Πανεπιστήμιο Πειραιώς – Τμήμα Πληροφορικής

Πρόγραμμα Μεταπτυχιακών Σπουδών

«Προηγμένα Συστήματα Πληροφορικής»

Μεταπτυχιακή Διατριβή

Τίτλος Διατριβής	**Υλοποίηση Πυραμίδας των Αναγκών του Μάσλοου σε Εικονικό Περιβάλλον** **Implementation of Maslow's Pyramid of Needs in Virtual Environment**
Ονοματεπώνυμο Φοιτητή	**Αλεξάνδρα Χαρατσή**
Πατρώνυμο	**Μάρκος**
Αριθμός Μητρώου	**ΜΠΣΠ/17072**
Επιβλέπων	**Θεμιστοκλής Παναγιωτόπουλος, Καθηγητής**

Ημερομηνία Παράδοσης **Σεπτέμβριος 2020**

Τριμελής Εξεταστική Επιτροπή

(υπογραφή)	(υπογραφή)	(υπογραφή)
Θεμιστοκλής Παναγιωτόπουλος Καθηγητής	Άγγελος Πικράκης Επίκουρος Καθηγητής	Δημήτριος Αποστόλου Αναπληρωτής Καθηγητής

Πίνακας περιεχομένων

ΠΙΝΑΚΑΣ ΕΙΚΟΝΩΝ

1. ΠΕΡΙΛΗΨΗ

Η συγκεκριμένη εφαρμογή εικονικής πραγματικότητας, είναι μια Μεταπτυχιακή Διατριβή στα πλαίσια του Μεταπτυχιακού Προγράμματος Σπουδών Πληροφορικής στο τμήμα Προηγμένων Συστημάτων για την κατεύθυνση Ευφυή Συστήματα Επικοινωνίας Ανθρώπου-Υπολογιστή του Πανεπιστημίου Πειραιώς.

Η ραγδαία ανάπτυξη παιχνιδιών και η ενασχόληση όλο και περισσότερων ατόμων με τον τομέα της Εικονικής Πραγματικότητας και Τεχνητής Νοημοσύνης, έχουν οδηγήσει σε καινοτόμα αποτελέσματα, κάνοντας μεγάλα βήματα στην εξέλιξη του κλάδου. Συνδυαστικά με το γεγονός ότι ορισμένες πλατφόρμες παρέχονται δωρεάν και ο κόσμος μπορεί να βρει ελεύθερα υλικό για να ξεκινήσει τον προγραμματισμό και την ανάπτυξη παιχνιδιών και προσομοιώσεων, τα παραπάνω αποτελέσματα συγκλίνουν πολύ στην πραγματικότητα.

Στην παρούσα εργασία γίνεται μια προσπάθεια δημιουργίας μιας μηχανής καταστάσεων που να αναγνωρίζει αυτόματα ποια είναι η ανάγκη που έχει προτεραιότητα να εκπληρωθεί από ένα χαρακτήρα, ο οποίος έχει ένα βασικό στόχο που πρέπει να εκπληρώσει για να τελειώσει η προσομοίωση. Όλα αυτά γίνονται με βάση τα τρία πρώτα επίπεδα της Πυραμίδας του Μάσλοου. Ο χαρακτήρας πρέπει να εργάζεται, να ικανοποιεί κάποιες βασικές του ανάγκες και παράλληλα να βρίσκει χρόνο για να διαβάζει και να γίνει αυτό που πάντα ήθελε.

Στην παρούσα εργασία επιχειρείται η παρουσίαση και ανάπτυξη ενός απλού μοντέλου της Πυραμίδας των Αναγκών του Μάσλοου. Οι πράξεις που γίνονται είναι ευθύνη του χαρακτήρα και όχι κανενός χειριστή ή παίκτη. Οι βασικές ανάγκες, καθώς και οι ανάγκες για διάβασμα και μελέτη εκπληρώνονται με βάση την επιθυμία του χαρακτήρα, ενώ η εργασία στην οποία πηγαίνει καθημερινά είναι βασισμένη στην ώρα του παιχνιδιού.

2. ABSTRACT

This virtual reality application is a Master's Thesis of the Postgraduate Program in Informatics, in the Department of Advanced Systems for Intelligent Human-Computer Communication Systems of the University of Piraeus.

The rapid development of games and the involvement of more and more people in the field of Virtual Reality and Artificial Intelligence, have led to innovative results, making great strides in the development of the industry. Combined with the fact that some platforms are provided for free and people can easily find material to start programming and developing games and simulations, the above results converge a lot in reality.

In the present work, an attempt is made to create a state machine that automatically recognizes what is the priority need to be met by a character, who has a key goal to fulfill in order to complete the simulation. All this is based on the first three levels of the Maslow's Pyramid. The character must work, satisfy some of his basic needs and at the same time, find the necessary time to study and become what he always wanted.

In the present work, the presentation and development of a simple model of Maslow's Pyramid of Needs is attempted. The actions performed are the responsibility of the character and not of any operator or player. The basic needs as well as the need to study are fulfilled based on the desire of the character, while the work he goes to every day is based on the time of the game.

3. ΕΙΣΑΓΩΓΗ

Παρακάτω θα αναλυθούν κάποιες γενικές πληροφορίες και θα χτιστεί το υπόβαθρο για την καλύτερη κατανόηση της συγκεκριμένης διπλωματικής εργασίας.

3.1. ΓΕΝΙΚΑ

Τα τελευταία χρόνια παρατηρείται αύξηση στην τεχνολογία δημιουργίας παιχνιδιών. Η ελεύθερη διανομή της πλατφόρμας Unity 3D, τα αμέτρητα assets που διανέμονται δωρεάν από χρήστες ή γραφίστες καθώς και η δωρεάν εκπαίδευση που παρέχει η ίδια η εταιρία, είναι αυτά που ωθούν πολύ κόσμο στη σύλληψη νέων ιδεών, στη δημιουργία δικών του παιχνιδιών, και κατ' επέκταση στην ανακάλυψη νέων εφαρμογών στον τομέα τεχνητής νοημοσύνης. Παράλληλα, η πλατφόρμα επιτρέπει την αναπαράσταση εικονικών κόσμων, οπότε ο χρήστης έχει μια συνολική εικόνα για το πώς ήταν, είναι ή θα είναι κοινωνίες και πολιτισμοί στο παρελθόν, το παρόν και το μέλλον αντίστοιχα.

Πώς επιτυγχάνεται όμως η αληθοφάνεια ενός εικονικού περιβάλλοντος; Οι τεχνολογίες καταγραφής κίνησης (motion capture) βοηθούν τα παραπάνω να εξελιχθούν ακόμα περισσότερο και ο δέκτης να βιώσει μια καθολική αναπαράσταση του εικονικού περιβάλλοντος. Με αυτό τον τρόπο, μπορεί ο προγραμματιστής να κάνει ένα χαρακτήρα να αποκτήσει ζωή. Για παράδειγμα αντί να κάθεται απόλυτα ακίνητος, μπορεί να τον κάνει να κινείται με τέτοιο τρόπο, που να φαίνεται πως αναπνέει. Ένας άλλος τρόπος που θα μπορούσε να επιτευχθεί αληθοφάνεια, είναι με τη χρήση ήχων όταν ο χαρακτήρας κάνει κάποια δραστηριότητα. Για παράδειγμα αν ο χαρακτήρας κόβει ξύλα, θα μπορούσε να χρησιμοποιηθεί ένα ηχητικό εφέ που να αναπαράγει την κοπή ξύλου με ένα τσεκούρι. Ένας άλλος τρόπος, που εφαρμόστηκε και στην εργασία, είναι η χρήση μηνυμάτων, δηλαδή ένα πλαίσιο μέσα στο οποίο ο χαρακτήρας μεταφέρει τις σκέψεις του στο δέκτη. Με τους παραπάνω τρόπους χρησιμοποιούνται όλες οι δυνατές αισθήσεις του δέκτη που μπορούν να χρησιμοποιηθούν από την πλατφόρμα.

3.2. ΠΥΡΑΜΙΔΑ ΤΟΥ ΑΒΡΑΑΜ ΜΑΣΛΟΟΥ

Η συγκεκριμένη διατριβή έχει ως στόχο, ο δέκτης να ζήσει σε ένα βαθμό τη ζωή ενός φοιτητή που εργάζεται για να καλύψει τις βιοποριστικές του ανάγκες. Οι ανάγκες αυτές βασίζονται στο πρότυπο της Πυραμίδας του Αβραάμ Μάσλοου. Ήταν ένας Αμερικανός ψυχολόγος (1 Απριλίου 1908 - 8 Ιουνίου 1970) ο οποίος έμεινε γνωστός για τη θεωρία της ιεράρχησης των αναγκών. Υποστήριζε ότι υπάρχουν πέντε επίπεδα αναγκών από τα οποία εξαρτάται η συμπεριφορά ενός ατόμου και κατέληξε σε αυτά κάνοντας κλινικές έρευνες. (1) Από αυτές, προκύπτουν οι εξής ανάγκες:

- Οι φυσιολογικές.
- Οι ανάγκες ασφάλειας ή σιγουριάς.
- Οι κοινωνικές.
- Οι ανάγκες εκτίμησης ή αναγνώρισης.
- Οι ανάγκες ολοκλήρωσης.

Εικόνα 1 Πυραμίδα Αναγκών Maslow

Με βάση αυτή τη διατύπωση ο Μάσλοου διατύπωσε τρεις προτάσεις για να εξηγήσει την ανθρώπινη παρακίνηση.

Πρώτον, ο άνθρωπος προσπαθεί συνεχώς να ικανοποιήσει καλύτερα τις ανάγκες του και επιθυμεί συνεχώς και περισσότερα, από αυτά που ήδη διαθέτει.

Δεύτερον, όσο περισσότερο ικανοποιείται μια ανάγκη τόσο λιγότερο παρακινεί, μέχρι του σημείου που ικανοποιείται πλήρως και παύει να παρακινεί.

Τρίτον, η ιεράρχηση των αναγκών προκύπτει από την προτεραιότητά τους για ικανοποίηση που σημαίνει ότι ένας άνθρωπος προσπαθεί πρώτα να εξασφαλίσει την ανάγκη του να έχει πόσιμο νερό και στη συνέχεια να αγοράσει ένα ρούχο που του αρέσει πολύ. Βέβαια, δεν είναι πάντα ανάγκη να ικανοποιηθεί πλήρως μια ανάγκη για να προσπαθήσει ένας άνθρωπος να καλύψει την επόμενη. Χρειάζεται όμως να ικανοποιηθεί σε ένα στοιχειώδη βαθμό ο οποίος είναι ασφαλώς υποκειμενικός, ενός Αμερικανού ψυχολόγου, που υποστήριζε ότι υπάρχουν πέντε επίπεδα αναγκών, στα οποία βασίζεται η συμπεριφορά του ατόμου.

3.3. ΠΑΙΧΝΙΔΙΑ ΠΡΟΣΟΜΟΙΩΣΗΣ

Η επίσημη ονομασία τους είναι «παιχνίδια προσομοίωσης ζωής» και ο σκοπός τους είναι η διατήρηση και η ανάπτυξη ενός διαχειρίσιμου πληθυσμού οργανισμών, που οι παίκτες ελέγχουν τις ζωές και τις αποφάσεις αυτών των χαρακτήρων-ανθρώπων. Κάποιες μεγάλες κατηγορίες τέτοιων παιχνιδιών είναι η διαχείριση φυλών, τεχνητών κατοικίδιων ζώων ή προσομοίωσης επαγγελμάτων.

Η λειτουργία της προσομοίωσης, είναι μια προγραμματιστική διαδικασία που χρησιμοποιείται ευρέως και από άλλες κατηγορίες παιχνιδιών, όπως για παράδειγμα τα αθλητικά. Στο NBA 2K ή στο FIFA, υπάρχει η δυνατότητα προσομοίωσης ενός αγώνα, όπου με βάση τα πραγματικά στατιστικά της εκάστοτε ομάδας, την πορεία του χειριστή/παίκτη καθώς και ενός ποσοστού τυχαιότητας, βγαίνει ένα αποτέλεσμα που φαίνεται αληθοφανές.

Η εργασία είναι εμπνευσμένη από τη σειρά παιχνιδιών προσομοίωσης The Sims, που ήταν πολύ δημοφιλής τη δεκαετία του 2000. Η σειρά παιχνιδιών προσομοίωνε ανθρώπους (Sim) στην πραγματική τους ζωή, που κάλυπταν τις βιολογικές τους ανάγκες, δημιουργούσαν οικογένειες, δούλευαν σε έναν κλάδο που επέλεγε ο παίκτης, κοινωνικοποιούνταν με άλλους Sim και είχαν γενικότερα έναν ολοκληρωμένο κύκλο ζωής. Πλέον οι φανατικοί του είδους, παίζουν αντίστοιχα παιχνίδια στα έξυπνα κινητά τους, χωρίς να χρειάζεται να ξοδέψουν χρήματα για να αγοράσουν το συγκεκριμένο παιχνίδι.

Εικόνα 2 Στιγμιότυπο από Sims 2

Όπως είναι αντιληπτό και από τη διεπαφή χρήστη, ο κάθε Sim πρέπει να έχει τις ανάγκες του (Needs) ικανοποιημένες σε ένα βαθμό. Οι συγκεκριμένες μπάρες κατάστασης θα μπορούσαν να μεταφραστούν ως οι τρεις πρώτες βαθμίδες αναγκών της Πυραμίδας του Μάσλοου.

Η στήλη “Wants” αποτελούν τις δυο τελευταίες βαθμίδες, της αυτοεκτίμησης και αυτοπραγμάτωσης.

3.4. ΕΠΙΛΟΓΗ ΜΗΧΑΝΗΣ ΠΑΙΧΝΙΔΙΟΥ

Η μεγάλη ερώτηση στην αρχή της υλοποίησης, ήταν η εξής: Unity 3D ή Unreal Engine;

Και οι δυο πλατφόρμες μπορούν να παράγουν εξαιρετικής ποιότητας γραφικά, έχουν εκτεταμένη βιβλιοθήκη που περιλαμβάνει πολλά χαρακτηριστικά, όπως πρόγραμμα επεξεργασίας εδάφους, physics, animations και υποστήριξη VR. Τα γραφικά και η αίσθηση του παιχνιδιού δεν είναι πλέον αποτέλεσμα της πλατφόρμας, αλλά των προγραμματιστών, κάτι που πλέον είναι πάρα πολύ καλό, γιατί μπορεί μια σειρά παιχνιδιών για οποιονδήποτε λόγο να αλλάξει πλατφόρμα υλοποίησης παιχνιδιών και ο χαρακτήρας της εταιρίας να παραμείνει αναλλοίωτος.

Ο βασικός λόγος που προτιμήθηκε η Unity 3D για τη δημιουργία του συγκεκριμένου παιχνιδιού είναι η προηγούμενη εμπειρία που υπήρχε πάνω στη συγκεκριμένη πλατφόρμα. Η εναλλακτική επιλογή ήταν η Unreal Engine, η οποία δεν διατίθεται δωρεάν και έχει ελάχιστα δωρεάν assets διαθέσιμα. Η Unity έχει καταφέρει με την ελεύθερη διανομή της να φέρει πολύ κόσμο που αγαπά τη δημιουργία παιχνιδιών πιο κοντά στην επίτευξη του στόχου και των ονείρων τους. Η μεγάλη διαθεσιμότητα assets και το ευρύ φάσμα από εκπαιδευτικά βίντεο που υπάρχουν σε πολλές ιστοσελίδες, καθιστούν τη Unity την κυρίαρχη πλατφόρμα για νέους χρήστες ή για αυτούς που έχουν περιορισμένη εμπειρία.

3.5. ΥΠΟΒΑΘΡΟ ΚΑΙ ΙΣΤΟΡΙΑ

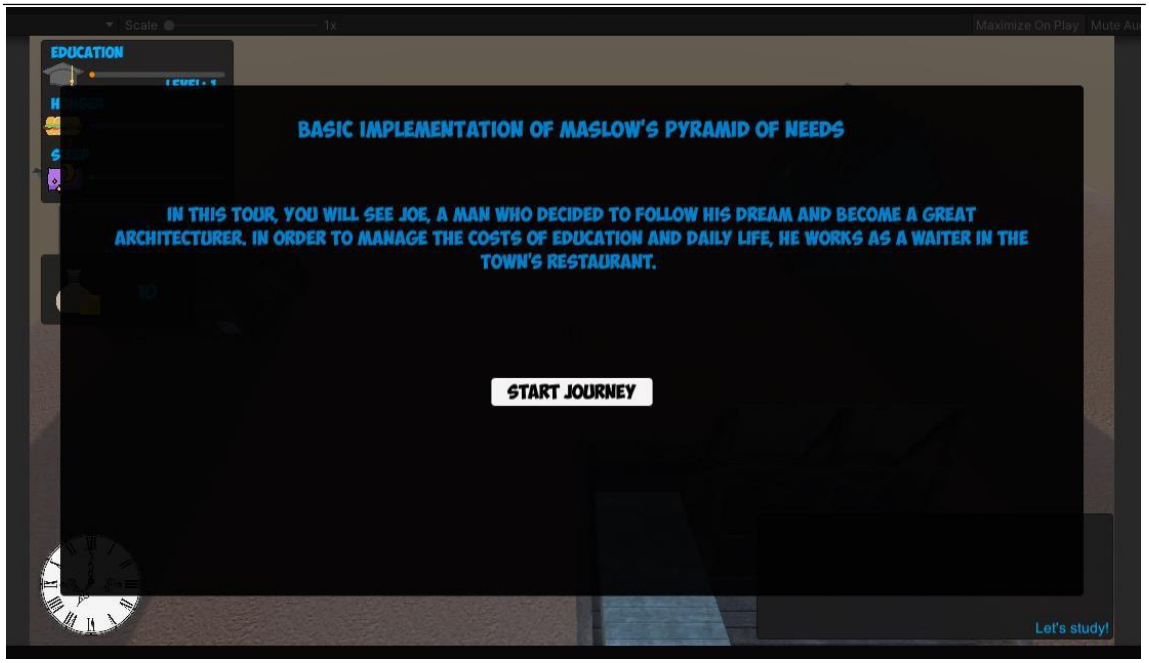
Η υλοποίηση αφορά ένα συνδυασμό μηχανισμών που καταφέρνουν να λειτουργούν αρμονικά και να προσομοιώνουν τη βασική καθημερινότητα ενός εργαζόμενου φοιτητή. Οι λειτουργίες που καλύπτονται είναι η φυσιολογική ανάγκη για τροφή και ύπνο. Ακολουθούν οι Ανάγκες Ασφάλειας και Προστασίας, όπου η εργασία καλύπτει την ανάγκη για ικανοποιητικό επίπεδο στέγης και για μόνιμη απασχόληση. Ο βασικός στόχος του NPC, είναι να καλύψει μια ανάγκη του τέταρτου επιπέδου της Πυραμίδας του Μάσλοου, την ανάγκη της αυτοεκτίμησης και της επίτευξης ενός στόχου. Η προαναφερόμενη καλύπτεται με την απόκτηση του πτυχίου Αρχιτεκτονικής. Αυτό είναι και το τέλος της προσομοίωσης, όπου φαίνονται και κάποια στατιστικά για το πώς κυμάνθηκαν οι μέρες της φοιτητικής ζωής του NPC.

4. ΥΛΟΠΟΙΗΣΗ

Στο συγκεκριμένο κεφάλαιο θα αναλυθεί ο κώδικας και η δομή του. Θα πραγματοποιηθεί ανάλυση των λειτουργιών του προγράμματος και η σύνδεση τους με την πραγματικότητα.

Η υλοποίηση αφορά ένα συνδυασμό μηχανισμών που καταφέρνουν να λειτουργούν αρμονικά και να προσομοιώνουν τη βασική καθημερινότητα ενός εργαζόμενου φοιτητή. Οι λειτουργίες που καλύπτονται είναι η φυσιολογική ανάγκη για τροφή και ύπνο. Ακολουθούν οι Ανάγκες Ασφάλειας και Προστασίας, όπου η εργασία καλύπτει την ανάγκη για ικανοποιητικό επίπεδο στέγης και για μόνιμη απασχόληση. Ο βασικός στόχος του NPC, είναι να καλύψει μια ανάγκη του τέταρτου επιπέδου της Πυραμίδας του Μάσλοου, την ανάγκη της αυτοεκτίμησης και της επίτευξης ενός στόχου. Η προαναφερόμενη καλύπτεται με την απόκτηση του πτυχίου Αρχιτεκτονικής. Αυτό είναι και το τέλος της προσομοίωσης, όπου φαίνονται και κάποια στατιστικά για το πώς κυμάνθηκαν οι μέρες της φοιτητικής ζωής του NPC.

4.1. ΕΙΣΑΓΩΓΙΚΟ ΠΑΝΕΛ



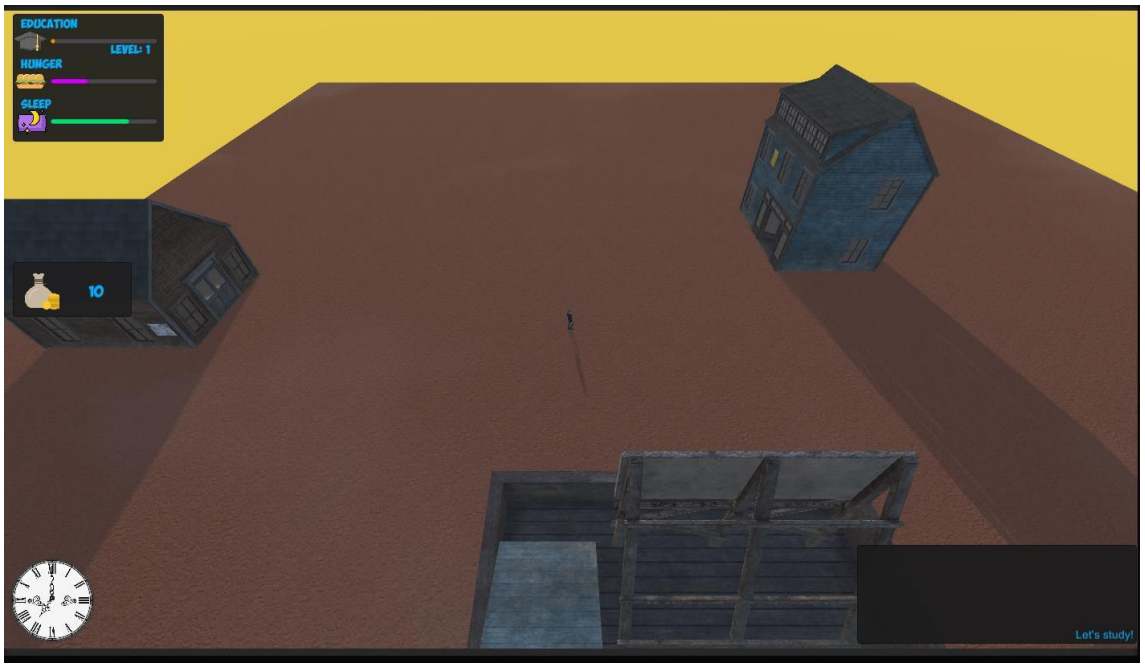
Εικόνα 3 Έναρξη Προσομοίωσης

Στην παραπάνω εικόνα φαίνεται η εκκίνηση της προσομοίωσης. Ο χρήστης λαμβάνει μια περιγραφή για το τι πρόκειται να δει κατά τη διάρκεια της εκτέλεσης του προγράμματος.

Η ιστορία που περιγράφεται, αφορά ένα φοιτητή που επιθυμεί να σπουδάσει Αρχιτεκτονική. Ο φοιτητής Joe, μένει σε ένα σπίτι.

Κάθε καινούργιο επίπεδο (level), σηματοδοτεί τη λήξη ενός εξαμήνου και την αρχή του επόμενου.

4.2. ΕΠΕΞΗΓΗΣΗ ΔΙΕΠΑΦΗΣ ΧΡΗΣΤΗ



Εικόνα 4 Έναρξη προσομοίωσης

Ο δέκτης, μπορεί να παρακολουθεί την κατάσταση στην οποία βρίσκεται ο χαρακτήρας μέσω της διεπαφής που υπάρχει κατά τη διάρκεια της προσομοίωσης.

Στην παρακάτω εικόνα βρίσκεται ο πίνακας με τις μπάρες που δείχνουν την τρέχουσα κατάσταση του χαρακτήρα. Στο παρακάτω παράδειγμα είναι εμφανές ότι ο Joe είναι στην αρχή της εκπαίδευσής του, «πεινάει» αλλά όχι αρκετά για να πάει για φαγητό και έχει κοιμηθεί αρκετά, οπότε δε χρειάζεται τη δεδομένη στιγμή να κοιμηθεί.

Εικόνα 5 Πίνακας με μπάρες κατάστασης

Στον ακριβώς από κάτω πίνακα από αυτόν με τις μπάρες, φαίνεται η οικονομική κατάσταση του χαρακτήρα. Στο παράδειγμα ο Joe διαθέτει δέκα νομίσματα.

Εικόνα 6 Πίνακας νομισμάτων

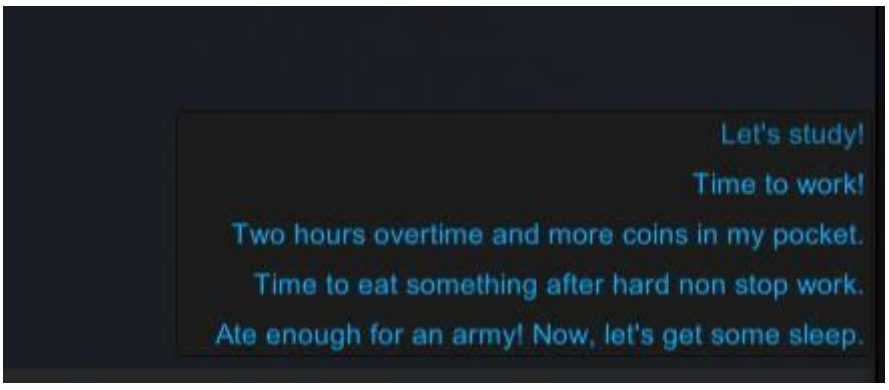
Κάτω αριστερά στην οθόνη, ο δέκτης μπορεί να βρει ένα ρολόι. Το ρολόι ξεκινάει στις 7 το πρωί σε κάθε έναρξη του παιχνιδιού και δείχνει την ώρα μέσα στο παιχνίδι.

Εικόνα 7 Ρολόι

Το παρακάτω παράθυρο εμφανίζεται μόνο όταν ο χαρακτήρας περάσει σε ένα καινούργιο επίπεδο. Εμφανίζεται για λίγα δευτερόλεπτα και περιγράφει τι έμαθε ο Joe μέσα στο εξάμηνο.

Εικόνα 8 Ειδοποίηση αύξησης επιπέδου

Παρακάτω βρίσκεται το παράθυρο των σκέψεων του χαρακτήρα. Κατά τη μετάβαση του σε κάποιο κτίριο, εμφανίζεται μια σκέψη που πιθανώς έχει. Υπάρχει και η περίπτωση ο χαρακτήρας να βρίσκεται σε κάποια άλλη κατάσταση, αλλά να εκφράζει την επιθυμία του για κάποια άλλη. Για παράδειγμα, μπορεί να βρίσκεται στη δουλειά και να αναφέρει ότι χρειάζεται να κοιμηθεί.



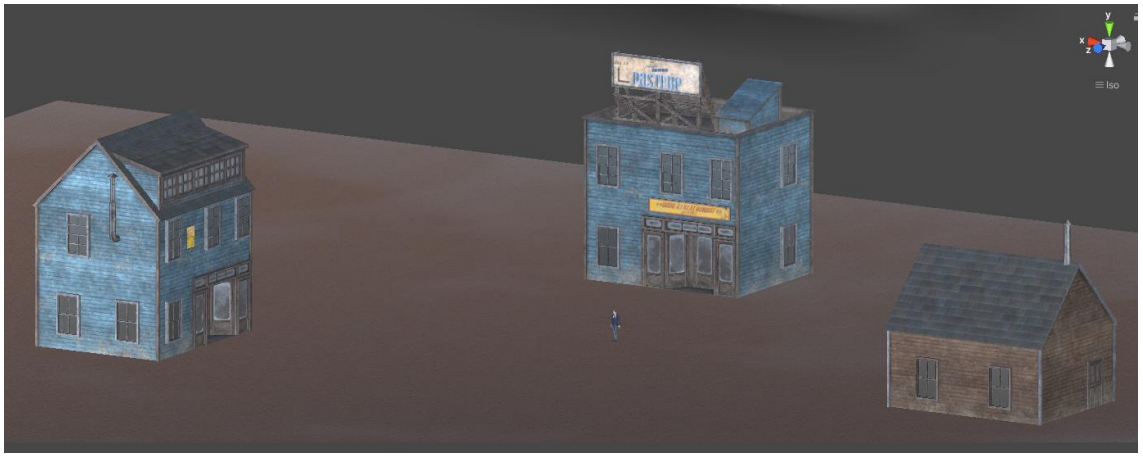
Εικόνα 9 Παράθυρο μηνυμάτων-σκέψεων

4.3. ΔΟΜΗ ΚΩΔΙΚΑ ΚΑΙ ΑΝΑΛΥΣΗ

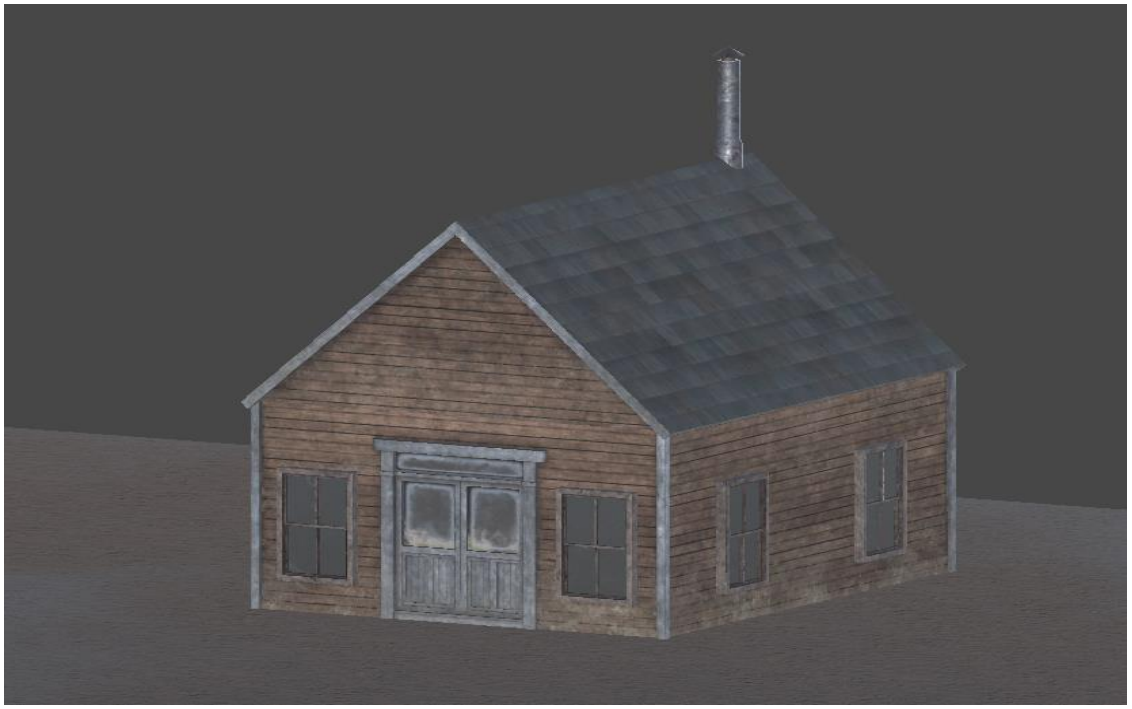
Παρακάτω αναλύεται ο κώδικας και οι λειτουργίες που διαθέτει η συγκεκριμένη διπλωματική εργασία.

4.3.1. ΥΛΟΠΟΙΗΣΗ ΕΔΑΦΟΥΣ ΚΑΙ ΚΤΙΡΙΩΝ

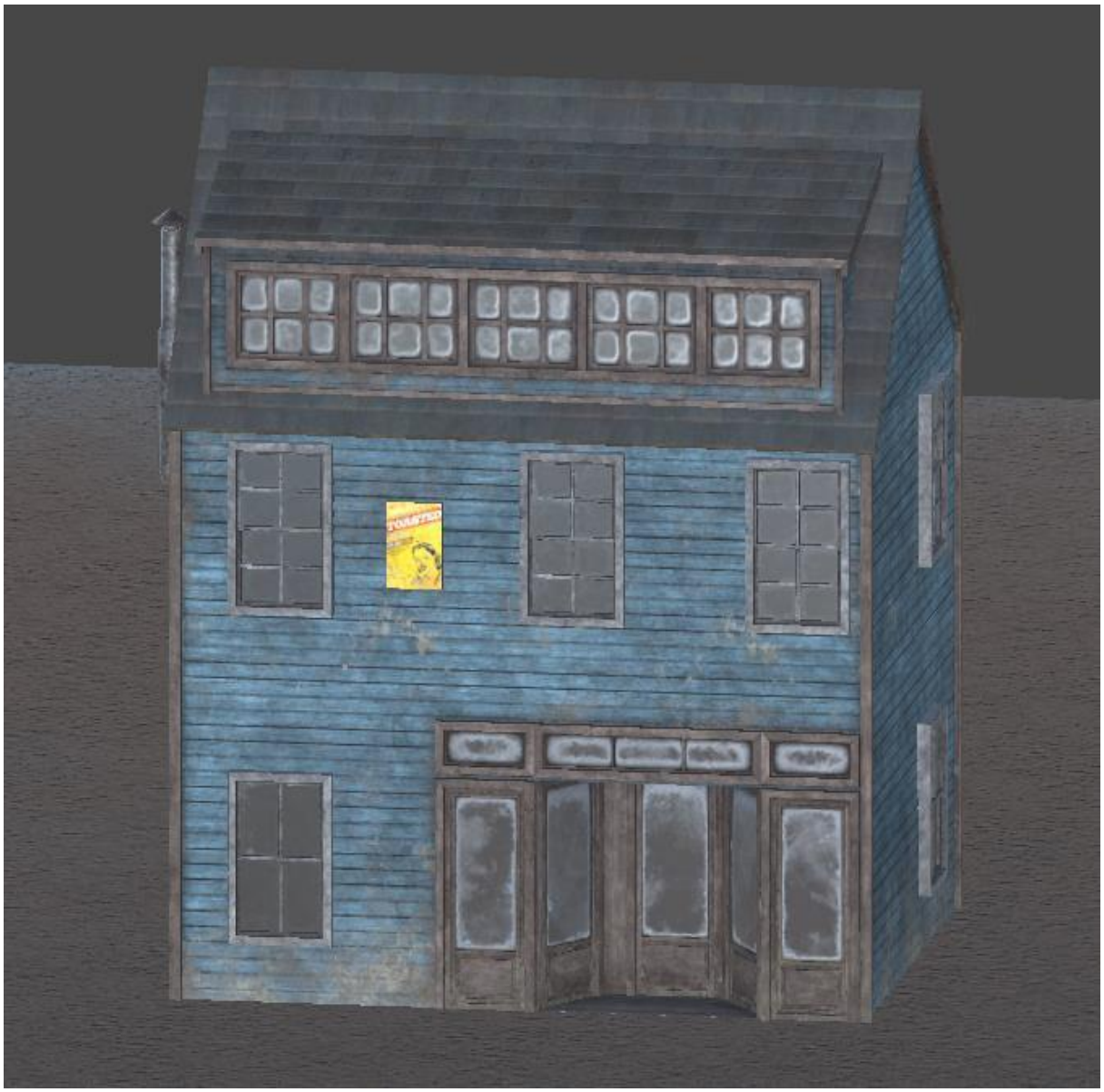
Η σκηνή αποτελείται από ένα Terrain, που έχει πάνω του τρία βασικά κτίρια: το σπίτι του χαρακτήρα, το εστιατόριο στο οποίο εργάζεται ή τρώει και τη βιβλιοθήκη-εκπαιδευτικό ίδρυμα στο οποίο πηγαίνει για να μάθει Αρχιτεκτονική. Στην παρακάτω εικόνα, το εστιατόριο βρίσκεται στη μέση, αριστερά είναι η βιβλιοθήκη και δεξιά το σπίτι του χαρακτήρα. Το μέγεθος του terrain είναι 100x100. Για να μοιάζει πιο αληθοφανές, μιας και δεν έχει υψώματα, πετρώματα ή χλωρίδα που να υποδεικνύουν ότι πρόκειται για έδαφος, έχουν χρησιμοποιηθεί διαφορετικού είδους textures.



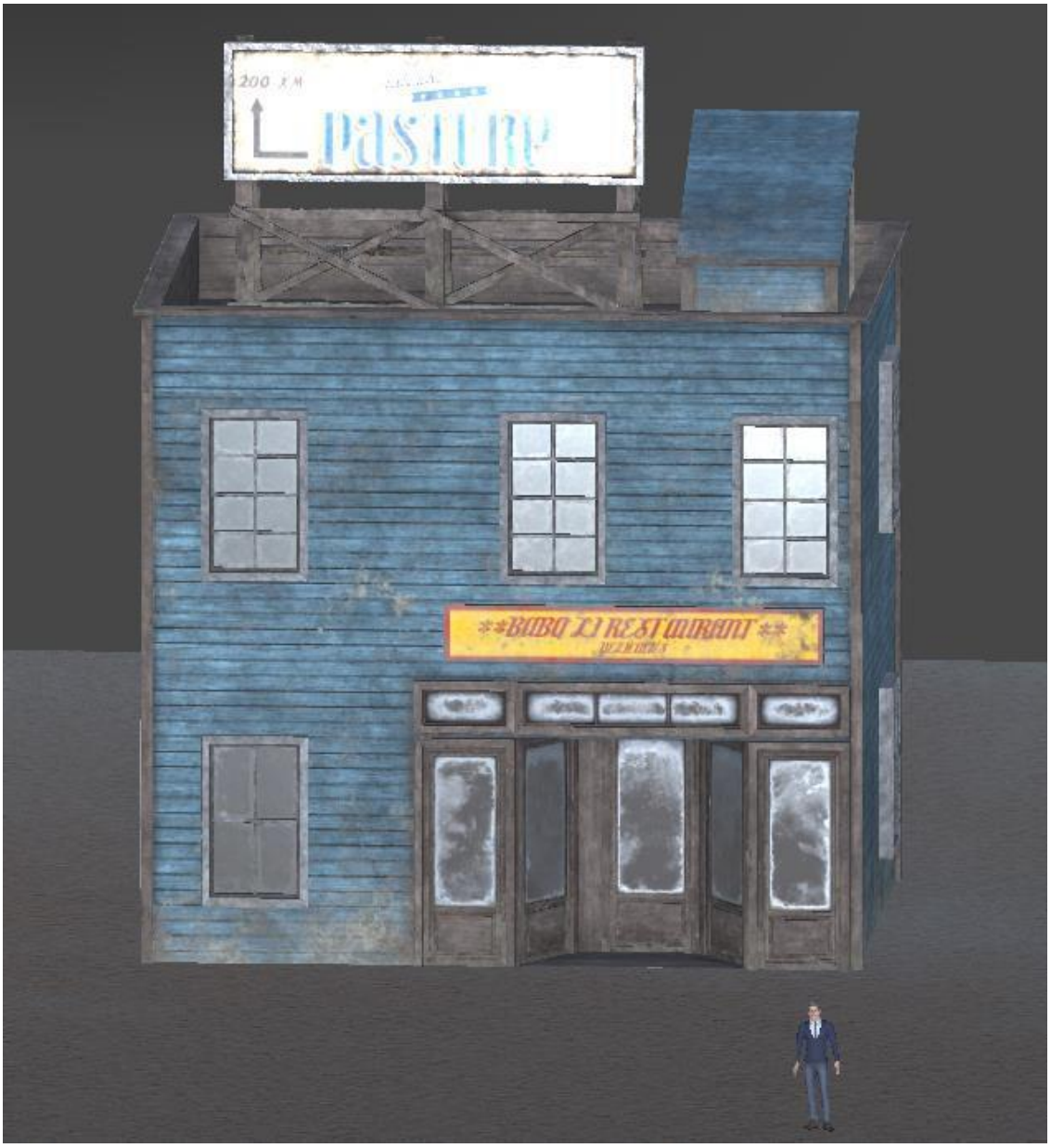
Εικόνα 10 Η σκηνή της προσομοίωσης



Εικόνα 11 Το σπίτι του Joe



Εικόνα 12 Το εκπαιδευτικό Ίδρυμα- Βιβλιοθήκη



Εικόνα 13 Το εστιατόριο που εργάζεται και τρώει

Τα κτίρια πάρθηκαν από το Asset Store της Unity. Έχουν προστεθεί σε όλα Triggered Box Colliders και ένα script έτσι ώστε να εντοπίζουν την είσοδο του Joe.

ΧΩΡΟΣ	SCRIPT ΟΤΑΝ Ο ΧΑΡΑΚΤΗΡΑΣ ΒΡΙΣΚΕΤΑΙ ΕΚΕΙ	TRIGGERED BY
ΕΣΤΙΑΤΟΡΙΟ	WorkMechanism.cs	WorkController.cs
ΕΣΤΙΑΤΟΡΙΟ	HungerMechanism.cs	WorkController.cs
ΣΠΙΤΙ	SleepMechanism.cs	HomeController.cs
ΒΙΒΛΙΟΘΗΚΗ	LevelUpMechanism.cs	SchoolController.cs

Πίνακας 1 Κατανομή script

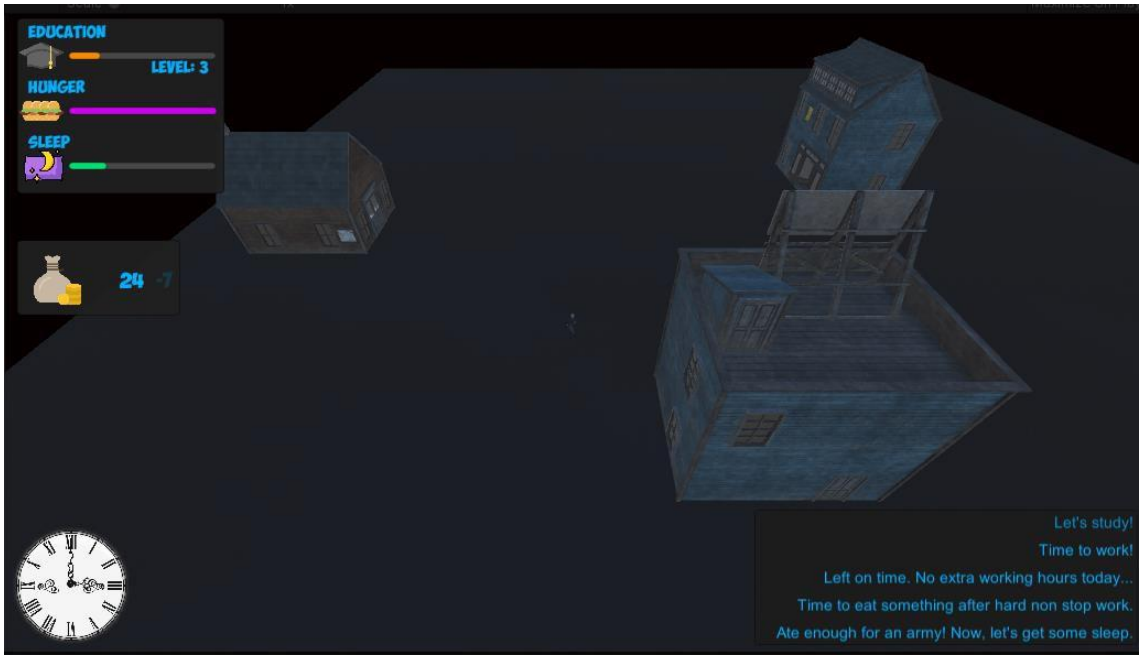
Η στήλη Triggered by αναφέρεται στα κτίρια ενώ το script είναι ανάλογο με τις καταστάσεις. Κάθε controller όταν ο collider του Joe συγκρούεται μαζί του, ενεργοποιεί την public μεταβλητή του και αυτόματα ενεργοποιείται ο ανάλογος μηχανισμός.

Παρακάτω παρατίθεται ο κώδικας του μεγαλύτερου Controller, με όνομα WorkController.cs, ο οποίος είναι υπεύθυνος για να ελέγχει την είσοδο του χαρακτήρα στο εστιατόριο, οπότε θα αναλυθεί ενδεικτικά. Διαθέτει δυο Boolean public μεταβλητές, που υποδηλώνουν την πρόθεση του χαρακτήρα, με βάση την κατάσταση στην οποία βρίσκεται. Για παράδειγμα, αν βρίσκεται στην κατάσταση Work, τότε ενεργοποιείται το WorkMechanism και το atWork γίνεται true. Αντίθετα, αν η κατάσταση είναι η Eat, τότε ενεργοποιείται το HungerMechanism. Παρατηρούμε πως η σειρά των πράξεων είναι ακριβώς αντίστροφη. Η πλειοψηφία των Controllers, δουλεύουν όπως το Hunger γιατί βασίζονται στην είσοδο του χαρακτήρα στο κτίριο. Η εργασία βασίζεται στην ώρα, όπως και στην πραγματικότητα και δεν μπορεί να είναι τυχαία. Οπότε ανάλογα με την ώρα προσέλευσης στη δουλειά, ενεργοποιείται το WorkMechanism και όταν ο χαρακτήρας φτάσει στο εστιατόριο, ενεργοποιείται το atWork που ενεργοποιεί κάποιες ακόμα δυνατότητες.

```
using _Scripts;
using UnityEngine;

public class WorkController : MonoBehaviour
{

    public static bool work;
    public static bool eat;
    private StatesMechanism states;

    public void Start()
    {
        work = false;
        eat = false;
        this.states =
GameObject.Find(Constants.GAME_MANAGER).GetComponent<StatesMechanism>();
    }

    public void OnTriggerEnter(Collider collider)
    {
        if (Constants.JOE.Equals(collider.gameObject.name))
        {
            // If Player is here to work
            if (States.Work.Equals(states.list[0]))
            {
                work = true;
            }
            // If Player is here to eat
            if (States.Eat.Equals(states.list[0]))
            {
                eat = true;
            }
        }
    }

    public void OnTriggerExit(Collider collider)
    {
        if (Constants.JOE.Equals(collider.gameObject.name))
        {
            work = false;
            eat = false;
        }
    }

}
```

4.3.2. ΥΛΟΠΟΙΗΣΗ ΚΑΜΕΡΑΣ ΚΑΙ ΧΑΡΑΚΤΗΡΑ

Σχετικά με την κάμερα, υπήρχε η σκέψη να είναι ελεύθερη για να μπορεί ο δέκτης να έχει μια αλληλεπίδραση με το περιβάλλον. Παρόλα αυτά, επειδή είναι αρκετά βασικό και δεν έχει πολλές λεπτομέρειες, αποφασίστηκε να είναι μια απλή κάμερα που θα ακολουθεί από μακριά την πορεία του χαρακτήρα.



Εικόνα 14 Η κάμερα της προσομοίωσης

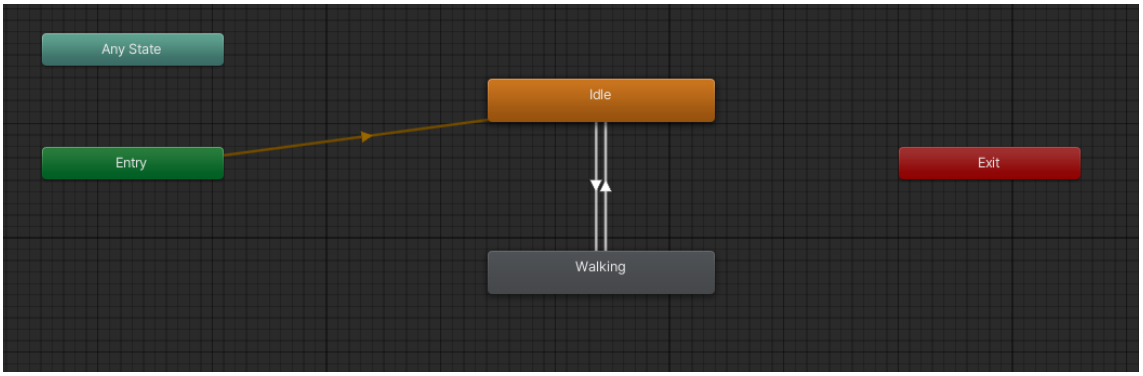
Εικόνα 15 Η κάμερα ακολουθεί τον NPC από ένα μακρινό πλάνο

```
public class CameraFollowNoRotation : MonoBehaviour
{

    public GameObject player;
    public int cameraHeight = 20;
    public int cameraOffset = 40;

    void Start()
    {
        player = GameObject.Find(Constants.JOE);
    }

    void Update()
    {
        Vector3 PlayerPOS = player.transform.position;
        transform.position = new Vector3(PlayerPOS.x, PlayerPOS.y + cameraHeight,
PlayerPOS.z - cameraOffset);
        transform.LookAt(player.transform);
    }

}
```

Από τον κώδικα της κάμερας γίνεται αντιληπτό ότι είναι μια πολύ απλή υλοποίηση, που ακολουθεί τη θέση του χαρακτήρα με κάποια απόκλιση.

Ένα εκ των δυο script που υπάρχουν πάνω στο χαρακτήρα είναι το PlayerController, που ρυθμίζει το Animation του Joe. Όταν βγαίνει από τα κτίρια του δίνεται το Animation “walk”.

Εικόνα 16 Ο Joe



Εικόνα 17 Ο Animator του Joe

```
public void Start()
{
    moveSpeed = 5f;

    home = GameObject.Find(Constants.HOME).transform; work
    = GameObject.Find(Constants.WORK).transform; school =
    GameObject.Find(Constants.SCHOOL).transform;

    animator = GameObject.Find(Constants.JOE).GetComponent<Animator>();
    statesMechanism = GameObject.Find(Constants.GAME_MANAGER).GetComponent<StatesMechanism>();
    deviation = new Vector3(1, 1, 1);
}

public void Update()
{
    if (States.Study.Equals(statesMechanism.list[0]) && !SchoolController.education)
    {
        GoTo(school);
    }
    else if ((States.Eat.Equals(statesMechanism.list[0]) && !WorkController.eat) ||
        (States.Work.Equals(statesMechanism.list[0]) && !WorkController.work))
    {
        GoTo(work);
    }
    else if (States.Rest.Equals(statesMechanism.list[0]) && !HomeController.sleep)
    {
        GoTo(home);
    }
}

public void GoTo(Transform target)
{
    animator.SetBool(Constants.IS_WALKING, true);

    float step = moveSpeed * Time.deltaTime; // calculate distance to move

    var targetPosition = target.position;
    targetPosition.y += 0.788f;

    transform.position = Vector3.MoveTowards(transform.position, targetPosition, step);

    if (transform.position != Vector3.zero)
    {
        transform.rotation = Quaternion.LookRotation(targetPosition - transform.position +
deviation);
    }

}
```

4.3.3. ΧΡΟΝΟΣ

Για να δημιουργηθεί μία πραγματικότητα και να είναι πιο αληθοφανές έπρεπε να υλοποιηθεί η έννοια του χρόνου. Κάθε μέρα μέσα στο παιχνίδι διαρκεί συνολικά δύο λεπτά, για να προλαβαίνει ο δέκτης να κατανοεί τι ακριβώς συμβαίνει κατά τη διάρκεια της προσομοίωσης. Η πρώτη μέρα ξεκινά από τα 35 δευτερόλεπτα δηλαδή στις 7:00 το πρωί Δοκιμάστηκαν αρκετά διαφορετικά δεδομένα, αλλά τελικά αποφασίστηκε η ώρα εκκίνησης της προσομοίωσης να είναι στις 7:00 το πρωί, γιατί οι μπάρες πεινάς και ύπνου βρίσκονται σε λογικά επίπεδα εκείνη την ώρα, αφού ο χαρακτήρας τρώει πριν κοιμηθεί.

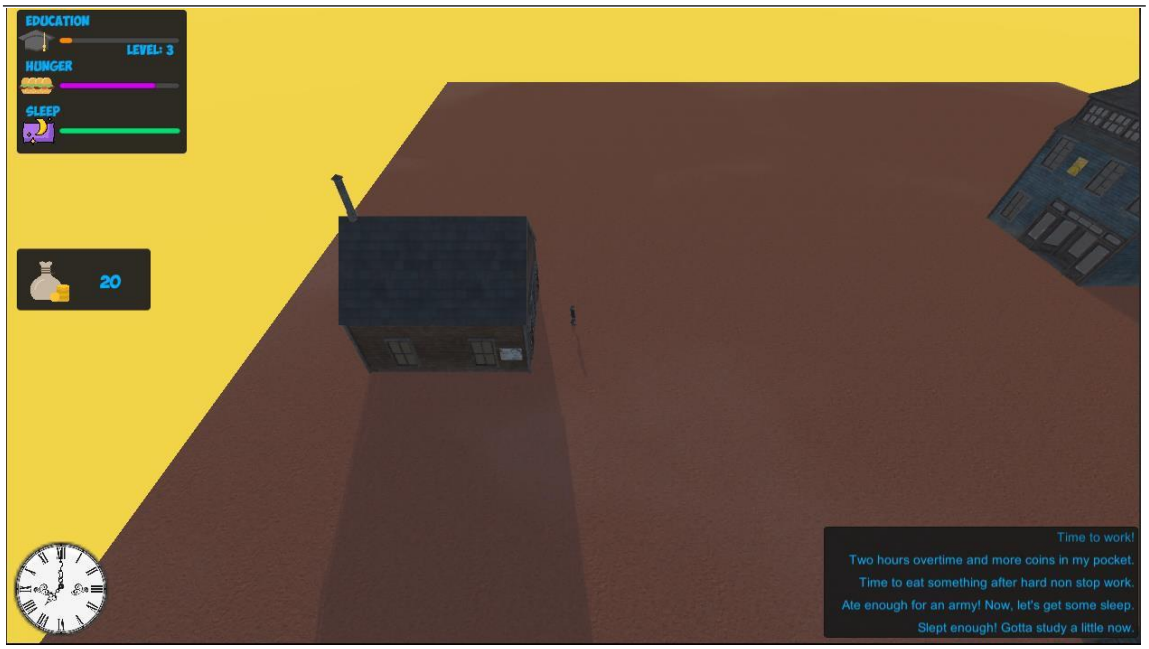
Με βάση τη συνάρτηση Time της Unity και μία μεταβλητή με όνομα gameTimer ελέγχουμε αν έχει περάσει ένα πραγματικό δευτερόλεπτο και αυξάνουμε την public μεταβλητή seconds κατά ένα, δίνοντας την αίσθηση του χρόνου. Μία καλύτερη εναλλακτική ίσως θα ήταν η χρήση της συνάρτησης InvokeRepeating() κάθε ένα δευτερόλεπτο για να μην είναι τόσο βαρύ το πρόγραμμα και κάνει συνεχείς ελέγχους σε πολλές διαφορετικές κλάσεις. Γιατί η Update function χρησιμοποιείται και στο μηχανισμό διαχείρισης εργασίας, όπως θα δούμε και παρακάτω.

Μέσα σε αυτήν τη συνάρτηση, ορίζεται και η έννοια του ρολογιού στη διεπαφή του χρήστη. Αρχικά υπήρξε η σκέψη να λειτουργεί και ο δείκτης των δευτερολέπτων, αλλά επειδή οι ώρες κυλούν γρήγορα θα ήταν πάρα πολύ κουραστικό για τον δέκτη. Οπότε χρησιμοποιήθηκε μόνο ο δείκτης της ώρας. Ουσιαστικά ορίζεται η έννοια της ημέρας μέσα στο παιχνίδι που είναι το deltaTime δια του 120 που είναι τα δευτερόλεπτα ανά ημέρα. Έτσι ώστε η κίνηση του δείκτη της ώρας να είναι λεία. Όταν τα δευτερόλεπτα γίνουν 120 ο μετρητής μηδενίζει και έτσι ξεκινάει η επόμενη μέρα.

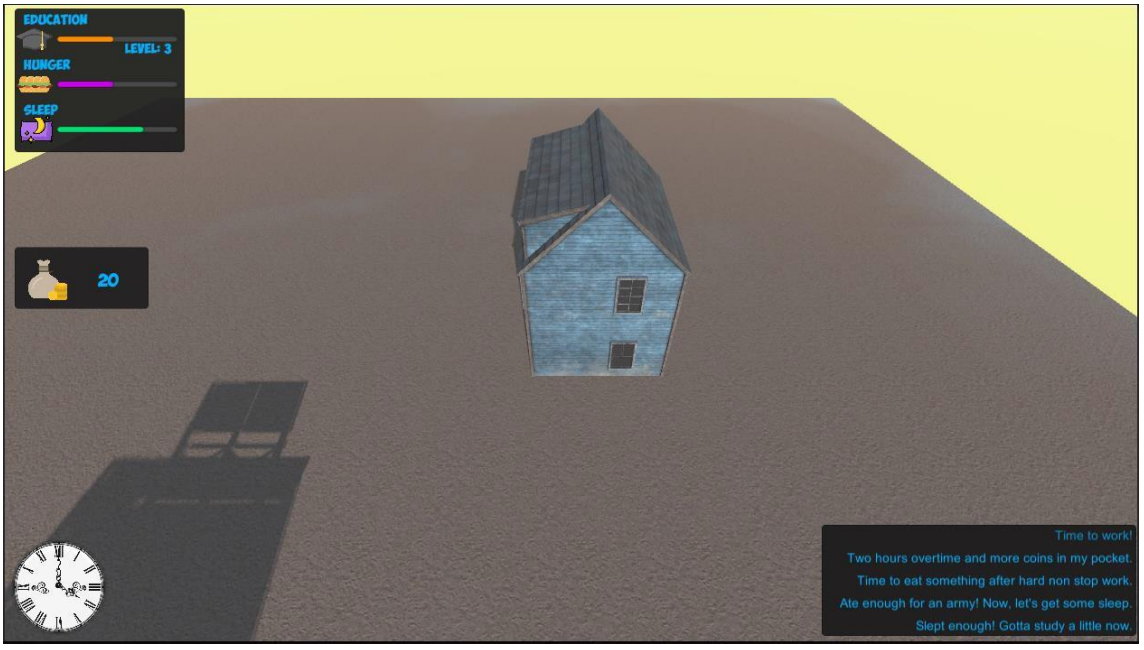
```
private void Start()
{
    hours = GameObject.Find(Constants.HOUR_HAND).transform;
    minutes = GameObject.Find(Constants.MINUTE_HAND).transform;
    gameTimer = Time.time;
    seconds = 35;
    counters =
GameObject.Find(Constants.GAME_MANAGER).GetComponent<LevelUpMechanism>();
    day = 0.583f;
}

private void Update()
{
    day += Time.deltaTime / Constants.REAL_SECONDS_PER_INGAME_DAY;
    float dayNormalized = day % 1f;
    float rotationDegreesPerDay = 360f;

    if (Time.time > gameTimer + 1)
    {
        gameTimer = Time.time;
        seconds++;
        WorkMechanism.sendOnlyOnce = true;
    }

    // Hour nail full circle
    hours.eulerAngles = new Vector3(0, 0, -dayNormalized * rotationDegreesPerDay);

    // Full Day/Night Cycle
    if (seconds == 120)
    {
        seconds = 0;
        counters.gameDaysSpentCounter++;
    }
}
```

Σε συνδυασμό με το χρόνο, υπάρχει κύκλος ημέρας νύχτας που είναι ορατός από τα ακόλουθα στιγμιότυπα:

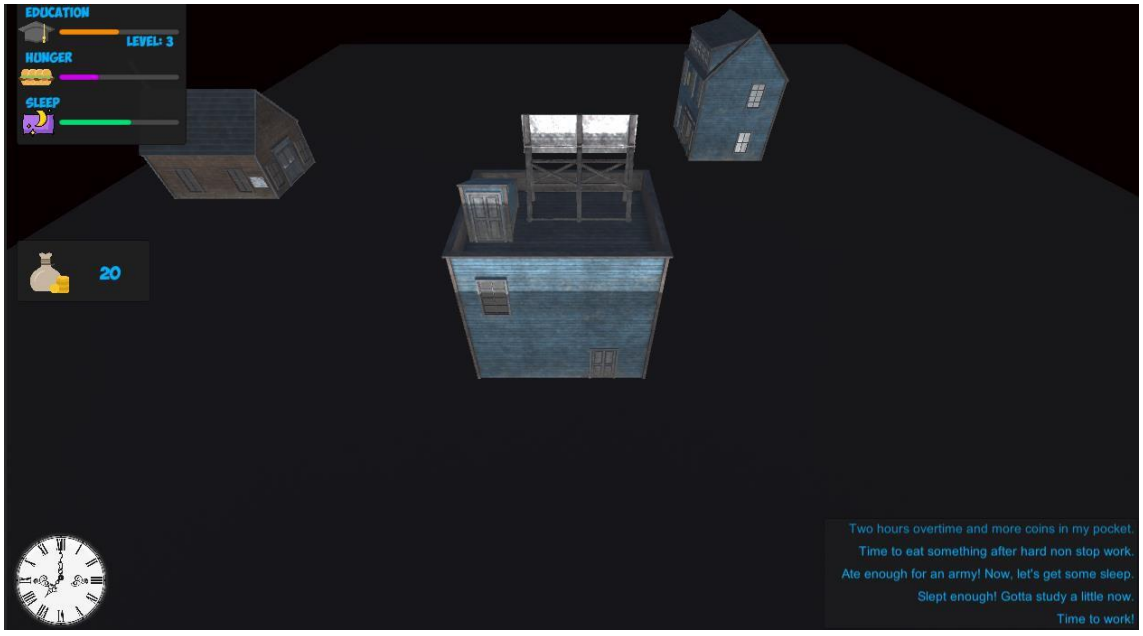
Εικόνα 18 6:00 το πρωί, δεν έχει ξημερώσει ακόμα στη σκηνή



Εικόνα 19 Μεταξύ 7:00 και 8:00 ο ήλιος αρχίζει να βγαίνει



Εικόνα 20 Στις 17:00 το απόγευμα είναι εμφανές ότι ο ήλιος δύει



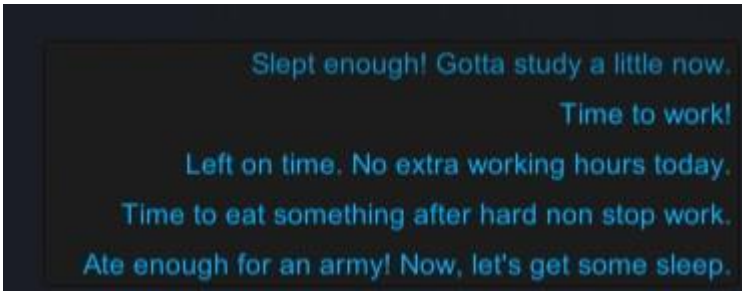
Εικόνα 21 Στις 20:00 είναι πλέον νύχτα

4.3.4. ΥΛΟΠΟΙΗΣΗ ΣΚΕΨΕΩΝ/ΜΗΝΥΜΑΤΩΝ

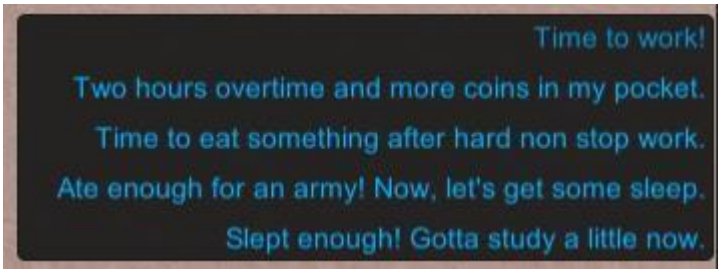
Μέσα στην πλειοψηφία των συναρτήσεων της προσομοίωσης χρησιμοποιείται η συνάρτηση UpdateChatBox(). Αυτή είναι υπεύθυνη για να ενημερώνει το μηχανισμό σκέψεων του χαρακτήρα. Για παράδειγμα αν ο Joe βρίσκεται στη δουλειά και πεινάει, τότε θα εμφανιστεί ένα μήνυμα το οποίο θα αναφέρει την ανάγκη του για φαγητό. Πρόκειται για σκέψη και όχι για ενημέρωση, γιατί δεν είναι δυνατό να σταματήσει την εργασία του για να φάει. Η έννοια του διαλείμματος στην εργασία δεν έχει υλοποιηθεί. Οπότε αν πεινάει ο χαρακτήρας συνήθως τρώει αμέσως μετά την εργασία του.

Σχετικά με το μηχανισμό σκέψεων-μηνυμάτων αρχικοποιείται ένα UI με πέντε διαφορετικά text. Οι νέες σκέψεις είναι αυτές που βρίσκονται κάτω και οι πιο παλιές είναι αυτές που βρίσκονται πιο ψηλά. Κάθε νέα σκέψη μεταφέρει την προηγούμενη μία θέση πιο πάνω. Η σκέψη που βρίσκεται στο πιο ψηλό text χάνεται. Από θέμα εμφάνισης για να δηλωθεί ότι η παλιά σκέψη είναι η πιο παλιά, έχει γίνει λίγο πιο θαμπή.

```
public Text pos1;
public Text pos2;
public Text pos3;
public Text pos4;
public Text pos5;

void Start()
{
    pos1 = GameObject.Find(Constants.THOUGHTS_TEXT_1).GetComponent<Text>();
    pos2 = GameObject.Find(Constants.THOUGHTS_TEXT_2).GetComponent<Text>();
    pos3 = GameObject.Find(Constants.THOUGHTS_TEXT_3).GetComponent<Text>();
    pos4 = GameObject.Find(Constants.THOUGHTS_TEXT_4).GetComponent<Text>();
    pos5 = GameObject.Find(Constants.THOUGHTS_TEXT_5).GetComponent<Text>();
    pos1.text = "Let's study!";
    pos2.text = pos3.text = pos4.text = pos5.text = " ";
}

public void UpdateChatBox(String newEntry)
{
    pos5.text =pos4.text;
    pos4.text =pos3.text;
    pos3.text =pos2.text;
    pos2.text =pos1.text;
    pos1.text = newEntry;
}
```



Εικόνα 22 Μηνύματα/Σκέψεις Joe. Ο Joe έφυγε κανονικά από τη δουλειά



Εικόνα 23 Μηνύματα/Σκέψεις Joe. Εδώ ο Joe έχει εργαστεί παραπάνω ώρες

4.3.5. ΥΛΟΠΟΙΗΣΗ ΜΗΧΑΝΙΣΜΟΥ ΕΠΙΠΕΔΩΝ

Η προσομοίωση ολοκληρώνεται όταν ο NPC φτάσει στο επίπεδο 7, που σηματοδοτεί την απόκτηση του πτυχίου του και τη λήξη των μαθήματων του. Κατά τη διάρκεια της

προσομοίωσης, ο NPC όταν βρίσκεται εντός του κτιρίου που θεωρείται ως “Εκπαιδευτικό Ίδρυμα”, με τη χρήση της OnTriggerEnter() ενεργοποιείται ο μηχανισμός που μετράει την εμπειρία του, η οποία αυξάνεται ανάλογα με το πόσο χρόνο περνάει εντός του εκπαιδευτικού ιδρύματος.

Όταν ο NPC εξέρχεται του κτιρίου, η καταμέτρηση της εμπειρίας σταματά με τη χρήση της συνάρτησης OnTriggerExit().

Αν ο NPC κατά τη διάρκεια της παραμονής του στο εκπαιδευτικό ίδρυμα, φτάσει την επιθυμητή εμπειρία για την απόκτηση ενός επιπέδου, τότε το επίπεδο αυξάνεται κατά ένα και εμφανίζεται ένα παράθυρο στην οθόνη το οποίο αναλύει τι ακριβώς έμαθε ο NPC σε αυτό το επίπεδο.

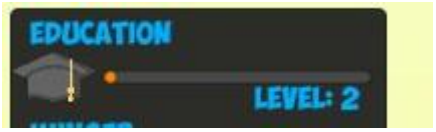
Σχετικά με την υλοποίηση του αλγορίθμου, δημιουργήθηκε η cs κλάση LevelUpMechanism η οποία αναλύεται ως εξής:

```
void Start()
{
    level = 1;
    experience = 0;
    experienceRequired = 100;
    levelText =GameObject.Find("LevelText").GetComponent<Text>();
    levelText.text = "Level: " + level;
    edu.maxValue = experienceRequired;
    sleptCounter = ateCounter = ateMoneySpentCounter = workedCounter =
        gameDaysSpentCounter = overtimesCounter =    overtimeExtraIncomeCounter =
0;
    workedMoneyEarnedCounter = 0f;
    startGame.transform.parent.gameObject.SetActive(true);

}

void LevelUp()
{
    isLevelingUp =true;
    level += 1;
    if (level ==5)
        EndGame();
    experience = 0;
    edu.value = 0;
    experienceRequired = SetExperienceRequired(level);
    levelText.text = "Level: " + level;
    edu.maxValue = experienceRequired;
    EducationAchievements.aCount = level;
    levelUpSound.Play();
}

public static void SetBoolFalse()
{
    isLevelingUp = false;
}

void Experience()
{
    if (experience >= experienceRequired)
        LevelUp();
    edu.value = experience / experienceRequired;
    if (SchoolController.education)
    {
        edu.value = experience++;
    }
    else
    {
        edu.value =experience;
        SetBoolFalse();
    }
}

private static float SetExperienceRequired(int level)
{
    //Thank you D&D!
    return 500 * Mathf.Pow(level,2) - (500 * level);
}
```

Στην αρχή του παιχνιδιού ορίζεται ως πρώτο επίπεδο το 1 και ως αρχική εμπειρία το 0. Το experienceRequired είναι η εμπειρία που χρειάζεται ο NPC για να φτάσει το επόμενο επίπεδο.

Στη συνάρτηση Update(), καλείται συνεχώς η συνάρτηση Experience(). Η δεύτερη ελέγχει συνεχώς αν η εμπειρία του NPC έχει ξεπεράσει το experienceRequired, έτσι ώστε να του δώσει

ένα επίπεδο, με τη συνάρτηση LevelUp(). Παράλληλα στην Experience() υπολογίζεται και η τρέχουσα εμπειρία του NPC. Αν βρίσκεται στο εκπαιδευτικό ίδρυμα, τότε αυξάνεται η εμπειρία του κατά μια μονάδα – μια μονάδα εντός της Update() που εκτελείται ανά frame, σημαίνει αλλεπάλληλες αυξήσεις ανά δευτερόλεπτο. Όταν ο NPC εξέλθει από το εκπαιδευτικό ίδρυμα, η εμπειρία του θα σταματήσει να αυξάνεται. Μέσα σε αυτή τη συνάρτηση ενημερώνεται και η αντίστοιχη μπάρα που ελέγχει την κατάσταση εμπειρίας του NPC.



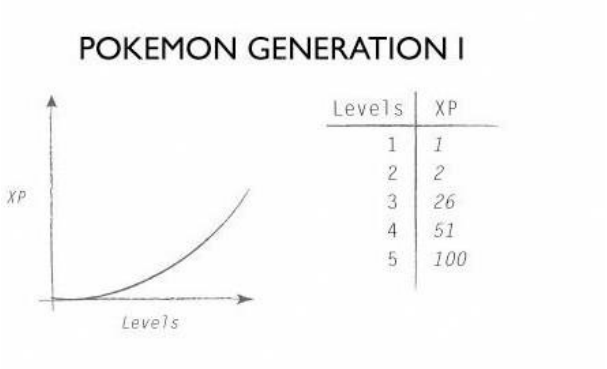
Εικόνα 24 Μπάρα Επιπέδων

Η συνάρτηση LevelUp() όπως προαναφέρθηκε, είναι υπεύθυνη για όλες τις διαδικασίες που συμβαίνουν όταν ο NPC, ανεβαίνει ένα επίπεδο. Αυτές είναι οι εξής:

- Αυξάνει το level του NPC κατά ένα.
- Αν το level είναι το 5, ενεργοποιεί τη συνάρτηση EndGame() που ορίζει το τέλος του παιχνιδιού.
- Μηδενίζει την μπάρα που δείχνει στο χρήστη την εμπειρία του NPC στο καινούργιο επίπεδο.
- Υπολογίζει τη νέα τιμή της μεταβλητής experienceRequired με τη συνάρτηση SetExperienceRequired().
- Αλλάζει το κείμενο στο επίπεδο -παράδειγμα στην εικόνα 23, ενώ παράλληλα ορίζει ως νέο μέγιστο στη μπάρα το experienceRequired.
- Αναλαμβάνει να ενημερώσει το χρήστη για την αλλαγή με την εμφάνιση μιας ειδοποίησης και την αναπαραγωγή ενός χαρακτηριστικού ήχου.

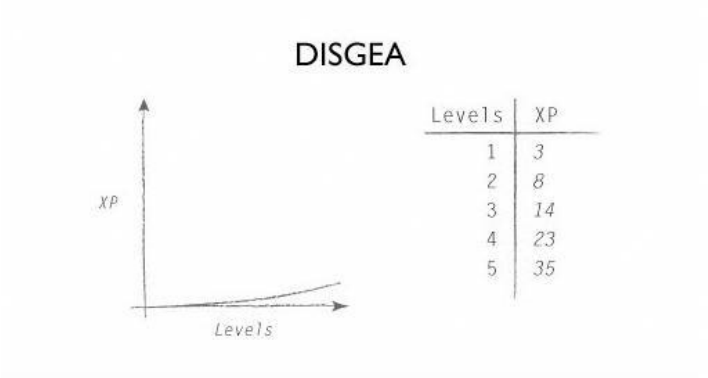
Η συνάρτηση SetExperienceRequired() επιστρέφει μια μαθηματική συνάρτηση. Η συνάρτηση αυτή πάρθηκε από το παιχνίδι Dungeons & Dragons

Πραγματοποιήθηκε έρευνα για τη συνάρτηση που θα χρησιμοποιούταν στο συγκεκριμένο πρόγραμμα. Λήφθηκαν υπόψιν οι συναρτήσεις του Pokemon Generation 1, του Dungeons and Dragons και του Disgea. Η συνάρτηση του Pokemon Generation 1 είναι πολύ αργή:



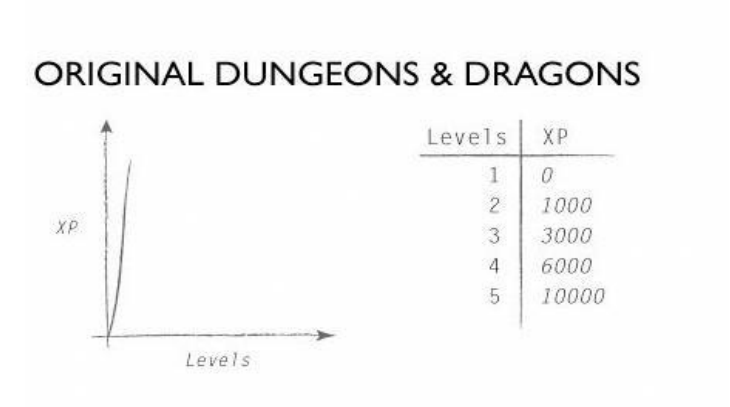

```
round((4 * (level ^ 3)) / 5)
```

Ακριβώς το ίδιο παρατηρείται και για την συνάρτηση του Disgea:



round(0.04 * (level ^ 3) + 0.8 * (level ^ 2) + 2 * level)

Αντίθετα ο μηχανισμός για την αύξηση στο επίπεδο του Dungeons & Dragons είναι ιδανικός. Ένας από τους στόχους αυτής της εργασίας είναι να φαίνεται όσο πιο ρεαλιστική γίνεται. Οπότε, η εμπειρία από επίπεδο σε επίπεδο πρέπει να αυξάνεται σταδιακά και σε σχετικά αργό ρυθμό, διότι προσομοιώνονται οι γνώσεις και ότι όσο περνάει ο καιρός τα μαθήματα του NPC γίνονται πιο δύσκολα και χρειάζονται περισσότερο χρόνο μελέτης. Ακολουθεί η συνάρτηση του Dungeons and Dragons:



500 * (level ^ 2) - (500 * level)

Εικόνα 25 Ειδοποίηση Επιπέδου 2

Εικόνα 26 Ειδοποίηση Επιπέδου 3

Εικόνα 27 Ειδοποίηση Επιπέδου 4

Εικόνα 28 Ειδοποίηση Επιπέδου 5

4.3.6. ΥΛΟΠΟΙΗΣΗ ΜΗΧΑΝΗΣ ΚΑΤΑΣΤΑΣΕΩΝ

Η μηχανή καταστάσεων, αποτελείται από τις παρακάτω διαθέσιμες καταστάσεις:

- Sleep
- Eat
- Work
- Study

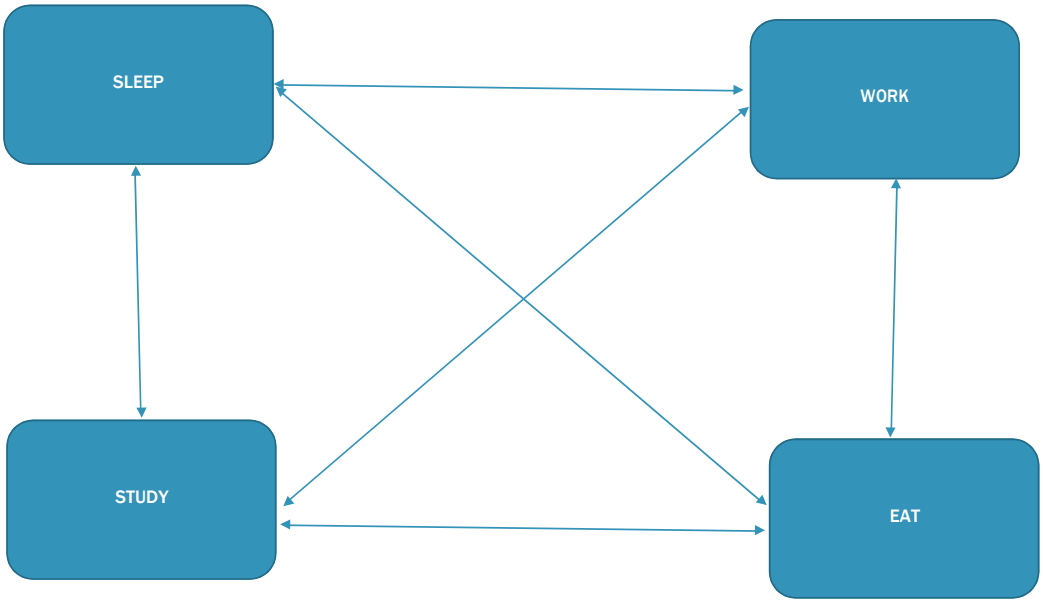
Αναλυτικά οι παραπάνω κατηγορίες μπορούν να μεταφραστούν ως εξής:

- Sleep: Κατάσταση στην οποία μεταβαίνει ο NPC όταν έχει ανάγκη για να κοιμηθεί και να επιστρέψει στο σπίτι του.
- Eat: Κατάσταση στην οποία μεταβαίνει ο NPC, όταν έχει ανάγκη για να φάει και να κοινωνικοποιηθεί.
- Work: Κατάσταση στην οποία μεταβαίνει ο NPC, όταν έχει έρθει η ώρα της ημέρας που πρέπει να εργαστεί. Είναι η μόνη κατάσταση που είναι βασισμένη στο χρόνο του παιχνιδιού, καθώς και στην πραγματικότητα υπάρχει μια συγκεκριμένη ώρα κατά την οποία ένας άνθρωπος πηγαίνει στη δουλειά του.
- Study: Κατάσταση στην οποία μεταβαίνει ο NPC, όταν οι ανάγκες για ύπνο και τροφή είναι καλυμμένες και ο NPC δεν εργάζεται. Είναι η κατάσταση κατά την οποία καλύπτει και τον στόχο της προσομοίωσης.

Οι αρχικές σκέψεις ήταν η δημιουργία μιας λίστας στην οποία θα αποθηκεύονται οι καταστάσεις στις οποίες ήθελε να μεταβεί κατά σειρά ο NPC.

Η λογική ήταν παρόμοια με τη σειρά παιχνιδιών Sims, όπου κάθε Sim ανάλογα με την κατάσταση και την ανάγκη για πρωτογενείς και δευτερογενείς ανάγκες της πυραμίδας του Maslow, διαθέτει μια λίστα αναγκών που πρέπει να ολοκληρωθούν. Όταν μια ανάγκη χρειάζεται να γίνει άμεσα, η λίστα διαγράφεται αυτόματα και τη θέση της παίρνει η προαναφερθείσα ανάγκη.

Τελικά η υλοποίηση βασίστηκε σε μια λίστα, αλλά στην πραγματικότητα το μόνο που συμβαίνει είναι να αλλάζει η τρέχουσα κατάσταση του NPC. Παρόλα αυτά, θα μπορούσαν να βασιστούν πολλές επεκτάσεις στον κώδικα και στη λογική της παρούσας διπλωματικής όπως θα αναφερθούν και στο κεφάλαιο 6.



Πίνακας 2 Οι καταστάσεις και οι επιτρεπόμενες εναλλαγές

Η λίστα δημιουργήθηκε ως εξής σε ένα cs αρχείο με όνομα StatesMechanism.cs, το οποίο σχεδιάστηκε αποκλειστικά για τη διαχείριση της:

```
public enum States { Study, Eat, Rest, Work }
public class StatesMechanism : MonoBehaviour
{

    public ArrayList list;

    private void Start()
    {
        Time.timeScale = 0;
        list = new ArrayList();
        InitializeList();
    }

    public void ImmediateChangeCurrentState(States
newCurrentState)
    {
        list.Clear();
        list.Add(newCurrentState);
    }

    public void InitializeList()
    {
        list.Add(States.Study);
    }
}
```

Η λίστα δημιουργείται κατά την αρχή της εκτέλεσης του προγράμματος και της δίνεται η αρχική κατάσταση STUDY. Οπότε, αρχικά ο NPC θα αρχίσει να κινείται προς το κτίριο του Πανεπιστημίου. Η συνάρτηση ImmediateChangeCurrentState() είναι αυτή που καλείται στο πρόγραμμα, όταν ο NPC αλλάζει την τρέχουσα κατάστασή του.

4.3.7. ΜΗΧΑΝΙΣΜΟΣ ΔΙΑΧΕΙΡΙΣΗΣ ΥΠΝΟΥ

Ο μηχανισμός διαχείρισης ύπνου ενεργοποιείται κάθε φορά που η μπάρα του ύπνου είναι κάτω από το 25%, δηλαδή όταν ο NPC διαθέτει μόνο το ¼ της αρχικής του ενέργειας. Ο μηχανισμός αυτός χωρίζεται σε δυο διαφορετικές καταστάσεις: την κατάσταση όπου ο NPC ξεκουράζεται και η μπάρα ύπνου γεμίζει και την κατάσταση στην οποία ο NPC ασχολείται με τις καθημερινές του δραστηριότητες, με αποτέλεσμα η μπάρα του ύπνου να αδειάζει σταδιακά.

Εικόνα 29 Μπάρα Ύπνου

Αρχικά πρέπει να αναφερθεί ότι σε κάθε εκτέλεση του προγράμματος η τρέχουσα κατάσταση ύπνου παίρνει μια τιμή μεταξύ των ακέραιων αριθμών 70 και 85, κάνοντας έτσι την κάθε εκτέλεση μοναδική. Οι αριθμοί αυτοί επιλέχθηκαν γιατί το πρόγραμμα ξεκινάει στις 7 το πρωί, οπότε ο NPC πρέπει να είναι σχετικά ξεκούραστος εκείνη την ώρα. Σε περίπτωση που επιλεχθούν άλλοι αριθμοί, για παράδειγμα 45 ως 85 και η τρέχουσα κατάσταση ύπνου είναι γύρω στο 55, τότε οι ώρες κατά τις οποίες ξεκουράζεται ο NPC, δεν ταιριάζουν απόλυτα στην πραγματικότητα. Μπορεί να τύχει να διαβάζει στις 3 το πρωί και να κοιμάται 5 ώρες αργότερα. Ο κώδικας που υλοποιήθηκε για το μηχανισμό διαχείρισης ύπνου είναι ο εξής:

```
private void Start()
{
    sleep.maxValue = 100;
    sleepNeedFulfilled = false;
    sendSleepyOnlyOnce = false;
    InvokeRepeating(Constants.SLEEP_LEVEL, 2.0f, 2.0f);
}

public void RandomizeInitialSleepness()
{
    sleep.value =
Random.Range(Constants.RANDOM_INITIAL_SLEEPNESS_MIN,Constants.RANDOM_INITIAL_
SLEEPNES_MAX);
}
```

```
public void SleepLevel()
{
    if (HomeController.sleep)
    {
        if (sleep.value < 100 && !sleepNeedFulfilled)
        {
            sleep.value += 8;
        }
        if (sleep.value >= 100)
        {
            sleep.value = 100;
            sleepNeedFulfilled =true;
            counters.sleptCounter++;
            sendSleepyOnlyOnce = false;
        }
        if (sleepNeedFulfilled)
        {
            HomeController.sleep = false;
            if (hunger.hunger.value - 25 >0)
            {
                states.ImmediateChangeCurrentState(States.Study);
                chatBox.UpdateChatBox(Constants.CHAT_BOX_SLEEP_TO_STUDY);
            }
            else
            {
                states.ImmediateChangeCurrentState(States.Eat);
                chatBox.UpdateChatBox(Constants.CHAT_BOX_SLEEP_TO_EAT);
            }
        }
    }
    else
    {
        if (timeScript.seconds % 5 == 0)
        {
            if (sleep.value - 25 >0)
            {
                sleep.value -= 11;
            }
            else if (sleep.value - 25 <= 0 && sleep.value - 8 > 0)
            {
                sleep.value -= 2;
            }
            else if (sleep.value - 8 <= 0)
            {
                sleepNeedFulfilled = false;
                states.ImmediateChangeCurrentState(States.Rest);
                if (!sendSleepyOnlyOnce)
                {
                    chatBox.UpdateChatBox(Constants.CHAT_BOX_NEED_SLEEP);
                    sendSleepyOnlyOnce = true;
                }
            }
        }
    }
}
```

Οι δύο καταστάσεις που προαναφέρθηκαν, δηλαδή όταν ο NPC είναι εντός της κατοικίας του και

κοιμάται και όταν είναι εκτός της κατοικίας του και χάνει ενέργεια ενώ παράλληλα ασχολείται με οποιαδήποτε άλλη δραστηριότητα, αναλύονται από τον κώδικα με τη συνάρτηση SleepLevel() η οποία εκτελείται κάθε δυο δευτερόλεπτα.

Σε αυτό το σημείο πρέπει να αναφερθεί ότι κάθε κτίσμα που έχει πρόσβαση ο NPC, διαθέτει ένα Box Collider, ο οποίος έχει μια μεταβλητή που ελέγχει αν ο Collider του NPC, συγκρούεται με τον Box Collider. Όταν συμβαίνει αυτό, θεωρείται ότι ο NPC βρίσκεται μέσα στο εκάστοτε κτίριο.

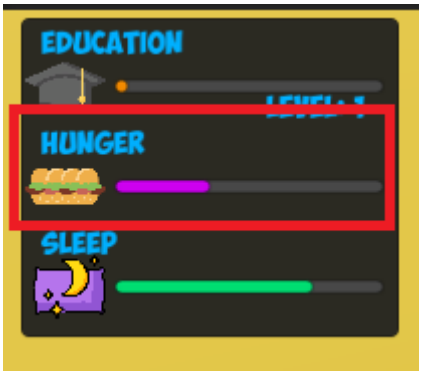
Σχετικά με αυτή την κατάσταση, ο κώδικας αναλύεται ως εξής:

Αν η τρέχουσα κατάσταση ύπνου δεν είναι στο 100%, τότε κάθε 2 δευτερόλεπτα, η τρέχουσα αυτή κατάσταση αυξάνεται κατά 8 μονάδες. Αφού η αύξηση γίνεται κατά 8 μονάδες, υπάρχει η πιθανότητα η τρέχουσα κατάσταση να είναι μεγαλύτερη από την μέγιστη κατάσταση της μπάρας. Σε αυτή την περίπτωση ορίζουμε ως τρέχουσα κατάσταση τη μέγιστη και παράλληλα ενημερώνουμε τη μπάρα κατάστασης ύπνου. Τέλος, με τη Boolean μεταβλητή sleepNeedFullfilled, επιβεβαιώνουμε ότι ο NPC έχει ξεκουραστεί στο μέγιστο δυνατό και ενεργοποιούμε τη διαδικασία αφύπνισης. Στην τελευταία, προσομοιώνεται η επόμενη απόφαση του NPC για το τι θα κάνει στη συνέχεια. Πρώτα ελέγχεται αν ο NPC πεινάει, δηλαδή αν η μπάρα πείνας είναι κάτω από 25%, όπου η τρέχουσα κατάσταση από Sleep, γίνεται Hunger και ο NPC κινείται προς το εστιατόριο όπου και εργάζεται για να φάει. Εναλλακτικά, πηγαίνει στο εκπαιδευτικό ίδρυμα για να μελετήσει και να κερδίσει εμπειρία.

Η άλλη κατάσταση που ελέγχεται στη συγκεκριμένη συνάρτηση είναι όταν ο NPC βρίσκεται εκτός σπιτιού και ασχολείται με άλλες δραστηριότητες. Αν η μπάρα ύπνου έχει ως τρέχουσα κατάσταση πάνω από το 25%, τότε κάθε δέκα δευτερόλεπτα αυτή μειώνεται κατά 6 μονάδες. Υπάρχει όμως και η πιθανότητα ο NPC να βρίσκεται στη δουλεία ή να τρέφεται, που σε αυτή την περίπτωση δεν γίνεται να φύγει προτού ολοκληρώσει τη δραστηριότητα. Οπότε ενεργοποιείται μια κατάσταση σχετικής αδράνειας, που κάθε δέκα δευτερόλεπτα χάνει 2 μονάδες από την τρέχουσα κατάσταση. Αν φτάσει στο σημείο η τρέχουσα κατάσταση ύπνου να είναι μικρότερη από το 8%, τότε ο μηχανισμός καταστάσεων, αλλάζει άμεσα την τρέχουσα δραστηριότητα και ο NPC πηγαίνει κατευθείαν σπίτι για να κοιμηθεί.

4.3.8. ΜΗΧΑΝΙΣΜΟΣ ΔΙΑΧΕΙΡΙΣΗΣ ΠΕΙΝΑΣ

Ο μηχανισμός διαχείρισης πείνας ενεργοποιείται κάθε φορά που η μπάρα πείνας είναι κάτω από το 25%, δηλώνοντας έτσι την ανάγκη του NPC για τροφή. Ο μηχανισμός αναλύεται σε δυο διαφορετικές καταστάσεις: την κατάσταση όπου ο NPC τρέφεται και η μπάρα πείνας γεμίζει και την κατάσταση στην οποία ο NPC ασχολείται με τις καθημερινές του δραστηριότητες, με αποτέλεσμα η μπάρα πείνας να αδειάζει σταδιακά.



Εικόνα 30 Μπάρα πείνας

Σε κάθε εκτέλεση του προγράμματος η τρέχουσα πείνα παίρνει μια τιμή μεταξύ των ακέραιων αριθμών 25 και 50. Οι αριθμοί αυτοί επιλέχθηκαν γιατί με βάση το χρόνο στον οποίο εκτυλίσσεται μια ημέρα του προγράμματος, ο NPC τρέφεται δυο φορές μέσα στην ημέρα, μια το

μεσημέρι και μια μόλις ολοκληρώσει τη δουλειά του. Ο κώδικας που υλοποιήθηκε για το μηχανισμό διαχείρισης πείνας είναι ο εξής:

```
public void Start()
{
    hunger.maxValue = 100;
    penaltyValue = randomValue = 0;
    curValue = 0.0f;
    fadeTimeCounter = 0;
    InvokeRepeating(Constants.HUNGER_LEVEL, 2.0f, 2.0f);
    spentMoney.canvasRenderer.SetAlpha(1.0f);
    spentMoney.text = "";
    hungerNeedFulfilled = false;
    fadeIn = false;
}

public void RandomizeInitialHunger()
{
    hunger.value = Random.Range(Constants.RANDOM_INITIAL_HUNGER_MIN,
Constants.RANDOM_INITIAL_HUNGER_MAX);
}
```

```
public void HungerLevel()
{
    if (WorkController.eat)
    {
        if (hunger.value < 100 && !hungerNeedFulfilled)
        {
            hunger.value += 18;
        }

        if (hunger.value >= 100)
        {
            hunger.value = 100;
            hungerNeedFulfilled = true;
            counters.ateCounter =+ counters.ateCounter+1;
        }

        if (hungerNeedFulfilled)
        {
            SpendRandomMoneyOnFood();

            WorkController.eat = false;
            if (sleep.sleep.value - 25 > 0)
            {
                states.ImmediateChangeCurrentState(States.Study);
                chatBox.UpdateChatBox(Constants.CHAT_BOX_HUNGER_TO_STUDY);
            }
            else
            {
                states.ImmediateChangeCurrentState(States.Rest);
                chatBox.UpdateChatBox(Constants.CHAT_BOX_HUNGER_TO_SLEEP);
            }
        }
    }
    else
    {
        if (timeScript.seconds % 5 == 0)
        {
            if (hunger.value - 25 > 0)
            {
                hunger.value -= 7;
            }
            // When he needs to feed, add in list Hunger State
            else if (hunger.value - 25 <= 0 && hunger.value - 8 > 0)
            {
                hunger.value -= 2;
            }
            else if (hunger.value - 8 <= 0)
            {
                hungerNeedFulfilled = false;
                states.ImmediateChangeCurrentState(States.Eat);
            }
        }
    }
    if (fadeIn)
    {
        FadeOut();
        fadeIn = false;
        spentOnFoodAudio.Play();
    }
}
```

```
private void SpendRandomMoneyOnFood()
{
    randomValue = GetRandomValue();
    coins.text = (int.Parse(coins.text) - randomValue).ToString();
    spentMoney.text = "-" + randomValue;
    FadeIn();
    fadeIn = true;
    counters.ateMoneySpentCounter += randomValue;
}

private int GetRandomValue()
{
    float rand = Random.value;
    if (rand <= .2f)
        return 5;
    if (rand <= .4f)
        return 6;
    if (rand <= .6f)
        return 7;
    if (rand <= .8f)
        return 8;
    return rand <= .9f ? 9 : 10;
}

private void FadeIn()
{
    spentMoney.CrossFadeAlpha(1, 2, false);
}

private void FadeOut()
{
    spentMoney.CrossFadeAlpha(0, 5, false);
}
```

Οι δύο καταστάσεις που προαναφέρθηκαν, δηλαδή όταν ο NPC είναι εντός του εστιατορίου για να φάει και όταν ασχολείται με οποιαδήποτε άλλη κατάσταση, αυξάνοντας την πείνα του, αναλύονται από τον κώδικα με τη συνάρτηση HungerLevel() η οποία εκτελείται κάθε δυο δευτερόλεπτα.

Σχετικά με αυτή την κατάσταση, ο κώδικας αναλύεται ως εξής:

Αν η τρέχουσα κατάσταση πείνας δεν είναι στο 100%, τότε κάθε 2 δευτερόλεπτα, αυξάνεται κατά 18 μονάδες. Υπάρχει έτσι η πιθανότητα, η τρέχουσα κατάσταση να είναι μεγαλύτερη από την μέγιστη κατάσταση της μπάρας. Σε αυτή την περίπτωση ορίζουμε ως τρέχουσα κατάσταση τη μέγιστη και παράλληλα ενημερώνουμε τη μπάρα κατάστασης πείνας. Τέλος, με τη Boolean μεταβλητή hungerNeedFulfilled, επιβεβαιώνουμε ότι ο NPC έχει τραφεί στο μέγιστο δυνατό. Αμέσως μετά ο NPC πληρώνει με νομίσματα για το φαγητό του και τέλος και ενεργοποιείται μια διαδικασία κατά την οποία, προσομοιώνεται η απόφαση του NPC για το τι θα κάνει στη συνέχεια. Πρώτα ελέγχεται αν ο NPC χρειάζεται να κοιμηθεί, δηλαδή αν η μπάρα ύπνου είναι κάτω από 25%, όπου η τρέχουσα κατάσταση από Hunger, γίνεται Sleep και ο NPC κινείται προς το σπίτι του για να κοιμηθεί. Εναλλακτικά, πηγαίνει στο εκπαιδευτικό ίδρυμα για να μελετήσει και να κερδίσει εμπειρία. Οπότε η τρέχουσα κατάσταση του μηχανισμού από Hunger γίνεται Study.

Η άλλη κατάσταση που ελέγχεται στη συγκεκριμένη συνάρτηση είναι όταν ο NPC δεν τρέφεται και ασχολείται με άλλες δραστηριότητες. Αν η μπάρα πείνας έχει ως τρέχουσα κατάσταση πάνω από το 25%, τότε κάθε δυο δευτερόλεπτα αυτή μειώνεται κατά 7 μονάδες. Υπάρχει όμως και η πιθανότητα ο NPC να βρίσκεται στη δουλειά ή να κοιμάται, που σε αυτή την περίπτωση δεν γίνεται να φύγει προτού ολοκληρώσει την τρέχουσα δραστηριότητα. Οπότε ενεργοποιείται μια κατάσταση σχετικής αδράνειας, που κάθε δέκα δευτερόλεπτα χάνει 2 μονάδες από την τρέχουσα κατάσταση. Αν φτάσει στο σημείο η τρέχουσα κατάσταση πείνας να είναι μικρότερη από το 8%, τότε ο μηχανισμός καταστάσεων, αλλάζει άμεσα την τρέχουσα δραστηριότητα και ο NPC πηγαίνει κατευθείαν στο εστιατόριο για να τραφεί.

Όταν χρειαστεί ο NPC να «πληρώσει» για το φαγητό που έφαγε από το εστιατόριο, καλείται η συνάρτηση SpendRandomMoneyOnFood(). Η συνάρτηση αυτή βασίζεται στην GetRandomValue, που στηρίζεται στις παρακάτω πιθανότητες:

Πιθανότητα	Νομίσματα
0.0-0.2	5
0.21-0.4	6
0.41-0.6	7
0.61-0.8	8
0.81-0.9	9
0.91-1.0	10

Πίνακας 3 Πιθανότητες χρημάτων για κατανάλωση φαγητού

Η πιθανότητα να ξοδέψει πολλά χρήματα στο φαγητό παρατηρούμε ότι είναι λιγότερες, όπως συμβαίνει και στην πραγματικότητα.

4.3.9. ΜΗΧΑΝΙΣΜΟΣ ΔΙΑΧΕΙΡΙΣΗΣ ΕΡΓΑΣΙΑΣ

Ο μηχανισμός διαχείρισης εργασίας είναι υπεύθυνος για τη διάρκεια της ημέρας που ο χαρακτήρας βρίσκεται στον τόπο εργασίας του. η προσομοίωση ξεκινάει και ο χρήστης έχει εξαρχής 10 νομίσματα. παράλληλα το συγκεκριμένο μηχανισμό αρχικό ποιούνται και κάποιες μεταβλητές που έχουν σχέση με τις αναφορές τα οποία εμφανίζονται ο στατιστικά στο τέλος της προσομοίωσης. Κάποια από αυτά είναι οι συνολικές ώρες εργασίας και ο μισθός.

```
private void Start()
{
    coins.text = "10";
    totalWorkingHours = 0;
    salary = 0;
    sleepBeforeWork =false;
    sendOnlyOnce = true;
    overtime = 0;
}
```

Στην διάρκεια της προσομοίωσης ο χαρακτήρας εργάζεται ως βοηθητικό προσωπικό σε ένα εστιατόριο. Αυτό σημαίνει ότι όπως και στην πραγματικότητα, έτσι και σε αυτή την περίπτωση εργάζεται περισσότερο απογευματινές και βραδινές ώρες. Ακριβώς επειδή είναι βοηθητικό προσωπικό, δεν θα μπορούσε η απασχόληση του να είναι πλήρης, επομένως είναι μερική. Πολλές φορές υπάρχει πιθανότητα να καθίσει και για υπερωρίες. Σύμφωνα με αυτά τα δεδομένα έχει θεωρηθεί πως ώρα προσέλευσης στη δουλειά είναι 5:00 το απόγευμα και η ώρα αναχώρησης είναι στις 9:00 το βράδυ. Υπάρχουν φορές που ο χαρακτήρας χρειάζεται να μείνει για υπερωρίες οι οποίες διαρκούν από μία έως τέσσερις παραπάνω ώρες και πληρώνονται με παραπάνω νομίσματα.

Παρακάτω παρατίθεται ο κώδικας της συγκεκριμένης κλάσης. Το πιο σημαντικό χαρακτηριστικό της είναι η Update function. Σε αυτή ελέγχεται διαρκώς η ώρα. Στο 80ο δευτερόλεπτο της κάθε ημέρας, δηλαδή στις 5:00 το απόγευμα, δίνεται η εντολή στο χαρακτήρα να αλλάξει αμέσως την τρέχουσα κατάστασή του στο μηχανισμό καταστάσεων και να πάει στη δουλειά του. Παράλληλα είναι το σημείο που αποφασίζεται αν ο Joe θα κάνει υπερωρίες ή όχι τη συγκεκριμένη ημέρα.

```
private void Update()
{

    // Time to go to work (Around 17:00) --> 16*5+2 = 80 = 80
    if (timeScript.seconds % 80 == 0 && timeScript.seconds != 0)
    {
        states.ImmediateChangeCurrentState(States.WORK);
        overtime = GetRandomValue();

        if (sendOnlyOnce)
        {
            chatBox.UpdateChatBox(Constants.CHAT_BOX_WORK_TIME_TO_WORK);
            sendOnlyOnce = false;
        }
    }

    // Time to leave from work (Around 22:00) --> 22*5=110
    // No overtime
    if (timeScript.seconds % 110 == 0 && overtime == 0 &&
WorkController.work)
    {
        CalculateWork(5, Constants.CHAT_BOX_WORK_LEFT_ON_TIME);
    }

    // Time to leave from work (Around 23:00) --> 23*5=115
    // One hour overtime
    if (timeScript.seconds % 115 == 0 && overtime == 1 &&
WorkController.work)
    {
        CalculateWork(6, Constants.CHAT_BOX_WORK_1_HOUR_OVERTIME);
        counters.overtimesCounter += 1;
        counters.overtimeExtraIncomeCounter += 3;
    }

    // Time to leave from work (Around 24:00) --> 24*5=120
    // Two hours overtime
    if (timeScript.seconds % 120 == 0 && overtime == 2 &&
WorkController.work)
    {
        CalculateWork(7, Constants.CHAT_BOX_WORK_2_HOURS_OVERTIME);
        counters.overtimesCounter += 2;
        counters.overtimeExtraIncomeCounter += 6;
    }

    // Time to leave from work (Around 1:00) --> 1*5=5
    // Three hours overtime
    if (timeScript.seconds / 5 == 1 && overtime == 3 && WorkController.work)
    {
        CalculateWork(8, Constants.CHAT_BOX_WORK_3_HOURS_OVERTIME);
        counters.overtimesCounter += 3;
        counters.overtimeExtraIncomeCounter += 9;
    }
}
```

Αφού έχει λοιπόν προαποφασιστεί τι ώρα θα σταματήσει τη δουλειά, το script περιμένει να ενεργοποιηθεί πάλι τη συγκεκριμένη ώρα. Τότε, ενεργοποιείται η CalculateWork() που δέχεται ως ορίσματα τις συνολικές ώρες που εργάστηκε ο NPC, καθώς και το μήνυμα-σκέψη που θα πρέπει να προβληθεί στη διεπαφή χρήστη.

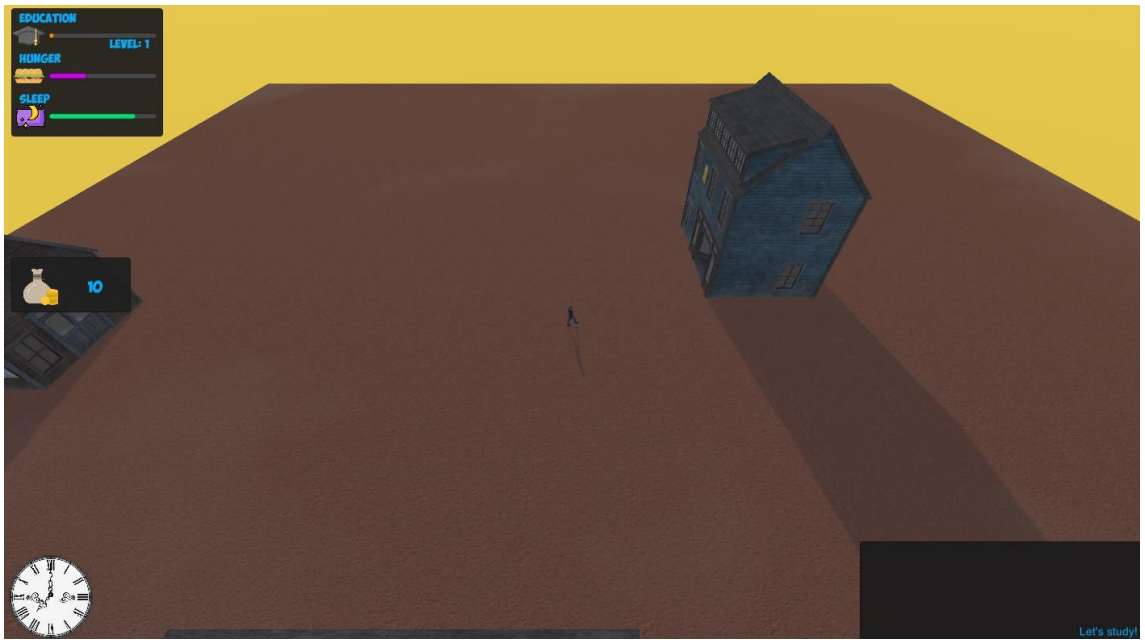
```
private void CalculateWork(int work, string message)
{
    totalWorkingHours = work;
    salary = totalWorkingHours * 3;
    coins.text = (int.Parse(coins.text) + salary).ToString();

    counters.workedCounter += totalWorkingHours;
    counters.workedMoneyEarnedCounter += salary;

    if (sendOnlyOnce)
    {
        chatBox.UpdateChatBox(message);
        sendOnlyOnce = false;
    }

    WorkController.work = false;

    if (hunger.hunger.value <= 50) //an eisai sth douleia kai peinas estw kai
ligo, ekmetalleusou to kai fae.
    {
        WorkController.eat = true;
        states.ImmediateChangeCurrentState(States.EAT);
        chatBox.UpdateChatBox(Constants.CHAT_BOX_WORK_TO_EAT);
        hunger.hungerNeedFulfilled = false;
        hunger.HungerLevel();
    }
    else if (sleep.sleep.value - 40 > 0)
    {
        // WorkController.eat = false;
        states.ImmediateChangeCurrentState(States.STUDY);
        chatBox.UpdateChatBox(Constants.CHAT_BOX_WORK_TO_STUDY);
    }
    else
    {
        // WorkController.eat = false;
        states.ImmediateChangeCurrentState(States.REST);
        chatBox.UpdateChatBox(Constants.CHAT_BOX_WORK_TO_SLEEP);
    }
}

private int GetRandomValue()
{
    float rand = Random.value;
    if (rand <= .3f)
        return  0;
    if (rand <= .6f)
        return 1;
    return rand <= .8f ? 2 : 3;
}
```

ΧΡΟΝΟΣ	ΜΗΝΥΜΑ-ΣΚΕΨΗ ΕΝΗΜΕΡΩΣΗΣ ΓΙΑ ΩΡΑ ΑΠΟΧΩΡΗΣΗΣ ΑΠ' ΤΗ ΔΟΥΛΕΙΑ
22:00	Left on time. No extra working hours today.

23:00	One-hour overtime and more coins in my pocket.
24:00	Two hours overtime! That was intense!
1:00	Three hours??? Wow. At least I got paid.

4.3.10. ΠΑΡΑΔΕΙΓΜΑ ΠΡΟΣΟΜΟΙΩΣΗΣ

Στις παρακάτω εικόνες παρατίθεται ένα παράδειγμα της προσομοίωσης.

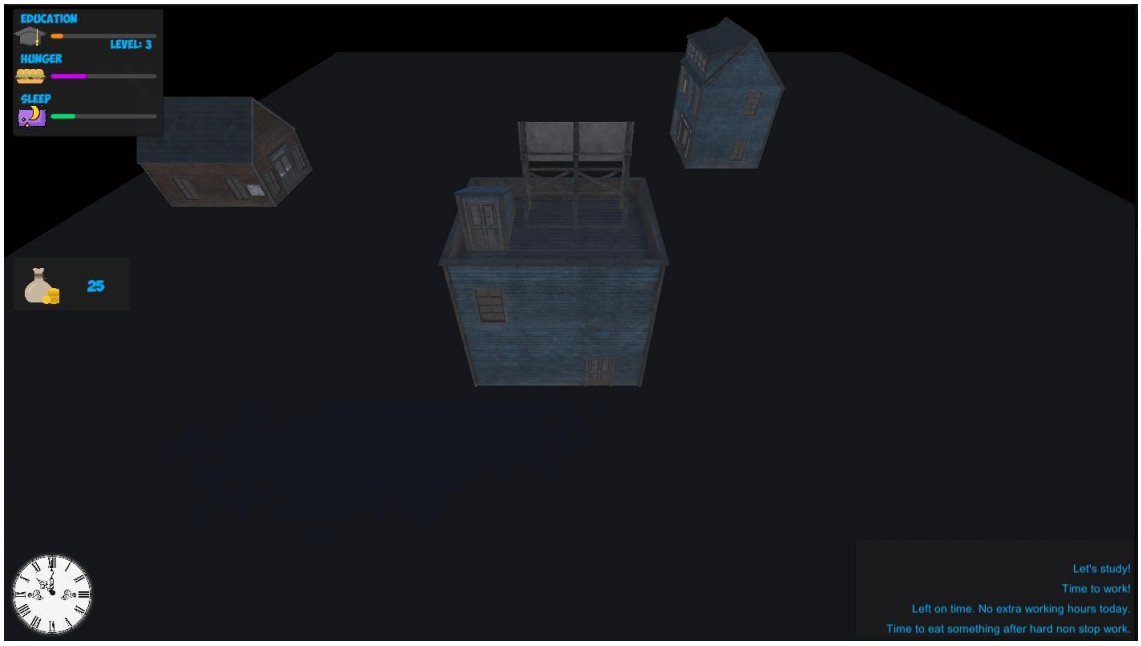


Εικόνα 31 Ο Joe ξεκινάει την ημέρα του

Στην αρχή της προσομοίωσης ο Joe βρίσκεται εκτός κτιρίων, στη μέση της σκηνής. Η πρώτη κατάσταση στην οποία βρίσκεται είναι η κατάσταση του διαβάσματος. Οπότε στέλνεται το κατάλληλο μήνυμα ενημέρωσης κάτω δεξιά και κατευθύνεται προς το κτίριο στα δεξιά για να μελετήσει.

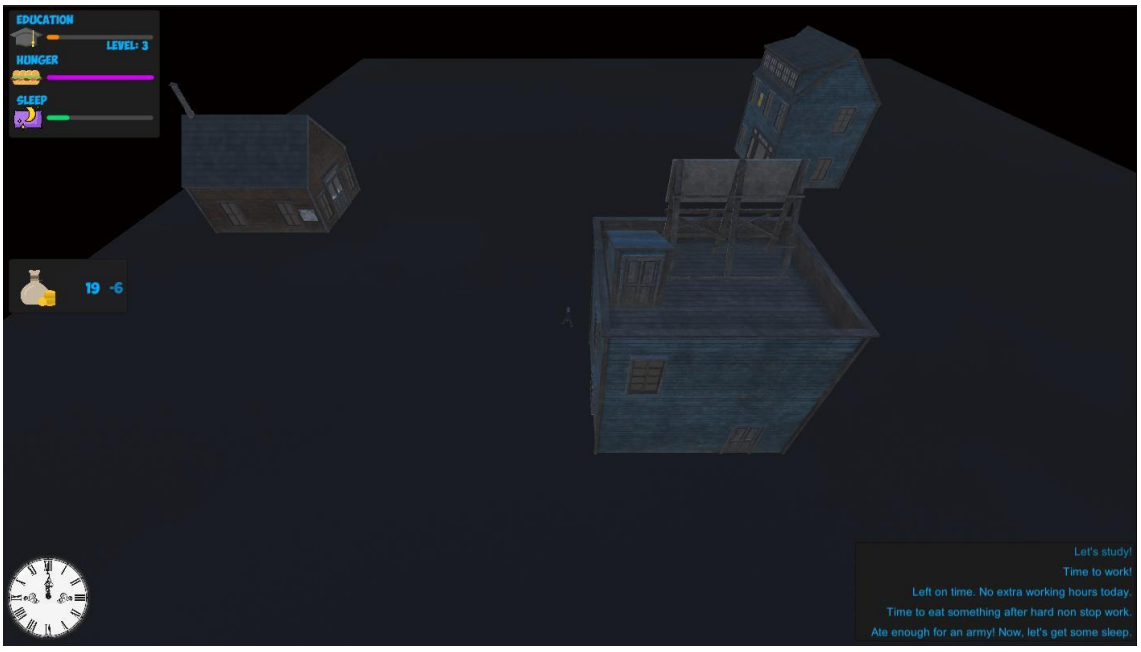
Εικόνα 32 Ο Joe πηγαίνει στη δουλειά

Εδώ παρατηρείται ο Joe στις 4-5 το μεσημέρι να κατευθύνεται προς το χώρο εργασίας του. Το ανάλογο μήνυμα δίνεται στο δέκτη από το παράθυρο μηνυμάτων.



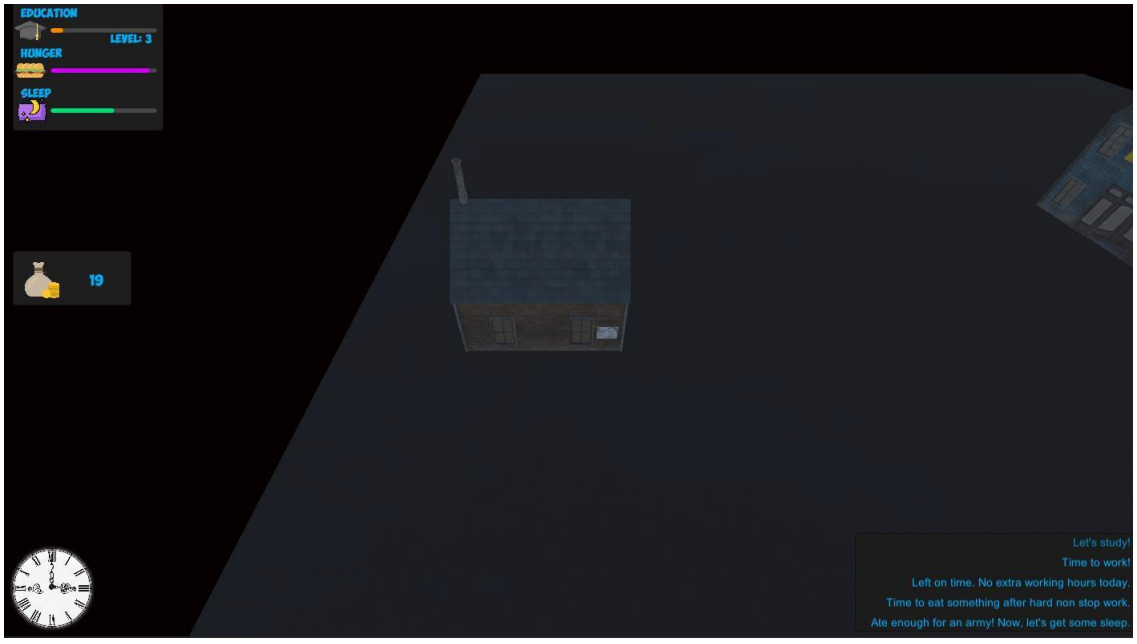
Εικόνα 33 Ο Joe τρώει μετά τη δουλειά

Ο Joe μόλις έχει τελειώσει με τη δουλειά του. Αμέσως μετά αποφασίζει να φάει κάτι μιας και πεινάει, αλλά παράλληλα επειδή βρίσκεται στο χώρο που σητίζεται.



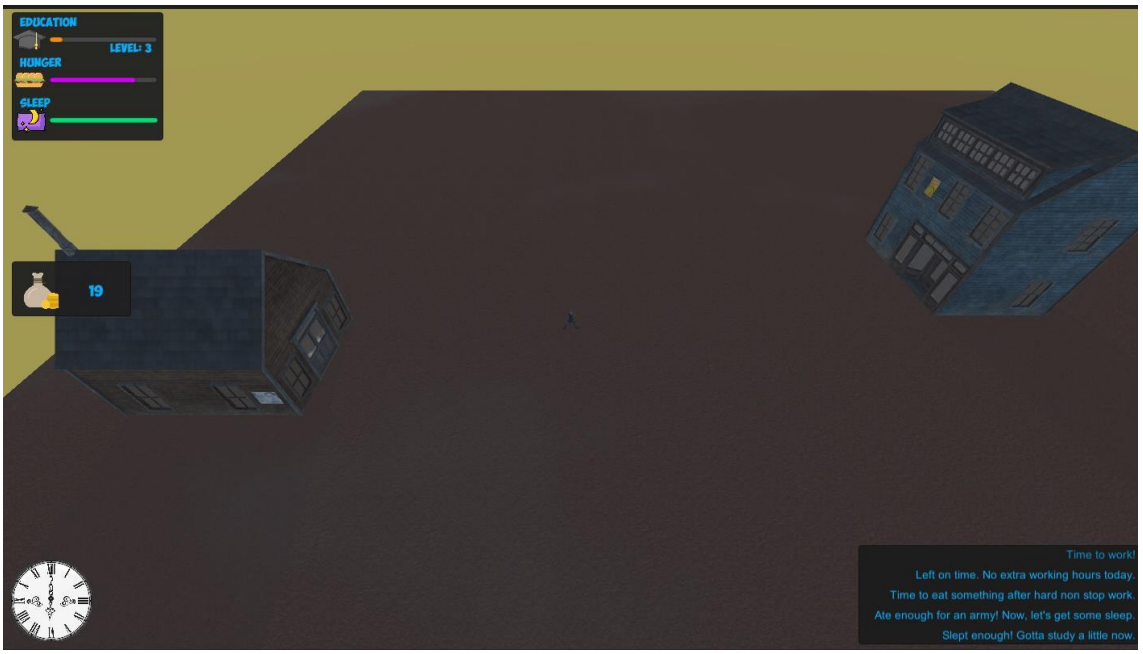
Εικόνα 34 Ο Joe κατευθύνεται στο σπίτι για να κοιμηθεί

Στο σημείο αυτό και αφού η μπάρα πείνας έχει γεμίσει, ο Joe κατευθύνεται προς το σπίτι του. Παράλληλα του αφαιρούνται 6 νομίσματα από τα 25 που είχε, επειδή έφαγε.



Εικόνα 35 Ο Joe κοιμάται και η μπάρα ύπνου γεμίζει

Ο Joe στις 3 το βράδυ πλέον και μετά τη δουλειά και το φαγητό αποφασίζει να κοιμηθεί. Σταδιακά παρατηρείται η μπάρα του ύπνου να γεμίζει.



Εικόνα 36 Η μέρα του Joe ξεκινά από την αρχή

Η μέρα του Joe ξεκινάει πάλι και κατευθύνεται προς το Εκπαιδευτικό Ίδρυμα για να αυξήσει την εμπειρία του και να ανέβει επίπεδο. Η παραπάνω διαδικασία θα επαναληφθεί μέχρι να φτάσει το επίπεδο 5 που είναι και το τελευταίο, σηματοδοτώντας την απόκτηση του πτυχίου του.

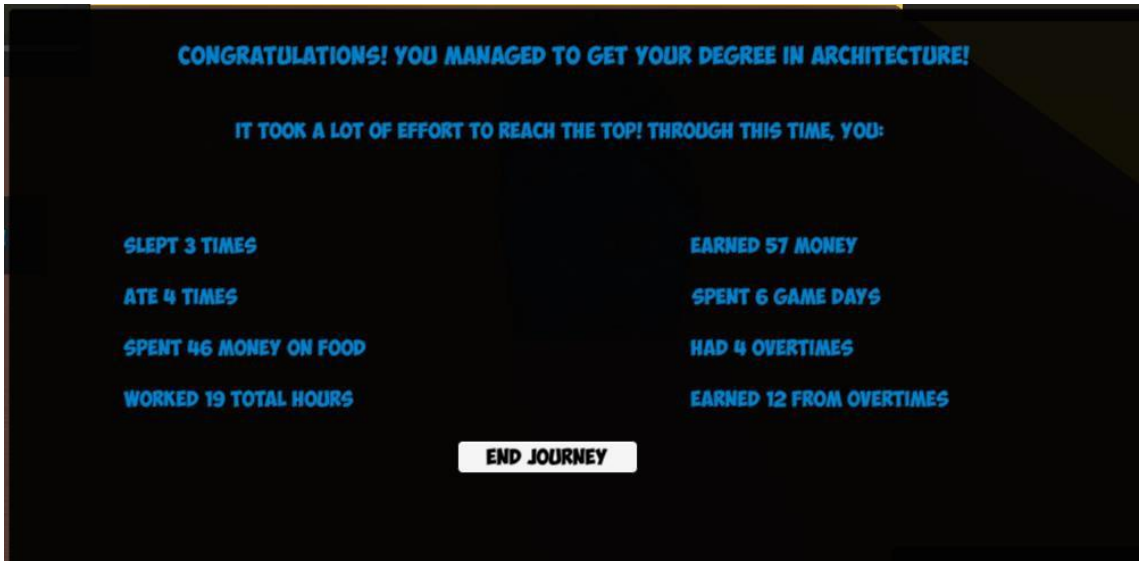
5. ΣΥΜΠΕΡΑΣΜΑΤΑ-ΠΕΡΙΛΗΨΗ

Το τέλος της προσομοίωσης πέρα από ένα μήνυμα σχετικά με την επιτυχία του χαρακτήρα να πάρει το πτυχίο που ήθελε και μοιραία να ολοκληρώσει το στόχο που είχε θέσει στην πυραμίδα του Μάσλοου, παρουσιάζει και μια σειρά στατιστικών. Αυτό συμβαίνει για να γίνει κατανοητό κατά πόσο είναι ρεαλιστικά τα αποτελέσματα με βάση τα δεδομένα που έχουν δοθεί.

Τα αποτελέσματα αυτά αφορούν τις παρακάτω πληροφορίες:

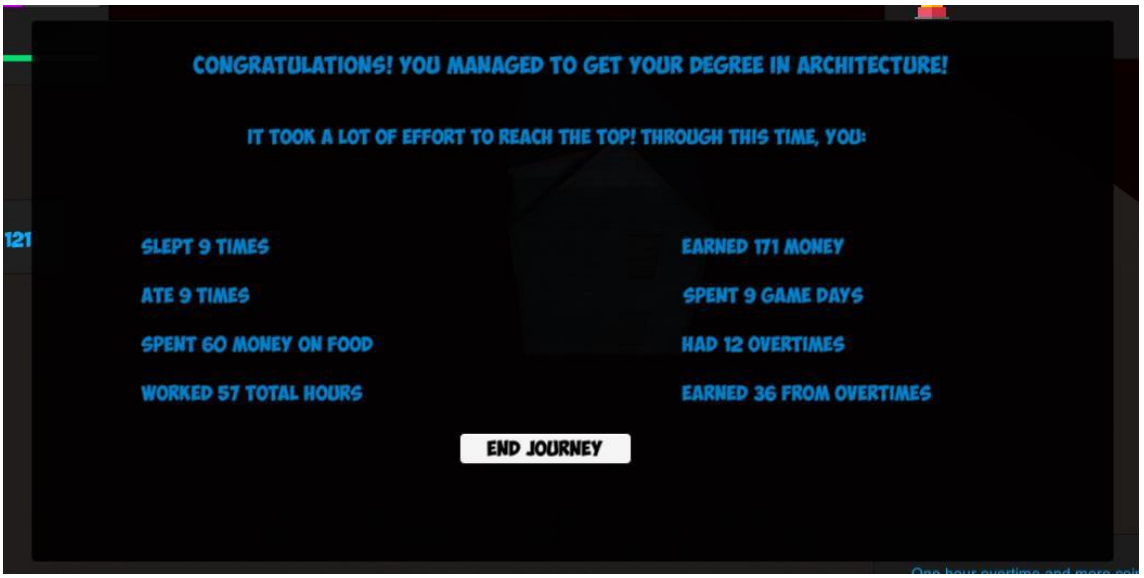
- Πόσες φορές κοιμήθηκε ο χαρακτήρας
- Πόσες φορές έφαγε
- Πόσα χρήματα ξόδεψε συνολικά για φαγητό
- Πόσες ώρες εργάστηκε συνολικά μαζί με τις υπερωρίες που μπορεί να προέκυψαν
- Πόσα χρήματα κέρδισε
- Πόσες μέρες χρειάστηκε για να ολοκληρώσει την προσομοίωση
- Πόσες συνολικά ώρες χρειάστηκε να μείνει παραπάνω στη δουλειά
- Πόσα χρήματα κέρδισε συνολικά από αυτές τις παραπάνω ώρες

Για παράδειγμα στην παρακάτω εικόνα είναι εμφανές ότι οι μεταβλητές που ρυθμίζουν τον ύπνο και την πείνα του χαρακτήρα δεν είναι σωστές. Στην πραγματικότητα και στην καθημερινή ζωή ενός ενήλικα ανθρώπου, το σωστό είναι να τρέφεται και να κοιμάται πάνω από 6 φορές σε 6 μέρες. Οπότε έπρεπε να ρυθμιστεί η μείωση των τιμών στις μπάρες ύπνου και τροφής.

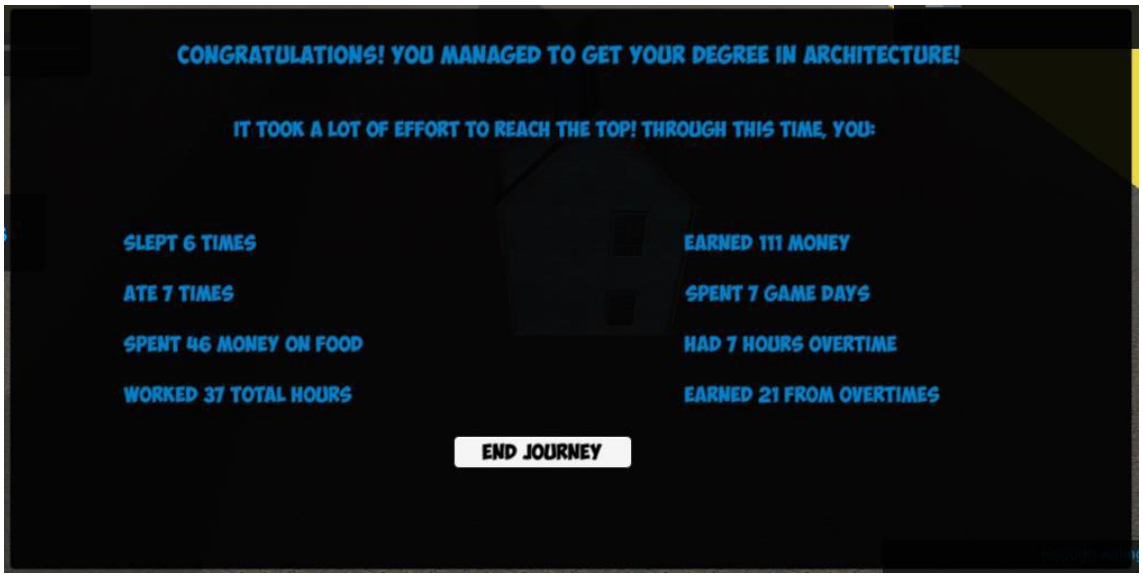


Εικόνα 37 Εσφαλμένα αποτελέσματα στο τέλος παιχνιδιού

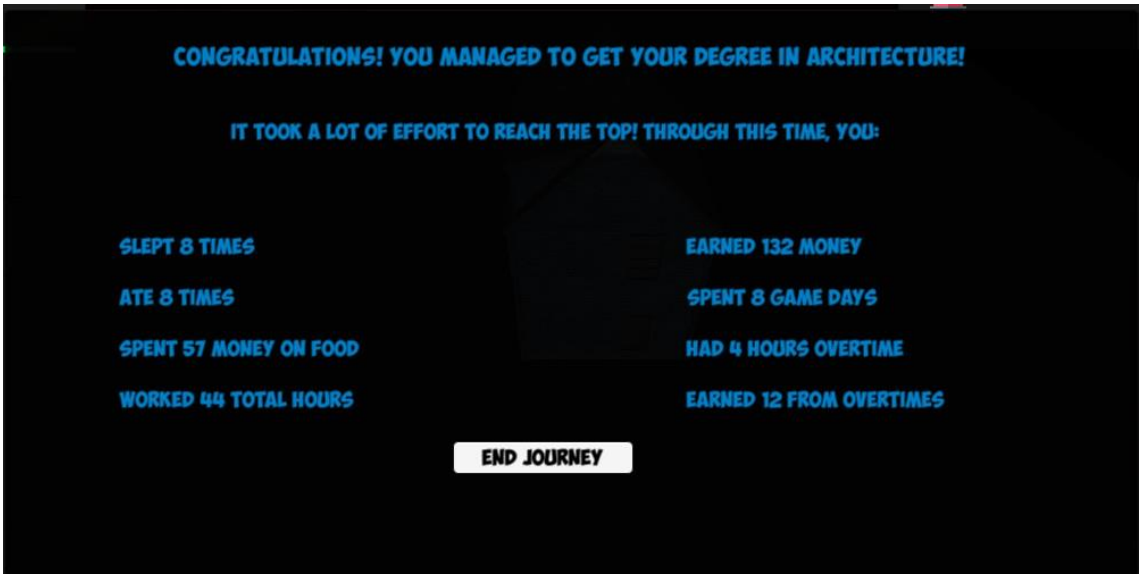
Στην παρακάτω εικόνα έγινε μια καλύτερη αξιολόγηση της κατάστασης, με αποτέλεσμα ο χαρακτήρας να έχει μια πιο ισορροπημένη ζωή και να κοιμάται και να τρώει 9 φορές μέσα σε 9 ημέρες. Τα χρήματα που κέρδισε επίσης, είναι αρκετά περισσότερα από αυτά που ξόδεψε για φαγητό. Στη συγκεκριμένη προσομοίωση έτυχε να κάνει και αρκετές υπερωρίες. Τα κείμενα σε αρκετές περιπτώσεις δεν είναι ξεκάθαρα, οπότε έπρεπε να διορθωθούν λίγο για να είναι πιο κατανοητά από τον εξωτερικό παρατηρητή.



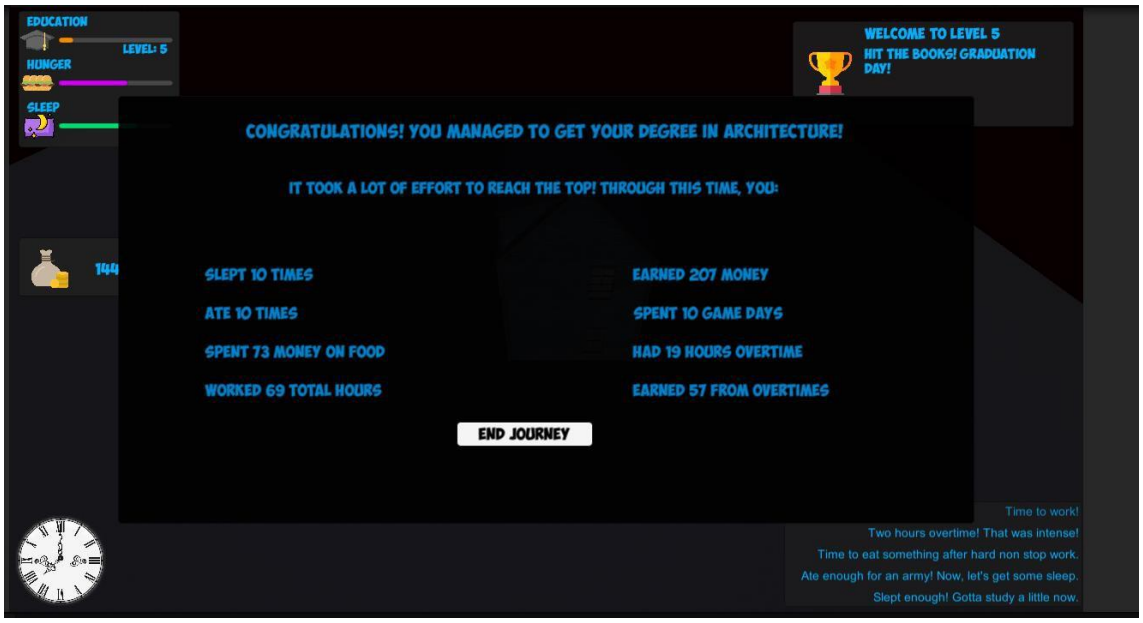
Εικόνα 38 Δεύτερη προσομοίωση



Εικόνα 39 Τρίτη προσομοίωση



Εικόνα 40 Τέταρτη προσομοίωση



Εικόνα 41 Πέμπτη προσομοίωση

6. ΠΙΘΑΝΕΣ ΕΠΕΚΤΑΣΕΙΣ

Οι βάσεις της παρούσας εργασίας είναι αρκετά καλές με αποτέλεσμα να μπορεί να αποτελέσει θεμέλιο για αρκετές άλλες εφαρμογές. Ο μηχανισμός διαχείρισης επιπέδων είτε με μετατροπές είτε χωρίς της συνάρτησης υπολογισμού επόμενου επιπέδου είναι άρτιος και μπορεί να χρησιμοποιηθεί σε οποιαδήποτε νέα εφαρμογή χρειαστεί.

6.1. ΠΑΙΧΝΙΔΙΑ ΡΟΛΩΝ

Μια πιθανή προέκταση της εργασίας θα ήταν ένα παιχνίδι Ρόλων, όπου ένας χαρακτήρας ολοκληρώνοντας κάποιες αποστολές θα μπορούσε να κερδίζει εμπειρία. Με την απόκτηση εμπειρίας θα οδηγούταν σε νέα επίπεδα. Στα επίπεδα αυτά θα αποκτούσε περισσότερη δύναμη, αλλάζοντας τα στατιστικά της επίθεσης και άμυνας του. Παράλληλα θα μπορούσαν να υπάρχουν εχθροί, πολίτες που βρίσκονται σε κίνδυνο ή που απλά εκτελούν τις καθημερινές τους εργασίες. Οι αποφάσεις και οι αλληλεπιδράσεις με αυτά τα άτομα θα μπορούσαν να επιφέρουν διαφορετικά αποτελέσματα στο τέλος της ιστορίας και του παιχνιδιού.

6.2. ΠΑΙΧΝΙΔΙΑ ΠΡΟΣΟΜΟΙΩΣΗΣ

Μια άλλη πιθανή προέκταση θα ήταν η χρήση του μηχανισμού αλλαγής καταστάσεων σε άλλες εφαρμογές όπως για παράδειγμα σε μια εφαρμογή που οι χαρακτήρες κατασκευάζουν σταδιακά μια πόλη. Θα μπορούσαν να υπάρχουν διαφορετικές ομάδες ατόμων που ασχολούνται με αγροτικές εργασίες, εργασίες κοπής ξύλου ή κατασκευής κτιρίων. Έτσι παράλληλα με την έρευνα για την πυραμίδα του Μάσλοου θα μπορούσε να δημιουργηθεί μια κοινωνία και να ερευνηθούν σε βάθος οι ανάγκες συνολικά και όχι ατομικά.

7. ASSETS

7.1. UNITY ASSETS

- Building Kit 1 by **MOJO STRUCTURE**
  https://assetstore.unity.com/packages/3d/environments/industrial/building-kit-1-68179
- Time of Day & Weather System by **Tobias Johansson**
  https://assetstore.unity.com/packages/tools/particles-effects/time-of-day-weather-system-40374
- Achievements SFX FREE **by B.G.M.**
  https://assetstore.unity.com/packages/audio/sound-fx/achievement-sfx-free-91639

7.2. MIXAMO

Character Joe, https://www.mixamo.com/#/?page=1&query=joe&type=Character

8. ΒΙΒΛΙΟΓΡΑΦΙΑ

1. Wikipedia. ***https://en.wikipedia.org/wiki/Abraham_Maslow.*** **[Ηλεκτρονικό]**

2. **Simulation Video Game.** ***https://en.wikipedia.org/wiki/Simulation_video_game.*** **[Ηλεκτρονικό]**

3. Sundays Sundae. ***https ://sundaysundae.co/unity-vs-unreal/.*** **[Ηλεκτρονικό]**

4. Μηχανή του Χρόνου. ***https://www.mixanitouxronou.gr/pia-anagki-ine-i-pio-simantiki-gia-kathe-anthropo-ti-lei-i-perifimi-piramida-tou-psichologou-maslow-pou-tin-axiopiisan- i-manatzer-gia-na-dinoun-bonous-ke-se-idos/.*** **[Ηλεκτρονικό]**

5. Player Level Up | C# Tutorials in Unity.
https://www.youtube.com/watch?v=BqoWo7GTM8E. **[Ηλεκτρονικό]**

6. Unity 5 health bar tutorial - Lerp values.
https://www.youtube.com/watch?v=Wx9TgWl4LAU. **[Ηλεκτρονικό]**

7. Unity 3D - Make a Basic AI State Machine. ***https://www.youtube.com/watch?v=PaLD1t-klwM.*** **[Ηλεκτρονικό]**

8. How to make a Clock in the UI (Unity Tutorial).
https://www.youtube.com/watch?v=pbTysQw-WNs&t=36s. **[Ηλεκτρονικό]**

9. ***HOW TO MAKE IN-GAME ACHIEVEMENTS FOR YOUR GAME IN UNITY #01 - COLLECTION BASED.*** **https://www.youtube.com/watch?v=-XuzwxkZ2Wk[Ηλεκτρονικό]**