ΕΘΝΙΚΟ ΜΕΤΣΟΒΙΟ ΠΟΛΥΤΕΧΝΕΙΟ
ΣΧΟΛΗ ΗΛΕΚΤΡΟΛΟΓΩΝ ΜΗΧΑΝΙΚΩΝ ΚΑΙ ΜΗΧΑΝΙΚΩΝ ΥΠΟΛΟΓΙΣΤΩΝ
ΤΟΜΕΑΣ ΤΕΧΝΟΛΟΓΙΑΣ ΠΛΗΡΟΦΟΡΙΚΗΣ ΚΑΙ ΥΠΟΛΟΓΙΣΤΩΝ

Διαπροσωπείες Φυσικής Αλληλεπίδρασης σε Περιβάλλοντα Παιχνιδιών

ΔΙΠΛΩΜΑΤΙΚΗ ΕΡΓΑΣΙΑ

του

ΧΑΤΖΗΤΟΦΗ ΑΝΑΡΓΥΡΟΥ

Επιβλέπων : Στέφανος Κόλλιας
Καθηγητής Ε.Μ.Π.

Αθήνα, Μάρτιος 2012

ΕΘΝΙΚΟ ΜΕΤΣΟΒΙΟ ΠΟΛΥΤΕΧΝΕΙΟ
ΣΧΟΛΗ ΗΛΕΚΤΡΟΛΟΓΩΝ ΜΗΧΑΝΙΚΩΝ
ΚΑΙ ΜΗΧΑΝΙΚΩΝ ΥΠΟΛΟΓΙΣΤΩΝ
ΤΟΜΕΑΣ ΤΕΧΝΟΛΟΓΙΑΣ ΠΛΗΡΟΦΟΡΙΚΗΣ
ΚΑΙ ΥΠΟΛΟΓΙΣΤΩΝ

Διαπροσωπείες Φυσικής Αλληλεπίδρασης σε Περιβάλλοντα Παιχνιδιών

ΔΙΠΛΩΜΑΤΙΚΗ ΕΡΓΑΣΙΑ

του

ΧΑΤΖΗΤΟΦΗ ΑΝΑΡΓΥΡΟΥ

Επιβλέπων : Στέφανος Κόλλιας
Καθηγητής Ε.Μ.Π.

Εγκρίθηκε από την τριμελή εξεταστική επιτροπή την 29η Μαρτίου 2012.

(Υπογραφή)	*(Υπογραφή)*	*(Υπογραφή)*
.....	.....	.....
Στέφανος Κόλλιας Καθηγητής Ε.Μ.Π.	Ανδρέας Σταφυλοπάτης Καθηγητής Ε.Μ.Π.	Γεώργιος Στάμου Λέκτορας Ε.Μ.Π.

Αθήνα, Μάρτιος 2012

(Υπογραφή)

.....

ΧΑΤΖΗΤΟΦΗΣ ΑΝΑΡΓΥΡΟΣ

Διπλωματούχος Ηλεκτρολόγος Μηχανικός και Μηχανικός Υπολογιστών Ε.Μ.Π.

© 2012 – All rights reserved

Περίληψη

Σκοπός της διπλωματικής εργασίας ήταν η ανάπτυξη διαπροσωπειών αλληλεπίδρασης ανθρώπου μηχανής μέσα σε γραφικά περιβάλλοντα παιχνιδιών σε ηλεκτρονικό υπολογιστή και άλλα υπολογιστικά συστήματα. Το λογισμικό αυτό πραγματεύεται τη φυσική αλληλεπίδραση μεταξύ ενός ανθρώπου και ενός ηλεκτρονικού υπολογιστή. Για το σκοπό αυτό χρησιμοποιήθηκε η συσκευή Microsoft Kinect σε συνδυασμό με μία μηχανή απόδοσης γραφικών μέσω XNA που αναπτύχθηκε σε γλώσσα προγραμματισμού C#.

Συγκεκριμένα, πραγματοποιήθηκε μελέτη του SDK του Microsoft Kinect το οποίο έδωσε πρόσβαση στο χειρισμό των 3 φακών που διαθέτει η συσκευή για την αναγνώριση της θέσης και της φύσης διαφόρων αντικειμένων - στη συγκεκριμένη περίπτωση του ανθρωπίνου σώματος - στον τρισδιάστατο χώρο. Κατόπιν, τα δεδομένα που λαμβάνουμε από την επεξεργασία της βιντεοσκόπησης σε πραγματικό χρόνο γίνονται είσοδοι που, έπειτα από μαθηματική επεξεργασία και ενσωμάτωσή τους στη μηχανή φυσικής της εφαρμογής, πραγματοποιούν αλληλεπίδραση μεταξύ του σώματος και του υπολογιστικού συστήματος μέσω ενός ηλεκτρονικού παιχνιδιού.

Η μελέτη και η υλοποίηση αυτή είχε σαν αποτέλεσμα την αποπεράτωση μιας εφαρμογής που δέχεται εντολές εξ αποστάσεως και είναι ένα σύγχρονο παράδειγμα αλληλεπίδρασης ανθρώπου μηχανής με εκπαιδευτικό και ψυχαγωγικό χαρακτήρα. Ο σχεδιασμός και η μεθοδολογία που ακολουθήθηκε μπορεί να γίνει οδηγός για την κατασκευή περισσότερων εφαρμογών που θα έδιναν μία άλλη διάσταση στη χρήση των ηλεκτρονικών μηχανών και θα είχαν εφαρμογή στην εκπαίδευση μέσω σωματικής άσκησης, στη φυσιοθεραπεία, στην χειρουργική, στη ρομποτική. Τέλος, δίνει το έρεισμα για τη σχεδίαση νέων αισθητήρων και συσκευών οπτικής αναγνώρισης με σύγχρονους τρόπους λήψης πληροφορίας.

Λέξεις Κλειδιά: Αλληλεπίδραση ανθρώπου μηχανής, Διαδραστικό Παιχνίδι, Αισθητήρας Kinect, Διαπροσωπείες Φυσικής Αλληλεπίδρασης, Εκπαιδευτικό Λογισμικό

Abstract

The main purpose of the present diploma thesis was the development of interactive interfaces between a human and a machine in graphical game environments for a personal computer or other computing systems. This software refers to the natural interaction between a person and a computer. For the development of the system, the Microsoft Kinect device was used in combination with a graphic engine operating through XNA, developed in programming language C#.

Concretely, SDK of Microsoft Kinect was studied to take advantage of handling the 3 available lenses of the device for the recognition of the place and the nature of various objects - in this case, the human body - in the three-dimensional space. The recorded video is processed in real-time and the received data are used as an input, which after mathematical process and incorporation in the physic engine interacts between the human body and the computer via a video game.

The outcome of this study was the completion of an application, which receives commands remotely and is an example of human-machine interaction with educational and recreational character. The design and the methodology that was followed gives the guidelines for the manufacture of applications, which will give another dimension in the computers' utilization and can be applied in education via physical exercise, in physiotherapy, in surgery, in robotics. Finally, it gives the basis for the design of new sensors and optical recognition devices with modern ways of information acquirement.

Keywords: Human Machine Interaction, Interactive Gaming, Sensor Kinect, Physical Interfaces, Educational Software

Ευχαριστίες

Η παρούσα διπλωματική εργασία εκπονήθηκε στον τομέα Τεχνολογίας Πληροφορικής & Υπολογιστών της Σχολής Ηλεκτρολόγων Μηχανικών και Μηχανικών Υπολογιστών του Ε.Μ.Π, στα πλαίσια των ερευνητικών δραστηριοτήτων γύρω από την αλληλεπίδραση ανθρώπου μηχανής.
Ιδιαίτερες ευχαριστίες οφείλω στον καθηγητή κ. Κόλλια Στέφανο που σε συνεννόηση με τον ερευνητή κ. Καρπούζη Κωνσταντίνο μου ανέθεσαν ένα τόσο ενδιαφέρον θέμα προς ανάλυση και εργασία, καθώς και για την καθοδήγηση, την επίβλεψη και την υποστήριξη με πληροφορίες, συμβουλές και υποδείξεις που μου παρείχαν καθ' όλη τη διάρκεια της εκπόνησης της διπλωματικής εργασίας.
Κλείνοντας, θα ήθελα να ευχαριστήσω ιδιαίτερα την οικογένειά μου, τους φίλους μου και όσους ανθρώπους ήταν κοντά μου όλο αυτό το διάστημα για την υποστήριξή τους. Χωρίς την υπομονή τους και την στήριξή τους η αποπεράτωση του έργου μου θα ήταν αρκετά πιο επώδυνη.

Πίνακας περιεχομένων

1

Εισαγωγή

1.1 Αλληλεπίδραση ανθρώπου-μηχανής

Η αλληλεπίδραση ανθρώπου-μηχανής είναι το επιστημονικό πεδίο που μελετά την αλληλεπίδραση μεταξύ ανθρώπων και υπολογιστικών μηχανών. Θεωρείται ως το σημείο τομής μεταξύ της τεχνολογίας, της πληροφορικής και της ανθρώπινης συμπεριφοράς. Η αλληλεπίδραση μεταξύ χρηστών και υπολογιστών γίνεται στο επίπεδο της διεπαφής χρήστη, μέσω κατάλληλου λογισμικού και υλικού. Από την κατασκευή των πρώτων συσκευών εισόδου (input devices) για υπολογιστές, όπως το ποντίκι και το πληκτρολόγιο, δυστυχώς δεν έχει εξελιχθεί πολύ ο τομέας εισόδου δεδομένων σε υπολογιστικά συστήματα, αφού μόνο τα τελευταία χρόνια έχουν κατασκευαστεί κάποιες καινοτόμες συσκευές, που βέβαια δεν έχουν αντικαταστήσει τις παλαιές κλασσικές μεθόδους. Όπως είναι φυσικό, ο άνθρωπος έχει την τάση να προσπαθεί να προσαρμόσει τις μηχανές στη δική του φύση και κοινωνική συμπεριφορά με αποτέλεσμα να είναι όλο και πιο απαιτητικός από πλευράς εξέλιξης των μεθόδων επικοινωνίας ώστε να νοιώθει την απόλυτη εξοικείωση με αυτές. Γίνονται λοιπόν συνεχείς έρευνες και μελέτες πάνω στην τελειοποίηση ψηφιακών αισθητήρων που να παρέχουν μεγαλύτερη ακρίβεια και αποτελεσματικότητα στην αντίληψη των μηχανών είτε με Ψηφιακή Επεξεργασία Εικόνας, Βίντεο και άλλων Πολυμέσων, είτε με εκπομπή ιδανικών ακτινών για την κατανόηση του φυσικού τρισδιάστατου χώρου, είτε αισθητήρων που μετρούν άλλα φυσικά μεγέθη με αποτέλεσμα τη συνεχή βελτίωση σφαλμάτων αλλά και μεθόδων συλλογής και εισαγωγής δεδομένων. Ένας συνδυασμός λοιπόν πρωτοποριακών αισθητήρων σε συνεργασία με την ανάπτυξη των απαραίτητων λογισμικών με υψηλή τεχνητή νοημοσύνη θα έδινε μια νέα πνοή στα όσα μέχρι τώρα γνωρίζουμε για την αλληλεπίδραση ανθρώπου μηχανής.

1.2 Αντικείμενο διπλωματικής

Στην παρούσα διπλωματική εργασία θα αναπτυχθεί η υλοποίηση ενός συστήματος αλληλεπίδρασης μεταξύ ανθρώπου και υπολογιστή μέσω ενός ηλεκτρονικού παιχνιδιού με τη βοήθεια αισθητήρων προγραμματισμένων να αντιλαμβάνονται κινήσεις στον τρισδιάστατο χώρο. Θα προσαρμόσουμε τη συσκευή Microsoft Kinect ώστε να δίνει θεμιτές πληροφορίες ως δεδομένα για την διάδραση του παιχνιδιού με το χρήστη, μέσω προγραμματιστικών τεχνικών που συγχωνεύουν την πληροφορία του πραγματικού χώρου με ένα δισδιάστατο τεχνητό περιβάλλον στο οποίο λειτουργεί η εφαρμογή του παιχνιδιού.

1.2.1 Συνεισφορά

Η συνεισφορά της διπλωματικής συνοψίζεται ως εξής:

1. Μελετήθηκε σε βάθος η λειτουργία της συσκευής Microsoft Kinect καθώς και το SDK της.
2. Μελετήθηκε το XNA Framework για τη σωστή απόδοση των γραφικών σε γλώσσα προγραμματισμού C#.
3. Υλοποιήθηκαν αλγόριθμοι για την απόδοση και οργάνωση του γραφικού περιβάλλοντος μέσω των βιβλιοθηκών του XNA Framework.
4. Υλοποιήθηκαν αλγόριθμοι για τη μηχανή φυσικής του παιχνιδιού.
5. Ενσωματώθηκαν οι αλγόριθμοι σε πρότυπο σύστημα.

1.3 Οργάνωση κειμένου

Στο Κεφάλαιο 2 πραγματοποιείται μια πρώτη γνωριμία με το βιντεοπαιχνίδι και δίνονται οδηγίες για τη σωστή χρήση του από τους παίκτες. Στο Κεφάλαιο 3 δίνεται το θεωρητικό υπόβαθρο της συσκευής την οποία χρησιμοποιούμε ώστε να γίνει κατανοητός ο τρόπος λειτουργίας της, του XNA Framework για την πλήρη αποσαφήνισή του και πληροφορίες για το τι εστί «μηχανή φυσικής». Ακολούθως, στο Κεφάλαιο 4 αναπτύσσεται η αρχιτεκτονική στην οποία σχεδιάστηκε η εφαρμογή, χωρίζοντάς τη σε τέσσερα κύρια τμήματα των οποίων δίνεται μια σύντομη περιγραφή για τον τρόπο με τον οποίο λειτουργούν μέσα στο παιχνίδι. Έπειτα, στο Κεφάλαιο 5 περιγράφεται η σχεδίαση και η λογική η οποία ακολουθήθηκε για

να γίνει εφικτή με τον καλύτερο δυνατό τρόπο, αλλά και πιο εύστοχο, η υλοποίηση της εφαρμογής. Στο κεφάλαιο αυτό περιγράφεται η κυρίως μελέτη και εργασία της διπλωματικής που εξηγεί της μεθόδους και τις κλάσεις στις οποίες χωρίστηκε το έργο για να αποπερατωθεί πληρέστερα και δομημένα. Κατόπιν, στο Κεφάλαιο 6, περιγράφεται η υλοποίηση της εφαρμογής σύμφωνα με τη σχεδίαση που είχε πραγματοποιηθεί αλλά και περιγραφεί στο κεφάλαιο 5. Μεγάλη έμφαση δίδεται σε σημεία της υλοποίησης τα οποία αποτελούν σημαντικά κομμάτια για την εύρυθμη λειτουργία του παιχνιδιού. Εν κατακλείδι, στο Κεφάλαιο 7 υπάρχει ο επίλογος της διπλωματικής εργασίας με τον οποίο οδηγούμαστε σε κάποια λογικά συμπεράσματα για την αλληλεπίδραση ανθρώπου-μηχανής και κλείνουμε την εργασία. Στο Κεφάλαιο 8 υπάρχει η βιβλιογραφία.

2

Το διαδραστικό παιχνίδι "Prehistoric Nuts"

Κύριος σκοπός και αποτέλεσμα της παρούσας διπλωματικής ήταν η ανάπτυξη ενός λογισμικού-παιχνιδιού το οποίο θα εκτελεί εντολές μέσω αλληλεπίδρασης του ανθρώπου-χρήστη και του ηλεκτρονικού υπολογιστή (ή της κονσόλας Xbox 360). Ο τίτλος του λογισμικού αυτού είναι “Prehistoric Nuts” και πρόκειται για ένα παιχνίδι που ανήκει στην κατηγορία των παιχνιδιών σκόπευσης σε περιβάλλον ύπαρξης φυσικών δυνάμεων. Η ιδιαιτερότητά του έγκειται στη συμβατότητά του με τη συσκευή Microsoft Kinect και αυτό το καθιστά ένα διαδραστικό παιχνίδι.

Εικόνα 2.1: Prehistoric Nuts

Ένα Διαδραστικό παιχνίδι είναι μία εφαρμογή που περιλαμβάνει την ανθρώπινη αλληλεπίδραση μέσω μίας συσκευής εισόδου, με μια διεπαφή χρήστη για να δημιουργήσει οπτική ανάδραση σε μια οθόνη. Η συσκευή εισόδου που χρησιμοποιείται για να χειριστούν οι χρήστες τα παιχνίδια ονομάζεται ελεγκτής παιχνιδιών, και ποικίλλει από πλατφόρμα σε πλατφόρμα. Για παράδειγμα, ένας ελεγκτής μπορεί να συνίσταται μόνο από κουμπιά, ενώ κάποιος άλλος μπορεί να χρησιμοποιεί αισθητήρες ανίχνευσής κίνησης προσφέροντας αλληλεπίδραση με το παιχνίδι μέσω φυσικών κινήσεων του χρήστη.

2.1 Ανίχνευση κίνησης

Ανίχνευση κίνησης είναι μια διαδικασία που επιβεβαιώνει την αλλαγή στη θέση ενός αντικειμένου σε σχέση με το περιβάλλον του ή την αλλαγή στον περιβάλλοντα χώρο σε σχέση με ένα αντικείμενο. Αυτή η ανίχνευση μπορεί να επιτευχθεί και από μηχανικές αλλά και από ηλεκτρονικές (κάμερα, μικρόφωνο, υπέρυθρες) μεθόδους. Η ανίχνευση κίνησης μπορεί είτε να πάρει διακριτές τιμές , δηλαδή εφόσον υπήρχε κίνηση ή όχι (1 η 0), ή μπορεί να αποτελείται από ανίχνευση μεγέθους που μπορεί να μετρήσει και να ποσοτικοποιήσει τη δύναμη ή την ταχύτητα της κίνησης αυτής ή το αντικείμενο που το δημιούργησε. Κίνηση μπορεί να ανιχνευθεί από: ήχο (αισθητήρες ήχου), αδιαφάνεια (οπτικά και υπέρυθρους αισθητήρες και επεξεργαστές εικόνας βίντεο), γεωμαγνητισμό (μαγνητικούς αισθητήρες, μαγνητόμετρα), αντανάκλαση της μεταφερόμενης ενέργειας (υπέρυθρο λέιζερ ραντάρ, αισθητήρες υπερήχων και αισθητήρες ραντάρ μικροκυμάτων), ηλεκτρομαγνητική επαγωγή και κραδασμούς.

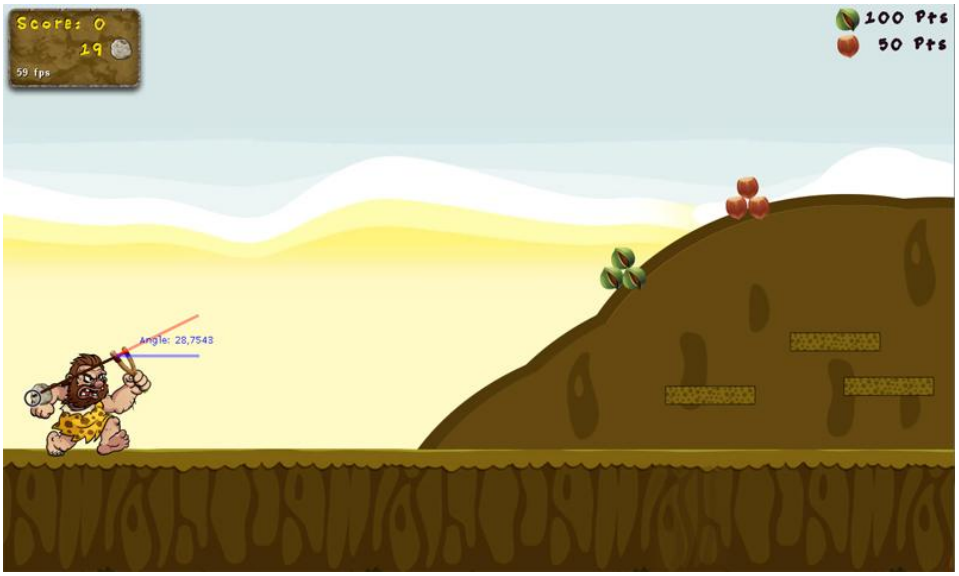
2.2 Γνωριμία

Το Prehistoric Nuts είναι ένα λογισμικό συμβατό με το λειτουργικό σύστημα των Windows 7 και την κονσόλα Xbox 360, το οποίο απαιτεί εγκατάσταση στον εκάστοτε υπολογιστή και σύνδεση του Kinect σε κάποια θύρα USB. Εφόσον οι προϋποθέσεις αυτές πληρούνται, μετά την εκτέλεση της εφαρμογής βρισκόμαστε στο περιβάλλον του παιχνιδιού.

Η παρούσα διπλωματική, από τη στιγμή που αποτελεί ένα εξειδικευμένο λογισμικό και συγκεκριμένα ένα ηλεκτρονικό παιχνίδι με ψυχαγωγικά αλλά και επιμορφωτικά στοιχεία, θεωρήθηκε απολύτως απαραίτητο να πληροί όλες τις απαραίτητες προϋποθέσεις που ορίζει η

κατασκευή ενός ολοκληρωμένου παιχνιδιού. Αυτό έκανε το παιχνίδι να χωριστεί σε διάφορα στάδια που ακολουθούνται μέχρι την ολοκλήρωση ενός πλήρους κύκλου, από τη στιγμή δηλαδή που ξεκινά το παιχνίδι μέχρι και το σημείο όπου ολοκληρώνεται.

Σε πρώτη φάση, προβάλλεται η εικόνα που δίνει της πληροφορίες για το ποιο είναι το λογισμικό, το λογότυπο του και φορτώνονται τα αρχεία και οι μηχανισμοί. Ακολούθως δε θα μπορούσε να λείπει από το παιχνίδι ένα μενού επιλογών για τις επιθυμητές ενέργειες του παίκτη, όπως η επιλογή νέου παιχνιδιού, η επιλογή επιπέδου και η προβολή γενικότερων πληροφοριών της εφαρμογής. Το μεγαλύτερο μέρος της μελέτης και της υλοποίησης προφανώς, αφορά το οπτικό περιβάλλον που προβάλει το κύριο μέρος δράσης του παιχνιδιού στα επίπεδα όπου και λαμβάνουν χώρα όλοι οι μηχανισμοί που υλοποιήθηκαν για την εύρωστη λειτουργία του παιχνιδιού αλληλεπίδρασης.

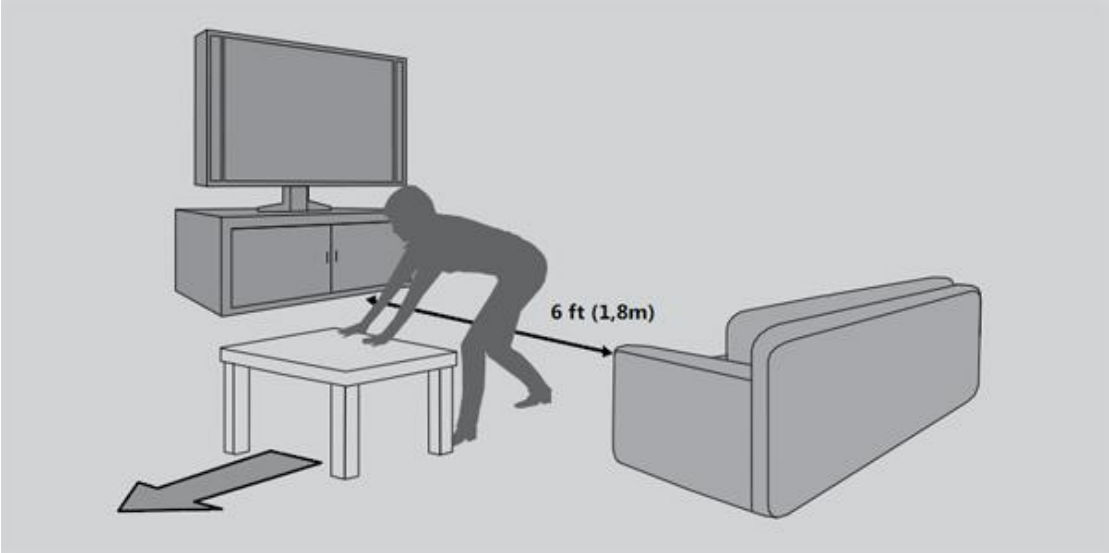


Εικόνα 2.2: Γραφικό Περιβάλλον Παιχνιδιού

2.3 Οδηγίες Χρήσης

Μια σύντομη περιγραφή της προετοιμασίας που χρειάζεται ώστε να λειτουργήσει απολύτως ορθά η αλληλεπίδραση του ανθρώπου με το Kinect και να γίνει σωστά η διάδραση με το ηλεκτρονικό παιχνίδι κρίνεται απαραίτητη.

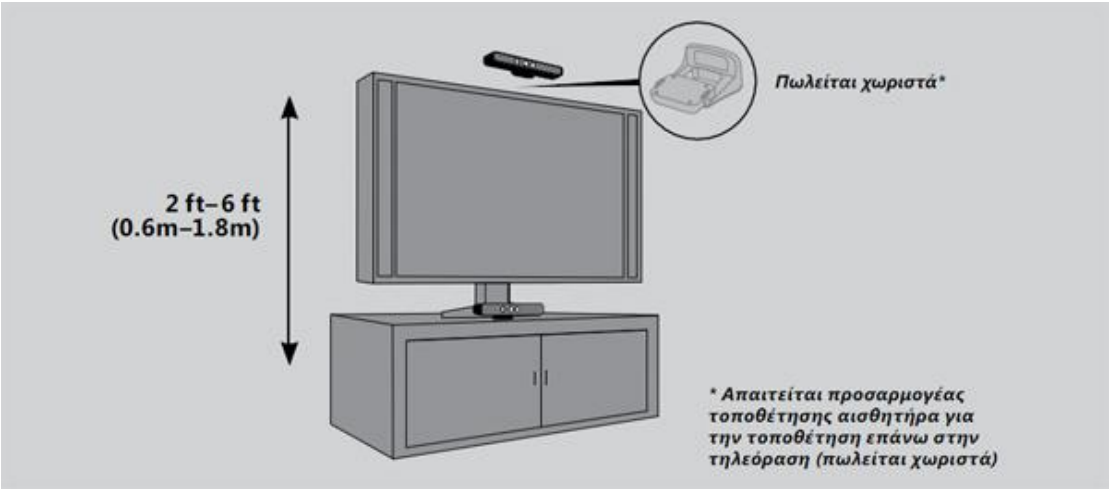
2.3.1 Ορθή χρήση αισθητήρα Microsoft Kinect



Εικόνα 2.2: Απόσταση αλληλεπίδρασης

Ο αισθητήρας Kinect πρέπει να έχει οπτική επαφή με το χρήστη και ο χρήστης χρειάζεται χώρο για να κινείται. Ο αισθητήρας μπορεί να "δει" όταν υπάρχει απόσταση περίπου 2 μέτρων από τον αισθητήρα.

Η περιοχή παιχνιδιού διαφέρει ανάλογα με τη θέση του αισθητήρα και άλλους παράγοντες.



Εικόνα 2.2: Απόσταση αλληλεπίδρασης

Για την καλύτερη απόδοση της περιοχής παιχνιδιού και του αισθητήρα, ο αισθητήρας πρέπει να τοποθετείται σε ύψος από 0,6 έως 1,8 μέτρα, όσο πιο κοντά στο κάτω ή το επάνω μέρος της οθόνης που προβάλει το παιχνίδι. Επίσης:

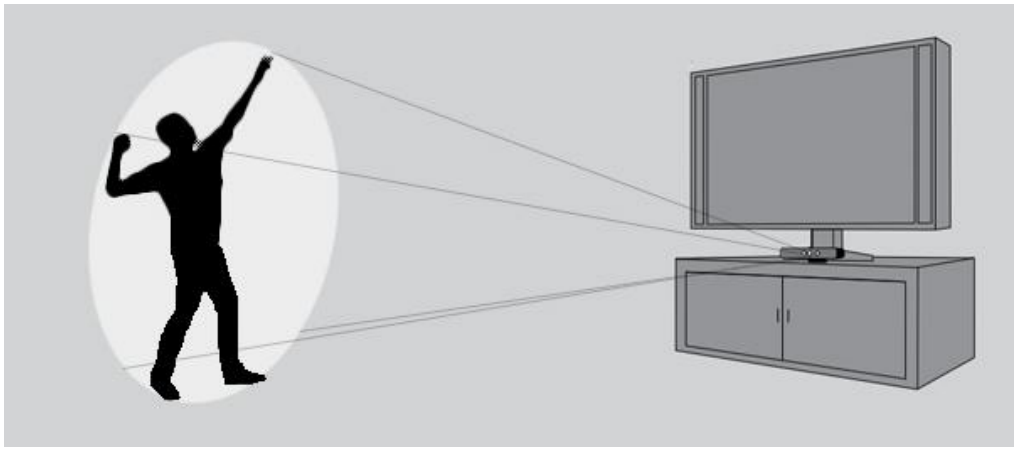
• Η τοποθέτηση του αισθητήρα πρέπει να γίνεται σε σταθερή επιφάνεια.

• Ο αισθητήρας πρέπει να είναι ευθυγραμμισμένος με το κέντρο της οθόνης και να βρίσκεται όσο το δυνατό πιο κοντά στο εμπρός άκρο του σημείο που έχει τοποθετηθεί..

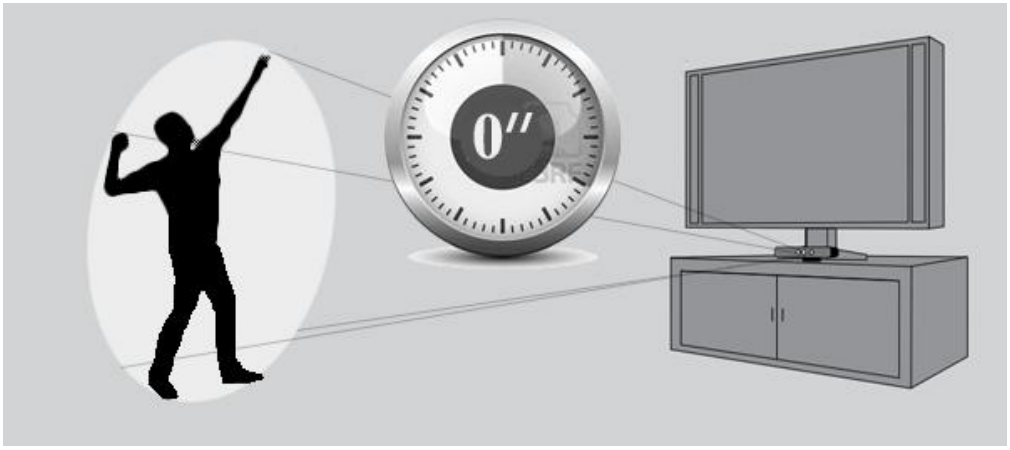
2.3.2 Σκοπός και έλεγχος του παιχνιδιού

Το παιχνίδι βασίζεται στην κεντρική ιδέα ενός προϊστορικού ήρωα ο οποίος συλλέγει ξηρούς καρπούς εκσφενδονίζοντας πέτρες με τη βοήθεια μιας σφεντόνας. Σκοπός λοιπόν του χρήστη είναι να συλλέξει όσο το δυνατόν περισσότερους καρπούς σπαταλώντας όσο το δυνατόν λιγότερες πέτρες. Κάθε καρπός έχει διαφορετική δυσκολία συλλογής αλλά ταυτόχρονα προσδίδει και περισσότερους πόντους στον παίκτη. Είτε μετά την εξάντληση όλων των λίθων είτε μετά τη συλλογή όλων τον καρπών το παιχνίδι κρατά τους συνολικούς πόντους και περνά το χρήστη στο επόμενο επίπεδο. Η εκσφενδόνιση γίνεται με τη βοήθεια μιας ένδειξης της γωνίας στόχευσης αλλά και με ένα δείκτη για το μέτρο της δύναμης με την οποία έχει τεντωθεί η σφεντόνα.

Ο έλεγχος του ήρωα, όπως έχει αναφερθεί ήδη, γίνεται μέσω της χρήσης του Kinect. Πιο συγκεκριμένα, εφόσον ο χρήστης βρίσκεται στο σημείο όπου καλείται να δράσει στο παιχνίδι και έχει πάρει τη σωστή θέση ώστε να εντοπίζεται σωστά από τον αισθητήρα του Kinect, ελέγχει με συγκεκριμένο τρόπο τον ήρωα. Ο τρόπος αυτός αφορά την τοποθέτηση των χεριών του με τέτοιο τρόπο ώστε να προσποιείται ότι έχει τεντώσει μια σφεντόνα με τα δύο του χέρια, κρατώντας τη με το δεξί του χέρι και τεντώνοντάς με το αριστερό. Εφόσον αυτό πραγματοποιηθεί, εισάγεται στην εφαρμογή η γωνία με την οποία σημαδεύει ο παίκτης αλλά ταυτόχρονα και η δύναμη η οποία βάζει στο λάστιχο. Μετά τη σταθεροποίηση των χεριών για ένα διάστημα της τάξης των 3 δευτερολέπτων, η πέτρα φεύγει από τη σφεντόνα και χτυπά τους καρπούς οι οποίοι αρχίζουν και σπάνε μέχρι να συλλεχθούν. Η διαδικασία επαναλαμβάνεται μέχρι την ολοκλήρωση του επιπέδου. Παρακάτω διακρίνουμε στο σχήμα την θέση ελέγχου της σφεντόνας του ήρωα από το χρήστη.



Εικόνα 2.3: Κίνηση στόχευσης



Εικόνα 2.4: Σταθεροποίηση Στόχευσης

Εικόνα 2.5: Εκσφενδονισμός

3

Θεωρητικό υπόβαθρο

Για την καλύτερη κατανόηση των μεθόδων και των εργαλείων που χρησιμοποιήθηκαν για την υλοποίηση της παρούσας διπλωματικής είναι απαραίτητη η περιγραφή της λειτουργίας του Microsoft Kinect, μια σύντομη περιγραφή του XNA Framework και μία αναφορά στις Μηχανές Φυσικής.

3.1 Microsoft Kinect Sensor

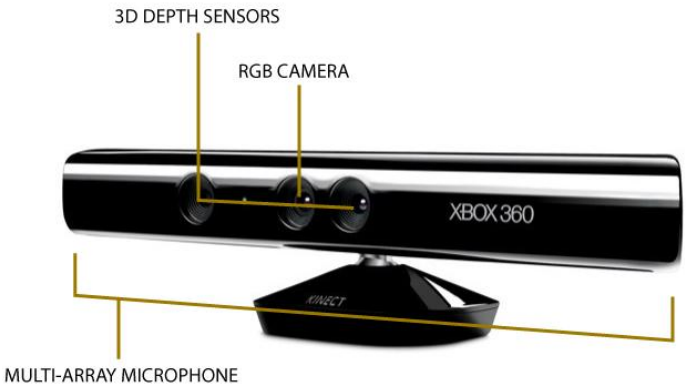
3.1.1 Εισαγωγή

Το Kinect είναι μια συσκευή ανίχνευσης κίνησης της Microsoft για το Xbox 360 και τα Windows PCs. Είναι στην ουσία μια περιφερειακή συσκευή εισόδου που διαθέτει φακούς-κάμερες και ηχητικούς αισθητήρες ώστε να επιτρέπει στους χρήστες να ελέγχουν και να αλληλεπιδρούν εξ αποστάσεως χωρίς να είναι απαραίτητη η επαφή τους με κάποιο άλλο χειριστήριο, ενσύρματο ή ασύρματο. Η αλληλεπίδραση πραγματοποιείται χρησιμοποιώντας χειρονομίες και προφορικές εντολές μέσω του φυσικού περιβάλλοντος του χρήστη.

Εικόνα 3.1: Microsoft Kinect

3.1.2 Τεχνολογία

Το Kinect βασίζεται σε τεχνολογίες λογισμικού που αναπτύσσονται από τη Rare, θυγατρική εταιρεία της Microsoft Game Studios, και την τεχνολογία της ισραηλινής PrimeSens πάνω στις κάμερες και τις συσκευές λήψης βίντεο. Η PrimeSens ανέπτυξε ένα σύστημα που μπορεί να αντιληφθεί χειρονομίες και καθιστά δυνατό τον έλεγχο της συσκευής χρησιμοποιώντας μια υπέρυθρη κάμερα, δύο φακούς και ένα ειδικό μικροτσίπ για την παρακολούθηση της κίνησης των αντικειμένων και των ατόμων σε τρεις διαστάσεις. Αυτό το σύστημα τρισδιάστατης σάρωσης, που ονομάζεται Light Coding, χρησιμοποιεί μια παραλλαγή τρισδιάστατης εικόνας η οποία μπορεί να αναπαρασταθεί σε τρισδιάστατο χώρο.



Εικόνα 3.2: Αισθητήρες του Kinect

Ο αισθητήρας Kinect είναι μια συσκευή που αποτελείται από μια οριζόντια διάταξη που συνδέεται με μια μικρή βάση με μηχανοκίνητο άξονα και έχει σχεδιαστεί για να τοποθετείται κατά μήκος πάνω ή κάτω από την οθόνη που προβάλει το λογισμικό που το χρησιμοποιεί. Η συσκευή διαθέτει έναν RGB φακό, έναν αισθητήρα βάθους που είναι συνδυασμός δύο φακών και ένα πολλαπλό μικρόφωνο τα οποία λειτουργούν με το συγκεκριμένο λογισμικό που περιλαμβάνει το Microsoft Kinect SDK και παρέχουν πλήρη τρισδιάστατη καταγραφή της κίνησης, αναγνώριση προσώπου και δυνατότητες αναγνώρισης φωνής.

Ο αισθητήρας βάθους αποτελείται από έναν υπέρυθρο προβολέα λέιζερ σε συνδυασμό με ένα 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 gameplay και το φυσικό περιβάλλον του παίκτη, ακόμα και αν υπάρχουν στο χώρο και άλλα αντικείμενα.

Το προσωπικό της Microsoft αναφέρει ως κύρια καινοτομία του Kinect την τεχνολογία του λογισμικού του που επιτρέπει την προηγμένη αναγνώριση χειρονομιών, αναγνώριση προσώπων και αναγνώριση φωνής. Σύμφωνα με τις πληροφορίες που παρέχονται, το Kinect είναι σε θέση να εντοπίζει ταυτόχρονα έως έξι άτομα, μεταξύ των οποίων δύο ενεργούς παίκτες των οποίων αναλύει την κίνηση με δυνατότητα αντίληψης 20 αρθρώσεων ανά παίκτη.

Εικόνα 3.3: Τρόπος Λειτουργίας

Ο αισθητήρας Kinect εξάγει βίντεο με ρυθμό καρέ 30 Hz. Η ροή βίντεο χρησιμοποιεί το σύστημα χρωμάτων RGB 8-bit και ανάλυση 640 × 480 pixel με χρωματικό φίλτρο Bayer, ενώ η μονόχρωμη αισθητήρια ροή βίντεο χρησιμοποιεί ανάλυση 640 × 480 pixel με βάθος 11-bit, το οποίο παρέχει 2.048 επίπεδα ευαισθησίας. Ο αισθητήρας Kinect έχει ένα πρακτικό όριο

που κυμαίνεται από 1,2 έως 3,5 μέτρα απόσταση. Η περιοχή που καλύπτει το οπτικό πεδίο του Kinect είναι περίπου 6 τ.μ., αν και ο αισθητήρας μπορεί να διατηρήσει την εστίαση παρακολούθησης σε ένα διευρυμένο φάσμα περίπου 0,7 έως 6 μέτρων. Ο αισθητήρας έχει οπτικό πεδίο 57 ° οριζόντια και 43 ° κατακόρυφα, ενώ ο κινητήριος κεντρικός άξονας είναι σε θέση να κινηθεί σε εύρος γωνίας έως και 27 ° προς τα πάνω ή προς τα κάτω. Το οριζόντιο πεδίο του αισθητήρα Kinect στην ελάχιστη απόσταση θέασης του (περίπου 0,8 μέτρα) είναι περίπου 87 εκατοστά, και το κατακόρυφο περίπου 63 εκατοστά, δηλαδή αντιστοιχούν περίπου 1,3 χιλιοστά ανά ψηφίδα (pixel). Το πολλαπλό μικρόφωνο διαθέτει τέσσερις μικροφωνικές συσκευές και καθεμία λειτουργεί με κανάλι των 16-bit ήχου με ρυθμό δειγματοληψίας 16 kHz.

Επειδή ο αισθητήρας Kinect έχει μηχανοκίνητο μηχανισμό ανάκλησης απαιτεί περισσότερη ενέργεια από αυτή που μπορεί να του παρέχει μια USB θύρα. Για το λόγο αυτό η συσκευή κάνει χρήση ενός ειδικού καλωδίου τροφοδοσίας (περιλαμβάνεται με τον αισθητήρα), το οποίο χωρίζει τη σύνδεση σε ξεχωριστές συνδέσεις USB και ισχύος. Ενέργεια παρέχεται από το ηλεκτρικό δίκτυο μέσω ενός μετασχηματιστή.

3.2 XNA Framework

3.2.1 Εισαγωγή

Το Microsoft XNA είναι ένα σύνολο εργαλείων διαχείρισης περιβάλλοντος πραγματικού χρόνου κατασκευασμένο από τη Microsoft που διευκολύνει την ανάπτυξη και τη διαχείριση ηλεκτρονικών παιχνιδιών. Το XNA επιχειρεί να ελευθερώσει τους δημιουργούς παιχνιδιών από τη γραφή επαναλαμβανόμενου στερεότυπου κώδικα και να βοηθήσει στη δημιουργία νέων πτυχών στην παραγωγή ηλεκτρονικών παιχνιδιών μέσω ενός ενιαίου συστήματος. Το XNA ανακοινώθηκε για πρώτη φορά το Μάρτιο του 2004 στην Καλιφόρνια. Το XNA σήμερα περιλαμβάνει ολόκληρα τμήματα ανάπτυξης της Microsoft Games, όπως το Xbox Development Kit και το XNA Game Studio.

Εικόνα 3.2: Λογότυπο του XNA Framework

3.2.2 Τεχνολογία

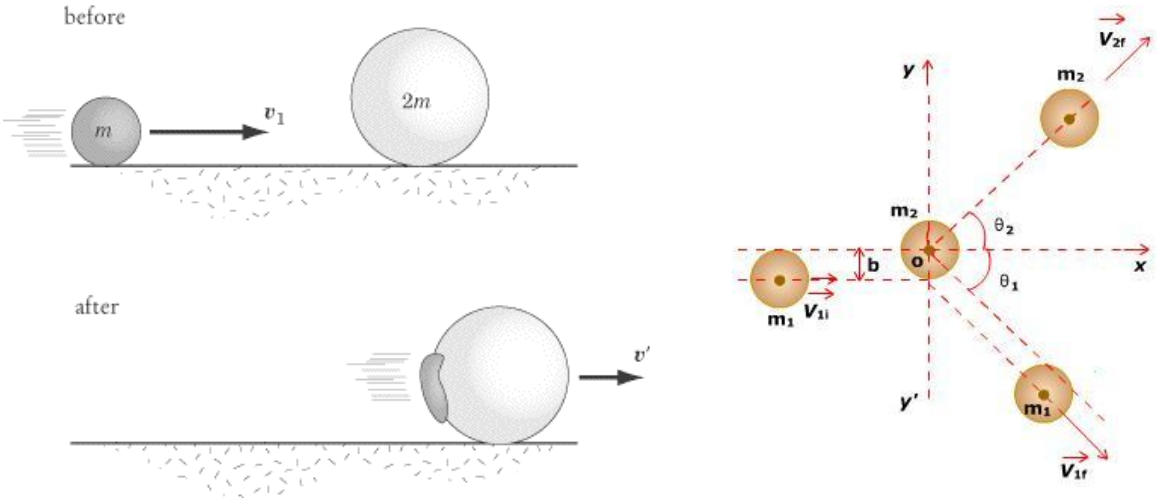
Το XNA Framework είναι βασισμένο στις εγγενείς εφαρμογές .NET Compact Framework 2.0 για ανάπτυξη παιχνιδιών για Xbox 360 και .NET Framework 2.0 για Windows. Περιλαμβάνει ένα εκτεταμένο σύνολο βιβλιοθηκών ειδικών για την ανάπτυξη παιχνιδιών, για να επιτύχει τη μεγιστοποίηση χρήσης κοινού κώδικα σε όλες τις πλατφόρμες. Είναι φιλικό και απλό προς το δημιουργό, πράγμα το οποίο το καθιστά αποδοτικό για να δημιουργεί στα παιχνίδια ένα καλοστημένο περιβάλλον λειτουργίας. Είναι διαθέσιμο για Windows XP, Windows Vista, Windows 7, Windows Phone 7 και Xbox 360. Εφόσον κάποιο παιχνίδι είναι γραμμένο σε XNA, μπορεί να τρέξει σε οποιαδήποτε πλατφόρμα που υποστηρίζει το XNA με ελάχιστη ή καμία τροποποίηση. Παιχνίδια που εκτελούνται στο framework του XNA είναι τεχνικά δυνατό να γραφτούν σε οποιαδήποτε .NET γλώσσα, αλλά μόνο το IDE της C# του XNA Game Studio Express και όλες οι εκδόσεις του Visual Studio 2008 και 2010 το υποστηρίζουν επίσημα.

Το XNA συμπυκνώνει χαμηλού επιπέδου κομμάτια κώδικα που κατασκευάζονται κατά την ανάπτυξη ενός παιχνιδιού, φροντίζοντας για τη συμβατότητα της λειτουργίας του παιχνιδιού στις διάφορες πλατφόρμες, ώστε τα παιχνίδια να μετατρέπονται με ευκολία από μία συμβατή πλατφόρμα σε άλλη. Έτσι επιτρέπει στους προγραμματιστές παιχνιδιών να επικεντρωθούν περισσότερο στο περιεχόμενο του παιχνιδιού και τους μηχανισμούς που κατασκευάζονται αποκλειστικά για το εκάστοτε παιχνίδι. Το XNA έχει ενσωματωμένο ένα μεγάλο αριθμό εργαλείων, όπως το XACT που είναι συμβατό με κάθε είδους πλατφόρμα, ώστε να ενισχύσει τη δημιουργία περιεχομένου στην ανάπτυξη παιχνιδιών. Το XNA υποστηρίζει τόσο τη δημιουργία δισδιάστατου όσο και τρισδιάστατου περιβάλλοντος, που μπορούν να αποτελούν το χώρο δράσης κάποιου παιχνιδιού.

3.3 Μηχανή Φυσικής

3.3.1 Εισαγωγή

Μια μηχανή φυσικής αποτελεί ένα λογισμικό που παρέχει μια προσέγγιση ορισμένων προσομοιώσεων φυσικών συστημάτων, όπως είναι η μηχανική στερεού σώματος (συμπεριλαμβανομένης της ανίχνευσης σύγκρουσης) ή η ροή ρευστών υλικών, που χρησιμοποιούνται στους τομείς των γραφικών ηλεκτρονικών υπολογιστών, των παιχνιδιών και των κινηματογραφικών ταινιών. Οι μηχανές αυτές χρησιμοποιούνται κατά κόρον σε ηλεκτρονικά παιχνίδια, όπου μάλιστα οι προσομοιώσεις συμβαίνουν σε πραγματικό χρόνο. Ο όρος «Μηχανή Φυσικής» χρησιμοποιείται μερικές φορές και γενικότερα για να περιγράψει οποιοδήποτε σύστημα λογισμικού για την προσομοίωση φυσικών φαινομένων.



Εικόνα 3.4: Κανόνες Μηχανής Φυσικής

3.3.2 Ιστορία & Τεχνολογία

Ένας από τους πρώτους ηλεκτρονικούς υπολογιστές γενικής χρήσης, ο ENIAC, χρησιμοποιήθηκε ως μία πολύ απλή μηχανή φυσικής. Χρησιμοποιήθηκε για να σχεδιάσουν βαλλιστικοί πίνακες ώστε να ενισχύσει τις Ηνωμένες Πολιτείες στις στρατιωτικές εκτιμήσεις, για το που θα προσγειωθούν τα βλήματα πυροβολικού διαφορετικών μαζών κατά την εκτόξευσή τους με διαφορετικές γωνίες και μάζες, συνυπολογίζοντας τη μετατόπιση που

προκαλείται από τον άνεμο. Τα αποτελέσματα υπολογίζονταν μόνο μία φορά και συνοψίζονταν σε έντυπους πίνακες που μοιράζονταν στους διοικητές πυροβολικού.

Οι μηχανές Φυσικής έχουν χρησιμοποιηθεί κατά κόρον σε υπερ-υπολογιστές από τη δεκαετία του 1980 για την εκτέλεση υπολογιστικών μοντέλων μηχανικής ρευστών, όπου τα σωματίδια έχουν διανυσματικές δυνάμεις που συνδυάζονται για να προσομοιώσουν την κυκλοφορία. Λόγω των απαιτήσεων της ταχύτητας και υψηλής ακρίβειας, ειδικοί επεξεργαστές υπολογιστών γνωστοί ως διανυσματικοί επεξεργαστές έχουν αναπτυχθεί για την επιτάχυνση των υπολογισμών. Οι τεχνικές αυτές μπορούν να χρησιμοποιηθούν για να μοντελοποιήσουν καιρικές συνθήκες για την πρόβλεψη του καιρού καθώς και την ψύξη επεξεργαστών ηλεκτρονικών υπολογιστών για τη βελτίωση της απορρόφησης θερμότητας. Όπως και με πολλές άλλες διαδικασίες υπολογισμού, η ακρίβεια της προσομοίωσης συνδέεται με την επίλυση της προσομοίωσης και της ακρίβειας των υπολογισμών. Μικρές διακυμάνσεις που δε θα ληφθούν υπόψη στην προσομοίωση μπορεί να αλλάξουν δραματικά τα προβλεπόμενα αποτελέσματα.

Εικόνα 3.4: Προσομοίωση Ροής Ρευστού σε Η/Υ

4

Αρχιτεκτονική Λογισμικού

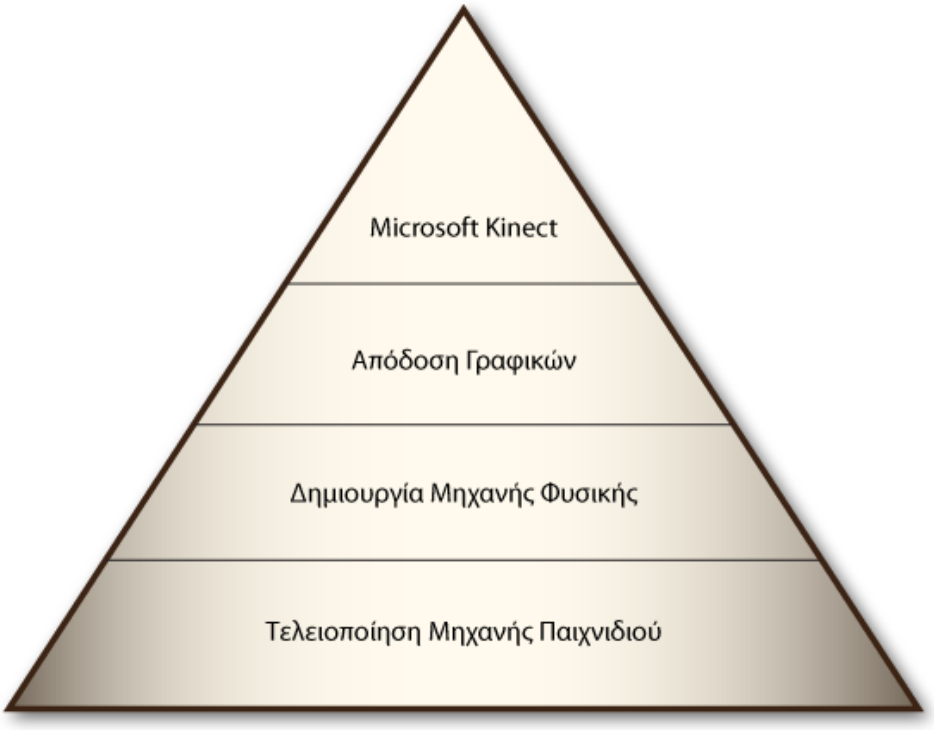
Αναλύοντας το σύστημα οδηγούμαστε σε μια αρχιτεκτονική πάνω στην οποία στηρίζεται και λειτουργεί το λογισμικό μας. Παρακάτω θα πραγματοποιηθεί η περιγραφή αυτής της αρχιτεκτονικής καθώς επίσης και η ανάλυση των επιλογών αλλά και των προϋποθέσεων που πρέπει να πληρούνται για τη σωστή υλοποίηση και λειτουργία.

4.1 Επιλογή πλατφόρμας ανάπτυξης της εφαρμογής

Μία από τις πρώτες και ενδεχομένως πιο κύριες αποφάσεις που έπρεπε να παρθούν σχετικά με την εκπόνηση της παρούσας διπλωματικής εργασίας είναι η επιλογή της πλατφόρμας που χρησιμοποιήθηκε για την ανάπτυξη του λογισμικού.

Για να παρθεί αυτή η απόφαση λήφθηκαν υπόψη πολλές παράμετροι. Πιο συγκεκριμένα, ήταν αναγκαίο να αναλογισθεί κανείς όλες τις ανάγκες που καλείται να καλύψει το λογισμικό και να γραφτεί σε γλώσσα συμβατή με το Microsoft Kinect. Σε πρώτη φάση, διερευνώντας τις επιλογές μας στις πιθανές λύσεις για τη χρήση ειδικών αισθητήρων που καθιστούν εφικτή την αλληλεπίδραση με τρόπο λειτουργικό και ικανό να δώσει το επιθυμητό αποτέλεσμα, έγινε η επιλογή του αισθητήρα της Microsoft, του Kinect. Καθοριστικής επίσης σημασίας ήταν επιλογή του τρόπου κατασκευής μιας μηχανής γραφικών που θα πληρούσε τις προϋποθέσεις που έθετε το περιβάλλον του τελικού έργου. Επιπρόσθετα, για να είμαστε σε θέση να θεωρούμε μια εφαρμογή επιτυχημένη πρέπει να συνοδεύεται από ένα όμορφο, εύχρηστο και λειτουργικό περιβάλλον.

Σχεδιάζοντας μια πυραμίδα που υποδηλώνει την προτεραιότητα των παραμέτρων για τον τρόπο σχεδίασης και υλοποίησης του λογισμικού, με κορυφή την πρωταρχική και κυριότερη παράμετρο έχουμε το εξής σχήμα (Σχήμα 4.1):



Σχήμα 4.1: Πυραμίδα προτεραιότητας περιορισμών

Το SDK του Microsoft Kinect είναι γραμμένο, όπως ήταν άλλωστε αναμενόμενο, σε γλώσσες που ανήκουν στην ευρύτερη οικογένεια των λογισμικών της Microsoft και προφανώς η χρήση και η λειτουργία της συσκευής θα μπορούσαν να γίνουν μόνο από αυτές. Οι γλώσσες αυτές είναι η C++ και η C#, δεδομένο που έδωσε την πρώτη γραμμή πλεύσης για τα εργαλεία που θα χρησιμοποιηθούν.

Έχοντας λάβει λοιπόν τον πρώτο περιορισμό ως προς τις γλώσσες που είναι ικανές να χρησιμοποιηθούν, περνάμε στο επόμενο στάδιο της πυραμίδας που είναι η απόδοση του γραφικού περιβάλλοντος. Λόγω του εύρους της μελέτης που χρειάζεται και των διαφορετικών χαρακτηριστικών που περιέχει η συγκεκριμένη εργασία, θα εξυπηρετούσε στην κατασκευή της μηχανής απόδοσης γραφικών ένα εξειδικευμένο πακέτο βιβλιοθηκών που θα μπορούσε να προσαρμοστεί πάνω σε μία εκ των δύο δυνατών επιλογών (C++, C#) και να εξυπηρετούσε στη γραφή κώδικα εξειδικευμένο για γραφικά περιβάλλοντα. Πραγματοποιώντας λοιπόν μια έρευνα σε τέτοιου είδους πακέτα, επιλέχθηκε το XNA Framework το οποίο δίνει λύσεις ανάλογες των απαιτούμενων. Σε αυτό το σημείο το XNA θέτει τον επόμενο περιορισμό για την επιλογή της γλώσσας που θα εργασθούμε, που δεν είναι άλλη από τη C#. Οπότε βασικό εργαλείο για την υλοποίηση επιλέγεται το «Microsoft Visual Studio 2010» της C#.

Προχωρώντας στο επόμενο επίπεδο, ακολουθεί η δημιουργία μηχανής φυσικής που θα περιέχετε στο ηλεκτρονικό παιχνίδι και θα αλληλεπιδρά με το χρήστη. Το να δημιουργήσουμε κλάσεις κατόπιν σχεδιασμού των δυνατοτήτων που θα έχει η μηχανή φυσικής δεν επιβάλλει κάποιο περιορισμό για τη χρήση συγκεκριμένης γλώσσας προγραμματιστικού εργαλείου, μιας και υπάρχει μια πλειάδα γλωσσών προγραμματισμού ικανές να υλοποιήσουν κάτι τέτοιο. Ωστόσο, η επιλογή έχει γίνει ήδη από τα δύο προηγούμενα στάδια, οπότε θα πραγματοποιηθεί σε γλώσσα C#. Η μηχανή φυσικής Farseer, γραμμένη επίσης σε C#, ήταν το πρότυπο στο οποίο στηρίχθηκε και υλοποιήθηκε η μηχανή.

Τέλος, στη βάση της πυραμίδας βρίσκεται η τελειοποίηση του παιχνιδιού με όλα τα στοιχεία που αποτελούν ένα ολοκληρωμένο ηλεκτρονικό παιχνίδι. Τα στοιχεία αυτά δεν είναι άλλα από την ύπαρξη λογικής στο παιχνίδι, τη σειρά με την οποία προβάλλονται τα εκάστοτε περιβάλλοντα (μενού, πίστες, λογότυπα), η καταμέτρηση πόντων, οι προσπάθειες κ.α. από πλευράς προγραμματισμού, που επίσης γράφτηκαν σε C#, αλλά επίσης και η δημιουργία γραφιστικού υλικού, κινούμενου και μη, που χρησιμοποιήθηκε για την εμφάνιση και τα γραφικά και σχεδιάστηκαν στα προγράμματα «Adobe Illustrator» και «Adobe Photoshop».

4.2 Περιγραφή Λειτουργιών

Το παραπάνω σύνολο σταδίων αποτελεί και το πλήρες πακέτο λειτουργιών που συνθέτει όλη την εφαρμογή, αφού κάθε στάδιο εξυπηρετεί ένα τμήμα της όλης λειτουργίας. Οι βασικές δηλαδή λειτουργίες που αποτελούν το παιχνίδι είναι η λειτουργία του Microsoft Kinect, η λειτουργία της μηχανής γραφικών, η λειτουργία της μηχανής φυσικής και η λειτουργία της μηχανής του παιχνιδιού. Περιγράφοντας λοιπόν αυτές τις τέσσερις λειτουργίες, δίδεται μια πλήρης εικόνα για το τι κάνει κάθε τμήμα και με ποιο τρόπο συνεισφέρει στο σύστημα.

4.2.1 Λειτουργία Microsoft Kinect

Η λειτουργία του Microsoft Kinect αποτελεί την κύρια βάση πάνω στην οποία έχει κτιστεί όλο το σύστημα. Η λειτουργία των αισθητήρων του κάνει εφικτή την είσοδο δεδομένων εξ αποστάσεως και δίνει την πρόσβαση στον έλεγχο του παιχνιδιού, δηλαδή στην πραγματοποίηση της αλληλεπίδρασης ανθρώπου-μηχανής, που είναι και ο κεντρικός άξονας πάνω στον οποίο εργαζόμαστε.

Συγκεκριμένα, στο "Prehistoric Nuts", το Kinect κάνει συγκεκριμένες λειτουργίες τις οποίες επωφελούνται η μηχανή φυσικής και γενικότερα ολόκληρη η μηχανή του παιχνιδιού ώστε να εισάγονται τα δεδομένα καταγράφοντας τις κινήσεις του ανθρωπίνου σώματος.

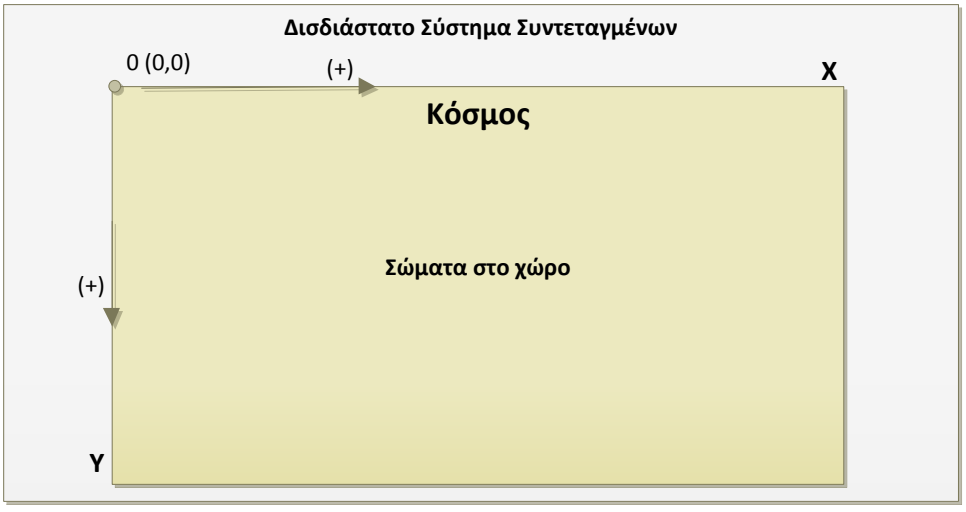
Στην ουσία, καταγράφοντας τις κινήσεις του σώματος, το Kinect αναγνωρίζει 20 αρθρώσεις, τις οποίες παρακολουθεί στον τρισδιάστατο χώρο. Έχοντας ήδη δεδομένα για κάθε άρθρωση ξεχωριστά, αναγνωρίζει τη θέση του αριστερού και του δεξιού χεριού καταγράφοντας τις συντεταγμένες τους με αποτέλεσμα να δίνει χρήσιμη πληροφορία για το πώς ακριβώς ο χρήστης κινεί τα χέρια του, την γωνία που τα τοποθετεί αλλά και την απόσταση.

Συμπερασματικά το Kinect λειτουργεί ως συσκευή εισόδου, αναγνωρίζει σε πραγματικό χρόνο τη συμπεριφορά του παίκτη και τις εντολές που δίνει και τις μεταφέρει στην εφαρμογή. Η λειτουργία του είναι ευνόητη και απλή αλλά παρόλα αυτά, η αναγνώριση και η χρήση των δεδομένων είναι αρκετά πολύπλοκη. Θα αναλυθεί εκτενώς σε επόμενο κεφάλαιο.

4.2.2 Λειτουργία Μηχανής Γραφικών

Η μηχανή γραφικών παρέχει τη δυνατότητα απεικόνισης του γραφικού περιβάλλοντος στην εφαρμογή. Μία πλειάδα μηχανισμών, με τη βοήθεια των βιβλιοθηκών του XNA Framework, προσδίδει τη δυνατότητα προβολής του περιβάλλοντος δράσης του παιχνιδιού.

Στην προκειμένη περίπτωση το παιχνίδι βασίζεται σε μία μηχανή δισδιάστατων γραφικών ψηφίδων (bitmap graphics) που προβάλλεται στην οθόνη σε ένα δισδιάστατο σύστημα συνταγμένων με αρχή των αξόνων την πάνω αριστερή γωνία του παραθύρου, σημείο O(0,0), θετικό ημιάξονα του X προς τα δεξιά και θετικό ημιάξονα του Y προς τα κάτω.



Σχήμα 4.2: Σύστημα Συντεταγμένων μηχανής Γραφικών

Στη μηχανή γραφικών υπάγονται όλοι οι μηχανισμοί που απαιτούνται για να προβληθούν τα γραφικά του παιχνιδιού στη μορφή και την κινησιολογία που είναι θεμιτή. Ορισμένα από τα πιο κύρια στοιχεία που εμπεριέχονται στη μηχανή αυτή είναι η δυνατότητα φόρτωσης εικόνων είτε μεμονωμένων στοιχείων για στατικά γραφικά, είτε πινάκων για κινούμενα. Ταυτόχρονα εμπεριέχονται μηχανισμοί δημιουργίας σχημάτων, όπως κύκλων, τριγώνων, τετραγώνων και γραμμών όταν κρίνεται απαραίτητη η χρήση τους καθώς επίσης και η προβολή κειμένων που μπορεί να δίνουν οδηγίες, πληροφορίες για τους πόντους ή τις επαναλήψεις. Δυνατή επίσης είναι η επιλογή χρωμάτων σε πολλά δυναμικά στοιχεία που προβάλλονται με δυνατότητα ρύθμισης και του Alpha καναλιού που δίνει την πληροφορία για τη διαφάνεια των ψηφίδων.

Η μηχανή γραφικών είναι το κομμάτι που λειτουργεί με τέτοιο τρόπο ώστε να έχουμε την εφαρμογή στο γραφιστικό επίπεδο που θέλουμε. Μέσω της μηχανής αυτής προβάλλεται το κεντρικό παράθυρο μέσα στο λειτουργικό σύστημα των Windows 7, ρυθμίζεται η ανάλυση του παραθύρου, ο ρυθμός ανανέωσης των προβολών (Frames Per Second – FPS), η φόρτωση αρχείων εικόνας ή βίντεο και η προβολή τους, ακόμα και η δυνατότητα σύνδεσης και χρήσης με το Kinect και με την πληροφορία την οποία αυτό καταγράφει.

4.2.3 Λειτουργία Μηχανής Φυσικής

Προέχοντα ρόλο στη λειτουργία του παιχνιδιού κατέχει η μηχανή φυσικής. Όπως περιγράφηκε και σε προηγούμενο κεφάλαιο, με τη χρήση της μηχανής φυσικής στο περιβάλλον της εφαρμογής, εφαρμόσαμε διάφορους φυσικούς νόμους που χρειάστηκαν ώστε να λειτουργεί σωστά η λογική του παιχνιδιού.

Το παιχνίδι ανήκει στην κατηγορία των παιχνιδιών σκόπευσης. Συγκεκριμένα, όπως έχει ήδη αναφερθεί, σκοπός του χρήστη είναι παράγει κρούσεις μεταξύ του λίθου που εκσφενδονίζει και των καρπών που βρίσκονται στο χώρο του επιπέδου του παιχνιδιού. Η μηχανή λοιπόν φυσικής χρησιμοποιεί τους εξής εσωτερικούς μηχανισμούς για να δώσει με ρεαλισμό και αληθοφάνεια την αίσθηση της φυσικής στο χώρο:

- Μηχανισμός βαρύτητας. Τα σώματα που υπάρχουν στο περιβάλλον δέχονται την έλξη της βαρύτητας με απόλυτη προσομοίωση της παγκόσμιας σταθεράς g (9,8 m/s^2) με φορά προς το θετικό ημιάξονα Y, διαθέτουν διαφορετική μάζα το καθένα και σταθεροποιούνται μόνο όταν υπάρχει φυσικό εμπόδιο.

- Μηχανισμός Κρούσης. Η Κρούση είναι φυσικό φαινόμενο και αναφέρεται στην στιγμιαία προσέγγιση δύο σωμάτων. Κρούση ονομάζεται η άσκηση δυνάμεων μεταξύ δύο σωμάτων για πολύ μικρό χρονικό διάστημα. Κατά την κρούση δύο σωμάτων, στην περίπτωση που δεν ασκούνται εξωτερικές δυνάμεις στα σώματα ή αυτές είναι πολύ μικρές σε σχέση με τις εσωτερικές, ισχύει η αρχή διατήρησης της ορμής. Αυτό σημαίνει ότι η συνολική ορμή των σωμάτων πριν και μετά την κρούση παραμένει σταθερή. Ο μηχανισμός λοιπόν αυτός δίνει μια φυσικότητα στην πρόσκρουση των σωμάτων μεταξύ τους με αποτέλεσμα να κινούνται φυσικά στο χώρο και να κατευθύνονται σε λογικές πορείες. Στο μηχανισμό αυτό υπάγεται και η κρούση με αντικείμενα μεγάλης μάζας όπως το δάπεδο ή άλλα σταθερά αντικείμενα που παρά την κρούση παραμένουν ακλόνητα, μιας και υπολογίζονται ως άπειρες μάζες.

- Μηχανισμός Διανυσματικής ταχύτητας. Η εφαρμογή έχει υλοποιηθεί όπως αναφέραμε σε δισδιάστατο σύστημα. Απόρροια του δισδιάστατου αυτού συστήματος είναι να ορίζονται όλα τα διανυσματικά μεγέθη (θέση, ταχύτητα, επιτάχυνση) με δισδιάστατα διανύσματα, δηλαδή με συντεταγμένες X και Y. Καθοριστικό ρόλο στη λειτουργία του όλου μηχανισμού έχει ο μηχανισμούς διανυσματικής ταχύτητας που ορίζει το μέτρο της ταχύτητας του κάθε αντικειμένου ανά πάσα στιγμή αλλά φυσικά και την κατεύθυνσή της. Κεντρικό παράδειγμα στη λειτουργία αυτή είναι ότι μετά τη δημιουργία τάσης στην σφεντόνα και την εκσφενδόνιση, η πέτρα αποκτά μία αρχική ταχύτητα λόγω της δυναμικής ενέργειας που έχει αποκτήσει ο λίθος λόγω της τάσης του παραμορφωμένου ελαστικού που έχει την κατεύθυνση με την οποία σημαδεύει ο χρήστης.

4.2.4 Λειτουργία Μηχανής Παιχνιδιού

Η μηχανή παιχνιδιού (game engine) είναι ο μηχανισμός που λειτουργεί σαν οργανωτής και μοιράζει με τέτοιο τρόπο τις λειτουργίες και τα δεδομένα ώστε η τελική εφαρμογή να έχει λογική, συνοχή, αρχή και τέλος.

Η μηχανή παιχνιδιού είναι το κομμάτι των αλγορίθμων που καταφέρνει να συνδυάσει τους τρεις υπόλοιπους κεντρικούς μηχανισμούς. Στην ουσία είναι ο μεσολαβητής για την επικοινωνία των λοιπόν λειτουργιών, πράγμα που επιτυγχάνεται με συγκεκριμένη σειρά και λογική. Πιο αναλυτικά, τα βήματα έχουν ως εξής:

- Η πληροφορία περνά από τους αισθητήρες του Kinect το οποίο καταγράφει τις κινήσεις του παίκτη.

- Η πληροφορία αυτή μετατρέπεται σε δεδομένα ικανά να εισαχθούν στη μηχανή φυσικής μέσω της μηχανής του παιχνιδιού ώστε το σύστημα να αλληλεπιδράσει.

- Έχοντας πλέον τα αντικείμενα εντός της μηχανής φυσικής λάβει τα νέα δεδομένα, αλληλεπιδρούν μέσα στο παιχνίδι με τρόπο που ορίζουν οι φυσικοί νόμοι του συστήματος.

- Λόγω της πραγματοποίησης αλληλεπίδρασης με το σύστημα, προσπαθώντας αυτό να έρθει σε μία νέα φυσική ισορροπία προσαρμόζει εκ νέου τα δεδομένα των αντικειμένων, τα οποία μέσω της μηχανής του παιχνιδιού φτάνουν στη μηχανή γραφικών η οποία ανανεώνει με τη σειρά της το οπτικό περιβάλλον.

Όπως είναι κατανοητό, έχει το ρόλο του διαχειριστή στο όλο σύστημα. Επιπρόσθετα, η μηχανή του παιχνιδιού δίνει εντολές για κάποιες πρόσθετες λειτουργίες και δυνατότητες του παιχνιδιού όσον αφορά τη λογική του ανάλογα με τα συμβάντα που διαδραματίζονται, για παράδειγμα αναγνωρίζει την επαφή μεταξύ λίθου και καρπού και δημιουργεί ρωγμές έως ότου το συλλέξει ή είναι ο μηχανισμός ο οποίος είναι υπεύθυνος για τη μέτρηση των πόντων, των λίθων του χρόνου και άλλων παραμέτρων.

5

Σχεδίαση Λογισμικού

Στο κεφάλαιο αυτό θα γίνει μια λεπτομερής αναφορά στον τρόπο με τον οποίο σχεδιάστηκε η υλοποίηση της παρούσας διπλωματικής εργασίας, αναλύοντας σε βάθος τις μεθόδους και τις διαδικασίες από τις οποίες αποτελείται το σύστημα και τον τρόπο με τον οποίο συμβάλλουν στην εύρυθμη λειτουργία του. Θα περιγραφούν σύντομα τμήματα της εφαρμογής που τέθηκαν ως αναγκαία κατά τη σχεδίαση του συστήματος, δίνοντας έμφαση με πιο εκτενή αναφορά στις κλάσεις που έχουν πρωτεύοντα ρόλο και αποτελούν αναπόσπαστα κομμάτια της αποτελεσματικής λειτουργίας της εφαρμογής.

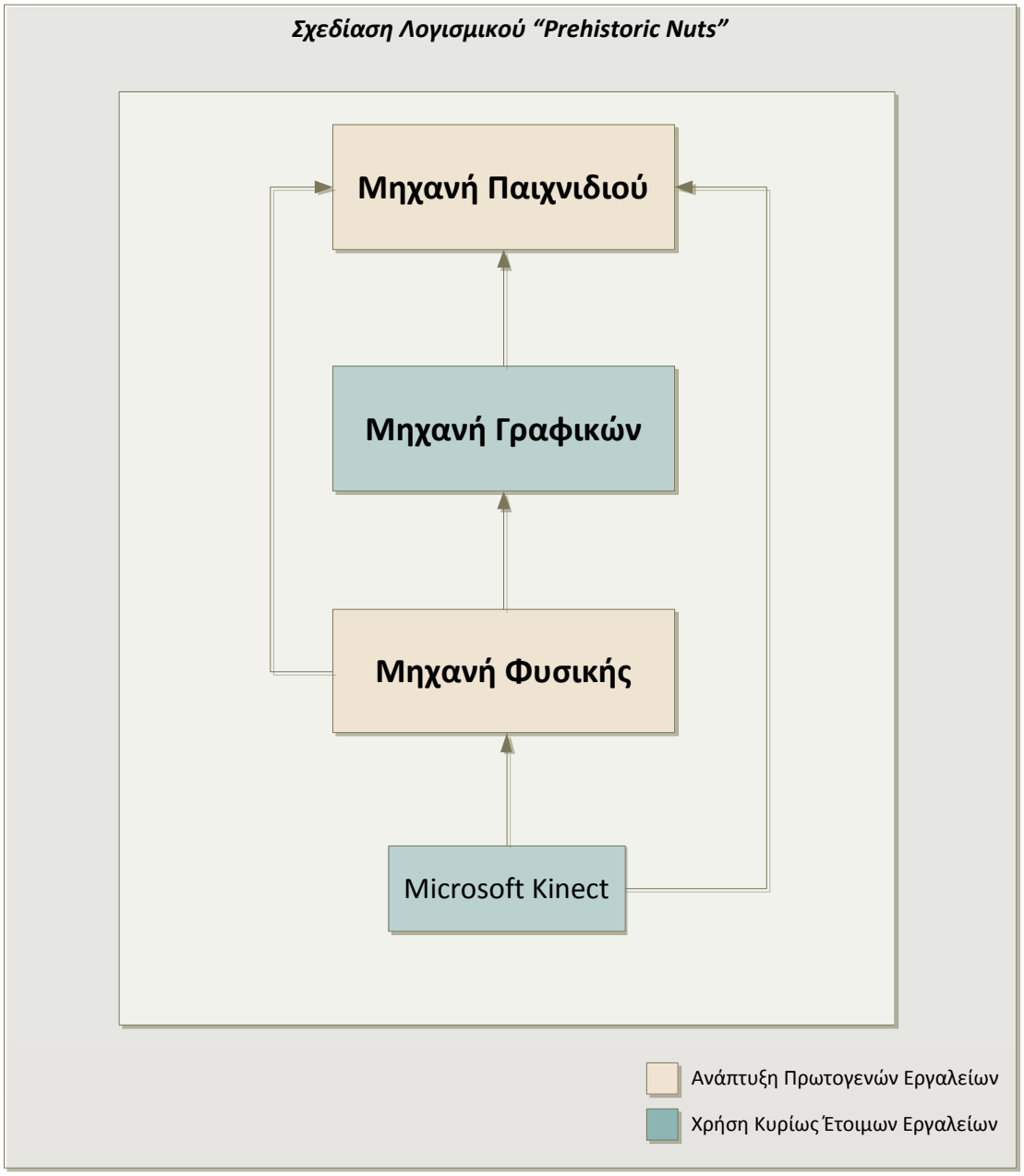
5.1 Αρχιτεκτονική

Οι κλάσεις οι οποίες δημιουργήθηκαν και χρησιμοποιούνται στον κώδικα της εφαρμογής είναι περισσότερες από εκατό, άλλες μικρότερες και άλλες μεγαλύτερες, άλλες λειτουργούν για να τελειοποιήσουν λεπτομέρειες και άλλες αποτελούν τη βάση για τη δημιουργία του συστήματος.

Ωστόσο, μέγιστη σημασία κατά τη σχεδίαση είχε η κατηγοριοποίηση των κλάσεων ως προς τη λειτουργία τους. Άλλες κλάσεις ανήκουν στο κομμάτι του μηχανισμού λήψης δεδομένων μέσω της αλληλεπίδρασης ανθρώπου-μηχανής μέσω του Kinect, άλλες εξυπηρετούν στην δημιουργία των προϋποθέσεων απόδοσης των γραφικών, κάποιες στη δημιουργία βασικών στοιχείων για τη δόμηση της φυσικής αλληλεπίδρασης μεταξύ των αντικειμένων και άλλες έχουν το ρόλο των βασικών κλάσεων διαχείρισης όλων των υπολοίπων ώστε να στηθεί ορθά το ηλεκτρονικό παιχνίδι.

Το παρακάτω σχήμα βοηθά στην κατανόηση των σχέσεων που υπάρχουν μεταξύ των λειτουργικών κατηγοριών που έχουν αναπτυχθεί και συντελούν στην αρμονική λειτουργία του λογισμικού. Όπως είναι εμφανές η γενική μηχανή του παιχνιδιού που βρίσκεται στην

κορυφή του σχήματος είναι ο μηχανισμός που δέχεται τις τελικές πληροφορίες από όλους τους υπόλοιπους μηχανισμούς ώστε να τους χειριστεί κατάλληλα και να συνθέσει το τελικό λογικό, οπτικό και θεμιτό αποτέλεσμα.



Σχήμα 5.1: Σχέσεις Δεδομένων

Παρακάτω, θα περιγραφούν οι σημαντικότερες κλάσεις ανά κατηγορία και θα αποδοθούν σχηματικά ώστε να γίνει κατανοητή η σύνδεση μεταξύ τους, η χρήση αλλά και η κληρονομικότητα μεταξύ τους. Τέλος θα αποδοθούν επίσης ανά κατηγορία για να μην υπάρξει σύγχυση λόγω της πληθώρας των κλάσεων.

5.2 Περιγραφή Κλάσεων

Ακολουθεί η περιγραφή του κυρίως μέρους της προγραμματιστικής σχεδίασης του συστήματος. Θα γίνει μια περιφραστική περιγραφή των μεθόδων που χρησιμοποιούνται σε κάθε κλάση ώστε να αποδοθεί ο τρόπος με τον οποίο λειτουργεί κάθε κομμάτι και συμπληρώνει την γενικότερη λειτουργία.

5.2.1 Κλάσεις & Στοιχεία λειτουργίας του Microsoft Kinect

Το SDK του Microsoft Kinect προσφέρει από πλευράς του στις βιβλιοθήκες του χρήσιμες κλάσεις, δομές, τύπους μεταβλητών και πολλά άλλα για τη διευκόλυνση της χρήσης του. Να σημειωθεί ότι στη συγκεκριμένη εργασία δεν χρησιμοποιήθηκε ο αισθητήρας ήχου του πολλαπλού μικροφώνου που διαθέτει η συσκευή αλλά μόνο οι κάμερες. Κάποια από τα στοιχεία που προσφέρει το SDK και χρησιμοποιούνται στο λογισμικό μας είναι τα εξής:

- Κλάση **Runtime**

  Η κλάση Runtime αποτελεί την πρώτη και κύρια κλάση για την αρχή λειτουργίας των αισθητήρων. Περιέχει συναρτήσεις για την ενεργοποίηση και απενεργοποίηση των αισθητήρων, μεταβλητές και events λειτουργίας.

- Κλάση **RuntimeOptions**

  Η κλάση RuntimeOptions περιέχει όλες τις πιθανές λειτουργίες που δύναται το Kinect να πραγματοποιήσει, των οποίων εντολοδόχο συνάρτηση αποτελεί η Initialize(RuntimeOptions) που βρίσκεται στην κλάση Runtime. Στην ουσία, με το συνδυασμό αυτό πραγματοποιείται η εκκίνηση του αισθητήρα με τη λειτουργία που επιλέγεται ως παράμετρος.

- Κλάση **SkeletonFrame**

  Η κλάση αυτή περιέχει τύπους μεταβλητών οι οποίοι αποθηκεύουν πληροφορίες που έχουν να κάνουν με την αλληλεπίδραση του ανθρώπινου σώματος και των στιγμιότυπων, όπως την πληροφορία για τον ανθρώπινο σκελετό μέσα από τις εικόνες (καρέ) που καταγράφει ο αισθητήρας του Kinect κατά τη λήψη.

- Κλάση **SkeletonTrackingState**

  Η κλάση αυτή περιέχει τις καταστάσεις στις οποίες μπορεί να βρίσκεται η καταγραφή των κινήσεων του ανθρώπινου σκελετού. Όταν βρίσκεται στο στάδιο παρακολούθησης, τότε είναι εφικτό να απομονώσουμε πληροφορίες για τις φυσικές κινήσεις των αρθρώσεων του χρήστη.

- Κλάση **SkeletonData**

  Στην κλάση αυτή ανήκουν τα δεδομένα που περιέχουν όλες τις πληροφορίες για το τι ακριβώς συμβαίνει στο χώρο με το σκελετό και τις αρθρώσεις του. Στην ουσία μέσω της κλάσης αυτής μπορούμε να ελέγξουμε τα δεδομένα που καταγράφει ο αισθητήρας και να εργασθούμε με αυτά.

- Δομή **Joint**

  Σημαντικό ρόλο στην ανίχνευση της κίνησης του σώματος διαδραματίζει η δομή Joint. Η δομή αυτή αποτελείται από έναν απαριθμητή, τον JointID, μια μεταβλητή θέσης και την μεταβλητή που δηλώνει την κατάσταση καταγραφής κίνησης. Η δομή αυτή αντιστοιχεί στον τρόπο χειρισμού κάθε μιας από τις είκοσι αρθρώσεις που ανιχνεύει το Kinect.

- Απαριθμητής **JointID**

  Ο απαριθμητής JointID περιέχει τιμές για τις είκοσι αρθρώσεις του σκελετού, μία προς μία, ώστε να σηματοδοτεί την ταυτοποίηση κάθε άρθρωσης και να ελέγχεται

ξεχωριστά από τις υπόλοιπες. Δεν αποτελεί κάτι άλλο παρά την «ταυτότητα» κάθε άρθρωσης.

Προφανώς οι κλάσεις που προσφέρει το SDK του Kinect είναι γενικές κλάσεις που διευκολύνουν την όσο το δυνατόν μεγαλύτερη και καλύτερη αξιοποίηση του αισθητήρα. Ωστόσο κάθε εφαρμογή χρειάζεται πολλές φορές την προσθήκη κάποιων νέων στοιχείων που θα χρειαστούν στην ανάπτυξή της και δεν είναι προσχεδιασμένα. Στη συγκεκριμένη περίπτωση ήταν αναγκαία η δημιουργία μια νέας κλάσης:

- Κλάση **KinectExtensions**

  Η νέα αυτή κλάση που δημιουργήθηκε έδωσε τις εξής δύο πρόσθετες δυνατότητες. Η μία είχε να κάνει με τη συνεχή καταγραφή των καρέ και την προβολή τους σε βίντεο πραγματικού χρόνου και η δεύτερη με τη θέση των αρθρώσεων σύμφωνα με το σύστημα συντεταγμένων που έχει κατασκευασθεί στο εν λόγω δισδιάστατο παιχνίδι, επίσης σε πραγματικό χρόνο.

5.2.2 Κλάσεις & Στοιχεία της Μηχανής Γραφικών

Για την ανάπτυξη της μηχανής γραφικών που χρησιμοποιήθηκε στο διαδραστικό παιχνίδι Prehistoric Nuts, έχουμε αναφέρει ήδη ότι χρησιμοποιήθηκε το XNA Framework, που προσέφερε άμεσα εργαλεία γραφιστικού χαρακτήρα με τα οποία σχεδιάστηκε το περιβάλλον του παιχνιδιού στις προδιαγραφές που απαιτούνταν. Συγκεκριμένα, έγινε χρήση των Microsoft.Xna.Framework, Microsoft.Xna.Framework.Graphics, Microsoft.Xna.Framework.Input και Microsoft.Xna.Framework.Content. Πιο συγκεκριμένα, κατά κόρον χρησιμοποιήθηκαν τα εξής στοιχεία:

Microsoft.Xna.Framework (Βιβλιοθήκη γενικότερης κατηγορίας εργαλείων παιχνιδιού)

- Κλάση **Game**

  Η κλάση Game είναι η κλάση στην οποία στηρίζεται η βασική μηχανή γραφικών, που έχει άμεση σύνδεση και επικοινωνία με τη μηχανή του παιχνιδιού και με το βασικό κομμάτι της απόδοσης γραφικών (rendering).

- Κλάση **GameTime**

  Η κλάση GameTime αποτελεί το στιγμιότυπο του χρόνου παιχνιδιού εκφράζοντάς το σε μεταβλητή συνεχούς ή διακριτού χρόνου ώστε να μπορεί να χρησιμοποιηθεί κατάλληλα.

- Κλάση **GameComponent**

  Η κλάση GameComponent είναι η κλάση που ουσιαστικά περιλαμβάνει όλα τα στοιχεία του παιχνιδιού που περιέχει το XNA Framework.

- Δομή **Color**

  Η δομή Color εκφράζει την παλέτα των RGB χρωμάτων, του κόκκινου, του πράσινου και του μπλε καναλιού, συμπληρώνοντάς με το Alpha κανάλι για τη ρύθμιση της διαφάνειας.

- Δομή **Point**

  Η δομή Point αποτελεί απλά μια κουκκίδα στον δισδιάστατο χώρο με X και Y συντεταγμένες.

- Δομή **Vector2**

  Η δομή Vector2 αποτελεί απλά ένα διάνυσμα στον δισδιάστατο χώρο με X και Y συντεταγμένες.

- Δομή **Rectangle**

  Η δομή Rectangle ορίζει ένα ορθογώνιο παραλληλεπίπεδο στον δισδιάστατο χώρο με X και Y συντεταγμένες θέσης, πλάτος και ύψος.

Microsoft.Xna.Framework.Graphics (Βιβλιοθήκη εργαλείων γραφιστικού χαρακτήρα)

- Κλάση **GraphicsDevice**

  Η κλάση GraphicsDevice είναι η κλάση που πραγματοποιεί την αρχική βάση απόδοσης γραφικών, δημιουργεί πόρους, χειρίζεται το σύστημα σε επίπεδο μεταβλητών, ρυθμίζει τα επίπεδα φωτεινότητα και δημιουργεί εφέ σκίασης.

- Κλάση **SpriteBatch**

  Η κλάση SpriteBatch είναι η κλάση που δίνει τη δυνατότητα σε μια ομάδα πανομοιότυπων στοιχείων να σχεδιαστούν και να προβληθούν στην οθόνη, σύμφωνα με το πώς έχει οριστεί το όλο σύστημα του γραφικού περιβάλλοντος.

- Κλάση **SpriteFont**

  Η κλάση SpriteFont δίνει τη δυνατότητα προβολής κειμένου στο παράθυρο, στη μορφή και τις ρυθμίσεις που είναι θεμιτό.

- Κλάση **Texture2D**

  Η κλάση Texture2D αντιπροσωπεύει ένα δισδιάστατο πλέγμα από ψηφίδες, στην ουσία δηλαδή, είναι η κλάση η οποία μας δίνει τη δυνατότητα φόρτωσης αρχείων εικόνας για τη χρήση τους μέσα στο παιχνίδι.

- Δομή **Viewport**

  Η δομή Viewport αποτελεί τη δομή με την οποία μας δίνεται η δυνατότητα να ορίσουμε τις διαστάσεις του κεντρικού παραθύρου, στο οποίο προβάλλουμε στη στην παρούσα εφαρμογή το δισδιάστατο περιβάλλον του παιχνιδιού.

Microsoft.Xna.Framework.Input (Βιβλιοθήκη εργαλείων εισόδου δεδομένων)

- Κλάση **Keyboard**

  Η κλάση Keyboard είναι η κλάση με την οποία μας δίνεται η δυνατότητα ανάκτησης των πλήκτρων του πληκτρολογίου.

- Κλάση **Mouse**

  Η κλάση Mouse είναι η κλάση με την οποία μας δίνεται η δυνατότητα ανάκτησης της θέσης και των πλήκτρων του ποντικιού.

- Δομή **KeyboardState**

  Η δομή KeyboardState αντιπροσωπεύει την κατάσταση των πλήκτρων που καταγράφεται από την είσοδο του πληκτρολογίου.

- Δομή **MouseState**

  Η δομή MouseState αντιπροσωπεύει την κατάσταση των πλήκτρων και της θέσης που καταγράφεται από την είσοδο του ποντικιού.

Να σημειωθεί ότι δεν έγινε μεγάλη χρήση των στοιχείων αυτών μιας και η βασική συσκευή εισόδου του παιχνιδιού ήταν η συσκευή Kinect. Παρόλα αυτά, η χρήση του κατά την ανάπτυξη του λογισμικού κρίθηκε απαραίτητη ώστε να γίνονται εύκολα και γρήγορα οι κατάλληλες δοκιμές. Έως και την τελική έκδοση της εφαρμογής, είναι εφικτή η είσοδος από ποντίκι και πληκτρολόγιο ώστε το παιχνίδι να μπορεί να λειτουργήσει ακόμα και χωρίς τη χρήση απαραίτητα του Microsoft Kinect.

Microsoft.Xna.Framework.Content (Βιβλιοθήκη εργαλείων περιεχομένου)

- Κλάση **ContentManager**

  Η κλάση ContentManager είναι το στοιχείο χρονοεκτέλεσης μέσω του οποίου καθίσταται δυνατή η διαχείριση των αντικειμένων που φορτώνουμε μέσω αρχείων. Επίσης, γίνεται η διαχείριση της διάρκεια ζωής των φορτωμένων αντικειμένων. Ουσιαστικά, αποτελεί το διαχειριστή περιεχομένου.

- Κλάση **ContentReader**

  Η κλάση ContentReader είναι το εργαλείο που λειτουργεί ως αναγνώστης περιεχομένου και στην ουσία πραγματοποιεί το μεγαλύτερο μέρος φόρτωσης αρχείων (μέσω του ContentManager.Load). Κάθε στοιχείο περιεχομένου φορτώνεται μέσω της κλάσης αυτής.

5.2.3 Κλάσεις & Στοιχεία της Μηχανής Φυσικής

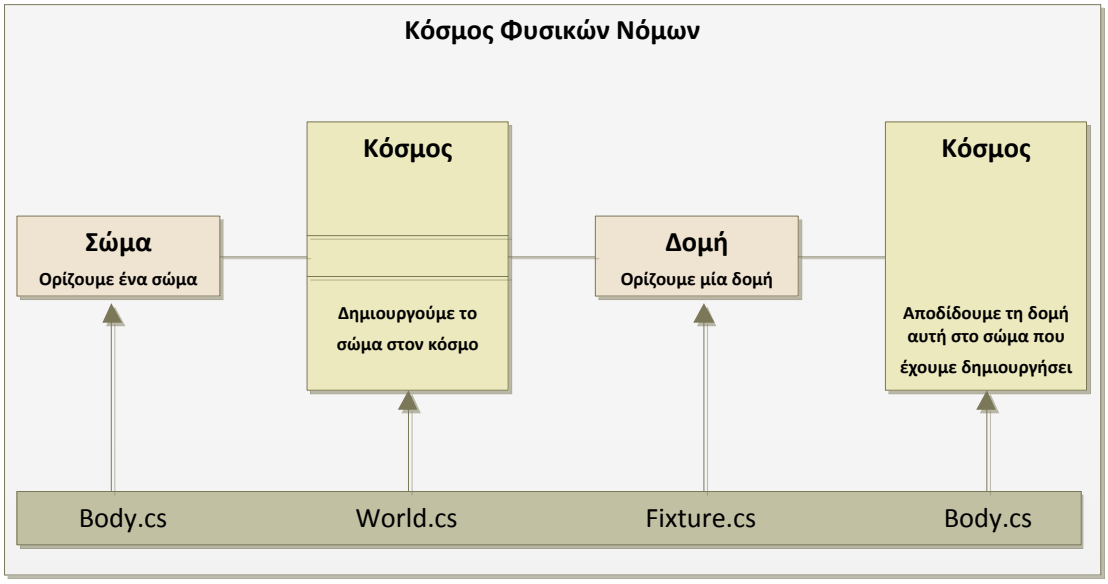
Το σχεδιαστικό κομμάτι που συντίθεται από τις κλάσεις και τα στοιχεία της μηχανής φυσικής είναι το πιο μακροσκελές, σύνθετο και ταυτόχρονα αμιγώς ανεπτυγμένο εκ του μηδενός κατά τη σχεδίαση του λογισμικού. Μετά από μελέτη του σχεδιασμού αντίστοιχων μηχανών, κρίθηκε απαραίτητη η ανάπτυξη κάποιων γενικών οντοτήτων που αποτελούν τη βάση αυτών των μηχανών και θα εξυπηρετούσαν στη σωστή λειτουργία της εφαρμογής γενικότερα.

Η αρχή της σχεδίασης της μηχανής, για το εν λόγω διαδραστικό παιχνίδι, ξεκινά από το σημείο που αποφασίζουμε ποιοι φυσικοί νόμοι πρέπει να διέπουν το χώρο. Προφανώς η βαρύτητα αποτελεί πρωτεύοντα και βασικό νόμο, έπειτα ακολουθούν, η αρχή διατήρησης της ενέργειας, η αρχή διατήρησης της ορμής, η αρχή διατήρησης της στροφορμής και η τριβή μεταξύ των σωμάτων. Οπότε είναι αναγκαία η δημιουργία κάποιων κλάσεων που να είναι υπεύθυνες για ακριβώς αυτές της λειτουργίες όσων σωμάτων επιθυμούμε να εισάγουμε στο περιβάλλον του παιχνιδιού και να αλληλεπιδρούν με αυτό αλλά και μεταξύ τους. Να σημειωθεί ότι προφανώς η προσομοίωση αυτών τον νόμων γίνεται στο δισδιάστατο επίπεδο στο οποίο διαδραματίζεται το παιχνίδι.

Επιπρόσθετα, κρίθηκε απαραίτητο κάθε αντικείμενο μέσα σε αυτό το περιβάλλον που δρουν οι παραπάνω νόμοι να χαρακτηρίζεται από κάποια βασικά χαρακτηριστικά που του δίνουν τις ιδιότητες που προϋποθέτει το σύστημα για να μπορεί να το συμπεριλάβει στους μαθηματικούς υπολογισμούς των φυσικών νόμων.

Όπως και στη φύση, τα χαρακτηριστικά τα οποία ήταν απαραίτητο να υπάρχουν, οπότε και να σχεδιαστούν ως μεταβλητές σε κάθε σώμα-αντικείμενο του περιβάλλοντος, ήταν μεγέθη όπως η μάζα, ο όγκος, η δύναμη, η θέση, η ταχύτητα, η ροπή, η ροπή αδράνειας, μεγέθη τα οποία πρέπει να χαρακτηρίζουν κάθε σώμα ώστε να μπορεί να συμπεριληφθεί στους υπολογισμούς των μηχανισμών του περιβάλλοντος.

Πριν την περιγραφή των κλάσεων που σχεδιάστηκαν θα παραθέσουμε ένα πρώτο σχήμα για να κάνουμε πιο κατανοητή την ιδέα κατά τη σχεδίαση του πρωταρχικού μηχανισμού:



Σχήμα 5.2: Σχέσεις Κλάσεων Αντικειμένου

Ξεκινώντας από αυτό το βασικό διάγραμμα παίρνουμε πληροφορίες για το ποιες πρέπει να είναι οι πρώτες κλάσεις και η βασική δομή της μηχανής φυσικής. Σε πρώτη φάση λοιπόν κρίνεται απαραίτητο να πραγματοποιηθεί μια μικρή συνοπτική περιγραφή αυτής της δομής, καθώς και λίγα λόγια για κάθε στοιχείο που την αποτελεί.

- Κλάση **World** (Κόσμος Φυσικών Νόμων)

  Ένας φυσικός κόσμος είναι ένα σύνθετο σύστημα που αποτελείται από ένα σύνολο σωμάτων, δομών και νόμων που αλληλεπιδρούν μεταξύ τους. Μέσα σε ένα κόσμο που χαρακτηρίζεται από κάποιες φυσικές ιδιότητες, καθετί που θέλει να αλληλεπιδράσει με αυτόν, πρέπει να πληροί κάποιες προϋποθέσεις και να συμβιβάζεται με τους γενικότερους νόμους και κανόνες που ο κόσμος αυτός επιβάλει. Εν προκειμένω δημιουργήθηκε μία κλάση World η οποία ορίζεται από κάποιες πεπερασμένες διαστάσεις στο δισδιάστατο χώρο, ισχύει ο νόμος της βαρύτητας ως προς τις μάζες των σωμάτων (σταθερά g), ισχύουν οι νόμοι δράσης αντίδρασης, οι νόμοι των κρούσεων και γενικότερα όλοι οι νόμοι της κινηματικής στερεών σωμάτων στο μακρόκοσμο. Τέλος, αναπόσπαστο κομμάτι της κλάσης αυτής είναι ύπαρξη και η καταγραφή του χρόνου. Ως εκ τούτου, η κλάση World είναι η πρωταρχική κλάση η οποία φιλοξενεί όλα τα υπόλοιπα στοιχεία και αντιπροσωπεύει το γενικότερο περιβάλλον του συστήματος.

- Κλάση **Body** (Σώμα - Αντικείμενο)

  Η κλάση Body είναι η πιο κύρια κλάση της μηχανής πάνω στην οποία βασίζονται όλα τα μεγέθη και τα στοιχεία τα οποία χαρακτηρίζουν το κάθε αντικείμενο μέσα στο φυσικό περιβάλλον. Στην κλάση αυτή ορίζεται το μεγαλύτερο μέρος των χαρακτηριστικών του σώματος, των φυσικών μεγεθών αλλά των γραφιστικών στοιχείων που αντιστοιχούν σε αυτό. Τα χαρακτηριστικά τα οποία ανήκουν στα φυσικά χαρακτηριστικά του αντικειμένου χωρίζονται στις μεταβλητές οι οποίες αντιστοιχούν στις τιμές των φυσικών του μεγεθών και στις συναρτήσεις οι οποίες ορίζουν φυσικές ιδιότητες. Δηλώνοντας μια μεταβλητή ως Body στην ουσία δημιουργούμε ένα αντικείμενο του οποίου μπορούμε να ορίσουμε τα εξής, ως προς τα φυσικά του μεγέθη και τα φυσικά του χαρακτηριστικά:

  - Τη μάζα
  - Τη ροπή αδράνειας
  - Τη θέση στον δισδιάστατο επίπεδο
  - Την περιστροφή στο δισδιάστατο επίπεδο
  - Την δύναμη που του ασκείται (διανυσματικά)
  - Την ταχύτητα (διανυσματικά)

- Τη ροπή (διανυσματικά)

Ταυτόχρονα, στην κλάση Body που αποτελεί την κλάση κάθε αντικειμένου, ορίζουμε και τα γραφικά που θέλουμε να αντιστοιχούν σε αυτό. Συγκεκριμένα, μετά τη φόρτωση των αρχείων μέσω της βοήθειας του XNA Framework, που έχουμε ήδη αναφέρει, αντιστοιχούμε όποιο φορτωμένο αρχείο ή όποιο μέρος αυτού είναι το κατάλληλο να εμφανίζεται ως εικόνα του, τη δεδομένη στιγμή. Στην πράξη σχεδιάστηκε με τέτοιο τρόπο ώστε να επιλέγονται τα κατάλληλα γραφικά, μέσα από εικόνες με πολλά καρέ, ώστε να εμφανίζεται το κατάλληλο μέρος (καρέ) ανάλογα με τη λειτουργία που συμβαίνει στο παιχνίδι. Οι συναρτήσεις που μας δίνουν τη δυνατότητα να ορίζουμε στο σώμα το οπτικό του στιγμιότυπο είναι:

- **GetTexture**()

  Η GetTexture() δεν παίρνει παράμετρο και μας επιστρέφει την bitmap πληροφορία που έχει τεθεί στο συγκεκριμένο Body.

- **GetCurrentFrame**()

  Η GetCurrentFrame() δεν παίρνει παράμετρο και μας επιστρέφει ένα σημείο δύο συντεταγμένων (Point) που είναι η πληροφορία για τη θέση που βρίσκεται το καρέ που αυτή τη στιγμή προβάλλεται ως το γραφικό του αντικειμένου.

- **GetFrameSize**()

  Η GetFrameSize() δεν παίρνει παράμετρο και μας επιστρέφει ένα σημείο δύο συντεταγμένων (Point) που είναι η πληροφορία για το πόσες ψηφίδες υπάρχουν στο κάθε καρέ, δηλαδή το μέγεθός του. Η πρώτη συντεταγμένη δίνει τον αριθμό που βρίσκονται σε κάθε γραμμή και η δεύτερη τον αριθμό που βρίσκονται σε κάθε στήλη.

- **GetSheetSize**()

  Η GetSheetSize() δεν παίρνει παράμετρο και μας επιστρέφει ένα σημείο δύο συντεταγμένων (Point) που είναι η πληροφορία για το πόσα καρέ υπάρχουν στο αρχείο bitmap. Η πρώτη συντεταγμένη δίνει τον αριθμό που βρίσκονται σε κάθε γραμμή και η δεύτερη τον αριθμό που βρίσκονται σε κάθε στήλη.

- **SetTexture**(Texture2D)

  Η SetTexture() παίρνει σαν παράμετρο την εικόνα που έχουμε φορτώσει σε κάποια μεταβλητή τύπου Texture2D και τη θέτει στο συγκεκριμένο Body.

- **SetCurrentFrame**(Point)

  Η SetCurrentFrame() παίρνει ως παράμετρο ένα σημείο δύο συντεταγμένων (Point) και ορίζει την πληροφορία για τη θέση που βρίσκεται το καρέ που αυτή τη στιγμή προβάλλεται ως γραφικό του αντικειμένου.

- **SetFrameSize**()

  Η SetFrameSize() παίρνει ως παράμετρο ένα σημείο δύο συντεταγμένων (Point) και ορίζει την πληροφορία για το πόσες ψηφίδες υπάρχουν στο κάθε καρέ, δηλαδή το μέγεθός του. Η πρώτη συντεταγμένη δίνει τον αριθμό που βρίσκονται σε κάθε γραμμή και η δεύτερη τον αριθμό που βρίσκονται σε κάθε στήλη.

- **SetSheetSize**()

  Η SetSheetSize() παίρνει ως παράμετρο ένα σημείο δύο συντεταγμένων (Point) και ορίζει την πληροφορία για το πόσα καρέ υπάρχουν στο αρχείο bitmap. Η πρώτη συντεταγμένη δίνει τον αριθμό που βρίσκονται σε κάθε γραμμή και η δεύτερη τον αριθμό που βρίσκονται σε κάθε στήλη.

Όπως γίνεται αντιληπτό, η κλάση Body είναι υπεύθυνη για το κάθε αντικείμενο ώστε να του δίνει τα πλήρη του χαρακτηριστικά στο χώρο ως προς τη συμπεριφορά του στο περιβάλλον αλλά και ως προς την εμφάνισή του μέσα σε αυτό. Το μόνο κύριο χαρακτηριστικό που μένει για να ολοκληρώσει την οντότητα του σώματος στο χώρο είναι η δομή του αντικειμένου, δηλαδή τα γεωμετρικά του χαρακτηριστικά όπως το μέγεθος (όγκος), το σχήμα και η τριβή ως προς τα άλλα αντικείμενα. Αυτά λοιπόν έρχεται να τα συμπληρώσει η κλάση Fixture, η οποία περιγράφεται συνοπτικά παρακάτω.

- Κλάση **Fixture** (Δομή - Γεωμετρία) & **FixtureFactory**

Η κλάση Fixture, όπως αναφέρθηκε, είναι υπεύθυνη για τα γεωμετρικά χαρακτηριστικά του αντικειμένου όπως το μέγεθος (όγκος), το σχήμα και η τριβή. Ειδικότερα, μέσω της κλάσης Fixture και των συναρτήσεων που υπάρχουν σε αυτή, δίνεται η δυνατότητα να αντιστοιχήσουμε μια γεωμετρική δομή σε ένα σώμα ώστε να έχει συγκεκριμένα χαρακτηριστικά ως προς τη συμπεριφορά του ως δομή. Σε αυτό βοηθά και η κλάση FixtureFactory που παράγει το Fixture για το κάθε body. Δηλαδή μέσω της Fixture, δίνουμε το σχήμα, τις διαστάσεις και τις τριβές του σώματος ώστε να είναι εφικτή η αλληλεπίδραση του με τα υπόλοιπα σώματα στο χώρο (άλλα αντικείμενα, δάπεδο κτλ). Συνοπτικά θα παρατεθούν κάποιες από τις συναρτήσεις της κλάσης και η λειτουργία τους:

- **Fixture**(Body, Shape)

  Καλώντας τη συνάρτηση αυτή της Fixture, με παραμέτρους ένα σώμα Body και ένα σχήμα Shape αποδίδουμε στο σώμα το σχήμα και του δίνουμε πλέον τα χαρακτηριστικά μιας δομής ώστε να μπορεί να αλληλεπιδρά με αυτά τα δεδομένα στο χώρο.

- **Friction**

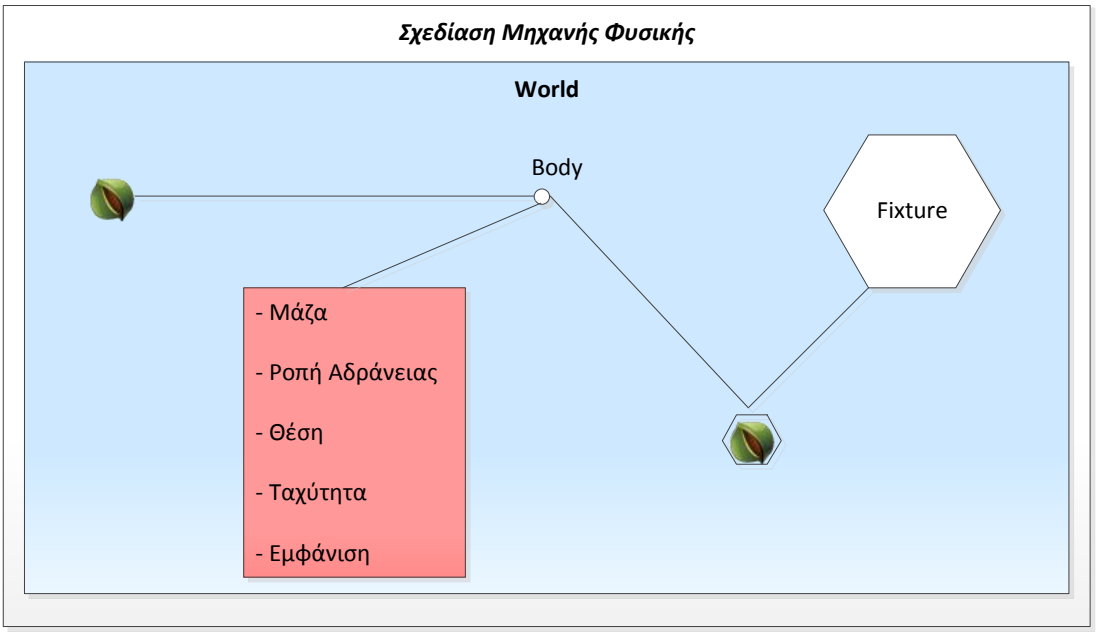
  Μέσω της Friction ορίζουμε την τριβή που θα έχει το σώμα καθώς εφάπτεται σε άλλα αντικείμενα με αποτέλεσμα να κόβει ταχύτητα σύμφωνα με το νόμο της τριβής.

- **RegisterFixture**()

  Μέσω της RegisterFixture γίνεται η «εγγραφή» της δομής ώστε να υπάρξει ως οντότητα στο περιβάλλον του παιχνιδιού.

- **Destroy**()

  Μέσω της Destroy γίνεται η «διαγραφή» της δομής ώστε να μην υπάρχει ως οντότητα στο περιβάλλον του παιχνιδιού πλέον δηλαδή είναι αντίθετη της RegisterFixture.



Σχήμα 5.3: Σχέση Body και Fixture

Περισσότερες λεπτομέρειες για τον τρόπο λειτουργίας αυτών των κλάσεων και των μηχανισμών θα δοθούν στο επόμενο κεφάλαιο που θα μιλήσουμε για την υλοποίηση.

5.2.4 Κλάσεις & Στοιχεία της Μηχανής Παιχνιδιού

Το τέταρτο σχεδιαστικό κομμάτι, που συντίθεται από τις κλάσεις της μηχανής παιχνιδιού, περιέχει τη σειρά με την οποία προβάλλονται στο χρήστη τα διάφορα μέρη-στάδια που απαρτίζουν την όλη εφαρμογή και αποτελούν τις κλάσεις που εκμεταλλεύονται και χρησιμοποιούν όλους τους μηχανισμούς που έχουν αναπτυχθεί ήδη κατά τη σχεδίαση του λογισμικού για την λειτουργία τους. Πιο συγκεκριμένα, στο τμήμα αυτό της σχεδίασης αναπτύχθηκαν οι κεντρικές κλάσεις που κάνουν τη διαχείριση των υπολοίπων ώστε η εφαρμογή να έχει λογική σειρά. Το παιχνίδι διαθέτει μία κεντρική κλάση, την PrehistoricNuts.cs μέσω της οποίας ξεκινά η εφαρμογή, ορίζεται το μέγεθος του παραθύρου και προβάλει το κεντρικό μενού. Παρακάτω ακολουθεί περιγραφή της γενικότερης σχεδίασης του μηχανισμού αυτού:

- Κεντρική κλάση **PrehistoricNuts**

  Η κλάση PrehistoricNuts αποτελεί την κύρια κλάση (τη main class) και όπως αναφέρθηκε καλεί κάποιες πρωταρχικές εντολές που αφορούν την εκκίνηση της εφαρμογής, και άλλες διαδικασίες κατά την εισαγωγή στο παιχνίδι όπως την προβολή του λογοτύπου και την εισαγωγή στο μενού. Στο στάδιο που προβάλλεται το μενού, η κλάση αυτή δίνει τη δυνατότητα επιλογής των επιπέδων του παιχνιδιού με τη βοήθεια μιας ακόμα σημαντικής κλάσης, της ScreenManagerComponent. Με τη βοήθεια αυτής και κάποιων δευτερευόντων μηχανισμών κάνει τη διαχείριση για την προβολή του επιλεγμένου οπτικού περιβάλλοντος κάθε φορά.

- Κλάση **ScreenManagerComponent**

  Η κλάση αυτή όπως δηλώνει και η ονομασία της, βοηθά στη διαχείριση του τι θέλουμε να προβάλλουμε στο χρήστη ανά πάσα στιγμή ώστε να είναι σύμφωνο με τις επιλογές του αλλά και με το στάδιο της εφαρμογής στο οποίο βρίσκεται. Έτσι μέσω της ScreenManagerComponent φορτώνουμε τα διάφορα επίπεδα προβολών που επιθυμούμε, εφόσον τα έχουμε δημιουργήσει ως οντότητες, και τα διαχειριζόμαστε ώστε να εμφανίζονται με τον τρόπο που επιθυμούμε.

- Κλάση **MenuScreen**

  Ένα παράδειγμα τέτοιας οντότητας είναι η κλάση MenuScreen. Είναι η κλάση που είναι υπεύθυνη για την προβολή του μενού επιλογών του παιχνιδιού, καλείται ως πρώτη οθόνη προβολής μετά την εμφάνιση του λογοτύπου και είναι χαρακτηριστικό παράδειγμα διαχειριζόμενου στοιχείου μέσω της ScreenManagerComponent. Στην κλάση αυτή δημιουργούμε τη γενικότερη σχεδίαση για τον τρόπο εμφάνισης των επιλογών και πραγματοποιείται η γενική διεπαφή με τον παίκτη ώστε να επιλέξει την επόμενο περιβάλλον στο οποίο επιθυμεί να βρεθεί. Έχει σχεδιαστεί με δυναμικό τρόπο για την εύκολη προσθαφαίρεση νέων επιπέδων προβολών.

- Κλάση **GameScreen & PhyshicsGameScreen**

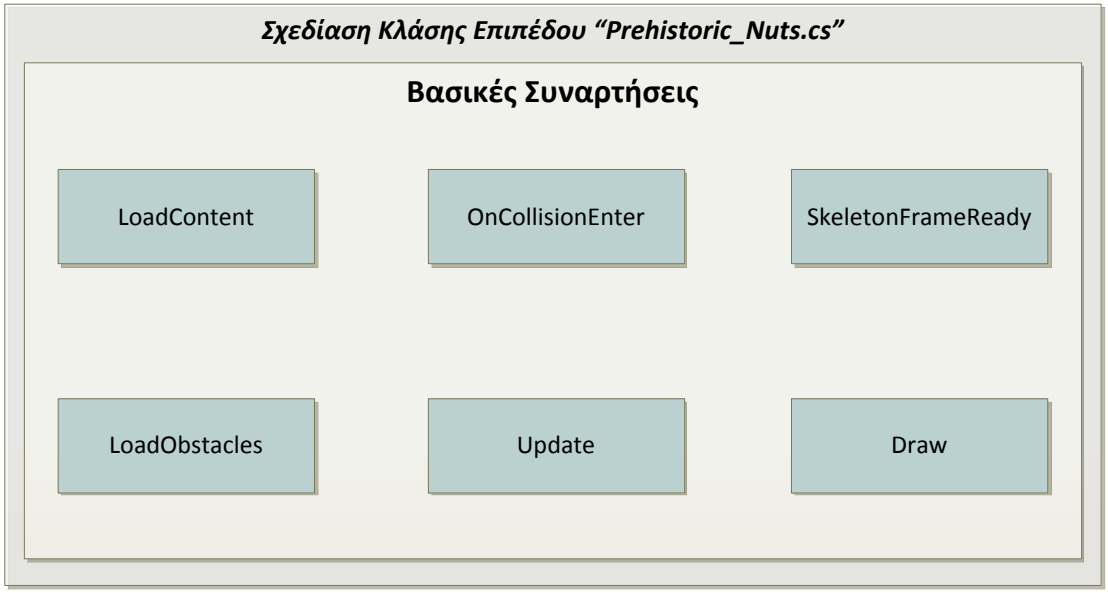
Η κλάση GameScreen είναι η γενικότερη κλάση που προσδίδει τα οπτικά χαρακτηριστικά που επιθυμούμε μέσα στο περιβάλλον που διαδραματίζεται το παιχνίδι. Μία GameScreen οντότητα έχει ιδιότητες που εξυπηρετούν στο να δίνεται η δυνατότητα προσαρμογής του περιβάλλοντος όπως ακριβώς το επιθυμούμε βάσει σχεδιασμού, δηλαδή μας δίνει την δυνατότητα να ορίσουμε αν στο επίπεδο αυτό που εμφανίζεται βασισμένο σε αυτή την οντότητα είναι εμφανής ο κέρσορας, η κατάσταση στην οποία βρίσκεται το συγκεκριμένο οπτικό επίπεδο, για παράδειγμα αν είναι ενεργό ή σε στάδιο αναμονής, αν είναι εστιασμένο το σύστημα σε αυτό, αν βρίσκεται σε φάση εξόδου ή αν είναι ενεργοποιημένο ως προς τη διάδραση μέσω των συσκευών εισόδου. Γενικότερα είναι μία κλάση που έχει γενικά χαρακτηριστικά που κρίνονται απαραίτητα για την ορθή σχεδίαση των επιπέδων του παιχνιδιού ώστε να εξυπηρετούν πλήρως στην ορθή αλληλεπίδραση μεταξύ του παίκτη και του περιβάλλοντος.

Η κλάση PhyshicsGameScreen είναι μια πιο εξειδικευμένη κλάση απόδοσης οπτικών επιπέδων που κληρονομεί την κλάση GameScreen. Ταυτόχρονα, προσθέτει στοιχεία σχετικά με την επίδραση της μηχανής φυσικής και τη διαχείριση των μηχανισμών της μέσα στο περιβάλλον αυτό. Με τη διαχείριση των αντικειμένων μέσω των συσκευών εισόδου, επιτρέπεται καλύτερος έλεγχος της λειτουργίας της εφαρμογής αλλά και άμεση πρόσβαση σε επιμορφωτικές πληροφορίες ως προς τη φυσική των αντικειμένων. Συγκεκριμένα, μέσω κάποιων πλήκτρων του πληκτρολογίου εμφανίζονται πληροφορίες επιμορφωτικού χαρακτήρα για ορισμένα από τα φυσικά φαινόμενα και τα φυσικά χαρακτηριστικά του περιβάλλοντος, όπως οι επαφές μεταξύ αντικειμένων, τα σχήματα των αντικειμένων, τα σημεία των πολυγώνων και άλλες πληροφορίες για το φυσικό σύστημα. Περισσότερες πληροφορίες θα δοθούν στο επόμενο κεφάλαιο κατά την περιγραφή της υλοποίησης.

- Κλάσεις **Prehistoric_Nuts1, Prehistoric_Nuts1, Prehistoric_Nuts2**

Στο τελικό στάδιο, αποδέκτης και χρήστης όλων αυτών των μηχανισμών γίνεται η κλάση Prehistoric_Nuts1 και οι παρόμοιου τύπου κλάσεις (Prehistoric_Nuts1, Prehistoric_Nuts2) που αποτελούν τα επίπεδα (levels) του παιχνιδιού. Λόγω του ότι είναι υπεύθυνα για το κύριο μέρος της διάδρασης του παίκτη με την εφαρμογή, όπου πραγματοποιείται και η αλληλεπίδραση ανθρώπου μηχανής, δηλαδή ο βασικός άξονας πάνω στον οποίο σχεδιάστηκε όλο το σύστημα, θα ακολουθήσει μια αναλυτική περιγραφή ώστε να γίνει πλήρως κατανοητή η λειτουργία των κλάσεων αυτών. Σε πρώτη φάση θα παρατεθεί ένα σχήμα ώστε να δώσει τη βασική δομή των

συναρτήσεων που αποτελούν την κλάση και κατόπιν θα περιγραφούν οι λειτουργίες τους.



Σχήμα 5.3: Βασικές Συναρτήσεις Κλάσης Επιπέδου

Η χρήση αυτών των έξι συναρτήσεων σε συνδυασμό με τις απαραίτητες μεταβλητές που χρησιμοποιούνται στην κλάση δίνουν το τελικό αποτέλεσμα για την εκτέλεση του παιχνιδιού σύμφωνα με τις προϋποθέσεις που έπρεπε να τηρούνται κατά τον αρχικό σχεδιασμό. Ακολουθεί περιγραφή των λειτουργιών κάθε συνάρτησης.

- Συνάρτηση **LoadContent**()

  Η συνάρτηση LoadContent δεν παίρνει ως όρισμα κάποια παράμετρο και δεν επιστρέφει κάποια τιμή. Η συνάρτηση αυτή είναι υπεύθυνη για τη φόρτωση των απαραίτητων δεδομένων περιεχομένου σε κάποιες μεταβλητές καθώς και για την αρχικοποίηση όλων των μεταβλητών γενικότερα που χρειάζονται κατά την εκτέλεση των αλγορίθμων. Εφόσον δηλαδή τα αρχεία που θέλουμε να φορτώσουμε δηλωθούν στο πρόγραμμα τότε μέσω της LoadContent γίνεται η φόρτωσή τους μέσα στον αλγόριθμο. Επιπρόσθετα, στη συνάρτηση αυτή δίνεται και η εντολή για την εκκίνηση της λειτουργίας του Microsoft Kinect.

- Συνάρτηση **Update**(GameTime, bool otherScreenHasFocus, bool coveredByOtherScreen)

  Η συνάρτηση Update παίρνει τρεις παραμέτρους και δεν επιστρέφει τιμή. Η συνάρτηση αυτή είναι υπεύθυνη για την ανανέωση των δεδομένων και της εμφάνισης του παιχνιδιού. Εκτελείται όταν η εστίαση του παιχνιδιού είναι πάνω στην κλάση GameScreen της κλάσης που ανήκει και δεν καλύπτεται από άλλο οπτικό περιβάλλον του παιχνιδιού. Επιπρόσθετα, ορίζεται η ροή των καρέ και η ταχύτητα του παιχνιδιού στον φυσικό κόσμο του επιπέδου.

- Συνάρτηση **Draw**(GameTime)

  Η συνάρτηση Draw παίρνει ως παράμετρο μία μεταβλητή τύπου GameTime και δεν επιστρέφει τιμή. Η συνάρτηση αυτή είναι υπεύθυνη για τη σχεδίαση των γραφικών και της εμφάνισης του παιχνιδιού. Με τη βοήθεια του ScreenManager.SpriteBatch αποκτά τη δυνατότητα σχεδιασμού γραφικών ψηφίδων στο δισδιάστατο χώρο ορίζοντάς τους θέση, χρώμα, διαφάνεια και γενικότερα ρυθμίσεις που είναι απαραίτητες για τη ρύθμιση της εμφάνισης των αντικειμένων και των στοιχείων στο χώρο.

- Συνάρτηση **SkeletonFrameReady**(object sender, SkeletonFrameReadyEventArgs e)

  Η συνάρτηση αυτή υλοποιεί την αλληλεπίδραση του Kinect με τον παίκτη. Μέσω των εργαλείων του SDK εντοπίζεται ο ανθρώπινος σκελετός με τη βοήθεια των αισθητήρων. Κατόπιν, με τη συνάρτηση SkeletonFrameReady λαμβάνουμε τα δεδομένα που επιθυμούμε από την παρακολούθηση και τα εισάγουμε στον αλγόριθμο για την είσοδο των κινήσεων του χρήστη ώστε να σημαδέψει με σωστή γωνία και δύναμη για το που θα εκσφενδονίσει την πέτρα.

- Συνάρτηση **LoadObtstacles**()

  Η δυσκολία της στόχευσης των καρπών έγκειται στο ότι υπάρχουν φυσικά εμπόδια στο χώρο, που κάνουν τη συλλογή των καρπών πιο δύσκολη. Με τη συνάρτηση LoadObstacles φορτώνουμε με στατικά εμπόδια ορθογώνιου σχήματος το χώρο, επιλέγοντας τον αριθμό τους, τις διαστάσεις τους, τη θέση του και την εμφάνισή τους. Δεν παίρνει κάποια παράμετρο και δε επιστρέφει τιμή.

- Συνάρτηση **OnCollisionEnter**()

  Η συνάρτηση OnCollisionEnter είναι η συνάρτηση στην οποία οφείλεται η πραγματοποίηση της κρούσης και της συλλογής των πόντων από τους καρπούς. Εντοπίζοντας την επαφή μεταξύ της πέτρας και του καρπού, δίνεται η εντολή μέσω αυτής της συνάρτησης να μεταβεί στην επόμενη γραφιστική κατάσταση έως ότου συλλεχθεί ο καρπός. Τέλος, κατά τη συλλογή των καρπών προσθέτει τους πόντους στους συνολικούς.

Με την αναφορά στις σημαντικότερες συναρτήσεις της κλάσης Prehistoric_Nuts1 ολοκληρώνεται η περιγραφή της σχεδίασης των μηχανισμών που συνθέτουν το σύστημα. Κατόπιν, εφόσον περιγράφηκαν πολλά από τα σημαντικότερα μέρη της σχεδίασης της εφαρμογής, σκοπός είναι να αναλυθούν εκτενέστερα κάποια κομμάτια ώστε να αναφερθούν οι μέθοδοι υλοποίησης των μηχανισμών, οι αλγόριθμοι που κατασκευάστηκαν και κάποια κομμάτια κώδικα που φανερώνουν βασικές λειτουργίες. Η περιγραφή της υλοποίησης ακολουθεί στο επόμενο κεφάλαιο.

6

Υλοποίηση

Η ενότητα αυτή πρόκειται να αφιερωθεί πλήρως στην ανάπτυξη και την λεπτομερή περιγραφή των μεθόδων που χρησιμοποιήθηκαν ώστε να υλοποιηθούν κάποιοι μηχανισμοί, αλγόριθμοι, κλάσεις, συναρτήσεις και διάφορα άλλα κομμάτια του συστήματος. Θα αναλυθεί ο τρόπος με τον οποίο όλα αυτά συνδέθηκαν μεταξύ τους, θα περιγραφεί η αλληλεξάρτηση μεταξύ των διαφόρων τμημάτων και κάποια σημαντικά σημεία που κρίθηκαν απαραίτητα για να φτάσουμε στην ολοκλήρωση της κατασκευής της εφαρμογής. Έμφαση θα δοθεί σε ότι κρίνεται πιο απαραίτητο και χρήσιμο να αναλυθεί ώστε να μην πλατειάσει και κουράσει η αναφορά σε αμέτρητους μηχανισμούς και κώδικες.

6.1 Λεπτομέρειες υλοποίησης

Κατατάσσοντας στις ίδιες τέσσερις κατηγορίες όπως και στα προηγούμενα κεφάλαια τα κομμάτια της διπλωματικής, θα περιγραφούν αναλυτικά όσα θέματα έχουν τεχνικό και αλγοριθμικό ενδιαφέρον. Ο τρόπος περιγραφής που ακολουθεί συντάσσεται με την παράθεση αρχικά του κώδικα που υλοποιεί τη λειτουργία και ύστερα ακολουθεί η περιγραφή της υλοποίησης και της αλγοριθμικής λογικής με λόγια. Κατά την περιγραφή, αναφέρεται ποιο απόσπασμα του κώδικα που παρατίθεται βρίσκεται υπό επεξήγηση για τη βέλτιστη κατανόηση της κάθε αλγοριθμικής διαδικασίας. Σε ορισμένες περιπτώσεις κρίνεται απαραίτητο να δοθούν παραδείγματα για τη βέλτιστη κατανόηση των μεθόδων.

Όπως προαναφέρθηκε, θα χωριστεί η περιγραφή της υλοποίησης σύμφωνα με το μηχανισμό που εξυπηρετεί κάθε μέθοδος στις εξής κατηγορίες:

- λειτουργία του Microsoft Kinect
- λειτουργία της μηχανής γραφικών
- λειτουργία της μηχανής φυσικής
- λειτουργία της μηχανής του παιχνιδιού

6.1.1 Υλοποίηση μεθόδων λειτουργίας του Microsoft Kinect

Οι κλάσεις οι οποίες επιλέχθηκαν να περιγραφούν στο στάδιο υλοποίησής τους και αφορούν τον αισθητήρα Kinect και τη λειτουργία του, είναι αλγοριθμικά κομμάτια που έχουν αναπτυχθεί αμιγώς για τη συγκεκριμένη εφαρμογή και εξυπηρετούν στη λειτουργία του αισθητήρα μέσα στο παιχνίδι σύμφωνα με τις προδιαγραφές που είχαν τεθεί εξ αρχής. Πιο συγκεκριμένα θα περιγραφούν η κλάση KinectExtensions.cs που προσδίδει απαραίτητα εργαλεία για την αξιοποίηση του αισθητήρα στο δισδιάστατο περιβάλλον και η συνάρτηση SkeletonFrameReady που είναι γραμμένη εσωτερικά στις κλάσεις που εκτελούν τα επίπεδα (Prehistoric_Nuts1) και είναι υπεύθυνη για την είσοδο των δεδομένων στο λογισμικό.

- KinectExtensions.cs

Παράθεση Κώδικα

```
using System;
using Microsoft.Xna.Framework;
using Microsoft.Xna.Framework.Graphics;

namespace Microsoft.Research.Kinect.Nui
{
    public static class KinectExtensions
    {
        public static Texture2D ToTexture2D(this PlanarImage image, GraphicsDevice
graphicsDevice)
        {
            Texture2D texture = new Texture2D(graphicsDevice, image.Width,
image.Height, false, SurfaceFormat.Color);
            Color[] colorData = new Color[image.Width * image.Height];

            int index = 0;
            for (int y = 0; y < image.Height; y++)
            {
                for (int x = 0; x < image.Width; x++, index += image.BytesPerPixel)
                    colorData[y * image.Width + x] = new Color(image.Bits[index + 2],
image.Bits[index + 1], image.Bits[index + 0]);
            }

            texture.SetData(colorData);
            return texture;
        }

        public static Vector2 GetScreenPosition(this Joint joint, Runtime
kinectRuntime, int screenWidth, int screenHeight)
        {
            float depthX;
            float depthY;

            kinectRuntime.SkeletonEngine.SkeletonToDepthImage(joint.Position, out
depthX, out depthY);
            depthX = Math.Max(0, Math.Min(depthX * 320, 320));  //convert to 320, 240
space
            depthY = Math.Max(0, Math.Min(depthY * 240, 240));  //convert to 320, 240
space

            int colorX;
            int colorY;
            // only ImageResolution.Resolution640x480 is supported at this point

kinectRuntime.NuiCamera.GetColorPixelCoordinatesFromDepthPixel(ImageResolution.Resolut
```

```
ion640x480, new ImageViewArea(), (int)depthX, (int)depthY, (short)0, out colorX, out
colorY);

            // map back to skeleton.Width & skeleton.Height
            return new Vector2(screenWidth * colorX / 640.0f, screenHeight * colorY /
480.0f);
        }
    }
}
```

Το παραπάνω απόσπασμα κώδικα συνθέτει την κλάση KinectExtensions η οποία προσφέρει δύο ουσιαστικές λειτουργίες, την συνάρτηση ToTexture2D και την συνάρτηση GetScreenPosition. Παρακάτω εξηγούνται οι λειτουργίες τους και οι αλγόριθμοι με τους οποίους υλοποιήθηκαν.

i. ToTexture2D

Η συνάρτηση αυτή εξυπηρετεί στην μετατροπή των δεδομένων ψηφίδων που δέχεται η κάμερα του Kinect κατά τη λήψη ενός στιγμιότυπου από τους φακούς της σε μεταβλητή τύπου Texture2D για την εύκολη ενσωμάτωσή της στο σύστημα του παιχνιδιού.

Πιο συγκεκριμένα, όπως διαπιστώνεται και στον κώδικα, θέτουμε με τη βοήθεια μιας διπλής for loop τα δεδομένα στη μεταβλητή Texture2D ένα προς ένα διαβάζοντας την πληροφορία που έχει δεχτεί η κάμερα.

ii. GetScreenPosition

Η συνάρτηση GetScreenPosition μας επιτρέπει να μετατρέπουμε την πληροφορία για τη θέση μιας άρθρωσης στον τρισδιάστατο χώρο, που την αντιλαμβάνεται ο αισθητήρας βάθους του Kinect, σε μία θέση στο δισδιάστατο χώρο ώστε να προσαρμόζεται στο παράθυρο προβολής του παιχνιδιού.

Η διαδικασία αυτή επιτυγχάνεται αξιοποιώντας τα δεδομένα για τη θέση της άρθρωσης στο χώρο μέσω εργαλείων του SDK και μετατρέποντας τα σε ένα διάνυσμα δύο συντεταγμένων που αντιστοιχούν στη θέση τους στο εκάστοτε δισδιάστατο παράθυρο προβολής.

Για περισσότερες λεπτομέρειες για τον τρόπο δήλωσης των μεταβλητών αλλά και την χρήση των απαραίτητων συναρτήσεων των αντίστοιχων βιβλιοθηκών για την υλοποίηση του αλγορίθμου σε γλώσσα προγραμματισμού C#, θα ήταν καλό να γίνει λεπτομερής μελέτη του κώδικα που παρατίθεται. Σε πολλά σημεία αναγράφονται εκτενή σχόλια πριν την κάθε λειτουργία ώστε να είναι κατανοητή η μεθοδολογία που ακολουθήθηκε.

- Συνάρτηση KinectFrameReady

Παράθεση Κώδικα

```
        private void SkeletonFrameReady(object sender, SkeletonFrameReadyEventArgs e)
        {
            SkeletonFrame skeletonFrame = e.SkeletonFrame;

            foreach (SkeletonData data in skeletonFrame.Skeletons)
            {
                if (data.TrackingState == SkeletonTrackingState.Tracked)
                {
                    Vector2 jointPosition_Left_Hand = new Vector2(0, 0);
                    Vector2 jointPosition_Right_Hand = new Vector2(0, 0);
                    Vector2 kinect right, kinect left;


                    foreach (Microsoft.Research.Kinect.Nui.Joint joint in data.Joints)
                    {

                        if (joint.ID == JointID.HandLeft)
                        {
                            jointPosition_Left_Hand =
joint.GetScreenPosition(kinectRuntime, ScreenManager.GraphicsDevice.Viewport.Width,
ScreenManager.GraphicsDevice.Viewport.Height);
                            //joint.
                        }



                        if (joint.ID == JointID.HandRight)
                        {
                            jointPosition Right Hand =
joint.GetScreenPosition(kinectRuntime, ScreenManager.GraphicsDevice.Viewport.Width,
ScreenManager.GraphicsDevice.Viewport.Height);
                        }

                        kinect right = jointPosition Right Hand / 5;
                        kinect left = jointPosition Left Hand / 5;



                        if ((float)Math.Abs(-kinect left.X + kinect right.X) > 10.0 &&
(float)Math.Abs(kinect left.Y - kinect right.Y) > 10.0 && time<2000)
                        {

                            spot.X = -493 - kinect_right.X + kinect_left.X;
                            spot.Y = +93 + kinect left.Y - kinect right.Y;

                             agent.Body.LinearVelocity = Vector2.Zero;
                             agent.Body.Rotation = 0;
                            _agent.Body.Position = spot / 25;
                            //_agent.Body.LinearVelocity = 8.0f * (-
 agent.Body.Position + new Vector2(-500.0f / 25.0f, 95.0f / 25.0f) - Vector2.One);

                            time++;
                            score = time;
                        }
                        else if ((float)Math.Abs(-kinect left.X + kinect right.X) >
10.0 && (float)Math.Abs(kinect left.Y - kinect right.Y) > 10.0 && time > 2000)
                        {
                            _agent.Body.LinearVelocity = 8.0f * (-spot / 25 + new
Vector2(-500.0f / 25.0f, 95.0f / 25.0f) - Vector2.One);
                        }
                        else
                        {
                            spot = new Vector2(-487, 93);
                            time = 0;
                        }
                    }
                }
            }
        }
```

Το παραπάνω απόσπασμα κώδικα αποτελεί την συνάρτηση KinectFrameReady η οποία προσφέρει τον βασικό αλγόριθμο λειτουργίας του Kinect ως ελεγκτή εισόδου στο παιχνίδι. Παρακάτω εξηγείται η αλγοριθμική μέθοδος με την οποία λειτουργεί ο μηχανισμός αλληλεπίδρασης που συνδέει την έκταση των χεριών υπό κάποια απόσταση και γωνία με την τελική πορεία του λίθου μετά την εκσφενδόνιση:

i. KinectFrameReady

Στη συνάρτηση αυτή, μέσω μηχανισμών που έχουν αναφερθεί ήδη, ο αλγόριθμος που έχει αναπτυχθεί καταφέρνει να ανιχνεύσει την κίνηση και τη θέση των δύο χεριών και να την αντιστοιχήσει με τον τρόπο σκόπευσης εντός του ψηφιακού περιβάλλοντος. Στην ουσία, είναι η συνάρτηση που προσομοιώνει την κίνηση σκόπευσης του χρήστη με την κίνηση σκόπευσης του ήρωα στο παιχνίδι.

Ειδικότερα, αρχίζοντας την ανίχνευση των αρθρώσεων, εντοπίζουμε και καταγράφουμε τη θέση των δύο άκρων που μας αφορούν στο χώρο, αποθηκεύοντας τις συντεταγμένες σε κατάλληλες μεταβλητές. Ακολούθως, ορίζοντας τη διανυσματική απόσταση των δύο χεριών ως τη διαφορά των συντεταγμένων τους, αυτόματα ορίζουμε την πληροφορία για το μέτρο και τη γωνία της διανυσματικής διαφοράς των δύο σημείων. Τότε, θέτουμε το άκρο του δεξιού χεριού του χρήστη ως το σταθερό σημείο στο οποίο βρίσκεται η σφεντόνα και το αριστερό άκρο ως το χέρι του ήρωα που την τεντώνει. Με τον αλγόριθμο αυτό επιτυγχάνουμε την προσομοίωση στόχευσης με σφεντόνα του χρήστη στον κόσμο του παιχνιδιού.

Κατόπιν, μετά την υλοποίηση του κομματιού που περιγράφεται παραπάνω, κατασκευάστηκε το τελευταίο κομμάτι για τον εκσφενδονισμό του λίθου μετά το πέρας τριών δευτερολέπτων. Ο αλγόριθμος αυτός βασίζεται στον έλεγχο της διαφοράς της απόστασης και της γωνίας ανά στιγμιότυπο. Στην περίπτωση που μένουν σταθερά και τα δύο εντός ενός εύρους τιμών για τρία δευτερόλεπτα (έλεγχος μέσω timer), εκσφενδονίζεται ο λίθος.

Για περισσότερες λεπτομέρειες για τον τρόπο δήλωσης των μεταβλητών αλλά και την χρήση των απαραίτητων συναρτήσεων των αντίστοιχων βιβλιοθηκών για την υλοποίηση του αλγορίθμου σε γλώσσα προγραμματισμού C# θα ήταν καλό να γίνει λεπτομερής μελέτη του κώδικα που παρατίθεται. Σε πολλά σημεία αναγράφονται εκτενή σχόλια πριν την κάθε λειτουργία ώστε να είναι κατανοητή η μεθοδολογία που ακολουθήθηκε.

6.1.2 Υλοποίηση μεθόδων λειτουργίας της Μηχανής Γραφικών

Οι μέθοδοι οι οποίες επιλέχθηκαν να περιγραφούν στο στάδιο υλοποίησής τους και αποτελούν μέρος της λειτουργίας της Μηχανής Γραφικών, είναι αλγοριθμικά κομμάτια που έχουν αναπτυχθεί και χρησιμοποιηθεί για το συγκεκριμένο λογισμικό. Συγκεκριμένα θα περιγραφούν η κλάση AssetCreator.cs, η κλάση Sprite.cs και η κλάση LineBatch.cs που συνιστούν απαραίτητα εργαλεία για την υλοποίηση του γραφικού περιβάλλοντος του παιχνιδιού ώστε να είναι ορθό και λειτουργικό.

- AssetCreator.cs

Παράθεση Κώδικα

```
using System;
using System.Collections.Generic;
using FarseerPhysics.Collision;
using FarseerPhysics.Collision.Shapes;
using FarseerPhysics.Common;
using FarseerPhysics.Common.Decomposition;
using FarseerPhysics.Dynamics;
using Microsoft.Xna.Framework;
using Microsoft.Xna.Framework.Content;
using Microsoft.Xna.Framework.Graphics;

namespace FarseerPhysics.SamplesFramework
{
    public enum MaterialType
    {
        Blank,
        Dots,
        Squares,
        Waves,
        Pavement
    }

    public class AssetCreator
    {
        private const int CircleSegments = 32;

        private GraphicsDevice _device;
        private BasicEffect  effect;
        private Dictionary<MaterialType, Texture2D>  materials = new
Dictionary<MaterialType, Texture2D>();

        public AssetCreator(GraphicsDevice device)
        {
             device = device;
             effect = new BasicEffect( device);
        }

        public static Vector2 CalculateOrigin(Body b)
        {
            Vector2 lBound = new Vector2(float.MaxValue);
            AABB bounds;
            Transform trans;
            b.GetTransform(out trans);

            for (int i = 0; i < b.FixtureList.Count; ++i)
            {
                for (int j = 0; j < b.FixtureList[i].Shape.ChildCount; ++j)
                {
                    b.FixtureList[i].Shape.ComputeAABB(out bounds, ref trans, j);
                    Vector2.Min(ref lBound, ref bounds.LowerBound, out lBound);
                }
```

```
            }
            // calculate body offset from its center and add a 1 pixel border
            // because we generate the textures a little bigger than the actual body's
fixtures
            return ConvertUnits.ToDisplayUnits(b.Position - lBound) + new Vector2(1f);
        }

        public void LoadContent(ContentManager contentManager)
        {
             materials[MaterialType.Blank] =
contentManager.Load<Texture2D>("Materials/blank");
            _materials[MaterialType.Dots] =
contentManager.Load<Texture2D>("Materials/dots");
             materials[MaterialType.Squares] =
contentManager.Load<Texture2D>("Materials/squares");
             materials[MaterialType.Waves] =
contentManager.Load<Texture2D>("Materials/waves");
            _materials[MaterialType.Pavement] =
contentManager.Load<Texture2D>("Materials/pavement");


        }

        public Texture2D TextureFromShape(Shape shape, MaterialType type, Color color,
float materialScale)
        {
            switch (shape.ShapeType)
            {
                case ShapeType.Circle:
                    return CircleTexture(shape.Radius, type, color, materialScale);
                case ShapeType.Polygon:
                    return TextureFromVertices(((PolygonShape) shape).Vertices, type,
color, materialScale);
                default:
                    throw new NotSupportedException("The specified shape type is not
supported.");
            }
        }

        public Texture2D TextureFromVertices(Vertices vertices, MaterialType type,
Color color, float materialScale)
        {
            // copy vertices
            Vertices verts = new Vertices(vertices);

            // scale to display units (i.e. pixels) for rendering to texture
            Vector2 scale = ConvertUnits.ToDisplayUnits(Vector2.One);
            verts.Scale(ref scale);

            // translate the boundingbox center to the texture center
            // because we use an orthographic projection for rendering later
            AABB vertsBounds = verts.GetCollisionBox();
            verts.Translate(-vertsBounds.Center);

            List<Vertices> decomposedVerts;
            if (!verts.IsConvex())
            {
                decomposedVerts = EarclipDecomposer.ConvexPartition(verts);
            }
            else
            {
                decomposedVerts = new List<Vertices>();
                decomposedVerts.Add(verts);
            }
            List<VertexPositionColorTexture[]> verticesFill =
                new List<VertexPositionColorTexture[]>(decomposedVerts.Count);

            materialScale /= _materials[type].Width;

            for (int i = 0; i < decomposedVerts.Count; ++i)
            {
                verticesFill.Add(new VertexPositionColorTexture[3 *
(decomposedVerts[i].Count - 2)]);
                for (int j = 0; j < decomposedVerts[i].Count - 2; ++j)
                {
                    // fill vertices
```

```
                    verticesFill[i][3 * j].Position = new
Vector3(decomposedVerts[i][0], 0f);
                    verticesFill[i][3 * j + 1].Position = new
Vector3(decomposedVerts[i].NextVertex(j), 0f);
                    verticesFill[i][3 * j + 2].Position = new
Vector3(decomposedVerts[i].NextVertex(j + 1), 0f);
                    verticesFill[i][3 * j].TextureCoordinate = decomposedVerts[i][0] *
materialScale;
                    verticesFill[i][3 * j + 1].TextureCoordinate =
decomposedVerts[i].NextVertex(j) * materialScale;
                    verticesFill[i][3 * j + 2].TextureCoordinate =
decomposedVerts[i].NextVertex(j + 1) * materialScale;
                    verticesFill[i][3 * j].Color =
                        verticesFill[i][3 * j + 1].Color = verticesFill[i][3 * j +
2].Color = color;
                }
            }

            // calculate outline
            VertexPositionColor[] verticesOutline = new VertexPositionColor[2 *
verts.Count];
            for (int i = 0; i < verts.Count; ++i)
            {
                verticesOutline[2 * i].Position = new Vector3(verts[i], 0f);
                verticesOutline[2 * i + 1].Position = new Vector3(verts.NextVertex(i),
0f);
                verticesOutline[2 * i].Color = verticesOutline[2 * i + 1].Color =
Color.Black;
            }

            Vector2 vertsSize = new Vector2(vertsBounds.UpperBound.X -
vertsBounds.LowerBound.X,
                                            vertsBounds.UpperBound.Y -
vertsBounds.LowerBound.Y);
            return RenderTexture((int)vertsSize.X, (int)vertsSize.Y,
                                 _materials[type], verticesFill, verticesOutline);
        }

        public Texture2D CircleTexture(float radius, MaterialType type, Color color,
float materialScale)
        {
            return EllipseTexture(radius, radius, type, color, materialScale);
        }

        public Texture2D EllipseTexture(float radiusX, float radiusY, MaterialType
type, Color color,
                                        float materialScale)
        {
            VertexPositionColorTexture[] verticesFill = new
VertexPositionColorTexture[3 * (CircleSegments - 2)];
            VertexPositionColor[] verticesOutline = new VertexPositionColor[2 *
CircleSegments];
            const float segmentSize = MathHelper.TwoPi / CircleSegments;
            float theta = segmentSize;

            radiusX = ConvertUnits.ToDisplayUnits(radiusX);
            radiusY = ConvertUnits.ToDisplayUnits(radiusY);
            materialScale /= _materials[type].Width;

            Vector2 start = new Vector2(radiusX, 0f);

            for (int i = 0; i < CircleSegments - 2; ++i)
            {
                Vector2 p1 = new Vector2(radiusX * (float)Math.Cos(theta), radiusY *
(float)Math.Sin(theta));
                Vector2 p2 = new Vector2(radiusX * (float)Math.Cos(theta +
segmentSize),
                                         radiusY * (float)Math.Sin(theta +
segmentSize));
                // fill vertices
                verticesFill[3 * i].Position = new Vector3(start, 0f);
                verticesFill[3 * i + 1].Position = new Vector3(p1, 0f);
                verticesFill[3 * i + 2].Position = new Vector3(p2, 0f);
                verticesFill[3 * i].TextureCoordinate = start * materialScale;
                verticesFill[3 * i + 1].TextureCoordinate = p1 * materialScale;
                verticesFill[3 * i + 2].TextureCoordinate = p2 * materialScale;
```

```
                verticesFill[3 * i].Color = verticesFill[3 * i + 1].Color =
verticesFill[3 * i + 2].Color = color;

                // outline vertices
                if (i == 0)
                {
                    verticesOutline[0].Position = new Vector3(start, 0f);
                    verticesOutline[1].Position = new Vector3(p1, 0f);
                    verticesOutline[0].Color = verticesOutline[1].Color = Color.Black;
                }
                if (i == CircleSegments - 3)
                {
                    verticesOutline[2 * CircleSegments - 2].Position = new Vector3(p2,
0f);
                    verticesOutline[2 * CircleSegments - 1].Position = new
Vector3(start, 0f);
                    verticesOutline[2 * CircleSegments - 2].Color =
                        verticesOutline[2 * CircleSegments - 1].Color = Color.Black;
                }
                verticesOutline[2 * i + 2].Position = new Vector3(p1, 0f);
                verticesOutline[2 * i + 3].Position = new Vector3(p2, 0f);
                verticesOutline[2 * i + 2].Color = verticesOutline[2 * i + 3].Color =
Color.Black;

                theta += segmentSize;
            }

            return RenderTexture((int)(radiusX * 2f), (int)(radiusY * 2f),
                                 _materials[type], verticesFill, verticesOutline);
        }

        private Texture2D RenderTexture(int width, int height, Texture2D material,
                                        VertexPositionColorTexture[] verticesFill,
                                        VertexPositionColor[] verticesOutline)
        {
            List<VertexPositionColorTexture[]> fill = new
List<VertexPositionColorTexture[]>(1);
            fill.Add(verticesFill);
            return RenderTexture(width, height, material, fill, verticesOutline);
        }

        private Texture2D RenderTexture(int width, int height, Texture2D material,
                                        List<VertexPositionColorTexture[]>
verticesFill,
                                        VertexPositionColor[] verticesOutline)
        {
            Matrix halfPixelOffset = Matrix.CreateTranslation(-0.5f, -0.5f, 0f);
            PresentationParameters pp = _device.PresentationParameters;
            RenderTarget2D texture = new RenderTarget2D( device, width + 2, height +
2, false, SurfaceFormat.Color,
                                                         DepthFormat.None,
pp.MultiSampleCount,

RenderTargetUsage.DiscardContents);
             device.RasterizerState = RasterizerState.CullNone;
            _device.SamplerStates[0] = SamplerState.LinearWrap;

            _device.SetRenderTarget(texture);
            _device.Clear(Color.Transparent);
             effect.Projection = Matrix.CreateOrthographic(width + 2f, -height - 2f,
0f, 1f);
            _effect.View = halfPixelOffset;
            // render shape;
            _effect.TextureEnabled = true;
             effect.Texture = material;
             effect.VertexColorEnabled = true;
             effect.Techniques[0].Passes[0].Apply();
            for (int i = 0; i < verticesFill.Count; ++i)
            {
                _device.DrawUserPrimitives(PrimitiveType.TriangleList,
verticesFill[i], 0, verticesFill[i].Length / 3);
            }
            // render outline;
            _effect.TextureEnabled = false;
            _effect.Techniques[0].Passes[0].Apply();
            _device.DrawUserPrimitives(PrimitiveType.LineList, verticesOutline, 0,
verticesOutline.Length / 2);
```

```
            _device.SetRenderTarget(null);
            return texture;
        }
    }
}
```

Το παραπάνω απόσπασμα κώδικα συνθέτει την κλάση AssetCreator η οποία δίνει τη δυνατότητα δημιουργίας γραφικών σε κάποιο σώμα σύμφωνα με το σχήμα του, το μέγεθος του και την επιλογή ταπετσαρίας σε αυτό σύμφωνα με τις επιλογές που έχουμε εισάγει στις αντίστοιχες μεταβλητές μέσω αρχείων. Παρακάτω εξηγούνται κάποια βασικά στοιχεία της κλάσης αυτής και οι βασικοί αλγόριθμοι με τους οποίους υλοποιήθηκαν.

i. Μεταβλητή **enum MaterialType**

Η μεταβλητή αυτή είναι απλή και ορίζει τους τύπους γραφικών που θα φορτώσουμε τις ταπετσαρίες για τη γραφική απεικόνιση των αντικειμένων του περιβάλλοντος.

ii. Συνάρτηση **CalculateOrigin**(Body)

Η συνάρτηση CalculateOrigin παίρνει ως όρισμα μία μεταβλητή τύπου Body και επιστρέφει μία τιμή μεταβλητής τύπου Vector2. Η συνάρτηση αυτή υπολογίζει το σημείο αναφοράς το αντικειμένου και μας το επιστρέφει.

iii. Συνάρτηση **TextureFromShapes**(Shape, MaterialType, Color, Scale)

Η συνάρτηση TextureFromShapes παίρνει ως όρισμα μία μεταβλητή τύπου Shape, μία MaterialType, μία Color και ένα float για το μέγεθος του σχήματος. Επιστρέφει μία τιμή μεταβλητής τύπου Texture2D. Να σημειωθεί ότι αποτελεί μια γενική συνάρτηση που δέχεται άμεσα την πληροφορία από την κλάση shape και για την επιστροφή της τιμής κάνει χρήση συναρτήσεων τύπου CircleTexture και TextureFromVertices ανάλογα με το σχήμα που του ορίζουμε. Η συνάρτηση TextureFromVertices εξηγείται παρακάτω.

iv. Συνάρτηση **TextureFromVertices**(Vertices, MaterialType, Color, materialScale)

Η συνάρτηση TextureFromVertices παίρνει ως όρισμα μία μεταβλητή τύπου Vertices, μία MaterialType, μία Color και ένα float για το μέγεθος του σχήματος. Επιστρέφει μία τιμή μεταβλητής τύπου Texture2D. Στο σημείο αυτό ακολουθείται μία πολύπλοκη αλγοριθμική διαδικασία με την οποία κατασκευάζουμε ένα σχήμα στο δισδιάστατο χώρο ενώνοντας σημεία με τη σειρά που δίνονται ως δεδομένα στην παράμετρο Vertices.

Ουσιαστικά, με την κλάση AssetCreator έχουμε ένα επιπλέον δυναμικό εργαλείο παραγωγής αντικειμένων διαφόρων σχημάτων και γραφικών που μπορεί να χρησιμοποιηθεί ως βασικό εργαλείο κατασκευής πληθώρας επιπέδων, εύκολα και γρήγορα.

Για περισσότερες λεπτομέρειες για τον τρόπο δήλωσης των μεταβλητών αλλά και την χρήση των απαραίτητων συναρτήσεων των αντίστοιχων βιβλιοθηκών για την υλοποίηση του αλγορίθμου σε γλώσσα προγραμματισμού C# θα ήταν καλό να γίνει λεπτομερής μελέτη του κώδικα που παρατίθεται. Σε πολλά σημεία αναγράφονται εκτενή σχόλια πριν την κάθε λειτουργία ώστε να είναι κατανοητή η μεθοδολογία που ακολουθήθηκε.

- Sprite.cs

Παράθεση Κώδικα

```
using Microsoft.Xna.Framework;
using Microsoft.Xna.Framework.Graphics;

namespace FarseerPhysics.SamplesFramework
{
    public struct Sprite
    {
        public Vector2 Origin;
        public Texture2D Texture;

        public Sprite(Texture2D texture, Vector2 origin)
        {
            this.Texture = texture;
            this.Origin = origin;
        }

        public Sprite(Texture2D sprite)
        {
            Texture = sprite;
            Origin = new Vector2(sprite.Width / 2f, sprite.Height / 2f);
        }
    }
}
```

Το παραπάνω απλό κομμάτι κώδικα ανήκει στην κλάση Sprite, η οποία αντιπροσωπεύει το το σημείο αναφοράς και το γραφικό που χρησιμοποιεί κάποιο σώμα σύμφωνα με το σχήμα του, το μέγεθος του και την επιλογή ταπετσαρίας σε αυτό σύμφωνα με τις επιλογές που έχουμε εισάγει στις αντίστοιχες μεταβλητές μέσω αρχείων.

- LineBatch.cs

Παράθεση Κώδικα

```
using System;
using FarseerPhysics.Collision.Shapes;
using Microsoft.Xna.Framework;
using Microsoft.Xna.Framework.Graphics;

namespace FarseerPhysics.SamplesFramework
{
    public class LineBatch : IDisposable
    {
        private const int DefaultBufferSize = 500;

        // a basic effect, which contains the shaders that we will use to draw our
        // primitives.
        private BasicEffect _basicEffect;

        // the device that we will issue draw calls to.
        private GraphicsDevice _device;

        // hasBegun is flipped to true once Begin is called, and is used to make
        // sure users don't call End before Begin is called.
        private bool  hasBegun;

        private bool _isDisposed;
        private VertexPositionColor[] _lineVertices;
        private int _lineVertsCount;

        public LineBatch(GraphicsDevice graphicsDevice)
            : this(graphicsDevice, DefaultBufferSize)
        {
        }

        public LineBatch(GraphicsDevice graphicsDevice, int bufferSize)
        {
            if (graphicsDevice == null)
            {
                throw new ArgumentNullException("graphicsDevice");
            }
             device = graphicsDevice;

            _lineVertices = new VertexPositionColor[bufferSize - bufferSize % 2];

            // set up a new basic effect, and enable vertex colors.
             basicEffect = new BasicEffect(graphicsDevice);
             basicEffect.VertexColorEnabled = true;
        }

        #region IDisposable Members

        public void Dispose()
        {
            Dispose(true);
            GC.SuppressFinalize(this);
        }

        #endregion

        protected virtual void Dispose(bool disposing)
        {
            if (disposing && ! isDisposed)
            {
                if ( basicEffect != null)
                    _basicEffect.Dispose();

                 isDisposed = true;
            }
        }

        public void Begin(Matrix projection, Matrix view)
        {
            if ( hasBegun)
            {
```

```
                throw new InvalidOperationException("End must be called before Begin
can be called again.");
            }

            _device.SamplerStates[0] = SamplerState.AnisotropicClamp;
            //tell our basic effect to begin.
            _basicEffect.Projection = projection;
             basicEffect.View = view;
             basicEffect.CurrentTechnique.Passes[0].Apply();

            // flip the error checking boolean. It's now ok to call DrawLineShape,
Flush,
            // and End.
             hasBegun = true;
        }

        public void DrawLineShape(Shape shape)
        {
            DrawLineShape(shape, Color.Black);
        }

        public void DrawLineShape(Shape shape, Color color)
        {
            if (!_hasBegun)
            {
                throw new InvalidOperationException("Begin must be called before
DrawLineShape can be called.");
            }
            if (shape.ShapeType != ShapeType.Edge &&
                shape.ShapeType != ShapeType.Loop)
            {
                throw new NotSupportedException("The specified shapeType is not
supported by LineBatch.");
            }
            if (shape.ShapeType == ShapeType.Edge)
            {
                if ( lineVertsCount >=  lineVertices.Length)
                {
                    Flush();
                }
                EdgeShape edge = (EdgeShape)shape;
                _lineVertices[_lineVertsCount].Position = new Vector3(edge.Vertex1,
0f);
                 lineVertices[ lineVertsCount + 1].Position = new
Vector3(edge.Vertex2, 0f);
                _lineVertices[_lineVertsCount].Color = _lineVertices[_lineVertsCount +
1].Color = color;
                _lineVertsCount += 2;
            }
            else if (shape.ShapeType == ShapeType.Loop)
            {
                LoopShape loop = (LoopShape)shape;
                for (int i = 0; i < loop.Vertices.Count; ++i)
                {
                    if ( lineVertsCount >=  lineVertices.Length)
                    {
                        Flush();
                    }
                    _lineVertices[_lineVertsCount].Position = new
Vector3(loop.Vertices[i], 0f);
                     lineVertices[ lineVertsCount + 1].Position = new
Vector3(loop.Vertices.NextVertex(i), 0f);
                    _lineVertices[_lineVertsCount].Color =
_lineVertices[_lineVertsCount + 1].Color = color;
                     lineVertsCount += 2;
                }
            }
        }

        public void DrawLine(Vector2 v1, Vector2 v2)
        {
            DrawLine(v1, v2, Color.Black);
        }

        public void DrawLine(Vector2 v1, Vector2 v2, Color color)
        {
            if (!_hasBegun)
```

```
            {
                throw new InvalidOperationException("Begin must be called before
DrawLineShape can be called.");
            }
            if (_lineVertsCount >= _lineVertices.Length)
            {
                Flush();
            }
             lineVertices[ lineVertsCount].Position = new Vector3(v1, 0f);
             lineVertices[ lineVertsCount + 1].Position = new Vector3(v2, 0f);
            _lineVertices[_lineVertsCount].Color = _lineVertices[_lineVertsCount +
1].Color = color;
            _lineVertsCount += 2;
        }

        // End is called once all the primitives have been drawn using AddVertex.
        // it will call Flush to actually submit the draw call to the graphics card,
and
        // then tell the basic effect to end.
        public void End()
        {
            if (! hasBegun)
            {
                throw new InvalidOperationException("Begin must be called before End
can be called.");
            }

            // Draw whatever the user wanted us to draw
            Flush();

             hasBegun = false;
        }

        private void Flush()
        {
            if (!_hasBegun)
            {
                throw new InvalidOperationException("Begin must be called before Flush
can be called.");
            }
            if (_lineVertsCount >= 2)
            {
                int primitiveCount =  lineVertsCount / 2;
                // submit the draw call to the graphics card
                _device.DrawUserPrimitives(PrimitiveType.LineList, _lineVertices, 0,
primitiveCount);
                _lineVertsCount -= primitiveCount * 2;
            }
        }
    }
}
```

Το παραπάνω απόσπασμα κώδικα συνθέτει την κλάση LineBatch η οποία δίνει τη δυνατότητα δημιουργίας γραφικών που αποτελούν στην ουσία γραμμές σε κάποιο σώμα σύμφωνα με το σχήμα του, είτε απλή γραμμή μεταξύ δύο σημείων. Παρακάτω εξηγούνται κάποια τα στοιχεία της κλάσης αυτής και οι βασικοί αλγόριθμοι με τους οποίους υλοποιήθηκαν.

i. Συνάρτηση **DrawLineShape**(Shape)

Η συνάρτηση DrawLineShape παίρνει ως όρισμα μία μεταβλητή τύπου Shape και δεν επιστρέφει τιμή. Η συνάρτηση αυτή σχεδιάζει το περίγραμμα ενός αντικειμένου.

ii. Συνάρτηση **DrawLine** (Vector2, Vector2)

Η συνάρτηση DrawLine παίρνει ως όρισμα δύο μεταβλητές τύπου Vector2 και δεν επιστρέφει καμία τιμή. Η συνάρτηση αυτή σχεδιάζει το ευθύγραμμο τμήμα μεταξύ δύο σημείων με δισδιάστατες συντεταγμένες.

Για περισσότερες λεπτομέρειες για τον τρόπο δήλωσης των μεταβλητών αλλά και την χρήση των απαραίτητων συναρτήσεων των αντίστοιχων βιβλιοθηκών για την υλοποίηση του αλγορίθμου σε γλώσσα προγραμματισμού C# θα ήταν καλό να γίνει λεπτομερής μελέτη του κώδικα που παρατίθεται. Σε πολλά σημεία αναγράφονται εκτενή σχόλια πριν την κάθε λειτουργία ώστε να είναι κατανοητή η μεθοδολογία που ακολουθήθηκε.

6.1.3 Υλοποίηση μεθόδων λειτουργίας της Μηχανής Φυσικής

Σε αυτό το σημείο θα αναλυθεί η υλοποίηση της κυριότερης κλάσης της Μηχανής Φυσικής και μόνο αυτή, μιας και αποτελεί ουσιαστικά και το βασικό στοιχείο αυτής. Με τις ιδιότητες και τους μηχανισμούς με τους οποίους κατασκευάστηκε πραγματοποιεί την φυσική αλληλεπίδραση των αντικειμένων μεταξύ τους στο περιβάλλον του παιχνιδιού. Να σημειωθεί ότι κατά την παράθεση του κώδικα της κλάσης αφαιρέθηκαν δευτερεύοντα κομμάτια τα οποία δε θα περιγραφούν και δεν χρήζουν ιδιαίτερης έμφασης για τη διευκόλυνση του αναγνώστη και αποφυγή παράθεσης μακροσκελούς συνεχόμενου κώδικα.

- Body.cs

Παράθεση Κώδικα

```
namespace FarseerPhysics.Dynamics
{
    /// <summary>
    /// The body type.
    /// </summary>
    public enum BodyType
    {
        /// <summary>
        /// Zero velocity, may be manually moved. Note: even static bodies have mass.
        /// </summary>
        Static,
        /// <summary>
        /// Zero mass, non-zero velocity set by user, moved by solver
        /// </summary>
        Kinematic,
        /// <summary>
        /// Positive mass, non-zero velocity determined by forces, moved by solver
        /// </summary>
        Dynamic,
    }


    public class Body : IDisposable
    {
```

```
        private static int _bodyIdCounter;
        internal float AngularVelocityInternal;
        public int BodyId;
        public ControllerFilter ControllerFilter;
        internal BodyFlags Flags;
        internal Vector2 Force;
        internal float InvI;
        internal float InvMass;
        internal Vector2 LinearVelocityInternal;
        public PhysicsLogicFilter PhysicsLogicFilter;
        internal float SleepTime;
        internal Sweep Sweep; // the swept motion for CCD
        internal float Torque;
        internal World World;
        internal Transform Xf; // the body origin transform
        private float  angularDamping;
        private BodyType _bodyType;
        private float _inertia;
        private float _linearDamping;
        private float  mass;
        private string objTypeName;
        Texture2D textureImage;
        protected Point frameSize;
        Point currentFrame;
        Point sheetSize;
        internal Body()
        {
            FixtureList = new List<Fixture>(32);
        }

        public Body(World world)
            : this(world, null)
        {
        }

        public Body(World world, object userData)
        {
            FixtureList = new List<Fixture>(32);
            BodyId =  bodyIdCounter++;

            World = world;
            UserData = userData;

            FixedRotation = false;
            IsBullet = false;
            SleepingAllowed = true;
            Awake = true;
            BodyType = BodyType.Static;
            Enabled = true;

            Xf.R.Set(0);

            world.AddBody(this);
        }

        public void SetTexture(Texture2D texture)
        {
            this.textureImage = texture;
        }

        public Texture2D GetTexture()
        {
            return this.textureImage;
        }

        public void SetFrameSize(Point frameSizeAdded)
        {
            this.frameSize = frameSizeAdded;
        }

        public Point GetFrameSize()
        {
            return this.frameSize;
        }

        public void SetCurrentFrame(Point currentFrameAdded)
        {
```

```
            this.currentFrame = currentFrameAdded;
        }

        public Point GetCurrentFrame()
        {
            return this.currentFrame;
        }

        public void SetSheetSize(Point sheetSizeAdded)
        {
            this.sheetSize = sheetSizeAdded;
        }

        public Point GetSheetSize()
        {
            return this.sheetSize;
        }

        public void SetTypeName(string objTypeName)
        { this.objTypeName = objTypeName; }

        public string GetTypeName() { return objTypeName; }

        /// <summary>
        /// Gets the total number revolutions the body has made.
        /// </summary>
        /// <value>The revolutions.</value>
        ///

        /// <summary>
        /// Gets or sets the body type.
        /// </summary>
        /// <value>The type of body.</value>
        public BodyType BodyType
        {
            get { return _bodyType; }
            set
            {
                if ( bodyType == value)
                {
                    return;
                }

                 bodyType = value;

                ResetMassData();

                if (_bodyType == BodyType.Static)
                {
                    LinearVelocityInternal = Vector2.Zero;
                    AngularVelocityInternal = 0.0f;
                }

                Awake = true;

                Force = Vector2.Zero;
                Torque = 0.0f;

                // Since the body type changed, we need to flag contacts for
filtering.
                for (int i = 0; i < FixtureList.Count; i++)
                {
                    Fixture f = FixtureList[i];
                    f.Refilter();
                }
            }
        }

        /// <summary>
        /// Get or sets the linear velocity of the center of mass.
        /// </summary>
        /// <value>The linear velocity.</value>
        public Vector2 LinearVelocity
        {
            set
            {
                Debug.Assert(!float.IsNaN(value.X) && !float.IsNaN(value.Y));
```

```
                if ( bodyType == BodyType.Static)
                    return;

                if (Vector2.Dot(value, value) > 0.0f)
                    Awake = true;

                LinearVelocityInternal = value;
            }
            get { return LinearVelocityInternal; }
        }



        public bool Enabled
        {
            set
            {
                if (value == Enabled)
                {
                    return;
                }

                if (value)
                {
                    Flags |= BodyFlags.Enabled;

                    // Create all proxies.
                    IBroadPhase broadPhase = World.ContactManager.BroadPhase;
                    for (int i = 0; i < FixtureList.Count; i++)
                    {
                        FixtureList[i].CreateProxies(broadPhase, ref Xf);
                    }

                    // Contacts are created the next time step.
                }
                else
                {
                    Flags &= ~BodyFlags.Enabled;

                    // Destroy all proxies.
                    IBroadPhase broadPhase = World.ContactManager.BroadPhase;

                    for (int i = 0; i < FixtureList.Count; i++)
                    {
                        FixtureList[i].DestroyProxies(broadPhase);
                    }

                    // Destroy the attached contacts.
                    ContactEdge ce = ContactList;
                    while (ce != null)
                    {
                        ContactEdge ce0 = ce;
                        ce = ce.Next;
                        World.ContactManager.Destroy(ce0.Contact);
                    }
                    ContactList = null;
                }
            }
            get { return (Flags & BodyFlags.Enabled) == BodyFlags.Enabled; }
        }

        /// <summary>
        /// Set this body to have fixed rotation. This causes the mass
        /// to be reset.
        /// </summary>
        /// <value><c>true</c> if it has fixed rotation; otherwise,
<c>false</c>.</value>
        public bool FixedRotation
        {
            set
            {
                if (value)
                {
                    Flags |= BodyFlags.FixedRotation;
                }
                else
```

```
                {
                    Flags &= ~BodyFlags.FixedRotation;
                }

                ResetMassData();
            }
            get { return (Flags & BodyFlags.FixedRotation) == BodyFlags.FixedRotation;
}
        }

        /// <summary>
        /// Gets all the fixtures attached to this body.
        /// </summary>
        /// <value>The fixture list.</value>
        public List<Fixture> FixtureList { get; internal set; }

        /// <summary>
        /// Get the list of all joints attached to this body.
        /// </summary>
        /// <value>The joint list.</value>

        /// <summary>
        /// Get the world body origin position.
        /// </summary>
        /// <returns>Return the world position of the body's origin.</returns>
        public Vector2 Position
        {
            get { return Xf.Position; }
            set
            {
                Debug.Assert(!float.IsNaN(value.X) && !float.IsNaN(value.Y));

                SetTransform(ref value, Rotation);
            }
        }

        /// <summary>
        /// Get the angle in radians.
        /// </summary>
        /// <returns>Return the current world rotation angle in radians.</returns>
        public float Rotation
        {
            get { return Sweep.A; }
            set
            {
                Debug.Assert(!float.IsNaN(value));

                SetTransform(ref Xf.Position, value);
            }
        }

        /// <summary>
        /// Gets or sets a value indicating whether this body is static.
        /// </summary>
        /// <value><c>true</c> if this instance is static; otherwise,
<c>false</c>.</value>
        public bool IsStatic
        {
            get { return _bodyType == BodyType.Static; }
            set
            {
                if (value)
                    BodyType = BodyType.Static;
                else
                    BodyType = BodyType.Dynamic;
            }
        }

        /// <summary>
        /// Gets or sets a value indicating whether this body ignores gravity.
        /// </summary>
        /// <value><c>true</c> if  it ignores gravity; otherwise,
<c>false</c>.</value>
        public bool IgnoreGravity
        {
            get { return (Flags & BodyFlags.IgnoreGravity) == BodyFlags.IgnoreGravity;
}
```

```
            set
            {
                if (value)
                    Flags |= BodyFlags.IgnoreGravity;
                else
                    Flags &= ~BodyFlags.IgnoreGravity;
            }
        }

        public float Mass
        {
            get { return _mass; }
            set
            {
                Debug.Assert(!float.IsNaN(value));

                if (_bodyType != BodyType.Dynamic)
                    return;

                 mass = value;

                if ( mass <= 0.0f)
                    _mass = 1.0f;

                InvMass = 1.0f /  mass;
            }
        }

        /// <summary>
        /// Get or set the rotational inertia of the body about the local origin.
usually in kg-m^2.
        /// </summary>
        /// <value>The inertia.</value>
        public float Inertia
        {
            get { return _inertia + Mass * Vector2.Dot(Sweep.LocalCenter,
Sweep.LocalCenter); }
            set
            {
                Debug.Assert(!float.IsNaN(value));

                if (_bodyType != BodyType.Dynamic)
                    return;

                if (value > 0.0f && (Flags & BodyFlags.FixedRotation) == 0)
                {
                    _inertia = value - Mass * Vector2.Dot(LocalCenter, LocalCenter);
                    Debug.Assert(_inertia > 0.0f);
                    InvI = 1.0f /  inertia;
                }
            }
        }



        public float Friction
        {
            get
            {
                float res = 0;

                for (int i = 0; i < FixtureList.Count; i++)
                {
                    Fixture f = FixtureList[i];
                    res += f.Friction;
                }

                return res / FixtureList.Count;
            }
            set
            {
                for (int i = 0; i < FixtureList.Count; i++)
                {
                    Fixture f = FixtureList[i];
                    f.Friction = value;
                }
            }
```

```
        }


        #endregion

        public Fixture CreateFixture(Shape shape)
        {
            return new Fixture(this, shape);
        }

        /// <summary>
        /// Creates a fixture and attach it to this body.
        /// If the density is non-zero, this function automatically updates the mass
of the body.
        /// Contacts are not created until the next time step.
        /// Warning: This function is locked during callbacks.
        /// </summary>
        /// <param name="shape">The shape.</param>
        /// <param name="userData">Application specific data</param>
        /// <returns></returns>
        public Fixture CreateFixture(Shape shape, object userData)
        {
            return new Fixture(this, shape, userData);
        }

        /// <summary>
        /// Destroy a fixture. This removes the fixture from the broad-phase and
        /// destroys all contacts associated with this fixture. This will
        /// automatically adjust the mass of the body if the body is dynamic and the
        /// fixture has positive density.
        /// All fixtures attached to a body are implicitly destroyed when the body is
destroyed.
        /// Warning: This function is locked during callbacks.
        /// </summary>
        /// <param name="fixture">The fixture to be removed.</param>
        public void DestroyFixture(Fixture fixture)
        {
            Debug.Assert(fixture.Body == this);

            // Remove the fixture from this body's singly linked list.
            Debug.Assert(FixtureList.Count > 0);

            // You tried to remove a fixture that not present in the fixturelist.
            Debug.Assert(FixtureList.Contains(fixture));

            // Destroy any contacts associated with the fixture.
            ContactEdge edge = ContactList;
            while (edge != null)
            {
                Contact c = edge.Contact;
                edge = edge.Next;

                Fixture fixtureA = c.FixtureA;
                Fixture fixtureB = c.FixtureB;

                if (fixture == fixtureA || fixture == fixtureB)
                {
                    // This destroys the contact and removes it from
                    // this body's contact list.
                    World.ContactManager.Destroy(c);
                }
            }

            if ((Flags & BodyFlags.Enabled) == BodyFlags.Enabled)
            {
                IBroadPhase broadPhase = World.ContactManager.BroadPhase;
                fixture.DestroyProxies(broadPhase);
            }

            FixtureList.Remove(fixture);
            fixture.Destroy();
            fixture.Body = null;

            ResetMassData();
        }

        /// <summary>
```

```
        /// Set the position of the body's origin and rotation.
        /// This breaks any contacts and wakes the other bodies.
        /// Manipulating a body's transform may cause non-physical behavior.
        /// </summary>
        /// <param name="position">The world position of the body's local
origin.</param>
        /// <param name="rotation">The world rotation in radians.</param>
        public void SetTransform(ref Vector2 position, float rotation)
        {
            SetTransformIgnoreContacts(ref position, rotation);

            World.ContactManager.FindNewContacts();
        }

        /// <summary>
        /// Set the position of the body's origin and rotation.
        /// This breaks any contacts and wakes the other bodies.
        /// Manipulating a body's transform may cause non-physical behavior.
        /// </summary>
        /// <param name="position">The world position of the body's local
origin.</param>
        /// <param name="rotation">The world rotation in radians.</param>
        public void SetTransform(Vector2 position, float rotation)
        {
            SetTransform(ref position, rotation);
        }

        /// <summary>
        /// Apply a force at a world point. If the force is not
        /// applied at the center of mass, it will generate a torque and
        /// affect the angular velocity. This wakes up the body.
        /// </summary>
        /// <param name="force">The world force vector, usually in Newtons
(N).</param>
        /// <param name="point">The world position of the point of
application.</param>
        public void ApplyForce(Vector2 force, Vector2 point)
        {
            ApplyForce(ref force, ref point);
        }

        /// <summary>
        /// Applies a force at the center of mass.
        /// </summary>
        /// <param name="force">The force.</param>
        public void ApplyForce(ref Vector2 force)
        {
            ApplyForce(ref force, ref Xf.Position);
        }

        /// <summary>
        /// Applies a force at the center of mass.
        /// </summary>
        /// <param name="force">The force.</param>
        public void ApplyForce(Vector2 force)
        {
            ApplyForce(ref force, ref Xf.Position);
        }

        /// <summary>
        /// Apply a force at a world point. If the force is not
        /// applied at the center of mass, it will generate a torque and
        /// affect the angular velocity. This wakes up the body.
        /// </summary>
        /// <param name="force">The world force vector, usually in Newtons
(N).</param>
        /// <param name="point">The world position of the point of
application.</param>
        public void ApplyForce(ref Vector2 force, ref Vector2 point)
        {
            Debug.Assert(!float.IsNaN(force.X));
            Debug.Assert(!float.IsNaN(force.Y));
            Debug.Assert(!float.IsNaN(point.X));
            Debug.Assert(!float.IsNaN(point.Y));

            if (_bodyType == BodyType.Dynamic)
            {
```

```
                if (Awake == false)
                {
                    Awake = true;
                }

                Force += force;
                Torque += (point.X - Sweep.C.X) * force.Y - (point.Y - Sweep.C.Y) *
force.X;
            }
        }

        /// <summary>
        /// Apply a torque. This affects the angular velocity
        /// without affecting the linear velocity of the center of mass.
        /// This wakes up the body.
        /// </summary>
        /// <param name="torque">The torque about the z-axis (out of the screen),
usually in N-m.</param>
        public void ApplyTorque(float torque)
        {
            Debug.Assert(!float.IsNaN(torque));

            if (_bodyType == BodyType.Dynamic)
            {
                if (Awake == false)
                {
                    Awake = true;
                }

                Torque += torque;
            }
        }

    }
}
```

Ο παραπάνω κώδικας συνθέτει την κλάση Body η οποία αποτελεί τον βασικό κορμό κάθε αντικειμένου το οποίο δύναται να αλληλεπιδρά μέσα στο περιβάλλον του παιχνιδιού με το υπόλοιπο σύστημα βάσει των φυσικών νόμων που ορίζει το περιβάλλον. Είναι ένα μακροσκελές τμήμα κώδικα, αλλά παρατίθεται σχεδόν ολόκληρο μιας και η σημασία του κατά την υλοποίηση των μηχανισμών φυσικής αποτέλεσε την επιτυχή ολοκλήρωση του μεγαλύτερου μέρους της λειτουργίας του μηχανισμού. Τα στοιχεία που περιγράφονται είναι τα εξής:

i. Μεταβλητή τύπου Enum, **BodyType**

   Η μεταβλητή αυτή ορίζει τον τύπο του Body. Περιέχει τις εξής τρεις επιλογές, Static, Kinematic και Dynamic.

   Στην πρώτη το σώμα έχει ταχύτητα και μπορεί να κινηθεί μόνο κατ' επιλογή του χρήστη. Μπορεί να οριστεί μάζα στο σώμα τύπου Static ώστε να αλληλεπιδρά με αυτή στους υπολογισμούς των φυσικών νόμων.

   Στην επιλογή Kinematic, το σώμα έχει ταχύτητα που ορίζεται από το χρήστη ή το σύστημα αλλά έχει μηδενική μάζα.

   Στην επιλογή Dynamic, το σώμα έχει ταχύτητα που ορίζεται από το χρήστη ή το σύστημα και πεπερασμένη μάζα.

ii. Μεταβλητή τύπου Vector2, **LinearVelocity**

Η μεταβλητή αυτή ορίζει την δισδιάστατη ταχύτητα του στοιχείου Body. Μπορεί να οριστεί και να διαβαστεί.

iii. Μεταβλητή τύπου Vector2, **Position**

Η μεταβλητή αυτή ορίζει τη δισδιάστατη θέση του στοιχείου Body. Μπορεί να οριστεί και να διαβαστεί.

iv. Μεταβλητή τύπου Vector2, **Rotation**

Η μεταβλητή αυτή ορίζει τη δισδιάστατη περιστροφή του στοιχείου Body. Μπορεί να οριστεί και να διαβαστεί.

v. Μεταβλητή τύπου float, **Mass**

Η μεταβλητή αυτή ορίζει τη μάζα του στοιχείου Body. Μπορεί να οριστεί και να διαβαστεί.

vi. Μεταβλητή τύπου float, **Inertia**

Η μεταβλητή αυτή ορίζει την αδράνεια του στοιχείου Body. Μπορεί να οριστεί και να διαβαστεί.

vii. Μεταβλητή τύπου float, **Friction**

Η μεταβλητή αυτή ορίζει την τριβή του στοιχείου Body σε σχέση με τα άλλα σώματα. Μπορεί να οριστεί και να διαβαστεί.

viii. Συνάρτηση **DestroyFixture**(Fixture)

Η συνάρτηση DestroyFixture αφαιρεί και καταστρέφει από κάποιο σώμα τη δομή του, το Fixture που ανήκει σε αυτό το Body. Ταυτόχρονα μηδενίζει και τη μάζα του ώστε το σώμα να μην αλληλεπιδρά με τα τους μαθηματικούς αλγορίθμους. Δεν επιστρέφει κάποια τιμή.

ix. Συνάρτηση **ApplyForce**(Vector2)

Η συνάρτηση ApplyForce όταν εκτελείται από κάποιο Body ασκεί δύναμη στο σώμα αυτό προς κάποια κατεύθυνση που ορίζεται από τις τιμές της μεταβλητής Vector2 που παίρνει ως όρισμα. Δεν επιστρέφει κάποια τιμή.

x. Συνάρτηση **ApplyTorque**(float)

Η συνάρτηση ApplyTorque όταν εκτελείται από κάποιο Body ασκεί ροπή στο σώμα αυτό μεγέθους που ορίζεται από την τιμή της μεταβλητής float που παίρνει ως όρισμα. Δεν επιστρέφει κάποια τιμή.

Για περισσότερες λεπτομέρειες για τον τρόπο δήλωσης των μεταβλητών αλλά και την χρήση των απαραίτητων συναρτήσεων των αντίστοιχων βιβλιοθηκών για την υλοποίηση του αλγορίθμου σε γλώσσα προγραμματισμού C# θα ήταν καλό να γίνει λεπτομερής μελέτη του κώδικα που παρατίθεται. Σε πολλά σημεία αναγράφονται εκτενή σχόλια πριν την κάθε λειτουργία ώστε να είναι κατανοητή η μεθοδολογία που ακολουθήθηκε.

6.1.4 Υλοποίηση μεθόδων λειτουργίας της Μηχανής Παιχνιδιού

Σε αυτό το σημείο θα αναλυθεί η υλοποίηση της κυριότερης κλάσης της Μηχανής Φυσικής και μόνο αυτή, μιας και αποτελεί ουσιαστικά και το βασικό στοιχείο το οποίο με τις ιδιότητες και τους μηχανισμούς με τους οποίους κατασκευάστηκε πραγματοποιεί την φυσική αλληλεπίδραση των αντικειμένων μεταξύ τους στο περιβάλλον του παιχνιδιού. Να σημειωθεί ότι κατά την παράθεση του κώδικα της κλάσης αφαιρέθηκαν δευτερεύοντα κομμάτια τα οποία δε θα περιγραφούν και δεν χρήζουν ιδιαίτερης έμφασης για τη διευκόλυνση του αναγνώστη και αποφυγή παράθεσης μακροσκελούς συνεχόμενου κώδικα.

- Prehistoric_Nuts1.cs

Παράθεση Κώδικα

```
using System.Text;
using System;
using System.Collections.Generic;
using System.Linq;
using FarseerPhysics.Collision.Shapes;
using FarseerPhysics.Common;
using FarseerPhysics.Dynamics;
using FarseerPhysics.Dynamics.Joints;
using FarseerPhysics.Factories;
using Microsoft.Xna.Framework;
using Microsoft.Xna.Framework.Audio;
using Microsoft.Xna.Framework.Content;
using Microsoft.Xna.Framework.GamerServices;
```

```
using Microsoft.Xna.Framework.Graphics;
using Microsoft.Xna.Framework.Input;
using Microsoft.Xna.Framework.Media;
using Microsoft.Research.Kinect.Nui;
using Microsoft.Research.Kinect.Audio;

namespace FarseerPhysics.SamplesFramework
{

    internal class Prehistoric Nuts1 : PhysicsGameScreen, IDemoScreen
    {
        static Runtime kinectRuntime;

        private readonly TextureInstance kinectRGBVideo = new TextureInstance();
        private readonly List<TextureInstance> hotSpots = new List<TextureInstance>();
        float credits = 20, angle text;
        static float score = 0;
        int time = 0;
        private Agent _agent;
        private Border  border;
        private Sprite  obstacle;
        private Body[]  obstacles = new Body[5];
        Texture2D Background, dot, stone, macnut, haznut, ui;
        Pyramid mac_nuts, haz_nuts;
        SpriteFont font, font serious;
        Vector2 spot;
        private Body man;
        private Sprite manBody;
        private float _scale;
        bool pressed;



        #region IDemoScreen Members

        public string GetTitle()
        {
            return "Level 1";
        }



        public string GetDetails()
        {
            StringBuilder sb = new StringBuilder();

            sb.AppendLine("This is the first level (Level 1) of Prehistoric Nuts");
            sb.AppendLine(string.Empty);
            sb.AppendLine("Information");
            sb.AppendLine(string.Empty);
            sb.AppendLine("Kinect");
            sb.AppendLine("  - Stretch your hands to be detected from Kinect");
            sb.AppendLine("  - Target with your hands");
            sb.AppendLine("  - Check the angle between your hands");
            sb.AppendLine("  - Check the the stretching of your hands");
            sb.AppendLine("  - Stay static for 3 seconds when you are sure for your
target");
            sb.AppendLine("  - Break more and more Nuts");
            sb.AppendLine(string.Empty);

            sb.AppendLine("  - Exit to menu: Escape");


            return sb.ToString();


        }

        #endregion
```

```
        //public static bool collided = false;
        bool collided = false;

        bool OnCollisionEnter(Fixture fixtureA, Fixture fixtureB,
FarseerPhysics.Dynamics.Contacts.Contact contact)
        {
            collided = false;
            if (!collided)
            {
                Point col point;
                col_point = fixtureB.Body.GetCurrentFrame();
                col_point.X++;
                fixtureB.Body.SetCurrentFrame(col_point);

                if (fixtureB.Body.GetTypeName() == "haz nut")
                {
                    if (fixtureB.Body.GetCurrentFrame().X >
fixtureB.Body.GetSheetSize().X - 2)
                    {
                        score += 50;
                        col point.X++;
                        fixtureB.Body.SetCurrentFrame(col point);
                        fixtureB.Body.DestroyFixture(fixtureB);
                    }

                }
                else if (fixtureB.Body.GetTypeName() == "mac nut")
                {
                    if (fixtureB.Body.GetCurrentFrame().X >
fixtureB.Body.GetSheetSize().X - 2)
                    {
                        score += 100;
                        col point.X++;
                        fixtureB.Body.SetCurrentFrame(col_point);
                        fixtureB.Body.DestroyFixture(fixtureB);
                    }
                }
            }

            return true;
        }


        public override void LoadContent()
        {



            kinectRuntime = new Runtime();
            kinectRuntime.Uninitialize();
            kinectRuntime.Initialize(RuntimeOptions.UseColor |
RuntimeOptions.UseSkeletalTracking);
            kinectRuntime.VideoStream.Open(ImageStreamType.Video, 2,
ImageResolution.Resolution640x480, ImageType.Color);

            kinectRuntime.VideoFrameReady += VideoFrameReady;
            kinectRuntime.SkeletonFrameReady += SkeletonFrameReady;

            kinectRGBVideo.Texture = new Texture2D(ScreenManager.GraphicsDevice, 320,
240, false, SurfaceFormat.Color);
            //******************

            base.LoadContent();
            stone = ScreenManager.Content.Load<Texture2D>("Common/agent");
            ui = ScreenManager.Content.Load<Texture2D>("Common/ui table");
            spot = new Vector2(-480, 90);
            macnut = ScreenManager.Content.Load<Texture2D>("Common/mac nut");
            haznut = ScreenManager.Content.Load<Texture2D>("Common/haz_nut");

            _scale = 1;
            font = ScreenManager.Content.Load<SpriteFont>("Fonts/SpriteFont2");
            font serious = ScreenManager.Content.Load<SpriteFont>("Fonts/1menufont");


            World.Gravity = new Vector2(0f, 20f);

            mac_nuts = new Pyramid(World, this, new Vector2(8.0f, -1.0f), 2, 0.6f);
```

```
            haz_nuts = new Pyramid(World, this, new Vector2(15.0f, -5.0f), 2, 0.6f);
            mac_nuts.LoadTexture(ScreenManager.Content.Load<Texture2D>("Common/mac"));

            mac_nuts.SetTypeName("mac nut");

            haz_nuts.LoadTexture(ScreenManager.Content.Load<Texture2D>("Common/haz"));
            haz_nuts.SetTypeName("haz_nut");

             border = new Border(World, this, ScreenManager.GraphicsDevice.Viewport);

            _agent = new Agent(World, this, new Vector2(1.0f, -24.5f));
            _agent.LoadTexture(ScreenManager.Content.Load<Texture2D>("Common/agent"));
            _agent.Body.Mass = 0.4f;
            dot = ScreenManager.Content.Load<Texture2D>("Common/dot");

            Background = ScreenManager.Content.Load<Texture2D>("Common/background");

            pressed = false;

            LoadObstacles();

            SetUserAgent( agent.Body, 1000f, 400f);

            // man
            {
                Vertices vertices = PolygonTools.CreateRectangle(0.01f, 0.01f);
                PolygonShape chassis = new PolygonShape(vertices, 0f);
                man = new Body(World);
                man.BodyType = BodyType.Static;
                man.Position = new Vector2(-22.0f, 6.5f);
                manBody = new
Sprite(ScreenManager.Content.Load<Texture2D>("Common/main unit"));

            }
            _agent.Body.OnCollision += OnCollisionEnter;
        }



        private void LoadObstacles()
        {
            for (int i = 0; i < 3; ++i)
            {
                _obstacles[i] = BodyFactory.CreateRectangle(World, 5f, 1f, 1f);
                _obstacles[i].IsStatic = true;
                _obstacles[i].Restitution = 0.2f;
                _obstacles[i].Friction = 0.2f;
            }

            _obstacles[0].Position = new Vector2(23f, 5.5f);
            _obstacles[1].Position = new Vector2(13f, 6f);
            _obstacles[2].Position = new Vector2(20f, 3f);

            // create sprite based on body
            _obstacle = new
Sprite(ScreenManager.Assets.TextureFromShape(_obstacles[0].FixtureList[0].Shape,
MaterialType.Dots, new Color(128, 100 , 17), 0.8f));

        }


        public override void Update(GameTime gameTime, bool otherScreenHasFocus, bool
coveredByOtherScreen)
        {
            if (!coveredByOtherScreen && !otherScreenHasFocus)
            {
                if (Keyboard.GetState().IsKeyDown(Keys.Escape))
                {
                    kinectRuntime.Uninitialize();
                    ExitScreen();
                }
                // variable time step but never less then 30 Hz
                World.Step(Math.Min((float)gameTime.ElapsedGameTime.TotalSeconds, (1f
/ 30f)));
                MouseState mouseState = Mouse.GetState();
```

```
                Vector2 mouse_pos = new Vector2(mouseState.X, mouseState.Y);

                if (credits == 0)
                {
                    this.ExitScreen();
                }

                if (mouseState.LeftButton == ButtonState.Pressed && credits > 0 &&
(Math.Abs(mouse pos.X - 153) + Math.Abs(mouse pos.Y - 471))<200)
                {
                    float angle = (float)Math.Atan2(spot.Y - 88, spot.X + 490);
                    if (angle > 0)
                        angle_text = -((angle / 3.14f * 180 - 180));
                    else if (angle < 0)
                        angle text = -((angle / 3.14f * 180 + 180));

                    _agent.Body.Rotation = 0f;
                    _agent.Body.Position = new Vector2(mouseState.X / 25.0f,
mouseState.Y / 25.0f) - new Vector2(640.0f / 25.0f, 384.0f / 25.0f) + new Vector2(-
2.5f, 0);      //div 25
                     agent.Body.LinearVelocity = 8.0f * (- agent.Body.Position + new
Vector2(-500.0f / 25.0f, 95.0f / 25.0f) - Vector2.One);
                    if (!pressed)
                    {
                        credits--;
                        if ( score >= 20 )
                            score -= 20;
                        pressed = true;
                    }
                    spot = (_agent.Body.Position - new Vector2(-2.5f, 0)) * 25f;
                    //score = mouse pos.X;
                    collided = false;
                    //score++;
                }
                else
                {
                    pressed = false;

                    angle text = 0;
                    spot = new Vector2(-487, 93);
                }

            }
            else
            {
                World.Step(0f);
            }

            Camera.Update(gameTime);
            base.Update(gameTime, otherScreenHasFocus, coveredByOtherScreen);
        }



        void DrawLine(SpriteBatch batch, Texture2D blank, float width, Color color,
Vector2 point1, Vector2 point2)
        {
            float angle = (float)Math.Atan2(point2.Y - point1.Y, point2.X - point1.X);
            float length = Vector2.Distance(point1, point2);

            batch.Draw(blank, point1, null, color, angle, Vector2.Zero, new
Vector2(length, width), SpriteEffects.None, 0);
        }



        private void VideoFrameReady(object sender, ImageFrameReadyEventArgs e)
        {
            PlanarImage image1 = e.ImageFrame.Image;
            kinectRGBVideo.Texture = image1.ToTexture2D(ScreenManager.GraphicsDevice);
        }


        private void SkeletonFrameReady(object sender, SkeletonFrameReadyEventArgs e)
        {
            SkeletonFrame skeletonFrame = e.SkeletonFrame;
```

```
            foreach (SkeletonData data in skeletonFrame.Skeletons)
            {
                if (data.TrackingState == SkeletonTrackingState.Tracked)
                {
                    Vector2 jointPosition_Left_Hand = new Vector2(0, 0);
                    Vector2 jointPosition_Right_Hand = new Vector2(0, 0);
                    Vector2 kinect_right, kinect_left;



                    foreach (Microsoft.Research.Kinect.Nui.Joint joint in data.Joints)
                    {


                        if (joint.ID == JointID.HandLeft)
                        {
                            jointPosition_Left_Hand =
joint.GetScreenPosition(kinectRuntime, ScreenManager.GraphicsDevice.Viewport.Width,
ScreenManager.GraphicsDevice.Viewport.Height);
                            //joint.
                        }



                        if (joint.ID == JointID.HandRight)
                        {
                            jointPosition Right Hand =
joint.GetScreenPosition(kinectRuntime, ScreenManager.GraphicsDevice.Viewport.Width,
ScreenManager.GraphicsDevice.Viewport.Height);
                        }

                        kinect right = jointPosition Right Hand / 5;
                        kinect left = jointPosition Left Hand / 5;



                        if ((float)Math.Abs(-kinect left.X + kinect right.X) > 10.0 &&
(float)Math.Abs(kinect left.Y - kinect right.Y) > 10.0 && time<2000)
                        {

                            spot.X = -493 - kinect_right.X + kinect_left.X;
                            spot.Y = +93 + kinect_left.Y - kinect_right.Y;

                             agent.Body.LinearVelocity = Vector2.Zero;
                            _agent.Body.Rotation = 0;
                            _agent.Body.Position = spot / 25;
                            //_agent.Body.LinearVelocity = 8.0f * (-
_agent.Body.Position + new Vector2(-500.0f / 25.0f, 95.0f / 25.0f) - Vector2.One);



                            time++;
                            score = time;
                        }
                        else if ((float)Math.Abs(-kinect left.X + kinect right.X) >
10.0 && (float)Math.Abs(kinect_left.Y - kinect_right.Y) > 10.0 && time > 2000)
                        {
                            _agent.Body.LinearVelocity = 8.0f * (-spot / 25 + new
Vector2(-500.0f / 25.0f, 95.0f / 25.0f) - Vector2.One);
                        }
                        else
                        {
                            spot = new Vector2(-487, 93);
                            time = 0;
                        }
                    }
                }
            }
        }



        public override void Draw(GameTime gameTime)
        {
            ScreenManager.SpriteBatch.Begin(0, null, null, null, null, null,
Camera.View);
```

```
            ScreenManager.SpriteBatch.Draw(Background, new Vector2(-640, -384),
Color.White);
            for (int i = 0; i < 3; ++i)
            {
                ScreenManager.SpriteBatch.Draw(_obstacle.Texture,
ConvertUnits.ToDisplayUnits(_obstacles[i].Position),
                                               null,
                                               Color.White,  obstacles[i].Rotation,
 obstacle.Origin, 1f,
                                               SpriteEffects.None, 0f);
            }



            ScreenManager.SpriteBatch.Draw(manBody.Texture,
ConvertUnits.ToDisplayUnits(man.Position), null,
                                           Color.White, man.Rotation, manBody.Origin,
_scale, SpriteEffects.None, 0f);
            mac nuts.Draw();
            haz nuts.Draw();
             agent.Draw();

            ScreenManager.SpriteBatch.Draw(ui, new Vector2(-640.0f, -379.0f),
Color.White);

            DrawLine(ScreenManager.SpriteBatch, dot, 4.0f, new Color(57, 20, 0), new
Vector2(-474, +88), spot);
            DrawLine(ScreenManager.SpriteBatch, dot, 4.0f, new Color(57, 20, 0), new
Vector2(-489, +95), spot);

            DrawLine(ScreenManager.SpriteBatch, dot, 4.0f, new Color(255, 0, 0, 0.4f),
new Vector2(-487, +93) + new Vector2(-spot.X-487,-spot.Y+93), new Vector2(-487, +93));
            DrawLine(ScreenManager.SpriteBatch, dot, 4.0f, new Color(0, 0, 255, 0.4f),
new Vector2(-487, +93) + new Vector2(-spot.X - 487, 0), new Vector2(-487, +93));

            ScreenManager.SpriteBatch.DrawString(font, "Score: " + score.ToString(),
new Vector2(-621.0f, -371.0f), Color.Black);
            ScreenManager.SpriteBatch.DrawString(font, "Score: " + score.ToString(),
new Vector2(-620.0f, -370.0f), new Color(255, 222, 0));

            ScreenManager.SpriteBatch.DrawString(font, "     " + credits.ToString(),
new Vector2(-601.0f, -337.0f), Color.Black);
            ScreenManager.SpriteBatch.DrawString(font, "     " + credits.ToString(),
new Vector2(-600.0f, -336.0f), new Color(255, 222, 0));

            ScreenManager.SpriteBatch.Draw(stone, new Vector2(-500.0f, -343.0f),
Color.White);
            ScreenManager.SpriteBatch.Draw(macnut, new Vector2(480.0f, -379.0f),
Color.White);
            ScreenManager.SpriteBatch.Draw(haznut, new Vector2(480.0f, -343.0f),
Color.White);

            ScreenManager.SpriteBatch.DrawString(font, "100 Pts", new Vector2(521.0f,
-379.0f), Color.Black);
            ScreenManager.SpriteBatch.DrawString(font, "100 Pts", new Vector2(520.0f,
-378.0f), new Color(40, 10, 30));

            ScreenManager.SpriteBatch.DrawString(font, "50 Pts", new Vector2(538.0f, -
343.0f), Color.Black);
            ScreenManager.SpriteBatch.DrawString(font, "50 Pts", new Vector2(537.0f, -
342.0f), new Color(40, 10, 30));


            ScreenManager.SpriteBatch.DrawString(font serious, "Angle: " +
angle text.ToString(), new Vector2(-455.0f, 60.0f), Color.Blue);

            ScreenManager.SpriteBatch.End();

            base.Draw(gameTime);
        }


    }
}
```

Ο παραπάνω κώδικας συνθέτει την κλάση Prehistoric_Nuts1η οποία αποτελεί τον βασικό διαχειριστή κάθε αντικειμένου το οποίο δύναται να αλληλεπιδρά μέσα στο περιβάλλον του παιχνιδιού με το υπόλοιπο σύστημα βάσει των φυσικών νόμων που ορίζει το περιβάλλον. Είναι ένα μακροσκελές τμήμα κώδικα, αλλά παρατίθεται σχεδόν ολόκληρο μιας και η σημασία του κατά την υλοποίηση των μηχανισμών φυσικής αποτέλεσε την επιτυχή ολοκλήρωση του μεγαλύτερου μέρους της λειτουργίας του μηχανισμού. Τα στοιχεία που περιγράφονται είναι τα εξής:

i. Συνάρτηση **LoadContent**()

   Η συνάρτηση LoadContent δεν παίρνει ως όρισμα κάποια παράμετρο και δεν επιστρέφει κάποια τιμή. Η συνάρτηση αυτή είναι υπεύθυνη για τη φόρτωση των απαραίτητων δεδομένων περιεχομένου σε κάποιες μεταβλητές καθώς και για την αρχικοποίηση όλων των μεταβλητών γενικότερα που χρειάζονται κατά την εκτέλεση των αλγορίθμων. Εφόσον δηλαδή τα αρχεία που θέλουμε να φορτώσουμε δηλωθούν στο πρόγραμμα τότε μέσω της LoadContent γίνεται η φόρτωσή τους μέσα στον αλγόριθμο. Επιπρόσθετα, στη συνάρτηση αυτή δίνεται και η εντολή για την εκκίνηση της λειτουργίας του Microsoft Kinect.

ii. Συνάρτηση **Update**(GameTime, bool otherScreenHasFocus, bool coveredByOtherScreen)

   Η συνάρτηση Update παίρνει τρεις παραμέτρους και δεν επιστρέφει τιμή. Η συνάρτηση αυτή είναι υπεύθυνη για την ανανέωση των δεδομένων και της εμφάνισης του παιχνιδιού. Εκτελείται όταν η εστίαση του παιχνιδιού είναι πάνω στην κλάση GameScreen της κλάσης που ανήκει και δεν καλύπτεται από άλλο οπτικό περιβάλλον του παιχνιδιού. Επιπρόσθετα, ορίζεται η ροή των καρέ και η ταχύτητα του παιχνιδιού στον φυσικό κόσμο του επιπέδου.

iii. Συνάρτηση **Draw**(GameTime)

   Η συνάρτηση Draw παίρνει ως παράμετρο μία μεταβλητή τύπου GameTime και δεν επιστρέφει τιμή. Η συνάρτηση αυτή είναι υπεύθυνη για τη σχεδίαση των γραφικών και της εμφάνισης του παιχνιδιού. Με τη βοήθεια του ScreenManager.SpriteBatch αποκτά τη δυνατότητα σχεδιασμού γραφικών ψηφίδων στο δισδιάστατο χώρο ορίζοντάς τους θέση, χρώμα, διαφάνεια και γενικότερα ρυθμίσεις που είναι απαραίτητες για τη ρύθμιση της εμφάνισης των αντικειμένων και των στοιχείων στο χώρο.

iv. Συνάρτηση **SkeletonFrameReady**(object sender, SkeletonFrameReadyEventArgs e)

Η συνάρτηση αυτή υλοποιεί την αλληλεπίδραση του Kinect με τον παίκτη. Μέσω των εργαλείων του SDK εντοπίζεται ο ανθρώπινος σκελετός με τη βοήθεια των αισθητήρων. Κατόπιν, με τη συνάρτηση SkeletonFrameReady λαμβάνουμε τα δεδομένα που επιθυμούμε από την παρακολούθηση και τα εισάγουμε στον αλγόριθμο για την είσοδο των κινήσεων του χρήστη ώστε να σημαδέψει με σωστή γωνία και δύναμη για το που θα εκσφενδονίσει την πέτρα.

v. Συνάρτηση **LoadObtstacles**()

Η δυσκολία της στόχευσης των καρπών έγκειται στο ότι υπάρχουν φυσικά εμπόδια στο χώρο, που κάνουν τη συλλογή των καρπών πιο δύσκολη. Με τη συνάρτηση LoadObstacles φορτώνουμε με στατικά εμπόδια ορθογώνιου σχήματος το χώρο, επιλέγοντας τον αριθμό τους, τις διαστάσεις τους, τη θέση του και την εμφάνισή τους. Δεν παίρνει κάποια παράμετρο και δε επιστρέφει τιμή.

vi. Συνάρτηση **OnCollisionEnter**()

Η συνάρτηση OnCollisionEnter είναι η συνάρτηση στην οποία οφείλεται η πραγματοποίηση της κρούσης και της συλλογής των πόντων από τους καρπούς. Εντοπίζοντας την επαφή μεταξύ της πέτρας και του καρπού, δίνεται η εντολή μέσω αυτής της συνάρτησης να μεταβεί στην επόμενη γραφιστική κατάσταση έως ότου συλλεχθεί ο καρπός. Τέλος, κατά τη συλλογή των καρπών προσθέτει τους πόντους στους συνολικούς.

Για περισσότερες λεπτομέρειες για τον τρόπο δήλωσης των μεταβλητών αλλά και την χρήση των απαραίτητων συναρτήσεων των αντίστοιχων βιβλιοθηκών για την υλοποίηση του αλγορίθμου σε γλώσσα προγραμματισμού C# θα ήταν καλό να γίνει λεπτομερής μελέτη του κώδικα που παρατίθεται. Σε πολλά σημεία αναγράφονται εκτενή σχόλια πριν την κάθε λειτουργία ώστε να είναι κατανοητή η μεθοδολογία που ακολουθήθηκε.

6.2 Πλατφόρμες και προγραμματιστικά εργαλεία

Η υλοποίηση αυτή πραγματοποιήθηκε στο περιβάλλον των Windows 7, χρησιμοποιώντας το Microsoft Visual Studio 2010 σε γλώσσα προγραμματισμού C#. Πρόσθετα εργαλεία που χρειάστηκαν για την κατασκευή του πλήρους λογισμικού ήταν το Microsoft Kinect SDK και XNA Framework, τα οποία ενσωματώθηκαν στο Visual Studio.

- **Microsoft Visual Studio 2010**

  Το Microsoft Visual Studio είναι ένα ισχυρό Ολοκληρωμένο Περιβάλλον Ανάπτυξης (IDE) το οποίο εξασφαλίζει την ποιότητα του κώδικα των εφαρμογών κατά την εξέλιξη της τεχνολογίας από το σχεδιασμό έως την εγκατάσταση. Δίνει τη δυνατότητα ανάπτυξης εφαρμογών σε Windows, Windows Phone, Web και άλλες.

- **Microsoft Kinect SDK**

  Το SDK του Kinect για Windows προσφέρει ένα πρόγραμμα εγκατάστασης, το οποίο καθιστά εύκολη την εγκατάσταση του Kinect και δίνει χρήσιμα εργαλεία για την ανάπτυξη εφαρμογών που το αναγνωρίζουν και εκμεταλλεύονται τις λειτουργίες των αισθητήρων του.

  Μία από τις λειτουργίες των εργαλείων του SDK είναι η άριστη παρακολούθηση του ανθρώπινου σκελετού, η οποία επιτρέπει στους προγραμματιστές τον έλεγχο του χρήστη που παρακολουθείται από τον αισθητήρα. Επιπλέον, υπάρχουν χρήσιμα εργαλεία για την αναγνώριση φωνής που μαζί με το βελτιωμένο σύστημα των ηχητικών αισθητήρων δίνουν μεγάλη ακρίβεια στην αναγνώριση ομιλίας.

- **Microsoft XNA Framework**

  Το XNA είναι βασισμένο σε γνωστά χαμηλού επιπέδου APIs όπως το Direct3D 9, το XACT, το XInput και το XContent. Περιλαμβάνει βιβλιοθήκες που ομαδοποιούνται σε κατηγορίες όπως γραφικών, ήχου, εισόδου, μαθηματικών και αποθήκευσης. Στην ουσία είναι ένα πακέτο εργαλείων που διευκολύνει την ανάπτυξη παιχνιδιών για ηλεκτρονικούς υπολογιστές και για Xbox 360.

Το λογισμικό είναι συμβατό με τα Windows 7 και το Xbox 360, όπου μπορεί να εκτελεστεί κανονικά και να λειτουργήσει με τον αισθητήρα του Kinect. Αναγκαία είναι η εγκατάσταση της συσκευής στον ηλεκτρονικό υπολογιστή ώστε να είναι εφικτή η λειτουργία του.

Το λογισμικό διατίθεται σε CD που περιλαμβάνει οδηγό εγκατάστασης (Setup Wizard) για να πραγματοποιείται η προσθαφαίρεση του προγράμματος στο λειτουργικό σύστημα των Windows.

7

Επίλογος

Φτάνοντας στον επίλογο της παρούσας διπλωματικής εργασίας, ακολουθεί μία σύνοψη του έργου που εκτελέστηκε καθώς και του τελικού λογισμικού που παράχθηκε, κάνοντας μια περιεκτική περιγραφή της εφαρμογής, των δυνατοτήτων της αλλά και των λειτουργιών που προσφέρει, δίνοντας έμφαση στους στόχους που επετεύχθησαν και τους σκοπούς που εξυπηρετούνται μέσω αυτής.

7.1 Σύνοψη και συμπεράσματα

Στόχος της παρούσας διπλωματικής εργασίας ήταν η ανάπτυξη ενός διαδραστικού παιχνιδιού φυσικής αλληλεπίδρασης μεταξύ ανθρώπου και υπολογιστικής μηχανής. Κατασκευάστηκε λοιπόν ένα λογισμικό συμβατό με Windows και Xbox 360, χρησιμοποιώντας το XNA, το οποίο προσφέρει λειτουργίες ψυχαγωγικού και επιμορφωτικού χαρακτήρα προς το χρήστη.

Ένα λογισμικό κρίνεται χρήσιμο όταν καταφέρνει να είναι εύχρηστο, λειτουργικό και προσφέρει καινοτομίες που δεν υπάρχουν σε άλλες εφαρμογές της κατηγορίας του ή τουλάχιστον σε πολλές από αυτές. Σημαντικό επίσης στοιχείο είναι το κάθε λογισμικό να επιτυγχάνει το σκοπό για τον οποίο έχει κατασκευαστεί και να περιέχει τουλάχιστον τα βασικά χαρακτηριστικά και εργαλεία της κατηγορίας λογισμικού στην οποία ανήκει.

Εν ολίγοις, αναπτύχθηκε ένα λογισμικό ψυχαγωγικού και επιμορφωτικού χαρακτήρα, προσφέροντας στο χρήστη καινοτομικές μεθόδους αλληλεπίδρασής του με τον υπολογιστή ή την κονσόλα. Αυτό επιτυγχάνεται προσφέροντας μια πρωτοποριακή αίσθηση ψυχαγωγίας μέσα σε αυτό ασκώντας φυσικές κινήσεις με το σώμα. Ταυτόχρονα συμβάλει, ως προς το επιμορφωτικό κομμάτι, στην εξοικείωση του παίκτη με ορισμένα φυσικά φαινόμενα και φυσικές αλληλεπιδράσεις μέσα σε ένα ψηφιακό περιβάλλον που προσομοιώνει το φυσικό, προβεβλημένο όμως σε δισδιάστατο και όχι τρισδιάστατο χώρο. Επιπροσθέτως, τον βοηθά στο να ελέγχει και να εντοπίζει μέσω αυτού την φυσική αλληλεπίδραση μεταξύ των σωμάτων

σύμφωνα με τα φυσικά τους χαρακτηριστικά, όπως η μάζα, το σχήμα, τα σημεία επαφής, τις τριβές.

Συνοπτικά, έγινε μία σύνθεση διαφόρων μηχανισμών ώστε να επιτευχθεί το τελικό αποτέλεσμα. Ανακεφαλαιώνοντας τα όσα έχουμε αναφέρει στον κύριο κορμό της παρούσας εργασίας, η εφαρμογή ολοκληρώθηκε με τα εξής βήματα:

- Κατασκευάστηκε ένας βασικός κορμός γραφικού δισδιάστατου περιβάλλοντος το οποίο αποτέλεσε τη βάση πάνω στην οποία χτίστηκε το όλο περιβάλλον της εφαρμογής.
- Κατασκευάστηκαν αλγόριθμοι για τη σωστή λειτουργία μιας μηχανής φυσικής που ενσωματώθηκε στο βασικό γραφικό περιβάλλον, ώστε να πραγματοποιείται φυσική αλληλεπίδραση των αντικειμένων μέσα στο περιβάλλον αυτό.
- Κατασκευάστηκαν αλγόριθμοι για τη εντοπισμό βασικών λειτουργιών της φυσικής των αντικειμένων που ενσωματώθηκαν στο βασικό γραφικό περιβάλλον, ώστε να δίδεται επιπλέον διδακτικός χαρακτήρας στα φυσικά φαινόμενα μεταξύ των αντικειμένων.
- Αναπτύχθηκε μηχανισμός αλληλεπίδρασης ανθρώπινου σκελετού με το παιχνίδι μέσω του αισθητήρα Microsoft Kinect και κατασκευάστηκε αλγόριθμος για το σύστημα ελέγχου του παιχνιδιού μέσω αυτού.
- Αναπτύχθηκε ένα απλό σύστημα που προσέδωσε τα βασικά χαρακτηριστικά που συνθέτουν ένα ηλεκτρονικό παιχνίδι και δίνουν το απαραίτητο ενδιαφέρον σε αυτό προσθέτοντας κάποια στατιστικά ώστε να γίνεται ελαφρώς ανταγωνιστικό.

Τέλος, με την κατασκευή του λογισμικού αυτού γίνεται φανερό το γεγονός ότι η τεχνολογία (ανάπτυξης λογισμικού και η τεχνολογία αισθητήρων) πλέον βρίσκονται σε σημείο ικανό για την παραγωγή νέων μεθόδων επικοινωνίας μεταξύ του ανθρώπου και των μηχανών. Εκμεταλλευόμενος κανείς αυτά τα εργαλεία, μπορεί να αντικαταστήσει τις μέχρι τώρα διαδεδομένες συσκευές και τους τετριμμένους υπάρχοντες τρόπους επικοινωνίας με τα υπολογιστικά συστήματα και να περάσει σε πιο σύγχρονες μεθόδους που θα έδιναν άλλη πνοή στα τεχνολογικά δρώμενα. Άλλωστε, δε θα μπορούσε να θεωρηθεί αμελητέα η χρήση τέτοιων μεθόδων και σε πιο πρακτικούς τομείς της επιστήμης όπως η ιατρική, αλλά και σε άλλες δραστηριότητες όπως η φυσιοθεραπεία ή η επικοινωνία με νοηματική, η απλή γυμναστική και πολλούς ακόμη τομείς.

7.2 Μελλοντικές επεκτάσεις

Προφανώς η ζωτικότητα ενός λογισμικού έγκειται στη συχνή ανανέωση του και επέκτασή του από τους δημιουργούς του. Λόγω της πληθώρας των δυνατοτήτων που προσφέρουν τα εργαλεία που χρησιμοποιήθηκαν κατά την κατασκευή του διαδραστικού αυτού παιχνιδιού, αναμφίβολα είναι εφικτή κάθε είδους προσθήκη και εξέλιξη του λογισμικού.

Μια αρχική σκέψη είναι η εκμετάλλευση των πολλαπλών μικροφώνων που διαθέτει το Kinect για την προσθήκη επιπλέον λειτουργιών όπως η επιλογή χρήστη κατόπιν φωνητικής εντολής, μέσω φωνητικής αναγνώρισης, και η επιλογή επιπλέον αντικειμένων εκτόξευσης επίσης με φωνητικές εντολές.

Επίσης, ενδεχόμενη προσθήκη είναι η ανάπτυξη επιπλέον μηχανισμών αναγνώρισης χειρονομιών για την επιλογή αντικειμένων που θα εισάγονται εντός του περιβάλλοντος και θα προσθέτουν την ύπαρξη νέων φυσικών φαινομένων μέσα στον υπάρχοντα χώρο, όπως ελατήρια, αντικείμενα μηδενικών τριβών, βλήματα και διάφορα άλλα.

8

Βιβλιογραφία

[OK2011] Orland, Kyle (February 21, 2011). "News - Microsoft Announces Windows Kinect SDK For Spring Release". Gamasutra. Retrieved March 16, 2011.

[TD2009] Takahashi, Dean (September 5, 2009), "How many vendors does it take to make Microsoft's Project Natal game control system?

[DJ2009] Dunham, Jeremy (June 2, 2009). "E3 2009: I've Played Natal and it Works". IGN. Retrieved March 18, 2011.

[WMBM09] Wilson, Mark; Buchanan, Matt (June 3, 2009). "Testing Project Natal: We Touched the Intangible"

[CC2009] Carter, Chad (March 7, 2009). Microsoft XNA Game Studio 3.0 Unleashed (1st ed.). Sams. pp. 792

[RA2011] Reed, Aaron (2011). "Learning XNA 4.0"

[TC2002] Ted, Coombs (2002). "Programming with C#.NET"

[PS2006] Paul, Schuytema (2006). "Game Design"