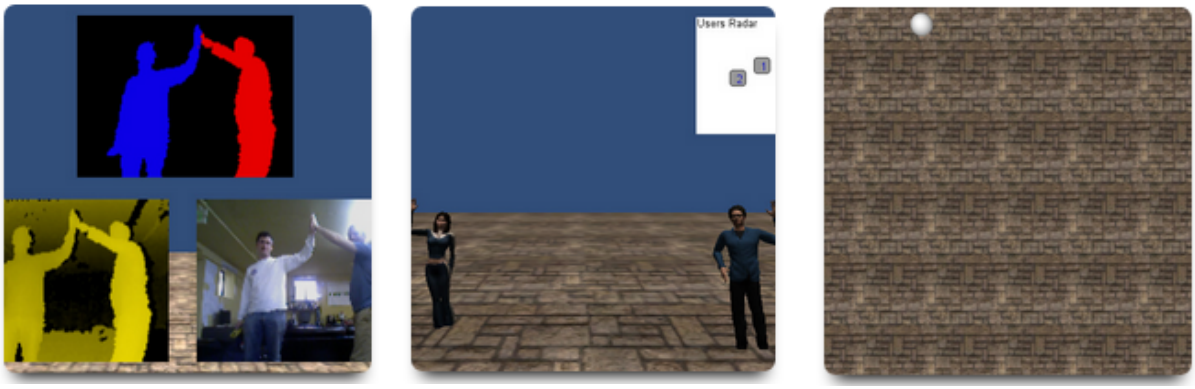
Για τους μεμονωμένους προγραμματιστές ή μικρές ομάδες ανάπτυξης λογισμικού το κόστος της άδειας είναι $200 ανά προγραμματιστή. Για μεγαλύτερες ομάδες όπως εταιρείες με κάτω από 10 εργαζόμενους το κόστος είναι $1,500 ενώ για εταιρείες με κάτω από 25 εργαζόμενους $3.000 δολάρια.

Υπάρχει διαθέσιμη και η trial έκδοση η οποία διατίθεται δωρεάν και τυπώνει ένα ολόγραμμα στην κάτω αριστερή γωνία της σκηνής.

Στις εικόνες που ακολουθούν μπορούμε να δούμε τις σκηνές παραδείγματα οι οποίες είναι διαθέσιμες με το ZigFu.

Εικόνα 3.8: Οι σκηνές παραδείγματα του ZDK για το Unity3D (α)

Εικόνα 3.9: Οι σκηνές παραδείγματα του ZDK για το Unity3D (β)



Εικόνα 3.10: Οι σκηνές παραδείγματα του ZDK για το Unity3D (γ)

Kinect & Unity, η δική μας προσεγγιση

4.1 Ανάλυση Προβλήματος

Παρόλο που το Unity έχει εκτεταμένη υποστήριξη για τα plugins, τα οποία είναι βιβλιοθήκες εγγενή κώδικα γραμμένου σε C, C++, Objective-C κλπ, για λόγους ασφαλείας ή χρήση plug-ins δεν επιτρέπεται στον webplayer. Οι περιορισμοί ασφαλείας ισχύουν μόνο για τον webplayer, και προς τον editor όταν active build target είναι ο WebPlayer.

Τα plugins επιτρέπουν στον κώδικα ενός παιχνιδιού (γραμμένο σε Javascript, C# ή Boo) να καλεί functions από αυτές τις βιβλιοθήκες. Αυτό το χαρακτηριστικό επιτρέπει στις τις βιβλιοθήκες ενδιάμεσου ή υπάρχοντος C/C++ κώδικα του παιχνιδιού να ενσωματώνονται στο Unity.

Το Kinect Wrapper Package for Unity3D και το Kinect with MS-SDK της RF Solutions, δύο από τις λύσεις που είδαμε προηγουμένως, βασίζονται για να λειτουργήσουν σε κλήσεις του τύπου [DllImport ("PluginName")] όπως η παρακάτω

```
1 [DllImportAttribute(@"Kinect10.dll", EntryPoint = "NuiSkeletonTrackingEnable")]
```

και αυτός είναι και ο λόγος για τον οποίο δεν λειτουργούν στον Web Player. Περισσότερα για το συγκεκριμένο ζήτημα μπορούμε να διαβάσουμε στην παρακάτω σελίδα από το εγχειρίδιο του Unity. http://docs.unity3d.com/Manual/Plugins.html Το ZigFu απεναντίας χρησιμοποιεί μια άλλη προσέγγιση η οποία του επιτρέπει να μπορεί να λειτουργεί στον WebPlayer χωρίς πρόβλημα.

Βασίζεται στην εγκατάσταση ενός πρόσθετου plugin για τον browser το οποίο και αναλαμβάνει την επικοινωνία με την HTML σελίδα που περιέχει το περιεχόμενο του Unity Web Player αποφεύγοντας έτσι την κλήση plugin μέσα από το Unity.

4.1.1 Απαιτήσεις Συστήματος

Οι απαιτήσεις που ορίστηκαν για το σύστημα μας και την εργασία αυτή ήταν οι εξής.

Kinect & Unity Interaction

Αρχικά μέσω του συστηματός μας θα έπρεπε μα μπορούμε να έχουμε πρόσβαση από τον Web player του Unity στα δεδομένα που παράγει το Kinect.

Gesture recognition

Πιο συγκεκριμένα μας ενδιέφερε η μετάδοση των δεδομένων του σκελετού του χρήστη, κάτι το οποίο μας δίνει τη δυνατότητα πέρα από το να μπορέσουμε να κάνουμε αναπαραγωγή του σκελετού μέσα στο 3D περιβάλλον του Unity Player, να μπορούμε επίσης να κάνουμε και αναγνώριση βασικών χειρονομιών.

Μια χειρονομία είναι μια ανθρώπινη κίνηση του σώματος ή ενέργεια που αντιπροσωπεύει ένα μήνυμα για την εφαρμογή μας.

Επίσης θέλαμε να γνωρίζουμε αν έχει γίνει κάποια grip ή release χειρονομία από τον χρήστη αν το χέρι δηλαδή του χρήστη βρίσκεται με κλειστή ή ανοιχτή την παλάμη.

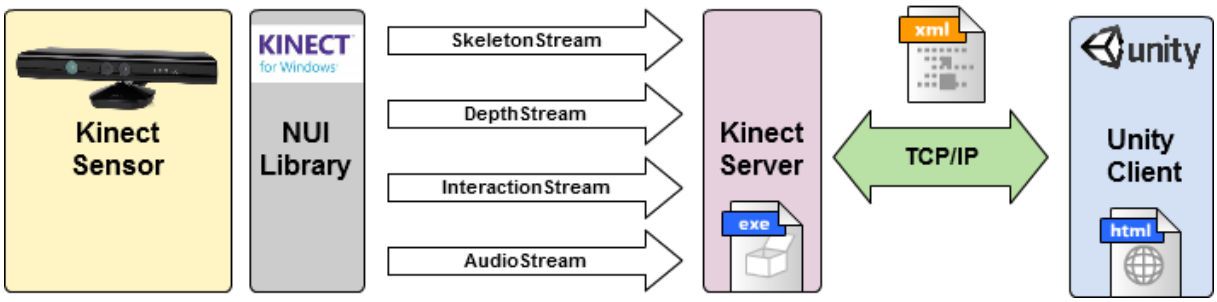
Speech recognition

Μία από τις βασικές πτυχές της φυσικής αλληλεπίδρασης χρήστη είναι η αναγνώριση ομιλίας. Η αναγνώριση ομιλίας επιτρέπει στους χρήστες να πουν κάποια εντολή μπροστά από το μικρόφωνο, και από την άλλη πλευρά ο υπολογιστής να εκτελέσει μια συγκεκριμένη ενέργεια, ανάλογα με την αναγνωρισμένη εντολή.

Το σύστημα μας θα έπρεπε παράλληλα να μπορεί να αναγνωρίσει και κάποιες φωνητικές εντολές.

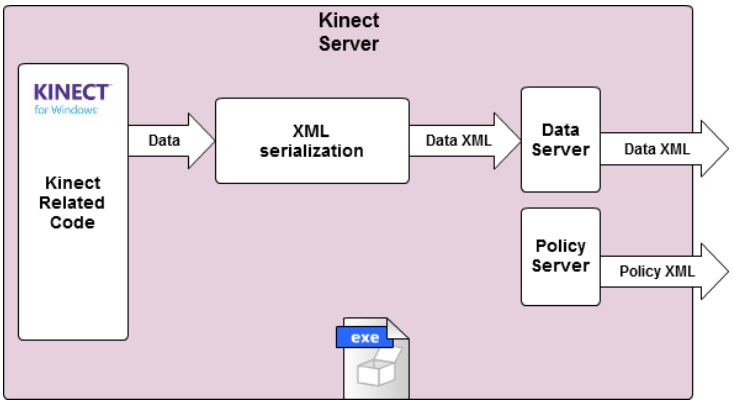
4.2 Σχεδιασμός Συστήματος

Για τη σχεδίαση και υλοποίηση του συστήματος μας χρησιμοποιήθηκε ένα client server μοντέλο με χρήση TCP/IP sockets.



Εικόνα 4.1: Η αρχιτεκτονική του συστήματος μας

Όπως βλέπουμε ο KinectServer αναλαμβάνει αρχικά την επεξεργασία των δεδομένων που διαβάζουμε από το Kinect και έπειτα στέλνει τα δεδομένα αυτά στον UnityClient έπειτα από το κατάλληλο serialization σε xml.



Εικόνα 4.2: Kinect Server

4.3 Υλοποίηση

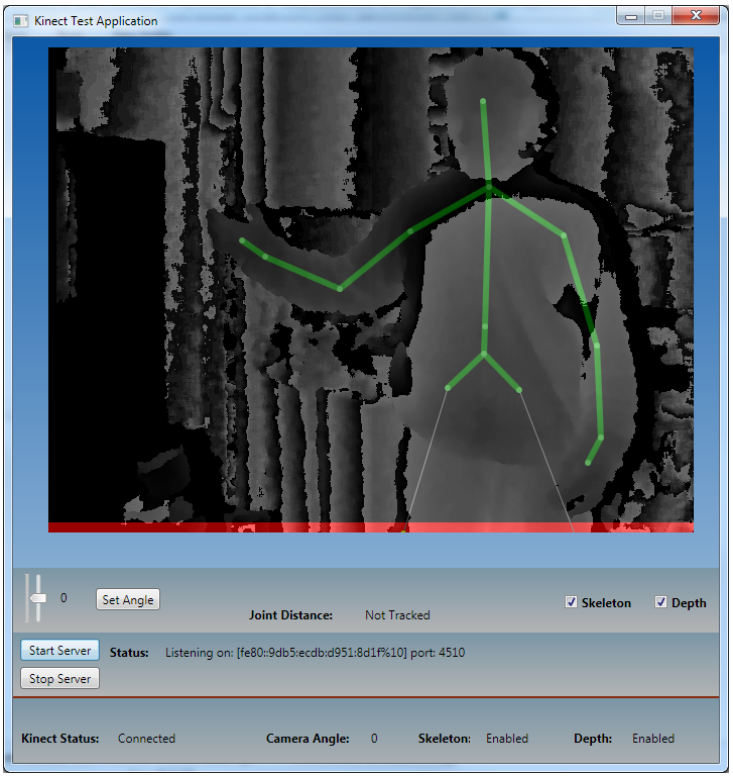
Ξεκινώντας την υλοποίηση αρχικά δημιουργήθηκε μια Windows Desktop εφαρμογή για την εξοικείωση μας με το Kinect for Windows SDK και το Kinect for Windows Toolkit, τα εργαλεία δηλαδή που έχουμε στη διάθεση μας όσον αφορά την επικοινωνία μας με το Kinect.

Η έκδοση του SDK που χρησιμοποιήθηκε είναι η v1.8 καθώς ήταν η πιο ενημερωμένη έκδοση εκείνη την περίοδο και ουσιαστικά είναι και η τελευταία έκδοση του SDK η οποία υποστηρίζεται από το Kinect v1 το οποίο και είχαμε στην διάθεση μας.

Σε αυτό το στάδιο είδαμε τη βασική λειτουργικότητα του Kinect όπως την ενεργοποίηση του αισθητήρα και των ροών δεδομένων που θέλουμε και πως μπορούμε έπειτα να πάρουμε αυτά τα δεδομένα και να τα χρησιμοποιήσουμε σε κάποια δική μας εφαρμογή.

Οι drivers του Kinect for Windows υποστηρίζουν τη χρήση πολλαπλών αισθητήρων σε έναν υπολογιστή. Το API περιλαμβάνει λειτουργίες που απαριθμούν τους αισθητήρες, έτσι ώστε να μπορούμε να προσδιορίσουμε πόσα Kinect είναι συνδεδεμένα στον υπολογιστή, να πάρουμε το όνομα ενός συγκεκριμένου αισθητήρα, και να ορίσουμε τα χαρακτηριστικά ροής για κάθε αισθητήρα. Μετά την εύρεση ενός συνδεδεμένου αισθητήρα, θα πρέπει να ενεργοποιήσουμε κάποια ή όλα τα ρεύματα δεδομένων (χρώμα, βάθος, και σκελετός) και στη συνέχεια να ξεκινήσουμε τον αισθητήρα ώστε να αρχίσει η αναπαραγωγή (streaming) των δεδομένων.

Στην εικόνα 4.3 μπορούμε να δούμε μια από τις εφαρμογές που αναπτύχθηκαν για την εξοικείωση μας με τον αισθητήρα και το SDK.



Εικόνα 4.3: Η δική μας Kinect for Windows Application

Σε αυτό το βήμα επίσης υλοποιήθηκε και ένας socket server ο οποίος ανέλαβε να στέλνει μερικά από τα δεδομένα που παράγει το Kinect σε μια client εφαρμογή επίσης σε Windows Desktop περιβάλλον για δοκιμή της επικοινωνίας μέσω sockets ανάμεσα στην κύρια εφαρμογή που χειρίζεται τον αισθητήρα και μιας δευτερεύουσας που απλά θα λαμβάνει δεδομένα του αισθητήρα.

Για περισσότερες πληροφορίες σχετικά με την ενεργοποίηση του Kinect και των ροών δεδομένων και τη δημιουργία ενός Asynchronous Socket Server και Client μπορούμε να αναφερθούμε στην επίσημη ιστοσελίδα του Microsoft Developer Network.

- https://msdn.microsoft.com/en-us/library/hh855348.aspx
- https://msdn.microsoft.com/en-us/library/fx6588te%28v=vs.110%29.aspx
- https://msdn.microsoft.com/en-us/library/bew39x2a%28v=vs.110%29.aspx

Το επόμενο βήμα περιελάμβανε την εγκαθίδρυση της σύνδεσης μέσω sockets του Unity web player με μια Windows Desktop εφαρμογή.

Λόγω τον περιορισμών που υπάρχουν στη χρήση των sockets στον webplayer του Unity, (ο webplayer εφαρμόζει ένα πρότυπο ασφάλειας πολύ παρόμοιο με αυτό που χρησιμοποιείται από τον Adobe Flash player), για να μπορέσουμε να πετύχουμε τη σύνδεση ο server μας κατά την εγκαθίδρυση της χρειάζεται να σερβίρει στον client ένα αρχείο που να περιέχει το socket policy σε μορφή xml.

Στην εικόνα 4.1 μπορούμε να δούμε ένα παράδειγμα του xml το οποίο περιμένει ο client ώστε να γίνει εφικτή η σύνδεση.

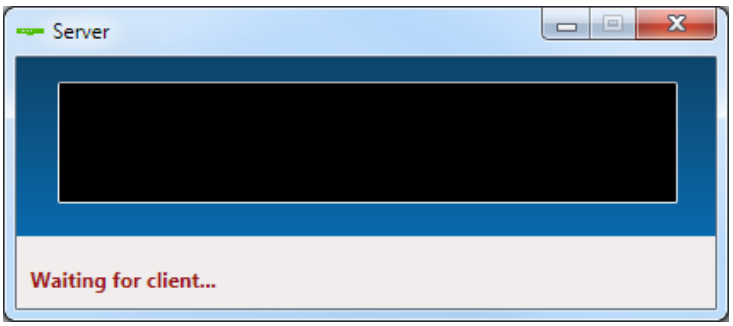
```
1 @"<?xml version='1.0'?>
2   <cross-domain-policy>
3       <allow-access-from domain=""*"" to-ports=""*"" />
4   </cross-domain-policy>";
```

Μπλοκ Κώδικα 4.1: Policy xml

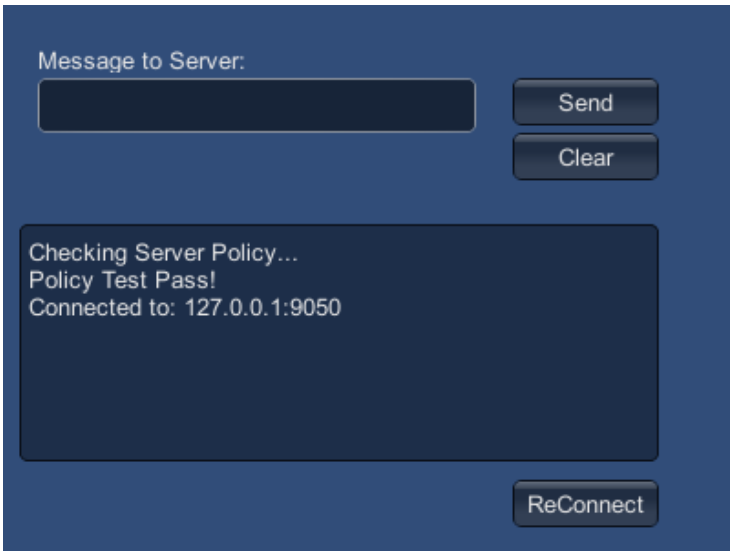
Ο Unity webplayer περιμένει το xml αυτό εξ'ορισμού στην πόρτα 843 μπορεί όμως να φιλοξενηθεί και σε κάποια άλλη πόρτα.

Στο Παράρτημα Β μπορούμε να δούμε τον Policy Server που χρησιμοποιήθηκε γ'αυτό το σκοπό και στις εικόνες 4.4 και 4.5 μπορούμε να δούμε τον Server και τον Client που υλοποιήθηκαν για την δοκιμή της σύνδεσης.

Περισσότερες πληροφορίες σχετικά με το πρότυπο ασφαλείας του web player του Unity μπορούμε να δούμε στην επίσημη σελίδα του εγχειρίδιου του Unity. http://docs.unity3d.com/Manual/SecuritySandbox.html



Εικόνα 4.4: Test Server



Εικόνα 4.5: Test Client

Το κομμάτι της αναγνώρισης ομιλίας δοκιμάστηκε και αυτό αρχικά σε ξεχωριστή εφαρμογή κάνοντας χρήση του Microsoft.Speech API το οποίο είναι απαραίτητο για να μπορέσουμε να κάνουμε αναγνώριση ομιλίας μέσω του Kinect.

Η ομιλία είναι ένα αποτελεσματικός και φυσικός τρόπος για τους ανθρώπους να αλληλεπιδρούν με τις εφαρμογές και μπορεί να συμπληρώσει ή ακόμη και να αντικαταστήσει τη χρήση του ποντικιού, του πληκτρολογίου, των controllers και των χειρονομιών.

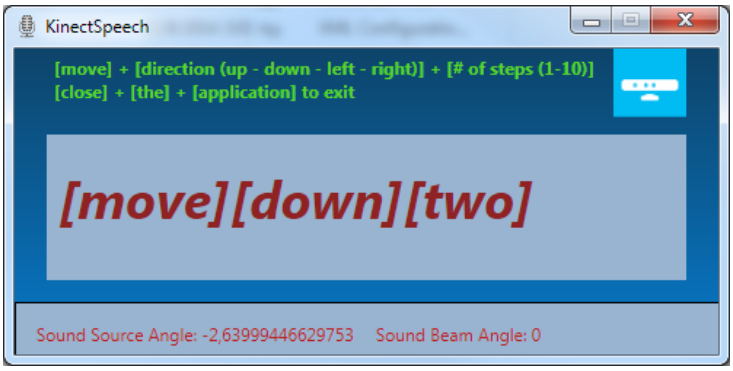
Χρησιμοποιώντας τα API της πλατφόρμας Speech SDK 11, μπορούμε να δημιουργήσουμε τις δικές μας γραμματικές αναγνώρισης ομιλίας τις οποίες έπειτα μπορούμε να χρησιμοποιήσουμε στις εφαρμογές μας.

Το σύστημα μας υλοποιήθηκε ώστε να μπορεί να αναγνωρίζει τις παρακάτω εντολές:

- εντολές του τύπου [move] + [direction] + [number of steps] όπως για παράδειγμα [move][down][two] ώστε να μπορούν οι εντολές αυτές να χρησιμοποιηθούν ως είσοδος για την κίνηση κάποιου χαρακτήρα στο Unity.
- [choose] για την ένδειξη κάποιας επιλογής μέσα στο Unity.
- [release] για την ένδειξη κάποιας αποεπιλογής μέσα στο Unity.
- [change focus] για την εναλλαγή ενεργού χαρακτήρα.
- [stop] για το σταμάτημα της κίνησης ενός χαρακτήρα.
- [close the server] η οποία τερματίζει την εφαρμογή.

Περισσότερες πληροφορίες σχετικά με το Microsoft Speech Platform SDK μπορείτε να δείτε στην επίσημη ιστοσελίδα του Microsoft Developer Network. https://msdn.microsoft.com/en-us/library/hh378466%28v=office.14%29.aspx

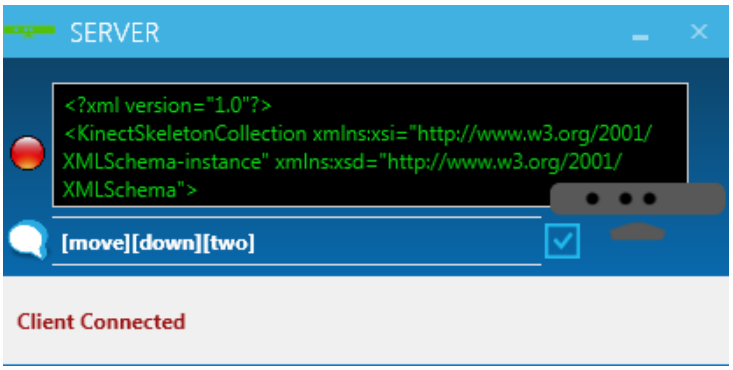
Στην εικόνα 4.6 μπορούμε να δούμε την εφαρμογή που υλοποιήθηκε για την δοκιμή του Microsoft Speech Platform SDK και την υλοποίηση της γραμματικής μας.



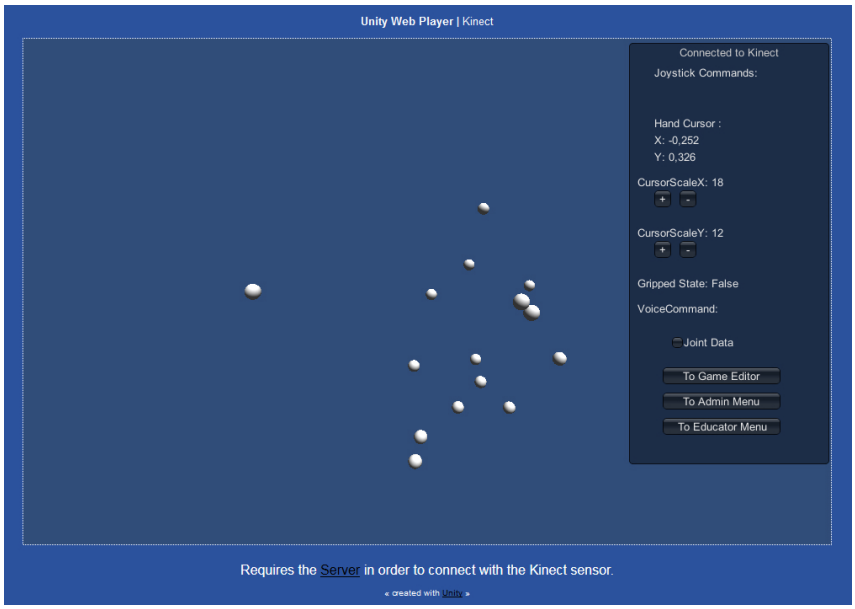
Εικόνα 4.6: Kinect & Microsoft Speech Platform SDK Aplication

Έχοντας πλέον στα χέρια μας όλα τα δομικά στοιχεία που χρειάζονται για την εφαρμογή μας προχωρήσαμε στην υλοποίηση του τελικού Kinect Server στον οποίο ενσωματώθηκαν όλες οι λειτουργίες και παράλληλα υλοποιήθηκε και ο τελικός client ο οποίος τώρα λαμβάνει το xml με τα δεδομένα που του στέλνει ο Kinect Server.

Στις εικόνες 4.6 και 4.7 μπορούμε να δούμε την τελική μορφή του Server και του Client αντιστοιχα.



Εικόνα 4.7: Ο τελικός Kinect Server



Εικόνα 4.8: Unity Kinect Client

Η λειτουργικότητα του συστήματος έχει ως εξής:

Αρχικά γίνεται η εκκίνησή του Policy Server ο οποίος τρέχει στο background και περιμένει τη στιγμή που θα του γίνει κάποιο request στην πόρτα 843 από τον client του Unity ώστε να σερβίρει το κατάλληλο policy xml αρχείο για να γίνει δυνατή ή σύνδεση.

Στη συνέχεια γίνεται η εκκίνησή του Kinect και των ροών δεδομένων που μας ενδιαφέρουν καθώς και η εκκίνησή του Data Server ο οποίος περιμένει τη σύνδεση κάποιου client στη πόρτα 9050 και αναλαμβάνει την αποστολή των δεδομένων του Kinect στον client.

Τη στιγμή που το Kinect αρχίσει να καταγράφει κάποιον χρήστη, ξεκινάει παράλληλα και το serialization των δεδομένων του χρήστη και αν είναι συνδεδεμένος και ο client τότε γίνεται άμεσα και η αποστολή των δεδομένων αυτών.

Πιο συγκεκριμένα κάθε φορά που είναι έτοιμο κάποιο SkeletonFrame καλείται επίσης η SerializeToXML η οποία αναλαμβάνει να κάνει serialize τα δεδομένα του εκάστοτε frame.

```
private void kinect_SkeletonFrameReady(object sender,
    SkeletonFrameReadyEventArgs e)
        {
          ..............
          xml = users.SerializeToXML(voiceCommand, cursorX, cursorY, isGripped);
         ..............
        }
```

Serialization είναι η διαδικασία μετατροπής της κατάστασης ενός αντικειμένου σε μια μορφή που να μπορεί να γίνει persisted η να μεταφερθεί. Η συμπληρωματική διαδικασία του serialization είναι το deserialization, η οποία μετατρέπει ένα stream σε ένα αντικείμενο. Μαζί, αυτές οι μέθοδοι επιτρέπουν σε δεδομένα να αποθηκεύονται εύκολα και να μεταφέρονται.

Στην περίπτωση μας η SerializeToXML αναλαμβάνει τη δημιουργία του πακέτου της πληροφορίας που θα σταλεί στον client.

```
public static string SerializeToXML(this List<Skeleton> skeletons, string vc,
    string cursorX, string cursorY, bool isGripped)
        {
            KinectSkeletonCollection xmlSkeletons = new KinectSkeletonCollection
     { Skeletons = new List<KinectSkeleton>() };

            xmlSkeletons.KinectVoiceCommand = vc;

            xmlSkeletons.kinectCursorX = cursorX;
            xmlSkeletons.kinectCursorY = cursorY;

            xmlSkeletons.isHandGripped = isGripped;

            foreach (var skeleton in skeletons)
            {
                KinectSkeleton xmlSkeleton = new KinectSkeleton
                {
                    ID = skeleton.TrackingId.ToString(),
                    Joints = new List<KinectJoint>()
                };

                foreach (Joint joint in skeleton.Joints)
                {
                    //Joint scaled = joint.ScaleTo(640, 480);
                    Joint scaled = joint.ScaleTo(960, 600);

                    xmlSkeleton.Joints.Add(new KinectJoint
                    {
                        Name = joint.JointType.ToString(),
                        X = scaled.Position.X,
                        Y = scaled.Position.Y,
                        Z = scaled.Position.Z
                    });
```

```
                }
                xmlSkeletons.Skeletons.Add(xmlSkeleton);
            }
            return SerializeToXML(xmlSkeletons);
        }
```

Επειδή η XML είναι ένα ανοιχτό πρότυπο, τα XML streams μπορούν να υποβληθούν σε επεξεργασία από οποιαδήποτε εφαρμογή, ανάλογα με τις ανάγκες, ανεξαρτήτως πλατφόρμας.

```
private static string SerializeToXML(object obj)
        {
            XmlSerializer serializer = new XmlSerializer(typeof(
    KinectSkeletonCollection));

            using (MemoryStream ms = new MemoryStream())
            {
                serializer.Serialize(ms, obj);
                return Encoding.Default.GetString(ms.ToArray());
            }
        }
```

Στην περίπτωση μας το xml που δημιουργείται αποτελείται από μια λίστα από KinectSkeleton καθώς επίσης και από την τρέχουσα φωνητική εντολή, αν υπάρχει κάποια, τις x,y συντεταγμένες του κέρσορα του χεριού και αν το ενεργό χέρι βρίσκεται σε κατάσταση grip, εάν είναι δηλαδή με κλειστή παλάμη σαν να κρατάει κάτι.

```
public class KinectSkeletonCollection
    {
        public List<KinectSkeleton> Skeletons { get; set; }
        public string KinectVoiceCommand { get; set; }
        public string kinectCursorX { get; set; }
        public string kinectCursorY { get; set; }
        public bool isHandGripped { get; set; }
    }
```

Η KinectSkeleton κλάση με τη σειρά της αποτελείται από το ID του τρέχοντος σκελετού και από μια λίστα από KinectJoint

```
public class KinectSkeleton
    {
        public string ID { get; set; }
        public List<KinectJoint> Joints { get; set; }
    }
```

τα οποία KinectJoint περιέχουν τις x,y,z συντεταγμένες της κάθε άρθρωσης που κάνει track το Kinect καθώς και το όνομα της κάθε άρθρωσης.

```
public class KinectJoint
    {
        public string Name { get; set; }
        public double X { get; set; }
        public double Y { get; set; }
        public double Z { get; set; }
    }
```

Στη συνέχεια το xml με τα δεδομένα αποστέλλεται από τον server στον client του Unity.

```
byte[] message = Encoding.ASCII.GetBytes(xml);

  client.BeginSend(message, 0, message.Length, SocketFlags.None,
        new AsyncCallback(SendData), client);
```

Ένα δείγμα ολόκληρου του xml το οποίο παράγεται και αποστέλλεται στον client μπορούμε να δούμε στο παράρτημα Α.

Στον Unity Client στην συνέχεια το deserialization των δεδομένων

```
xmlSkeletons = Deserialize<KinectSkeletonCollection>(stringData);
```

```
private static T Deserialize<T>(string str)
    {
        XmlDocument xdoc = new XmlDocument();
        try
        {
            xdoc.LoadXml(str);
            XmlNodeReader reader = new XmlNodeReader(xdoc.DocumentElement);
            XmlSerializer ser = new XmlSerializer(typeof(T));
            object obj = ser.Deserialize(reader);
            return (T)obj;
        }
        catch
        {
            return default(T);
        }
    }
```

και έπειτα γίνεται η αντιστοίχηση στο κατάλληλο KinectJoint.

```
_Head = xmlSkeletons.Skeletons[0].Joints[3];
..............
```

```
HeadVector.x = (float)_Head.X * scale;
HeadVector.y = (float)_Head.Y * -scale;
HeadVector.z = (float)_Head.Z * scale;
..............
```

Για την αναπαράσταση του αρθρώσεων του σκελετού στη σκηνή χρησιμοποιήθηκαν σφαίρες GameObject τα οποία τοποθετούνται στην κατάλληλη θέση ως εξής.

```
Head.transform.position = camera.ScreenToViewportPoint(HeadVector);
..............
```

Όπως βλέπουμε η τιμή των συντεταγμένων κάθε άρθρωσης περιέχεται σε μια μεταβλητή τύπου Vector3 το οποίο παίρνει σαν όρισμα η ScreenToViewportPoint ώστε να τοποθετήσει το αντίστοιχο GameObject στην κατάλληλη θέση στον 3D χώρο.

Μια απλή αναγνώριση χειρονομιών για χρήση ως input για joystick commands υλοποιήθηκε ως εξής.

Έχοντας επιλέξει το δεξί χέρι ως κύριο και υπολογίζοντας την απόσταση του από δυο σταθερά σημεία αναφοράς του σώματος, τον αριστερό ώμο και το κεφάλι στην περίπτωση μας, μπορούμε να προσδιορίσουμε την απόκλιση της κίνησης και να δώσουμε τις ανάλογες εντολές.

Πιο συγκεκριμένα όταν η απόσταση του δεξιού χεριού προς το κεφάλι μικραίνει ξέρουμε ότι το χέρι σηκώνεται προς τα πάνω και όταν αυξάνεται ξέρουμε ότι το χέρι κατεβαίνει.

Αντίστοιχα όταν η απόσταση του δεξιού χεριού προς τον αριστερό ώμο μικραίνει ξέρουμε ότι το χέρι κινήθηκε προς τα αριστερά ενώ όταν αυξάνεται ξέρουμε ότι το χέρι κινήθηκε προς τα δεξιά.

Έτσι προσδιορίζοντας και μια εικονική περιοχή αδράνειας μπορούμε πλέον να πάρουμε τις κατάλληλες εντολές που χρειαζόμαστε για την κίνηση ενός χαρακτήρα μέσω χειρονομιών.

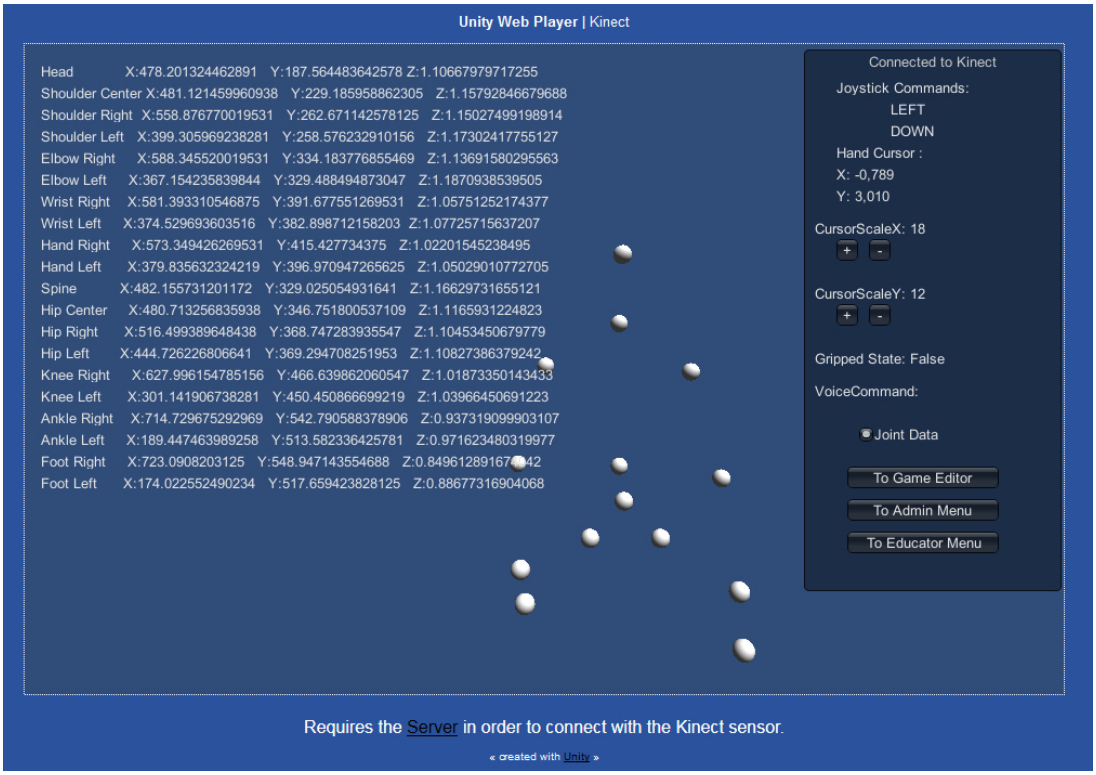
Για τον υπολογισμό της απόστασης μεταξύ δύο αρθρώσεων μπορούμε να κάνουμε χρήση του Πυθαγόρειου θεωρήματος

```
private double GetJointDistance(KinectJoint firstJoint, KinectJoint secondJoint)
        {
            double distanceX = firstJoint.X - secondJoint.X;
            double distanceY = firstJoint.Y - secondJoint.Y;
            double distanceZ = firstJoint.Z - secondJoint.Z;
            return (double)Math.Sqrt(Math.Pow(distanceX, 2) + Math.Pow(distanceY
    , 2) + Math.Pow(distanceZ, 2));
```

```
7            }
```

Εναλλακτικά ένας δεύτερος και καλύτερος τρόπος να πετύχουμε την ίδια λειτουργικότητα είναι να ελέγχουμε την τιμή των InteractionHandPointer.X και InteractionHandPointer.Y του Microsoft.Kinect.Toolkit.Interaction (microsoft.kinect.toolkit.interaction.dll).

Με αυτόν το τρόπο μπορούμε εύκολα να καταλάβουμε πότε το χέρι κινήθηκε και προς ποια κατεύθυνση απλά ελέγχοντας την τιμή των InteractionHandPointer.X και InteractionHandPointer.Y.



Επίλογος

5.1 Συμπεράσματα

Ο τομέας της αλληλεπίδρασης ανθρώπου-υπολογιστή παίζει σήμερα έναν ζωτικής σημασίας ρόλο στην έρευνα της επιστήμης των υπολογιστών. Η κατανόηση του πώς να δημιουργήσουμε το υλικό και το λογισμικό του υπολογιστή έτσι ώστε να διευκολυνθεί η χρήση τους από τους ανθρώπους είναι απλά μια θεμελιώδη περιοχή της επιστήμης των υπολογιστών. Ακόμη και τα πιο γρήγορα, πιο ισχυρά συστήματα θα είναι άχρηστα εάν οι άνθρωποι δεν μπορούν να τα κατανοήσουν επαρκώς και να τα χρησιμοποιήσουν.

Η δυνατότητα που μας δίνεται πλέον να μπορούμε να χειριζόμαστε τεχνολογικές συσκευές με φυσικό τρόπο σίγουρα ανοίγει νέους ορίζοντες και θα επιφέρει τεράστια αλλαγή στον τρόπο που αλληπεδρούμε με αυτές.

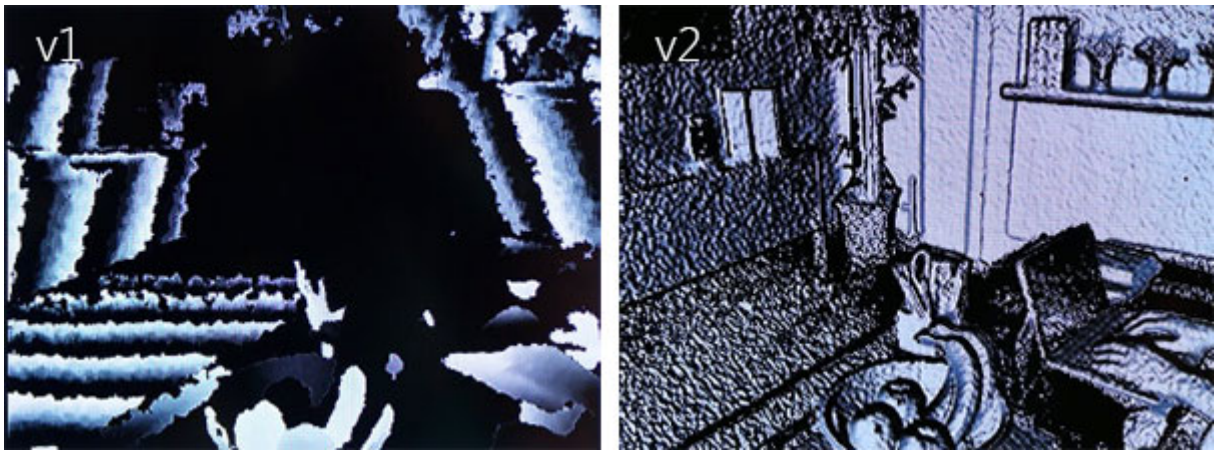
Όσον αφορά την εργασία μας είδαμε ξεκάθαρα κάποια από τα εργαλεία τα οποία θα έχουν οι προγραμματιστές στη διάθεση τους ώστε να δημιουργήσουν εφαρμογές φυσικής αλληλεπίδρασης χρήστη και παρόλο που όπως είπαμε οι τεχνολογίες αυτές είναι ακόμη σε πρώιμο στάδιο κάτι που σημαίνει ότι έχουμε ακόμη λίγο χρόνο μέχρι να φτάσουμε στην τελειοποίηση τους και την ευρεία αποδοχή τους από το κοινό, η λειτουργικότητα που μας δίνεται ήδη στη διάθεση μας είναι ήδη αρκετά ενδιαφέρουσα και συναρπαστική και αναμένεται φυσικά να γίνει αρκετά καλύτερη στην συνέχεια.

5.2 Μελλοντική Εργασία και Επεκτάσεις

Θα μπορούσε να πει κάποιος ότι ίσως ήταν χρήσιμη και μια πιο ολοκληρωμένη υλοποίηση με σκοπό την εισαγωγή περισσότερων στοιχείων του Kinect στο Unity, ήδη η Microsoft όμως έχει κάνει το επόμενο βήμα με την v2 έκδοση του Kinect και του SDK τα οποία αποτελούν μια πολύ καλή αναβάθμιση των ήδη υπαρχόντων και υπόσχονται ένα νέο επίπεδο στην αλληλεπίδραση χρήστη υπολογιστή προσφέροντας μεγαλύτερη συνολική ακρίβεια, ανταπόκριση και διαισθητικές ικανότητες ώστε να επιταχυνθεί η ανάπτυξη των εφαρμογών που ανταποκρίνονται σε κίνηση, χειρονομίες, και φωνή.

Το SDK 2.0 παρέχει εκτός των άλλων και επίσημη υποστήριξη για το Unity μέσω ενός Unity Package το οποίο είναι διαθέσιμο για την Pro έκδοση.

Στην εικόνα 5.1 μπορούμε να δούμε χαρακτηριστικά τον βαθμό της βελτίωσης στην πιστότητα που έχει υποστεί ο αισθητήρας και στην εικόνα 5.2 μπορούμε να δούμε πως είναι εξωτερικά ο αισθητήρας.



Εικόνα 5.1: Kinect v1 Vs Kinect v2

Περισσότερες πληροφορίες για την v2 έκδοση του Kinect και του SDK μπορούμε να δούμε στην επίσημη ιστοσελίδα της Microsoft.

http://www.microsoft.com/en-us/kinectforwindows/meetkinect/features.aspx
https://msdn.microsoft.com/en-us/library/dn782041.aspx

Εικόνα 5.2: Kinect v2 sensor

Αναφορές - Πηγές

1. • Wikipedia, The free encyclopedia
   • http://en.wikipedia.org/wiki/Main_Page
2. • MSDN, The microsoft developer network
   • https://msdn.microsoft.com/en-US/
3. • Unity Technologies, Unity Manual
   • http://docs.unity3d.com/Manual/index.html
4. • Microsoft, Kinect for Windows
   • http://www.microsoft.com/en-us/kinectforwindows/
5. • Zigfu, Kinect Development in HTML, Unity3D and Flash
   • http://zigfu.com/
6. • ETC Internal Unity3D Wiki, Microsoft Kinect - Microsoft SDK
   • http://wiki.etc.cmu.edu/unity3d/index.php/Microsoft_Kinect_-_Microsoft_SDK
7. • Kinect with MS-SDK by RF Solutions, Unity Asset Store
   • http://u3d.as/content/rf-solutions/kinect-with-ms-sdk/4sb
8. • KinectExtras with MsSDK by RF Solutions, Unity Asset Store
   • http://u3d.as/content/rf-solutions/kinect-extras-with-ms-sdk/5ej
9. • UCSIM, Center for Simulations & Virtual Environments Research
   • http://ucsim.uc.edu/blog/

Παράρτημα Α

Α.1 Μπλοκ Κώδικα

Α.1.1 PolicyServer

```
class PolicyServer
    {
        TcpListener _Listener = null;
        TcpClient _Client = null;

        static ManualResetEvent _TcpClientConnected = new ManualResetEvent(false
    );
        const string _PolicyRequestString = "<policy-file-request/>";

        int _ReceivedLength = 0;
        //byte[] _Policy = null;
        byte[] _ReceiveBuffer = null;

        byte[] policyBytes;

        private void InitializeData()
        {
            _ReceiveBuffer = new byte[_PolicyRequestString.Length];

            const string AllPolicy =
            @"<?xml version='1.0'?>
            <cross-domain-policy>
                    <allow-access-from domain=""*"" to-ports=""*"" />
            </cross-domain-policy>";

            policyBytes = Encoding.UTF8.GetBytes(AllPolicy);
        }

        public void RunPolicyServer()
        {
            InitializeData();

            try
            {
                _Listener = new TcpListener(IPAddress.Any, 843);
                _Listener.Start();

                while (true)
                {
                    _TcpClientConnected.Reset();
                    _Listener.BeginAcceptTcpClient(new AsyncCallback(
    OnBeginAccept), null);
                    _TcpClientConnected.WaitOne(); //Block until client connects
                }
            }
            catch (Exception exc)
```

```
45              {
46                  MessageBox.Show(exc.ToString());
47              }
48          }
49
50          private void OnBeginAccept(IAsyncResult ar)
51          {
52              _Client = _Listener.EndAcceptTcpClient(ar);
53              _Client.Client.BeginReceive(_ReceiveBuffer, 0, _PolicyRequestString.
    Length, SocketFlags.None,
54                  new AsyncCallback(OnReceiveComplete), null);
55          }
56
57          private void OnReceiveComplete(IAsyncResult ar)
58          {
59              try
60              {
61                  _ReceivedLength += _Client.Client.EndReceive(ar);
62
63                  //See if there's more data that we need to grab
64                  if (_ReceivedLength < _PolicyRequestString.Length)
65                  {
66                      //Need to grab more data so receive remaining data
67                      _Client.Client.BeginReceive(_ReceiveBuffer, _ReceivedLength,
68                          _PolicyRequestString.Length - _ReceivedLength,
69                          SocketFlags.None, new AsyncCallback(OnReceiveComplete),
    null);
70                      return;
71                  }
72
73                  //Check that <policy-file-request/> was sent from client
74                  string request = System.Text.Encoding.UTF8.GetString(
    _ReceiveBuffer, 0, _ReceivedLength);
75
76                  if (StringComparer.InvariantCultureIgnoreCase.Compare(request,
    _PolicyRequestString) != 0)
77                  {
78                      MessageBox.Show("Data received isn't a valid policy request!
    ");
79                      //Data received isn't valid so close
80                      _Client.Client.Close();
81                      return;
82                  }
83
84                  //Valid request received....send policy file
85                  _Client.Client.BeginSend(policyBytes, 0, policyBytes.Length,
    SocketFlags.None,
86                      new AsyncCallback(OnSendComplete), null);
87
88              }
89              catch (Exception exc)
90              {
91                  MessageBox.Show(exc.ToString());
92                  _Client.Client.Close();
93              }
94
95              _ReceivedLength = 0;
96              _TcpClientConnected.Set(); //Allow waiting thread to proceed
97          }
98
99          private void OnSendComplete(IAsyncResult ar)
100         {
101
```

```
            try
            {
                _Client.Client.Close();
            }
            catch (Exception exc)
            {
                MessageBox.Show(exc.ToString());
            }
            finally
            {
                _Client.Client.Close();
            }
        }

        public void StopPolicyServer()
        {
            _Listener.Stop();
            _TcpClientConnected.Close();
        }
    }
```

Μπλοκ Κώδικα A.1: Policy Server Code

A.1.2 Data Server

```
private byte[] data = new byte[5120];
        private int size = 5120;
        private Socket server;

        void StartServer()
        {
            try
            {
                server = new Socket(AddressFamily.InterNetwork,
                                    SocketType.Stream, ProtocolType.Tcp);
                IPEndPoint iep = new IPEndPoint(IPAddress.Any, 9050);
                server.Bind(iep);
                server.Listen(-1);
                server.BeginAccept(new AsyncCallback(AcceptConn), server);

                this.Dispatcher.BeginInvoke(DispatcherPriority.Normal, (
   ThreadStart)delegate()
                {
                    Label.Content = "Waiting for client...";
                }
                );
            }
            catch (Exception ex)
            {
                MessageBox.Show(ex.ToString());
            }
        }

        void AcceptConn(IAsyncResult iar)
        {
            Socket oldserver = (Socket)iar.AsyncState;

            Socket client = oldserver.EndAccept(iar);

            client.NoDelay = true;

```

```
36              this.Dispatcher.BeginInvoke(DispatcherPriority.Normal, (ThreadStart)
    delegate()
37              {
38                  Label.Content = "Client Connected";
39              }
40              );
41
42              string stringData = "";
43
44              byte[] message1 = Encoding.ASCII.GetBytes(stringData);
45              client.BeginSend(message1, 0, message1.Length, SocketFlags.None,
46                          new AsyncCallback(SendData), client);
47          }
48
49          void SendData(IAsyncResult iar)
50          {
51              try
52              {
53                  Socket client = (Socket)iar.AsyncState;
54                  int sent = client.EndSend(iar);
55                  client.NoDelay = true;
56
57                  client.BeginReceive(data, 0, size, SocketFlags.None,
58                              new AsyncCallback(ReceiveData), client);
59              }
60              catch (Exception)
61              {
62                  //throw;
63              }
64          }
65
66          void ReceiveData(IAsyncResult iar)
67          {
68              try
69              {
70                  Socket client = (Socket)iar.AsyncState;
71                  int recv = client.EndReceive(iar);
72                  client.NoDelay = true;
73
74                  if (recv == 0)
75                  {
76                      client.Close();
77
78                      this.Dispatcher.BeginInvoke(DispatcherPriority.Normal, (
    ThreadStart)delegate()
79                      {
80                          Label.Content = "Waiting for client...\n";
81                      }
82                      );
83
84                      server.BeginAccept(new AsyncCallback(AcceptConn), server);
85                      return;
86                  }
87
88                  byte[] message2 = Encoding.ASCII.GetBytes(xml);
89
90                  client.BeginSend(message2, 0, message2.Length, SocketFlags.None,
91                              new AsyncCallback(SendData), client);
92
93                  //voiceCommand = null;
94                  //Thread.Sleep(1000);
95              }
96              catch (Exception)
```

```
97              {
98                  this.Dispatcher.BeginInvoke(DispatcherPriority.Normal, (
    ThreadStart)delegate()
99                  {
100                     Label.Content = "Waiting for client...\n";
101                 }
102                 );
103
104                 //throw;
105             }
106         }
```

Μπλοκ Κώδικα A.2: Data Server Code

A.1.3 Kinect Code

```
1  public KinectSensor kinect = null;
2
3  private InteractionStream _interactionStream;
4
5  private UserInfo[] _userInfos; //the information about the interactive users
6
7  private Skeleton[] skeletonData = new Skeleton[6];
8
9  private string xml = string.Empty;
10
11 const string RecognizerId = "SR_MS_en-US_Kinect_11.0";
12
13 KinectAudioSource source;
14
15 bool keepRunning = true;
16
17 public String voiceCommand;
18
19 public String cursorX;
20
21 public String cursorY;
22
23 public bool isGripped;
24
25
26         void StartKinectST()
27         {
28
29             kinect = KinectSensor.KinectSensors.FirstOrDefault(s => s.Status ==
    KinectStatus.Connected); // Get first Kinect Sensor
30
31             kinect.SkeletonStream.Enable(); // Enable skeletal tracking
32
33             skeletonData = new Skeleton[kinect.SkeletonStream.
    FrameSkeletonArrayLength]; // Allocate ST data
34
35             _userInfos = new UserInfo[InteractionFrame.UserInfoArrayLength];
36
37             kinect.SkeletonFrameReady += new EventHandler<
    SkeletonFrameReadyEventArgs>(kinect_SkeletonFrameReady); // Get Ready for
    Skeleton Ready Events
38
39             kinect.DepthStream.Enable();
40             kinect.DepthFrameReady += kinect_DepthFrameReady;
41
```

```
42              _interactionStream = new InteractionStream(kinect, new
    DummyInteractionClient());
43              _interactionStream.InteractionFrameReady +=
    InteractionStreamOnInteractionFrameReady;
44
45              kinect.Start(); // Start Kinect sensor
46
47              StartKinectRecognizer();
48          }
49
50          private void kinect_SkeletonFrameReady(object sender,
    SkeletonFrameReadyEventArgs e)
51          {
52              using (SkeletonFrame skeletonFrame = e.OpenSkeletonFrame()) // Open
    the Skeleton frame
53              {
54                  if (skeletonFrame != null && this.skeletonData != null) // check
     that a frame is available
55                  {
56                      skeletonFrame.CopySkeletonDataTo(this.skeletonData); // get
    the skeletal information in this frame
57
58                      var accelerometerReading = kinect.
    AccelerometerGetCurrentReading();
59                      _interactionStream.ProcessSkeleton(skeletonData,
    accelerometerReading, skeletonFrame.Timestamp);
60
61
62
63                      #region  XML Serialization
64                      var users = skeletonData.Where(s => s.TrackingState ==
    SkeletonTrackingState.Tracked).ToList();
65
66                      if (users.Count > 0)
67                      {
68                          xml = users.SerializeToXML(voiceCommand, cursorX,
    cursorY, isGripped);
69
70                          this.Dispatcher.BeginInvoke(DispatcherPriority.Normal, (
    ThreadStart)delegate()
71                          {
72                              Log.Text = xml;
73                          }
74                          );
75
76                          //Debug.WriteLine(xml);
77
78                          voiceCommand = null;
79                      }
80                      else
81                      {
82                          //Log.Text = "";
83                          //xml = null;
84                      }
85                      #endregion
86                  }
87              }
88          }
89
90          private void kinect_DepthFrameReady(object sender,
    DepthImageFrameReadyEventArgs depthImageFrameReadyEventArgs)
91          {
```

```
92              using (DepthImageFrame depthFrame = depthImageFrameReadyEventArgs.
    OpenDepthImageFrame())
93              {
94                  if (depthFrame == null)
95                      return;
96
97                  try
98                  {
99                      _interactionStream.ProcessDepth(depthFrame.GetRawPixelData()
    , depthFrame.Timestamp);
100                 }
101                 catch (InvalidOperationException)
102                 {
103                     // DepthFrame functions may throw when the sensor gets
104                     // into a bad state.  Ignore the frame in that case.
105                 }
106             }
107         }
108
109 private Dictionary<int, InteractionHandEventType> _lastLeftHandEvents = new
    Dictionary<int, InteractionHandEventType>();
110
111 private Dictionary<int, InteractionHandEventType> _lastRightHandEvents = new
    Dictionary<int, InteractionHandEventType>();
112
113         private void InteractionStreamOnInteractionFrameReady(object sender,
    InteractionFrameReadyEventArgs args)
114         {
115             using (var iaf = args.OpenInteractionFrame()) //dispose as soon as
    possible
116             {
117                 if (iaf == null)
118                     return;
119
120                 iaf.CopyInteractionDataTo(_userInfos);
121
122                 iaf.Dispose();
123             }
124
125             StringBuilder dump = new StringBuilder();
126
127             var hasUser = false;
128
129             foreach (var userInfo in _userInfos)
130             {
131                 var userID = userInfo.SkeletonTrackingId;
132                 if (userID == 0)
133                     continue;
134
135                 hasUser = true;
136                 dump.AppendLine("User ID = " + userID);
137                 dump.AppendLine("  Hands: ");
138                 var hands = userInfo.HandPointers;
139                 if (hands.Count == 0)
140                     dump.AppendLine("    No hands");
141                 else
142                 {
143                     foreach (var hand in hands)
144                     {
145                         var lastHandEvents = hand.HandType ==
    InteractionHandType.None
146                                                  ? _lastLeftHandEvents
147                                                  : _lastRightHandEvents;
```

```
148
149                         if (hand.HandEventType != InteractionHandEventType.None)
150                             lastHandEvents[userID] = hand.HandEventType;
151
152                         var lastHandEvent = lastHandEvents.ContainsKey(userID)
153                                                 ? lastHandEvents[userID]
154                                                 : InteractionHandEventType.None;
155
156
157                         if (lastHandEvent.ToString() == "Grip")
158                         {
159                             //lb.Content = "Grip";
160
161                             this.Dispatcher.BeginInvoke(DispatcherPriority.
    Normal, (ThreadStart)delegate()
162                             {
163                                 JointPositionLabel.Content = string.Format("X:
    {0} Y: {1}", hand.X.ToString("N3"), hand.Y.ToString("N3"));
164                                 red.Opacity = 0;
165                                 green.Opacity = 10;
166                             }
167                             );
168
169                             isGripped = true;
170
171                             cursorX = hand.X.ToString("N3");
172                             cursorY = hand.Y.ToString("N3");
173
174                         }
175                         if (lastHandEvent.ToString() == "GripRelease" ||
    lastHandEvent.ToString() == "None")
176                         {
177                             //lb.Content = "Release";
178
179                             this.Dispatcher.BeginInvoke(DispatcherPriority.
    Normal, (ThreadStart)delegate()
180                             {
181                                 JointPositionLabel.Content = "";
182                                 red.Opacity = 0.9;
183                                 green.Opacity = 0;
184                             }
185                             );
186
187                             isGripped = false;
188
189                             cursorX = hand.X.ToString("N3");
190                             cursorY = hand.Y.ToString("N3");
191                         }
192
193                         dump.AppendLine();
194                         dump.AppendLine("    HandType: " + hand.HandType);
195                         dump.AppendLine("    HandEventType: " + hand.
    HandEventType);
196                         dump.AppendLine("    LastHandEventType: " +
    lastHandEvent);
197                         dump.AppendLine("    IsActive: " + hand.IsActive);
198                         dump.AppendLine("    IsPrimaryForUser: " + hand.
    IsPrimaryForUser);
199                         dump.AppendLine("    IsInteractive: " + hand.
    IsInteractive);
200                         dump.AppendLine("    PressExtent: " + hand.PressExtent.
    ToString("N3"));
201                         dump.AppendLine("    IsPressed: " + hand.IsPressed);
```

```
202                         dump.AppendLine("    IsTracked: " + hand.IsTracked);
203                         dump.AppendLine("    X: " + hand.X.ToString("N3"));
204                         dump.AppendLine("    Y: " + hand.Y.ToString("N3"));
205                         dump.AppendLine("    RawX: " + hand.RawX.ToString("N3"))
    ;
206                         dump.AppendLine("    RawY: " + hand.RawY.ToString("N3"))
    ;
207                         dump.AppendLine("    RawZ: " + hand.RawZ.ToString("N3"))
    ;
208                     }
209                 }
210             }
211 
212             if (!hasUser)
213             {
214                 //tb.Text = "No user detected.";
215                 //Label2.Text = "No user detected.";
216                 //red.Opacity = 0.8;
217                 //green.Opacity = 0;
218             }
219             //else green.Opacity = 1;
220         }
221 
222         private void StartKinectRecognizer()
223         {
224             RecognizerInfo recognizerInfo = SpeechRecognitionEngine.
    InstalledRecognizers().Where(r => r.Id == RecognizerId).FirstOrDefault();
225 
226             if (recognizerInfo == null)
227             {
228                 MessageBox.Show("Could not find Kinect speech recognizer");
229                 return;
230             }
231 
232             Thread.Sleep(1000);
233             Thread newThread = new Thread(new ParameterizedThreadStart(
    BuildGrammarforRecognizer));
234             newThread.Start(recognizerInfo);
235         }
236 
237         private void BuildGrammarforRecognizer(object recognizerInfo)
238         {
239             EnableKinectAudioSource();
240 
241             var grammarBuilder = new GrammarBuilder { Culture = (recognizerInfo
    as RecognizerInfo).Culture };
242 
243             // first say move
244             grammarBuilder.Append(new Choices("move"));
245 
246             var directions = new Choices();
247             directions.Add("up");
248             directions.Add("down");
249             directions.Add("left");
250             directions.Add("right");
251             directions.Add("stop");
252 
253             // New Grammar builder for directions
254             grammarBuilder.Append(directions);
255 
256             // Another Grammar Builder for steps
257             grammarBuilder.Append(new Choices("one", "two", "three", "four", "
    five", "six", "seven", "eight", "nine", "ten"));
```

```
258
259             // Create Grammar from GrammarBuilder
260             var grammar = new Grammar(grammarBuilder);
261
262
263             var actions = new GrammarBuilder();
264             actions.Append(new Choices("choose", "release", "change focus"));
265
266             // New Grammar builder for actions
267             var grammarActions = new Grammar(actions);
268
269             // Creating another Grammar and load
270             var newGrammarBuilder = new GrammarBuilder();
271             newGrammarBuilder.Append("close the server");
272             var grammarClose = new Grammar(newGrammarBuilder);
273
274             int SamplesPerSecond = 16000;
275             int bitsPerSample = 16;
276             int channels = 1;
277             int averageBytesPerSecond = 32000;
278             int blockAlign = 2;
279
280             using (var speechRecognizer = new SpeechRecognitionEngine((
    recognizerInfo as RecognizerInfo).Id))
281             {
282                 speechRecognizer.LoadGrammar(grammar);
283                 speechRecognizer.LoadGrammar(grammarClose);
284                 speechRecognizer.LoadGrammar(grammarActions);
285
286                 speechRecognizer.SpeechRecognized += SreSpeechRecognized;
287                 speechRecognizer.SpeechHypothesized += SreSpeechHypothesized;
288                 speechRecognizer.SpeechRecognitionRejected +=
    SreSpeechRecognitionRejected;
289
290                 using (Stream s = source.Start())
291                 {
292
293                     speechRecognizer.SetInputToAudioStream(s, new
    SpeechAudioFormatInfo(EncodingFormat.Pcm, SamplesPerSecond, bitsPerSample,
    channels, averageBytesPerSecond, blockAlign, null));
294
295                     //speechRecognizer.SetInputToAudioStream(s, new
    SpeechAudioFormatInfo(44100, AudioBitsPerSample.Sixteen, AudioChannel.Mono))
    ;
296
297                     while (keepRunning)
298                     {
299                         try
300                         {
301                             RecognitionResult result = speechRecognizer.
    Recognize(new TimeSpan(0, 0, 5));
302                         }
303                         catch (Exception)
304                         {
305                             break;
306                             throw;
307                         }
308                     }
309
310                     speechRecognizer.RecognizeAsyncStop();
311                 }
312             }
313         }
```

```
314
315         private void EnableKinectAudioSource()
316         {
317             source = kinect.AudioSource;
318             //source.BeamAngleChanged += new EventHandler<
    BeamAngleChangedEventArgs>(source_BeamAngleChanged);
319             //source.SoundSourceAngleChanged += new EventHandler<
    SoundSourceAngleChangedEventArgs>(source_SoundSourceAngleChanged);
320             source.AutomaticGainControlEnabled = false;
321             source.NoiseSuppression = true;
322         }
323
324
325         private void SreSpeechRecognized(object sender,
    SpeechRecognizedEventArgs e)
326         {
327             float confidenceThreshold = 0.6f;
328
329             if (e.Result.Confidence > confidenceThreshold)
330             {
331                 Dispatcher.BeginInvoke(new Action<SpeechRecognizedEventArgs>(
    CommandsParser), e);
332             }
333         }
334
335         private void CommandsParser(SpeechRecognizedEventArgs e)
336         {
337             checkedImg.Opacity = 100;
338
339             var result = e.Result;
340
341             System.Collections.ObjectModel.ReadOnlyCollection<RecognizedWordUnit
    > words = e.Result.Words;
342
343             DisplayWords(result);
344
345             if (words[0].Text == "move")
346             {
347                 string direction = words[1].Text;
348                 switch (direction)
349                 {
350                     case "up": // Do Action
351                         break;
352                     case "down": // Do Action
353                         break;
354                     case "left": // Do Action
355                         break;
356                     case "right": // Do Action
357                         break;
358                     case "stop": // Do Action
359                         break;
360                     default:
361                         return;
362                 }
363
364                 var steps = words[2].Text;
365                 switch (steps)
366                 {
367                     case "one":// Do Action
368                         break;
369                     case "two":// Do Action
370                         break;
371                     case "three":// Do Action
```

```
                        break;
                    case "four":// Do Action
                        break;
                    default:
                        return;
                }
            }


            if (words[0].Text == "choose")
            {

            }

            if (words[0].Text == "release")
            {

            }

            //var actions = words[0].Text;
            //switch (actions)
            //{
            //    case "choose":// Do Action
            //        break;
            //    case "release":// Do Action
            //        break;
            //    case "change focus":// Do Action
            //        break;
            //    default:
            //        return;
            //}


            if (words[0].Text == "change" && words[1].Text == "focus")
            {
                // Do Action
            }

            if (words[0].Text == "close" && words[1].Text == "the" && words[2].
    Text == "server")
            {
                Close();
            }
        }


        private void SreSpeechRecognitionRejected(object sender,
    SpeechRecognitionRejectedEventArgs e)
        {
            if (e.Result != null)
            {
            }
        }

        void source_SoundSourceAngleChanged(object sender,
    SoundSourceAngleChangedEventArgs e)
        {
            //this.Dispatcher.BeginInvoke(DispatcherPriority.Normal, (
    ThreadStart)delegate()
            //{
            //    Label4.Content = "Sound Source Angle: " + e.Angle.ToString();
            //}
            //);
```

```
431         }
432 
433 
434         private void SreSpeechHypothesized(object sender,
    SpeechHypothesizedEventArgs e)
435         {
436 
437             this.Dispatcher.BeginInvoke(DispatcherPriority.Normal, (ThreadStart)
    delegate()
438             {
439                 checkedImg.Opacity = 0;
440                 speechlabel.Content = e.Result.Text;
441             }
442             );
443 
444             voiceCommand = null;
445 
446             //voiceCommand = e.Result.Text;
447         }
448 
449 
450         private void source_BeamAngleChanged(object sender,
    BeamAngleChangedEventArgs e)
451         {
452             //this.Dispatcher.BeginInvoke(DispatcherPriority.Normal, (
    ThreadStart)delegate()
453             //{
454             //    Label3.Content = "Sound Beam Angle: " + e.Angle.ToString();
455             //}
456             //);
457 
458         }
459 
460 
461         private void DisplayWords(RecognitionResult result)
462         {
463 
464             StringBuilder sb = new StringBuilder();
465 
466             foreach (var word in result.Words)
467             {
468                 sb.Append(string.Format("[{0}]", word.Text));
469             }
470 
471             //Debug.WriteLine(sb.ToString());
472 
473             //voiceCommand = sb.ToString();
474 
475             this.Dispatcher.BeginInvoke(DispatcherPriority.Normal, (ThreadStart)
    delegate()
476             {
477                 speechlabel.Content = sb.ToString();
478             }
479             );
480 
481             switch (sb.ToString())
482             {
483                 case "[choose]": voiceCommand = "choose";
484                     break;
485                 case "[release]": voiceCommand = "release";
486                     break;
487                 case "[change][focus]": voiceCommand = "changefocus";
488                     break;
```

```
                default: Debug.WriteLine("default");
                    break;

            }
            //Debug.WriteLine(voiceCommand);

            //voiceCommand = null;
        }
```

Μπλοκ Κώδικα A.3: Kinect Code

A.1.4 XML Data

```
<?xml version="1.0"?>
<KinectSkeletonCollection xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
    xmlns:xsd="http://www.w3.org/2001/XMLSchema">
  <Skeletons>
    <KinectSkeleton>
      <ID>69</ID>
      <Joints>
        <KinectJoint>
          <Name>HipCenter</Name>
          <X>480</X>
          <Y>300</Y>
          <Z>0</Z>
        </KinectJoint>
        <KinectJoint>
          <Name>Spine</Name>
          <X>480</X>
          <Y>300</Y>
          <Z>0</Z>
        </KinectJoint>
        <KinectJoint>
          <Name>ShoulderCenter</Name>
          <X>480</X>
          <Y>300</Y>
          <Z>0</Z>
        </KinectJoint>
        <KinectJoint>
          <Name>Head</Name>
          <X>489.53878784179688</X>
          <Y>316.99786376953125</Y>
          <Z>0.31318032741546631</Z>
        </KinectJoint>
        <KinectJoint>
          <Name>ShoulderLeft</Name>
          <X>480</X>
          <Y>300</Y>
          <Z>0</Z>
        </KinectJoint>
        <KinectJoint>
          <Name>ElbowLeft</Name>
          <X>480</X>
          <Y>300</Y>
          <Z>0</Z>
        </KinectJoint>
        <KinectJoint>
          <Name>WristLeft</Name>
          <X>498.16299438476562</X>
          <Y>329.64492797851562</Y>
          <Z>0.31219619512557983</Z>
```

```
        </KinectJoint>
        <KinectJoint>
          <Name>HandLeft</Name>
          <X>487.78619384765625</X>
          <Y>346.44659423828125</Y>
          <Z>0.29760891199111938</Z>
        </KinectJoint>
        <KinectJoint>
          <Name>ShoulderRight</Name>
          <X>480</X>
          <Y>300</Y>
          <Z>0</Z>
        </KinectJoint>
        <KinectJoint>
          <Name>ElbowRight</Name>
          <X>480</X>
          <Y>300</Y>
          <Z>0</Z>
        </KinectJoint>
        <KinectJoint>
          <Name>WristRight</Name>
          <X>490.73251342773438</X>
          <Y>339.24359130859375</Y>
          <Z>0.30077379941940308</Z>
        </KinectJoint>
        <KinectJoint>
          <Name>HandRight</Name>
          <X>494.31002807617188</X>
          <Y>352.32479858398438</Y>
          <Z>0.39247590303421021</Z>
        </KinectJoint>
        <KinectJoint>
          <Name>HipLeft</Name>
          <X>480</X>
          <Y>300</Y>
          <Z>0</Z>
        </KinectJoint>
        <KinectJoint>
          <Name>KneeLeft</Name>
          <X>480</X>
          <Y>300</Y>
          <Z>0</Z>
        </KinectJoint>
        <KinectJoint>
          <Name>AnkleLeft</Name>
          <X>528.742431640625</X>
          <Y>330.83782958984375</Y>
          <Z>0.39768362045288086</Z>
        </KinectJoint>
        <KinectJoint>
          <Name>FootLeft</Name>
          <X>529.49163818359375</X>
          <Y>342.51596069335938</Y>
          <Z>0.49435096979141235</Z>
        </KinectJoint>
        <KinectJoint>
          <Name>HipRight</Name>
          <X>480</X>
          <Y>300</Y>
          <Z>0</Z>
        </KinectJoint>
        <KinectJoint>
          <Name>KneeRight</Name>
```

```
          <X>480</X>
          <Y>300</Y>
          <Z>0</Z>
        </KinectJoint>
        <KinectJoint>
          <Name>AnkleRight</Name>
          <X>500.71566772460938</X>
          <Y>333.47280883789062</Y>
          <Z>0.40671679377555847</Z>
        </KinectJoint>
        <KinectJoint>
          <Name>FootRight</Name>
          <X>528.98175048828125</X>
          <Y>314.6995849609375</Y>
          <Z>0.34773671627044678</Z>
        </KinectJoint>
      </Joints>
    </KinectSkeleton>
  </Skeletons>
  <kinectCursorX>-1,649</kinectCursorX>
  <kinectCursorY>1,195</kinectCursorY>
  <isHandGripped>false</isHandGripped>
</KinectSkeletonCollection>
```

Μπλοκ Κώδικα A.4: XML Data

A.1.5 Unity Client

```
1
2  public class KinectClient : MonoBehaviour
3  {
4      private string clientLogLabel;
5      private string mouseLabel;
6
7      private KinectSkeletonCollection xmlSkeletons;
8
9      private KinectJoint Hip_Center;
10     private KinectJoint Spine;
11     private KinectJoint Shoulder_Center;
12     private KinectJoint Head;
13     private KinectJoint Shoulder_Left;
14     private KinectJoint Elbow_Left;
15     private KinectJoint Wrist_Left;
16     private KinectJoint Hand_Left;
17     private KinectJoint Shoulder_Right;
18     private KinectJoint Elbow_Right;
19     private KinectJoint Wrist_Right;
20     private KinectJoint Hand_Right;
21     private KinectJoint Hip_Left;
22     private KinectJoint Knee_Left;
23     private KinectJoint Ankle_Left;
24     private KinectJoint Foot_Left;
25     private KinectJoint Hip_Right;
26     private KinectJoint Knee_Right;
27     private KinectJoint Ankle_Right;
28     private KinectJoint Foot_Right;
29
30     private GameObject artifact;
31
32     private float speed = 10;
33
34     private Vector3 movement = new Vector3(0, 0, 0);
```

```
35
36      private double distance;
37      private double distance2;
38
39      private Vector2 cursor;
40
41      #region Standard Unity Functions
42      void Start()
43      {
44          xmlSkeletons = new KinectSkeletonCollection { Skeletons = new List<
    KinectSkeleton>() };
45          artifact = GameObject.FindGameObjectWithTag("main_actor");
46
47          Connect();
48      }
49
50      void Update()
51      {
52          byte[] message = Encoding.ASCII.GetBytes("Send Data");
53          client.BeginSend(message, 0, message.Length, SocketFlags.None,
54                           new AsyncCallback(SendData), client);
55      }
56
57      void FixedUpdate()
58      {
59          Ray ray = Camera.main.ScreenPointToRay(new Vector3(Input.mousePosition.x
    , Input.mousePosition.y, 0));
60
61          artifact.transform.position = ray.GetPoint(9);
62
63          movement.z = Mathf.Clamp(artifact.transform.position.z, -9f, 0f);
64          movement.y = Mathf.Clamp(artifact.transform.position.y, -3.5f, 5.5f);
65          movement.x = Mathf.Clamp(artifact.transform.position.x, -7.5f, 7.4f);
66
67          artifact.transform.position = movement;
68      }
69
70      void OnGUI()
71      {
72          #region Joint Labels
73          //GUI.Label(new Rect(15, 15, 600, 20), "Head            X:" + Head.X + "
       Y:" + Head.Y + " Z:" + Head.Z);
74          //GUI.Label(new Rect(15, 35, 600, 20), "Shoulder Center X:" +
    Shoulder_Center.X + "   Y:" + Shoulder_Center.Y + "   Z:" + Shoulder_Center.
    Z);
75          //GUI.Label(new Rect(15, 55, 600, 20), "Shoulder Right  X:" +
    Shoulder_Right.X + "   Y:" + Shoulder_Right.Y + "   Z:" + Shoulder_Right.Z);
76          //GUI.Label(new Rect(15, 75, 600, 20), "Shoulder Left   X:" +
    Shoulder_Left.X + "   Y:" + Shoulder_Left.Y + "   Z:" + Shoulder_Left.Z);
77          //GUI.Label(new Rect(15, 95, 600, 20), "Elbow Right     X:" +
    Elbow_Right.X + "   Y:" + Elbow_Right.Y + "   Z:" + Elbow_Right.Z);
78          //GUI.Label(new Rect(15, 115, 600, 20), "Elbow Left     X:" + Elbow_Left
    .X + "   Y:" + Elbow_Left.Y + "   Z:" + Elbow_Left.Z);
79          //GUI.Label(new Rect(15, 135, 600, 20), "Wrist Right    X:" +
    Wrist_Right.X + "   Y:" + Wrist_Right.Y + "   Z:" + Wrist_Right.Z);
80          //GUI.Label(new Rect(15, 155, 600, 20), "Wrist Left     X:" + Wrist_Left
    .X + "   Y:" + Wrist_Left.Y + "  Z:" + Wrist_Left.Z);
81          //GUI.Label(new Rect(15, 175, 600, 20), "Hand Right     X:" + Hand_Right
    .X + "   Y:" + Hand_Right.Y + "   Z:" + Hand_Right.Z);
82          //GUI.Label(new Rect(15, 195, 600, 20), "Hand Left      X:" + Hand_Left.
    X + "   Y:" + Hand_Left.Y + "   Z:" + Hand_Left.Z);
83          //GUI.Label(new Rect(15, 215, 600, 20), "Spine          X:" + Spine.X +
    "   Y:" + Spine.Y + "   Z:" + Spine.Z);
```

```
84          //GUI.Label(new Rect(15, 235, 600, 20), "Hip Center      X:" + Hip_Center
    .X + "    Y:" + Hip_Center.Y + "    Z:" + Hip_Center.Z);
85          //GUI.Label(new Rect(15, 255, 600, 20), "Hip Right       X:" + Hip_Right.
    X + "    Y:" + Hip_Right.Y + "    Z:" + Hip_Right.Z);
86          //GUI.Label(new Rect(15, 275, 600, 20), "Hip Left        X:" + Hip_Left.X
     + "    Y:" + Hip_Left.Y + "    Z:" + Hip_Left.Z);
87          //GUI.Label(new Rect(15, 295, 600, 20), "Knee Right      X:" + Knee_Right
    .X + "    Y:" + Knee_Right.Y + "    Z:" + Knee_Right.Z);
88          //GUI.Label(new Rect(15, 315, 600, 20), "Knee Left       X:" + Knee_Left.
    X + "    Y:" + Knee_Left.Y + "    Z:" + Knee_Left.Z);
89          //GUI.Label(new Rect(15, 335, 600, 20), "Ankle Right     X:" +
    Ankle_Right.X + "    Y:" + Ankle_Right.Y + "    Z:" + Ankle_Right.Z);
90          //GUI.Label(new Rect(15, 355, 600, 20), "Ankle Left      X:" + Ankle_Left
    .X + "    Y:" + Ankle_Left.Y + "    Z:" + Ankle_Left.Z);
91          //GUI.Label(new Rect(15, 375, 600, 20), "Foot Right      X:" + Foot_Right
    .X + "    Y:" + Foot_Right.Y + "    Z:" + Foot_Right.Z);
92          //GUI.Label(new Rect(15, 395, 600, 20), "Foot Left       X:" + Foot_Left.
    X + "    Y:" + Foot_Left.Y + "    Z:" + Foot_Left.Z);
93          #endregion
94          GUI.Label(new Rect(800, 10, 100, 20), mouseLabel);
95          GUI.Label(new Rect(15, 580, 600, 20), clientLogLabel);
96      }
97      #endregion
98
99
100
101     #region SocketCode
102     private Socket client;
103     private byte[] data = new byte[5120];
104     private int size = 5120;
105
106     private void Connect()
107     {
108         Socket newsock = new Socket(AddressFamily.InterNetwork,
109                                 SocketType.Stream, ProtocolType.Tcp);
110         IPEndPoint iep = new IPEndPoint(IPAddress.Parse("127.0.0.1"), 9050);
111         newsock.BeginConnect(iep, new AsyncCallback(Connected), newsock);
112     }
113
114     private void Connected(IAsyncResult iar)
115     {
116         client = (Socket)iar.AsyncState;
117
118         try
119         {
120             client.EndConnect(iar);
121             //clientLogLabel = "Connected to: " + client.RemoteEndPoint.ToString
    ();
122             clientLogLabel = "Connected to Kinect...";
123             client.BeginReceive(data, 0, size, SocketFlags.None,
124                                 new AsyncCallback(ReceiveData), client);
125         }
126         catch (SocketException)
127         {
128             clientLogLabel = "Error connecting to Kinect!";
129         }
130     }
131
132     private void ReceiveData(IAsyncResult iar)
133     {
134         Socket remote = (Socket)iar.AsyncState;
135         int recv = remote.EndReceive(iar);
136
```

```
137         string stringData = Encoding.ASCII.GetString(data, 0, recv);
138
139         xmlSkeletons = Deserialize<KinectSkeletonCollection>(stringData);
140
141         Head = xmlSkeletons.Skeletons[0].Joints[3];
142         Shoulder_Center = xmlSkeletons.Skeletons[0].Joints[2];
143         Shoulder_Right = xmlSkeletons.Skeletons[0].Joints[8];
144         Shoulder_Left = xmlSkeletons.Skeletons[0].Joints[4];
145         Elbow_Right = xmlSkeletons.Skeletons[0].Joints[9];
146         Elbow_Left = xmlSkeletons.Skeletons[0].Joints[5];
147         Wrist_Right = xmlSkeletons.Skeletons[0].Joints[10];
148         Wrist_Left = xmlSkeletons.Skeletons[0].Joints[6];
149         Hand_Right = xmlSkeletons.Skeletons[0].Joints[11];
150         Hand_Left = xmlSkeletons.Skeletons[0].Joints[7];
151         Spine = xmlSkeletons.Skeletons[0].Joints[1];
152         Hip_Center = xmlSkeletons.Skeletons[0].Joints[0];
153         Hip_Right = xmlSkeletons.Skeletons[0].Joints[16];
154         Hip_Left = xmlSkeletons.Skeletons[0].Joints[12];
155         Knee_Right = xmlSkeletons.Skeletons[0].Joints[17];
156         Knee_Left = xmlSkeletons.Skeletons[0].Joints[13];
157         Ankle_Right = xmlSkeletons.Skeletons[0].Joints[18];
158         Ankle_Left = xmlSkeletons.Skeletons[0].Joints[14];
159         Foot_Right = xmlSkeletons.Skeletons[0].Joints[19];
160         Foot_Left = xmlSkeletons.Skeletons[0].Joints[15];
161
162         cursor.x = (float)Hand_Right.X;
163         cursor.y = (float)Hand_Right.Y;
164     }
165
166     private void SendData(IAsyncResult iar)
167     {
168         Socket remote = (Socket)iar.AsyncState;
169         int sent = remote.EndSend(iar);
170         remote.BeginReceive(data, 0, size, SocketFlags.None,
171                         new AsyncCallback(ReceiveData), remote);
172     }
173     #endregion
174
175     #region Helpers
176     private double GetJointDistance(KinectJoint firstJoint, KinectJoint
    secondJoint)
177     {
178         double distanceX = firstJoint.X - secondJoint.X;
179         double distanceY = firstJoint.Y - secondJoint.Y;
180         double distanceZ = firstJoint.Z - secondJoint.Z;
181         return (double)Math.Sqrt(Math.Pow(distanceX, 2) + Math.Pow(distanceY, 2)
     + Math.Pow(distanceZ, 2));
182     }
183
184     private static T Deserialize<T>(string str)
185     {
186         XmlDocument xdoc = new XmlDocument();
187         try
188         {
189             xdoc.LoadXml(str);
190             XmlNodeReader reader = new XmlNodeReader(xdoc.DocumentElement);
191             XmlSerializer ser = new XmlSerializer(typeof(T));
192             object obj = ser.Deserialize(reader);
193             return (T)obj;
194         }
195         catch
196         {
197             return default(T);
```

```
198             }
199         }
200         #endregion
```

Μπλοκ Κώδικα A.5: Unity Client

Παράρτημα Β

Β.1 Διαφάνειες Παρουσίασης

Ανάπτυξη Εφαρμογής για την Εισαγωγή Δεδομένων από NUI Devices στην Πλατφόρμα Unity

Σφενδύλης Εμμανουήλ (ΑΜ: 1065)

Τεχνολογικό Εκπαιδευτικό Ίδρυμα Κρήτης
Σχολή Τεχνολογικών Εφαρμογών
Τμήμα Μηχανικών Πληροφορικής

State Of The Art

Natural User Interaction (NUI)

- Microsoft Kinect
- Leap Motion
- ThalmicLabs Myo Armband

Game Engines

- Unity
- Unreal Engine 4
- CryENGINE

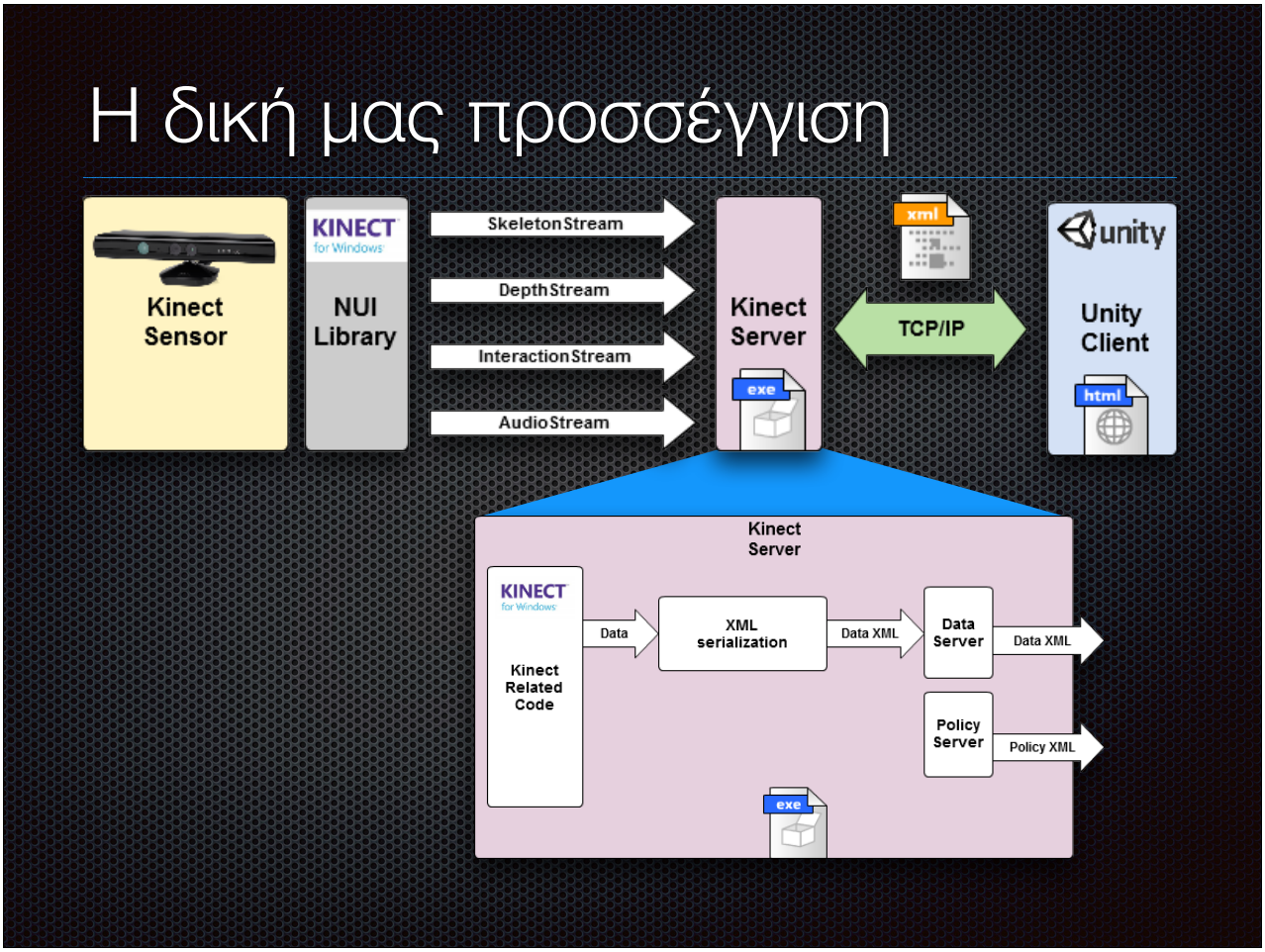
Το πρόβλημα

Για λόγους ασφαλείας η χρήση plugins δεν επιτρέπεται στον web player του Unity.

- Τα plugins είναι βιβλιοθήκες εγγενή κώδικα γραμμένου σε C, C++, Objective-C κλπ.
- Τα plugins επιτρέπουν στον κώδικα ενός παιχνιδιού (γραμμένο σε Javascript, C# ή Boo) να καλεί functions από αυτές τις βιβλιοθήκες.
- Αυτό το χαρακτηριστικό επιτρέπει στις τις βιβλιοθήκες ενδιάμεσου ή υπάρχοντος C/C++ κώδικα του παιχνιδιού να ενσωματώνονται στο Unity.

Οι περιορισμοί ασφαλείας ισχύουν μόνο για τον webplayer, και προς τον editor όταν active build target είναι ο WebPlayer.

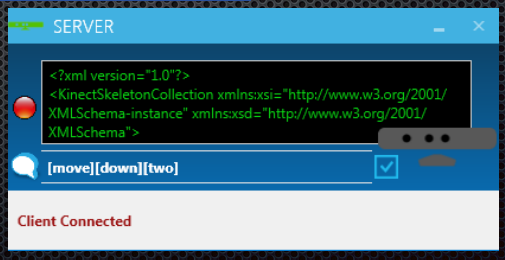
Η δική μας προσσέγγιση



Live Demo

Unity Kinect Client

Kinect Server



Ευχαριστίες

"Στους γονείς μου οι οποίοι με στηρίζουν όλα αυτά τα χρόνια και στον κ.Βιδάκη Νικόλαο ο οποίος επιμελήθηκε αυτής της εργασίας."

Ερωτήσεις;