ΠΑΝΕΠΙΣΤΗΜΙΟ
ΘΕΣΣΑΛΙΑΣ

ΣΧΟΛΗ ΘΕΤΙΚΩΝ ΕΠΙΣΤΗΜΩΝ

ΤΜΗΜΑ ΠΛΗΡΟΦΟΡΙΚΗΣ ΚΑΙ ΤΗΛΕΠΙΚΟΙΝΩΝΙΩΝ

Ανάπτυξη Στρατηγικού Παιχνιδιού με χρήση Α.Ι.

Γαβριηλίδης Αλέξανδρος

ΠΤΥΧΙΑΚΗ ΕΡΓΑΣΙΑ

ΥΠΕΥΘΥΝΟΣ

ΚΟΛΟΜΒΑΤΣΟΣ ΚΩΝΣΤΑΝΤΙΝΟΣ
Διδάσκων

Λαμία έτος 2026

ΠΑΝΕΠΙΣΤΗΜΙΟ
ΘΕΣΣΑΛΙΑΣ

ΣΧΟΛΗ ΘΕΤΙΚΩΝ ΕΠΙΣΤΗΜΩΝ

ΤΜΗΜΑ ΠΛΗΡΟΦΟΡΙΚΗΣ ΚΑΙ ΤΗΛΕΠΙΚΟΙΝΩΝΙΩΝ

Ανάπτυξη Στρατηγικού Παιχνιδιού με χρήση Α.Ι.

Γαβριηλίδης Αλέξανδρος

ΠΤΥΧΙΑΚΗ ΕΡΓΑΣΙΑ

ΥΠΕΥΘΥΝΟΣ

ΚΟΛΟΜΒΑΤΣΟΣ ΚΩΝΣΤΑΝΤΙΝΟΣ
Διδάσκων

Λαμία έτος 2026

UNIVERSITY OF
THESSALY

SCHOOL OF SCIENCE

DEPARTMENT OF COMPUTER SCIENCE & TELECOMMUNICATIONS

Strategy Game Development using A.I.

Gavriilidis Alexandros

FINAL THESIS

ADVISOR

KOLOMVATSOS KONSTANTINOS
Professor

Lamia year 2026

«Με ατομική μου ευθύνη και γνωρίζοντας τις κυρώσεις [(1)], που προβλέπονται από της διατάξεις της παρ. 6 του άρθρου 22 του Ν. 1599/1986, δηλώνω ότι:

*1. Δεν παραθέτω κομμάτια βιβλίων ή άρθρων ή εργασιών άλλων αυτολεξεί **χωρίς να τα περικλείω σε εισαγωγικά** και χωρίς να αναφέρω το συγγραφέα, τη χρονολογία, τη σελίδα. Η αυτολεξεί παράθεση χωρίς εισαγωγικά χωρίς αναφορά στην πηγή, είναι λογοκλοπή. Πέραν της αυτολεξεί παράθεσης, λογοκλοπή θεωρείται και η παράφραση εδαφίων από έργα άλλων, συμπεριλαμβανομένων και έργων συμφοιτητών μου, καθώς και η παράθεση στοιχείων που άλλοι συνέλεξαν ή επεξεργάσθηκαν, χωρίς αναφορά στην πηγή. Αναφέρω πάντοτε με πληρότητα την πηγή κάτω από τον πίνακα ή σχέδιο, όπως στα παραθέματα.*

*2. Δέχομαι ότι η αυτολεξεί **παράθεση χωρίς εισαγωγικά**, ακόμα κι αν συνοδεύεται από αναφορά στην πηγή σε κάποιο άλλο σημείο του κειμένου ή στο τέλος του, είναι αντιγραφή. Η αναφορά στην πηγή στο τέλος π.χ. μιας παραγράφου ή μιας σελίδας, δεν δικαιολογεί συρραφή εδαφίων έργου άλλου συγγραφέα, έστω και παραφρασμένων, και παρουσίασή τους ως δική μου εργασία.*

3. Δέχομαι ότι υπάρχει επίσης περιορισμός στο μέγεθος και στη συχνότητα των παραθεμάτων που μπορώ να εντάξω στην εργασία μου εντός εισαγωγικών. Κάθε μεγάλο παράθεμα (π.χ. σε πίνακα ή πλαίσιο, κλπ), προϋποθέτει ειδικές ρυθμίσεις, και όταν δημοσιεύεται προϋποθέτει την άδεια του συγγραφέα ή του εκδότη. Το ίδιο και οι πίνακες και τα σχέδια

4. Δέχομαι όλες τις συνέπειες σε περίπτωση λογοκλοπής ή αντιγραφής.

Ημερομηνία: ……/…../20……

ΓΑΒΡΙΗΛΙΔΗΣ ΑΛΕΞΑΝΔΡΟΣ

(1) «Όποιος εν γνώσει του δηλώνει ψευδή γεγονότα ή αρνείται ή αποκρύπτει τα αληθινά με έγγραφη υπεύθυνη δήλωση
του άρθρου 8 παρ. 4 Ν. 1599/1986 τιμωρείται με φυλάκιση τουλάχιστον τριών μηνών. Εάν ο υπαίτιος αυτών των πράξεων
σκόπευε να προσπορίσει στον εαυτόν του ή σε άλλον περιουσιακό όφελος βλάπτοντας τρίτον ή σκόπευε να βλάψει άλλον, τιμωρείται με κάθειρξη μέχρι 10 ετών.»

ΠΕΡΙΛΗΨΗ

Η παρούσα πτυχιακή εργασία πραγματεύεται τον σχεδιασμό, την ανάπτυξη και την αξιολόγηση ενός τρισδιάστατου (3D) στρατηγικού παιχνιδιού προσομοίωσης και διαχείρισης πόλεων (City-Builder), με ενσωμάτωση προηγμένων τεχνικών Μηχανικής Μάθησης. Κύριος στόχος της έρευνας είναι η διερεύνηση του παραδείγματος της Συνεργασίας Ανθρώπου-Πράκτορα (Human-Agent Teaming) στα σύγχρονα βιντεοπαιχνίδια. Η υλοποίηση πραγματοποιήθηκε στο περιβάλλον ανάπτυξης Unity 6, αξιοποιώντας τις αρχές του Αντικειμενοστραφούς Προγραμματισμού (OOP) σε γλώσσα C#. Το υποκείμενο σύστημα προσομοίωσης βασίζεται σε ένα δυναμικό πλέγμα (grid) και ενσωματώνει ένα πολύπλοκο μακροοικονομικό μοντέλο διαχείρισης πέντε βασικών πόρων (χρήματα, τροφή, νερό, πληθυσμός, ευημερία), το οποίο επηρεάζεται από στοχαστικά περιβαλλοντικά γεγονότα. Σε αντίθεση με τις παραδοσιακές προσεγγίσεις όπου η Τεχνητή Νοημοσύνη λειτουργεί ως ανταγωνιστής, στο παρόν έργο αναπτύχθηκε ένας ευφυής πράκτορας (AI Advisor) μέσω του πακέτου Unity ML-Agents. Ο πράκτορας εκπαιδεύτηκε με τον αλγόριθμο Βαθιάς Ενισχυτικής Μάθησης (Deep Reinforcement Learning) Proximal Policy Optimization (PPO), προκειμένου να κατανοεί τις χωρικές και οικονομικές εξαρτήσεις της πόλης. Μέσω προσεκτικού σχεδιασμού ανταμοιβών (Reward Shaping) και μηχανισμών αποτροπής εκφυλισμένων συμπεριφορών, το Νευρωνικό Δίκτυο έμαθε να εξάγει συμπεράσματα σε πραγματικό χρόνο και να υποδεικνύει οπτικά στον χρήστη τις βέλτιστες χωροταξικές επιλογές για την αποφυγή χρεοκοπίας και λιμού. Τα πειραματικά αποτελέσματα της εκπαίδευσης επιβεβαιώνουν τη μαθηματική σταθερότητα του μοντέλου και αποδεικνύουν πώς η Ενισχυτική Μάθηση μπορεί να αντικαταστήσει τους στατικούς αλγορίθμους, προσφέροντας δυναμικά, διαδραστικά και έξυπνα συστήματα υποδείξεων στον τομέα των παιχνιδιών.

ABSTRACT

This thesis deals with the design, development, and evaluation of a three-dimensional (3D) strategic city simulation and management game (City-Builder), incorporating advanced Machine Learning techniques. The main objective of the research is to investigate the Human-Agent Teaming paradigm in modern video games. The implementation was carried out in the Unity 6 development environment, utilizing the principles of Object-Oriented Programming (OOP) in C# language. The underlying simulation system is based on a dynamic grid and incorporates a complex macroeconomic model of management of five basic resources (money, food, water, population, prosperity), which is affected by stochastic environmental events. In contrast to traditional approaches where Artificial Intelligence acts as an adversary, in this project an intelligent agent (AI Advisor) was developed through the Unity ML-Agents package. The agent was trained with the Deep Reinforcement Learning algorithm Proximal Policy Optimization (PPO), to understand the spatial and economic dependencies of the city. Through careful reward shaping and mechanisms to prevent degenerate behaviors, the Neural Network learned to draw conclusions in real time and visually indicate to the user the optimal spatial planning options to avoid bankruptcy and famine. The experimental training results confirm the mathematical stability of the model and demonstrate how Reinforcement Learning can replace static algorithms, offering dynamic, interactive, and intelligent hint systems in the gaming sector.

Table of Contents

ΚΕΦΑΛΑΙΟ 1 Εισαγωγή

1.1 Η Εξέλιξη των Παιχνιδιών Προσομοίωσης Πόλης (City-Building Games)

Τα παιχνίδια κατασκευής πόλεων (City-Building Games - CBGs) είναι από τα πιο περίπλοκα και επιδραστικά είδη στρατηγικών βιντεοπαιχνιδιών προσομοίωσης. Το είδος αυτό ξεκίνησε στα τέλη της δεκαετίας του 1980 με μια διαφορετική φιλοσοφία σχεδιασμού: σε αντίθεση με τα παραδοσιακά παιχνίδια δράσης (όπου ο στόχος ήταν αυστηρά γραμμικός, για παράδειγμα η κατατρόποση ενός εχθρού), τα CBGs είχαν τον παίκτη στον ρόλο του δημιουργού και διαχειριστή. Με την εξέλιξη του υλικού και του λογισμικού στις σύγχρονες εποχές, αυτά τα παιχνίδια πλέον είναι σε θέση να αναπαράγουν ρεαλιστικά πολύπλοκα αστικά συστήματα και μακροοικονομικά μοντέλα, με τους παίκτες αναλαμβάνουν αποφάσεις μέσα σε δυναμικά συστήματα πλέγματος, τη συμφιλίωση της παραγωγής και κατανάλωσης πόρων, και τη φροντίδα της ευημερίας του ψηφιακού πληθυσμού.

Εικόνα 1 Το πρώτο city builder, SimCity, το οποίο κυκλοφόρησε το 1989

Η εμφάνιση της κατηγορίας χρονολογείται από το 1989 με το SimCity, που σχεδιάστηκε από τον Will Wright. Το καινοτόμο στοιχείο του SimCity ήταν η απουσία προκαθορισμένου όρου νίκης, αλλά ο παίκτης λειτουργούσε ως ένας ανοιχτός ελεγκτής συστήματος, ρυθμίζοντας μεταβλητές όπως ο φορολογικός συντελεστής και η διαχωριστική γραμμή περιοχών (Zoning: Residential, Commercial, Industrial).

Εικόνα 2 Το anno 117 Pax Romana, ένα City Builder που κυκλοφόρησε το 2025

Με την πάροδο των δεκαετιών και την ταχεία εξέλιξη της υπολογιστικής ισχύος, η αρχιτεκτονική των city-builders μεταφέρθηκε από τα στατικά μακροοικονομικά μοντέλα σε συστήματα που βασίζονται σε πράκτορες (Agent-based Modeling - ABM), Παιχνίδια ορόσημα, όπως το SimCity 4 (2003) και το Cities: Skylines (2015), ενέταξαν αλγορίθμους που προσομοιώνουν χιλιάδες αυτόνομους πολίτες, κάθε ένα με δική του ρουτίνα, ανάγκες στέγασης και αλγορίθμους εύρεσης διαδρομής (pathfinding algorithms, όπως A* ή Dijkstra) για την πλοήγησή τους στο οδικό δίκτυο.

Παρά τη ραγδαία γραφική και υπολογιστική εξέλιξη αυτών των τίτλων, ο ρόλος της Τεχνητής Νοημοσύνης παρέμεινε αυστηρά συστημικός και ανταγωνιστικός (ή παθητικός). Η ΤΝ στα παραδοσιακά City-Builders λειτουργεί στο παρασκήνιο, ελέγχοντας τους πολίτες (agents) μέσω ντετερμινιστικών αλγορίθμων πλοήγησης. Σε καμία περίπτωση δεν αναλαμβάνει τον ρόλο του "συνεργάτη" που μαθαίνει το παιχνίδι και συμβουλεύει τον παίκτη (Human-Agent Teaming). Τα συστήματα αυτά βασίζονται σε στατικούς, σκληρά κωδικοποιημένους κανόνες (heuristics), αδυνατώντας να προσαρμοστούν δυναμικά σε νέες στρατηγικές. Αυτό ακριβώς το ακαδημαϊκό κενό έρχεται να καλύψει η ενσωμάτωση αλγορίθμων Ενισχυτικής Μάθησης στο παρόν έργο.

1.1.1 Αλγόριθμοι Εύρεσης Βέλτιστης Διαδρομής (Pathfinding Algorithms)

Η προσομοίωση της κίνησης χιλιάδων πολιτών (agents) εντός του αστικού ιστού στα σύγχρονα City-Builders βασίζεται ιστορικά σε ντετερμινιστικούς αλγορίθμους θεωρίας γράφων (Graph Theory). Ο χάρτης του παιχνιδιού (είτε είναι πλέγμα είτε ελεύθερος χώρος) μεταφράζεται σε ένα σύνολο από κόμβους (nodes) και ακμές (edges). Για την επίλυση του προβλήματος της πλοήγησης στο οδικό δίκτυο, η βιομηχανία των βιντεοπαιχνιδιών έχει βασιστεί κυρίως σε δύο αλγορίθμους:

Ο Αλγόριθμος του Dijkstra (1956)

Ο αλγόριθμος του Dijkstra αποτελεί τον θεμέλιο λίθο της αναζήτησης σε γράφους. Αναπτύχθηκε για να βρίσκει το μονοπάτι ελάχιστου κόστους μεταξύ ενός κόμβου αφετηρίας και όλων των άλλων κόμβων σε έναν γράφο με μη αρνητικά βάρη στις ακμές.

Ο αλγόριθμος λειτουργεί διατηρώντας μια λίστα με τους μη-επισκεφθέντες κόμβους. Ξεκινώντας από την αφετηρία, επισκέπτεται πάντα τον γειτονικό κόμβο με το μικρότερο γνωστό συνολικό κόστος, εγγυώμενος μαθηματικά την εύρεση της βέλτιστης (συντομότερης) διαδρομής.

Ωστόσο, στο πλαίσιο ενός βιντεοπαιχνιδιού, ο αλγόριθμος αυτός παρουσιάζει ένα σοβαρό μειονέκτημα: πραγματοποιεί «τυφλή» (uninformed) αναζήτηση. Επειδή δεν γνωρίζει προς ποια κατεύθυνση βρίσκεται ο στόχος (π.χ. το σπίτι ενός πολίτη), εξερευνά κυκλικά προς όλες τις κατευθύνσεις εξίσου, σπαταλώντας τεράστιους υπολογιστικούς πόρους της μνήμης και του επεξεργαστή.

Ο Αλγόριθμος Α (A-Star, 1968)

Για την αντιμετώπιση του προβλήματος της εξαντλητικής αναζήτησης του Dijkstra, το Game AI στράφηκε στον αλγόριθμο A*. Ο A* αποτελεί έναν αλγόριθμο Ευριστικής Αναζήτησης (Heuristic Search). Το καινοτόμο στοιχείο του είναι ότι προσθέτει μια «κατεύθυνση» στην αναζήτηση.

Μαθηματικά, ο A* επιλέγει το επόμενο βήμα του ελαχιστοποιώντας τη συνάρτηση αξιολόγησης f(n) = g(n) + h(n) [2] Όπου:

- n: Ο τρέχων κόμβος του πλέγματος (π.χ. η διασταύρωση ενός δρόμου).
- g(n): Το ακριβές κόστος της διαδρομής από την αφετηρία μέχρι τον κόμβο n.
- h(n): Η **Ευριστική Συνάρτηση (Heuristic Function)**, η οποία αποτελεί μια μαθηματική "εκτίμηση" (πρόβλεψη) του κόστους από τον κόμβο n μέχρι τον τελικό στόχο. Σε παιχνίδια βασισμένα σε πλέγμα (όπως το δικό μας City-Builder), η ευριστική συνάρτηση που χρησιμοποιείται συχνότερα είναι η *Απόσταση Manhattan (Manhattan Distance)*, η οποία υπολογίζει το άθροισμα των απόλυτων διαφορών των συντεταγμένων X και Z.

Χάρη στο h(n), ο A* εστιάζει την αναζήτησή του απευθείας προς τον στόχο, αγνοώντας τα περιττά μονοπάτια, καθιστώντας τον την απόλυτη επιλογή (State of the Art) για την κίνηση των χαρακτήρων στα παιχνίδια.

Το Πρόβλημα της Κλιμάκωσης (Scalability) έναντι της Μηχανικής Μάθησης

Παρά τη μαθηματική τους κομψότητα, οι παραπάνω αλγόριθμοι παρουσιάζουν ανυπέρβλητα όρια στην αστική προσομοίωση μεγάλης κλίμακας. Η ταυτόχρονη εκτέλεση του αλγορίθμου A* για εκατοντάδες χιλιάδες αυτόνομους πολίτες σε κάθε καρέ (frame) δημιουργεί τεράστια συμφόρηση στον επεξεργαστή (CPU Bottleneck), κάτι που αποτελεί γνωστό πρόβλημα βελτιστοποίησης σε τίτλους όπως το *Cities: Skylines*.

Το κυριότερο πρόβλημα, ωστόσο, είναι η **περιορισμένη φύση** τους. Οι αλγόριθμοι A* και Dijkstra επιλύουν αποκλειστικά το τοπικό πρόβλημα πλοήγησης ("Πώς θα πάω από το σημείο Α στο σημείο Β"), αγνοώντας πλήρως τη μακροοικονομική διαχείριση (οικονομία, τροφή, ύδρευση). Είναι παθητικοί μηχανισμοί που δεν «μαθαίνουν» ούτε προσαρμόζονται στις αποφάσεις του παίκτη.

Για τη δημιουργία ενός συνολικού συστήματος υποδείξεων (AI Advisor), ικανού να αντιληφθεί **όλες** τις μεταβλητές μιας πόλης (χωρικές και οικονομικές) και να προβλέψει τον λιμό ή τη χρεοκοπία, τα συστήματα γραφών αποδεικνύονται ανεπαρκή. Γι' αυτόν ακριβώς

τον λόγο, η παρούσα εργασία εγκαταλείπει τους κλασικούς ντετερμινιστικούς αλγορίθμους και εισάγει την ευελιξία της Βαθιάς Ενισχυτικής Μάθησης (Deep Reinforcement Learning).

1.2 Δυναμική Αστικού Ιστού (Urban Dynamics) και Μαθηματική Μοντελοποίηση

Για να παραχθεί ένα πειστικό περιβάλλον, ο σχεδιαστής (Game Designer) οφείλει να μεταφράσει την πολυπλοκότητα της πραγματικής ζωής σε διακριτά μαθηματικά μοντέλα. Η προσέγγιση αυτή είναι γνωστή ως Αστική Δυναμική (Urban Dynamics).
Στα συστήματα που βασίζονται σε γύρους (Turn-Based Strategy), όπως η παρούσα εργασία, η εξέλιξη του συστήματος μοντελοποιείται συχνά ως μια διαδικασία μεταβολής διακριτού χρόνου (Discrete-Time Process). Οι πόροι (όπως τα χρήματα, η τροφή, το νερό και ο πληθυσμός) θεωρούνται μεταβλητές κατάστασης (state variables) x(t) τη χρονική στιγμή t. Η μετάβαση στον επόμενο γύρο t+1 υπολογίζεται από έναν πίνακα μετασχηματισμού που ενσωματώνει τις παραγωγικές υποδομές, καθώς και τις καταναλωτικές ανάγκες της πόλης.
Ένα κρίσιμο στοιχείο στη μοντελοποίηση είναι η έννοια του βρόχου ανατροφοδότησης (Feedback Loop). Διακρίνονται δύο βασικοί τύποι:

1. **Θετικοί Βρόχοι Ανατροφοδότησης (Positive Feedback Loops):** Μηχανισμοί που ενισχύουν μια κατάσταση. Για παράδειγμα, η αύξηση των κατοικιών προσελκύει περισσότερο πληθυσμό, ο οποίος με τη σειρά του αυξάνει τα έσοδα, επιτρέποντας την κατασκευή ακόμη περισσότερων κατοικιών. Οι θετικοί βρόχοι, αν αφεθούν ανεξέλεγκτοι, οδηγούν σε εκθετική ανάπτυξη και εν τέλει σε κατάρρευση του συστήματος λόγω εξάντλησης πόρων (Resource Exhaustion).
2. **Αρνητικοί Βρόχοι Ανατροφοδότησης (Negative Feedback Loops):** Μηχανισμοί σταθεροποίησης. Όταν ο πληθυσμός αυξάνεται ραγδαία, η κατανάλωση νερού και τροφής ξεπερνά τον ρυθμό παραγωγής. Η επακόλουθη έλλειψη μειώνει την ευημερία και προκαλεί μείωση του πληθυσμού, επαναφέροντας το σύστημα σε κατάσταση ισορροπίας.

Η εξισορρόπηση αυτών των βρόχων αποτελεί το δυσκολότερο κομμάτι του σχεδιασμού. Ένα σύστημα πρέπει να είναι αρκετά αυστηρό ώστε να τιμωρεί τα λάθη (π.χ. συνθήκη χρεοκοπίας), αλλά ταυτόχρονα να προσφέρει τα περιθώρια ανάκαμψης για να διατηρείται η προσοχή του παίκτη.

1.3 Κυτταρικά Αυτόματα (Cellular Automata) και Προσομοίωση σε Πλέγμα (Grid)

Για την οπτική και χωρική αναπαράσταση της πολυπλοκότητας της Αστικής Δυναμικής, τα στρατηγικά παιχνίδια χρησιμοποιούν συνήθως αρχιτεκτονικές πλέγματος (Grid-based architecture). Η επιστημονική θεωρία πίσω από αυτή την προσέγγιση είναι τα Κυτταρικά Αυτόματα, τα οποία έγιναν ευρέως γνωστά μέσω του μαθηματικού μοντέλου *Game of Life* του John Conway (1970). Ένα Κυτταρικό Αυτόματο αποτελείται από ένα πλέγμα κελιών (cells), καθένα από τα οποία βρίσκεται σε μια συγκεκριμένη κατάσταση. Η κατάσταση κάθε κελιού στον επόμενο γύρο καθορίζεται από την τρέχουσα κατάστασή του και τις καταστάσεις των γειτονικών του κελιών (συνήθως μέσω της γειτονίας Moore, η οποία ελέγχει τα 8 περιμετρικά κελιά)[1]. Στο υπό ανάπτυξη λογισμικό, το πλέγμα διαστάσεων 10x10 υπακούει σε παρόμοιους κανόνες τοπικότητας: η αποτελεσματικότητα ή η αναβάθμιση ενός κτιρίου

(όπως η παροχή νερού από ένα WaterPlant) εξαρτάται άμεσα από τη χωρική του τοποθέτηση σε σχέση με τις οικιστικές δομές.

1.4 Σκοπός και Αντικείμενο της Πτυχιακής Εργασίας

Παραδοσιακά, τα παιχνίδια στρατηγικής, ιδιαίτερα τα παιχνίδια διαχείρισης πόλεων, απευθύνονταν κυρίως στο κοινό που ενδιαφερόταν για τέτοιου είδους παιχνίδια. Ωστόσο, τα τελευταία χρόνια έχουν κερδίσει δημοτικότητα ως δοκιμαστικά περιβάλλοντα (testbeds) για αλγορίθμους Τεχνητής Νοημοσύνης. Αυτά τα περιβάλλοντα, όπου οι πόροι είναι πολλοί και αλληλεξαρτώμενοι, δοκιμάζουν τα όρια της Ενισχυτικής Μάθησης (Reinforcement Learning), καθώς ο αλγόριθμος πρέπει να θυσιάζει το βραχυπρόθεσμο κέρδος για μεγαλύτερη μακροπρόθεσμη σταθερότητα.
Ο σκοπός αυτής της πτυχιακής εργασίας είναι ο σχεδιασμός και η ανάπτυξη ενός τρισδιάστατου (3D) παιχνιδιού στρατηγικής βασισμένου σε πλέγμα (Grid-Based City Builder) στην πλατφόρμα Unity. Εκτός από την υλοποίηση ενός ολοκληρωμένου οικονομικού συστήματος που περιλαμβάνει κανόνες παραγωγής, κατανάλωσης και τυχαία γεγονότα, η εργασία εστιάζει επίσης στην ενσωμάτωση Μηχανικής Μάθησης μέσω του πακέτου Unity ML-Agents.
Το λογισμικό δεν αντιμετωπίζει την Τεχνητή Νοημοσύνη ως αντίπαλο, αλλά ενός βοηθού/συμβούλου που ανα πάσα στιγμή ο παίκτης μπορεί να συβλουλευτεί για την επόμενή του κίνηση. Με την εκπαίδευση ενός Νευρωνικού Δικτύου για αυτόνομη διαχείριση πόρων και χώρου, το σύστημα εξελίσσεται σε έναν «Έξυπνο Σύμβουλο». Ο πράκτορας λαμβάνει παρατηρήσεις από την κατάσταση της πόλης σε πραγματικό χρόνο, προβλέπει πιθανές μελλοντικές ελλείψεις και μέσω της διεπαφής χρήστη εμφανίζει την ιδανική χωροταξική και οικονομική ενέργεια στον παίκτη.

ΚΕΦΑΛΑΙΟ 2 ΜΗΧΑΝΕΣ ΑΝΑΠΤΥΞΗΣ ΠΑΙΧΝΙΔΙΩΝ & ΕΠΙΛΟΓΗ ΛΟΓΙΣΜΙΚΟΥ

2.1 Εισαγωγή στις Μηχανές Παιχνιδιών (Game Engines)

Η ανάπτυξη ενός σύγχρονου τρισδιάστατου βιντεοπαιχνιδιού αποτελεί μια υπολογιστικά και σχεδιαστικά πολύπλοκη διαδικασία. Στις απαρχές της βιομηχανίας, οι προγραμματιστές αναγκάζονταν να γράφουν κώδικα απευθείας στο υλικό (hardware) ή να χρησιμοποιούν χαμηλού επιπέδου βιβλιοθήκες γραφικών (όπως η OpenGL και το DirectX) για τον υπολογισμό κάθε πολυγώνου στην οθόνη. Σήμερα, το πρόβλημα αυτό έχει επιλυθεί μέσω των Μηχανών Παιχνιδιών (Game Engines).

Μια Μηχανή Παιχνιδιού είναι ένα ολοκληρωμένο περιβάλλον ανάπτυξης λογισμικού (IDE) που παρέχει έτοιμα υποσυστήματα (frameworks) για την απόδοση γραφικών (rendering), την προσομοίωση φυσικής (physics engines, π.χ. PhysX), τη διαχείριση ήχου, τον έλεγχο εισόδων (input handling) και τη δημιουργία διεπαφών (UI). Η αφαίρεση (abstraction) αυτών των χαμηλού επιπέδου λειτουργιών επιτρέπει στους δημιουργούς να εστιάσουν αποκλειστικά στη λογική (gameplay logic) και τον σχεδιασμό (game design).

Για την υλοποίηση του 3D City-Builder με ενσωμάτωση Τεχνητής Νοημοσύνης, εξετάστηκαν οι τρεις δημοφιλέστερες μηχανές της σύγχρονης βιομηχανίας: η Unreal Engine, η Godot και η Unity.

2.1.1 Βασικά Υποσυστήματα μιας Σύγχρονης Μηχανής Παιχνιδιών

Παρόλο που οι μηχανές παιχνιδιών διαφέρουν στη φιλοσοφία σχεδιασμού τους, η εσωτερική τους δομή (Engine Architecture) ακολουθεί ένα κοινό πρότυπο κατανομής ευθυνών. Μια σύγχρονη μηχανή δεν αποτελεί ένα μονολιθικό πρόγραμμα, αλλά ένα σύνολο ενσωματωμένων και διακριτών υποσυστημάτων που λειτουργούν ταυτόχρονα κατά τον Κεντρικό Βρόχο (Game Loop). Τα πιο σημαντικά από αυτά είναι:

- **Μηχανή Απόδοσης Γραφικών:** Είναι το πιο απαιτητικό μέρος σε υπολογιστική ισχύ. Μετατρέπει τα τρισδιάστατα μαθηματικά μοντέλα και τα υλικά σε δισδιάστατους εικονοστοιχεία (pixels) στην οθόνη. Για αυτό, επικοινωνεί άμεσα με την Κάρτα Γραφικών (GPU) μέσω APIs όπως τα DirectX, Vulkan και OpenGL. Στο συγκεκριμένο λογισμικό (City-Builder), η μηχανή γραφικών χειρίζεται τον φωτισμό, τις σκιές και τη γεωμετρία των κτιρίων, βασιζόμενη στις ισομετρικές προβολές της κάμερας.
- **Η Μηχανή Προσομοίωσης Φυσικής:** Αποτελεί ένα ειδικό υποσύστημα, όπως το PhysX της NVIDIA που ενσωματώνεται στην Unity. Εκτελεί πολύπλοκους υπολογισμούς Νευτώνειας φυσικής, όπως τη βαρύτητα, τις τριβές και, κυρίως, την ανίχνευση συγκρούσεων (Collision Detection). Ακόμα και σε στρατηγικά παιχνίδια βασισμένα σε πλέγμα, η φυσική μηχανή είναι κρίσιμη για τη λειτουργία του Raycasting, επιτρέποντας στον κέρσορα να εντοπίσει με ακρίβεια τα όρια (colliders) του εδάφους.

- **Το Σύστημα Διαχείρισης Ήχου:** είναι υπεύθυνο για την αναπαραγωγή και τη χωρική μίξη των ηχητικών εφέ. Δυνατότητά του είναι η δημιουργία ψευδαίσθησης 3D ήχου, ρυθμίζοντας την ένταση και τον τονισμό (pitch) ανάλογα με την απόσταση του παίκτη από την πηγή, όπως π.χ., ενός εργοστασίου.
- **Σύστημα Εκτέλεσης Σεναρίων:** Αυτό είναι το «εγκεφαλικό» σημείο της μηχανής, όπου εκτελείται ο κώδικας λογικής (Gameplay Logic) που δημιουργεί ο προγραμματιστής. Στη Unity, το σύστημα αυτό μεταφράζει τον κώδικα C# σε C++ μέσω της τεχνολογίας IL2CPP, διασφαλίζοντας την υψηλότερη ταχύτητα εκτέλεσης σε επίπεδο υλικού.

2.1.2 Αρχιτεκτονικά Πρότυπα: Component-Based Architecture έναντι ECS

Η διαχείριση των οντοτήτων σε τρισδιάστατο κόσμο αποτελεί ένα κλασικό πρόβλημα στη Μηχανική Λογισμικού. Οι παραδοσιακές εφαρμογές βασίζονται αυστηρά στον Αντικειμενοστραφή Προγραμματισμό (OOP), χρησιμοποιώντας βαθιά δέντρα κληρονομικότητας. Όμως, στα βιντεοπαιχνίδια, όπου χιλιάδες αντικείμενα απαιτούν ενημέρωση 60 φορές το δευτερόλεπτο, η βαθιά κληρονομικότητα προκαλεί δυσκαμψία και μνήμη (Cache Misses) προβλήματα.

Για το λόγο αυτό, οι σύγχρονες μηχανές, συμπεριλαμβανομένης της Unity, υιοθετούν το πρότυπο της Αρχιτεκτονικής Βασισμένης σε Συστατικά (Component-Based Architecture). Σύμφωνα με αυτό, κάθε αντικείμενο στον κόσμο (GameObject) είναι απλώς ένα άδειο δοχείο. Η συμπεριφορά του δεν ορίζεται από το «τι είναι", αλλά από τα συστατικά (Components) που προσθέτουμε. Για παράδειγμα, στο 3D City-Builder, ένα κελί (Tile) αποκτά τη μορφή του μέσω προσθήκης ενός MeshFilter (για τη γεωμετρία), ενός BoxCollider (για τη φυσική) και του προσαρμοσμένου script TileCell.cs (για τη λογική της δομής)[4].

Στη σύγχρονη έρευνα των μηχανών παιχνιδιών, έχει αρχίσει να επικρατεί μια νέα προσέγγιση γνωστή ως Entity-Component-System (ECS) ή Σχεδιασμός Προσανατολισμένος στα Δεδομένα. Στο μοντέλο ECS, τα δεδομένα διαχωρίζονται πλήρως από τη λογική και αποθηκεύονται συνεχόμενα στη μνήμη RAM. Αυτό επιτρέπει στον επεξεργαστή να αξιοποιεί καλύτερα την κρυφή μνήμη (L1/L2 Cache), καθιστώντας δυνατή την ταυτόχρονη προσομοίωση εκατοντάδων χιλιάδων πρακτόρων[3]. Αν και το έργο αυτής της πτυχιακής υλοποιήθηκε στο παραδοσιακό Component-Based μοντέλο, λόγω της στενής συνεργασίας του με το ML-Agents, η θεωρία του ECS θεωρείται το μέλλον στην αστική προσομοίωση μεγάλης κλίμακας.

2.2 Ανασκόπηση και Σύγκριση Δημοφιλών Μηχανών

2.2.1 Unreal Engine (Epic Games)

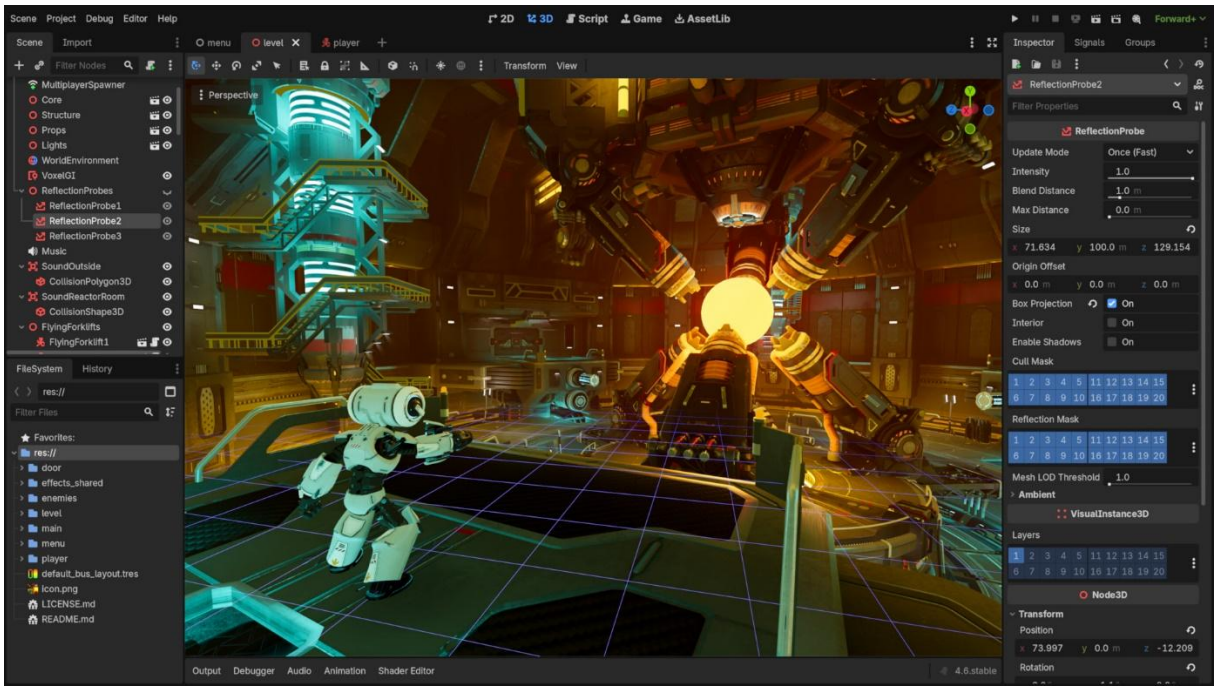
Η Unreal Engine αποτελεί το βιομηχανικό πρότυπο (industry standard) για παραγωγές υψηλού προϋπολογισμού (ΑΑΑ τίτλους). Ξεχωρίζει για τις υπερσύγχρονες τεχνολογίες φωτορεαλιστικής απόδοσης γραφικών, όπως το σύστημα δυναμικού φωτισμού (Lumen) και το σύστημα γεωμετρίας (Nanite). Η ανάπτυξη λογισμικού στην Unreal πραγματοποιείται είτε μέσω της γλώσσας C++, η οποία προσφέρει απόλυτο έλεγχο στη μνήμη αλλά είναι δυσκολη στον χειρισμό για κάποιον αρχάριο ώστε να πρώτη του εμπειρία με μηχανή παιχνιδιού, είτε μέσω του οπτικού συστήματος προγραμματισμού "Blueprints" (Visual Scripting). Αν και πανίσχυρη, η Unreal θεωρήθηκε υπερβολικά «βαριά» (resource-heavy) για ένα grid-based παιχνίδι διαχείρισης. Επιπλέον, η ενσωμάτωση εξωτερικών βιβλιοθηκών Python για Μηχανική Μάθηση απαιτεί πολύπλοκη διαχείριση C++ plugins, γεγονός που θα αποσπούσε τον χρόνο ανάπτυξης από τον πυρήνα της εργασίας.

Εικόνα 3: Το λογότυπο της μηχανής unreal

Εικόνα 4 Το περιβάλλον της μηχανής unreal

2.2.2 Godot Engine

Εικόνα 5: Το λογότυπο της μηχανής GODOT



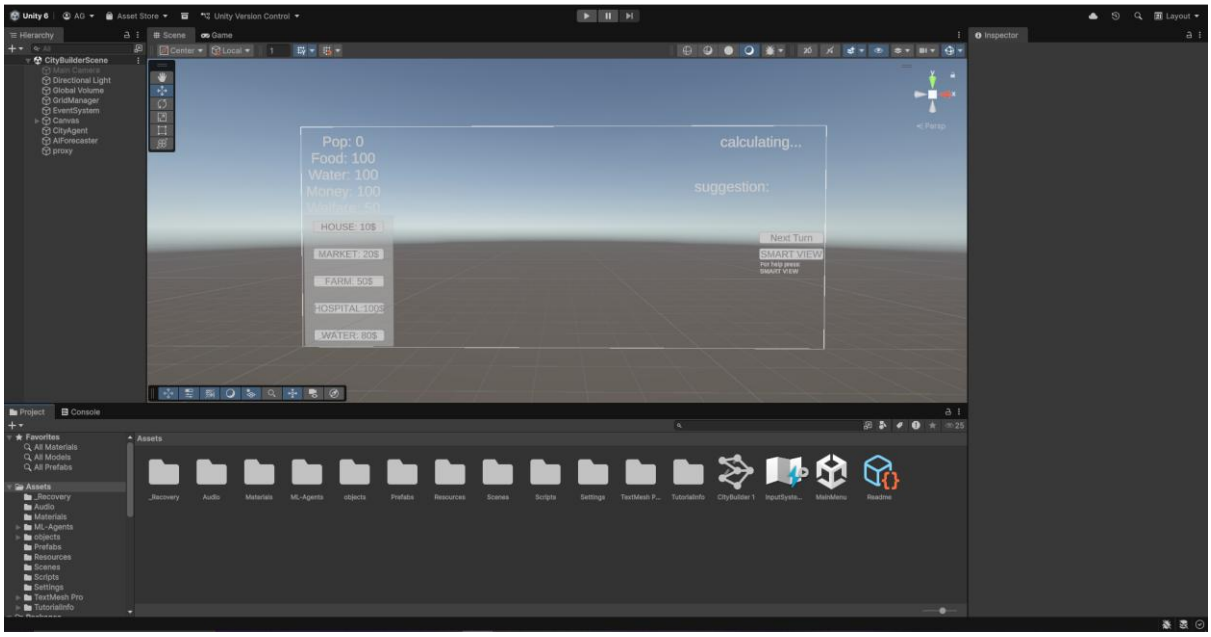
Εικόνα 6: Το περιβάλλον της μηχανής GODOT

Η Godot είναι μια ραγδαία αναπτυσσόμενη, ανοικτού κώδικα (open-source) μηχανή. Η αρχιτεκτονική της είναι μοναδική, καθώς βασίζεται σε ένα ιεραρχικό σύστημα Κόμβων (Node-based architecture), όπου τα πάντα (σκηνές, χαρακτήρες, UI) είναι κόμβοι που συνδέονται σε ένα δέντρο (Scene Tree). Χρησιμοποιεί κατά κύριο λόγο την GDScript, μια scripting γλώσσα με συντακτικό παρόμοιο της Python. Είναι εξαιρετικά ελαφριά και ιδανική για δισδιάστατα (2D) και απλά τρισδιάστατα (3D) παιχνίδια. *Μειονέκτημα για το έργο:* Ενώ η κοινότητα της Godot έχει ξεκινήσει πειραματικές προσπάθειες ενσωμάτωσης Μηχανικής Μάθησης (μέσω εργαλείων όπως το Godot RL Agents), το οικοσύστημα δεν είναι ακόμα

αρκετά ώριμο. Η απουσία επίσημης και πλήρως τεκμηριωμένης υποστήριξης για διασύνδεση με το PyTorch σε κλίμακα παραγωγής, κατέστησε τη μηχανή ακατάλληλη για τη συγκεκριμένη πτυχιακή.

2.2.3 Unity Engine (Unity Technologies)

Εικόνα 7: Το λογότυπο της μηχανής Unity



Εικόνα 8: Το περιβάλλον της μηχανής Unity

Η Unity είναι κορυφαία στον τομέα της ανάπτυξης εφαρμογών για πολλαπλές πλατφόρμες, προσφέροντας μια εξαιρετική ισορροπία μεταξύ ευκολίας χρήσης, απόδοσης και προηγμένων λειτουργιών. Με τη μεγάλη κοινότητα υποστήριξης και το Asset Store, διευκολύνει την ταχεία δημιουργία πρωτοτύπων. Η χρήση της C# ως κύριας γλώσσας προγραμματισμού διασφαλίζει την κατασκευή καθαρού, στατικά τυποποιημένου και εύκολα επεκτάσιμου κώδικα, που είναι κρίσιμο για τη διαχείριση δικτύων και μεγάλων δεδομένων.

2.3 Κριτήρια Επιλογής: Γιατί επιλέχθηκε η Unity 6

Η τελική επιλογή της Unity 6 ως του περιβάλλοντος ανάπτυξης για το συγκεκριμένο 3D City-Builder βασίστηκε σε τρία θεμελιώδη κριτήρια:

1. **Αντικειμενοστρεφής Αρχιτεκτονική και C#:** Η C# διευκόλυνε την αποτελεσματική μοντελοποίηση πολύπλοκων συστημάτων μέσω αντικειμενοστραφούς προγραμματισμού . Η δημιουργία ειδικών κλάσεων για το διαχείριση του πλέγματος (GridManager.cs) και των κελιών (TileCell.cs) βοήθησε στην απομόνωση των λειτουργιών, ακολουθώντας την αρχή του Ενιαίου Καθήκοντος.

2. **Επίσημο Οικοσύστημα ML-Agents:** Το πιο σημαντικό κριτήριο επιλογής. Η Unity Technologies αναπτύσσει επίσημα το πακέτο Unity Machine Learning Agents (ML-Agents). Είναι ένα ανοιχτού κώδικα πλαίσιο που μετατρέπει τη μηχανή σε περιβάλλον εκπαίδευσης για ευφυείς πράκτορες, συνδέοντας τη C# με την Python.

3. **Υψηλή επεκτασιμότητα στο UI:** Η μηχανή ενσωματώνει το TextMeshPro και το Unity UI, που είναι ουσιώδη για την υλοποίηση του σύνθετου συστήματος ανατροφοδότησης (Floating Texts, Advisor Panels) σε ένα παιχνίδι διαχείρισης πόρων.

2.4 Ο Ρόλος του Πρωτοκόλλου Επικοινωνίας gRPC στο ML-Agents

Για να κατανοήσουμε το πλεονέκτημα της επιλογής της Unity, πρέπει να λυθεί το πρόβλημα της επικοινωνίας. Ιστορικά, για να συνδεθεί μια μηχανή παιχνιδιών με ένα Νευρωνικό Δίκτυο, ο προγραμματιστής έπρεπε να δημιουργήσει χειροκίνητα κανάλια επικοινωνίας μέσω TCP/IP sockets (Client-Server), μεταφέροντας δεδομένα σε μορφή JSON. Αυτή η μέθοδος επέφερε μεγάλες καθυστερήσεις (latency) στην εκπαίδευση.

Η Unity, με το πακέτο ML-Agents, αξιοποιεί το πρωτόκολλο gRPC (gRPC Remote Procedure Calls), που ανέπτυξε η Google. Το gRPC επιτρέπει γρήγορη, αμφίδρομη και χαμηλού επιπέδου επικοινωνία (μέσω HTTP/2) μεταξύ του περιβάλλοντος προσομοίωσης της Unity, το οποίο στέλνει παρατηρήσεις και λαμβάνει ενέργειες, και του Python API, που τρέχει τον αλγόριθμο PPO μέσω PyTorch. Αυτή η αποδοτική και βελτιστοποιημένη «γέφυρα» επέτρεψε, στο πλαίσιο αυτής της εργασίας, να εστιάσουμε στην αρχιτεκτονική των ανταμοιβών (Reward Shaping) και όχι στη δημιουργία δικτυακών υποδομών.

ΚΕΦΑΛΑΙΟ 3: ΘΕΩΡΗΤΙΚΟ ΥΠΟΒΑΘΡΟ ΤΕΧΝΗΤΗΣ ΝΟΗΜΟΣΥΝΗΣ ΚΑΙ ΕΝΙΣΧΥΤΙΚΗΣ ΜΑΘΗΣΗΣ

3.1 Παραδοσιακές Αρχιτεκτονικές Τεχνητής Νοημοσύνης στα Βιντεοπαιχνίδια

Ιστορικά, η εφαρμογή της Τεχνητής Νοημοσύνης στα βιντεοπαιχνίδια (Game AI) διέφερε ριζικά από την ακαδημαϊκή έρευνα. Ενώ η ακαδημαϊκή κοινότητα εστίαζε στη δημιουργία συστημάτων που "μαθαίνουν" (Machine Learning), η βιομηχανία των παιχνιδιών χρειαζόταν συστήματα που να είναι υπολογιστικά ελαφριά, απολύτως προβλέψιμα και ελεγχόμενα από τους σχεδιαστές (Designers). Αυτή η ανάγκη οδήγησε στην κυριαρχία της Συμβολικής Τεχνητής Νοημοσύνης (Symbolic AI ή Good Old-Fashioned AI - GOFAI), η οποία βασίζεται σε αυστηρά προγραμματισμένους κανόνες (Hardcoded Logic) και δέντρα αποφάσεων[5].
Για να γίνει κατανοητή η αναγκαιότητα μετάβασης στη Βαθιά Ενισχυτική Μάθηση (Deep RL) στο παρόν έργο, κρίνεται σκόπιμη η εξέταση των επικρατέστερων παραδοσιακών μοντέλων:

3.1.1 Μηχανές Πεπερασμένων Καταστάσεων (Finite State Machines - FSM)

Η Μηχανή Πεπερασμένων Καταστάσεων αποτελεί την πιο παλιά και διαδεδομένη αρχιτεκτονική Game AI. Ένα σύστημα FSM αποτελείται από ένα σύνολο διακριτών καταστάσεων (π.χ. «Περιπολία», «Επίθεση», «Υποχώρηση») και κανόνων μετάβασης (Transitions) μεταξύ αυτών. Σε κάθε χρονική στιγμή, ο πράκτορας (agent) μπορεί να βρίσκεται σε ακριβώς *μία* κατάσταση. Όταν πληρούται μια συνθήκη (π.χ. "Η υγεία έπεσε κάτω από 20%"), ενεργοποιείται μια μετάβαση και ο πράκτορας περνά στην κατάσταση «Υποχώρηση».

- **Πλεονεκτήματα:** Είναι εξαιρετικά εύκολες στην υλοποίηση, καταναλώνουν ελάχιστη μνήμη και η ροή εκτέλεσής τους είναι άμεσα κατανοητή.
- **Μειονεκτήματα:** Πάσχουν από το πρόβλημα της "Έκρηξης των Καταστάσεων" (State Explosion). Σε ένα παιχνίδι διαχείρισης πόρων, όπως το 3D City-Builder, οι συνδυασμοί μεταβλητών (Χρήματα, Νερό, Τροφή, Πληθυσμός) δημιουργούν χιλιάδες πιθανές καταστάσεις. Η χειροκίνητη συγγραφή κανόνων (if-else) για κάθε πιθανό συνδυασμό πόρων καθιστά τη συντήρηση του κώδικα αδύνατη.

3.1.2 Δέντρα Συμπεριφοράς (Behavior Trees - BT)

Τα Δέντρα Συμπεριφοράς αναπτύχθηκαν για να λύσουν το πρόβλημα της πολυπλοκότητας των FSM. Ένα BT είναι μια δενδρική δομή ελέγχου ροής. Ξεκινάει από έναν Κόμβο Ρίζας (Root Node) και διακλαδίζεται σε Κόμβους Ελέγχου (Selectors και Sequences), καταλήγοντας σε Κόμβους-Φύλλα (Leaf Nodes) που περιέχουν τις τελικές ενέργειες (Actions). Σε κάθε καρέ (frame) του παιχνιδιού, το σύστημα σαρώνει το δέντρο (tick). Ένας Κόμβος Επιλογής (Selector) εκτελεί τα παιδιά του από αριστερά προς τα δεξιά μέχρι ένα από αυτά να επιστρέψει "Επιτυχία" (Success).

- **Πλεονεκτήματα:** Είναι αρθρωτά (modular). Ο προγραμματιστής μπορεί να ανακυκλώσει "κλαδιά" του δέντρου για διαφορετικούς εχθρούς.

- **Μειονεκτήματα:** Παραμένουν απολύτως ντετερμινιστικά συστήματα. Ένα BT δεν "μαθαίνει", απλώς εκτελεί το μονοπάτι που έγραψε ο προγραμματιστής. Εάν ο παίκτης βρει ένα λογικό κενό (exploit) στο δέντρο, ο πράκτορας δεν θα μπορέσει ποτέ να προσαρμόσει τη στρατηγική του.

3.1.3 Συστήματα Βασισμένα σε Χρησιμότητα (Utility AI)

Το Utility AI αντιπροσωπεύει το μεταβατικό στάδιο μεταξύ της αυστηρής λογικής (if-else) και της πιο προηγμένης "σκέψης". Αντί να αποφασίζει τι θα κάνει με βάση ένα δέντρο, ο πράκτορας βαθμολογεί κάθε πιθανή ενέργεια με έναν μαθηματικό τύπο (Συνάρτηση Χρησιμότητας / Utility Function). Ο πράκτορας υπολογίζει το Utility για όλες τις δράσεις και επιλέγει αυτή με το μεγαλύτερο σκορ. Το πρώτο πρωτότυπο του αυτόματου συστήματος υποδείξεων που αναπτύχθηκε στην παρούσα εργασία (Η συνάρτηση SuggestBestAction) βασίστηκε στη φιλοσοφία του Utility AI, χρησιμοποιώντας την αξιολόγηση von Neumann για τον υπολογισμό του σκορ κάθε κελιού.

3.1.4 Σχεδιασμός Ενεργειών με Βάση τον Στόχο (Goal-Oriented Action Planning - GOAP)

Το GOAP αποτελεί μια αρχιτεκτονική Προγραμματισμού Ακολουθιών. Σε αντίθεση με το FSM όπου του λέμε "Τι να κάνει", στο GOAP ορίζουμε στον πράκτορα "Τι θέλουμε να πετύχει" (έναν Στόχο) και του δίνουμε ένα λεξικό με διαθέσιμες πράξεις. Ο πράκτορας χρησιμοποιεί έναν αλγόριθμο αναζήτησης προς τα πίσω (backward chaining), για να βρει ποια σειρά από μικρές ενέργειες θα τον οδηγήσει στην εκπλήρωση του στόχου.

3.2 Η Μετάβαση στη Μηχανική Μάθηση (Machine Learning)

Όλα τα παραπάνω συστήματα έχουν έναν κοινό παρονομαστή: αποτελούν Κανόνες Βασισμένους σε Γνώση (Knowledge-Based Rules). Ο προγραμματιστής πρέπει να γνωρίζει εκ των προτέρων τη λύση του προβλήματος και να την κωδικοποιήσει. Ωστόσο, σε περιβάλλοντα Διαχείρισης Πόρων (Resource Management), η αλληλεξάρτηση των μεταβλητών είναι χαοτική. Η επίλυση αυτού του αδιεξόδου απαιτεί ένα σύστημα που δεν ακολουθεί οδηγίες, αλλά ανακαλύπτει την ιδανική στρατηγική μέσω εμπειρίας. Αυτή η ανάγκη αποτέλεσε τον επιστημονικό λόγο που η παρούσα εργασία απέρριψε τις παραδοσιακές μεθόδους της βιομηχανίας και στράφηκε στην Ενισχυτική Μάθηση.

3.3 Εισαγωγή στην Τεχνητή Νοημοσύνη (A.I.) στα Ψηφιακά Παιχνίδια

Η Τεχνητή Νοημοσύνη (Artificial Intelligence) διακρίνεται παραδοσιακά σε τρεις μεγάλους κλάδους μηχανικής μάθησης: την Επιβλεπόμενη Μάθηση (Supervised Learning), τη Μη Επιβλεπόμενη Μάθηση (Unsupervised Learning) και την Ενισχυτική Μάθηση (Reinforcement Learning - RL).

Στην παρούσα πτυχιακή εργασία, το σύστημα υποδείξεων (AI Advisor) υλοποιήθηκε αποκλειστικά με χρήση Ενισχυτικής Μάθησης. Σε αντίθεση με την Επιβλεπόμενη Μάθηση, όπου το μοντέλο εκπαιδεύεται σε έτοιμα, επισημειωμένα δεδομένα (labeled data), η Ενισχυτική Μάθηση βασίζεται στην αλληλεπίδραση μέσω δοκιμής και πλάνης (trial and error). Ένας τεχνητός πράκτορας (agent) τοποθετείται σε ένα περιβάλλον (environment) και καλείται να λάβει μια σειρά από αποφάσεις. Για κάθε του ενέργεια, το περιβάλλον επιστρέφει μια ανατροφοδότηση, η οποία ονομάζεται ανταμοιβή (reward) – θετική για επιθυμητές συμπεριφορές και αρνητική (ποινή) για ανεπιθύμητες. Ο τελικός σκοπός του

πράκτορα είναι να ανακαλύψει μια στρατηγική που μεγιστοποιεί το συνολικό, μακροπρόθεσμο άθροισμα αυτών των ανταμοιβών.

3.4 Μαρκοβιανές Διαδικασίες Λήψης Αποφάσεων (Markov Decision Processes - MDP)

Για να μπορέσει ένας υπολογιστής να λύσει ένα πρόβλημα Ενισχυτικής Μάθησης, το πρόβλημα αυτό πρέπει πρώτα να μοντελοποιηθεί μαθηματικά. Το καθιερωμένο πλαίσιο για αυτή τη μοντελοποίηση είναι οι Μαρκοβιανές Διαδικασίες Λήψης Αποφάσεων (MDPs). Ένα MDP ορίζεται τυπικά ως μια πλειάδα (tuple) 5 στοιχείων: (S, A, P, R, γ).

- **S (State Space - Χώρος Καταστάσεων):** Είναι το σύνολο όλων των πιθανών καταστάσεων στις οποίες μπορεί να βρεθεί το περιβάλλον. Στο 3D City-Builder, η κατάσταση S αποτελείται από τους παγκόσμιους πόρους (χρήματα, φαγητό, νερό) και τη χωρική διάταξη των κτιρίων στο πλέγμα.
- **A (Action Space - Χώρος Ενεργειών):** Το σύνολο των διαθέσιμων κινήσεων του πράκτορα. Στην περίπτωσή μας, είναι ένας διακριτός χώρος (Discrete Action Space) που περιλαμβάνει την επιλογή συντεταγμένων (x, z), τον τύπο κτιρίου ή τον τερματισμό του γύρου.
- **P (Transition Probability):** Η πιθανότητα το περιβάλλον να μεταβεί στη νέα κατάσταση s', δεδομένου ότι ο πράκτορας βρισκόταν στην κατάσταση s και εκτέλεσε την ενέργεια a. Επειδή το παιχνίδι μας περιλαμβάνει τυχαία γεγονότα (όπως ο καύσωνας ή η επιδημία), η μετάβαση είναι στοχαστική.
- **R (Reward Function - Συνάρτηση Ανταμοιβής):** Η άμεση ανταμοιβή που λαμβάνει ο πράκτορας μετά τη μετάβαση.
- **γ (Discount Factor - Συντελεστής Απόσβεσης):** Μια παράμετρος που καθορίζει πόσο "αξίζουν" οι μελλοντικές ανταμοιβές σε σχέση με τις άμεσες. Μια τιμή κοντά στο 1 (π.χ. gamma = 0.99 που χρησιμοποιήθηκε στην εκπαίδευση) αναγκάζει τον πράκτορα να έχει μακροπρόθεσμο ορίζοντα σχεδιασμού.

Η βασική ιδιότητα ενός MDP είναι η **Μαρκοβιανή Ιδιότητα (Markov Property)**, η οποία ορίζει ότι το μέλλον εξαρτάται αποκλειστικά από το παρόν και όχι από το παρελθόν. Δηλαδή, η τρέχουσα κατάσταση s (όπως εξάγεται από τη μέθοδο CollectObservations) περιέχει όλη την απαραίτητη πληροφορία για τη λήψη της επόμενης απόφασης.

3.4.1 Πολιτική (Policy) και Εξίσωση Bellman

Η λύση ενός MDP είναι μια Πολιτική π, η οποία είναι μια συνάρτηση που αντιστοιχίζει κάθε κατάσταση s σε μια ενέργεια a,π(a|s). Σκοπός είναι η εύρεση της βέλτιστης πολιτικής π* που μεγιστοποιεί την Αναμενόμενη Απόδοση (Expected Return).

Για την αξιολόγηση μιας κατάστασης χρησιμοποιείται η Συνάρτηση Αξίας (Value Function) $V^{\pi}(s)$, η οποία υπολογίζεται αναδρομικά μέσω της **Εξίσωσης Bellman** [6]:

$$V^{\pi}(s) = \sum_{a} p(a|s) \sum_{s',r} P(s', r|s, a)[r + \gamma V^{\pi}(s')] \qquad (1)$$

- $V^{\pi}(s)$:Είναι η **Αξία** (Value) της τρέχουσας κατάστασης s, ακολουθώντας την πολιτική π. Στο παιχνίδι σου, αυτός είναι ένας αριθμός που δείχνει πόσο "καλό" είναι το τρέχον grid (π.χ. αν έχεις πολλά λεφτά και νερό, το V είναι υψηλό).

- Σ_{α}: Άθροισμα για όλες τις δυνατές ενέργειες a που μπορεί να κάνει το AI (π.χ. χτίσε σπίτι, χτίσε φάρμα, end turn).
- $p(a|s)$: Η **Πολιτική** (Policy). Είναι η πιθανότητα το δίκτυο να επιλέξει την ενέργεια a δεδομένης της κατάστασης s.
- $\Sigma_{s',r}$: Άθροισμα για όλες τις πιθανές επόμενες καταστάσεις s' και τις ανταμοιβές r που μπορεί να προκύψουν.
- $P(s', r|s, a)$: Η πιθανότητα μετάβασης. Αν το AI κάνει την κίνηση a στην κατάσταση s, ποια είναι η πιθανότητα να βρεθεί στο νέο grid s' και να πάρει την ανταμοιβή r;
- r: Η άμεση ανταμοιβή (Immediate Reward) που παίρνει το AI σε αυτό το βήμα (π.χ. το -0.01 αν κάνει λάθος κλικ).
- γ: Ο **Συντελεστής Απόσβεσης** (Discount Factor, π.χ. 0.99). Καθορίζει πόσο μας νοιάζει το μέλλον.
- $V^{\pi}(s')$: Η αξία της *επόμενης* κατάστασης s'.

Η εξίσωση λέει κάτι πολύ λογικό: *"Η αξία της πόλης μου σήμερα, ισούται με την άμεση ανταμοιβή που θα πάρω από την επόμενη κίνησή μου, ΣΥΝ την αξία που θα έχει η πόλη μου αύριο"*. Επειδή όμως ο agent δεν ξέρει σίγουρα τι θα γίνει, υπολογίζει έναν *μέσο όρο* (αναμενόμενη τιμή) δοκιμάζοντας νοητά όλες τις πιθανές κινήσεις και τις πιθανές συνέπειές τους.

Η εξίσωση αυτή δηλώνει ότι η αξία μιας τρέχουσας κατάστασης ισούται με την άμεση ανταμοιβή που θα λάβουμε, συν την προεξοφλημένη αξία της επόμενης κατάστασης.

3.5 Βαθιά Ενισχυτική Μάθηση (Deep Reinforcement Learning)

Σε απλά προβλήματα, η εξίσωση Bellman μπορεί να λυθεί ακριβώς δημιουργώντας έναν πίνακα (Q-Table) που αποθηκεύει την αξία κάθε κατάστασης. Ωστόσο, στο παρόν παιχνίδι, το πλέγμα έχει μέγεθος 10 x 10, και κάθε κελί μπορεί να πάρει πολλαπλές τιμές (τύπος εδάφους, τύπος κτιρίου). Αυτό δημιουργεί έναν χώρο καταστάσεων (State Space) της τάξης του $O(c^{100})$, ένας αριθμός μεγαλύτερος από τα άτομα στο σύμπαν. Είναι αδύνατον να αποθηκευτεί σε συμβατική μνήμη υπολογιστή.

Για την αντιμετώπιση αυτού του προβλήματος –γνωστού ως «Κατάρα της Διαστατικότητας» (Curse of Dimensionality)– χρησιμοποιείται η **Βαθιά Ενισχυτική Μάθηση (Deep RL)**. Αντί για πίνακες, χρησιμοποιούνται Τεχνητά Νευρωνικά Δίκτυα (Artificial Neural Networks - ANNs) ως συναρτήσεις προσέγγισης (Function Approximators). Το Νευρωνικό Δίκτυο λαμβάνει ως είσοδο (Input Layer) το διάνυσμα των 205 τιμών της CollectObservations. Η πληροφορία περνάει μέσα από κρυφά επίπεδα (Hidden Layers) νευρώνων, όπου πολλαπλασιάζεται με πίνακες βαρών (weights) και περνάει από μη-γραμμικές συναρτήσεις ενεργοποίησης (όπως η ReLU). Η έξοδος του δικτύου (Output Layer) είναι μια κατανομή πιθανοτήτων πάνω στις διαθέσιμες ενέργειες (Action Space). Κατά την εκπαίδευση, τα βάρη του δικτύου ανανεώνονται συνεχώς μέσω του αλγορίθμου Backpropagation.

3.6 Ο Αλγόριθμος Proximal Policy Optimization (PPO)

Το Unity ML-Agents χρησιμοποιεί ως προεπιλογή τον αλγόριθμο **Proximal Policy Optimization (PPO)**, ο οποίος αναπτύχθηκε από την OpenAI το 2017 [7]. Ο PPO ανήκει στην οικογένεια των αλγορίθμων Policy Gradient και χρησιμοποιεί την αρχιτεκτονική **Actor-Critic**.

- **Actor (Ηθοποιός):** Είναι το τμήμα του Νευρωνικού Δικτύου που "βλέπει" την κατάσταση και αποφασίζει ποια ενέργεια θα εκτελέσει (ελέγχει την πολιτική π).

- **Critic (Κριτικός):** Είναι το τμήμα που αξιολογεί την απόφαση του Actor, υπολογίζοντας τη Συνάρτηση Αξίας V(s). Συγκρίνοντας την προσδοκώμενη αξία με την πραγματική ανταμοιβή, ο Critic παράγει ένα σφάλμα (Advantage, $\hat{A}_t$). Αν η ενέργεια ήταν καλύτερη από το αναμενόμενο, το Advantage είναι θετικό.

Η Μαθηματική Καινοτομία του PPO (Clipping)

Το μεγαλύτερο πρόβλημα των παλαιότερων αλγορίθμων (όπως ο REINFORCE) ήταν η αστάθεια. Ένα «κακό» βήμα ανανέωσης των βαρών μπορούσε να καταστρέψει εντελώς την πολιτική του πράκτορα (Catastrophic Forgetting).

Ο PPO λύνει αυτό το πρόβλημα εισάγοντας έναν μαθηματικό περιορισμό (Clipping) στην Αντικειμενική του Συνάρτηση. Ορίζουμε τον λόγο πιθανοτήτων $r_t(\theta)$ ως την πιθανότητα της νέας πολιτικής προς την πιθανότητα της παλιάς πολιτικής. Η συνάρτηση απωλειών (Loss Function) του PPO ορίζεται ως:

$$L^{CLIP}(\Theta) = \hat{E}_t[\min(r_t(\theta)\hat{A}_t, clip(r_t(\theta), 1-\varepsilon, 1+\varepsilon)\hat{A}_t) \quad (2)$$

Επεξήγηση:

- $L^{CLIP}(\Theta)$: **Η συνάρτηση Απώλειας/Στόχου** (Loss) για τα βάρη θ του Νευρωνικού Δικτύου. Θέλουμε να τη μεγιστοποιήσουμε για να βελτιωθεί το AI.
- $\hat{E}_t$: **Η αναμενόμενη τιμή** (Expected value) σε ένα χρονικό βήμα t. (Ο μέσος όρος ενός batch δεδομένων κατά την εκπαίδευση).
- $r_t(\theta)$: Ο **Λόγος Πιθανοτήτων** (Probability Ratio). Είναι το κλάσμα $\frac{\pi_{new}(a|s)}{\pi_{old}(a|s)}$. Δείχνει πόσο πιο πιθανό είναι το AI να κάνει αυτή την κίνηση *τώρα* σε σχέση με το *πριν* ανανεώσει τα βάρη του.
- $\hat{A}_t$: Το **Πλεονέκτημα** (Advantage). Δείχνει πόσο καλύτερη ήταν η κίνηση που έκανε το AI σε σχέση με τον μέσο όρο. Αν το $\hat{A}_t > 0$, η κίνηση ήταν τέλεια (π.χ. έχτισε WaterPlant όταν διψούσαν). Αν $\hat{A}_t < 0$, η κίνηση ήταν κακή (π.χ. έκανε End Turn χωρίς λόγο).
- $clip(r_t(\theta), 1-\varepsilon, 1+\varepsilon)$: Κόβει τον λόγο r_tώστε να μην ξεφύγει κάτω από 1-ε (π.χ. 0.8) και πάνω από 1 + ε (π.χ. 1.2). Το ε είναι συνήθως 0.2.
- Min(…): Παίρνει τη μικρότερη τιμή μεταξύ της κανονικής εξίσωσης και της "κομμένης" (clipped) εξίσωσης.

Η συνάρτηση clip() "κόβει" δραστικά (περιορίζει) τον λόγο $r_t(\theta)$ στο διάστημα [1-ε,1+ε] (συνήθως ε = 0.2). Πρακτικά, αυτός ο τύπος εγγυάται ότι το Νευρωνικό Δίκτυο **δεν θα αλλάξει τη συμπεριφορά του υπερβολικά πολύ σε ένα μόνο βήμα εκπαίδευσης**, εξασφαλίζοντας σταθερή (monotonic) και ασφαλή σύγκλιση, γεγονός που επέτρεψε στον πράκτορα CityBuilderAgent να ανακαλύψει με επιτυχία τις πολύπλοκες εξαρτήσεις πόρων του παιχνιδιού.

ΚΕΦΑΛΑΙΟ 4: ΥΛΟΠΟΙΗΣΗ ΚΑΙ ΑΡΧΙΤΕΚΤΟΝΙΚΗ ΤΟΥ 3D CITY-BUILDER

Η ανάπτυξη του παιχνιδιού πραγματοποιήθηκε εξ ολοκλήρου στο περιβάλλον Unity 6, με τον κώδικα να υλοποιείται στη γλώσσα C#. Σε αυτό το κεφάλαιο εξετάζεται η διαδρομή από τον αρχικό σχεδιασμό μέχρι τη δομή του κώδικα, ξεκινώντας με τους βασικούς μηχανισμούς και φτάνοντας στην αρχιτεκτονική των αυτόνομων υποσυστημάτων.

4.1 Concept Παιχνιδιού, Σενάριο και Βασικοί Μηχανισμοί

Το έργο είναι ένα τρισδιάστατο παιχνίδι διαχείρισης πόλεων (city-builder) που εξελίσσεται πάνω σε ένα δυναμικά παραγόμενο πλέγμα (grid). Ο παίκτης αναλαμβάνει τον ρόλο του "Δημάρχου" και έχει ως κύριο στόχο (win condition) να φτάσει τους 100 κατοίκους προτού λήξουν οι 20 γύροι, αποφεύγοντας την οικονομική καταστροφή.

Ο πυρήνας του συστήματος (core loop) βασίζεται στην εξισορρόπηση πέντε βασικών μακροοικονομικών πόρων:

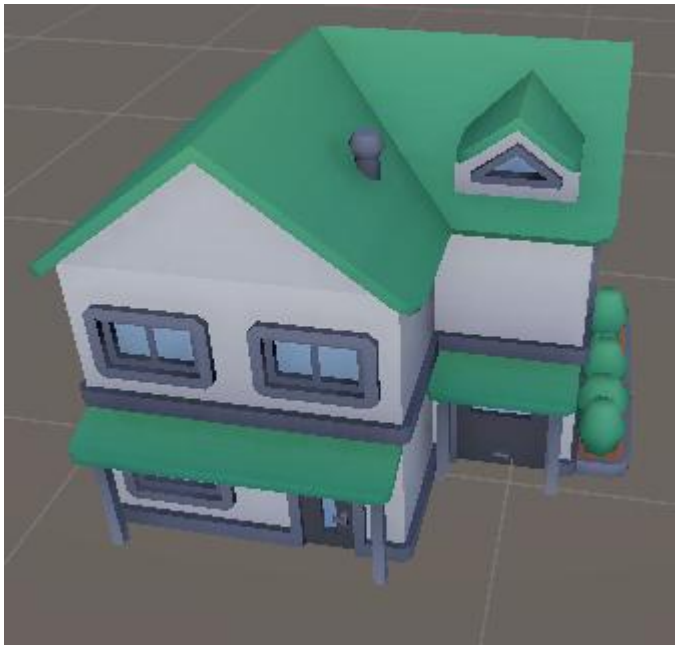
1. **Χρήματα (Money):** Ο βασικός πόρος κατασκευής και συντήρησης.

2. **Φαγητό (Food) & Νερό (Water):** Πόροι επιβίωσης που καταναλώνονται σταδιακά από τον πληθυσμό

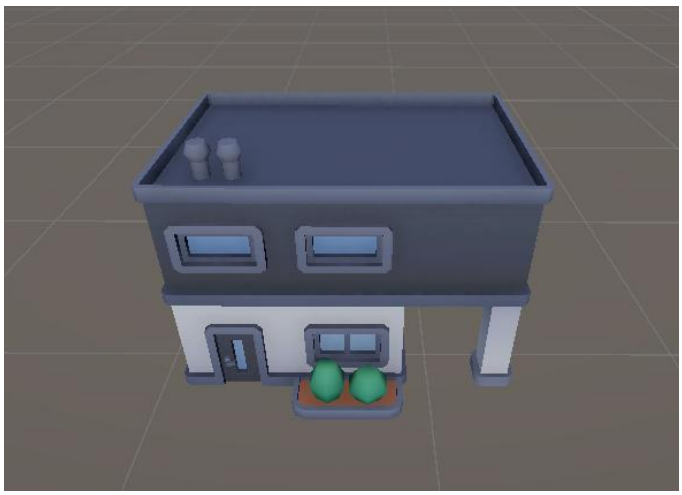
3. **Πληθυσμός (Population):** Ο δείκτης επιτυχίας του παίκτη, ο οποίος αυξάνεται όταν υπάρχει διαθέσιμη χωρητικότητα (μέσω σπιτιών) και επάρκεια πόρων επιβίωσης.

4. **Ευημερία (Welfare):** Μια μεταβλητή (από 0 έως 100) που επηρεάζει τον ρυθμό ανάπτυξης του πληθυσμού και μεταβάλλεται από τις αποφάσεις χωροθέτησης και τα τυχαία γεγονότα.

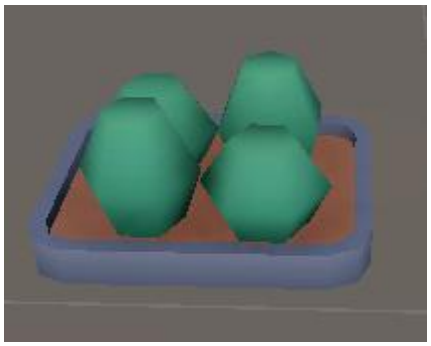
Η ανάπτυξη του αστικού ιστού πραγματοποιείται μέσω της κατασκευής πέντε διαφορετικών τύπων κτιρίων: Σπίτια (House), Φάρμες (Farm), Αντλίες Νερού (WaterPlant), Αγορές (Market) και Νοσοκομεία (Hospital). Κάθε κτίριο διαθέτει μοναδικές ιδιότητες παραγωγής ή κατανάλωσης, δημιουργώντας ένα πολύπλοκο δίκτυο αλληλεξαρτήσεων που καλείται να διαχειριστεί ο χρήστης.



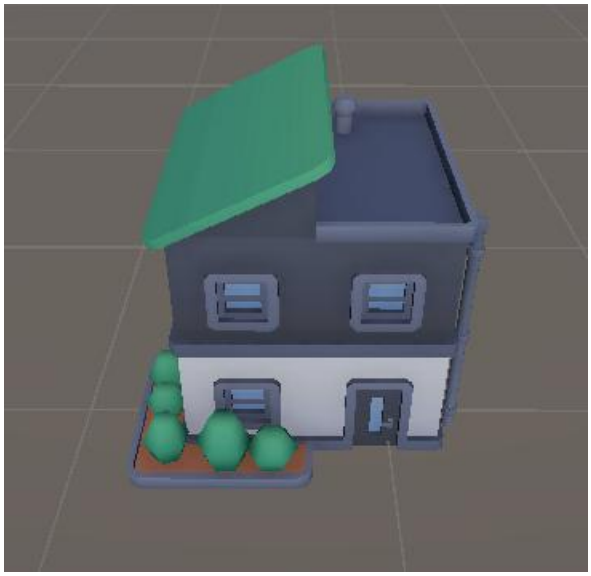
Εικόνα 9 Το μοντέλο του κτηρίου house



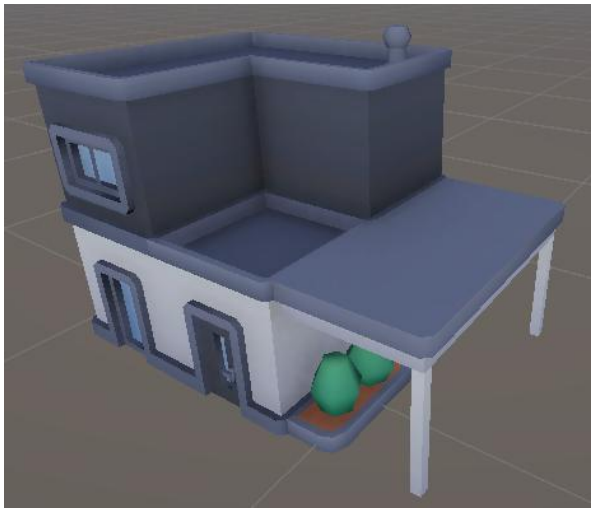
Εικόνα 10 Το μοντέλο του κτηρίου market



Εικόνα 11 Το μοντέλο του κτηρίου farm



Εικόνα 12 Το μοντέλο του κτηρίου waterplant



Εικόνα 13 το μοντέλο του κτηρίου hospital

4.1.1 Μαθηματική Θεμελίωση και Ισορροπία Συστηματικών Πόρων

Η σταθερότητα του περιβάλλοντος προσομοίωσης καθορίζεται από έναν πίνακα κανόνων (Deterministic State Rules). Για να είναι δυνατή η εκπαίδευση του πράκτορα (ML-Agent), οι οικονομικές αλληλεπιδράσεις περιγράφονται από το παρακάτω μαθηματικό μοντέλο, όπου η κατάσταση των πόρων στον γύρο t+1 εξαρτάται από την κατάσταση στον γύρο t και το διάνυσμα των υπαρχόντων κτιρίων:

- Οικονομικός Πόρος (Money):

$$money_{t+1} = money_t + (2 * Houses) - (5 * Farms) - (5 * WaterPlants) - Cost_{Build} \quad (3)$$

- Αποθέματα Φαγητού (Food):

$$food_{t+1} = food_t + (10 * Farms) - (2 * Houses) \quad (4)$$

- Αποθέματα Νερού (Water):

$$Water_{t+1} = Water_t + (10 * WaterPlants) - (2 * Houses) \quad (5)$$

4.1.2 Μοντελοποίηση Πληθυσμιακής Δυναμικής

Η μεταβλητή του πληθυσμού (Pop) αποτελεί τη βασική συνάρτηση στόχου. Η αύξησή της είναι δυναμική και ελέγχεται από τη χωρητικότητα στέγασης (Cap = 5 *Houses) και το επίπεδο ευημερίας (Welfare):

$$pop = \begin{cases} \left(\frac{welfare}{10}\right) + 1 \; \alpha v \; food > 0 \; \wedge water > 0 \; \wedge pop < cap \\ -5 \; \alpha v \; Food \; \leq 0 \; \vee Water \leq 0 \end{cases} \quad (6)$$

4.2 Σχεδιασμός Διεπαφής Χρήστη (UI) και Αλληλεπίδραση Ανθρώπου-Πράκτορα (Human-Agent Teaming)

4.2.1 Αρχές HCI και Διαχείριση Γνωστικού Φόρτου (Cognitive Load)

Τα παιχνίδια στρατηγικής τύπου city-builder χαρακτηρίζονται από υψηλή πυκνότητα πληροφορίας. Ο σχεδιασμός της διεπαφής (HUD) πρέπει να ακολουθεί τους κανόνες της οπτικής ιεραρχίας και της άμεσης ανατροφοδότησης (feedback loops). Η ανάλυση θα εστιάσει πώς οι πέντε βασικοί πόροι (χρήματα, φαγητό, νερό, πληθυσμός, ευημερία) τοποθετούνται σταθερά στην κορυφή της οθόνης, διασφαλίζοντας ότι ο παίκτης διατηρεί συνεχή επίγνωση της κατάστασης (situational awareness) χωρίς να αποσπάται από το κεντρικό πλέγμα δόμησης.

```
[Header("Resources (Πόροι)")]
public int population = 0;
public int food = 100;
public int water = 100;
public int money = 100;
public int welfare = 50;
```

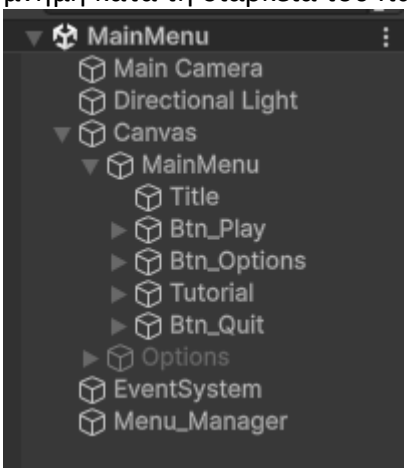
Εικόνα 12 οι αρχικες τιμές του κάθε πόρου

Εικόνα 13 οι αρχικές τιμές των πόρων όταν γίνεται εκκίνηση του παιχνιδιού

4.2.2 Αρχιτεκτονική Μενού και Μηχανισμοί Σειριοποίησης (MainMenuManager.cs)

Εδώ παρουσιάζεται η διαχείριση των καταστάσεων του UI (UI State Management) ως μια απλή μηχανή καταστάσεων (Finite State Machine).

- **Εναλλαγή Panels**: Η λειτουργικότητα των μεθόδων OpenOptions() και CloseOptions() επιτυγχάνεται μέσω ενεργοποίησης ή απενεργοποίησης των αντικειμένων με SetActive.
- **Σειριοποίηση Δεδομένων (Data Serialization):** Η χρήση της κλάσης PlayerPrefs για την τοπική αποθήκευση ρυθμίσεων (όπως PlayerPrefs.SetFloat("volume")) αποτελεί έναν μηχανισμό διατήρησης της κατάστασης του συστήματος (persistence). Αυτός ενεργοποιείται κατά την εκκίνηση (Start()) και αποτρέπει περιττές εγγραφές στη μνήμη κατά τη διάρκεια του παιχνιδιού.



Εικόνα 14: το hierarchy της σκηνής main menu

4.2.3 Σύστημα Χωρικής Ανατροφοδότησης και Billboarding (FloatingText.cs)

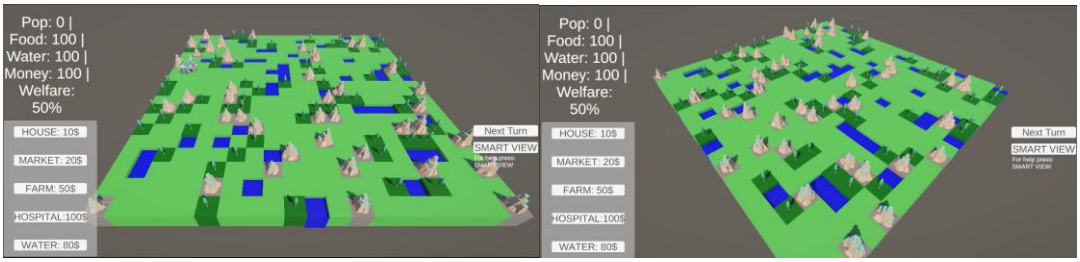
Το FloatingText.cs αποτελεί παράδειγμα χωρικού UI (Spatial UI), καθώς τα στοιχεία του παράγονται απευθείας στον τρισδιάστατο κόσμο και όχι σε στατικό δισδιάστατο επίπεδο (Screen Space)

```
Unity Message | 0 references
void Update()
{
    transform.position += Vector3.up * moveSpeed * Time.deltaTime;

    if (Camera.main != null)
    {
        transform.LookAt(transform.position + Camera.main.transform.rotation * Vector3.forward,
                         Camera.main.transform.rotation * Vector3.up);
    }
}
```

Απόσπασμα από το script FloatingText.cs

- **Μαθηματικά Προσανατολισμού (Billboarding):** Η χρήση της συνάρτησης transform.LookAt σε συνδυασμό με το (Camera.main.transform.rotation) λειτουργεί ως μετασχηματισμός πλέγματος (matrix transformation), ο οποίος ευθυγραμμίζει το διάνυσμα προ πορείας (forward vector) του κειμένου με το διάνυσμα της κάμερας. Αυτό διασφαλίζει ότι το κείμενο παραμένει ευανάγνωστο από οποιαδήποτε γωνία θέασης ή περιστροφή της κάμερας, που γίνεται μέσω των πλήκτρων Q και E.



Εικόνες 15 και 16, το παιχνιδι από δυο διαφορετικές γωνίες θέασης

- **Διαχείριση Πόρων Μνήμης:** Η χρήση της Destroy(gameObject, destroyTime) λειτουργεί ως αυτόματος μηχανισμός εκκαθάρισης μνήμης για την αποφυγή συσσώρευσης ανενεργών αντικειμένων στη σκηνή.

```
Unity Message | 0 references
void Start()
{
    // Καταστροφή μετά από X δευτερόλεπτα
    Destroy(gameObject, destroyTime);

    transform.position += offset;
    transform.position += new Vector3(Random.Range(-0.5f, 0.5f), 0, Random.Range(-0.5f, 0.5f));
}
```
Απόσπασμα από το script FloatingText.cs

4.2.4 Διεπαφή Συνεργασίας Ανθρώπου-Πράκτορα (Human-Agent Teaming)

Μετάφραση του Action Space σε Σημαντική και Χωρική Πληροφορία Στο πλαίσιο της Συνεργασίας Ανθρώπου-Πράκτορα (Human-Agent Teaming), η διεπαφή σχεδιάστηκε ακολουθώντας το πρότυπο Παρατηρητή/Ανάθεσης (Observer/Delegation Pattern). Όταν το AI εξάγει ένα συμπέρασμα (Inference), δεν παρεμβαίνει απευθείας στα στοιχεία του UI. Αντίθετα, μεταφράζει τον ακέραιο αριθμό της δράσης του (buildTypeIndex) στον αντίστοιχο τύπο κτιρίου (BuildingType) και αναθέτει την οπτικοποίηση στον κεντρικό ελεγκτή GridManager μέσω της συνάρτησης ApplyAIVisuals(x, z, suggestedBuilding).

Αποσυμφόρηση Γνωστικού Φόρτου με Ασύγχρονες Ρουτίνες (Coroutines) Μόλις ο κεντρικός ελεγκτής λάβει την υπόδειξη, χρωματίζει το προτεινόμενο κελί (Magenta) και ενημερώνει το HUD. Ωστόσο, στην Αλληλεπίδραση Ανθρώπου-Υπολογιστή (HCI), η μόνιμη παραμονή έντονων οπτικών ενδείξεων στην οθόνη οδηγεί σε κόπωση του χρήστη και οπτική συμφόρηση (visual clutter). Για την επίλυση αυτού του προβλήματος, υλοποιήθηκε ένας μηχανισμός χρονικής λήξης μέσω της ρουτίνας (Coroutine) ClearAIVisualAfterDelay(). Η ρουτίνα αυτή εκτελείται ασύγχρονα και, έπειτα από καθυστέρηση τριών δευτερολέπτων (yield return new WaitForSeconds(3f)), καλεί τη μέθοδο ResetVisuals(), η οποία επαναφέρει τον χάρτη στα φυσικά του χρώματα, καθαρίζοντας τη διεπαφή.

4.3 Μεθοδολογία Σεναρίου, Ροή Παιχνιδιού και Δυναμικά Συστήματα

Η επιτυχία ενός παιχνιδιού στρατηγικής (city-builder) δεν εξαρτάται μόνο από την οπτική του αναπαράσταση, αλλά κυρίως από τη συνοχή του κεντρικού βρόχου (core gameplay loop) και την καμπύλη εκμάθησης (learning curve). Στην παρούσα υλοποίηση, το σύστημα σχεδιάστηκε ώστε να εισάγει ομαλά τον παίκτη στους κανόνες και στη συνέχεια να τον προκαλεί μέσω περιορισμών και δυναμικών γεγονότων.

4.3.1 Σύστημα Ομαλής Ενσωμάτωσης και Εκμάθησης (Player Onboarding / Tutorial)

Η αποτροπή της γνωστικής υπερφόρτωσης (cognitive overload) του χρήστη κατά την πρώτη επαφή με το πολύπλοκο UI του παιχνιδιού, επιλύθηκε με τη δημιουργία ενός απομονωμένου συστήματος εκμάθησης. Η κλάση TutorialManager.cs λειτουργεί ως μια απλή μηχανή πεπερασμένων καταστάσεων (Finite State Machine) που καθοδηγεί τον παίκτη βήμα προς βήμα.

Δομή Δεδομένων και Ροή (Control Flow) Η πληροφορία αποθηκεύεται κεντρικά σε έναν πίνακα συμβολοσειρών (array of strings) με όνομα steps, ο οποίος περιέχει τα διαδοχικά μηνύματα:

```
Unity Script (1 asset reference) | 0 references
public class TutorialManager : MonoBehaviour
{
    public GameObject tutorialPanel; // Το κουτί που θα έχει το κείμενο
    public TextMeshProUGUI instructionText; // Το κείμενο οδηγιών
    public GameObject nextButton; // Κουμπί για το επόμενο βήμα

    private int currentStep = 0;

    //τα βήματα του Tutorial
    private string[] steps = new string[]
    {
        "Καλωσήρθες Δήμαρχε!\nΣκοπός σου είναι να φτάσεις τους 100 κατοίκους.",
        "Χειρισμός Κάμερας:\nW,A,S,D για κίνηση.\nΠοδέλα για Zoom.\nQ,E για Περιστροφή.",
        "Πώς Χτίζεις:\nΔιάλεξε ένα κτίριο από κάτω και κάνε Αριστερό Κλικ στο γρασίδι.",
        "Οικονομία:\nΤα Σπίτια δίνουν λεφτά αλλά θέλουν Νερό και Φαγητό.\nΧτίσε Φάρμες και Αντλίες!",
        "Συμβουλή:\nΑν κολλήσεις, ενεργοποίησε τον AI Βοηθό από το κουμπί Scan.",
        "Καλή τύχη!\nΠάτα 'Next' για να ξεκινήσεις."
    };

    Unity Message | 0 references
    void Start()
    {
        ShowStep(0);
        Time.timeScale = 0; // Παγώνουμε τον χρόνο όσο διαβάζει!
    }
```

Κατά την εκκίνηση της σκηνής (Start()), η μεταβλητή χρόνου της Unity μηδενίζεται (Time.timeScale = 0), διακόπτοντας πλήρως τον βρόχο φυσικής και τις ενημερώσεις των συναρτήσεων Update() που εξαρτώνται από το Time.deltaTime. Αυτό επιτρέπει στον παίκτη να μελετήσει το περιβάλλον χωρίς τον κίνδυνο απώλειας χρόνου ή πόρων. Ένας εσωτερικός μετρητής currentStep αυξάνεται κάθε φορά που πατάει το κουμπί «Next», και ενημερώνει το UI (instructionText). Όταν ολοκληρωθεί ο πίνακας, το UI κρύβεται και ο χρόνος επανέρχεται κανονικά (Time.timeScale = 1).

4.3.2 Διαχείριση Καταστάσεων Παιχνιδιού (Game State & Pause Mechanics)

Η κύρια ροή του παιχνιδιού ελέγχεται εσωτερικά από την κλάση GridManager.cs μέσω μεταβλητών κατάστασης (boolean flags).

Μηχανισμός Παύσης (Pause System): Ο παίκτης μπορεί να σταματήσει την εξέλιξη του παιχνιδιού πατώντας το πλήκτρο Escape. Η λειτουργία ελέγχεται από τη συνάρτηση TogglePause():

```
public void TogglePause()
{
    isPaused = !isPaused;

    if (pauseMenuObject != null)
        pauseMenuObject.SetActive(isPaused); // Εμφάνιση/Απόκρυψη

    // Παγώνει/Ξεπαγώνει τον χρόνο
    if (isPaused)
        Time.timeScale = 0f; // Ο χρόνος σταματάει
    else
        Time.timeScale = 1f; // Ο χρόνος τρέχει κανονικά
}
```

Αυτή η αρχιτεκτονική διασφαλίζει ότι οι λειτουργίες αλληλεπίδρασης, όπως η ριψοβολία (Raycasting) για την κατασκευή κτιρίων, ακυρώνονται άμεσα αν η μεταβλητή isPaused ή isGameOver είναι αληθής. Αυτό γίνεται με έλεγχο πρώιμης εξόδου στην αρχή της λειτουργίας Update().

Συνθήκες Τερματισμού (Win/Loss Conditions):

Σε κάθε κλήση της EndTurn(), το σύστημα επικυρώνει τα μακροοικονομικά δεδομένα.

1. **Συνθήκη Νίκης:** Εάν ο πληθυσμός ξεπεράσει ή ισούται με 100 (population >= 100), το παιχνίδι τερματίζεται επιτυχώς.
2. **Συνθήκη Ήττας (Χρεοκοπία):** Εάν τα χρήματα γίνουν αρνητικά (money < 0), επιβάλλεται αυστηρή ποινή μηδενισμού του πληθυσμού (population = 0) και καλείται η συνάρτηση GameOver(). Αυτή η ακραία ποινή εξασφαλίζει ότι το ML-Agent, κατά την εκπαίδευσή του, θα λάβει τελική ανταμοιβή 0 σε περίπτωση χρεοκοπίας, αποτρέποντάς το από ριψοκίνδυνες οικονομικές σπατάλες.

4.3.3 Μηχανισμός Δυναμικών Γεγονότων (Random Events & Coroutines)

Για να αποφευχθεί η δημιουργία μιας απόλυτα ντετερμινιστικής και συνεπώς προβλέψιμης εμπειρίας, το παιχνίδι ενσωματώνει συστήματα στοχαστικότητας (stochasticity). Σε κάθε τέλος γύρου (EndTurn), ο αλγόριθμος υπολογίζει την πιθανότητα ενός τυχαίου γεγονότος (TriggerRandomEvent) να ενεργοποιηθεί.

```
void TriggerRandomEvent()
{
    int chance = Random.Range(0, 100);

    // 20% πιθανότητα να συμβεί κάτι
    if (chance > 20) return;

    string message = "";

    // Επιλογή τυχαίου γεγονότος
    int eventType = Random.Range(0, 4); // 0, 1, 2 ή 3

    switch (eventType)
    {
        case 0: // Ξηρασία
            message = "ΚΑΥΣΩΝΑΣ! Η παραγωγή νερού μειώθηκε.";
            water -= 30;
            break;
        case 1: // Επιχορήγηση
            message = "ΚΡΑΤΙΚΗ ΕΠΙΧΟΡΗΓΗΣΗ! Έλαβες επιπλέον χρήματα.";
            money += 50;
            break;
        case 2: // Αρρώστια
            message = "ΕΠΙΔΗΜΙΑ! Η υγεία των κατοίκων έπεσε.";
            welfare -= 15;
            break;
        case 3: // Φεστιβάλ
            message = "ΤΟΠΙΚΟ ΠΑΝΗΓΥΡΙ! Οι κάτοικοι είναι χαρούμενοι.";
            welfare += 10;
            break;
    }
```

Αυτά τα γεγονότα διαταράσσουν την ισορροπία των πόρων:

- **Αρνητικά Γεγονότα:** Όπως ένας "Καύσωνας" που εξαντλεί άμεσα τεράστια ποσότητα νερού, ή μια "Επιδημία" που ρίχνει κατακόρυφα το ποσοστό ευημερίας του πληθυσμού.
- **Θετικά Γεγονότα:** Όπως μια "Κρατική Επιχορήγηση" που αυξάνει απρόσμενα τον προυπολογισμό του παίκτη, ή ένα "Τοπικό Φεστιβάλ" που ενισχύει την ευημερία.

Η εισαγωγή τυχαιότητας εδώ έχει δύο βασικούς σκοπούς: να αυξήσει την επαναληψιμότητα για τον παίκτη και να βελτιώσει τη γενική ανθεκτικότητα του νευρωνικού δικτύου του ML-Agent. Κατά την εκπαίδευση, το AI μαθαίνει να προσαρμόζεται σε απρόβλεπτες αλλαγές του περιβάλλοντος, αποφεύγοντας την αποστήθιση μιας σταθερής σειράς κινήσεων.

```
    // Ελέγχουμε να μην πέσουν κάτω από το 0
    if (water < 0) water = 0;
    if (welfare < 0) welfare = 0;
    if (population < 0) population = 0;
```

Αμέσως μετά την αφαίρεση πόρων, πραγματοποιείται έλεγχος κατωτάτου ορίου (Clamping), διασφαλίζοντας ότι κρίσιμοι δείκτες, όπως το water ή το welfare, δεν θα λάβουν ποτέ αρνητικές τιμές (if (water < 0) water = 0;), αποφεύγοντας έτσι λογικά σφάλματα στα μαθηματικά της προσομοίωσης.

Μια εξαιρετικά σημαντική τεχνική λεπτομέρεια στην υλοποίηση είναι η διαχείριση της οπτικής ενημέρωσης του χρήστη. Όταν ενεργοποιείται ένα γεγονός, καλείται η μέθοδος

ShowEventMessage(string msg). Για να εξαφανιστεί το μήνυμα αυτόματα μετά από 3 δευτερόλεπτα, χρησιμοποιείται η ρουτίνα (Coroutine) HideEventPanel():

```
0 references
System.Collections.IEnumerator HideEventPanel()
{
    yield return new WaitForSeconds(3f);
    if (eventPanel != null) eventPanel.SetActive(false);
}
```

Πριν ξεκινήσει η ρουτίνα, το σύστημα καλεί το StopCoroutine("HideEventPanel"). Αυτή η τεχνική διακόπτει παλαιότερα χρονομέτρα (Reset/Debouncing) για να αποτρέψει συγκρούσεις (race conditions) εάν πολλά μηνύματα εμφανίζονται σε σύντομο χρονικό διάστημα. Επιπλέον, διασφαλίζει ότι αυτό το χρονόμετρο δεν παρεμβαίνει στο χρονόμετρο του AI Advisor (ClearAIVisualAfterDelay), διατηρώντας τα συστήματα οπτικής ανατροφοδότησης πλήρως ανεξάρτητα και αποσυνδεδεμένα.

4.4 Αναλυτική Περιγραφή και Αρχιτεκτονική Συναρτήσεων

Η μεταβαση από τον θεωρητικό σχεδιασμό στην πρακτική υλοποίηση του 3D City-Builder απαιτεί μια αυστηρή ιεραρχική αρχιτεκτονική λογισμικού. Ο κώδικας αναπτύχθηκε αποκλειστικά σε C#, τηρώντας τις αρχές του αντικειμενοστραφούς προγραμματισμού (OOP) και χρησιμοποιώντας το πρότυπο Component-Based Architecture της μηχανής Unity.
Για να διασφαλιστεί χαμηλή σύζευξη και υψηλή συνοχή, το σύστημα χωρίστηκε σε τέσσερις βασικούς πυλώνες.

1. **Ο Κεντρικός Ελεγκτής (Core Controller):** Η κλάση GridManager.cs λειτουργεί ως ο βασικός διαχειριστής της κατάστασης του παιχνιδιού, εποπτεύοντας τους πόρους, τη ροή χρόνου και την παραγωγή του κόσμου.
2. **Οι Δομικές Οντότητες (Data Containers):** που τοποθετούνται σε κάθε κελί μέσω της κλάσης TileCell.cs, η οποία αποθηκεύει την τοπική κατάσταση του κελιού, όπως έδαφος ή κτίριο.
3. **Υποσυστήματα Διεπαφής (UI/UX):** Τα αρχεία όπως το MainMenuManager.cs και το FloatingText.cs είναι υπεύθυνα αποκλειστικά για την οπτική αλληλεπίδραση.
4. **Μονάδα Τεχνητής Νοημοσύνης (AI Module):** Η κλάση CityBuilderAgent.cs διατρέχει τα δεδομένα του κεντρικού ελεγκτή και εξάγει συμπεράσματα, συνδέοντας τη Unity με το ML-Agents API.

Οι υποενότητες που ακολουθούν προσφέρουν μια αναλυτική εξέταση της αλγοριθμικής δομής των σημαντικών συναρτήσεων που αποτελούν αυτούς τους πυλώνες.

4.4.1 Διαχείριση Πλέγματος και Παραγωγή Κόσμου (GridManager.cs & TileCell.cs)

Η χωρική θεμελίωση του τρισδιάστατου city-builder βασίζεται σε ένα δυναμικό, δομημένο σύστημα πλέγματος (Grid-based architecture). Αυτό το σύστημα αποτελεί το βασικό πλαίσιο πάνω στο οποίο γίνονται οι λειτουργίες εγκατάστασης, η διαχείριση οικονομικών πόρων και η συλλογή δεδομένων από τον πράκτορα τεχνητής νοημοσύνης. Η υλοποίηση χωρίζεται σε δύο κύρια επίπεδα: τη δομική μονάδα του περιβάλλοντος (TileCell.cs) και τον κεντρικό ελεγκτή που διαχειρίζεται τον κόσμο (GridManager.cs).

Η Δομή Δεδομένων του Κελιού (TileCell.cs)

Κάθε τετράγωνο στον χάρτη αντιπροσωπεύεται από την κλάση TileCell, η οποία κληρονομεί από τη MonoBehaviour, επιτρέποντας την άμεση σύνδεση με την οπτική και φυσική υπόσταση του αντικειμένου στην Unity. Αυτή η κλάση λειτουργεί ως αυτόνομος αποθηκευτικός χώρος πληροφορίας (State Container) και διατηρεί τα ακόλουθα σημαντικά δεδομένα:

```
public class TileCell : MonoBehaviour
{
    public int x, z;
    public BuildingType currentBuilding = BuildingType.None;

    public TileType tileType = TileType.Grass;

    private GameObject currentModelObject;
```

- public int x, z;: Οι ακέραιες συντεταγμένες του κελιού εντός του δισδιάστατου πίνακα.
- public BuildingType currentBuilding: Ο τύπος του κτιρίου που είναι κατασκευασμένος στο κελί (με αρχική τιμή BuildingType.None).
- public TileType tileType: Ο φυσικός τύπος του εδάφους (Grass, Water, Rock, Forest).

Η λογική ελέγχου δόμησης απομονώνεται αλγοριθμικά εντός της συνάρτησης IsBuildable():

```
public bool IsBuildable()
{
    // Αν υπάρχει ήδη κτίριο, δεν χτίζουμε
    if (currentBuilding != BuildingType.None) return false;

    // Αν είναι Νερό ή Βράχος, δεν χτίζουμε
    if (tileType == TileType.Water || tileType == TileType.Rock) return false;

    return true;
}
```

Η συνάρτηση αυτή επιστρέφει μια Boolean τιμή, διασφαλίζοντας ότι η κατασκευή επιτρέπεται μόνο όταν το κελί είναι κενό και το έδαφος δεν αποτελεί φυσικό εμπόδιο (όπως νερό ή βράχο). Για τη διαχείριση των τρισδιάστατων μοντέλων, η κλάση υλοποιεί τη μέθοδο SetBuilding(), η οποία αναλαμβάνει την καταστροφή του προηγούμενου μοντέλου μέσω της Destroy(currentModelObject) και τη στιγμιοτύπηση (Instantiate) του νέου prefab ως εξαρτημένο αντικείμενο-παιδί (transform.parent = this.transform), βελτιστοποιώντας έτσι την ιεραρχία της σκηνής.

```
public void SetBuilding(BuildingType newType, GameObject buildingPrefab, Quaternion rotation)
{
    currentBuilding = newType;

    if (currentModelObject != null) Destroy(currentModelObject);

    if (newType != BuildingType.None && buildingPrefab != null)
    {
        currentModelObject = Instantiate(buildingPrefab, transform.position, rotation);

        currentModelObject.transform.parent = this.transform;
        currentModelObject.transform.localPosition = new Vector3(0, 0.5f, 0);
    }
}
```

Ο Αλγόριθμος Διαδικαστικής Παραγωγής (GridManager.cs)

Η παραγωγή του κόσμου εκτελείται διαδικαστικά (procedural generation) κατά την εκκίνηση του παιχνιδιού μέσω της συνάρτησης GenerateGrid(). Ο GridManager διατηρεί έναν δισδιάστατο πίνακα αναφοράς gridArray με διαστάσεις που καθορίζονται από τις μεταβλητές width και height.

Η μαθηματική αποτύπωση και η χωρική τοποθέτηση των αντικειμένων στον τρισδιάστατο χώρο (World Space) προκύπτει από τον γραμμικό μετασχηματισμό των δεικτών του πίνακα με βάση τη σταθερά απόστασης spacing:

$$PosX = x * spacing \quad (7)$$

$$PosZ = Z * spacing \quad (8)$$

Κατά τη διάρκεια των εμφωλευμένων βρόχων επανάληψης (for), το σύστημα εκτελεί μια στοχαστική κατανομή εδάφους χρησιμοποιώντας έναν ψευδοτυχαίο αριθμό στο διάστημα [0, 100) μέσω της Random.Range(). Ανάλογα με την τιμή του "roll", το κελί αποκτά την αντίστοιχη ταυτότητα TileType και τοποθετούνται τα αντίστοιχα περιβαλλοντικά prefabs (treePrefab, rockPrefab).

```
void GenerateGrid()
{
    for (int x = 0; x < width; x++)
    {
        for (int z = 0; z < height; z++)
        {
            float posX = x * spacing;
            float posZ = z * spacing;

            GameObject spawnedTile = Instantiate(tilePrefab, new Vector3(posX, 0, posZ), Quaternion.identity);
            spawnedTile.transform.parent = this.transform;
            spawnedTile.name = $"Tile {x}_{z}";

            TileCell cell = spawnedTile.AddComponent<TileCell>();
            cell.x = x;
            cell.z = z;

            // --- RANDOM GENERATION LOGIC ---
            int roll = Random.Range(0, 100);
```

```
// ΠΕΡΙΠΤΩΣΗ 1: ΝΕΡΟ
if (roll < 10)
{
    cell.tileType = TileType.Water;
    spawnedTile.GetComponent<Renderer>().material.color = Color.blue;
    spawnedTile.transform.position += Vector3.down * 0.2f; // πιο χαμηλά απο κανονικο tile
}
```

```
// ΠΕΡΙΠΤΩΣΗ 2: ΒΡΑΧΟΣ
else if (roll < 20)
{
    cell.tileType = TileType.Rock;
    spawnedTile.GetComponent<Renderer>().material.color = Color.gray;

    if (rockPrefab != null)
    {
        GameObject rock = Instantiate(rockPrefab, spawnedTile.transform.position, Quaternion.identity);
        rock.transform.parent = spawnedTile.transform; // Το κάνουμε παιδί του tile
        rock.transform.localPosition = new Vector3(0, 0.5f, 0);
        // Τυχαία περιστροφή και μέγεθος
        rock.transform.rotation = Quaternion.Euler(0, Random.Range(0, 360), 0);
        rock.transform.localScale = Vector3.one * Random.Range(0.8f, 1.2f);
    }
}
```

```
// ΠΕΡΙΠΤΩΣΗ 3: ΔΑΣΟΣ
else if (roll < 35)
{
    cell.tileType = TileType.Forest;
    spawnedTile.GetComponent<Renderer>().material.color = new Color(0.0f, 0.4f, 0.0f);

    if (treePrefab != null)
    {
        GameObject tree = Instantiate(treePrefab, spawnedTile.transform.position, Quaternion.identity);
        tree.transform.parent = spawnedTile.transform;
        tree.transform.localPosition = new Vector3(0, 0.5f, 0);

        // Τυχαία περιστροφή
        tree.transform.rotation = Quaternion.Euler(0, Random.Range(0, 360), 0);
    }
}
```

```
// ΠΕΡΙΠΤΩΣΗ 4: ΓΡΑΣΙΔΙ
else
{
    cell.tileType = TileType.Grass;
    spawnedTile.GetComponent<Renderer>().material.color = new Color(0.3f, 0.8f, 0.3f);
}

// Βάφουμε το tile με το σωστό χρώμα
spawnedTile.GetComponent<Renderer>().material.color = GetColorForTileType(cell.tileType);

gridArray[x, z] = cell;
```

Μια σημαντική τεχνική λεπτομέρεια αφορά τα κελιά τύπου Water: το αντικείμενο μετατοπίζεται κατακόρυφα προς τα κάτω στον άξονα Y (Y_{offset} = -0.2f), δημιουργώντας γεωμετρικό βάθος στον χάρτη, ενώ παράλληλα αποκλείεται αυτόματα από κάθε μελλοντική προσπάθεια δόμησης ή υπόδειξης του AI.

Ζητήματα Απόδοσης και Μελλοντικές Βελτιώσεις (Scalability)

Στην τρέχουσα μορφή του, το σύστημα χρησιμοποιεί τη μέθοδο Instantiate για κάθε tile και περιβαλλοντικό στοιχείο. Αν και η προσέγγιση αυτή είναι ιδανική για πλέγματα μικρής κλίμακας (π.χ. 10 x 10), παρουσιάζει περιορισμούς σε μεγαλύτερους χάρτες λόγω του αυξημένου αριθμού Draw Calls και του φόρτου στο Garbage Collection της Unity.

Για την επέκταση της αρχιτεκτονικής, προτείνεται η ενσωμάτωση του συστήματος **Scriptable Build Pipeline** ή η χρήση **GPU Instancing** μέσω του Universal Render Pipeline (URP) της Unity 6, όπου πανομοιότυπα γεωμετρικά αντικείμενα (όπως δέντρα και βράχοι) σχεδιάζονται σε μία μόνο υπολογιστική πάσα, διατηρώντας τη σταθερότητα των καρέ ανα δευτερόλεπτο (FPS) σε υψηλά επίπεδα.

Σύστημα Προεπισκόπησης Δόμησης (Ghost System) Για την αποφυγή λανθασμένων κινήσεων (user errors) και τη βελτίωση της χωρικής αντίληψης, αναπτύχθηκε ένα σύστημα δυναμικής προεπισκόπησης (Ghost System). Η λειτουργία αυτή ελέγχεται από τις συναρτήσεις UpdateGhostPosition() και RefreshGhost().

Σε κάθε καρέ της μηχανής γραφικών, εκπέμπεται μια ακτίνα (Raycasting) από την κάμερα προς τη θέση του κέρσορα (Camera.main.ScreenPointToRay). Εάν η ακτίνα τέμνει το πλέγμα, ένα ημιδιαφανές τρισδιάστατο μοντέλο (Ghost Object) μετατοπίζεται δυναμικά στις συντεταγμένες του αντίστοιχου κελιού, εφαρμόζοντας παράλληλα μια κατακόρυφη μετατόπιση (Vector3(0, 0.5f, 0)) για την αποφυγή οπτικών σφαλμάτων (clipping) με το έδαφος.

Μια κρίσιμη τεχνική λεπτομέρεια στην αρχιτεκτονική του συστήματος είναι η καταστροφή της γεωμετρίας σύγκρουσης του μοντέλου προεπισκόπησης (Destroy(col)) κατά τη δημιουργία του. Αν το Component του Collider παρέμενε ενεργό, το Ghost Object θα μπλόκαρε (raycast blocking) τις ακτίνες του ποντικιού, καθιστώντας αδύνατη την ανίχνευση του πραγματικού εδάφους και συνεπώς τη δόμηση.

4.4.2 Σύστημα Αξιολόγησης Εδάφους και Υπόδειξης Δράσεων (AI Advisor)

Στα παραδοσιακά παιχνίδια διαχείρισης, η αξιολόγηση της καταλληλότητας μιας τοποθεσίας πραγματοποιείται συνήθως μέσω ντετερμινιστικών ευριστικών αλγορίθμων (heuristic algorithms), οι οποίοι υπολογίζουν βαθμολογίες με βάση προκαθορισμένους κανόνες. Ωστόσο, στην παρούσα υλοποίηση επιλέχθηκε μια πιο προηγμένη αρχιτεκτονική: η πλήρης αντικατάσταση των στατικών αλγορίθμων από ένα δυναμικό Σύστημα Υπόδειξης (Advisor System), το οποίο αντλεί αποφάσεις απευθείας από το εκπαιδευμένο μοντέλο Μηχανικής Μάθησης.

Αυτό το υποσύστημα ελέγχει τη γέφυρα επικοινωνίας μεταξύ της διεπαφής του χρήστη (UI) και του CityBuilderAgent, επιτρέποντας στον παίκτη να ζητά χωροταξικές συμβουλές σε πραγματικό χρόνο.

Η διαδικασία λήψης απόφασης εκκινείται από τον χρήστη μέσω της συνάρτησης TogglePredictionMode() εντός του GridManager. Η αλγοριθμική ροή λειτουργεί ως εξής:

```
0 references
public void TogglePredictionMode()
{

    CityBuilderAgent aiAgent = FindObjectOfType<CityBuilderAgent>();

    if (aiAgent != null)
    {
        aiAgent.RequestOneShotSuggestion();
    }
    else
    {
        ShowNotification("Δεν βρέθηκε το AI Agent!");
    }
}
```

Η μέθοδος ανιχνεύει δυναμικά τον πράκτορα στη σκηνή χρησιμοποιώντας FindObjectOfType και καλέσει τη μέθοδο RequestOneShotSuggestion(). Αυτή η προσέγγιση (One-Shot Inference) διασφαλίζει ότι το νευρωνικό δίκτυο δεν καταναλώνει συνεχώς υπολογιστικούς πόρους, αλλά ενεργοποιείται μόνο όταν απαιτείται από τον παίκτη.

Οπτικοποίηση της Αξιολόγησης (ApplyAIVisuals)

Εφόσον ο πράκτορας αναλύσει τα δεδομένα του πλέγματος και επιστρέψει μια έγκυρη πρόταση δόμησης, ο έλεγχος επιστρέφει στον GridManager μέσω της κλήσης ApplyAIVisuals(int x, int z, BuildingType suggestedBuilding). Η συγκεκριμένη συνάρτηση υλοποιεί μια ακολουθία πέντε βημάτων για τη βέλτιστη καθοδήγηση του παίκτη:

```
public void ApplyAIVisuals(int x, int z, BuildingType suggestedBuilding)
{
    // 1. Καθαρίζουμε τον χάρτη απο προηγούμενες προτάσεις
    ResetVisuals();

    // 2. Ελέγχουμε αν οι συντεταγμένες είναι εντός χάρτη
    if (x >= 0 && x < width && z >= 0 && z < height)
    {
        TileCell cell = gridArray[x, z];

        // 3. Εμφάνιση Κειμένου
        string msg = $"Πρόταση AI: Χτίσε {suggestedBuilding} στη θέση ({x}, {z})";
        if (advisorText != null) advisorText.text = msg;
        ShowNotification(msg);

        // 4. Χρωματισμός του tile σε Magenta
        cell.GetComponent<Renderer>().material.color = Color.magenta;

        // 5. Αυτόματη επιλογή του κτιρίου στο UI
        selectedBuilding = suggestedBuilding;
        RefreshGhost();

        // Σταματάμε τυχόν προηγούμενο χρονόμετρο (αν ο παίκτης σπαμάρει το κουμπί)
        // και ξεκινάμε να μετράμε αντίστροφα για να σβήσουμε το χρώμα
        StopCoroutine("ClearAIVisualAfterDelay");
        StartCoroutine("ClearAIVisualAfterDelay");
    }
}
```

1. **Καθαρισμός Οπτικού Φόρτου:** Αρχικά καλείται η μέθοδος ResetVisuals(), η οποία επαναφέρει όλα τα κελιά στα φυσικά τους χρώματα, διαγράφοντας τυχόν προηγούμενες υποδείξεις.

2. **Ενημέρωση Διεπαφής (UI):** Εφόσον οι συντεταγμένες (x, z) είναι εντός των εγκύρων ορίων του πίνακα gridArray, το σύστημα συνθέτει ένα μήνυμα (π.χ. *"Πρόταση AI: Χτίσε House στη θέση (3, 4)"*) και το προβάλλει τόσο στο στοιχείο advisorText όσο και στο σύστημα ειδοποιήσεων (ShowNotification).
3. **Χωρική Επισήμανση:** Το προτεινόμενο κελί απομονώνεται και το χρώμα του υλικού του (Material Color) μετατρέπεται σε Color.magenta, τραβώντας άμεσα την προσοχή του χρήστη.
4. **Αυτόματη Προετοιμασία Δόμησης:** Για τη βελτίωση της Εμπειρίας Χρήστη (UX), το σύστημα αυτοματοποιεί τις ενέργειες του παίκτη ορίζοντας τη μεταβλητή selectedBuilding ίση με την πρόταση του AI (suggestedBuilding) και ανανεώνοντας το μοντέλο προεπισκόπησης (RefreshGhost()). Έτσι, ο παίκτης χρειάζεται απλώς να κάνει κλικ στο φωτισμένο κελί για να αποδεχθεί την πρόταση.

Διαχείριση Χρόνου Ζωής της Υπόδειξης (Asynchronous Cleanup)

Για να αποφευχθεί η μόνιμη εμφάνιση του χρώματος magenta στον χάρτη, η συνάρτηση ApplyAIVisuals τελειώνει με την έναρξη μιας ασύγχρονης ρουτίνας (Coroutine)

Η λειτουργία ClearAIVisualAfterDelay() περιμένει 3 δευτερόλεπτα (yield return new WaitForSeconds(3f)) και στη συνέχεια καλεί ξανά τη ResetVisuals(). Η χρήση του StopCoroutine πριν από το StartCoroutine είναι πολύ σημαντική: λειτουργεί ως μηχανισμός αποθορυβοποίησης (debouncing). Αν ο χρήστης πατήσει το κουμπί πολλαπλές φορές συνεχόμενα, τα προηγούμενα χρονόμετρα ακυρώνονται, αποτρέποντας συγκρούσεις συνθηκών (race conditions) και διασφαλίζοντας την ομαλή επαναφορά των γραφικών.

4.4.3 Μηχανισμός Οικονομίας, Μοντελοποίηση Συστήματος Γύρων και Πληθυσμιακή Δυναμική

Η κεντρική δομή της προσομοίωσης και το σύστημα ελέγχου της ροής (Game Loop) υλοποιούνται στην κλάση GridManager.cs. Επειδή το παιχνίδι ανήκει στην κατηγορία Turn-Based Strategy (TBS), ο χρόνος προχωρά σε διακριτά χρονικά βήματα.

Η μακροπρόθεσμη μετάβαση του συστήματος από τον γύρο t στον γύρο t+1 ελέγχεται αποκλειστικά από τη συνάρτηση EndTurn(), η οποία πραγματοποιεί πολλαπλές φάσεις υπολογισμού για να διασφαλίσει τη λογική συνεκτικότητα του περιβάλλοντος.

Φάση 1η: Τοπικός Υπολογισμός Πόρων και Σάρωση Πλέγματος

Η συνάρτηση αρχικά μηδενίζει τους προσωρινούς αθροιστές μεταβολής (foodChange, waterChange, moneyChange, welfareChange, housingCapacity). Στη συνέχεια, διατρέχει ολόκληρο τον πίνακα gridArray (Grid Scanning). Για κάθε κελί, αναλύει τον τύπο του υπάρχοντος κτιρίου (currentBuilding) μέσω μιας δομής ελέγχου switch και αθροίζει τους αντίστοιχους οικονομικούς τροποποιητές (modifiers).

Στην τρέχουσα αρχιτεκτονική ισορροπίας, το μοντέλο παραγωγής-κατανάλωσης ορίζεται ως εξής:

```
public void EndTurn()
{
    int foodChange = 0;
    int waterChange = 0;
    int moneyChange = 0;
    int welfareChange = 0;
    int housingCapacity = 0;

    foreach (TileCell cell in gridArray)
    {
        switch (cell.currentBuilding)
        {
            case BuildingType.House:
                housingCapacity += 5;
                waterChange -= 2;
                foodChange -= 2;
                moneyChange += 2;
                ShowFloatingText(cell.transform.position, "+2$", Color.green);
                break;
            case BuildingType.Farm:
                foodChange += 15;
                waterChange -= 3;
                moneyChange -= 1;
                ShowFloatingText(cell.transform.position, "+1$", Color.green);
                break;
            case BuildingType.Market:
                moneyChange += 10;
                break;
            case BuildingType.WaterPlant:
                waterChange += 25;
                moneyChange -= 3;
                ShowFloatingText(cell.transform.position, "-3$", Color.red);
                break;
            case BuildingType.Hospital:
                welfareChange += 5;
                moneyChange -= 5;
                ShowFloatingText(cell.transform.position, "-5$", Color.red);
                break;
        }
    }
```

- **House:** Προσφέρει +5 στη χωρητικότητα στέγασης και +2 έσοδα, αλλά καταναλώνει πόρους επιβίωσης (-2 νερό, -2 φαγητό).
- **Farm:** Παράγει +15 φαγητό, αλλά έχει κόστος συντήρησης σε νερό (-3) και χρήμα (-1).
- **WaterPlant:** Αποτελεί τη βασική πηγή υδροδότησης (+25 νερό), με κόστος συντήρησης -3 χρήματα.
- **Market & Hospital:** Λειτουργούν ως υποστηρικτικές δομές. Η αγορά ενισχύει την οικονομία (+10 χρήματα), ενώ το νοσοκομείο αυξάνει τον δείκτη ευημερίας (+5 welfare) με οικονομικό κόστος (-5 χρήματα).

Φάση 2η: Ενημέρωση Παγκόσμιων Πόρων και Έλεγχος Ορίων (Clamping)

Μετά την ολοκλήρωση της σάρωσης, οι συνολικές μεταβολές προστίθενται στα παγκόσμια αποθέματα (food += foodChange, water += waterChange, money += moneyChange). Ιδιαίτερη έμφαση δίνεται στη μεταβλητή της ευημερίας (welfare), η οποία, ως ποσοστιαίος δείκτης ικανοποίησης, περιορίζεται αυστηρά στο κλειστό μαθηματικό διάστημα [0, 100] μέσω της συνάρτησης περιορισμού της μηχανής: welfare = Mathf.Clamp(welfare + welfareChange, 0, 100).

Φάση 3η: Μαθηματικό Μοντέλο Πληθυσμιακής Δυναμικής

Ο αλγόριθμος αξιολογεί τις συνθήκες διαβίωσης για τον υπολογισμό του πληθυσμού. Εάν υπάρχουν θετικά αποθέματα επιβίωσης (food > 0 και water > 0) και ταυτόχρονα η πόλη διαθέτει ελεύθερη χωρητικότητα (population < housingCapacity), ενεργοποιείται ο μηχανισμός ανάπτυξης. Η αύξηση δεν είναι σταθερή, αλλά συναρτάται γραμμικά από την ευημερία. Η ευημερία υπολογίζεται από την παρακατω εξίσωση:

$$Growth = \left(\frac{welfare}{10}\right) + 1$$

Από την άλλη πλευρά, εάν παρουσιαστεί έλλειψη νερού ή τροφής, το σύστημα επιβάλλει αυστηρή ποινή λιμού, μειώνοντας τον πληθυσμό κατά 5 μονάδες ανά γύρο. Για την αποφυγή σφαλμάτων (bugs) στη λογική του παιχνιδιού, εφαρμόζονται αμέσως μετά δύο απόλυτοι κανόνες (Boundary Checks).

```
if (population < 0) population = 0;
if (population > housingCapacity) population = housingCapacity;
currentTurn++;
```

Ο δεύτερος κανόνας αποτελεί μια κρίσιμη δικλίδα ασφαλείας, διασφαλίζοντας ότι ο συνολικός πληθυσμός δεν μπορεί σε καμία περίπτωση να υπερβεί τη φυσική ικανότητα στέγασης που προσφέρουν οι υποδομές (Houses).

Φάση 4η: Συνθήκες Τερματισμού και Ενίσχυση Εκπαίδευσης AI

Στο τελικό στάδιο της μεθόδου, ο δείκτης των γύρων αυξάνεται (currentTurn++). Εάν ο γύρος υπερβεί το μέγιστο επιτρεπτό όριο (currentTurn > maxTurns), η προσομοίωση ολοκληρώνεται ομαλά.

```
if (currentTurn > maxTurns)
{
    GameOver();
}
else if (money < 0)
{
    // Ποινή: Μηδενισμός σκορ
    population = 0;
    GameOver();
}
```

Είναι η αντιμετώπιση της χρεοκοπίας παρουσιάζει ένα ενδιαφέρον (όταν το money < 0). Σε αυτή την περίπτωση, η συνάρτηση δεν καλεί απλώς τη μέθοδο GameOver(), αλλά και μηδενίζει ρητά τον πληθυσμό (population = 0) πριν τον τερματισμό. Αυτή η επιλογή στοχεύει στο στάδιο εκπαίδευσης (Reward Shaping) του ML-Agent: αφού το τελικό σκορ του πράκτορα ισούται με τον πληθυσμό, ο μηδενισμός του διασφαλίζει ότι το Νευρωνικό Δίκτυο λαμβάνει ανταμοιβή 0, βοηθώντας το να μάθει να αποφεύγει την επικίνδυνη οικονομική διαχείριση.

Βελτιστοποίηση Απόδοσης για Επιτάχυνση Εκπαίδευσης (ML Rendering Guard)

Μία από τις μεγαλύτερες προκλήσεις κατά την εκπαίδευση αλγορίθμων Ενισχυτικής Μάθησης είναι η συμφόρηση πόρων από τα υποσυστήματα διεπαφής (UI rendering). Όταν το περιβάλλον τρέχει σε επιταχυμένο χρόνο για τη συλλογή χιλιάδων εμπειριών (Time.timeScale = 20f), η συνεχής δημιουργία αναδυόμενων κειμένων (Floating Texts) μπορεί να εξαντλήσει τη διαθέσιμη μνήμη του συστήματος (Memory Leak / CPU Overload).
Για την επίλυση αυτού του τεχνικού προβλήματος, ενσωματώθηκε ένας μηχανισμός προστασίας (Guard Clause) στη συνάρτηση ShowFloatingText(Vector3 position, string message, Color color):

```
public void ShowFloatingText(Vector3 position, string message, Color color)
{
    if (Time.timeScale > 5f) return;

    if (popupTextPrefab != null)
    {
        // Δημιουργούμε το αντικείμενο στην ακριβή θέση του tile
        GameObject go = Instantiate(popupTextPrefab, position, Quaternion.identity);

        FloatingText ft = go.GetComponent<FloatingText>();

        if (ft != null)
        {
            // Στέλνουμε το κείμενο και το χρώμα στο Setup
            ft.Setup(message, color);
        }
    }
}
```

Αυτός ο κώδικας λειτουργεί ως θερμοστάτης απόδοσης: αναγνωρίζει την κατάσταση προσομοίωσης υψηλής ταχύτητας (όπου ο χρόνος τρέχει πέντε φορές πιο γρήγορα) και ακυρώνει την κλήση της συνάρτησης Instantiate για τα οπτικά εφέ. Με αυτόν τον τρόπο, η εκπαίδευση του μοντέλου σε Python παραμένει εξαιρετικά γρήγορη, επιτρέποντας την εκτέλεση χιλιάδων βημάτων ανά δευτερόλεπτο χωρίς να επηρεάζεται η σταθερότητα της μηχανής γραφικών.

4.4.4 Αρχιτεκτονική του Πράκτορα Μηχανικής Μάθησης (CityBuilderAgent.cs)

Η κλάση CityBuilderAgent αποτελεί τον κεντρικό πυρήνα ελέγχου της Τεχνητής Νοημοσύνης του συστήματος. Κληρονομεί απευθείας από την εξειδικευμένη κλάση Agent της βιβλιοθήκης Unity ML-Agents, λειτουργώντας ως η γέφυρα επικοινωνίας (μέσω πρωτοκόλλων RPC) μεταξύ του περιβάλλοντος προσομοίωσης της Unity (C#) και του αλγοριθμικού backend που εκτελείται σε Python (PyTorch).

Η αρχιτεκτονική του πράκτορα σχεδιάστηκε με μια ευέλικτη διττή λειτουργία, η οποία ελέγχεται από τη μεταβλητή trainingMode. Στη λειτουργία εκπαίδευσης, ο πράκτορας αναλαμβάνει τον πλήρη έλεγχο του συστήματος, ενώ στη λειτουργία παιχνιδιού λειτουργεί ως παθητικός σύμβουλος (Assistant Mode).

Παρακάτω αναλύονται οι τρεις κρίσιμες φάσεις του κύκλου ζωής του πράκτορα: ο χρονισμός, η αντίληψη (Observations) και η δράση (Actions).

1. Χρονισμός και Αποφυγή Υπερφόρτωσης (Frame Skipping)

Σε παιχνίδια γύρων, η συνεχής λήψη αποφάσεων σε κάθε καρέ (frame) είναι υπολογιστικά περιττή και μπορεί να προκαλέσει συμφόρηση στο δίκτυο επικοινωνίας με την Python (socket throttling). Για την αποφυγή αυτού του φαινομένου, ο χρονισμός των

αποφάσεων ενσωματώθηκε στη συνάρτηση σταθερού ρυθμού FixedUpdate():

```
void FixedUpdate()
{
    if (trainingMode)
    {
        frameCounter++;
        if (frameCounter >= 5)
        {
            RequestDecision();
            frameCounter = 0;
        }
    }
}
```

Η τεχνική αυτή, γνωστή στη βιβλιογραφία της Μηχανικής Μάθησης ως **Frame Skipping** (Παράλειψη Καρέ), εξασφαλίζει ότι το Νευρωνικό Δίκτυο καλείται να παράγει μια νέα απόφαση (RequestDecision()) μόνο μία φορά ανά 5 σταθερά καρέ, μειώνοντας δραματικά το υπολογιστικό φορτίο (CPU/GPU overhead) κατά τη φάση της εκπαίδευσης.

2. Χώρος Παρατηρήσεων (Observation Space): Η Αντίληψη του Περιβάλλοντος

Για να μπορέσει ο αλγόριθμος να λάβει ορθολογικές αποφάσεις, πρέπει το τρισδιάστατο περιβάλλον και οι οικονομικοί δείκτες να μετατραπούν σε ένα μονοδιάστατο διάνυσμα χαρακτηριστικών (1D Feature Vector ή Tensor). Αυτό επιτυγχάνεται μέσω της υπερφορτωμένης (overridden) συνάρτησης CollectObservations(VectorSensor sensor).
Το διάνυσμα εισόδου (Input Layer) του Νευρωνικού Δικτύου δομείται ιεραρχικά:

- **Μακροοικονομικά Δεδομένα:** Προστίθενται 5 ακέραιες τιμές που αντιπροσωπεύουν τους παγκόσμιους πόρους της πόλης (χρήματα, φαγητό, νερό, πληθυσμός, ευημερία).
- **Χωρικά Δεδομένα (Grid Flattening):** Στη συνέχεια, ο αλγόριθμος διατρέχει ολόκληρο τον δισδιάστατο πίνακα gridArray. Για κάθε κελί, πραγματοποιείται ρητή μετατροπή (explicit cast) των απαριθμητών (enums) σε ακέραιους αριθμούς και εισάγονται δύο τιμές στο διάνυσμα: (int)cell.currentBuilding και (int)cell.tileType.

Εάν το πλέγμα έχει διαστάσεις W * H, το τελικό μέγεθος του διανύσματος παρατηρήσεων που αποστέλλεται στο ML-Agents ισούται με:

$$Size = 5 + (W * H * 2)$$

Αυτή η προσέγγιση εξασφαλίζει ότι το AI διαθέτει πλήρη γνώση του περιβάλλοντος.

3. Χώρος Ενεργειών (Action Space): Εκτέλεση και Έλεγχος Σφαλμάτων

Όταν το Νευρωνικό Δίκτυο επιστρέψει την πρόβλεψή του, εκτελείται η μέθοδος OnActionReceived(ActionBuffers actions). Ο πράκτορας έχει σχεδιαστεί να εξάγει 4 διακριτές ενέργειες (Discrete Actions):

1. actionType: Καθορίζει αν το AI θέλει να χτίσει (0) ή να τερματίσει τον γύρο (1).
2. x: Η συντεταγμένη X του επιθυμητού κελιού.
3. z: Η συντεταγμένη Z του επιθυμητού κελιού.
4. buildTypeIndex: Ο τύπος του κτιρίου προς κατασκευή.

Η αρχιτεκτονική περιλαμβάνει κρίσιμες δικλίδες ασφαλείας (fail-safes) για την αποφυγή τερματισμού της εκπαίδευσης λόγω άπειρων βρόχων (infinite loops). Εάν το AI επιλέξει να χτίσει (actionType == 0), καλείται η μέθοδος TryBuildFromAI(x, z, buildTypeIndex).

```
if (trainingMode)
{
    if (actionType == 0) // ΕΠΙΛΟΓΗ 0: ΤΟ AI ΘΕΛΕΙ ΝΑ ΧΤΙΣΕΙ
    {
        bool success = gridManager.TryBuildFromAI(x, z, buildTypeIndex);

        if (!success)
        {
            // Ποινή MONO για αποτυχημένο κλικ (π.χ. πήγε να χτίσει χωρίς λεφτά)
            AddReward(-0.01f);

            // Αν δεν έχει λεφτά, τον αναγκάζουμε να αλλάξει γύρο για να μην κολλήσει
            // προσπαθώντας ασταμάτητα να χτίσει στο ίδιο frame
            gridManager.EndTurn();
        }
```

Αν η κίνηση είναι παράνομη (π.χ. προσπάθεια δόμησης στο νερό ή χωρίς επαρκή κεφάλαια), το σύστημα δεν απορρίπτει απλώς την κίνηση, αλλά τιμωρεί τον πράκτορα με μικρή αρνητική ενίσχυση (AddReward(-0.01f)) και **επιβάλλει άμεσα αλλαγή γύρου** (gridManager.EndTurn()). Αυτός ο μηχανισμός αναγκάζει τον πράκτορα να προχωρήσει τη χρονική ροή, αποτρέποντάς τον από το να "κολλήσει" προσπαθώντας ασταμάτητα να χτίσει το ίδιο κτίριο στο ίδιο καρέ χωρίς τους απαραίτητους πόρους.

4.4.5 Σύστημα Χωρικής Ανίχνευσης και Αλληλεπίδρασης Χρήστη (Raycasting & Input Handling)

Ενώ ο πράκτορας Τεχνητής Νοημοσύνης αλληλεπιδρά με το περιβάλλον στέλνοντας απευθείας συντεταγμένες (x, z) στον κεντρικό ελεγκτή, η αλληλεπίδραση του ανθρώπινου χρήστη απαιτεί τη μαθηματική μετάφραση της δισδιάστατης οθόνης (Screen Space) στον τρισδιάστατο κόσμο (World Space). Αυτή η διαδικασία υλοποιείται μέσω της τεχνικής της ριψοβολίας (Raycasting).

Σε κάθε καρέ της μηχανής (εντός της μεθόδου Update()), όταν ο παίκτης πραγματοποιεί αριστερό κλικ, εκτελείται μια αυστηρή σειρά ελέγχων:

1. **Προστασία Διεπαφής (UI Raycast Block):** Το πρώτο και σημαντικότερο βήμα είναι να διασφαλιστεί ότι το κλικ του χρήστη προοριζόταν για τον 3D κόσμο και όχι για κάποιο κουμπί της διεπαφής (όπως το μενού κατασκευής). Γι' αυτό χρησιμοποιείται η μέθοδος του συστήματος γεγονότων της Unity: if (EventSystem.current.IsPointerOverGameObject()) return; Αυτή η συνθήκη αποτρέπει τα "διαπεραστικά κλικ" (click-through bugs), όπου ο παίκτης, προσπαθώντας να επιλέξει ένα κτίριο από το μενού, έχτιζε κατά λάθος στο έδαφος πίσω από το UI.
2. **Μαθηματική Προβολή Ακτίνας (Screen-to-World Projection):** Αν το κλικ είναι έγκυρο, η κάμερα δημιουργεί ένα μαθηματικό διάνυσμα (Ray) που ξεκινά από τον φακό (Camera.main) και περνά μέσα από τις συντεταγμένες των pixel του ποντικιού (Input.mousePosition): Ray ray = Camera.main.ScreenPointToRay(Input.mousePosition);
3. **Ανίχνευση Σύγκρουσης (Physics Collision):** Η μηχανή φυσικής της Unity (Physics Engine) αναλαμβάνει να υπολογίσει αν αυτή η μαθηματική ακτίνα τέμνει κάποιο γεωμετρικό αντικείμενο στον κόσμο:

```
if (Physics.Raycast(ray, out hit))
{
    TileCell cell = hit.collider.GetComponent<TileCell>();

    if (cell != null)
    {
        ghostObject.SetActive(true);
        // Τοποθέτηση στο κέντρο του tile + ύψος 0.5 για να μην εμφανίζεται μεσα στο tile
        ghostObject.transform.position = cell.transform.position + new Vector3(0, 0.5f, 0);
        // Εφαρμογή περιστροφής
        ghostObject.transform.rotation = Quaternion.Euler(0, buildRotation, 0);
    }
}
else
{
    ghostObject.SetActive(false);
}
```

4.5 Διαδικασία Εκπαίδευσης και Σχεδιασμός Ανταμοιβών (Reward Shaping)

Η επιτυχής εκπαίδευση ενός πράκτορα Ενισχυτικής Μάθησης (Reinforcement Learning) εξαρτάται απόλυτα από τη σωστή διαμόρφωση της συνάρτησης ανταμοιβής (Reward Function). Η διαδικασία αυτή, γνωστή ως **Reward Shaping**, αποτελεί μια κρίσιμη πρόκληση, καθώς καθορίζει τη στρατηγική που θα αναπτύξει το Νευρωνικό Δίκτυο για την επίλυση του προβλήματος. Στο 3D City-Builder, η αρχιτεκτονική των ανταμοιβών σχεδιάστηκε προσεκτικά για να αποτρέψει εκφυλισμένες συμπεριφορές και να προωθήσει τη μακροπρόθεσμη βιωσιμότητα.

4.5.1 Φιλοσοφία Σχεδιασμού και Αποτροπή Εκμετάλλευσης (Reward Hacking)

Σε περιβάλλοντα προσομοίωσης διαχείρισης πόρων, η άμεση και συνεχής επιβράβευση (Dense Rewards) συχνά οδηγεί σε απρόβλεπτες συνέπειες. Εάν το σύστημα παρείχε θετική ανταμοιβή κάθε φορά που ο πράκτορας κατασκεύαζε ένα κτίριο, το Νευρωνικό Δίκτυο ενδέχεται να ανέπτυσσε υποβέλτιστες πολιτικές (sub-optimal policies), γνωστές ως **Reward Hacking**. Για παράδειγμα, θα μπορούσε να μάθει να χτίζει και να κατεδαφίζει συνεχώς το ίδιο φθηνό κτίριο (όπως το House) αποκλειστικά για τη συγκέντρωση πόντων, αγνοώντας πλήρως τη διαχείριση της τροφής ή του νερού.

Για την αποφυγή αυτού του φαινομένου, η παρούσα υλοποίηση βασίζεται σε ένα υβριδικό μοντέλο:

1. **Μικρή Αρνητική Ενίσχυση (Penalty)** για την αποτροπή παράνομων ή καταχρηστικών ενεργειών κατά τη διάρκεια του γύρου.
2. **Αραιή Τελική Ανταμοιβή (Sparse Terminal Reward)** στο τέλος του επεισοδίου, η οποία αξιολογεί τη συνολική στρατηγική επιβίωσης.

3.

4.5.2 Αρνητική Ενίσχυση και «Διάσωση Βρόχου» (Loop Rescue Mechanism)

Κατά τα αρχικά στάδια της εκπαίδευσης (Exploration Phase), ο πράκτορας δοκιμάζει τυχαίες ενέργειες. Αυτό μεταφράζεται σε χιλιάδες άκυρες προσπάθειες δόμησης, όπως η τοποθέτηση κτιρίων στο νερό ή η προσπάθεια κατασκευής χωρίς επαρκές κεφάλαιο.
Η αντιμετώπιση αυτών των σφαλμάτων πραγματοποιείται στη συνάρτηση OnActionReceived του CityBuilderAgent. Εάν η μέθοδος επικύρωσης TryBuildFromAI επιστρέψει false, το σύστημα εκτελεί τον παρακάτω κώδικα:

```
if (actionType == 0) // ΧΤΙΣΙΜΟ
{
    bool success = gridManager.TryBuildFromAI(x, z, buildTypeIndex);
    if (!success)
    {
        AddReward(-0.01f);
        gridManager.EndTurn();
    }
}
else if (actionType == 1) // END TURN
{
    gridManager.EndTurn();
}
```

Η ανάθεση της ποινής AddReward(-0.01f) διδάσκει στο μοντέλο να αποφεύγει τις σπατάλες ενεργειών. Ωστόσο, η πιο κρίσιμη αρχιτεκτονική επιλογή είναι η ταυτόχρονη, αναγκαστική κλήση της gridManager.EndTurn(). Αυτή η εντολή λειτουργεί ως ένας **Μηχανισμός Διάσωσης Βρόχου (Loop Rescue)**. Χωρίς αυτήν, εάν ο πράκτορας δεν διέθετε χρήματα, θα εγκλωβιζόταν σε μια άπειρη λούπα: θα επιχειρούσε ξανά και ξανά να χτίσει στο ίδιο καρέ (frame) της μηχανής, προκαλώντας υπερφόρτωση μνήμης και κατάρρευση (crash) του Python backend. Οδηγώντας τον βίαια στον επόμενο γύρο, το σύστημα διασφαλίζει την απρόσκοπτη συνέχιση της εκπαίδευσης.

4.5.3 Αραιή Ανταμοιβή και Μακροοικονομική Συσχέτιση

Ο απώτερος στόχος του πράκτορα είναι η δημιουργία μιας ακμάζουσας πόλης. Για να αξιολογηθεί αυτό, το σύστημα αποδίδει βαθμολογία μόνο τη στιγμή του τερματισμού (IsGameOver()). Η τελική ανταμοιβή δεν είναι ένας αυθαίρετος αριθμός, αλλά ταυτίζεται με τον τελικό πληθυσμό:

```
if (gridManager.IsGameOver())
{
    SetReward(gridManager.population);
    EndEpisode();
}
```

Η επιλογή του πληθυσμού (population) ως συνάρτηση στόχου αποτελεί μια εξαιρετικά κομψή λύση Reward Shaping. Για να επιτύχει υψηλό σκορ (π.χ. 100), το AI πρέπει να κατανοήσει βαθιά τις εξαρτήσεις του συστήματος: τα Houses αυξάνουν τη χωρητικότητα, αλλά καταναλώνουν νερό και φαγητό. Αν το AI χτίσει μόνο Houses, οι κάτοικοι θα

λιμοκτονήσουν και ο πληθυσμός θα καταρρεύσει (λόγω της ποινής population -= 5 στην EndTurn).

Εξίσου σημαντική είναι η διασύνδεση με τη χρεοκοπία. Όπως αναλύθηκε στο Κεφάλαιο 4.4.3, εάν τα χρήματα γίνουν αρνητικά (money < 0), ο GridManager μηδενίζει ρητά τον πληθυσμό (population = 0) πριν τερματίσει το παιχνίδι. Ως άμεσο αποτέλεσμα, η SetReward διαβάζει την τιμή 0, τιμωρώντας σιωπηρά αλλά απόλυτα τη χρεοκοπία και αναγκάζοντας το μοντέλο να εξισορροπεί τα έξοδά του.

4.5.4 Διαχείριση Σκηνής και Βελτιστοποίηση Εκπαίδευσης (Episode Reset)

Κατά τη διάρκεια του training, ένα επεισόδιο διαρκεί λίγα δευτερόλεπτα και επαναλαμβάνεται χιλιάδες φορές. Η γρήγορη και αποδοτική εκκαθάριση της μνήμης είναι επιβεβλημένη. Όταν καλείται η OnEpisodeBegin, το σύστημα αναθέτει την αναδόμηση στη μέθοδο RestartGameForML() του GridManager.

```
0 references
public override void OnEpisodeBegin()
{
    if (trainingMode)
    {
        gridManager.RestartGameForML();
    }
}
```

Δύο ενέργειες καθορίζουν την επιτυχία αυτής της φάσης:

1. **Επιτάχυνση Χρόνου:** Ο χρόνος της μηχανής πολλαπλασιάζεται (Time.timeScale = 20f), επιτρέποντας στο περιβάλλον να παράγει χιλιάδες παρατηρήσεις (observations) ανά λεπτό χωρίς καθυστερήσεις γραφικών.

```
1 reference
public void RestartGameForML()
{
    Time.timeScale = 20f;

    money = 100;
    population = 0;
    food = 0;
    water = 0;
    welfare = 50;
    currentTurn = 1;
    isGameOver = false;

    // Διατρέχουμε όλα τα παιδιά (tiles) που έχει το GridManager και τα διαγράφουμε
    foreach (Transform child in transform)
    {
        Destroy(child.gameObject);
    }

    // Τώρα φτιάχνουμε τον νέο, καθαρό χάρτη
    GenerateGrid();

    UpdateUI();

    if (winTextObject != null) winTextObject.SetActive(false);
    if (loseTextObject != null) loseTextObject.SetActive(false);
}
```

2. **Αποτροπή Διαρροής Μνήμης (Memory Leak Guard):** Πριν παραχθεί το νέο πλέγμα, το σύστημα διατρέχει δυναμικά το δέντρο ιεραρχίας του GridManager και καταστρέφει συστηματικά τα παλιά κελιά (Destroy(child.gameObject)). Αυτή η ρητή

διαχείριση αποτρέπει τη συσσώρευση χιλιάδων "νεκρών" 3D μοντέλων στη μνήμη RAM, η οποία διαφορετικά θα προκαλούσε κατάρρευση (freeze) του Unity Editor.

3.

4.5.5 Η Ροή Εκτέλεσης της Εκπαίδευσης (Training Pipeline) και Επικοινωνία Συστημάτων

Η εκπαίδευση του πράκτορα δεν πραγματοποιείται εγγενώς εντός του κώδικα της C#, αλλά απαιτεί την ταυτόχρονη εκτέλεση ενός εξωτερικού περιβάλλοντος Python που φιλοξενεί τη βιβλιοθήκη PyTorch [8]. Η διαδικασία εκκινείται μέσω της διεπαφής γραμμής εντολών (CLI) με την εντολή mlagents-learn, στην οποία περνιέται ως όρισμα το αρχείο υπερπαραμέτρων (YAML config file).
Μόλις η εντολή εκτελεστεί, το σύστημα ML-Agents ανοίγει μια θύρα επικοινωνίας (socket) και αναμένει τη σύνδεση της μηχανής Unity. Όταν ο χρήστης πατήσει το πλήκτρο εκκίνησης (Play) στον Unity Editor, ξεκινά ο Κύκλος Εκπαίδευσης (Training Loop), ο οποίος ακολουθεί τα εξής αυστηρά βήματα ανά καρέ:

1. **Συλλογή (Collection):** Η μέθοδος CollectObservations διαβάζει τα δεδομένα του πλέγματος και τα παγκόσμια αποθέματα.
2. **Διαβίβαση (Transmission):** Τα δεδομένα σειριοποιούνται και αποστέλλονται ταχύτατα μέσω του πρωτοκόλλου gRPC στο backend της Python.
3. **Υπολογισμός (Computation):** Το Νευρωνικό Δίκτυο (PyTorch) εκτελεί την πολιτική του (Actor) και επιστρέφει μια δράση. Ταυτόχρονα, το δίκτυο αξιολόγησης (Critic) υπολογίζει το σφάλμα με βάση τις ανταμοιβές και ανανεώνει τα βάρη του (Weights).
4. **Εκτέλεση (Execution):** Η Unity λαμβάνει την εντολή στη μέθοδο OnActionReceived, μεταβάλλει τον τρισδιάστατο κόσμο και ο κύκλος επαναλαμβάνεται.

Για την αποτροπή ταλαντώσεων στην εκμάθηση (oscillation), ορίστηκε ένας σημαντικά μεγάλος συντελεστής απόσβεσης (Discount Factor) gamma = 0.99. Η μαθηματική αυτή επιλογή εξαναγκάζει τον πράκτορα να δίνει τεράστια βαρύτητα στη μελλοντική επιβίωση της πόλης έναντι της άμεσης ανταμοιβής, κάτι απολύτως αναγκαίο σε ένα παιχνίδι στρατηγικής που απαιτεί προγραμματισμό (planning) για την αποφυγή μελλοντικού λιμού.

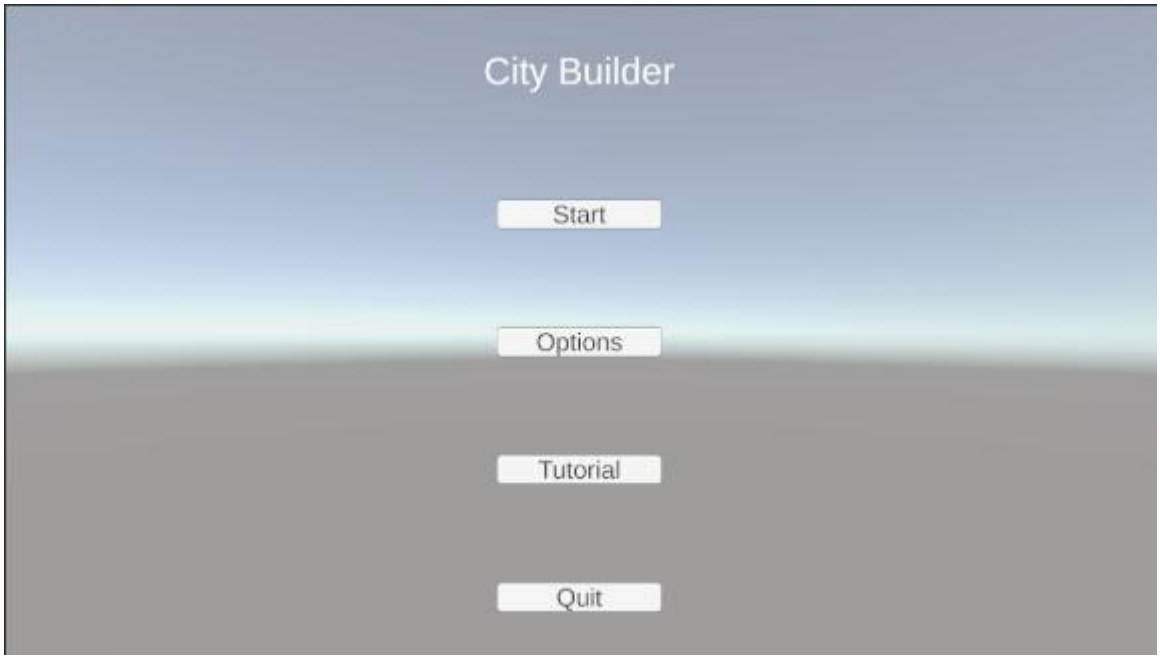
4.5.6 Παραγωγή Μοντέλου (Inference) και η Λειτουργία του "Advisor"

Η διαδικασία εκπαίδευσης τερματίζεται όταν το μοντέλο φτάσει το καθορισμένο όριο βημάτων (max_steps = 500000). Το τελικό προϊόν αυτής της εξαιρετικά απαιτητικής υπολογιστικής διαδικασίας είναι ένα εκπαιδευμένο μοντέλο Τεχνητής Νοημοσύνης σε μορφή .onnx (Open Neural Network Exchange). Αυτό το αρχείο περιέχει παγιωμένα τα βέλτιστα "βάρη" των νευρώνων που ανακάλυψε ο αλγόριθμος PPO.
Το αρχείο .onnx εισάγεται απευθείας στα assets του παιχνιδιού και συνδέεται με το Component του CityBuilderAgent. Κατά τη διάρκεια του κανονικού παιχνιδιού (όταν η μεταβλητή trainingMode είναι ψευδής), το Νευρωνικό Δίκτυο μεταβαίνει σε κατάσταση εξαγωγής συμπερασμάτων (Inference Mode).
Αυτή η αρχιτεκτονική αλλαγή είναι θεμελιώδης για το σχεδιασμό του παιχνιδιού. Όταν ο παίκτης αντιμετωπίζει δυσκολίες και πατάει το κουμπί του "Advisor" στο γραφικό περιβάλλον, το σύστημα δεν ανατρέχει σε κάποια παλαιότερη, προγραμματισμένη αρχιτεκτονική κανόνων (όπως ένα Utility AI ή ντετερμινιστικούς ευριστικούς αλγορίθμους). Αντίθετα, το πλήκτρο καλεί άμεσα το εκπαιδευμένο μοντέλο .onnx να εκτελέσει ένα "πέρασμα" (forward pass) στα τρέχοντα δεδομένα της πόλης. Το δίκτυο αξιολογεί αστραπιαία τον χώρο και επιστρέφει την υπόδειξή του, επιτυγχάνοντας την ομαλή και αξιόπιστη Συνεργασία Ανθρώπου-Πράκτορα στο τελικό λογισμικό.

ΚΕΦΑΛΑΙΟ 5: ΟΠΤΙΚΗ ΠΑΡΟΥΣΙΑΣΗ ΚΑΙ ΕΓΧΕΙΡΙΔΙΟ ΧΡΗΣΗΣ (USER MANUAL)

Για την πληρέστερη κατανόηση της αρχιτεκτονικής του συστήματος και της ροής του παιχνιδιού, στο παρόν κεφάλαιο παρατίθενται στιγμιότυπα από το τελικό παραδοτέο λογισμικό, συνοδευόμενα από ανάλυση της λειτουργικότητάς τους και των βασικών μηχανισμών αλληλεπίδρασης.

5.1 Αρχικό Μενού και Σύστημα Ομαλής Ένταξης (Onboarding)



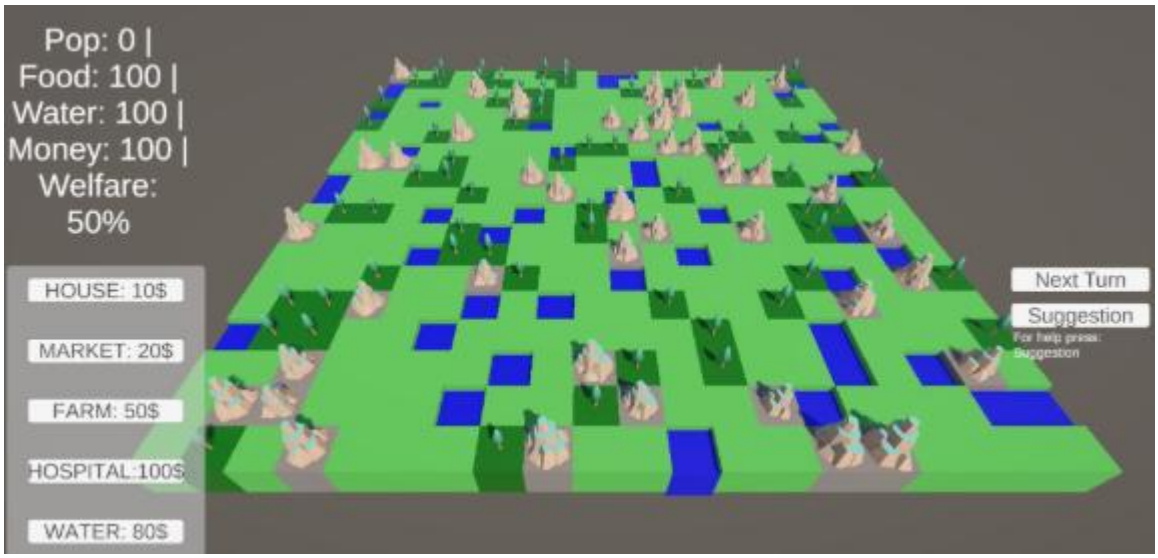
Εικόνα 5.1: *Η κεντρική οθόνη του παιχνιδιού (Main Menu).*

Εικόνα 5.2: *Οθόνη από το σύστημα εκμάθησης (Tutorial), η οποία εξηγεί τους οικονομικούς συσχετισμούς.*

Η Εικόνα 5.1 παρουσιάζει την αρχική διεπαφή του συστήματος, η οποία παρέχει τις βασικές επιλογές πλοήγησης: εκκίνηση της προσομοίωσης, παραμετροποίηση ήχου/οθόνης και πρόσβαση στο σύστημα εκμάθησης. Η μετάβαση στο Tutorial (Εικόνα 5.2) μεταφέρει τον παίκτη σε ένα ελεγχόμενο περιβάλλον, όπου η ροή του χρόνου έχει ανασταλεί. Μέσω μιας

σειράς διαδοχικών μηνυμάτων, εξηγείται με σαφήνεια ο χειρισμός της κάμερας, ο τρόπος δόμησης, καθώς και οι μακροοικονομικοί κανόνες του συστήματος, διασφαλίζοντας την αποφυγή γνωστικής υπερφόρτωσης (cognitive overload) του χρήστη πριν από την έναρξη του κανονικού παιχνιδιού.

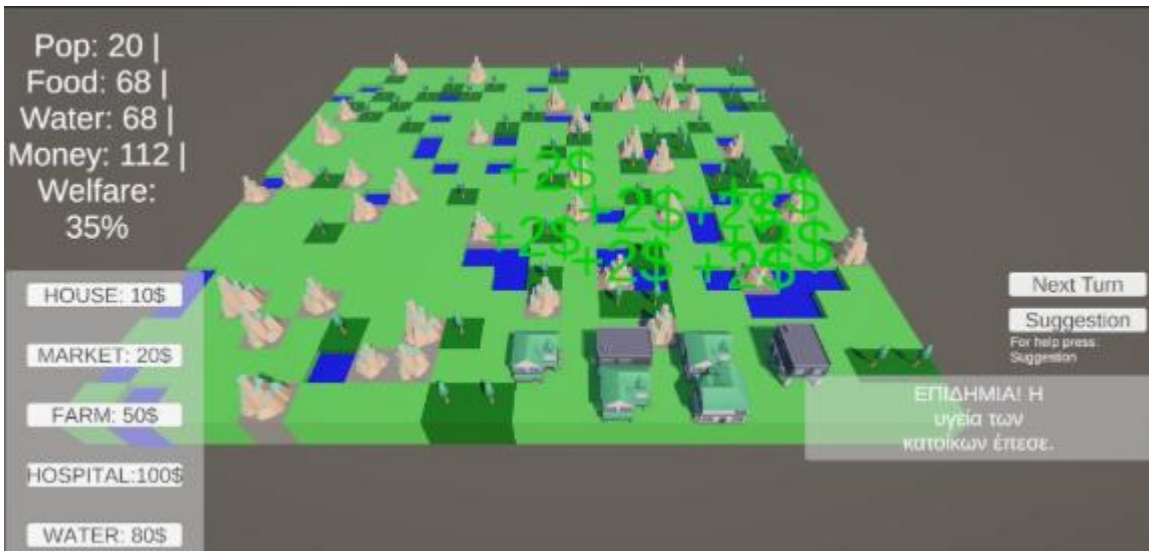
5.2 Διεπαφή Χρήστη (UI) και Μακροοικονομική Διαχείριση



Εικόνα 5.3: *Το περιβάλλον προσομοίωσης (Grid) και η κεντρική διεπαφή (HUD).*

Στην Εικόνα 5.3 αποτυπώνεται ο πυρήνας της προσομοίωσης. Στο αριστερό τμήμα δεσπόζει ο πίνακας ελέγχου των βασικών πόρων (Πληθυσμός, Τροφή, Νερό, Χρήματα, Ευημερία), οι οποίοι αποτελούν τις κύριες μεταβλητές κατάστασης του μαθηματικού μοντέλου. Ακριβώς από κάτω, παρατίθενται τα διαθέσιμα κτίρια προς κατασκευή συνοδευόμενα από το απαιτούμενο κεφάλαιο. Στη δεξιά πλευρά, ο παίκτης ελέγχει τη ροή του χρόνου μέσω του πλήκτρου «Next Turn», ενώ διαθέτει άμεση πρόσβαση στο σύστημα υποδείξεων «SMART VIEW».

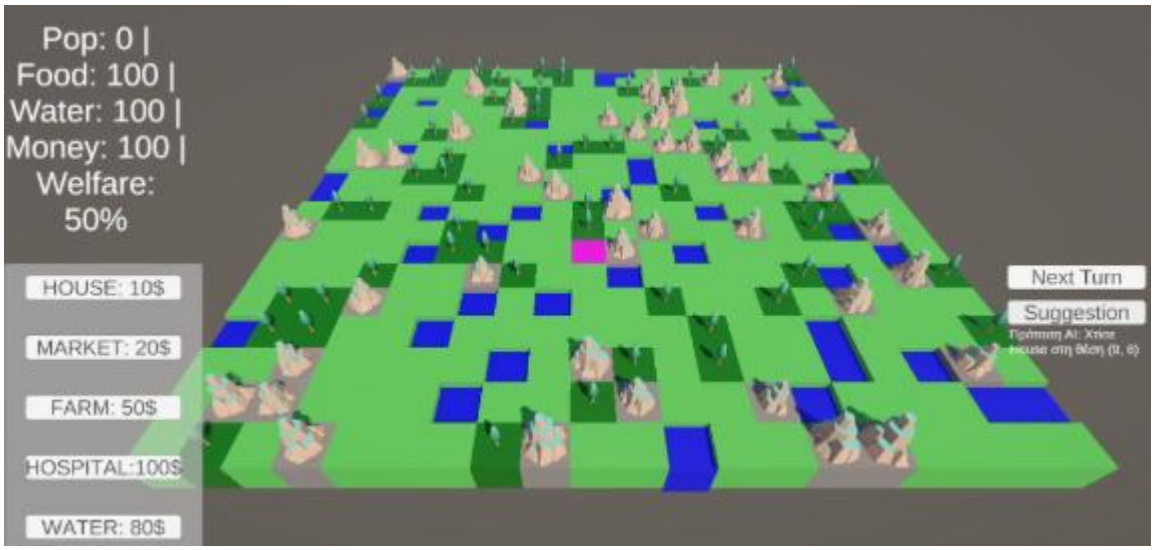
5.3 Οπτική Ανατροφοδότηση και Στοχαστικά Γεγονότα



Εικόνα 5.4: *Χωρική ανατροφοδότηση εσόδων (Floating Texts) και ενεργοποίηση τυχαίου αρνητικού γεγονότος (Επιδημία).*

Μια θεμελιώδης αρχή της αλληλεπίδρασης Ανθρώπου-Υπολογιστή (HCI) είναι η άμεση ανατροφοδότηση (feedback loops). Όπως διακρίνεται στην Εικόνα 5.4, κατά τη μετάβαση στον επόμενο γύρο, το σύστημα υπολογίζει την παραγωγή και εμφανίζει αιωρούμενα μηνύματα (π.χ. «+2$» με πράσινο χρώμα) πάνω από τις κατοικίες, πληροφορώντας τον παίκτη για την ακριβή οικονομική συνεισφορά της κάθε δομής. Παράλληλα, στην ίδια εικόνα παρουσιάζεται η στοχαστική φύση του παιχνιδιού: ένα δυναμικό γεγονός («ΕΠΙΔΗΜΙΑ») έχει μόλις ενεργοποιηθεί στο κάτω δεξί μέρος της οθόνης, προκαλώντας την άμεση πτώση του δείκτη ευημερίας (Welfare) στο 35%, αναγκάζοντας τον παίκτη να προσαρμόσει άμεσα τη στρατηγική του.

5.4 Λειτουργία του AI Advisor (Συνεργασία Ανθρώπου-Πράκτορα)



Εικόνα 5.5: *Ο εκπαιδευμένος πράκτορας (ML-Agent) υποδεικνύει χωρικά τη βέλτιστη ενέργεια στο πλέγμα.*

Η Εικόνα 5.5 αναδεικνύει την τεχνολογική καινοτομία του έργου. Ο χρήστης έχει ζητήσει τη συνδρομή της Τεχνητής Νοημοσύνης. Το εκπαιδευμένο Νευρωνικό Δίκτυο έχει ολοκληρώσει την ανάλυση του πλέγματος και έχει επιλέξει τη βέλτιστη ενέργεια δόμησης. Για να επικοινωνήσει την απόφασή του, το σύστημα χρωματίζει το προτεινόμενο κελί με έντονο χρώμα (magenta), καθοδηγώντας τον παίκτη με απόλυτη χωρική ακρίβεια. Ταυτόχρονα, η διεπαφή ανανεώνεται καταγράφοντας ρητά την εντολή («Πρόταση AI: Χτίσε House στη θέση (9, 6)»), επιτυγχάνοντας την απρόσκοπτη επικοινωνία στο πλαίσιο της Συνεργασίας Ανθρώπου-Πράκτορα (Human-Agent Teaming).

5.5 Τερματισμός Προσομοίωσης και Τελική Αξιολόγηση

Εικόνα 5.6: *Οθόνη τερματισμού της προσομοίωσης και προβολή του τελικού σκορ.*

Στην Εικόνα 5.6 παρουσιάζεται η τελική κατάσταση του παιχνιδιού κατά την ολοκλήρωση της προσομοίωσης. Με τη λήξη των διαθέσιμων γύρων, η λογική του συστήματος σταματά την εξέλιξη του κόσμου και προβάλλει κεντρικά το τελικό σκορ («Score: 57»). Όπως διακρίνεται στον πίνακα ελέγχου πάνω αριστερά, το τελικό σκορ ταυτίζεται απόλυτα με τον συνολικό πληθυσμό της πόλης. Στο συγκεκριμένο επιτυχές σενάριο, αποτυπώνεται μια εξαιρετική μακροοικονομική ισορροπία: τα αποθέματα νερού και τροφής είναι πλεονασματικά (143 και 197 αντίστοιχα), το κεφάλαιο είναι θετικό (189$), και ο δείκτης ευημερίας έχει αγγίξει το 100%. Αυτό το στιγμιότυπο επιβεβαιώνει την ορθή λειτουργία του μαθηματικού μοντέλου του παιχνιδιού και ολοκληρώνει τον κύκλο αλληλεπίδρασης του παίκτη με το ψηφιακό περιβάλλον.

ΚΕΦΑΛΑΙΟ 6 Συμπεράσματα

6.1 Πειραματική Διαδικασία και Υπερπαράμετροι Εκπαίδευσης

Η τελική αξιολόγηση του συστήματος Συνεργασίας Ανθρώπου-Πράκτορα (Human-Agent Teaming) πραγματοποιήθηκε μέσω μιας εκτενούς σειράς προσομοιώσεων εντός της μηχανής Unity, αξιοποιώντας το πακέτο ML-Agents. Στόχος του πειράματος ήταν να διαπιστωθεί εάν ο πράκτορας (CityBuilderAgent) μπορεί να μάθει αυτόνομα τις ισορροπίες της μικροοικονομίας του παιχνιδιού και να λειτουργήσει ως αποτελεσματικός σύμβουλος δόμησης.

Για τη συλλογή επαρκούς αριθμού εμπειριών χωρίς τεράστιο χρονικό κόστος, το περιβάλλον εκπαίδευσης ρυθμίστηκε να εκτελείται σε επιταχυμένο χρόνο. Μέσω της συνάρτησης RestartGameForML(), ο χρόνος της μηχανής πολλαπλασιάστηκε (Time.timeScale = 20f), ενώ ενσωματώθηκε μηχανισμός (rendering guard) που απενεργοποιούσε την οπτική παραγωγή κειμένων (floating texts) όταν η προσομοίωση έτρεχε σε υψηλές ταχύτητες.

Η εκπαίδευση βασίστηκε στον αλγόριθμο Proximal Policy Optimization (PPO). Το αρχείο ρυθμίσεων (YAML) παραμετροποιήθηκε ειδικά για τη διακριτή (turn-based) φύση του προβλήματος, ως εξής:

```yaml
behaviors:
  CityBuilder:
    trainer_type: ppo
    hyperparameters:
      batch_size: 128
      buffer_size: 2048
      learning_rate: 3.0e-4
      beta: 5.0e-4
      epsilon: 0.2
      lambd: 0.99
      num_epoch: 3
      learning_rate_schedule: linear
    network_settings:
      normalize: false
      hidden_units: 256
      num_layers: 3
    reward_signals:
      extrinsic:
        gamma: 0.99
        strength: 1.0
    max_steps: 500000
    time_horizon: 128
    summary_freq: 10000
```

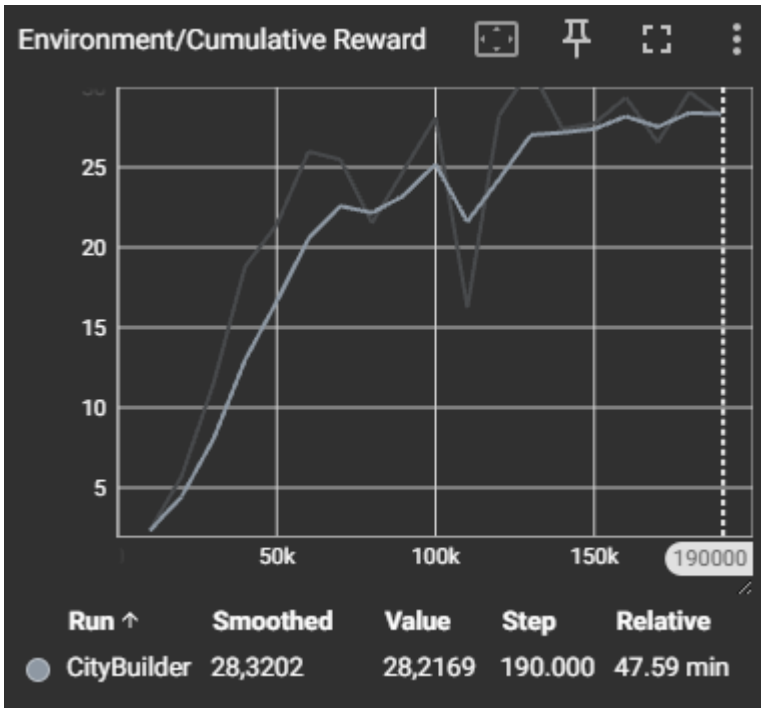
- **Batch Size (128) & Buffer Size (2048):** Μικρές τιμές, ιδανικές για συχνή ανανέωση βαρών σε περιβάλλοντα που εξελίσσονται με γύρους.
- **Learning Rate (3.0e -4):** Σταθερός ρυθμός μάθησης με γραμμική μείωση (linear schedule) για την αποτροπή ταλαντώσεων κατά τη σύγκλιση.
- **Max Steps (500.000):** Ο μέγιστος αριθμός υπολογιστικών βημάτων (δράσεων) που κρίθηκε επαρκής για την εύρεση της βέλτιστης πολιτικής σε ένα πλέγμα 10*10.

6.2 Ανάλυση Αποτελεσμάτων Εκπαίδευσης (TensorBoard)

Η παρακολούθηση της διαδικασίας μάθησης του Νευρωνικού Δικτύου πραγματοποιήθηκε μέσω του εργαλείου οπτικοποίησης TensorBoard. Η ανάλυση των παρακάτω μετρικών επιβεβαιώνει την επιτυχή εκπαίδευση του πράκτορα.

6.2.1 Εξέλιξη Συνολικής Ανταμοιβής (Cumulative Reward)

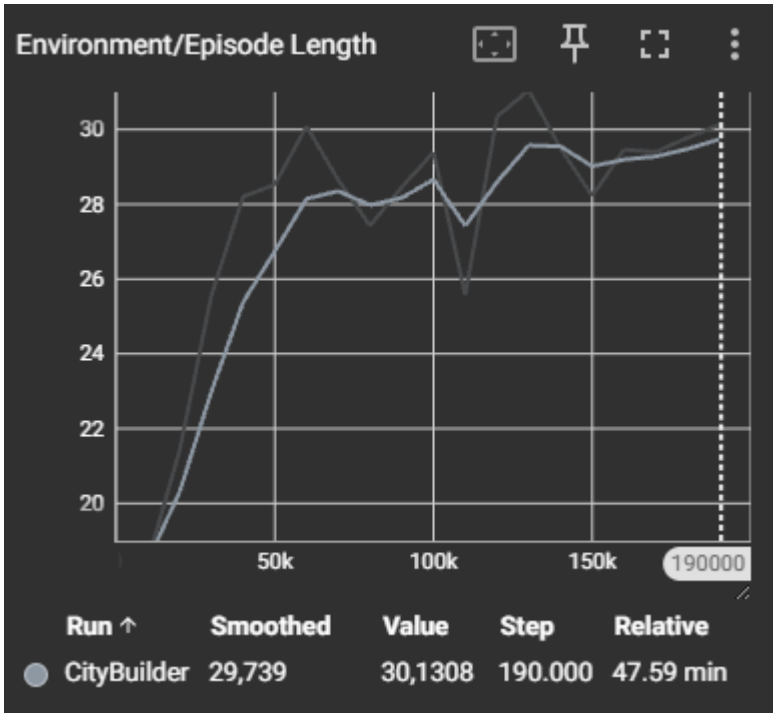
Το γράφημα της συνολικής ανταμοιβής αποτελεί τον πιο άμεσο δείκτη απόδοσης. Στα αρχικά βήματα της εκπαίδευσης (Φάση Εξερεύνησης - Exploration), το Νευρωνικό Δίκτυο λαμβάνει τυχαίες αποφάσεις. Παρατηρείται ότι το Cumulative Reward διατηρούνταν σε χαμηλά ή και αρνητικά επίπεδα. Αυτό δικαιολογείται από το γεγονός ότι ο πράκτορας δαπανούσε τα χρήματά του χωρίς σχεδιασμό, οδηγώντας την πόλη σε χρεοκοπία (η οποία μηδένιζε τεχνητά τον πληθυσμό και άρα το τελικό reward), ή επιχειρούσε παράνομες κινήσεις λαμβάνοντας αρνητική ενίσχυση (-0.01).



Με την πάροδο των βημάτων, η καμπύλη παρουσιάζει σταθερή ανοδική τάση και τελικά συγκλίνει. Η σύγκλιση αποδεικνύει ότι ο πράκτορας έμαθε τον θεμελιώδη κανόνα του συστήματος: η αύξηση του πληθυσμού προϋποθέτει την ταυτόχρονη κατασκευή κατοικιών (για αύξηση χωρητικότητας) και υποδομών υδροδότησης/σίτισης (WaterPlant, Farm) για την αποτροπή του λιμού.

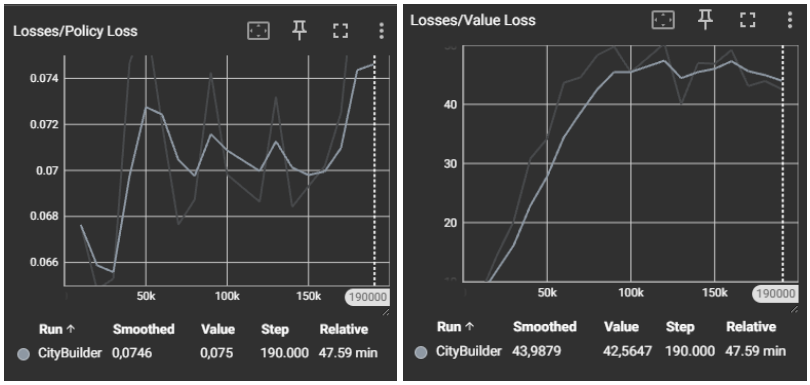
6.2.2 Διάρκεια Επεισοδίου (Episode Length)

Στο παρόν λογισμικό, το μέγιστο όριο ζωής ενός παιχνιδιού ορίστηκε στους 20 γύρους (maxTurns = 20). Στα πρώιμα στάδια της εκπαίδευσης, η διάρκεια των επεισοδίων ήταν σημαντικά μικρότερη από το μέγιστο όριο. Ο πράκτορας αυτοκτονούσε οικονομικά επιχειρώντας πανάκριβες κατασκευές χωρίς επαρκή έσοδα, τερματίζοντας το παιχνίδι πρόωρα λόγω της συνθήκης money < 0. Όπως φαίνεται στο γράφημα, η διάρκεια του επεισοδίου αυξήθηκε προοδευτικά και σταθεροποιήθηκε στην ανώτατη τιμή, επιβεβαιώνοντας ότι το μοντέλο έμαθε να ελέγχει τη μακροπρόθεσμη ρευστότητα των πόρων του.



6.2.3 Σφάλματα Πολιτικής και Αξίας (Policy Loss & Value Loss)

Τα συγκεκριμένα γραφήματα αντικατοπτρίζουν τη μαθηματική σταθερότητα του αλγορίθμου PPO. Το Value Loss αποτυπώνει την αρχική δυσκολία του δικτύου να προβλέψει σωστά την αξία της κάθε κατάστασης (Εξίσωση Bellman). Καθώς το AI ανακάλυπτε νέους τρόπους ανάπτυξης της πόλης, το σφάλμα αυτό παρουσίαζε διακυμάνσεις, μέχρι να αρχίσει να μειώνεται. Το Policy Loss, το οποίο υποδηλώνει το μέγεθος των αλλαγών στη στρατηγική του πράκτορα, διατηρήθηκε σε χαμηλά επίπεδα, αποδεικνύοντας ότι ο μηχανισμός "Clipping" λειτούργησε, αποτρέποντας την καταστροφική λήθη (Catastrophic Forgetting).



6.3 Προκλήσεις και Τεχνικές Επίλυσης

Η υλοποίηση του συστήματος συνοδεύτηκε από κρίσιμες τεχνικές προκλήσεις:

1. **Διαρροή Μνήμης (Memory Leaks):** Η συνεχής και ταχύτατη εκκίνηση νέων επεισοδίων άφηνε εκατοντάδες 3D αντικείμενα στη μνήμη (RAM), οδηγώντας σε κατάρρευση (crash) του προγράμματος. Το πρόβλημα λύθηκε με τη δημιουργία μιας βίαιης ρουτίνας εκκαθάρισης (Garbage Collection) στην OnEpisodeBegin, η οποία διατρέχει δυναμικά το δέντρο ιεραρχίας και καταστρέφει ρητά όλα τα παλαιά κελιά πριν δημιουργηθεί ο νέος χάρτης.

```
0 references
public override void OnEpisodeBegin()
{
    if (trainingMode)
    {
        gridManager.RestartGameForML();
    }
}
```

2. **Εγκλωβισμός σε Άπειρους Βρόχους (Infinite Loops):** Η απουσία χρόνου αναμονής (cooldowns) επέτρεπε στο AI να επιχειρεί να χτίσει επαναλαμβανόμενα χωρίς χρήματα μέσα στο ίδιο καρέ γραφικών. Η λύση (Loop Rescue Mechanism) περιελάμβανε την επιβολή μικρής ποινής και την αυτόματη, εξαναγκασμένη αλλαγή γύρου (gridManager.EndTurn()), διατηρώντας τη ροή του χρόνου.

```
if (actionType == 0) // ΧΤΙΣΙΜΟ
{
    bool success = gridManager.TryBuildFromAI(x, z, buildTypeIndex);
    if (!success)
    {
        AddReward(-0.01f);
        gridManager.EndTurn();
    }
}
```

3. **Οπτική Συμφόρηση (Visual Clutter):** Κατά την αλληλεπίδραση του χρήστη, τα αναδυόμενα κείμενα και τα χρωματισμένα κελιά υπερφόρτωναν την οθόνη. Εισήχθη το πρότυπο "Asynchronous Cleanup" μέσω Coroutines, όπου η διεπαφή καθαρίζεται αυτόματα μετά από 3 δευτερόλεπτα (Debouncing).

6.4 Μελλοντικές Επεκτάσεις

Η παρούσα πτυχιακή εργασία προσφέρει μια σταθερή αρχιτεκτονική (framework), η οποία επιδέχεται πολλαπλών επεκτάσεων:

- **Δυναμικό Οδικό Δίκτυο & Αλγόριθμοι Πλοήγησης:** Η εισαγωγή δρόμων και η προσομοίωση κίνησης αυτοκινήτων μέσω αλγορίθμων Pathfinding (π.χ. αλγόριθμος A*), όπου η κυκλοφοριακή συμφόρηση θα δρα ως αρνητικός παράγοντας στον δείκτη Ευημερίας.
- **Πολυπρακτορική Ενισχυτική Μάθηση (Multi-Agent RL):** Ενσωμάτωση δεύτερου πράκτορα Τεχνητής Νοημοσύνης στο ίδιο πλέγμα. Ο δεύτερος πράκτορας θα μπορούσε να δρα είτε ανταγωνιστικά (π.χ. παίζοντας τον ρόλο του "Δημάρχου Αντίπαλης Πόλης") είτε ανασταλτικά (προκαλώντας φυσικές καταστροφές).
- **Σύστημα Ζωνοποίησης (Zoning):** Μετάβαση από την άμεση κατασκευή κτιρίων στον μακροοικονομικό καθορισμό ζωνών (Οικιστικές, Εμπορικές, Βιομηχανικές) στα πρότυπα του *SimCity*, επιτρέποντας στον πληθυσμό να χτίζει μόνος του με βάση την αξία της γης (Land Value).

6.5 Τελικό Συμπέρασμα

Η παρούσα πτυχιακή εργασία καταδεικνύει επιτυχώς τον τρόπο με τον οποίο ο Αντικειμενοστραφής Προγραμματισμός (OOP) και ο Σχεδιασμός Παιχνιδιών (Game Design) μπορούν να διασυνδεθούν ομαλά με τεχνολογίες Βαθιάς Ενισχυτικής Μάθησης (Deep Reinforcement Learning). Αναπτύχθηκε ένα πλήρως λειτουργικό 3D περιβάλλον διαχείρισης πόρων με στοχαστικά γεγονότα και αυστηρούς μαθηματικούς κανόνες επιβίωσης.

Το πλέον καινοτόμο στοιχείο, ωστόσο, είναι η εφαρμογή της Τεχνητής Νοημοσύνης: Αντί το AI να λειτουργήσει παραδοσιακά ως αντίπαλος, εκπαιδεύτηκε ώστε να κατανοεί τον χώρο και να λειτουργεί ως έξυπνος σύμβουλος, εισάγοντας τον παίκτη στο πρότυπο της Συνεργασίας Ανθρώπου-Πράκτορα (HAT). Η επιτυχής σύγκλιση του αλγορίθμου PPO και η εύρυθμη λειτουργία της διεπαφής αποτελούν ισχυρή απόδειξη των δυνατοτήτων που προσφέρουν τα σύγχρονα εργαλεία (Unity ML-Agents) στην ανάπτυξη ευφυών διαδραστικών συστημάτων.

ΒΙΒΛΙΟΓΡΑΦΙΑ

[1] Noor Shaker, Julian Togelius, and Mark J. Nelson (2016). *Procedural Content Generation in Games: A Textbook and an Overview of Current Research*
Ανάκτηση από: https://www.pcgbook.com/

[2] Yannakakis, Georgios N., and Julian Togelius. *Artificial Intelligence and Games*. Springer, 2025.
Ανάκτηση από: https://gameaibook.org/

[3] Unity Technologies. (2024). *Data-Oriented Technology Stack (DOTS) & Entity Component System (ECS).*
Ανάκτηση από: https://unity.com/dots

[4] Nystrom, R. (2014). *Game programming patterns*. Genever Benning.
Ανάκτηση από: https://gameprogrammingpatterns.com/contents.html

[5] Colledanchise, M., & Ögren, P. (2017). *Behavior Trees in Robotics and AI: An Introduction.*
Ανάκτηση από: https://arxiv.org/pdf/1709.00084

[6] Sutton, R. S., & Barto, A. G. (2018). *Reinforcement Learning: An Introduction* (2nd Edition). MIT press.
Ανάκτηση από: http://incompleteideas.net/book/the-book-2nd.html

[7] Schulman, J., Wolski, F., Dhariwal, P., Radford, A., & Klimov, O. (2017). *Proximal policy optimization algorithms.*
Ανάκτηση από: https://arxiv.org/pdf/1707.06347

[8] Juliani, A., Berges, V. P., Teng, E., Cohen, A., Harper, J., Elion, C., ... & Lange, D. (2018). *Unity: A general platform for intelligent agents.*
Ανάκτηση από: https://arxiv.org/pdf/1809.02627