ΠΑΝΕΠΙΣΤΗΜΙΟ ΠΕΙΡΑΙΩΣ – ΤΜΗΜΑ ΠΛΗΡΟΦΟΡΙΚΗΣ

Πρόγραμμα Μεταπτυχιακών Σπουδών

«Προηγμένα Συστήματα Πληροφορικής»

Μεταπτυχιακή Διατριβή

Τίτλος Διατριβής	**Διαδικαστική δημιουργία πόλης, προσομοίωση κοινωνίας και χρήση του μοντέλου OCEAN σε Unity3D** **Procedural town generation, simulation of society and implementation of the OCEAN Model in Unity3D**
Ονοματεπώνυμο Φοιτητή	**Γεώργιος Πολυχρόνης**
Πατρώνυμο	**Αναστάσιος**
Αριθμός Μητρώου	**ΜΠΣΠ/17075**
Επιβλέπων	**Παναγιωτόπουλος Θεμιστοκλής, Καθηγητής**

Ημερομηνία Παράδοσης **Δεκέμβριος 2022**

Τριμελής Εξεταστική Επιτροπή

Θεμιστοκλής Παναγιωτόπουλος Καθηγητής	Ιωάννης Τασούλας Επίκουρος Καθηγητής	Άγγελος Πικράκης Επίκουρος Καθηγητής

Πίνακας Περιεχομένων

Πίνακας Εικόνων

ΠΕΡΙΛΗΨΗ

Η παρούσα διπλωματική εργασία έχει στόχο τη κατασκευή μιας πόλης η οποία έχει δημιουργηθεί διαδικαστικά με τη χρήση δυναμικών μεθόδων κατασκευής. Με την βοήθεια της πλατφόρμας ανάπτυξης Unity3D, θα γίνει η προσομοίωση των λειτουργιών μιας πραγματικής πόλης, τόσο σε ατομικό επίπεδο, με την επεξήγηση και την υλοποίηση των αναγκών ενός ανθρώπου όπως τροφή και στέγη, των διαδικασιών που ακολουθεί σε καθημερινή βάση, όπως μετακίνηση προς και από την εργασία ή την οικία, καθώς και σε κοινωνικό επίπεδο, με την κατασκευή της έννοιας «οικογένεια» προγραμματιστικά για τη δημιουργία επαφών μεταξύ των μελών της οικογένειας ή την δημιουργία σχέσεων σε επαγγελματικό χώρο, όσο και την εισαγωγή κατάλληλων μοντέλων χαρακτηριστικών ενός ανθρώπου όπως το μοντέλο OCEAN, για την ακριβή αναπαράσταση των αλληλεπιδράσεων μεταξύ των ατόμων στη πόλη.

Η διπλωματική εργασία υλοποιείται για το Πρόγραμμα Μεταπτυχιακών Σπουδών του Πανεπιστημίου Πειραιώς.

ABSTRACT

This thesis' aim is the construction of a procedurally generated city using dynamic construction methods. With the help of the Unity3D development platform, a realistic city will be created with the use of simulative methods, at an individual level, with the explanation and implementation of a person's needs such as food and shelter, the procedures he/she follows on a daily basis, such as movement to and from work or home, as well as at the social level, by constructing the concept of "family" programmatically to create bonds between family members or create relationships in a professional area, as well as introducing appropriate scientific models of a person's characteristics such as OCEAN model, to accurately represent the interactions between individuals in the city.

The thesis is being implemented for the Master's Program of the University of Piraeus.

1. ΕΙΣΑΓΩΓΗ

1.1. Γενικά

Για να καταστεί πραγματοποιήσιμη η υλοποίηση του στόχου που αναφέρεται στον τίτλο της διπλωματικής εργασίας, χρειάζεται να γίνει η απαραίτητη προετοιμασία σε θεωρητικό επίπεδο ώστε να κατανοηθούν πλήρως οι διαδικασίες και οι ορολογίες, όπως και να γίνει η επεξήγηση των διαφόρων εργαλείων και μέσων υλοποίησης που θα χρησιμοποιηθούν. Παρακάτω θα γίνει η αναφορά και η επεξήγηση των βασικών ορισμών ώστε να κατασκευαστεί μια πόλη διαδικαστικά με τη χρήση κατάλληλα διαμορφωμένων μεθόδων και να γίνει η προσομοίωση μιας κοινωνίας.

1.2. Διαδικαστική δημιουργία

Ο όρος «Διαδικαστική Δημιουργία» είναι απευθείας μετάφραση του αγγλικού όρου «Procedural Generation». Διαχωρίζοντας τον όρο, η λέξη «Διαδικαστική» περιγράφει τη διεργασία που υπολογίζει μια ορισμένη συνάρτηση, ενώ η «Δημιουργία» την υλοποίηση της. Συνδυαστικά, ο όρος περιγράφει μία ορισμένη μέθοδο η οποία δημιουργεί δεδομένα για μία συγκεκριμένη λειτουργία. Βρίσκει ευρεία χρήση στον τομέα της Πληροφορικής, ιδιαίτερα στον τομέα των βιντεοπαιχνιδιών, όπου συνδυάζοντας στοιχεία ή μοντέλα και ανάλογους αλγορίθμους μαζί με υλοποίηση μιας γεννήτριας τυχαιότητας, γίνεται εφικτή η παραγωγή μοντέλων και δεδομένων με παραλλαγές και τυχαίες αποκλείσεις ή μεταβολές των μοντέλων, δίνοντας έτσι την «ψευδαίσθηση» της διαφορετικότητας σε έναν χώρο ή την δημιουργία τυχαίων συμβάντων, ενώ παράλληλα ελαχιστοποιεί την ανάγκη δημιουργίας νέων μοντέλων από ανθρώπους, μειώνοντας αποτελεσματικά το συνολικό χρόνο υλοποίησης καθώς και το συνολικό χώρο που καταλαμβάνει μια υλοποίηση επειδή χρησιμοποιείται η ίδια βάση. Η παραγωγή ονομάζεται «Δυναμική παραγωγή», ενώ το αποτέλεσμα με αυτή η διαδικασία ονομάζεται «Δυναμικά παραγόμενα μοντέλα». Ο όρος συχνά συγχέεται με τον όρο «Δυναμικός Προγραμματισμός» ο οποίος αναφέρεται στην υπολογιστική μέθοδο επίλυσης προβλημάτων.

Εικόνα 1: διαδικαστική δημιουργία δένδρων, en.wikipedia.org/wiki/Procedural_generation

Για την περίπτωση της διπλωματικής εργασίας, η χρήση της «Διαδικαστικής Δημιουργίας» αποσκοπεί στην κατασκευή του κόσμου – πόλης, των ανθρωπόμορφων χαρακτήρων, των χαρακτηριστικών του μοντέλου OCEAN το οποίο θα αναλύσουμε παρακάτω, καθώς και το σύστημα πλοήγησης των χαρακτήρων στην πόλη.

1.3. Η έννοια της προσομοίωσης

Ο όρος «Προσομοίωση» χαρακτηρίζει την αναπαράσταση λειτουργίας ενός συστήματος ή μιας διαδικασίας. Χρησιμοποιώντας μοντέλα τα οποία περιγράφουν και αποτυπώνουν βασικά χαρακτηριστικά και δεδομένα καθώς και συμπεριφορά, παράγουν αποτελέσματα για το συγκεκριμένο μοντέλο σε σχέση με την είσοδο των δεδομένων. Χρησιμοποιείται ευρέως στον τεχνολογικό τομέα για την ανάλυση, το συντονισμό και τη βελτιστοποίηση απόδοσης, την παραγωγή και τον έλεγχο συστημάτων ασφαλείας στον εργασιακό τομέα, την δοκιμή, την μάθηση και τα βιντεοπαιχνίδια. Επίσης, χρησιμοποιείται στον επιστημονικό τομέα για την μοντελοποίηση φυσικών ή τεχνητών συστημάτων ώστε να μελετηθεί η λειτουργία τους, όπως και στον οικονομικό τομέα με την παραγωγή αποτελεσμάτων στην οικονομία βραχυχρόνια ή μακροχρόνια.

Φυσικά, η προσομοίωση πέρα από λύσεις, περιέχει και προβλήματα. Η αποτυχία εύρεσης και χρήσης έγκυρων δεδομένων καθώς και η χρήση δεδομένων που είναι ημιτελή ή δεν διατάσσονται αρμονικά με τη λειτουργία του μοντέλου έχει ως αποτέλεσμα την παραγωγή λανθασμένων αποτελεσμάτων. Επίσης, ανεξάρτητα από την ποιότητα των δεδομένων, περιπτώσεις μη αποτελεσματικών αλγορίθμων ή αλγορίθμων που δεν καλύπτουν αποτελεσματικά το φάσμα των περιπτώσεων χρήσης ή η τυχόν αμέλεια των επιπτώσεων που παράγονται ταυτόχρονα, μπορούν να οδηγήσουν σε πραγματικά προβλήματα κατά την εφαρμογή των διαδικασιών σε περίπτωση που αυτά χρησιμοποιηθούν χωρίς το ανάλογο επανέλεγχο του μοντέλου μέσω διαδικασιών για την επαλήθευση και επικύρωση του αρχικού μοντέλου.

Παρόμοιος όρος με την «Προσομοίωση» (Simulation) αποτελεί η «Εξομοίωση» (Emulation). Ειδικότερα, στον τομέα της Πληροφορικής, η προσομοίωση αποτελεί τον τρόπο δοκιμών λογισμικού για την παραγωγή αποτελεσμάτων και δεδομένων για την διόρθωση ή την βελτιστοποίηση συστημάτων χρησιμοποιώντας ένα περιβάλλον με μεταβλητές λογισμικού μέσα στο οποίο θα δοκιμαστεί το λογισμικό που έχει ορίσει ο προγραμματιστής – μηχανικός λογισμικού. Ο εξομοιωτής επιχειρεί να μιμηθεί το τελικό σύστημα που θα καταλήξει μελλοντικά το δοκιμαστικό λογισμικό σε επίπεδο υλικού. Συνήθως, ένας εξομοιωτής γράφεται σε γλώσσα Assembly, σε αντίθεση με τους προσομοιωτές που λειτουργούν χωρίς την μίμηση υλικού και μπορούν να γραφτούν σε υψηλού επιπέδου γλώσσες προγραμματισμού.

1.4. Το μοντέλο OCEAN

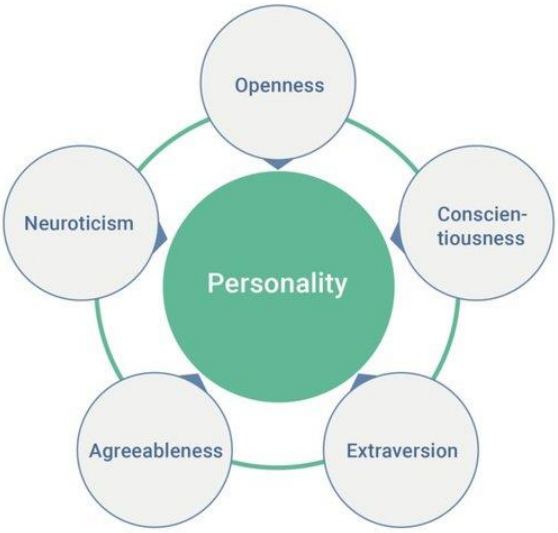
Το μοντέλο OCEAN αποτελεί προϊόν ψυχολογίας για την περιγραφή χαρακτηριστικών της προσωπικότητας. Συναντάται και με την ονομασία “Big Five Traits” και δημιουργήθηκε για την κατανόηση της σχέσης της προσωπικότητας με την ακαδημαϊκή ικανότητα και συμπεριφορά. Ορίστηκε από ανεξάρτητα σύνολα ερευνητών που χρησιμοποίησαν παραγοντική ανάλυση λεκτικών περιγραφικών παραγόντων της ανθρώπινης συμπεριφοράς. Η μελέτη ξεκίνησε από τις σχέσεις μεταξύ ενός συνόλου λεκτικών περιγραφών, σχετιζόμενες με διακριτά χαρακτηριστικά προσωπικότητας. Στη συνέχεια έγινε μείωση του πλήθους κατά 5-10 φορές και χρησιμοποιήθηκε η παραγοντική ανάλυση για την ομαδοποίηση των χαρακτηριστικών με ερωτηματολόγια αυτοαναφοράς και αντίστοιχη αξιολόγηση, προκειμένου να βρουν τους υποκείμενους παράγοντες της προσωπικότητας.

Το αρχικό μοντέλο προωθήθηκε από τον Ernest Tupes και τον Raymond Christal το 1961, αλλά απέτυχε στην προσέγγιση της ακαδημαϊκής κοινότητας μέχρι τη δεκαετία του 1980. Το 1990 ο J.M. Digman προώθησε το μοντέλο προσωπικότητας πέντε παραγόντων του (Five Figure Model), το οποίο ο Lewis Goldberg επέκτεινε στο υψηλότερο επίπεδο οργάνωσης. Αυτά τα πέντε γενικά πεδία έχουν βρεθεί ότι περιέχουν και υποκαθιστούν τα πιο γνωστά χαρακτηριστικά προσωπικότητας και θεωρείται ότι αντιπροσωπεύουν τη βασική δομή για όλα τα χαρακτηριστικά της προσωπικότητας.

Για την λεξιλογική υπόθεση στη θεωρία της προσωπικότητας εργάστηκαν τουλάχιστον τέσσερα σύνολα ερευνητών σε ανεξάρτητο βαθμό σε εύρος δεκαετιών και εντόπισαν γενικά τους ίδιους πέντε παράγοντες. Αρχικά οι Tupes και Christal, στη συνέχεια ο Goldberg στο Ερευνητικό Ινστιτούτο του Όρεγκον,, ο Cattell στο Πανεπιστήμιο του Ιλινόις, και οι Costa και McCrae. Αυτά τα τέσσερα σύνολα ερευνητών, χρησιμοποίησαν διαφορετικές μεθόδους για να καταλήξουν στα πέντε βασικά χαρακτηριστικά, με αποτέλεσμα να υπάρχουν διαφοροποιήσεις στην ονομασία και τον ορισμό του κάθε χαρακτηριστικό. Ωστόσο, όλα έχουν βρεθεί ότι αλληλοσχετίζονται μεταξύ τους σε μεγάλο βαθμό.

Κάθε ένα από τα πέντε βασικά χαρακτηριστικά περιέχει δύο έννοιες άρρητα συνδεδεμένες μεταξύ τους, οι οποίες αντιπροσωπεύουν τα άκρα του βασικού χαρακτηριστικού. Επιπλέον, βαθμολογία ενός χαρακτηριστικού κοντά στο ουδέτερο φαίνεται να ορίζει τον χαρακτήρα του ατόμου κοντά στο «Αδιάφορο», σε σχέση πάντα με τον ορισμό του χαρακτηριστικού, με αποτέλεσμα άτομα με αυτά τα χαρακτηριστικά να μπορούν να οριστούν ως προσαρμοστικά, μέτρια και λογικά ή ακόμη και ανεξήγητα, δυσκολονόητα και υπολογιστικά. Αναφορικά, αυτά είναι:

- Openness to experience (Intellect – Openness)
- Conscientiousness (Industriousness – Orderliness)
- Extraversion (Enthusiasm – Assertiveness)
- Agreeableness (Compassion – Politeness)
- Neuroticism (Volatility – Withdrawal)



Εικόνα 2: Τα πέντε χαρακτηριστικά

Πιο συγκεκριμένα, το κάθε χαρακτηριστικό έχει τους παρακάτω ορισμούς:

Openness to experience

Το χαρακτηριστικό «Openness» ορίζει τη γενική εκτίμηση για την τέχνη, το συναίσθημα, την περιπέτεια, τις ασυνήθιστες ιδέες, τη φαντασία και την ποικιλία στην απόκτηση εμπειριών. Οι άνθρωποι που είναι ανοικτοί στην εμπειρία είναι διανοητικά περίεργοι, ανοικτοί στα συναισθήματα και πρόθυμοι να δοκιμάσουν νέα πράγματα. Τείνουν να είναι πιο δημιουργικοί και πιο πιθανό να έχουν αντισυμβατικές πεποιθήσεις, ενώ υπάρχει μεγαλύτερη πιθανότητα να εμφανίσουν επικίνδυνη συμπεριφορά επιδιώκοντας την αυτοπραγμάτωση με έντονες εμπειρίες. Αντιθέτως, εκείνοι με χαμηλή διαφάνεια επιδιώκουν να κερδίσουν την ολοκλήρωση μέσω της επιμονής, χαρακτηρίζονται πραγματιστές και βασισμένοι στα δεδομένα, ενώ θεωρούνται δογματικοί και κλειστοί. Παραμένει κάποια διαφωνία σχετικά με το πώς να ερμηνευθεί και να ενσωματωθεί ο

παράγοντας της διαφάνειας, καθώς υπάρχει έλλειψη βιολογικής υποστήριξης για το συγκεκριμένο χαρακτηριστικό. Η «Ανοιχτότητα» δεν έχει δείξει σημαντική συσχέτιση σε κάποια περιοχή του εγκεφάλου σε αντίθεση με τα υπόλοιπα χαρακτηριστικά. Πιθανές προτάσεις οι οποίες χρησιμοποιούνται για το χαρακτηριστικό είναι: «Έχω υπερβολικά ζωντανή φαντασία», «είμαι γρήγορος στην κατανόηση», «χρησιμοποιώ δύσκολες λέξεις», «έχω πολλές ιδέες». Αντίστροφα, για το άλλο άκρο, χρησιμοποιούνται προτάσεις όπως: «έχω δυσκολία στην κατανόηση αφηρημένων ιδεών» ή «δεν έχω καλή/δυνατή φαντασία».

Conscientiousness

Ως «Conscientiousness» ορίζεται η τάση των ανθρώπων να επιδεικνύουν αυτοπειθαρχία, να ενεργούν με υπευθυνότητα και να επιδιώκουν την επίτευξη έναντι μέτρων ή εξωτερικών προσδοκιών. Σχετίζεται με τον τρόπο με τον οποίο οι άνθρωποι ελέγχουν, ρυθμίζουν και κατευθύνουν τις παρορμήσεις τους. Η υψηλή «Ευσυνειδησία» συχνά εκλαμβάνεται ως πεισματάρης και συγκεντρωμένος. Η χαμηλή ευσυνειδησία συνδέεται με την ευελιξία και τον αυθορμητισμό, αλλά μπορεί επίσης να εμφανιστεί ως προχειρότητα ή έλλειψη αξιοπιστίας. Ανάλογα με τον χαρακτήρα, το χαρακτηριστικό εμφανίζεται να περιγράφει την προτίμηση για προγραμματισμένη και όχι αυθόρμητη συμπεριφορά. Πρόκειται για ένα χαρακτηριστικό το οποίο έχει γενικά αυξημένο μέσο επίπεδο σε ηλικίες του εύρους νεαρών ενηλίκων και στη συνέχεια μειώνεται μεταξύ του ηλικιακού εύρους των ηλικιωμένων, με την αντίστοιχη αύξηση της «Εργατικότητας». Πιθανές προτάσεις οι οποίες χρησιμοποιούνται για το χαρακτηριστικό είναι: «Είμαι πάντα προετοιμασμένος», «Προσέχω τις λεπτομέρειες σε μεγάλο βαθμό», «Μου αρέσει η σειρά», «Ακολουθώ πάντα πρόγραμμα» κ.α.. Αντίστροφα, «αφήνω τα πράγματα μου τριγύρω», «Συνήθως ξεχνάω να βάλω τα πράγματα μου στη θέση που ήταν» κ.α..

Extraversion

Ως «Extraversion» χαρακτηρίζεται το εύρος των δραστηριοτήτων σε σχέση με το βάθος τους, την έξαρση από εξωτερικές δραστηριότητες και τη δημιουργία ενέργειας με εξωγενείς τρόπους. Το χαρακτηριστικό ορίζεται από την έντονη δέσμευση με τον εξωτερικό κόσμο. Οι εξωστρεφείς απολαμβάνουν την αλληλεπίδραση τους με τους ανθρώπους και συχνά θεωρούνται γεμάτοι ενέργεια. Τείνουν να είναι ενθουσιώδη άτομα, με προσανατολισμό στη δράση. Διαθέτουν υψηλή ομαδική ορατότητα, τους αρέσει να μιλάνε και να διεκδικούν τον εαυτό τους. Τα άτομα αυτά μπορεί να εμφανίζονται πιο κυρίαρχα σε κοινωνικά περιβάλλοντα, σε αντίθεση με τα εσωστρεφή άτομα. Τα εσωστρεφή άτομα έχουν χαμηλότερη κοινωνική δέσμευση σε σχέση με τα εξωστρεφή. Τείνουν να φαίνονται ήσυχοι, χαμηλών τόνων και λιγότερο εμπλεκόμενοι στο κοινωνικό περιβάλλον. Παρόλα αυτά, η έλλειψη κοινωνικής εμπλοκής δεν πρέπει να ερμηνεύεται ως ντροπαλότητα ή κατάθλιψη. Αντίθετα, ορίζονται πιο εύκολα ως “ανεξάρτητοι” από το κοινωνικό τους κόσμο. Χρειάζονται λιγότερη διέγερση και περισσότερο χρόνο μόνοι τους, χωρίς αυτό να σημαίνει ότι είναι εχθρικοί ή αντικοινωνικοί. Αντίθετα, είναι επιφυλακτικοί σε κοινωνικές καταστάσεις. Πιθανές προτάσεις οι οποίες χρησιμοποιούνται είναι: «Είμαι η ζωή του πάρτι», «Νιώθω άνετα ανάμεσα σε άλλους ανθρώπους», «μου αρέσει να ξεκινάω συνομιλίες», «δεν με πειράζει να είμαι το κέντρο της προσοχής». Αντίστροφα, για τους εσωστρεφείς: «Δε μιλάω πολύ», «δεν μου αρέσει να τραβάω τη προσοχή», «είμαι ήσυχος μπροστά σε αγνώστους».

Agreeableness

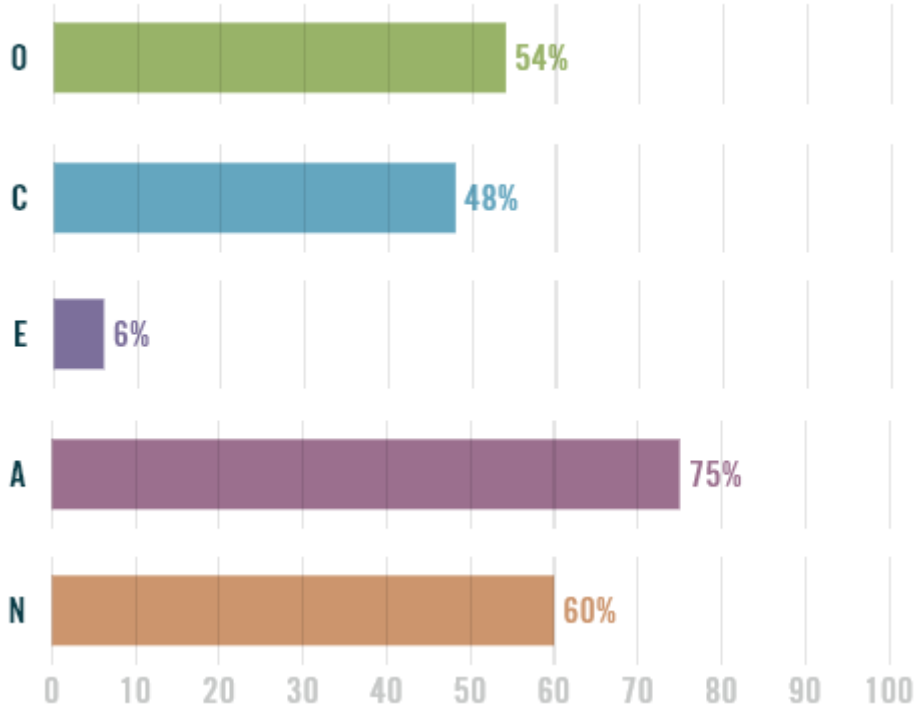
Το χαρακτηριστικό της «Τερπνότητας» αντανακλά τις ατομικές διαφορές στη γενική μέριμνα του πληθυσμού και την κοινωνική αρμονία. Τα ευγενικά (Politeness) άτομα εκτιμούν τη συνεννόηση με τους άλλους. Είναι γενικά διακριτικοί, ευγενικοί, γενναιόδωροι, έμπιστοι και εμπιστεύσιμοι, εξυπηρετικοί και πρόθυμοι να συμβιβάσουν τα συμφέροντα τους σε σχέση με τους γύρω τους. Οι ευγενικοί άνθρωποι έχουν επίσης μια αισιόδοξη άποψη για την ανθρώπινη φύση. Αντιθέτως, τα δυσάρεστα άτομα βάζουν το προσωπικό συμφέρον τους πάνω από την ικανότητα συνύπαρξης τους με τους άλλους. Γενικά, δεν ενδιαφέρονται για την ευημερία των άλλων και είναι λιγότερο πιθανό να αλληλοεπιδράσουν με άλλους ανθρώπους. Μερικές φορές, ο σκεπτικισμός τους για τα κίνητρα των άλλων τους καθιστά καχύποπτους, εχθρικούς και μη συνεργάσιμους, ενώ μπορούν να θεωρηθούν εριστικά ή αναξιόπιστα.

Η «Ευγένεια» είναι ένα κοινωνικό χαρακτηριστικό. Οι έρευνες έχουν δείξει ότι η συμφιλίωση κάποιου συσχετίζεται θετικά με την ποιότητα των σχέσεων με τα μέλη της ομάδας του, ενώ προβλέπουν την ανάδειξη των ατόμων σε θέσεις ηγετικού χαρακτήρα και την εμφανή ένδειξη παρόμοιων δεξιοτήτων. Αντίθετα, η Τερπνότητα έχει βρεθεί ότι σχετίζεται αρνητικά με τη συναλλακτική ηγεσία στον στρατό, σύμφωνα με μελέτη που έγινε σε ασιατικό στρατό, όπου βρέθηκε πως αρχηγοί με υψηλή βαθμολογία τερπνότητας είχαν χαμηλή βαθμολογία στις μετασχηματιστικές ηγετικές δεξιότητες. Πιθανές προτάσεις οι οποίες χρησιμοποιούνται είναι: «Με ενδιαφέρουν οι υπόλοιποι άνθρωποι», «νιώθω τα συναισθήματα των υπολοίπων», «κάνω τους γύρω μου να αισθάνονται άνετα». Αντίστροφα, άτομα με χαμηλή τερπνότητα συμφωνούν περισσότερο με τις προτάσεις: «Δεν με ενδιαφέρουν τα προβλήματα των άλλων», «Προσβάλλω τους γύρω μου συχνά», «Δεν μου προκαλούν ενδιαφέρον οι γύρω μου» κ.α..

Neuroticism

Το χαρακτηριστικό του «Νευρωτισμού» αναφέρεται στην τάση των ανθρώπων να βιώνουν αρνητικά συναισθήματα, όπως θυμό, άγχος ή κατάθλιψη. Μερικές φορές ονομάζεται και συναισθηματική «Αστάθεια» (Volatility) ή αντιστρέφεται ως συναισθηματική σταθερότητα. Είναι αλληλένδετος με χαμηλή ανοχή στο στρες ή τα αποτρεπτικά ερεθίσματα, και αφορά ένα χαρακτηριστικό το οποίο έχει μελετηθεί στην έρευνα της ιδιοσυγκρασίας του ανθρώπου για δεκαετίες, πριν προσαρμοστεί στο μοντέλο OCEAN. Όσοι έχουν υψηλή βαθμολογία στον Νευρωτισμό είναι αντιδραστικά άτομα και ευάλωτα στο στρες, ενώ μικρές απογοητεύσεις μπορούν να εκληφθουν ως απελπιστικά δύσκολες. Παράλληλα, τείνουν να διατηρούν τις αρνητικές συναισθηματικές αντιδράσεις τους για ασυνήθιστα μεγάλα χρονικά διαστήματα, με αποτέλεσμα να είναι συχνά σε κακή διάθεση. Στο άλλο άκρο της κλίμακας, τα άτομα, τα άτομα με χαμηλές βαθμολογίες αναστατώνονται λιγότερο εύκολα και είναι λιγότερο συναισθηματικά αντιδραστικά. Τείνουν να είναι ήρεμοι και απαλλαγμένοι από επίμονα αρνητικά συναισθήματα. Όμως, η ελευθερία από τα αρνητικά συναισθήματα δε σημαίνει ότι τα άτομα αυτά βιώνουν πολλά θετικά συναισθήματα. Πιθανές προτάσεις οι οποίες χρησιμοποιούνται είναι: «Αγχώνομαι εύκολα», «Αναστατώνομαι εύκολα», «Έχω συχνές εναλλαγές διάθεσης». Αντίστροφα: «Τις περισσότερες φορές είμαι χαλαρός», «Σπάνια αγχώνομαι», «Σπάνια είμαι μελαγχολικός».

Μέσω συγκεκριμένων ερωτήσεων οι οποίες είναι δομημένες για να στοχεύουν κατάλληλα ένα χαρακτηριστικό όπως αυτές που δόθηκαν παραπάνω στις περιγραφές των χαρακτηριστικών, με χρήση τεστ καθίσταται δυνατή η παραγωγή αποτελεσμάτων ώστε να καταλήξουμε στη δημιουργία ενός προφίλ βασισμένο στα πέντε χαρακτηριστικά. Αν και πλέον το μοντέλο FFM (Five Factor Model) έχει αναλυθεί και δεν χρησιμοποιείται αυτούσιο για την παραγωγή αποτελεσμάτων διότι δεν βρίσκει μεγάλη συγκρισιμότητα στον πραγματικό κόσμο, νέα μοντέλα όπως το NEO PI-R (Revised NEO Personality Inventory) το χρησιμοποιούν ως βάση για την ανάπτυξη και την χρήση περισσότερων δευτερευόντων χαρακτηριστικών για τον ορισμό ενός χαρακτήρα, ενώ υπάρχουν και μοντέλα όπως το HEXACO που αναπτύχθηκαν με μερική συσχέτιση με το μοντέλο Big Five το οποίο χρησιμοποιεί τρία από τα πέντε χαρακτηριστικά (ECO) ενώ διαφοροποιεί τα υπόλοιπα. Το NEO PI-R χρησιμοποιεί μέχρι και 240 προτάσεις ενώ το NEO-FFI είναι ένα πιο σύντομο ερωτηματολόγιο με 60 προτάσεις, ενώ πέρα από τα πέντε χαρακτηριστικά, αναφέρουν 6 υποκατηγορίες για το κάθε χαρακτηριστικό. Επίσης, το 16PF(16 Personality Factor questionnaire) το οποίο αναπτύχθηκε από τον Cattell, με το σκεπτικό της συσχέτισης των χαρακτηριστικών μεταξύ τους για την παραγωγή αποτελεσμάτων του χαρακτήρα.



Εικόνα 3: Αποτελέσματα ενός τεστ OCEAN

1.5. Η πλατφόρμα ανάπτυξης Unity3D

Για την ανάπτυξη της διαδικαστικής δημιουργίας πόλης σε συνδυασμό με χρήση μοντέλων για την εμφάνιση των αποτελεσμάτων με εύκολο και κατανοητό τρόπο κρίνεται αναγκαία η χρήση μιας πλατφόρμας η οποία έχει τη δυνατότητα να κάνει συνδυασμό προγραμματισμού και ανάπτυξης εφαρμογών μαζί με την εύκολη εμφάνιση μοντέλων ως εικόνα και video, καθώς και συγχρονισμού με τις επιπλέον πληροφορίες που χρειάζεται ο χρήστης. Έτσι, υπάρχει δυνατή συσχέτιση της υλοποίησης με χρήση μηχανών που χρησιμοποιούνται για την ανάπτυξη βιντεοπαιχνιδιών. Τη στιγμή που γίνεται η συγγραφή αυτής της διπλωματικής εργασίας, πλατφόρμες που προσφέρουν πολλά εργαλεία και είναι δωρεάν είναι κάποιες από τις παρακάτω:

- **Unreal Engine (2D, 3D, high portability Game Engine)**
- **Unity (2D, 3D, cross-platform Game Engine)**
- **Construct Engine (HTML5 2D Game Engine)**
- **GoDot (2D, 3D, cross-platform, open-source Game Engine, MIT)**
- **OpenSAGE (3D SAGE open reimplementation for Strategy Games)**
- **CryEngine (Crytek proprietary Game Engine, 3D)**
- **Amazon Lumberyard (3D, based on CryEngine)**
- **GameMaker (2D Development Platform, cross-platform)**
- **Open3D Engine (3D, based on Amazon Lumberyard, open source)**

Από τις παραπάνω μηχανές, κατάλληλες για την διπλωματική εργασία είναι η Unreal Engine, η Unity, η GoDot και η Open3DE. Λόγω της εξοικείωσης του συγγραφέα με την πλατφόρμα Unity, επιλέχθηκε αυτή για την υλοποίηση του θέματος της διπλωματικής εργασίας.

Η πλατφόρμα ανάπτυξης παιχνιδιών Unity3D

Πρόκειται για μία μηχανή παιχνιδιών η οποία εμφανίστηκε πρώτη φορά το 2005 στο AWDC (Apple Worldwide Developers Conference). Έχει σχεδιαστεί ώστε να υποστηρίζει πολλές πλατφόρμες (Desktop, Mobile, Console & VR). Οι συνηθισμένες μηχανές παιχνιδιών είχαν την ανάγκη να καλύπτουν τις αλλαγές του υλικολογισμικού των πλατφορμών, με αποτέλεσμα την ανάγκη δημιουργίας κώδικα που καλύπτει διαφορετικές περιπτώσεις. Ως αποτέλεσμα, η Unity3D καλύπτει τις περιπτώσεις αυτές ενθυλακώνοντας τες στην ίδια τη μηχανή, ώστε ο προγραμματιστής να μην χρειάζεται να ασχοληθεί με επιπλέον κώδικα ο οποίος είναι χρονοβόρος στην δημιουργία, ή μη βέλτιστος, δίνοντας έτσι τη δυνατότητα στον προγραμματιστή να ασχοληθεί μόνο με την ανάπτυξη του παιχνιδιού. Η μηχανή υποστηρίζει δημιουργία τρισδιάστατων και δισδιάστατων παιχνιδιών, ενώ έχει χρησιμοποιηθεί και ως μηχανή για ανάπτυξη προσομοιώσεων για την παραγωγή πρακτόρων Τεχνητής Νοημοσύνης, συστημάτων αντίληψης, ενώ έχει χρησιμοποιηθεί και στην αυτοκινητοβιομηχανία για την παραγωγή εκπαίδευσης συστημάτων συντήρησης σε πραγματικό χρόνο καθώς και στην λειτουργία των εργοστασίων προσφέροντας Real-Time 3D για τη διαμόρφωση της κατασκευής. Η Unity χρησιμοποιεί τη γλώσσα προγραμματισμού C# για τη δημιουργία κώδικα από τον προγραμματιστή σε συνδυασμό με το Mono (.NET subsystem). Αυτή τη στιγμή κυκλοφορεί η έκδοση 2022.1.13, ενώ προσφέρεται με δωρεάν άδειας χρήσης για προγραμματιστές βάσει των εσόδων τους από τα παραγόμενα έργα τους.

1.5.1. Χρήση προσθέτων στην πλατφόρμα Unity3D

Ένα βασικό χαρακτηριστικό το οποίο έχει η πλατφόρμα Unity είναι η δυνατότητα εισαγωγής προσθέτων (plug-ins). Χρησιμοποιώντας το AssetStore ο χρήστης μπορεί να βρει λύσεις σε θέματα τα οποία θέλει να καλύψει, όπως την εύρεση μοντέλων (2D sprites, 3D Models) ή ακόμη και τη κάλυψη συγκεκριμένης λειτουργικότητας (πχ Σύστημα διαλόγου χαρακτήρων). Αυτό έχει ως αποτέλεσμα τη μείωση χρόνου στη συνολική υλοποίηση της ιδέας του χρήστη, καθώς και του κόστους παραγωγής. Παράλληλα, υπάρχει η δυνατότητα εισαγωγής μοντέλων από διαφορετικές πλατφόρμες, όπως το Mixamo (Adobe 3D character models & model animations). Φυσικά, η χρήση άλλων πλατφορμών για δημιουργία μοντέλων είναι εφικτή, δίνοντας έτσι στον χρήστη την δυνατότητα να χρησιμοποιήσει την εμπειρία του σε άλλες πλατφόρμες όπως το Blender (3D computer graphics software tool) για την παραγωγή μοντέλων.

2. ΥΛΟΠΟΙΗΣΗ

2.1. Επεξήγηση στόχου

Έχοντας πλέον κατανοήσει τις βασικές έννοιες που χρησιμοποιούνται στη διπλωματική εργασία, είναι εφικτή πλέον η επεξήγηση του θέματος καθώς και η ανάλυση των επιμέρους λειτουργιών που χρησιμοποιούνται για την υλοποίηση του. Ο στόχος της διπλωματικής ορίζεται στην διαδικαστική δημιουργία μιας περιοχής η οποία θα προσομοιώνει μία πόλη. Στη πόλη αυτή, θα δημιουργηθούν διαδικαστικά χαρακτήρες και πάνω σε αυτούς θα υλοποιηθεί το μοντέλο OCEAN με τα χαρακτηριστικά που αναλύθηκαν στο κεφάλαιο 1.4. .

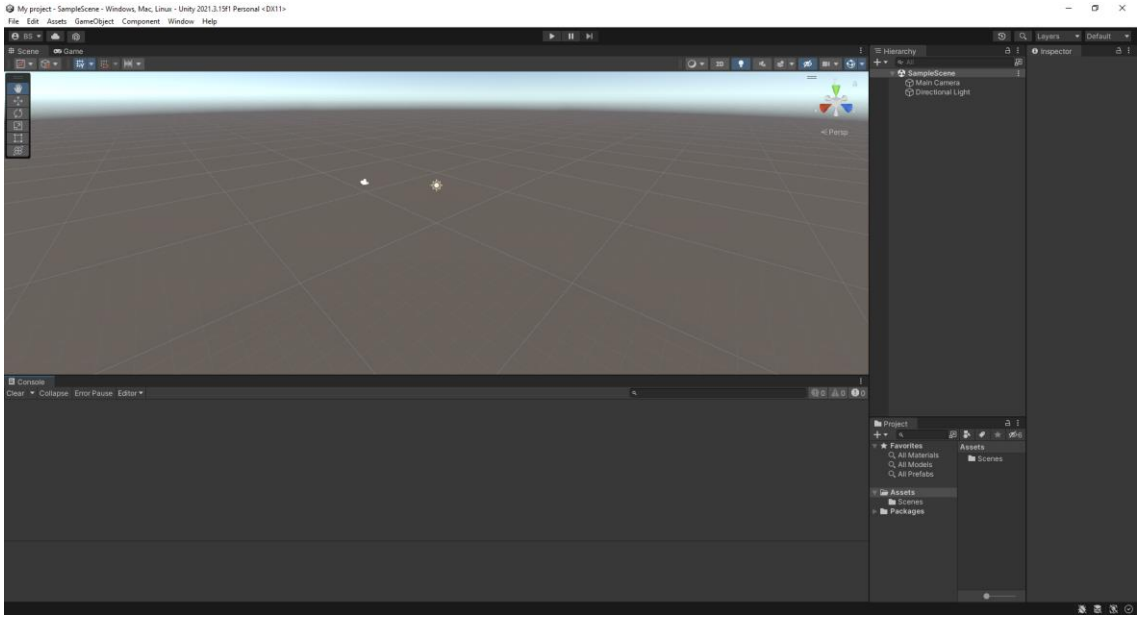
Μέσα σε αυτό το κεφάλαιο, θα γίνει ανάλυση των επιμέρους διαδικασιών, των αναγκών και των συστημάτων που χρησιμοποιήθηκαν ώστε να δημιουργηθεί μια πόλη με προγραμματισμό ο οποίος θα παράγει διαδικαστικά ένα αποτέλεσμα. Έχοντας υπόψιν τις λειτουργίες, θα γίνει πλήρης αναφορά στην κατασκευή του κόσμου από οικοδομικά τετράγωνα μέχρι την τοποθέτηση των κτηρίων στην κατάλληλη θέση, την δυναμική δημιουργία των ανθρωπόμορφων χαρακτήρων, την κίνηση και τα χαρακτηριστικά τους, καθώς και τις ανάγκες που θα προσομοιωθούν κατά την εκτέλεση της υλοποίησης.

Στις επόμενες ενότητες του κεφαλαίου, θα γίνει η επεξήγηση των επιμέρους διαδικασιών, των εξωτερικών πακέτων που χρησιμοποιήθηκαν, την διασύνδεση των επιμέρους scripts για μεταφορά πληροφοριών και δεδομένων, καθώς και την αλληλοεπίδραση μεταξύ τους για την παραγωγή του ανάλογου αποτελέσματος.

Επισημαίνεται ότι η συγκεκριμένη διπλωματική δεν αποτελεί μέρος της επεξήγησης της λειτουργίας της πλατφόρμας ανάπτυξης Unity3D. Ως αποτέλεσμα, για λόγους περιεκτικότητας θα γίνει προσπάθεια η υλοποίηση του θέματος να παραμείνει ανεξάρτητη της πλατφόρμας, κυρίως σε θέματα που δεν αφορούν την βασική λειτουργία της υλοποίησης.

2.2. Υλοποίηση επιμέρους διαδικασιών

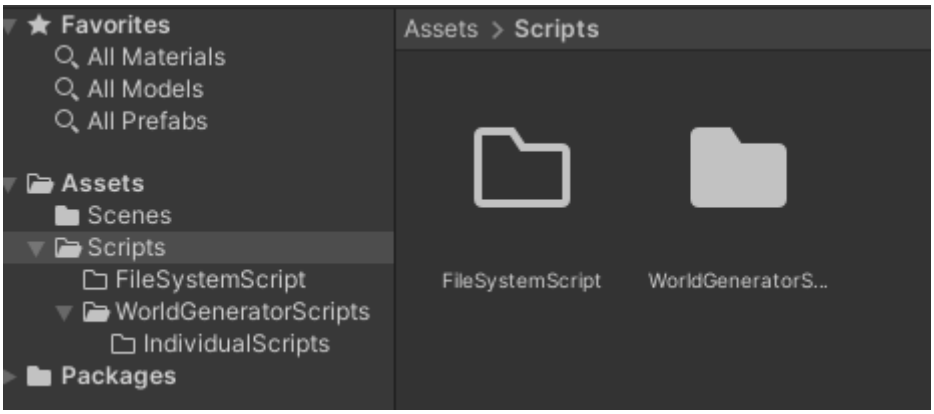
Κατά την έναρξη της υλοποίησης, η δημιουργία ενός project και η εκκίνηση του καταλήγει στην παρακάτω αρχική οθόνη της Unity3D.



Εικόνα 4: Κενός editor σε νέο project, Unity3D.

Για την καλύτερη τακτοποίηση της υλοποίησης, η δημιουργία φακέλων είναι ένα σημαντικό βήμα για την τακτοποίηση του project. Στο κεφάλαιο 2.2. θα γίνει η δημιουργία κώδικα χωρίς την χρήση

μοντέλων ή υλικών. Για τον λόγο αυτό, θα χρειαστεί ο φάκελος “Scripts” ο οποίος θα εμπλουτιστεί σταδιακά με κώδικα. Εκεί, θα γίνει και ο διαχωρισμός των scripts σε διάφορες κατηγορίες, ανάλογα την λειτουργικότητα που προσφέρουν.

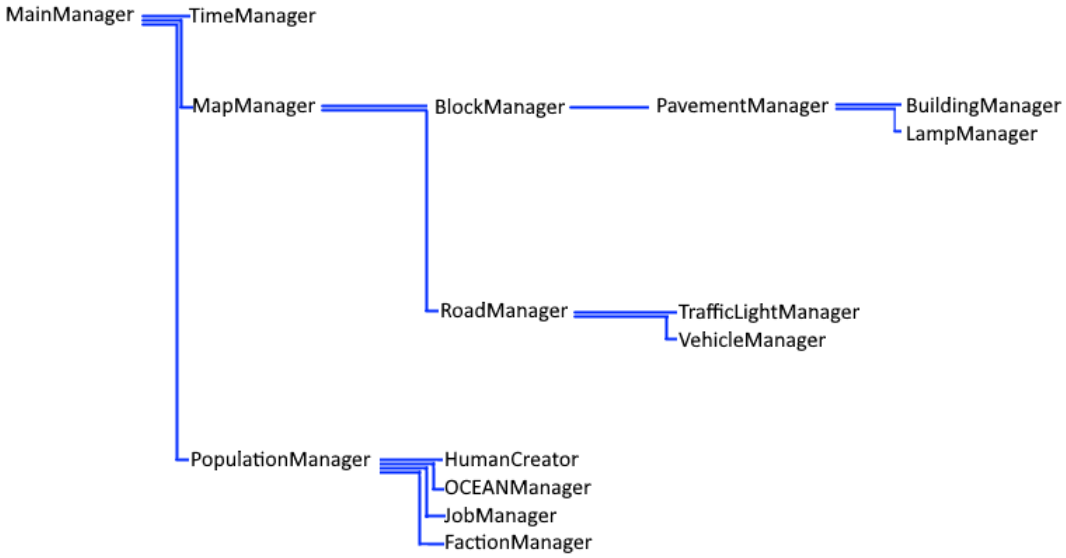


Εικόνα 5: Ο αρχικός φάκελος Scripts

Στις επόμενες υποενότητες του κεφαλαίου 2.2 θα γίνει ανάλυση των βασικών *scripts* που θα αποτελέσουν το σημείο έναρξης της υλοποίησης.

2.2.1. Υλοποίηση βασικών Διαχειριστών

Για να υλοποιηθεί σε βέλτιστο επίπεδο η υλοποίηση, απαιτείται η διακριτή κατακερμάτιση του κώδικα σε πολλά επίπεδα, έτσι ώστε να αποσαφηνιστεί η λειτουργία του κάθε «Διαχειριστή» (Manager). Ο κάθε διαχειριστής θα αναλάβει από μία βασική λειτουργία, ενώ διαχειριστές που χρησιμοποιούν άλλους διαχειριστές και μεταφέρουν δεδομένα θα βρίσκονται σε υψηλότερο επίπεδο από τους προηγούμενους. Τέλος, η ύπαρξη ενός «Υπέρ-Διαχειριστή» είναι επίσης απαραίτητη, μιας και θα αναλάβει την διακίνηση πληροφορίας προς όλα τα επίπεδα, ενώ ταυτόχρονα θα δέχεται και τις ανάλογες εισόδους από τον χρήστη για την κατασκευή της πόλης και τον έλεγχο της προσομοίωσης. Παρακάτω, εμφανίζεται το διάγραμμα με όλους τους διαχειριστές καθώς και του επιπέδου που βρίσκονται.



Εικόνα 6: Οι βασικοί διαχειριστές και τα επίπεδα που βρίσκονται

Αναφορικά, σύμφωνα με το παραπάνω διάγραμμα, οι διαχειριστές είναι οι εξής:

- MainManager

- TimeManager
- MapManager
- PopulationManager
- BlockManager
- RoadManager
- HumanCreator
- OCEANManager
- JobManager
- PavementManager
- TrafficLightManager
- VehicleManager
- BuildingManager
- LampManager

Όπως αναφέρθηκε ήδη, κάθε διαχειριστής έχει μία διακριτή λειτουργία. Ασχολείται με αυτή καθ΄ όλη τη διάρκεια της υλοποίησης, ενώ επικοινωνεί με τους υπόλοιπους διαχειριστές και μεταφέρει πληροφορία στα ανάλογα επίπεδα. Με τη σειρά θα γίνει η ανάλυση του καθενός ξεχωριστά παρακάτω.

MainManager

Ο κεντρικός Υπέρ-Διαχειριστής της υλοποίησης. Επικοινωνεί και δίνει εντολές σε όλα τα επόμενα επίπεδα διαχειριστών. Παράλληλα, μεταφέρει δεδομένα από το UI και χρησιμοποιεί τις παραμέτρους για την παραγωγή της πόλης. Κατά την εκτέλεση της υλοποίησης, ο MainManager αναμένει τις οριστικοποιημένες παραμέτρους από τον χρήστη. Μόλις αυτό γίνει, ενεργοποιείται και εκτελείται ο παρακάτω κώδικας:

```
1.    void Awake()
2.    {
3.        CallToStaticMainManager = this;
4.        mapManager.Initialize(range);
5.        firstTime = true;
6.        activate = true;
7.        BuildTown();
8.        SpawnCars();
9.        phaseTwo = true;
10.       phaseThree = false;
11.   }
```

Σε αυτό το κομμάτι κώδικα, γίνεται η εκτέλεση των μεθόδων *mapManager.Initialize, BuildTown, SpawnCars,* καθώς και ορίζονται κάποιες μεταβλητές. Η πρώτη μέθοδος ορίζει μία μεταβλητή για το μέγεθος της πόλης, η δεύτερη μέθοδος «χτίζει» την πόλη ενώ η τρίτη εμφανίζει οχήματα στη πόλη. Δίνοντας βάρος στις μεθόδους που βρίσκονται στον MainManager, η δεύτερη μέθοδος δημιουργεί ένα εικονικό «οικοδομικό τετράγωνο» στο σημείο 0,0 με μέγεθος 3x3, ενώ στη πορεία αποφασίζει το σημείο καθώς και το μέγεθος του τετραγώνου τυχαία, και το παραδίδει στον MapManager για τη δημιουργία του. Η μέθοδος συνεχίζει να εκτελείται μέχρι να μη μπορεί να γίνει η εισαγωγή νέων τετραγώνων στο ορισμένο μέγεθος της πόλης.

```
1.    void BuildTown()
2.    {
3.        do
4.        {
5.            if (firstTime)
6.            {
7.                pos.x = 0;
8.                pos.y = 0;
9.                size.x = 3;
10.               size.y = 3;
11.               addedBlockLocations = new List<Vector2Int>();
12.               firstTime = false;
13.           }
14.           else
15.           {
16.               if (posLocDictionary.Keys.Count == 0)
17.                   return;
18.               List<Vector2Int> availKeys = new List<Vector2Int>();
19.               foreach (Vector2Int key in posLocDictionary.Keys)
20.                   availKeys.Add(key);
21.               Vector2Int localKey = availKeys[UnityEngine.Random.Range(0, availK
eys.Count)];
22.               size = localKey;
23.               pos = posLocDictionary[size][UnityEngine.Random.Range(0, posLocDic
tionary[size].Count)];
24.           }
25.
26.           mapManager.CreateNewBlock(pos, size);
27.           addedBlockLocations.Add(new Vector2Int(pos.x, pos.y));
28.           CalculateNewDictionaryOfPossibleLocations();
29.       } while (posLocDictionary.Keys.Count > 0);
30.       CreateBuildings();
31.   }
```

Στη συνέχεια, όπως φαίνεται στο τέλος της μεθόδου, γίνεται η δημιουργία των κτηρίων, σύμφωνα με τις παραμέτρους, καθώς και η τρίτη μέθοδος που αναφέρθηκε παραπάνω για την εμφάνιση οχημάτων.

```
1.    void CreateBuildings()
2.    {
3.        mapManager.CreateBuildings(totalApartmentBuildings, totalWorkplaceBuilding
s, totalSuperMarketBuildings, totalEntertainmentBuildings, totalSchoolBuildings, tot
alLibraryBuildings);
4.    }
5.    void SpawnCars()
6.    {
7.        mapManager.SpawnCars(carCount);
8.    }
```

Μέσω των αρχικών μεταβλητών, ενεργοποιείται ένας αλυσιδωτός μηχανισμός εκκίνησης. Μετά τη δημιουργία του χάρτη, στο δεύτερο Frame εκτέλεσης δημιουργείται ο χάρτης πλοήγησης που βασίζεται στη δημιουργημένη πόλη, ενώ στο τρίτο Frame (μέσω χρήσης της μεθόδου Update) δημιουργείται ο πληθυσμός.

```
1.    private void Update()
2.    {
3.        if(phaseTwo)
4.        {
5.            phaseTwo = false;
6.            BuildNavmesh();
7.            phaseThree = true;
8.        }
9.        else if(phaseThree)
10.       {
11.           phaseThree = false;
12.           CreateFamilies();
13.           this.enabled = false;
14.       }
15.   }
```

Άλλες βασικές λειτουργίες του MainManager είναι η επικοινωνία του TimeManager με τον πληθυσμό για την ενημέρωση τους για την τρέχουσα ώρα, καθώς και την αλλαγή ηλικίας, καθώς και την ενημέρωση του TimeManager για τις λειτουργίες του πληθυσμού, ώστε να γίνει η προσομοίωση της ενασχόλησης τους μέσα στην ημέρα.

TimeManager

Διαχειριστής χρόνου. Ασχολείται με τον υπολογισμό θέσης του ήλιου και του φεγγαριού για την δημιουργία ρεαλιστικών σκιών, καθώς και την ενημέρωση του πληθυσμού για τη χρονιά, την μέρα και την ώρα, ενώ δίνει την δυνατότητα στον χρήστη να αυξήσει χειροκίνητα ώρα και χρονιά, καθώς και την ταχύτητα εκτέλεσης της προσομοίωσης. Ταυτόχρονα, ελέγχει τους χρόνους ενασχόλησης του πληθυσμού σε επίπεδο ατόμου και τους ενημερώνει ώστε να συνεχίσουν στην επόμενη εργασία ή ενασχόληση.

Η βασική λειτουργία του ενθυλακώνεται σε μία μέθοδο *Update,* η οποία εκτελείται κάθε frame. Συνήθως, αυτό ορίζεται από την ταχύτητα εμφάνισης καρέ (frame), που από προεπιλογή ορίζεται σε 60frames per second.

```
1.    private void Update()
2.    {
3.        timer += Time.deltaTime;
4.        if (timer > interval)
5.        {
6.            RefreshTime();
7.            RefreshText();
8.            timer = 0f;
9.        }
10.       else
11.       {
12.           float seconds = (int)(timer * 60);
13.           eulerVector.x = (60 * currTime.x + currTime.y+(seconds/60f))*0.25f;
14.           eulerVector.x -= 100;
15.           celestialTransform.SetLocalPositionAndRotation(celestialTransform.posi
tion, Quaternion.Euler(eulerVector));
16.       }
17.   }
```

Σε περίπτωση που υπάρχει αλλαγή ώρας, εκτελείται ο ανάλογος κώδικας που μειώνει τις εναπομείναντες ώρες του κάθε ατόμου που βρίσκεται σε κατάσταση ενασχόλησης.

```
1.        public void IncrementHour()
2.        {
3.            List<Person> idlersToActivate;
4.
5.            if (idlers.ContainsKey(1))
6.            {
7.                idlersToActivate = idlers[1];
8.
9.
10.               foreach (Person person in idlersToActivate)
11.               {
12.                   person.NotifyToContinue();
13.               }
14.           }
15.           Dictionary<int,   List<Person>>   newIdlers   =   new   Dictionary<int,
List<Person>>();
16.           foreach (int key in idlers.Keys)
17.           {
18.               if (key == 1)
19.                   continue;
20.               newIdlers.Add(key - 1, idlers[key]);
21.           }
22.           idlers = newIdlers;
23.       }
```

Σημειώνεται ότι η ονομασία «*idlers*» προέρχεται από την κατάσταση «αναμονής» που μελλοντικά θα βρίσκεται το μοντέλο του χαρακτήρα στη πόλη κατά τη διαδικασία στην οποία ασχολείται με κάτι και φαίνεται ως ανενεργός, δηλαδή «idle».

MapManager

Διαχειριστής χάρτη. Ασχολείται με την δημιουργία του εικονικού χάρτη (Dictionary περιοχής) και συγκρατεί τα σημεία που βρίσκονται τα οικοδομικά τετράγωνα για εύκολη και άμεση λειτουργία. Επικοινωνεί με τον BlockManager, τον RoadManager και τον BuildingManager, λαμβάνει οδηγίες από τον MainManager και παρέχει πληροφορίες για τη τοποθεσία των κτηρίων στον πληθυσμό. Περιέχει τις βασικές λειτουργίες ελέγχου της τοποθέτησης αντικειμένων στη πόλη και κατασκευάζει τις πιθανές νέες τοποθεσίες των επόμενων τετραγώνων.

```
1.        public Dictionary<Vector2Int, List<Vector2Int>>
CalculateNewDictionaryOfPossibleLocations(List<Vector2Int> addedBlockLocations,
Vector2Int range)
2.        {
3.            Vector2Int tempSize;
4.            Vector2Int tempCalc;
5.            Block tempBlock;
6.            Dictionary<Vector2Int, List<Vector2Int>> tempDict;
7.            Dictionary<Vector2Int, List<Vector2Int>> finalDict;
8.            //tempDict = new Dictionary<Vector2Int, List<Vector2Int>>();
9.            finalDict = new Dictionary<Vector2Int, List<Vector2Int>>();
10.           foreach (Vector2Int addedBlockLocation in addedBlockLocations)
11.           {
12.               if (addedBlockLocation.x < range.x || addedBlockLocation.x >
range.y || addedBlockLocation.y < range.x || addedBlockLocation.y > range.y)
13.                   continue;
14.               tempBlock = mapReferences[addedBlockLocation];
15.               tempSize = tempBlock.GetSize();
16.               List<Vector2Int> posToCheck = new List<Vector2Int>();
```

```
17.
18.                 for (int i = -1; i < tempSize.x + 1; i++)
19.                 {
20.                     for (int j = -1; j < tempSize.y + 1; j++)
21.                     {
22.                         if (i >= 0 && i < tempSize.x && j >= 0 && j < tempSize.y)
23.                             continue;
24.                         posToCheck.Add(new Vector2Int(i, j));
25.
26.                     }
27.                 }
28.                 foreach (Vector2Int position in posToCheck)
29.                 {
30.                     for (int i = 1; i < 4; i++)
31.                     {
32.                         for (int j = 1; j < 4; j++)
33.                         {
34.                             Vector2Int newPos = new Vector2Int((position.x < 0) ?
position.x * i : position.x, (position.y < 0) ? position.y * j : position.y);
35.                             newPos += addedBlockLocation;
36.                             Vector2Int localSize = new Vector2Int(i, j);
37.                             if (CheckPossibleLocationForValidPosition(newPos,
localSize))
38.                             {
39.                                 if (!finalDict.ContainsKey(localSize))
40.                                     finalDict.Add(localSize, new
List<Vector2Int>());
41.                                 finalDict[localSize].Add(newPos);
42.                             }
43.                         }
44.                     }
45.                 }
46.
47.             }
48.             return finalDict;
49.         }
```

Βασική επίσης λειτουργία του είναι η διαδικαστική επιλογή των κτηρίων που θα χρησιμοποιηθούν κατά την εκτέλεση, σύμφωνα με τις επιλογές του χρήστη.

```
1.     public void CreateBuildings(int totalApartmentBuildings, int
totalWorkplaceBuildings, int totalSuperMarketBuildings, int
totalEntertainmentBuildings, int totalSchoolBuildings, int totalLibraryBuildings)
2.         {
3.             Vector2Int key;
4.             int failTries = 0;
5.             for (int i = 0; i < totalApartmentBuildings; i++)
6.             {
7.                 key =
mapReferences.ElementAt(Random.Range(0,mapReferences.Count)).Key;
8.                 if (!blockManager.CreateBuilding("ApartmentBuilding",
mapReferences[key]) && failTries < 10)
9.                 {
10.                     failTries++;
11.                     i--;
12.                 }
13.             }
14.             failTries = 0;
15.             for (int i = 0; i < totalWorkplaceBuildings; i++)
16.             {
```

```
17.                 key = mapReferences.ElementAt(Random.Range(0,
mapReferences.Count)).Key;
18.                 if(!blockManager.CreateBuilding("WorkplaceBuilding",
mapReferences[key]) && failTries<10)
19.                 {
20.                     failTries++;
21.                     i--;
22.                 }
23.             }
24.             failTries = 0;
25.             for (int i = 0; i < totalSuperMarketBuildings; i++)
26.             {
27.                 key = mapReferences.ElementAt(Random.Range(0,
mapReferences.Count)).Key;
28.                 if(!blockManager.CreateBuilding("SuperMarket",
mapReferences[key])&&failTries<10)
29.                 {
30.                     failTries++;
31.                     i--;
32.                 }
33.             }
34.             failTries = 0;
35.             for (int i = 0; i < totalEntertainmentBuildings; i++)
36.             {
37.                 key = mapReferences.ElementAt(Random.Range(0,
mapReferences.Count)).Key;
38.                 if(!blockManager.CreateBuilding("Entertainment",
mapReferences[key])&&failTries<10)
39.                 {
40.                     failTries++;
41.                     i--;
42.                 }
43.             }
44.             failTries = 0;
45.             for (int i = 0; i < totalSchoolBuildings; i++)
46.             {
47.                 key = mapReferences.ElementAt(Random.Range(0,
mapReferences.Count)).Key;
48.                 if(!blockManager.CreateBuilding("School",
mapReferences[key])&&failTries<10)
49.                 {
50.                     failTries++;
51.                     i--;
52.                 }
53.
54.             }
55.             failTries = 0;
56.             for (int i = 0; i < totalLibraryBuildings; i++)
57.             {
58.                 key = mapReferences.ElementAt(Random.Range(0,
mapReferences.Count)).Key;
59.                 if(!blockManager.CreateBuilding("Library",
mapReferences[key])&&failTries<10)
60.                 {
61.                     failTries++;
62.                     i--;
63.                 }
64.             }
65.         }
```

PopulationManager

Διαχειριστής πληθυσμού. Λαμβάνει εντολή και παραμέτρους μέσω του MainManager και δημιουργεί τον ανάλογο πληθυσμό που ζήτησε ο χρήστης κατασκευάζοντας αρχικές οικογένειες για την παραγωγή ανθρώπων με τυχαία χαρακτηριστικά και μορφή. Παράγει τις ενδιάμεσες γενιές και στη συνέχεια οριστικοποιεί την τελευταία γενιά ανθρώπων και την ορίζει ως τον ενεργό πληθυσμό της προσομοίωσης. Οι λειτουργίες τους είναι ορισμένες στενά με τους υποδιαχειριστές *HumanCreator, OCEANManager και JobManager*.

```
1.    //δημιουργία αρχικών ανθρώπων και οικογενειών
2.                _PIDCOUNTER = _FIDCOUNTER = 0;
3.                Dictionary<long, Family> firstGenFamilies = new Dictionary<long,
Family>();
4.                //create _InitFamilyCountSingleFamilies
5.                for (int i = 0; i < _InitFamilyCount; i++)
6.                {
7.                    //Person person = new Person();
8.                    GameObject humanInstance = Instantiate(femalePrefab, GetPos(),
Quaternion.identity);
9.                    gos.Add(humanInstance);
10.                   //Person person = new Person(_PIDCOUNTER, true);
11.                   Person person = humanInstance.GetComponent<Person>();
12.                   person.InitPerson(_PIDCOUNTER, true, true, mapManager,this);
13.                   _PIDCOUNTER++;
14.                   humanCreator.SetRandomValues(ref person, Gender.Female);
15.                   oceanManager.SetRandomValues(ref person);
16.                   humanCreator.SetEagerAndLearnValues(ref person);
17.                   humanInstance.name = person.GetFullName();
18.                   jobManager.SetRandomJobWithOCEAN(person);
19.                   Family family = Instantiate(familyPrefab,
familiesHolder.transform).GetComponent<Family>();
20.                   family.InitFamily(_FIDCOUNTER, -1, person);
21.                   person.transform.SetParent(family.transform);
22.                   firstGenFamilies.Add(_FIDCOUNTER, family);
23.                   family.gameObject.name = "family_" + _FIDCOUNTER;
24.                   person.SetFamilies(family, null);
25.                   _FIDCOUNTER++;
26.
27.               }
```

Και η οριστικοποίηση των τελικών οικογενειών:

```
1.    foreach (long key in proceduralFamilies.Keys)
2.                {
3.                    GameObject humanInstance =
Instantiate(proceduralFamilies[key].GetParent(0).GetGender().Equals(Gender.Male) ?
femalePrefab : malePrefab, GetPos(), Quaternion.identity);
4.                    gos.Add(humanInstance);
5.                    //Person person = new Person(_PIDCOUNTER, true);
6.                    Person person = humanInstance.GetComponent<Person>();
7.                    person.InitPerson(_PIDCOUNTER, false, true, mapManager, this);
8.                    person.SetFamilies(proceduralFamilies[key], null);
9.                    _PIDCOUNTER++;
10.                   proceduralFamilies[key].GetParent(0).SetInitHuman(false);
11.                   humanCreator.SetRandomValues(ref person,
proceduralFamilies[key].GetParent(0).GetGender().Equals(Gender.Male) ? Gender.Female
: Gender.Male);
12.                   oceanManager.SetRandomValues(ref person);
13.                   humanCreator.SetEagerAndLearnValues(ref person);
14.                   humanInstance.name = person.GetFullName();
15.                   jobManager.SetRandomJobWithOCEAN(person);
```

```
16.                     proceduralFamilies[key].AddParent(person, -1);
17.                     humanInstance.transform.SetParent(proceduralFamilies[key].game
Object.transform);
18.                     mapManager.CreateRandomApartment(proceduralFamilies[key]);
19.                     int childCount = Random.Range(_MinMaxChildCount.x,
_MinMaxChildCount.y + 1);
20.                     Dictionary<int, Family> children = new Dictionary<int,
Family>();
21.                     for (int i = 0; i < childCount; i++)
22.                     {
23.                         Gender gender = Random.Range(0, 2) == 0 ? Gender.Male :
Gender.Female;
24.                         humanInstance = Instantiate(gender.Equals(Gender.Male) ?
malePrefab : femalePrefab, GetPos(), Quaternion.identity);
25.                         gos.Add(humanInstance);
26.                         //Person person = new Person(_PIDCOUNTER, true);
27.                         person = humanInstance.GetComponent<Person>();
28.                         person.InitPerson(_PIDCOUNTER, false, false,
mapManager,this);
29.                         _PIDCOUNTER++;
30.                         Family childFamily =
Instantiate(familyPrefab).GetComponent<Family>();
31.                         childFamily.InitFamily(_FIDCOUNTER, key, person);
32.                         childFamily.transform.SetParent(proceduralFamilies[key].tr
ansform);
33.                         humanInstance.transform.SetParent(childFamily.gameObject.t
ransform);
34.                         //Family childFamily = new Family(_FIDCOUNTER, key,
person);
35.                         childFamily.gameObject.name = "family_" + _FIDCOUNTER;
36.                         _FIDCOUNTER++;
37.                         humanCreator.SetValues(ref person,
proceduralFamilies[key], gender);
38.                         oceanManager.SetValues(ref person,
proceduralFamilies[key], true);
39.                         humanCreator.SetEagerAndLearnValues(ref person);
40.                         humanInstance.name = person.GetFullName();
41.                         person.SetFamilies(childFamily, proceduralFamilies[key]);
42.                         jobManager.SetRandomJobWithOCEAN(person, "Student");
43.                         person.SetAsChild();
44.                         children.Add(i, childFamily);
45.                     }
46.                     proceduralFamilies[key].AddChildren(children);
47.                 }
```

Κατά τη δημιουργία, ο *PopulationManager* αποφασίζει το σημείο στο οποίο θα εμφανιστεί ο κάθε άνθρωπος στο χάρτη, καθώς και τη δημιουργία νέων διαμερισμάτων σε συνεργασία με τον *MapManager*.

BlockManager

Διαχειριστής Τετραγώνων. Είναι η διασύνδεση της δημιουργίας οικοδομικών τετραγώνων ως δεδομένα και την υλοποίηση τους στην εκτέλεση της προσομοίωσης. Περιέχει βασικές λειτουργίες κατασκευής *GameObject* και τοποθέτησης τους σε σωστό *Position.*

```
1.        public Block CreateBlock(Vector2Int position, Vector2Int size)
2.        {
3.            GameObject go = SelectPrefab(size);
4.            Block newBlock = go.GetComponent<Block>();
5.            newBlock.SetInitialData(position, size);
6.            return newBlock;
7.        }
```

```
1.        public void InitializeBlock(Block newBlock)
2.        {
3.            if (blockList == null)
4.                blockList = new List<Block>();
5.            blockList.Add(newBlock);
6.            Vector2Int position = newBlock.GetPosition();
7.            Vector2Int size = newBlock.GetSize();
8.
9.            newBlock.transform.SetParent(this.transform);
10.           newBlock.transform.localPosition = new Vector3(position.x *
sizeMultiplier + newBlock.transform.localScale.x * 0.5f, 0, position.y *
sizeMultiplier + newBlock.transform.localScale.z * 0.5f);
11.       }
```

Επίσης, παρέχει πληροφορίες στους χαρακτήρες για την εύρεση κτηρίων που ίσως αναζητούν, όπως καταστήματα, μέσω του *BuildingManager*.

```
1.        public Library GetClosestLibrary(Person person)
2.        {
3.            return buildingManager.SearchClosestLibrary(person);
4.        }
5.
6.        public Store GetClosestStore(Person person)
7.        {
8.            return buildingManager.SearchClosestStore(person);
9.        }
10.
11.       public Store GetCheapestStore(Person person)
12.       {
13.           return buildingManager.SearchCheapestStore(person);
14.       }
15.
16.       public ActivitiesBuilding GetClosestEntertainment(Person person)
17.       {
18.           return buildingManager.SearchClosestEntertainment(person);
19.       }
```

RoadManager

Διαχειριστής οδών. Παρόμοια με τον *BlockManager*, πρόκειται για έναν διαχειριστή διασύνδεσης με την πλατφόρμα Unity3D για την κατασκευή και τοποθέτηση *GameObjects*. Χρησιμοποιώντας ως μεταβλητές το νέο οικοδομικό τετράγωνο καθώς και έναν πίνακα των γύρω ήδη υπάρχοντων οικοδομικών τετραγώνων, αποφασίζει και δημιουργεί τα κατάλληλα *GameObjects* οδών για την χρήση τους από τους ανθρώπους και τα οχήματα.

HumanCreator

Κατασκευαστής ανθρώπων. Πρόκειται για έναν διαχειριστή οριστικοποίησης ανθρώπινων μοντέλων με διαδικαστικές μεθόδους. Οριστικοποιεί τα ανθρώπινα χαρακτηριστικά καθώς και τον ρουχισμό των μοντέλων σύμφωνα με το εξωτερικό πρόσθετο πακέτο UMA που θα αναλυθεί στο επόμενο κεφάλαιο, καθώς και αρχικοποιεί τα Attributes των χαρακτήρων που θα προκύψουν από τους *OCEANManager και JobManager*. Επίσης, στην περίπτωση των αρχικών γενεών που δεν έχουν μοντέλο, ορίζει τυχαία όνομα ανάλογα το φύλο. Σε περίπτωση κατασκευής ανθρώπινου μοντέλου σύμφωνα με την οικογένεια, ορίζει τα χαρακτηριστικά χρώματος μαλλιών και δέρματος ανάλογα τους γονείς. Διαφορετικά, επιλέγει διαδικαστικά και παραμετροποιεί το μοντέλο.

```
1.    //οριστικοποίηση τιμών και «ένδυση» ανθρώπινου μοντέλου.
2.    public void SetValues(ref Person person, Family family,Gender gender)
3.    {
4.        PersonAttributes pAttr = new PersonAttributes();
5.        pAttr.isAlive = true;
6.        pAttr.gender = gender;
7.        pAttr.age = Random.Range(3, 18);
8.        pAttr.name = GetRandomName(pAttr.gender);
9.        pAttr.surname = family.GetParent(Random.Range(0,2)).GetSurname();
10.       person.SetInitAttributes(pAttr);
11.       DressHuman(person,gender,family);
12.   }
```

```
1.        void DressHuman(Person person,Gender gender,Family family)
2.        {
3.            DynamicCharacterAvatar                        dca                        =
person.gameObject.GetComponent<DynamicCharacterAvatar>();
4.            if(supplSkinColor.r ==0)
5.            {
6.                supplSkinColor = new Color(1 - skinColor.r, 1 - skinColor.g, 1 -
skinColor.b, 1);
7.            }
8.            if (gender.Equals(Gender.Male))
9.            {
10.               dca.SetSlot(maleShirts[Random.Range(0, maleShirts.Count)]);
11.               dca.SetSlot(malePants[Random.Range(0, malePants.Count)]);
12.               dca.SetSlot(maleHairs[Random.Range(0, maleHairs.Count)]);
13.               dca.SetSlot(maleShoes[Random.Range(0, maleShoes.Count)]);
14.               dca.SetSlot(socks[Random.Range(0, socks.Count)]);
15.
16.               if (Random.Range(0,2)==1&&family==null)
17.                   dca.SetSlot(maleFacialHairs[Random.Range(0,
maleFacialHairs.Count)]);
18.
19.               dca.SetColor(clothingTopName,   new   Color(Random.Range(0f,   1f),
Random.Range(0f, 1f), Random.Range(0f, 1f), 1f));
20.               dca.SetColor(clothingBottomName,  new  Color(Random.Range(0f,  1f),
Random.Range(0f, 1f), Random.Range(0f, 1f), 1f));
21.               dca.SetColor(clothingSocksName,  new  Color(Random.Range(0f,  1f),
Random.Range(0f, 1f), Random.Range(0f, 1f), 1f));
22.               dca.SetColor(clothingShoesINITName+"01", new Color(Random.Range(0f,
1f), Random.Range(0f, 1f), Random.Range(0f, 1f), 1f));
23.
24.
25.               dca.ReapplyWardrobeCollections();
26.           }
27.           else
28.           {
29.               if (Random.Range(0, 2) == 1)
```

```
30.                 {
31.                     dca.SetSlot(femaleDresses[Random.Range(0,
femaleDresses.Count)]);
32.                     dca.SetColor(clothingTopName, new Color(Random.Range(0f, 1f),
Random.Range(0f, 1f), Random.Range(0f, 1f), 1f));
33.                 }
34.                 else
35.                 {
36.                     dca.SetSlot(femaleShirts[Random.Range(0,
femaleShirts.Count)]);
37.                     dca.SetSlot(femalePants[Random.Range(0, femalePants.Count)]);
38.                     dca.SetColor(clothingTopName, new Color(Random.Range(0f, 1f),
Random.Range(0f, 1f), Random.Range(0f, 1f), 1f));
39.                     dca.SetColor(clothingBottomName, new Color(Random.Range(0f,
1f), Random.Range(0f, 1f), Random.Range(0f, 1f), 1f));
40.                 }
41.                 //shoes
42.                 UMATextRecipe utrShoes = femaleShoes[Random.Range(0,
femaleShoes.Count)];
43.                 dca.SetSlot(utrShoes);
44.                 Color color = new Color(Random.Range(0f, 1f), Random.Range(0f, 1f),
Random.Range(0f, 1f), 1f);
45.                 if (utrShoes.DisplayValue.Equals("High Heels"))
46.                 {
47.                     dca.SetColor(clothingShoesINITName + "01", color);
48.                     dca.SetColor(clothingShoesINITName + "02", color);
49.                     dca.SetColor(clothingShoesINITName + "03", color);
50.                     dca.SetColor(clothingShoesINITName + "04", color);
51.                 }
52.                 else
53.                     dca.SetColor(clothingShoesINITName + "01", color);
54.
55.                 dca.SetSlot(femaleHairs[Random.Range(0, femaleHairs.Count)]);
56.                 dca.SetSlot(socks[Random.Range(0, socks.Count)]);
57.                 dca.SetColor(clothingSocksName, new Color(Random.Range(0f, 1f),
Random.Range(0f, 1f), Random.Range(0f, 1f), 1f));
58.
59.                 dca.ReapplyWardrobeCollections();
60.             }
61.             if (family != null)
62.             {
63.                 DynamicCharacterAvatar dcap1 =
family.GetParent(0).gameObject.GetComponent<DynamicCharacterAvatar>();
64.                 DynamicCharacterAvatar dcap2 =
family.GetParent(1).gameObject.GetComponent<DynamicCharacterAvatar>();
65.                 Color parent1SkinColor = dcap1.GetColor("Skin").color;
66.                 Color parent1HairColor = dcap1.GetColor("Hair").color;
67.                 Color parent2SkinColor = dcap2.GetColor("Skin").color;
68.                 Color parent2HairColor = dcap2.GetColor("Hair").color;
69.                 Color mixedHairColor = Random.Range(0, 2) == 1 ? parent1HairColor :
parent2HairColor;
70.                 Color mixedSkinColor = new Color((parent1SkinColor.r +
parent2SkinColor.r) * 0.5f, (parent1SkinColor.g + parent2SkinColor.g) * 0.5f,
(parent1SkinColor.b + parent2SkinColor.b) * 0.5f, 1);
71.                 dca.SetColor("Skin", mixedSkinColor);
72.                 dca.SetColor("Hair", mixedHairColor);
73.             }
74.             else
75.             {
76.                 float skinR = Random.Range(0f, 1f);
77.
```

```
78.                 Color color1 = new Color(skinColor.r + supplSkinColor.r * skinR,
skinColor.g + supplSkinColor.g * skinR, skinColor.b + supplSkinColor.b * skinR, 1);
79.                 Color       color2       =       naturalHairColor[Random.Range(0,
naturalHairColor.Count)];
80.                 dca.SetColor("Skin", color1);
81.                 dca.SetColor("Hair", color2);
82.             }
83.             dca.UpdateColors();
84.         }
```

Για λόγους συνοπτικής υλοποίησης, η επιλογή των διαφοροποιήσεων μέσω των ανθρώπινων πέντε βασικών χαρακτηριστικών είναι τυπική, ώστε να γίνει η εμφάνιση των διαφορών διακριτή κατά την εκτέλεση. Έτσι, επιλέγονται διακριτά ποσοστά ανάλογα το FFM κάθε χαρακτήρα. Παραδείγματος χάριν, ένας χαρακτήρας με υψηλό **Extraversion** έχει την ανάγκη να διασκεδάζει πιο συχνά από έναν χαρακτήρα με χαμηλό βαθμό (**Introversion**).

```
1.          public void SetEagerAndLearnValues(ref Person person)
2.          {
3.              //openess/conscientiousness/extraversion/agreeableness/neuroticism
4.
5.              float[] traits = new float[2];
6.
7.              traits[0] = person.GetTrait(2);
8.              traits[1] = person.GetTrait(1);
9.
10.             float entertainPerc = 0;
11.             if (traits[0]<0)
12.             {
13.                 if (traits[0] < -70)
14.                     entertainPerc = 0.125f;
15.                 else if (traits[0] < -40)
16.                     entertainPerc = 0.167f;
17.                 else
18.                     entertainPerc = 0.2f;
19.             }
20.             else
21.             {
22.                 if (traits[0] > 70)
23.                     entertainPerc = 0.5f;
24.                 else if (traits[0] > 40)
25.                     entertainPerc = 0.33f;
26.                 else
27.                     entertainPerc = 0.25f;
28.             }
29.             person.SetWantsToBeEntertainedPerc(entertainPerc);
30.             float conscperc = 0;
31.             if (traits[1] < 0)
32.             {
33.                 conscperc = 0.1f;
34.             }
35.             else
36.             {
37.                 if (traits[1] > 70)
38.                     conscperc = 0.5f;
39.                 else if (traits[1] > 40)
40.                     conscperc = 0.35f;
41.                 else
42.                     conscperc = 0.2f;
43.             }
```

```
44.             person.SetConsciencePercentage(conscperc);
45.         }
```

OCEANManager

Διαχειριστής του μοντέλου χαρακτηριστικών OCEAN. Χρησιμοποιείται για την διαδικαστική επιλογή των πέντε βασικών χαρακτηριστικών κάθε δημιουργημένου χαρακτήρα, σε συνδυασμό με το εξωτερικό πακέτο Love/Hate το οποίο περιέχει υλοποίηση του OCEAN. Σε περίπτωση που ο χαρακτήρας είναι μέλος οικογένειας και δεν αποτελεί χαρακτήρα πρώτης γενιάς, επιλέγονται χαρακτηριστικά στο εύρος των γονέων, για διακριτή συμπεριφορά. Στην ενότητα «3.4. Πιθανές κατ' επέκταση χρήσεις» θα γίνει αναφορά στον τρόπο επέκτασης με δημιουργία «εμπειριών» ενός χαρακτήρα στην παιδική ηλικία και πόσο επηρεάζεται η επιλογή των χαρακτηριστικών.

```
1.          public void SetRandomValues(ref Person person)
2.          {
3.              Traits traits = person.gameObject.GetComponent<Traits>();
4.              //Debug.Log("traits: " + traits != null);
5.              traits.traits = new float[5];
6.              for (int i = 0; i < traits.traits.Length; i++)
7.              {
8.                  traits.traits[i] = (int)(Random.Range(-100, 101));
9.                  if (traits.traits[i] > 100)
10.                     traits.traits[i] = 100;
11.                 if (traits.traits[i] < -100)
12.                     traits.traits[i] = -100;
13.             }
14.         }
15.
16.         public void SetValues(ref Person person, Family family,bool isFutureStudent)
17.         {
18.             Traits traits = person.gameObject.GetComponent<Traits>();
19.             //Debug.Log("traits: " + traits != null);
20.             traits.traits = new float[5];
21.             for (int i = 0; i < traits.traits.Length; i++)
22.             {
23.                 int parentTrait1, parentTrait2;
24.                 parentTrait1 = (int)family.GetParent(0).GetTrait(i);
25.                 parentTrait2 = (int)family.GetParent(1).GetTrait(i);
26.                 if (isFutureStudent) //pseudo
27.                 {
28.                     if (parentTrait1 < parentTrait2)
29.                         traits.traits[i]     =     (int)(Random.Range(parentTrait1,
parentTrait2 + 1));
30.                     else
31.                         traits.traits[i]     =     (int)(Random.Range(parentTrait2,
parentTrait1 + 1));
32.                 }
33.                 else
34.                 {
35.                     if (parentTrait1 < parentTrait2)
36.                         traits.traits[i]     =     (int)(Random.Range(parentTrait1,
parentTrait2 + 1));
37.                     else
38.                         traits.traits[i]     =     (int)(Random.Range(parentTrait2,
parentTrait1 + 1));
39.                 }
40.
41.                 if (traits.traits[i] > 100)
42.                     traits.traits[i] = 100;
43.                 if (traits.traits[i] < -100)
```

```
                    traits.traits[i] = -100;
            }
        }
```

JobManager

Διαχειριστής Εργασίας. Χρησιμοποιείται από τον *PopulationManager* για τον ορισμό εργασίας ενός ατόμου. Σε περίπτωση στατικής εργασίας που δίνεται αναγκαστικά για κατασκευή ατόμων μικρής ηλικίας ορίζεται εκείνη δίχως να επιλεχθεί τυχαία. Στη συνέχεια, σε συνεργασία με τον *BuildingManager* γίνεται εύρεση του χώρου εργασίας στη πόλη. Η διαδικαστική επιλογή εργασίας γίνεται τυχαία από ένα εύρος ορισμένων εργασιών και στην ενότητα «3.4. Πιθανές κατ' επέκταση χρήσεις» θα αναφερθούν τρόποι επέκτασης.

```
        public void SetRandomJobWithOCEAN(Person person)
        {
            person.SetJob(new Job(GetRandomJob()));
            buildingManager.SearchForWorkplace(person.GetJobName(), person);
        }

        public void SetRandomJobWithOCEAN(Person person, string staticJob)
        {
            person.SetJob(new Job(staticJob));
            if (staticJob.Equals("Student"))
                buildingManager.SearchForSchool(person);
            else
                buildingManager.SearchForWorkplace(person.GetJobName(), person);
        }
```

PavementManager

Διαχειριστής πεζοδρομίου. Ένας ακόμη διαχειριστής που ασχολείται με την οριστικοποίηση *GameObjects*. Δέχεται πληροφορίες από τον *BlockManager*, και ασχολείται με την εύρεση σημείου για την τοποθέτηση των κτηρίων στο πεζοδρόμιο καθώς και τον ορισμό ενός κτηρίου για χρήση διαμερισμάτων ή εργασιών. Παράλληλα, αποστέλλει εντολές προς τον *BuildingManager* για την κατασκευή των κτηρίων.

```
        public bool CreateBuilding(string type, Block block)
        {
            GameObject go = Instantiate(buildingManager.GetBuildingPrefab(type,out
bool valid));
            int tries = 0;
            do
            {
                bool spaceAvailable = block.CheckAvailableSlots(go);
                if (spaceAvailable)
                    break;
                tries++;
            } while (tries < 10);
            if (tries >= 10)
            {
                Destroy(go);
                return false;
            }
            block.AddToExistingObjects(go);
            go.transform.SetParent(block.buildingsParent.transform);
            go.transform.tag = "Building";
```

```
20.             go.transform.GetChild(0).tag = "Building";
21.             if (type.Equals("ApartmentBuilding")||!valid)
22.             {
23.                 type = "ApartmentBuilding";
24.                 go.AddComponent<ApartmentBuilding>();
25.             }
26.             else if(type.Equals("WorkplaceBuilding"))
27.             {
28.                 go.AddComponent<WorkplaceBuilding>();
29.             }
30.             buildingManager.AddCreatedBuilding(go, type, block);
31.             return true;
32.         }
```

TrafficLightManager

Διαχειριστής φωτεινών σηματοδοτών. Ελέγχει την συχνότητα εναλλαγής της φωτεινής σηματοδότησης, για την διαχείριση των οχημάτων και των πεζών. Δημιουργήθηκε για να προσθέσει έναν βαθμό ρεαλισμού στη προσομοίωση της πόλης κατά την εκτέλεση. Χρησιμοποιεί την μέθοδο *Update* για να αλλάζει τη σηματοδότηση ανά ένα ορισμένο χρονικό διάστημα.

```
1.          void Update()
2.          {
3.              timer += Time.deltaTime;
4.              if(timer>interval)
5.              {
6.                  timer = 0f;
7.                  if (trafficRoads == null)
8.                      return;
9.                  foreach(Vector2Int key in trafficRoads.Keys)
10.                 {
11.                     trafficRoads[key].RefreshTrafficLights();
12.                 }
13.             }
14.         }
```

VehicleManager

Διαχειριστής οχημάτων. Έχει δημιουργηθεί για την κατασκευή και καταστροφή *GameObjects* οχημάτων για την εύρυθμη λειτουργία κίνησης κατά την προσομοίωση. Κατά τη δημιουργία του οχήματος, ορίζει σημείο έναρξης και τερματικού σημείου του οχήματος και ενεργοποιεί το όχημα.

```
1.          public void SpawnCars(int count)
2.          {
3.              for(int i =0;i<count;i++)
4.              {
5.                  GameObject go=null;
6.                  try
7.                  {
8.                      roadBezierStart = roadManager.GetRandomRoad();
9.                      roadBezierEnd = roadManager.GetRandomRoad();
10.                     char eh;
11.                     Transform[]                    transforms                    =
roadManager.GetAllTransformDrivepoints(roadBezierStart, roadBezierEnd, 'f', out eh);
12.                     Vector4[]                     bezierPoints                     =
StaticBezierCurve.GetPoints(transforms, lr);
13.                     go = Instantiate(carPrefab,carParent.transform);
14.                     go.transform.position = (Vector3)bezierPoints[0];
```

```
15.                 go.GetComponent<CarTest>().Init(bezierPoints,     eh,     this,
roadBezierStart, roadBezierEnd);
16.             }
17.             catch { if(go!=null)Destroy(go); }
18.         }
19.     }
```

BuildingManager

Διαχειριστής κτηρίων. Έχει δημιουργηθεί για την επιλογή, την διαχείριση και την εύρεση κτηρίων ανάλογα την ανάγκη του πληθυσμού. Διατηρεί τα κτήρια σε λίστες και ανάλογα την αναζήτηση από τους ενεργούς χαρακτήρες επιστρέφει το κατάλληλο αποτέλεσμα. Χρησιμοποιείται για την αρχικοποίηση των διαμερισμάτων και των σημείων εργασίας. Επίσης, ενημερώνει τυχόν κτήρια ότι έχει γίνει αλλαγή ημέρας (Καταστήματα για αλλαγή τιμών, Κτήρια «Διασκέδασης» για αλλαγή του χαρακτηριστικού εξυπηρέτησης).

```
1.      public GameObject GetBuildingPrefab(string type,out bool valid)
2.      {
3.          valid = true;
4.          if(type.Equals("ApartmentBuilding"))
5.          {
6.              return genericBuildings[UnityEngine.Random.Range(0,
genericBuildings.Length)];
7.          }
8.          else if(type.Equals("WorkplaceBuilding"))
9.          {
10.             return genericBuildings[UnityEngine.Random.Range(0,
genericBuildings.Length)];
11.         }
12.         else if(type.Equals("SuperMarket"))
13.         {
14.             return superMarketBuildings[UnityEngine.Random.Range(0,
superMarketBuildings.Length)];
15.         }
16.         else if(type.Equals("School"))
17.         {
18.             return schoolBuildings[UnityEngine.Random.Range(0,
schoolBuildings.Length)];
19.         }
20.         else if(type.Equals("Library"))
21.         {
22.             return libraryBuildings[UnityEngine.Random.Range(0,
libraryBuildings.Length)];
23.         }
24.         else if(type.Equals("Entertainment"))
25.         {
26.             return entertainmentBuildings[UnityEngine.Random.Range(0,
entertainmentBuildings.Length)];
27.         }
28.         else
29.         {
30.             Debug.LogWarning("unknown type of building, returning generic");
31.             valid = false;
32.             return genericBuildings[UnityEngine.Random.Range(0,
genericBuildings.Length)];
33.         }
34.
35.     }
```

```
1.        public void RefreshStoreCosts()
2.        {
3.            foreach (GameObject go in builtBuildings["SuperMarket"])
4.                go.GetComponent<Store>().RefreshCosts();
5.        }
6.
7.        public void RefreshEntertainmentInterests()
8.        {
9.            foreach(GameObject go in builtBuildings["Entertainment"])
10.           {
11.               go.GetComponent<ActivitiesBuilding>().DayChange();
12.           }
13.       }
```

LampManager

Διαχειριστής φωτισμού οδών. Ένας απλοποιημένος διαχειριστής ο οποίος λαμβάνει εντολή από τον *RoadManager* λόγω του *TimeManager*. Ελέγχει τη φωταγώγηση των οδών κατά την εναλλαγή μέρας/νύκτας.

```
1.        public void SetLights(bool isOn)
2.        {
3.            List<Road> roads = roadManager.GetAllRoads();
4.
5.            foreach(Road road in roads)
6.            {
7.                road.ChangeLampActivity(isOn);
8.            }
9.
10.       }
11.
12.       public void SetLights(Road road, bool isOn)
13.       {
14.           road.ChangeLampActivity(isOn);
15.       }
```

2.2.2. Υλοποίηση ατομικών scripts

Έπειτα από την επεξήγηση των βασικών διαχειριστών, για την υλοποίηση επόμενος στόχος είναι η δημιουργία κώδικα για τις λειτουργίες σε ατομικό επίπεδο. Παρακάτω, θα γίνει αναφορά σε βασικά «Scripts» τα οποία χρησιμοποιούνται για την προσομοίωση της πόλης και των χαρακτήρων. Τα «Scripts» που θα αναφερθούν είναι:

- Person
- Family
- ApartmentBuilding
- Apartment
- WorkspaceBuilding
- Workspace
- School
- Library
- Store

- ActivitiesBuilding

Person

Ο βασικός κώδικας λειτουργίας του ατόμου. Εδώ ορίζονται όλες οι διαδικασίες και τα χαρακτηριστικά ενός ατόμου, στη προσπάθεια προσομοίωσης του με ένα πραγματικό άτομο. Διαχειρίζεται όλες τις λειτουργίες μέσα στην ημέρα, την κάλυψη προσωπικών και οικογενειακών αναγκών, την δημιουργία σχέσεων και αλληλεπιδράσεων με άλλα άτομα σε περίπτωση συσχετισμού, καθώς και τον *Animator* για τις βασικές λειτουργίες κίνησης. Περιέχει όλες τις λεπτομέρειες που δόθηκαν από τον *PopulationManager* καθώς και τα σημεία αναφοράς της οικογένειας που ανήκει και δρα σύμφωνα με αυτές.

Family

Βασικός κώδικας λειτουργίας οικογένειας. Περιέχει τα άτομα της οικογένειας, τα κλειδιά εύρεσης των γονεϊκών οικογενειών, το σύνολο των χρημάτων και του φαγητού που υπάρχει στην οικία. Λειτουργεί κυρίως ως αναφορά για την εύρεση των μελών της οικογένειας. Σε περιπτώσεις χρήσης, μια οικογένεια ορίζεται από 2 γονείς και μία λίστα (υπο)οικογενειών που ανήκουν τα παιδιά. Διαφορετικά, ορίζεται ως «ψευδοοικογένεια» και γίνεται αναφορά κυρίως ανήλικων μελών ή μελών που δεν έχουν δημιουργήσει ακόμη μία πλήρη οικογένεια βάσει των αλληλοεπιδράσεων με άλλους χαρακτήρες.

ApartmentBuilding

Κώδικας λειτουργίας κτηρίου τύπου διαμερισμάτων. Περιέχει το σύνολο των διαμερισμάτων που ανήκουν στο ίδιο κτήριο, ενώ λειτουργεί ως σημείο αναφοράς για την μετακίνηση των χαρακτήρων ως «κίνηση προς» στη προσομοίωση.

Apartment

Κώδικας λειτουργίας διαμερίσματος. Περιέχει την οικογένεια που αντιστοιχεί στο διαμέρισμα. Χρησιμοποιείται για την εμφάνιση πληροφοριών στη διεπαφή χρήστη.

WorkspaceBuilding

Κώδικας λειτουργίας κτηρίου τύπου εργασίας. Περιέχει το σύνολο των διαμερισμάτων που ανήκουν στο ίδιο κτήριο, ενώ λειτουργεί ως σημείο αναφοράς για την μετακίνηση των χαρακτήρων ως «κίνηση προς» στη προσομοίωση.

Workspace

Κώδικας λειτουργίας εργασίας. Περιέχει την ονομασία της εργασίας, τα άτομα που εργάζονται καθώς και τις λειτουργίες αλληλεπίδρασης των υπαλλήλων κατά την είσοδο τους στην εργασία. Επίσης, χρησιμοποιείται για την εμφάνιση πληροφοριών στη διεπαφή χρήστη.

School

Κώδικας λειτουργίας σχολείου. Περιέχει τους μαθητές που έχουν οριστεί σε αυτό από τον *JobManager*, περιέχει λειτουργίες αλληλεπίδρασης μαθητών, ενώ λειτουργεί ως σημείο αναφοράς για την μετακίνηση των χαρακτήρων ως «κίνηση προς» στη προσομοίωση. Επίσης, χρησιμοποιείται για την εμφάνιση πληροφοριών στη διεπαφή χρήστη.

Library

Κώδικας λειτουργίας βιβλιοθήκης. Περιέχει λειτουργίες αλληλεπίδρασης ατόμων που βρίσκονται εκείνη τη στιγμή στη βιβλιοθήκη, ενώ λειτουργεί ως σημείο αναφοράς για την μετακίνηση των χαρακτήρων ως «κίνηση προς» στη προσομοίωση. Επίσης, χρησιμοποιείται για την εμφάνιση πληροφοριών στη διεπαφή χρήστη.

Store

Κώδικας λειτουργίας καταστήματος. Περιέχει λειτουργίες αλληλεπίδρασης ατόμων που βρίσκονται εκείνη τη στιγμή στο κατάστημα, καθώς επίσης και λειτουργίες υπολογισμού χρέωσης και χρέωσης σε περίπτωση που ο χαρακτήρας αγοράσει προϊόντα για την οικογένειά του, ενώ λειτουργεί ως σημείο αναφοράς για την μετακίνηση των χαρακτήρων ως «κίνηση προς» στη προσομοίωση. Επίσης, χρησιμοποιείται για την εμφάνιση πληροφοριών στη διεπαφή χρήστη.

ActivitiesBuilding

Κώδικας λειτουργίας κτηρίου δραστηριοτήτων. Περιέχει λειτουργίες αλληλεπίδρασης ατόμων που βρίσκονται εκείνη τη στιγμή στο κτήριο, καθώς επίσης και λειτουργίες για την καθημερινή εναλλαγή του χαρακτηριστικού που εξυπηρετεί το κτήριο (FFM) και του θετικού ή αρνητικού πρόσημου που εξυπηρετεί, ενώ λειτουργεί ως σημείο αναφοράς για την μετακίνηση των χαρακτήρων ως «κίνηση προς» στη προσομοίωση. Επίσης, χρησιμοποιείται για την εμφάνιση πληροφοριών στη διεπαφή χρήστη.

2.2.3. Κατασκευή βασικών αναγκών ανθρώπου & οικογένειας

Έχοντας πλέον ορίσει της διαδικασίες λειτουργίας των διαχειριστών και του ατόμου, είναι πλέον εφικτή η επεξήγηση των λειτουργιών του χαρακτήρα σε επίπεδο ατόμου και οικογένειας. Στον κώδικα του ατόμου, έχουν οριστεί διάφορες μεταβλητές ώστε να γίνει προσέγγιση του πως κινείται και λειτουργεί ένα άτομο μέσα στην ημέρα, ανάλογα τις ανάγκες του και της οικογένειας. Οι βασικές μεταβλητές στο επίπεδο ατόμου είναι τέσσερις και αφορούν την πείνα, την υπνηλία, την ανάγκη για μάθηση και την ανάγκη για δραστηριότητες. Μέσα στην διάρκεια της ημέρας, ο χαρακτήρας χάνει ένα ποσοστό ύπνου και τροφής, με αποτέλεσμα την ανάγκη αναπλήρωσης τους μέσα στην ημέρα. Επίσης, κατά την έναρξη μιας νέας μέρας, γίνεται αύξηση των αναγκών μάθησης και δραστηριοτήτων. Σε περίπτωση που γίνει υπέρβαση ενός ορισμένου κατωφλιού, ο χαρακτήρας εισάγει στο χρονοδιάγραμμα του την ανάγκη να κινηθεί προς μία βιβλιοθήκη ή/και ένα κτήριο δραστηριοτήτων. Στο επίπεδο οικογένειας, υπάρχουν οι μεταβλητές φαγητού και χρημάτων, οι οποίες αυξομειώνονται ανάλογα την χρήση, π.χ. μείωση συνολικού φαγητού αν φάνε οι χαρακτήρες της οικογένειας, αύξηση χρημάτων αν ο χαρακτήρας πάει στη δουλειά, ή μείωση αν ο χαρακτήρας κάνει αναπλήρωση του οικογενειακού φαγητού από κατάστημα. Η συγκεκριμένη υλοποίηση χρησιμοποιεί κυρίως βασικές μεταβλητές για επίδειξη λειτουργιών και στην ενότητα «3.4. Πιθανές κατ' επέκταση χρήσεις της υλοποίησης» θα γίνει αναφορά για επεκτάσεις μου μπορούν να υλοποιηθούν.

2.2.4. Δημιουργία αόριστου χρονοδιαγράμματος

Στη προσπάθεια δημιουργίας ρεαλιστικών κινήσεων και δραστηριοτήτων «από/προς» των δημιουργημένων χαρακτήρων, προέκυψε η ανάγκη δημιουργίας ενός χρονοδιαγράμματος για τον κάθε χαρακτήρα που δεν είναι αυστηρά προκαθορισμένο. Κατά την έναρξη της ημέρας, ο κάθε χαρακτήρας δημιουργεί το δικό του χρονοδιάγραμμα σύμφωνα με τις ανάγκες του και της οικογένειας που ανήκει. Έτσι, δημιουργήθηκαν οι μεταβλητές «χρόνου που ξοδεύτηκε», όπου ορίζονται οι χρόνοι που ξοδεύει ο χαρακτήρας ανάλογα τη διαδικασία που ασχολείται.

Παραδείγματος χάριν, στην υλοποίηση έχουν υλοποιηθεί οι παρακάτω διαδικασίες με τον ανάλογο χρόνο:

- Sleep,X
- Work,4
- Shopping,2
- Read,2
- GetEntertainment,2
- Eat,2

- School,4

Με εξαίρεση την διαδικασία «Sleep» που ορίζεται ως μεταβλητή ανάλογα την ώρα που επιστρέφει ο χαρακτήρας στο διαμέρισμα του μετά το πέρας των υπόλοιπων δραστηριοτήτων, οι δραστηριότητες εισάγονται σε μία λίστα η οποία εκτελείται σειριακά. Η λίστα περιέχει διαδικασίες όπου θέλει/χρειάζεται ο χαρακτήρας, όπως εξηγήθηκε σε προηγούμενη ενότητα, όπως την αγορά τροφής για την οικογένεια. Κατά την άφιξη του χαρακτήρα σε ένα σημείο εισόδου (σύμφωνα με τη διαδικασία), ο χαρακτήρας εισάγεται σε λειτουργία αναμονής στον TimeManager και αφαιρείται από τον χάρτη. Εκεί, περιμένει μέχρι το πέρας των ωρών που αναφέρονται, όπου και δέχεται σήμα επαναφοράς για να συνεχίσει το χρονοδιάγραμμα του. Στο τέλος της ημέρας, όταν πλέον δεν υπάρχει κάποια διαδικασία, ο χαρακτήρας ξεκινάει τη διαδικασία «Sleep» η οποία ορίζεται με χρόνο λήξης την επόμενη μέρα στις 7πμ., όπου και δημιουργεί ένα νέο χρονοδιάγραμμα.

```
1.        public void DayChanged()
2.        {
3.            if(pAttr.DayChanged())
4.            {
5.                personFamily.IncrementMoney(pAttr.GetPayout());
6.            }
7.            nextPersonAction = null;
8.            pActs.Clear();
9.            bool eagerToLearn = CheckEagerness();
10.           bool wantsToBeEntertained = CheckEntertainment();
11.           if (pAttr.currentJob.GetJobName().Equals("Student"))
12.           {
13.               nextPersonAction = new
PersonActions(PersonAttributes.PersonState.Working, pAttr.GetJobLocation(),
pAttr.GetSchool(), "School", "School");
14.               if (eagerToLearn)
15.               {
16.                   Library library = mapManager.GetClosestLibrary(this);
17.                   pActs.Add(new
PersonActions(PersonAttributes.PersonState.Studying, library.GetEntrance(), library,
"Library", "Read"));
18.               }
19.
20.               pActs.Add(new PersonActions(PersonAttributes.PersonState.Idle,
personFamily.GetApartment().GetEntrance(),null, "Apartment", "Eat"));
21.           }
22.           else
23.           {
24.               nextPersonAction = new
PersonActions(PersonAttributes.PersonState.Working, pAttr.GetJobLocation(),
pAttr.GetWorkplace(), "Workplace", "Work") ;
25.               if (eagerToLearn)
26.               {
27.                   Library library = mapManager.GetClosestLibrary(this);
28.                   pActs.Add(new
PersonActions(PersonAttributes.PersonState.Studying, library.GetEntrance(), library,
"Library", "Read"));
29.               }
30.
31.               pActs.Add(new PersonActions(PersonAttributes.PersonState.Idle,
personFamily.GetApartment().GetEntrance(), null, "Apartment", "Eat"));
32.
33.               if (!HasPseudoFamily())
34.               {
35.                   if (personFamily.NeedFood())
36.                   {
37.                       personFamily.SetToBuyFood();
```

```
38.
39.                         Store store = mapManager.GetClosestStore(this);
40.
41.                         int cost =
store.PreCalculateCost(GetFamilyCountForStore());
42.                         if (personFamily.totalMoney > cost)
43.                             pActs.Add(new
PersonActions(PersonAttributes.PersonState.Shopping, store.GetEntrance(),store,
"SuperMarket", "Shopping"));
44.                         else
45.                         {
46.                             store = mapManager.GetCheapestStore(this);
47.                             pActs.Add(new
PersonActions(PersonAttributes.PersonState.Shopping, store.GetEntrance(),store,
"SuperMarket", "Shopping"));
48.                         }
49.                     }
50.                 }
51.
52.                 if (wantsToBeEntertained)
53.                 {
54.                     ActivitiesBuilding actB =
mapManager.GetClosestEntertainment(this);
55.                     pActs.Add(new
PersonActions(PersonAttributes.PersonState.GettingEntertained,
actB.GetEntrance(),actB, "Entertainment", "GettingEntertained"));
56.                 }
57.
58.             }
59.
60.             pActs.Add(new PersonActions(PersonAttributes.PersonState.Idle,
personFamily.GetApartment().GetEntrance(), null, "Apartment", "Sleep"));
61.             StartNewAction();
62.         }
```

δημιουργία χρονοδιαγράμματος

2.3. Εισαγωγή εξωτερικών πακέτων και διαδικασιών

Όπως εξηγήσαμε και στην εισαγωγική ενότητα, η πλατφόρμα ανάπτυξης Unity3D συνδέεται ισχυρά με τη πλατφόρμα που έχει αναπτύξει η ίδια για την εύρεση μοντέλων και εξωτερικών πακέτων για ευκολότερες και πιο γρήγορες υλοποιήσεις. Μέσα από τη πλατφόρμα AssetStore θα καλυφθούν οι ανάγκες της υλοποίησης της πόλης με εύρεση μοντέλων οικοδομικών τετραγώνων, κτηρίων, δρόμων, ανθρωπόμορφων χαρακτήρων καθώς και η βασική λειτουργία OCEAN μέσω του πακέτου Love/Hate και η λειτουργία NavMeshAgent για την πλοήγηση των χαρακτήρων.

2.3.1. Εισαγωγή μοντέλων για τη δημιουργία της πόλης

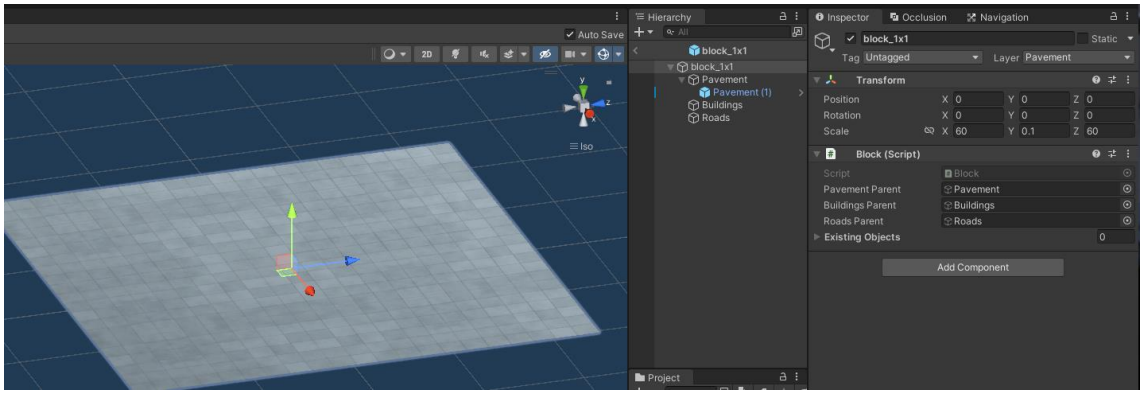
Γνωρίζοντας τις ανάγκες τις υλοποίησης, η διαδικασία για την κατασκευή, τουλάχιστον εμφανισιακά, χωρίζεται στην εύρεση μοντέλων για τις οδούς και τα οικοδομικά τετράγωνα, καθώς και των κτηρίων. Χρησιμοποιώντας την αναζήτηση στο AssetStore, βρέθηκαν και χρησιμοποιήθηκαν τα παρακάτω πακέτα:

- City Voxel Pack
- Modular Lowpoly Streets
- City 'Traffic Lights' Pack
- Polygon City Pack

Για την υλοποίηση παραμετροποιήθηκαν κατάλληλα τα οικοδομικά τετράγωνα για να καλύψουν διαφορετικά μεγέθη περιοχών (από 60x60UU μέχρι 180x180UU), διαμορφώθηκαν οι δρόμοι για την εισαγωγή φωτεινής σηματοδότησης και φωτισμού και ενημερώθηκαν κατάλληλα τα κτήρια που θα χρησιμοποιηθούν για την λειτουργία της προσομοίωσης.

Οικοδομικά τετράγωνα:

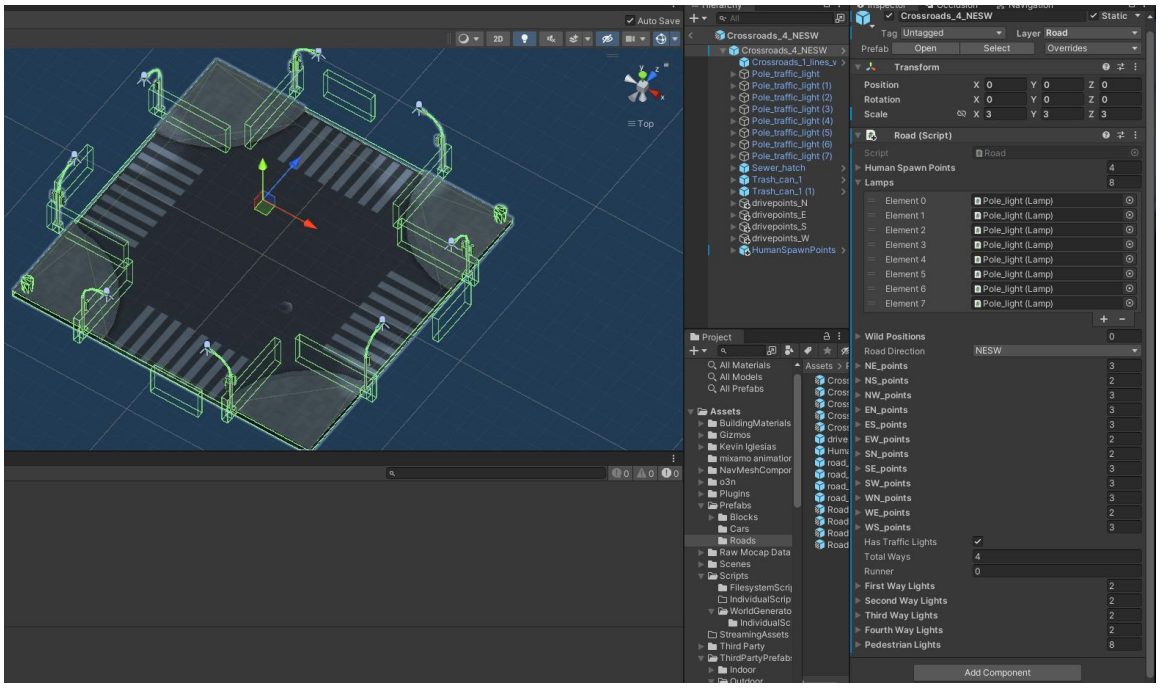
Διαμορφώθηκαν κατάλληλα για την εισαγωγή πληροφοριών στον κώδικα του και έγινε η εισαγωγή *GameObjects* για καλύτερο διαχωρισμό των αντικειμένων που θα περιέχει.



Εικόνα 7: ένα βασικό τετράγωνο μεγέθους 1x1

Δρόμοι:

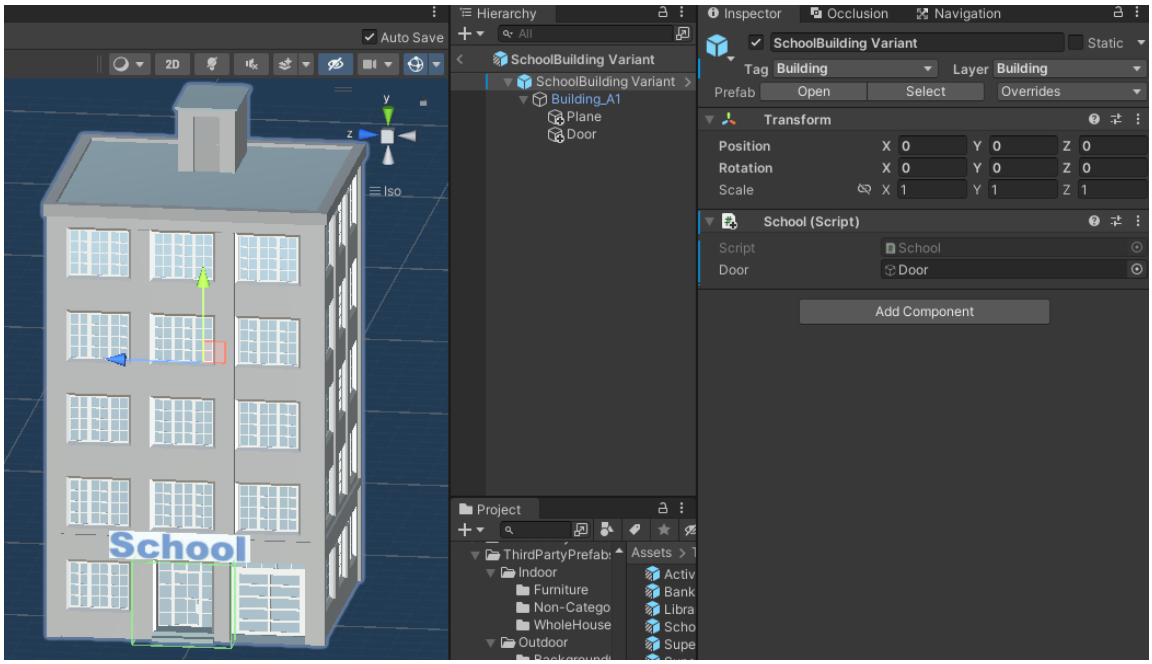
Διαμορφώθηκαν κατάλληλα τα υπάρχοντα *Prefabs* που υπάρχουν στο πακέτο LowPoly Streets και έγινε η εισαγωγή του κώδικα «Road» που αποτελεί μέρος της υλοποίησης για τη μετακίνηση των οχημάτων και την φωτεινή σηματοδότηση



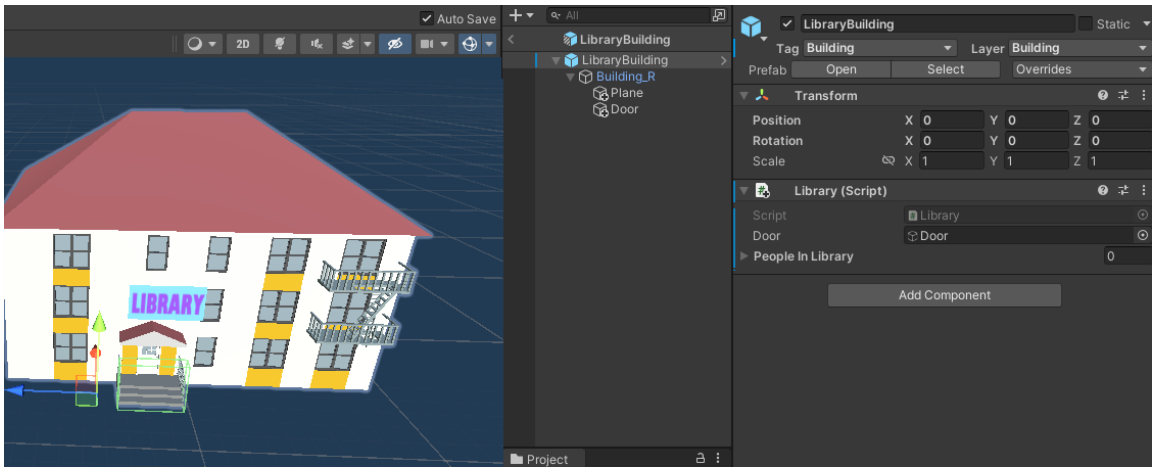
Εικόνα 8: Παράδειγμα διασταύρωσης οδών με σηματοδότες

Κτήρια:

Δημιουργήθηκαν τα κτήρια που θα χρησιμοποιηθούν και ορίστηκαν τα σημεία εισόδου των χαρακτήρων. Στην περίπτωση των κτηρίων με διαμερίσματα ή εργασίες, γίνεται η εισαγωγή των ανάλογων scripts «*ApartmentBuilding*» και «*WorkplaceBuilding*» μέσω του διαχειριστή «BuildingManager».



Εικόνα 9: Κτήριο που χρησιμοποιείται ως σχολείο

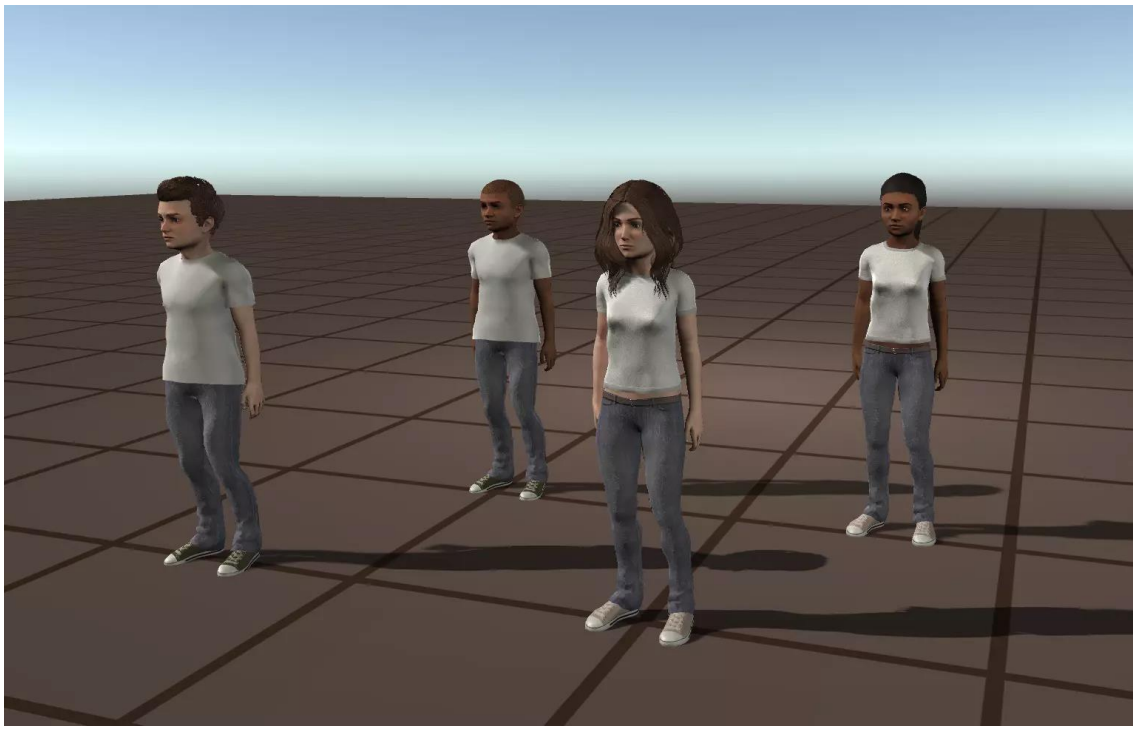


Εικόνα 10: Κτήριο που χρησιμοποιείται ως βιβλιοθήκη

2.3.2. Υλοποίηση του πρόσθετου πακέτου UMA

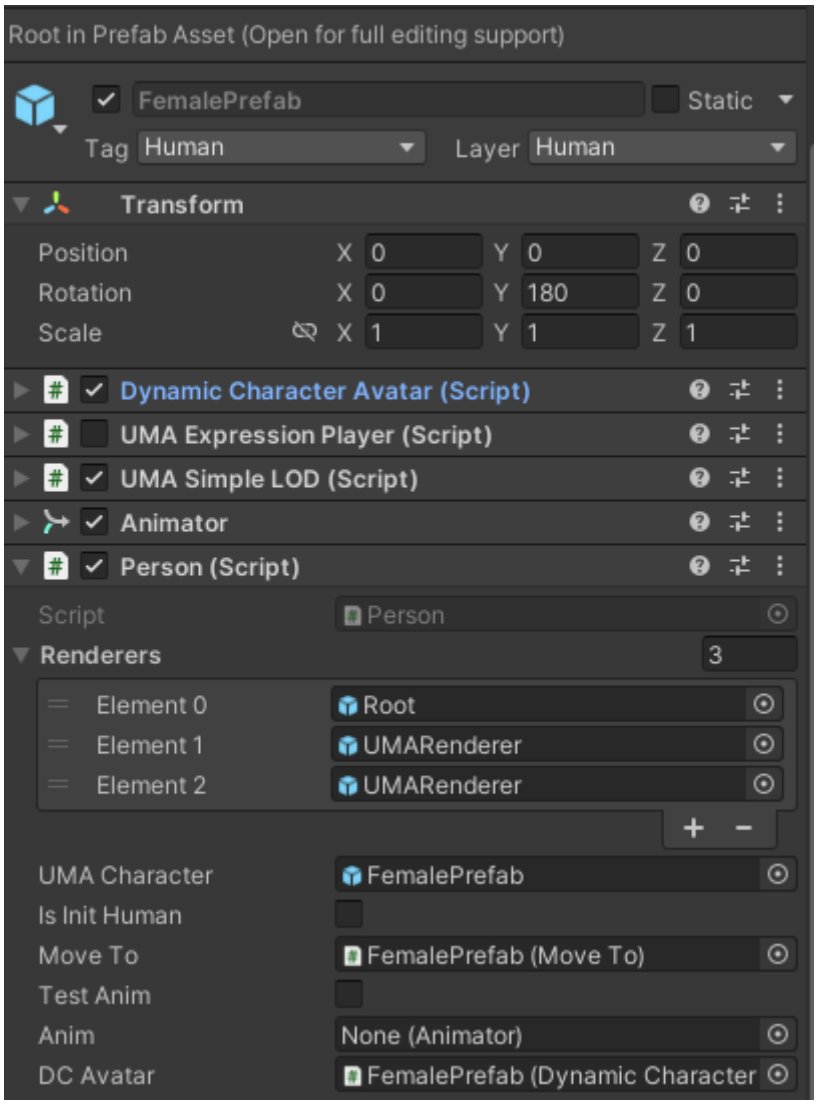
Στην πλατφόρμα AssetStore υπάρχει ένα σύνολο πακέτων με μοντέλα χαρακτήρων, έτοιμα για χρήση. Ένα από αυτά, παρέχει δυνατότητα παραμετροποίησης των χαρακτήρων χειροκίνητα ή ακόμη και διαδικαστικά. Ονομάζεται «UMA 2», δηλαδή «Unity Multipurpose Avatar» και διατίθεται δωρεάν. Σε συνδυασμό με το πακέτο UMA 2, στην υλοποίηση θα χρησιμοποιηθεί και το πακέτο «o3n Male and Female UMA Races» το οποίο παρέχει περαιτέρω αυτόματες παραμετροποιήσεις χαρακτήρων, όπως την αυτόματη ρύθμιση χαρακτήρα παιδικής ηλικίας.

Εικόνα 11: Δείγμα χαρακτήρων o3n, assetstore.unity.com



Εικόνα 12: δείγμα παιδικών χαρακτήρων, assetstore.unity.com

Στη συνέχεια, σύμφωνα με τις οδηγίες που αναφέρονται κατά την εισαγωγή των πακέτων UMA και o3n οριστικοποιούμε τις ρυθμίσεις των εξωτερικών πακέτων. Χρησιμοποιώντας τα *Prefabs* που δίνονται, γίνεται η εισαγωγή του *script* «Person» καθώς επίσης του εξωτερικού κώδικα «Traits» που χρησιμοποιείται στη διαδικασία αρχικοποίησης του μοντέλου OCEAN ανά χαρακτήρα, καθώς και του *Component* «NavMeshAgent» που θα χρησιμοποιηθεί για την πλοήγηση του χαρακτήρα.

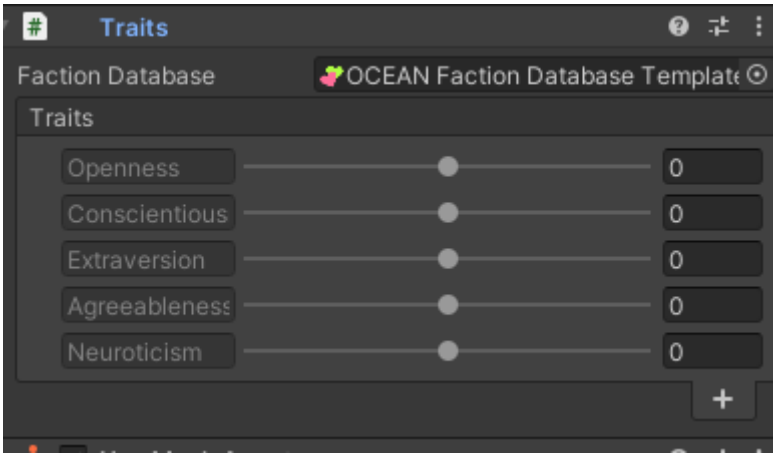


Εικόνα 13: Το femalePrefab όπως είναι αυτή τη στιγμή

2.3.3. Υλοποίηση του μοντέλου OCEAN στο πρόσθετο πακέτο LoveHate

Χρησιμοποιώντας το πακέτο Love/Hate το οποίο προσφέρεται στη πλατφόρμα assetstore, γίνεται εφικτή η εισαγωγή διαφόρων μοντέλων χαρακτηριστικών στους χαρακτήρες ανάλογα τις απαιτήσεις του δημιουργού. Για την υλοποίηση της διπλωματικής, χρησιμοποιήθηκε μια μορφή του μοντέλου OCEAN, στο οποίο ο δημιουργός μπορεί να εισάγει δεδομένα ανάλογα τον χαρακτήρα που θέλει να δημιουργήσει, ή ακόμη και να τα ορίσει διαδικαστικά. Το εύρος κάθε χαρακτηριστικού είναι 200 μονάδες, ορίζεται από -100 έως 100. Όπως και στο κεφάλαιο «1.4. Το μοντέλο OCEAN», η ονομασία των χαρακτηριστικών παραμένει σε αυτές επιπέδου χαρακτηριστικού, χωρίς αναφορά των άκρων σε -100 και 100, ενώ δε προσφέρεται επιπλέον διαχωρισμός της προσωπικότητας ενός χαρακτήρα μέσω μοντέλων που έχουν υλοποιηθεί με

χρήση του μοντέλου OCEAN. Επιπλέον, για λόγους απλούστευσης της υλοποίησης της διπλωματικής, επιλέχθηκε η εισαγωγή τυχαίων βαθμών σε κάθε χαρακτηριστικό κατά τη διαδικαστική δημιουργία ενός χαρακτήρα ή την επιλογή μεταξύ του εύρους των γονέων του χαρακτήρα. Στην ενότητα 3.4 θα αναλυθούν τρόποι επέκτασης της εφαρμογής του μοντέλου OCEAN.

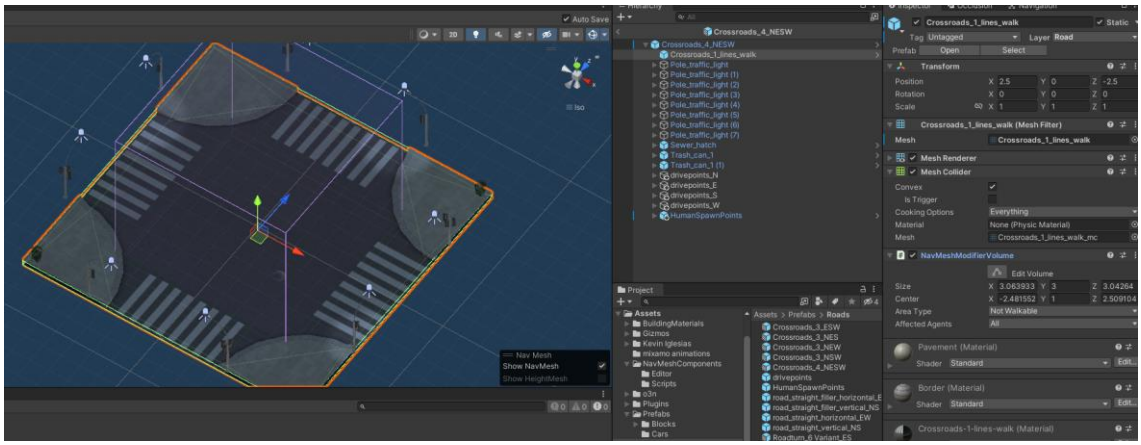


Εικόνα 14: Script ορισμού χαρακτηριστικών σε μηδενική κατάσταση

Παράλληλα, όπως έγινε αναφορά και κατά τον ορισμό της υλοποίησης του *script* «Person» και του script «HumanCreator», ο χαρακτήρας χρησιμοποιεί τα χαρακτηριστικά για το πώς θα δημιουργήσει το χρονοδιάγραμμα του ώστε να καλύψει τις ανάγκες του.

2.3.4. Υλοποίηση του συστήματος πλοήγησης NavMeshAgent

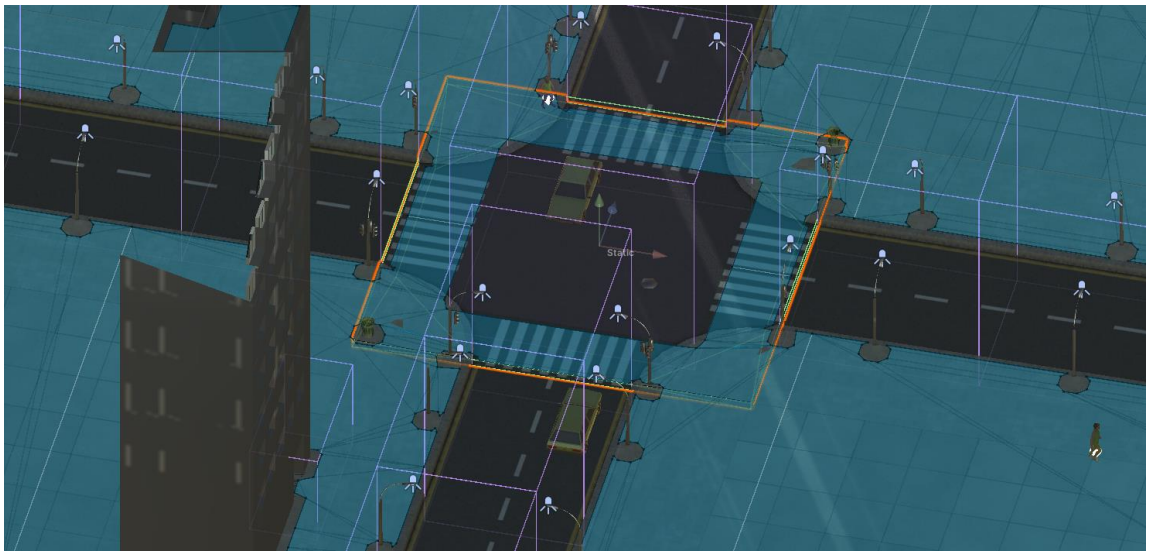
Λόγω της ανάγκης κίνησης των χαρακτήρων δυναμικά μέσω της δημιουργίας χρονοδιαγράμματος, όπως επίσης και της διαδικαστικής κατασκευής της πόλης και της τυχαιότητας τοποθέτησης κτηρίων σε όλο το εύρος της, καθίσταται αδύνατη η δημιουργία σταθερών μονοπατιών για τους χαρακτήρες ώστε αν μετακινηθούν από ένα σημείο σε ένα άλλο. Η πλατφόρμα Unity3D, έχοντας αναπτύξει πράκτορες τεχνητής νοημοσύνης για εύρεση μονοπατιών, προσφέρει τη δυνατότητα δημιουργίας μονοπατιών, τα οποία όμως δημιουργούνται πριν την εκτέλεση της υλοποίησης. Για να καλυφθεί η ανάγκη της διπλωματικής, θα χρησιμοποιηθεί ένα δοκιμαστικό πακέτο που παρέχει την κατασκευή μονοπατιών και ορίων κατά την εκτέλεση. Το πακέτο ονομάζεται «*NavMeshComponents*» και παρέχεται μέσω της Unity3D στο GitHub. Το πακέτο προσφέρει ένα εύρος *scripts* τα οποία θα χρησιμοποιηθούν για την παραμετροποίηση των δρόμων, των κτηρίων και των πεζοδρομίων ώστε να παρέχουν την ανάλογη λειτουργικότητα κατά την εκτέλεση.



Εικόνα 15: Εισαγωγή του "NavMeshModifierVolume" (δεξιά), με ορισμό κύβου με μωβ χρώμα (αριστερά)

Χρησιμοποιώντας κατάλληλα τα εργαλεία, θα δημιουργηθούν δρόμοι και κτήρια τα οποία ενώ επιτρέπουν την κίνηση των χαρακτήρων σε όλο το εύρος των αντικειμένων, απαγορεύει την κίνηση εντός των ορισμένων περιοχών μέσω του *NavMeshModifierVolume*.

Για τη λειτουργία της περιοχής στο σύνολο του χάρτη, δημιουργήθηκε ένα κενό αντικείμενο με όνομα *NavMesh* και ορίστηκε μέσα στον διαχειριστή *MapManager.* Από εκεί, κατά τη δημιουργία ο *MainManager* θα εκτελέσει τη μέθοδο *BuildNavMesh()* στη δεύτερη φάση της ενεργοποίησης και θα κατασκευάσει την περιοχή μετακίνησης.



Εικόνα 16:Δημιουργημένο NavMesh κατά την εκτέλεση της υλοποίησης

Τέλος, έγινε η εισαγωγή του *Component* NavMeshAgent στα *Prefabs* των χαρακτήρων ώστε να δοθεί η δυνατότητα κίνησης στους χαρακτήρες με χρήση του NavMesh.

2.4. Επεξήγηση σημείων διαδικαστικής τυχαιότητας

Έχοντας εξηγήσει τις βασικές λειτουργίες των διαχειριστών καθώς και του κώδικα που βρίσκεται στα αντικείμενα που θα χρησιμοποιηθούν κατά την εκτέλεση της υλοποίησης, κρίνεται δυνατό να γίνει η επεξήγηση των δυνατοτήτων που παρέχονται από την διαδικαστική κατασκευή της πόλης και των παραμέτρων που διαφοροποιούνται κατά την εκτέλεση, σε επίπεδο χαρακτήρα, οικογένειας και επιμέρους χαρακτηριστικών. Η διαδικασία αναφέρεται επίσης ως «Randomization» λόγω της φύσης της να παράγει τυχαία αποτελέσματα κατά την επιστροφή δεδομένων που ζητήθηκαν.

2.4.1. Παραμετροποίηση ανθρώπων

Όπως αναφερθήκαμε και στις προηγούμενες ενότητες, κατά τη διαδικασία της δημιουργίας, ένας χαρακτήρας περνάει από διάφορα επίπεδα στα οποία τυχαιοκρατικά αποκτά δεδομένα τα οποία χρειάζονται για την λειτουργία του. Ένας χαρακτήρας κατά τη δημιουργία του αποκτά διαδικαστικά:

- Φύλο
- Ηλικία
- Ενδυμασία και χρώμα μαλλιών & δέρματος
- Χαρακτηριστικά OCEAN
- Κατοικία
- Εργασία

Ενώ κατά την λειτουργία του επιλέγεται τυχαία η απόφαση να ασχοληθεί με δραστηριότητες ή διάβασμα, καθώς και το ποσοστό του να δημιουργήσει νέες αλληλεπιδράσεις με άλλους χαρακτήρες.

Αναλύοντας, κατά την δημιουργία του χαρακτήρα, ο HumanCreator αποφασίζει την ενδυμασία ενός ατόμου αφού πρώτα γνωστοποιηθεί το φύλο που θα χρησιμοποιηθεί, όπως φαίνεται στο τμήμα κώδικα παρακάτω:

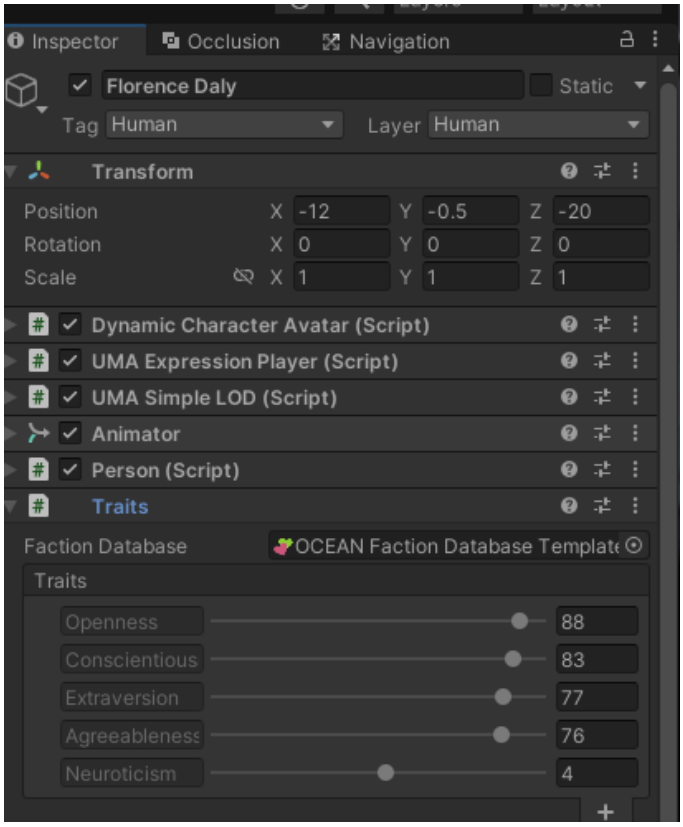
```
1.     void DressHuman(Person person,Gender gender,Family family)
2.         {
3.             DynamicCharacterAvatar dca =
person.gameObject.GetComponent<DynamicCharacterAvatar>();
4.             if(supplSkinColor.r ==0)
5.             {
6.                 supplSkinColor = new Color(1 - skinColor.r, 1 - skinColor.g, 1 -
skinColor.b, 1);
7.             }
8.
9.             if (gender.Equals(Gender.Male))
10.            {
11.                dca.SetSlot(maleShirts[Random.Range(0, maleShirts.Count)]);
12.                dca.SetSlot(malePants[Random.Range(0, malePants.Count)]);
13.                dca.SetSlot(maleHairs[Random.Range(0, maleHairs.Count)]);
14.                dca.SetSlot(maleShoes[Random.Range(0, maleShoes.Count)]);
15.                dca.SetSlot(socks[Random.Range(0, socks.Count)]);
16.
17.                if (Random.Range(0,2)==1&&family==null)
18.                    dca.SetSlot(maleFacialHairs[Random.Range(0,
maleFacialHairs.Count)]);
19.
20.
21.                dca.SetColor(clothingTopName, new Color(Random.Range(0f, 1f),
Random.Range(0f, 1f), Random.Range(0f, 1f), 1f));
22.                dca.SetColor(clothingBottomName, new Color(Random.Range(0f, 1f),
Random.Range(0f, 1f), Random.Range(0f, 1f), 1f));
23.                dca.SetColor(clothingSocksName, new Color(Random.Range(0f, 1f),
Random.Range(0f, 1f), Random.Range(0f, 1f), 1f));
24.                dca.SetColor(clothingShoesINITName+"01", new
Color(Random.Range(0f, 1f), Random.Range(0f, 1f), Random.Range(0f, 1f), 1f));
25.
26.
27.                dca.ReapplyWardrobeCollections();
28.            }
29.            else
30.            {
31.                if (Random.Range(0, 2) == 1)
32.                {
33.                    dca.SetSlot(femaleDresses[Random.Range(0,
femaleDresses.Count)]);
34.                    dca.SetColor(clothingTopName, new Color(Random.Range(0f, 1f),
Random.Range(0f, 1f), Random.Range(0f, 1f), 1f));
35.                }
36.                else
37.                {
38.                    dca.SetSlot(femaleShirts[Random.Range(0,
femaleShirts.Count)]);
39.                    dca.SetSlot(femalePants[Random.Range(0, femalePants.Count)]);
40.                    dca.SetColor(clothingTopName, new Color(Random.Range(0f, 1f),
Random.Range(0f, 1f), Random.Range(0f, 1f), 1f));
41.                    dca.SetColor(clothingBottomName, new Color(Random.Range(0f,
1f), Random.Range(0f, 1f), Random.Range(0f, 1f), 1f));
42.                }
43.                //shoes
44.                UMATextRecipe utrShoes = femaleShoes[Random.Range(0,
femaleShoes.Count)];
45.                dca.SetSlot(utrShoes);
46.                Color color = new Color(Random.Range(0f, 1f), Random.Range(0f,
1f), Random.Range(0f, 1f), 1f);
47.                if (utrShoes.DisplayValue.Equals("High Heels"))
48.                {
```

```
49.                     dca.SetColor(clothingShoesINITName + "01", color);
50.                     dca.SetColor(clothingShoesINITName + "02", color);
51.                     dca.SetColor(clothingShoesINITName + "03", color);
52.                     dca.SetColor(clothingShoesINITName + "04", color);
53.                 }
54.                 else
55.                     dca.SetColor(clothingShoesINITName + "01", color);
56.
57.                 dca.SetSlot(femaleHairs[Random.Range(0, femaleHairs.Count)]);
58.                 dca.SetSlot(socks[Random.Range(0, socks.Count)]);
59.                 dca.SetColor(clothingSocksName, new Color(Random.Range(0f, 1f),
Random.Range(0f, 1f), Random.Range(0f, 1f), 1f));
60.
61.                 dca.ReapplyWardrobeCollections();
62.             }
63.
64.
65.
66.             if (family != null)
67.             {
68.                 DynamicCharacterAvatar dcap1 =
family.GetParent(0).gameObject.GetComponent<DynamicCharacterAvatar>();
69.                 DynamicCharacterAvatar dcap2 =
family.GetParent(1).gameObject.GetComponent<DynamicCharacterAvatar>();
70.                 Color parent1SkinColor = dcap1.GetColor("Skin").color;
71.                 Color parent1HairColor = dcap1.GetColor("Hair").color;
72.                 Color parent2SkinColor = dcap2.GetColor("Skin").color;
73.                 Color parent2HairColor = dcap2.GetColor("Hair").color;
74.                 Color mixedHairColor = Random.Range(0, 2) == 1 ? parent1HairColor
: parent2HairColor;
75.                 Color mixedSkinColor = new Color((parent1SkinColor.r +
parent2SkinColor.r) * 0.5f, (parent1SkinColor.g + parent2SkinColor.g) * 0.5f,
(parent1SkinColor.b + parent2SkinColor.b) * 0.5f, 1);
76.                 dca.SetColor("Skin", mixedSkinColor);
77.                 dca.SetColor("Hair", mixedHairColor);
78.             }
79.             else
80.             {
81.                 float skinR = Random.Range(0f, 1f);
82.
83.                 Color color1 = new Color(skinColor.r + supplSkinColor.r * skinR,
skinColor.g + supplSkinColor.g * skinR, skinColor.b + supplSkinColor.b * skinR, 1);
84.                 Color color2 = naturalHairColor[Random.Range(0,
naturalHairColor.Count)];
85.                 dca.SetColor("Skin", color1);
86.                 dca.SetColor("Hair", color2);
87.             }
88.             dca.UpdateColors();
89.         }
```

Σε περίπτωση εκτέλεσης, κατά την μαζική παραγωγή ανθρώπων, το αποτέλεσμα θα είναι παρόμοιο με το παρακάτω:

Εικόνα 17: διαδικαστική παραγωγή γυναικείων χαρακτήρων

Στη συνέχεια, ο χαρακτήρας εισάγεται στον OCEAN Manager, ο οποίος θέτει τυχαία τις τιμές των χαρακτηριστικών:



Εικόνα 18: Τυχαίες τιμές στο μοντέλο OCEAN

Τέλος, γίνεται η τυχαία επιλογή διαμερίσματος, και ορίζεται τυχαία η εργασία μέσα από μία λίστα εργασιών.

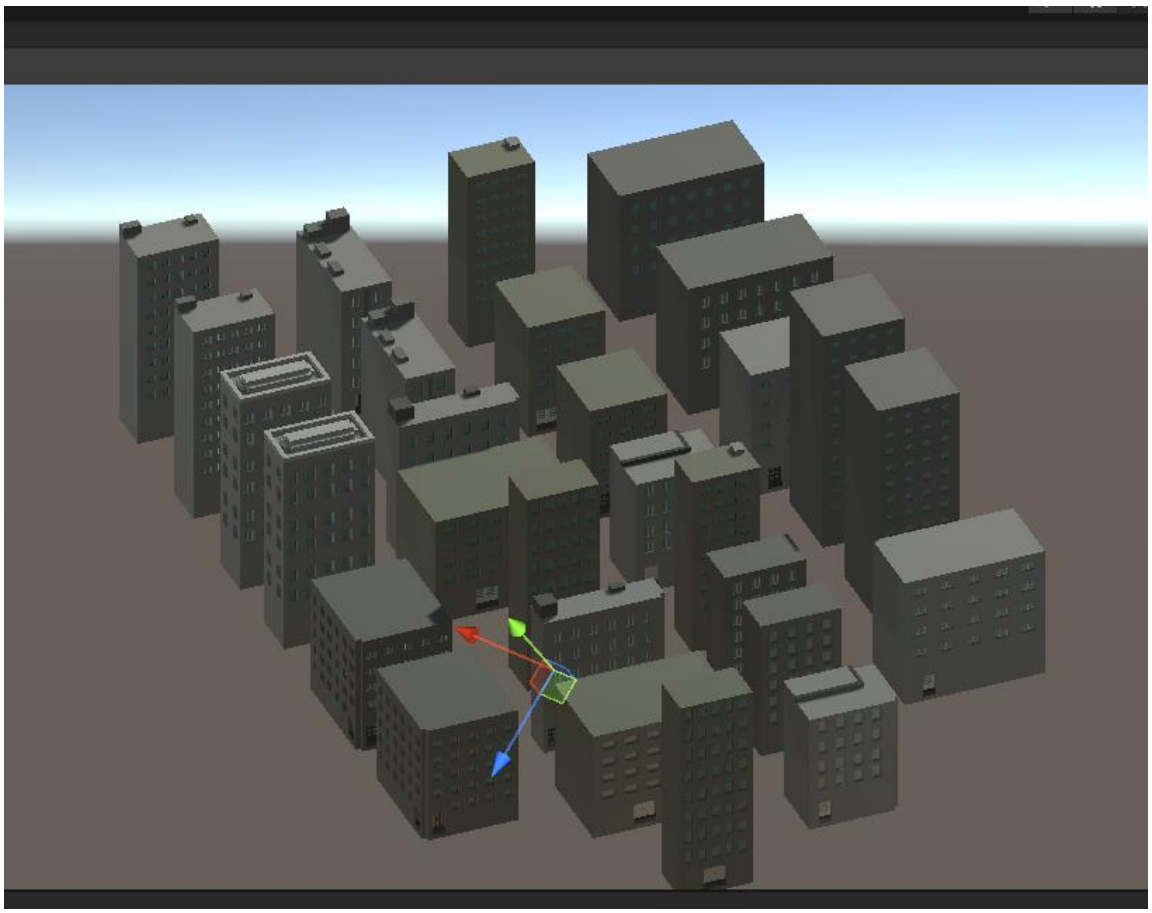
2.4.2. Παραμετροποίηση οικογένειας

Κατά τη δημιουργία των ατόμων από τον διαχειριστή «PopulationManager», ιδιαίτερα κατά την αρχικοποίηση του συστήματος, γίνεται η τυχαία επιλογή των παιδιών που έχει μία οικογένεια. Αυτό έχει ως αποτέλεσμα, ειδικά σε συνδυασμό με τις μεταβλητές που ορίζουν το πλήθος των αρχικών οικογενειών, καθώς και του πλήθους των ενδιάμεσων γενεών που θα παραχθούν, να αλλάζει δραματικά το τελικό πλήθος των δύο τελευταίων γενεών που παραμένουν ενεργοί στη πόλη. Ενδεικτικά, οι βασικές ρυθμίσεις με 20 οικογένειες και 2 ενδιάμεσες γενιές με 3 το πολύ παιδιά, δημιουργούν ένα πλήθος του εύρους των 300 ατόμων με περίπου 80 οικογένειες, ενώ μια εκτέλεση με 40 αρχικές οικογένειες και 3 ενδιάμεσες γενιές μπορεί να φτάσει πάνω από τις 300 οικογένειες και τα 1200 ενεργά άτομα στη πόλη.

Εικόνα 19: Αρχικές ρυθμίσεις του PopulationManager

2.4.3. Παραμετροποίηση κτηρίων

Κατά τη δημιουργία της πόλης, γίνεται η τυχαία επιλογή των κτηρίων που θα χρησιμοποιηθούν για την εγκατάσταση κατοίκων καθώς και την έναρξη εργασιών. Επίσης, η επιλογή ενός κτηρίου γίνεται με τυχαίο τρόπο, όπως επίσης και κατά την επιλογή εργασίας, αν υπάρχει ήδη παρόμοια εργασία ενεργή στον χάρτη, υπάρχει μία μικρή πιθανότητα να δημιουργηθεί μια καινούρια αντί της εισαγωγής του ατόμου στην ήδη υπάρχουσα.



Εικόνα 20: Τα κτήρια που χρησιμοποιήθηκαν για διαμερίσματα και εργασίες

Τέλος, σε επίπεδο λειτουργίας, τα κτήρια τύπου «Store» (Καταστήματα αγοράς) και τα «ActivitiesBuilding» (Κτήρια δραστηριοτήτων) περιέχουν λειτουργίες αλλαγής τιμών και αλλαγής εξυπηρέτησης χαρακτηριστικού σε καθημερινή βάση, επιλέγοντας αντίστοιχα από ένα εύρος τιμών και από το σύνολο των χαρακτηριστικών ορίζοντας αρνητικό ή θετικό πρόσημο.

2.4.4. Τυχαιότητα αλληλεπιδράσεων

Για την αλληλοεπίδραση των χαρακτήρων, χρησιμοποιήθηκε η πιθανότητα του να βρεθεί ένας χαρακτήρας σε έναν χώρο. Παραδείγματος χάριν, σε περίπτωση που ένας χαρακτήρας μπει στον χώρο εργασίας του, υπάρχει μία πιθανότητα να συνομιλήσει (δηλ. να αλληλοεπιδράσει) με έναν άλλο υπάλληλο που βρίσκεται επίσης στον χώρο. Η πιθανότητα αλληλεπίδρασης ορίζεται από το παρακάτω κομμάτι κώδικα:

```
1.        public void CameToWork(Person person)
```

```
2.        {
3.            if (staffActiveInWorkplace == null)
4.                staffActiveInWorkplace = new List<Person>();
5.            foreach(Person p in staffActiveInWorkplace)
6.            {
7.                if (UnityEngine.Random.Range(0, 20) == 0)
8.                    person.NewInteraction(p, true);
9.            }
10.           staffActiveInWorkplace.Add(person);
11.       }
```

Όπως φαίνεται, η πιθανότητα να αλληλοεπιδράσουν δύο χαρακτήρες που βρίσκονται στην εργασία είναι 5%. Παρόμοιος κώδικας εκτελείται και στις περιπτώσεις που:

- Ένας μαθητής πάει στο σχολείο (10%)
- Ένας χαρακτήρας μπει στο κατάστημα για να αγοράσει φαγητό (10%)
- Ένας χαρακτήρας πάει για διάβασμα σε μία βιβλιοθήκη (3.3%)
- Ένας χαρακτήρας πάει σε ένα κτήριο δραστηριοτήτων (20%)

Ενώ ο κώδικας που ελέγχει την αλληλεπίδραση χωρίζεται σε δύο μέρη τα οποία εκτελούνται ανάλογα το ποιος χαρακτήρας έκανε την έναρξη της αλληλεπίδρασης:

```
1.        public void NewInteraction(Person otherPerson,bool initiator)
2.        {
3.            if (initiator)
4.            {
5.                if (!personInteractions.ContainsKey(otherPerson.PID))
6.                {
7.                    Interaction interaction = new Interaction(otherPerson.PID,
otherPerson.GetFullName(), Random.Range(0, 21));
8.                    personInteractions.Add(otherPerson.PID, interaction);
9.                }
10.               bool chatResult =
personInteractions[otherPerson.PID].ChatResult();
11.               bool otherPersonChatResult = otherPerson.ContinueInteraction(this,
chatResult);
12.               personInteractions[otherPerson.PID].UpdateProgress(chatResult ||
otherPersonChatResult);
13.               if(personInteractions[otherPerson.PID].MaximumLikeness() &&
personInteractions[otherPerson.PID].CanBeEngaged(this,otherPerson)&&otherPerson.Want
sToBeEngaged(this))
14.               {
15.                   MergePseudoFamilies(this,otherPerson);
16.                   mapManager.CreateRandomApartment(personFamily);
17.                   CreateChildren(personFamily);
18.               }
19.           }
20.           else //wait forInteraction fromOtherUser
21.           {
22.           }
23.       }
```

Και:

```
1.        bool ContinueInteraction(Person otherPerson, bool otherPersonChatResult)
2.        {
3.            if (!personInteractions.ContainsKey(otherPerson.PID))
4.            {
5.                Interaction interaction = new Interaction(otherPerson.PID,
otherPerson.GetFullName(), Random.Range(0, 21));
6.                personInteractions.Add(otherPerson.PID, interaction);
7.            }
8.            bool chatResult = personInteractions[otherPerson.PID].ChatResult();
9.            personInteractions[otherPerson.PID].UpdateProgress(chatResult ||
otherPersonChatResult);
10.
11.
12.           //now let the firstPerson continue.
13.           return chatResult;
14.       }
```

Σε περίπτωση που η αλληλεπίδραση γίνεται για πρώτη φορά, υπάρχει τυχαιότητα η οποία ορίζει την «πρώτη εντύπωση» μεταξύ δύο χαρακτήρων και έχει 5% πιθανότητα να είναι κακή, και 5% να είναι καλή, με το υπόλοιπο ποσοστό να είναι σε φυσιολογικό ποσοστό. Αν η αλληλεπίδραση μεταξύ δύο ατόμων έχει ξανασυμβεί, αυξάνεται η εντύπωση των χαρακτήρων ανάλογα με ένα ποσοστό το οποίο συνδέεται με την επιτυχία της κάθε αλληλεπίδρασης, ενώ σε περίπτωση που υπάρχει πλήρως καλή εντύπωση και στα δύο άτομα, αν υπάρχει η δυνατότητα γίνεται διασύνδεση των «ψευδοοικογενειών» που ανήκουν οι δύο χαρακτήρες, δημιουργείται νέα οικογένεια και δημιουργούνται νέοι χαρακτήρες ως παιδιά της νέας οικογένειας.

2.4.5. Δημιουργία και κίνηση οχημάτων

Κατά την κατασκευή των οχημάτων από τον *VehicleManager* γίνεται η τυχαία επιλογή ενός σημείου έναρξης επιλέγοντας ένα αντικείμενο δρόμου μέσω του *RoadManager* καθώς επίσης και του σημείου προορισμού. Στη συνέχεια, υπολογίζεται η βέλτιστη διαδρομή για να μπορέσει να καταλήξει το όχημα στον προορισμό του και γίνεται και ο υπολογισμός των καμπυλών Bezier σε περίπτωση που υπάρχουν στροφές ώστε να φαίνεται ρεαλιστική η κίνηση του οχήματος. Το όχημα μετακινείται από σημείο σε σημείο σύμφωνα με τη διαδρομή που περιέχει πληροφορία θέσης και ποσοστού στρέψης και παρακολουθεί τη φωτεινή σηματοδότηση, ενώ κατά την άφιξη του, επιλέγεται ένα νέο τυχαίο σημείο ως νέο σημείο προορισμού, και το παλιό σημείο προορισμού ορίζεται πλέον ως η νέα έναρξη του οχήματος και υπολογίζεται η νέα διαδρομή.

Εικόνα 21: Μία διαδρομή ενός τυχαίου οχήματος

3. ΕΚΤΕΛΕΣΗ

3.1. Εκτέλεση προσομοίωσης

Έχοντας πλέον ολοκληρώσει την κατασκευή του κώδικα στο σύνολο της υλοποίησης, πλέον είναι δυνατή η εκτέλεση της προσομοίωσης της πόλης. Στο κεφάλαιο αυτό θα γίνει η αναφορά στη διεπαφή του χρήστη, τα μενού που εμφανίζονται με τις ρυθμίσεις που περιλαμβάνουν, τους χρόνους ταχύτητας εκτέλεσης, την κίνηση της κάμερας και τις πληροφορίες που εμφανίζονται για τους ενεργούς χαρακτήρες στη πόλη. Σε περίπτωση που η εκτέλεση γίνεται μέσω της πλατφόρμας Unity3D σε κατάσταση Editor, μπορούμε με να εκκινήσουμε την προσομοίωση μέσω του “Play”, διαφορετικά μέσω της έναρξης του εκτελέσιμου αρχείου που έχει παραχθεί από τη διαδικασία Build της Unity3D.

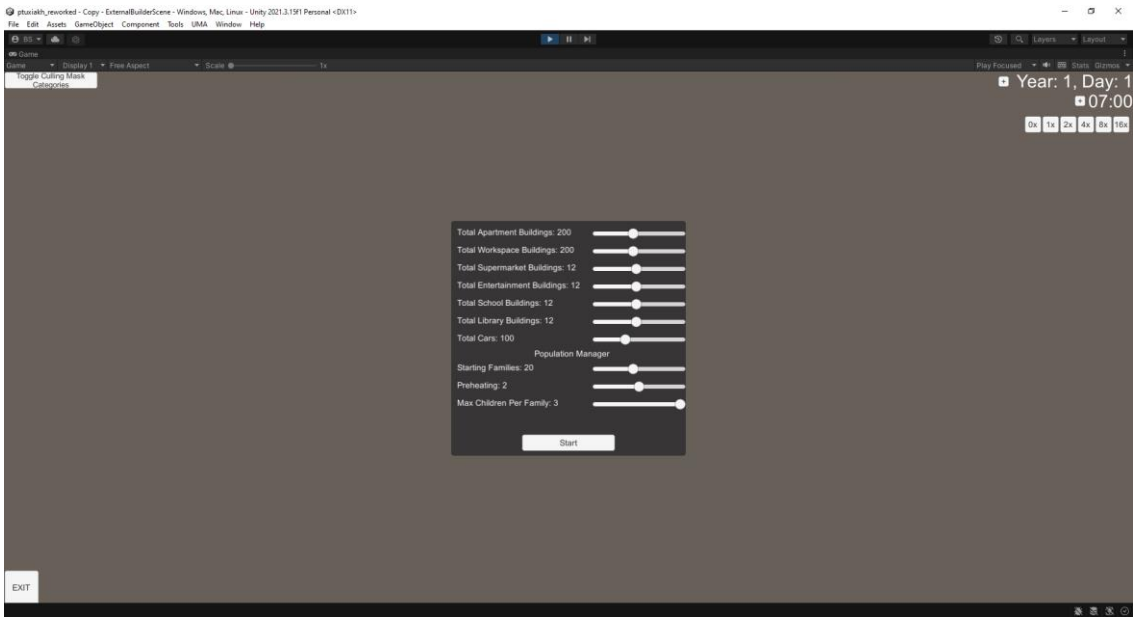
3.2. Διεπαφή χρήστη

Ένα βασικό στοιχείο της λειτουργίας της υλοποίησης είναι η επικοινωνία με τον χρήστη. Αυτή γίνεται μέσω της διεπαφής χρήστη, η οποία δέχεται δεδομένα τα οποία έχουν οριστεί ως παραμετροποιήσιμα και στη συνέχεια τα μεταφέρει στους κατάλληλους διαχειριστές για να γίνει η κατάλληλη ενημέρωση όπου χρειάζεται. Παρακάτω, θα αναλυθεί η διεπαφή χρήστη της υλοποίησης.

3.2.1. Αρχικοποίηση παραμέτρων

Κατά την έναρξη της προσομοίωσης, ο χρήστης συναντά ρυθμίσεις οι οποίες αφορούν διάφορες μεταβλητές οι οποίες χρησιμοποιούνται για την οριστικοποίηση κάποιων περιπτώσεων χρήσης. Αναφορικά, αυτές είναι:

- Total Apartment Buildings
- Total Workspace Buildings
- Total Supermarket Buildings
- Total Entertainment Buildings
- Total School Buildings
- Total Cars
- Starting Families
- Preheating
- Max Children Per Family

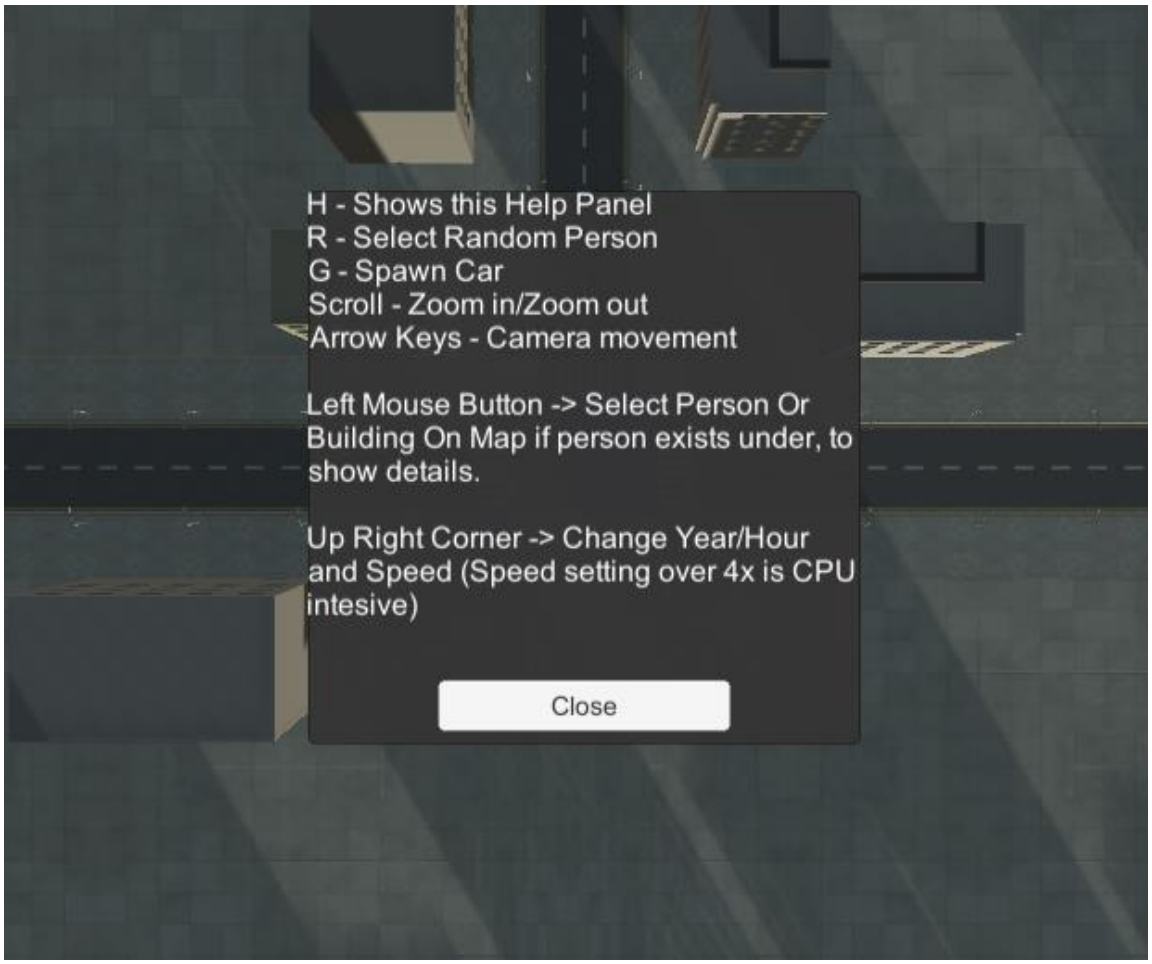


Εικόνα 22: Αρχική οθόνη εκτέλεσης προσομοίωσης

Οι τρείς τελευταίες μεταβλητές αφορούν την διαμόρφωση των οικογενειών, το μέγιστο πλήθος των παιδιών σε μία οικογένεια καθώς και τις ενδιάμεσα δημιουργημένες γενιές οι οποίες δεν αφορούν ενεργό πληθυσμό, όπως είδαμε και σε προηγούμενο κεφάλαιο. Οι υπόλοιπες μεταβλητές, αφορούν την αραιότητα ή πυκνότητα του χάρτη. Βοηθούν τους δημιουργημένους χαρακτήρες να βρίσκουν κτήρια για τις δραστηριότητες της ημέρας πιο κοντά στην κατοικία τους, αν είναι πιο πυκνός ο χάρτης, ενώ τους δυσκολεύει αν είναι πιο αραιός. Παράλληλα, λόγω του φόρτου των μοντέλων, ένας αραιός χάρτης με λιγότερο πληθυσμό είναι πιο εύκολα διαχειρίσιμος από τον Η/Υ που εκτελεί την προσομοίωση σε σχέση με μια εκτέλεση προσομοίωσης με τις ρυθμίσεις στο μέγιστο βαθμό.

3.2.2. Μενού βοήθειας

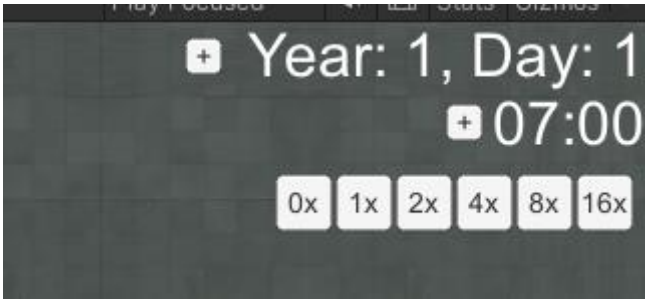
Με το που πατηθεί το κουμπί έναρξης, γίνεται η διαδικαστική δημιουργία της πόλης και ξεκινάει η προσομοίωση. Εκείνη τη χρονική στιγμή, εμφανίζεται και ένα παράθυρο το οποίο ενημερώνει τον χρήστη για τις δυνατές επιλογές που μπορεί να κάνει για να πλοηγηθεί και να μάθει τις λειτουργίες της προσομοίωσης.



Εικόνα 23: Μενού βοήθειας όπως εμφανίζεται στη προσομοίωση

3.2.3. Χρόνος, ώρα και ταχύτητα εκτέλεσης

Πάνω δεξιά στην οθόνη, είναι οι εξωτερικές λειτουργίες του *TimeManager*. Ο χρήστης μπορεί να δει την τρέχουσα χρονιά, μέρα, ώρα και λεπτό της προσομοίωσης, ενώ μπορεί να ελέγξει την ταχύτητα εκτέλεσης της.



Εικόνα 24: Ημερομηνία, ώρα και ταχύτητα εκτέλεσης προσομοίωσης

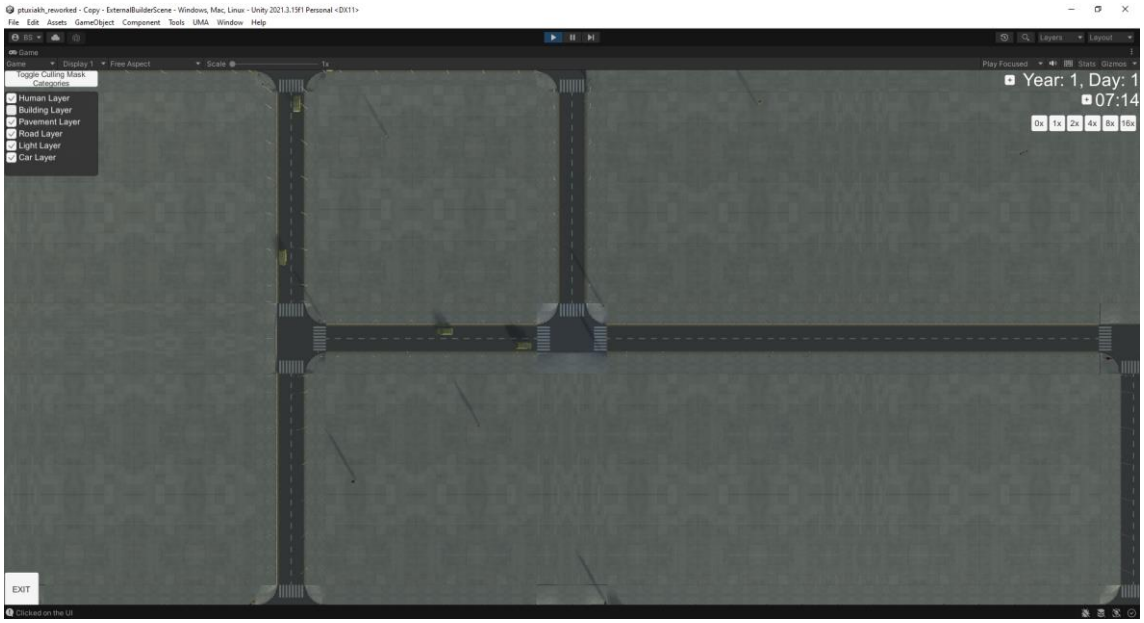
3.2.4. Culling Mask

Σε περίπτωση που ο χρήστης θέλει να δώσει το βάρος της προσοχής του μόνο σε ένα επίπεδο της προσομοίωσης, μπορεί να επιλέξει από πάνω αριστερά ποια *Layers* της κάμερας χρησιμοποιούνται. Αυτόματα, η κάμερα θα αποκρύψει τα υπόλοιπα.

Εικόνα 25: Culling Mask Options

Εικόνα 26: Προσομοίωση με όλα τα Layers ενεργά

Παραδείγματος χάριν, αν ένας χρήστης θέλει να δει την προσομοίωση χωρίς τα κτήρια, μπορεί να αποεπιλέξει το *Building Layer*, έτσι η προσομοίωση θα φαίνεται κάπως έτσι:

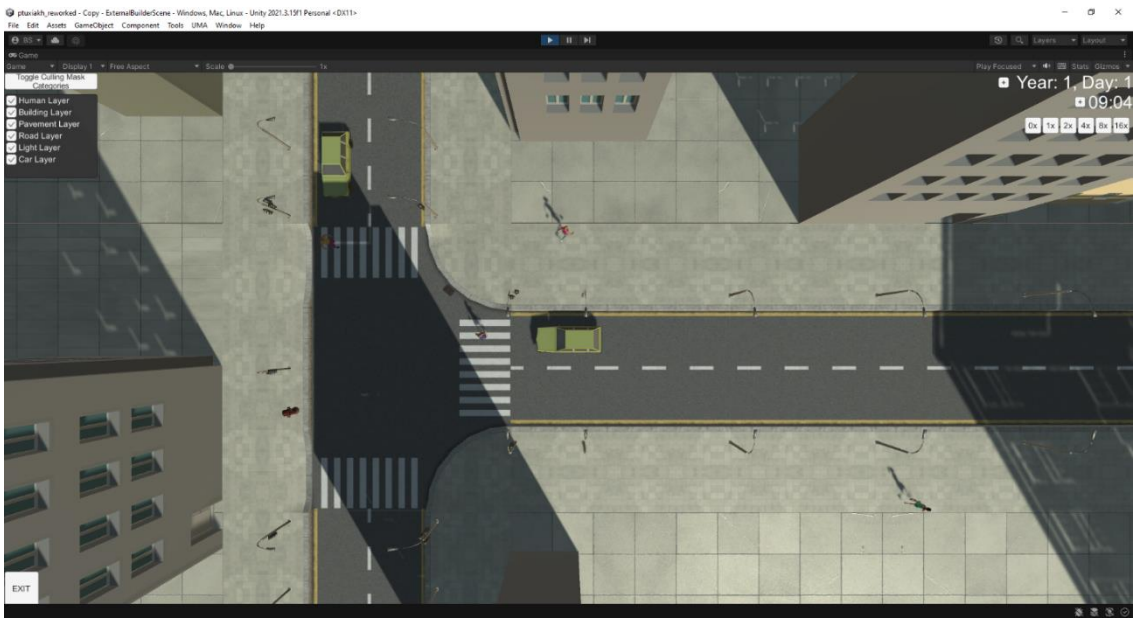


Εικόνα 27: Προσομοίωση με το Building Layer απενεργοποιημένο

Ως σημείωση, αναφέρεται ότι η απόκρυψη ενός επιπέδου δεν σημαίνει ότι τα αντικείμενα του επιπέδου σταματάνε να υπάρχουν, απλά δεν εμφανίζονται στην κάμερα. Ως αποτέλεσμα, υπάρχει περίπτωση ο χρήστης να παρατηρήσει ότι κάποιος χαρακτήρας εξαφανίστηκε, ενώ στη πραγματικότητα μπήκε σε κάποιο κτήριο.

3.2.5. Κίνηση κάμερας

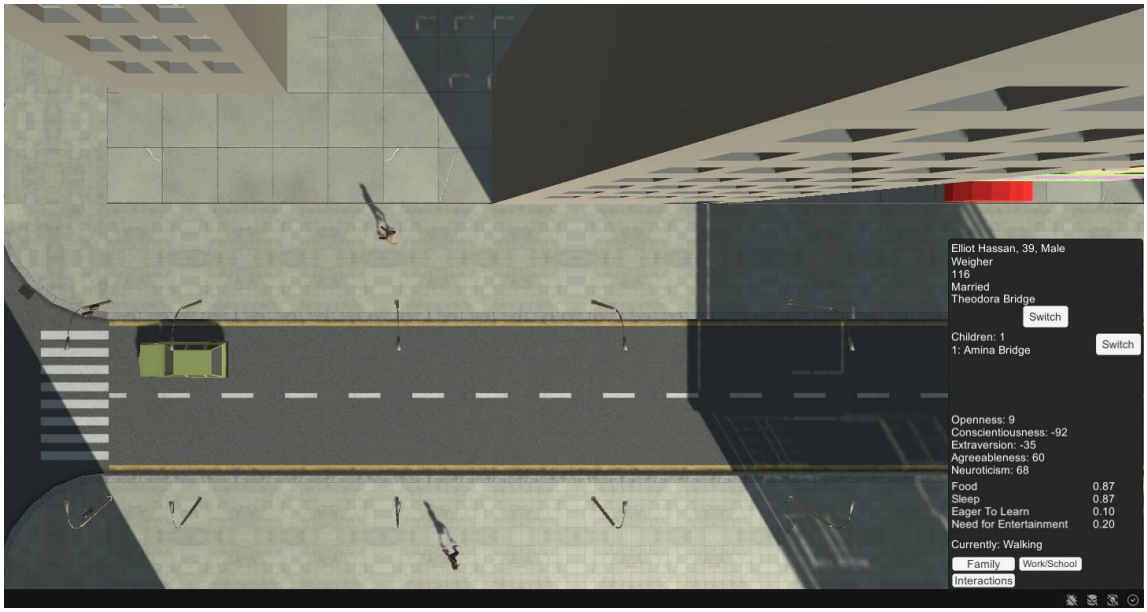
Σύμφωνα και με το μενού βοηθείας, οι δυνατές κινήσεις της κάμερας είναι στους τρείς άξονες. Για την μετακίνηση σε βάθος (άξονας Y) χρησιμοποιείται το *scroll wheel* του ποντικιού, ενώ για την μετακίνηση οριζόντια και κάθετα (άξονας X και Z) χρησιμοποιούνται τα βέλη του πληκτρολογίου. Επίσης, αν επιλεχθεί ένα αντικείμενο που περιέχει πληροφορίες (χαρακτήρες/κτήρια) η κάμερα κλειδώνει στο ανάλογο αντικείμενο και το ακολουθεί. Τέλος, σε περίπτωση χρήσης του πλήκτρου «Γ», δημιουργείται ένα νέο όχημα και η κάμερα ακολουθεί το όχημα.



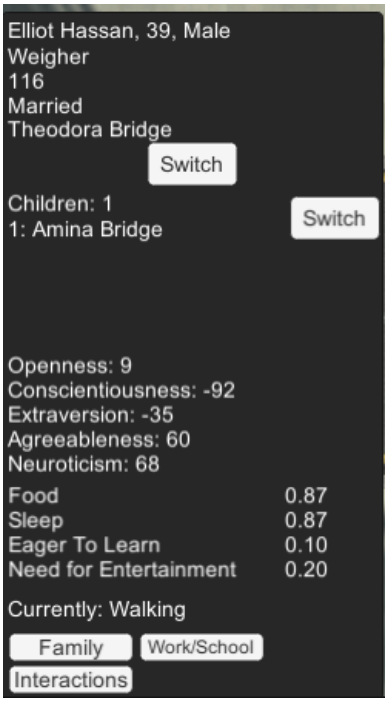
Εικόνα 28: κάμερα που ακολουθεί όχημα το οποίο αναμένει τον φωτεινό σηματοδότη. Μπροστά του, χαρακτήρας διασχίζει τη διάβαση.

3.2.6. Πληροφορίες ατόμου

Κάνοντας χρήση του πλήκτρου «P», ή επιλέγοντας έναν χαρακτήρα ο οποίος περπατάει στη πόλη, μπορούμε να μάθουμε πληροφορίες για αυτόν. Για παράδειγμα, ένας χαρακτήρας εμφανίζει αυτό το παράθυρο πληροφοριών:



Εικόνα 29: Παράθυρο πληροφοριών ατόμου σε επιλεγμένο χαρακτήρα



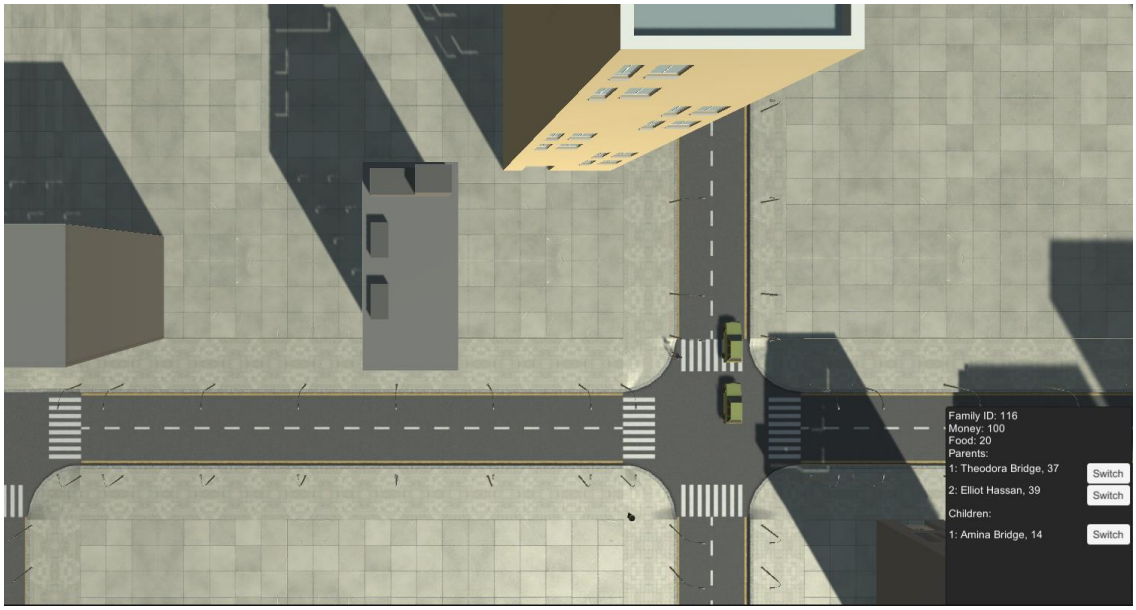
Εικόνα 30: Το παράθυρο πληροφοριών

Σε αυτό, μπορούμε να μάθουμε πληροφορίες όπως το όνομά του, την ηλικία του, το φύλο του, την εργασία του (Weighter), το ID της οικογένειας που ανήκει, το αν είναι παντρεμένος ή όχι με έναν άλλο χαρακτήρα της πόλης (μαζί με κουμπί εναλλαγής σε αυτό), το πλήθος και τα ονόματα των παιδιών της οικογένειας (1, Amina Bridge, μαζί με κουμπί εναλλαγής σε αυτό), τα χαρακτηριστικά OCEAN που διαθέτει, τις μεταβλητές αναγκών του (Τροφή, ύπνος, προθυμία μάθησης, ανάγκη για διασκέδαση), την κατάσταση που βρίσκεται καθώς και ένα κουμπί για εμφάνιση των πληροφοριών της οικογένειας του, της εργασίας του και των αλληλοεπιδράσεων με άλλους χαρακτήρες.

Εικόνα 31: Αλληλεπιδράσεις του χαρακτήρα Elliot Hassan (προηγ. φωτογραφία) με τον χαρακτήρα Chris Barnett

3.2.7. Πληροφορίες οικογένειας

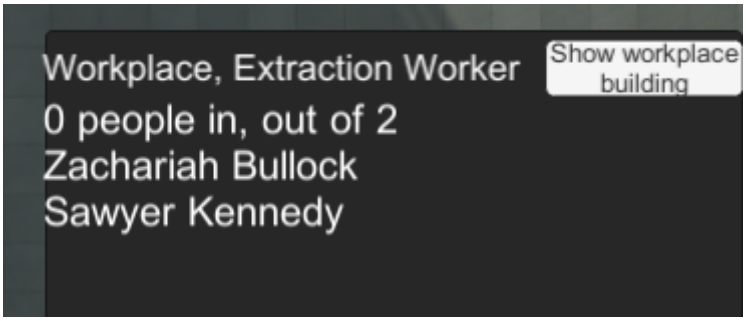
Πατώντας το πλήκτρο Family κατά την παρακολούθηση των πληροφοριών ενός χαρακτήρα, εμφανίζονται στοιχεία της οικογένειας που ανήκει ο χαρακτήρας, των χρημάτων και της τροφής που περιέχει το διαμέρισμα, τα μέλη της οικογένειας και γίνεται μετακίνηση της κάμερας στο κτήριο που έχει το διαμέρισμά της η επιλεγμένη οικογένεια. Λόγω της έναρξης της προσομοίωσης οι τιμές *Money* και *Food* έχουν αρχικοποιηθεί με τις τιμές 100 και 20, κάτι που θα αρχίσει να διαφοροποιείται όσο προχωράει η προσομοίωση.



Εικόνα 32:Το κτήριο στο οποίο διαμένει η οικογένεια Bridge/Hassan με FamilyID 116

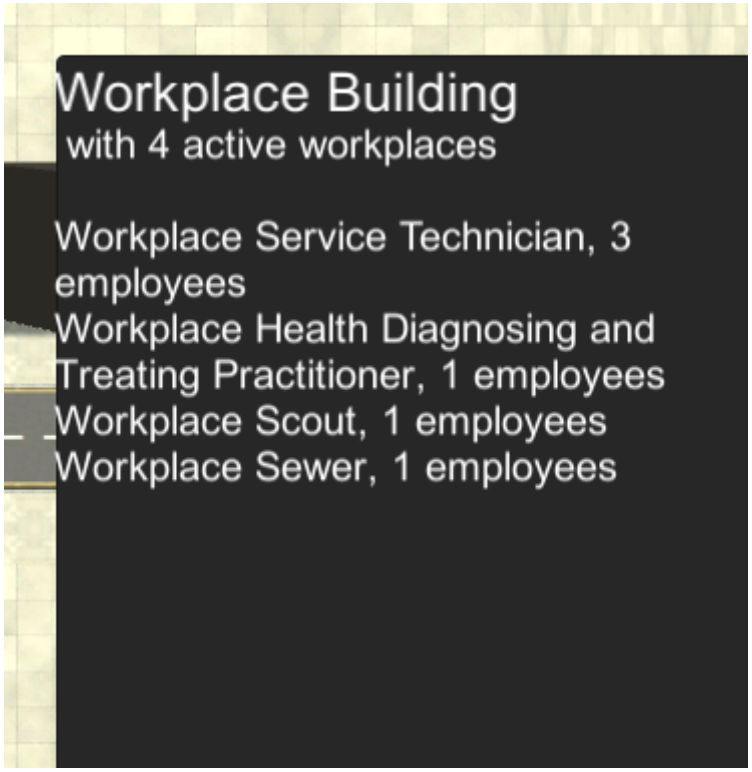
3.2.8. Πληροφορίες κτηρίων

Εκτός από την εμφάνιση των πληροφοριών της οικογένειας, μέσω των πληροφοριών του ατόμου ο χρήστης μπορεί να εμφανίσει πληροφορίες για την εργασία του χαρακτήρα.



Εικόνα 33: Εργασία ενός ατόμου. Στη συγκεκριμένη εργάζονται δύο άτομα.

Επίσης, ο χρήστης επιλέγοντας το κουμπί «Show workplace building» ή επιλέγοντας ένα κτήριο στη πόλη μπορεί να εμφανίσει πληροφορίες για αυτό. Επιλέγοντας ένα κτήριο τύπου εργασίας, εμφανίζονται οι παρακάτω πληροφορίες:



Εικόνα 34: πληροφορίες κτηρίου που στεγάζει εταιρείες/εργασίες

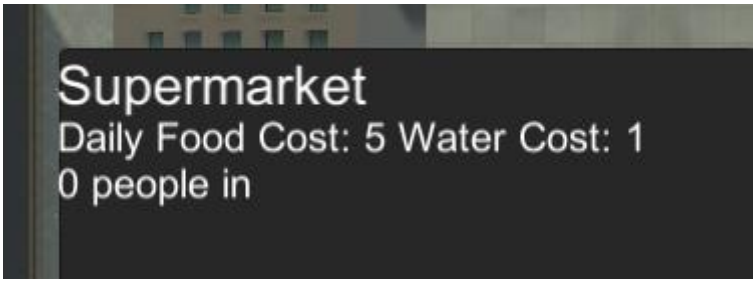
Στην παραπάνω εικόνα, διακρίνεται η λειτουργία τεσσάρων εργασιών, μιας που έχει τρεις υπαλλήλους και άλλες τρεις εργασίες που έχουν από έναν εργαζόμενο.

Άλλο κτήριο, το οποίο περιέχει διαμερίσματα εμφανίζει τις παρακάτω πληροφορίες:

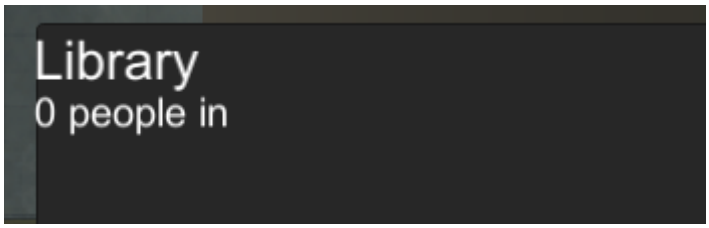
Εικόνα 35: πληροφορίες κτηρίου που στεγάζει διαμερίσματα

Οπότε, ο χρήστης μπορεί να λάβει την πληροφορία ότι στο συγκεκριμένο κτήριο στεγάζονται δύο τριμελείς οικογένειες.

Παρόμοια, για τα κτήρια που λειτουργούν ως καταστήματα, βιβλιοθήκες, σχολεία και κτήρια δραστηριοτήτων εμφανίζονται οι παρακάτω πληροφορίες:

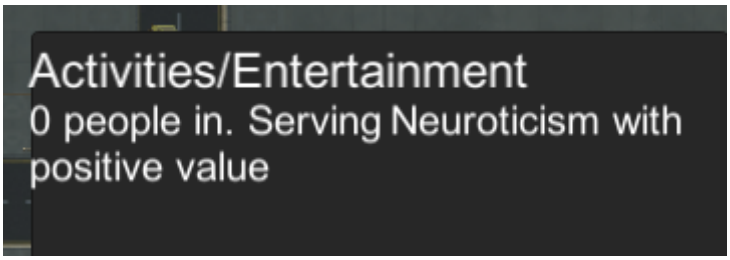


Εικόνα 36: πληροφορίες καταστήματος για αγορά τροφής



Εικόνα 37: πληροφορίες κτηρίου βιβλιοθήκης

Εικόνα 38: πληροφορίες σχολικού κτηρίου



Εικόνα 39: πληροφορίες κτηρίου δραστηριοτήτων. Το συγκεκριμένο κτήριο εξυπηρετεί μόνο χαρακτήρες με θετικό πρόσημο στο χαρακτηριστικό «Νευρωτισμός»

3.3. Συνολική εικόνα της προσομοίωσης

Έχοντας πλέον εξηγήσει την διεπαφή του χρήστη καθώς και τις λειτουργίες της προσομοίωσης, μπορούμε να δούμε το σύνολο της προσομοίωσης. Στο επόμενο κεφάλαιο, θα γίνει επεξήγηση της λειτουργικότητας της υλοποίησης, προβλήματα και λύσεις καθώς και τυχόν επεκτάσεις ή που μπορούν να υλοποιηθούν με βάση τη διπλωματική.

Εικόνα 40:Το σύνολο της υλοποίησης κατά την εκτέλεση.

4. ΣΥΜΠΕΡΑΣΜΑΤΑ

4.1. Συμπεράσματα διπλωματικής εργασίας

Κατά τη διάρκεια της υλοποίησης υπήρξε τριβή σε αρκετούς τομείς της πληροφορικής, των μαθηματικών της τεχνητής νοημοσύνης, ακόμη και της ψυχολογίας. Η διπλωματική σε αρκετά σημεία της ανέδειξε προβληματισμούς για τον τρόπο υλοποίησης της, απαιτώντας κριτική σκέψη για διαχωρισμό της εισαγόμενης πληροφορίας κατά την αναζήτηση πληροφοριών ώστε να υλοποιηθεί. Λόγω του μεγέθους των πεδίων που βασίζεται για την πραγματοποίηση του θέματος της, υπήρξε η ανάγκη μείωσης αρκετών τομέων που θα μπορούσαν να εισαχθούν για μία πληρέστερη εικόνα της προσομοίωσης και της λειτουργικότητας σε συνδυασμό με το μοντέλο OCEAN, χωρίς αυτό να σημαίνει ότι είναι ελλιπής, μιας και προσδιορίζει τις βασικές λειτουργίες της διαδικαστικής δημιουργίας μιας ολόκληρης πόλης, την δυναμική μετακίνηση πολλών χαρακτήρων χωρίς την ανάγκη εισόδου δεδομένων του χρήστη καθώς και αφηρημένη λειτουργία του χρονοδιαγράμματος τους χωρίς τον ισχυρό ορισμό του. Ως αποτέλεσμα, θεωρώ ότι η διπλωματική εργασία πέτυχε τον στόχο που αναφέρει ο τίτλος της, ενώ μπορεί να χρησιμοποιηθεί μελλοντικά ως βάση για επιμέρους υλοποιήσεις άλλων λειτουργιών.

4.2. Πιθανά προβλήματα και λύσεις

Όπως είναι λογικό, κατά την υλοποίηση της διπλωματικής εργασίας, προέκυψαν διάφορα προβλήματα τα οποία αφορούσαν τεχνολογικούς περιορισμούς καθώς και περιορισμούς στην κατανόηση.

Απαιτήσεις Υλικού

Λόγω της εμφάνισης πολλών τρισδιάστατων μοντέλων στην οθόνη, κρίνεται αναγκαία η χρήση ενός ισχυρού συστήματος για να υπερβεί προβλήματα χαμηλής ταχύτητας εκτέλεσης, ή δημιουργίας προβλημάτων στις μετακινήσεις χαρακτήρων. Ένα ακόμη βασικό πρόβλημα αφορά την εκτέλεση κώδικα από πολλά αντίγραφα κώδικα. Η εκτέλεση εντολών 200-1000 χαρακτήρων ανά frame μπορεί να οδηγήσει σε υπερβολικά μεγάλη αναμονή, λόγω της μη ικανότητας να γίνει αποφυγή εκτέλεσης. Ξεπερνώντας την έναρξη της διαδικαστικής δημιουργίας που μπορεί να χρειαστεί μέχρι και μερικά λεπτά ανάλογα την περίπτωση χρήσης, με στόχο τα 60 frames per second, η εμφάνιση πληροφοριών και ενημερώσεων σε ένα δευτερόλεπτο μπορεί εύκολα να προσπεράσει τις 85 χιλιάδες κλήσεις συστήματος της εφαρμογής ανά δευτερόλεπτο.

Απαιτήσεις Λογισμικού

Στο επίπεδο της υλοποίησης, η διαδικασία του αρχικού ορισμού των διαδικασιών που θα χρειαστεί η υλοποίηση αποτελεί ένα από τα βασικότερα βήματα πριν την έναρξη της υλοποίησης στη πραγματικότητα. Σε περίπτωση που δεν υπάρχει διακριτός στόχος, με κατάλληλο διαχωρισμό των scripts, υπάρχει μεγάλη πιθανότητα το τελικό αποτέλεσμα να είναι ένα σύνολο κώδικα χωρίς συνοχή, ενώ καταλήγει να είναι δυσκολονόητος και συνήθως επιρρεπής σε λάθη και δύσκολος στη διόρθωση του.

Ανάγκη χρήσης του AssetStore και προβληματικά πακέτα

Ένα από τα βασικά προβλήματα του AssetStore είναι ο μη έλεγχος των λύσεων που μεταφορτώνονται. Υπάρχουν αρκετές περιπτώσεις όπου ένα πακέτο φαίνεται ότι καλύπτει τις ανάγκες του χρήστη που αναζητά κάτι, αλλά τελικά ίσως να μη λειτουργεί ή να μην έχει ανανεωθεί για μεγάλο χρονικό διάστημα με αποτέλεσμα να αναλώνεται χρόνος αναζητώντας διορθώσεις ή λύσεις σε προβλήματα που δεν θα έπρεπε να υπάρχουν.

4.3. Πιθανές κατ' επέκταση χρήσεις της υλοποίησης

Με βάση την υλοποίηση, παρακάτω θα αναφερθούν συνοπτικά πιθανές επεκτάσεις που μπορούν να γίνουν για να προσθέσουν λειτουργικότητα ή να δημιουργήσουν δεδομένα για την επίλυση προβλημάτων ή την παραγωγή αποτελεσμάτων σε εκτελέσεις προσομοιώσεων για ένα ζήτημα.

Κατασκευή δρομολογίων οχημάτων και MMM

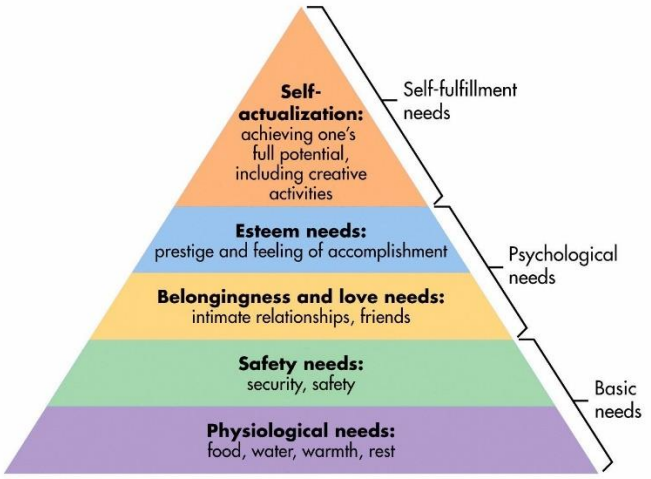
Μια από τις λειτουργίες που προστέθηκε άλλα δεν χρησιμοποιήθηκε πέρα από απλοποιημένη κίνηση, είναι αυτή των οχημάτων. Με εισαγωγή ακριβών πληροφοριών για την κίνηση των ανθρώπων από και προς σημείων ενός ακριβώς σχεδιασμένου χάρτη, μπορεί να εισαχθεί με τεχνητή νοημοσύνη η παραγωγή βέλτιστων διαδρομών για εξυπηρέτηση των ατόμων με MMM.

Ρεαλιστική υλοποίηση μοντέλων που βασίζονται στο μοντέλο OCEAN

Η υλοποίηση OCEAN που χρησιμοποιήθηκε στην διπλωματική εργασία είχε συνοπτικό χαρακτήρα λόγω της πολυπλοκότητας του μοντέλου για την δημιουργία του ως πλήρες σύστημα. Για τον λόγο αυτό, έγινε μεγάλη χρήση τυχαίων δεδομένων, σε αντίθεση με πραγματικά δεδομένα, όπως το συνδυασμό και των πέντε χαρακτηριστικών και την υλοποίηση εξελιγμένων μοντέλων.

Υλοποίηση της ιεραρχικής πυραμίδας ανθρώπινων αναγκών Maslow

Η πυραμίδα αναγκών Maslow αποτελεί έναν συμβατό τρόπο για τον έλεγχο των αναγκών των ανθρώπων μέσα σε μία κοινωνία. Η διπλωματική εργασία με κάποιες διαφοροποιήσεις στον τρόπο λειτουργίας του χρονοδιαγράμματος και των αναγκών του ατόμου μπορεί να εισάγει την χρήση της πυραμίδας Maslow και την εμφάνιση πληροφοριών για τους χαρακτήρες.



Εικόνα 41: πυραμίδα αναγκών Maslow

Ανάλυση αναγκών πόλης

Μια πόλη μακροσκοπικά μπορεί να θεωρηθεί ως ένας ζωντανός οργανισμός ο οποίος συνεχώς αναπτύσσεται και διαφοροποιείται. Η ανάλυση των αναγκών μιας πόλης είναι ένα ζήτημα το οποίο βρίσκεται συνεχώς ενεργό. Για τον λόγο αυτό, μια ανάλυση των θεμάτων και δημιουργία μιας προσομοίωσης σε βάθος χρόνου μπορεί να προσφέρει κατάλληλα δεδομένα για την πρόληψη ζητημάτων που μπορεί να υπάρξουν μελλοντικά.

Ανάλυση εργασίας και εισαγωγή πραγματικών δεδομένων και διασύνδεση με μοντέλα χαρακτήρων

Οργανισμοί όπως η Ευρωπαϊκή ένωση και η ΕΛΣΤΑΤ προσφέρουν πραγματικά δεδομένα όπως την ταξινόμηση εργασίας ISCO-08 για την ταξινόμηση του πληθυσμού σε κατηγορίες εργασίας, το επίπεδο μάθησης καθώς και το ποσοστό πληθυσμού και ανά φύλο που εργάζεται ανά τομέα εργασίας. Συνδυάζοντας τα δεδομένα αυτά με ένα μοντέλο χαρακτηριστικών, μπορεί να δημιουργηθεί μια προσομοίωση η οποία περιέχει πιο ακριβείς λειτουργίες κατά την εκτέλεση της.

Δημιουργία βιντεοπαιχνιδιού διαχείρισης πόλης (City Simulation / Survival Simulation)

Ο χώρος των βιντεοπαιχνιδιών παραμένει ενεργός και συνεχώς αναπτυσσόμενος. Μέσω της εξέλιξης της τεχνολογίας και της αύξησης του πληθυσμού που ασχολείται με τα παιχνίδια, έχουν πλέον προκύψει τίτλοι που αξιοποιούν την τεχνολογία της προσομοίωσης αναγκών στο έπακρο. Παράλληλα, παιχνίδια όπως το Oxygen not Included & Frostpunk έχουν δημιουργήσει ένα νέο είδος παιχνιδιών με την ονομασία «Survival Simulation» στο οποίο χρειάζεται να ελέγξεις τις ανάγκες του πληθυσμού ώστε να επιβιώσουν.

5. ASSETS

https://assetstore.unity.com/packages/3d/environments/urban/city-traffic-lights-pack-free-low-poly-3d-art-154053

https://assetstore.unity.com/packages/3d/environments/urban/city-voxel-pack-136141

https://assetstore.unity.com/packages/3d/environments/urban/modular-lowpoly-streets-free-192094

https://assetstore.unity.com/packages/3d/characters/uma-2-unity-multipurpose-avatar-35611

https://assetstore.unity.com/packages/3d/characters/humanoids/o3n-male-and-female-uma-races-102187

https://assetstore.unity.com/packages/3d/vehicles/land/simple-cars-pack-97669

https://assetstore.unity.com/packages/tools/ai/love-hate-33063

https://assetstore.unity.com/packages/3d/animations/basic-motions-157744

https://assetstore.unity.com/packages/3d/environments/urban/city-package-107224

https://answers.unity.com/questions/1787567/how-to-add-bezier-curves-on-a-randomly-moving-obje.html

Runtime NavMeshAgent, https://github.com/Unity-Technologies/NavMeshComponents

Mixamo animation Library, https://www.mixamo.com/

6. ΒΙΒΛΙΟΓΡΑΦΙΑ

Διαδικαστική δημιουργία

https://www.mit.edu/~jessicav/6.S198/Blog_Post/ProceduralGeneration.html

https://en.wikipedia.org/wiki/Procedural_generation

https://books.google.gr/books?hl=el&lr=&id=Rj4PEAAAQBAJ&oi=fnd&pg=PP1&dq=procedural+generation&ots=HDuT5J1j0I&sig=RKp3_0auHDS6_7QqjWu-iDWDMck&redir_esc=y#v=onepage&q=procedural%20generation&f=false

Προσομοίωση

https://xpro.mit.edu/courses/course-v1:xPRO+MLx1/

https://en.wikipedia.org/wiki/Simulation

https://education.mit.edu/project-type/coding-tools/

https://saucelabs.com/blog/simulators-vs-emulators-whats-the-difference-anyway

OCEAN Model & Variations

https://mettl.com/glossary/o/ocean-model/

https://en.wikipedia.org/wiki/Big_Five_personality_traits

https://blog.flexmr.net/ocean-personality-types

https://marcr.net/marcr-for-career-professionals/career-theory/career-theories-and-theorists/five-factor-model-ffm-or-ocean-model/

https://en.wikipedia.org/wiki/Revised_NEO_Personality_Inventory

https://en.wikipedia.org/wiki/16PF_Questionnaire#Relationship_to_five-factor_models

UNITY3D

https://unity.com/

https://en.wikipedia.org/wiki/Unity_(game_engine)

https://assetstore.unity.com/