ΠΑΝΕΠΙΣΤΗΜΙΟ ΔΥΤΙΚΗΣ ΑΤΤΙΚΗΣ
ΣΧΟΛΗ ΜΗΧΑΝΙΚΩΝ
ΤΜΗΜΑ ΗΛΕΚΤΡΟΛΟΓΩΝ & ΗΛΕΚΤΡΟΝΙΚΩΝ ΜΗΧΑΝΙΚΩΝ

Διπλωματική Εργασία

«Ανάπτυξη μοντέλου απλοποιημένου επιτραπέζιου παιχνιδιού πολλών χρηστών Dominion σε Python και υλοποίηση Αλγορίθμου Τεχνητής Νοημοσύνης αυτο-εκμάθησης ενός Τεχνητού Παίκτη»

Φοιτητής: Δουλάμης Κωνσταντίνος
Α.Μ.: 44040

Επιβλέπων Καθηγητής

Μετάφας Δημήτριος, Επίκ. Καθηγητής

ΑΘΗΝΑ-ΑΙΓΑΛΕΩ, ΟΚΤΩΒΡΙΟΣ 2024

UNIVERSITY OF WEST ATTICA
FACULTY OF ENGINEERING
DEPARTMENT OF ELECTRICAL & ELECTRONICS ENGINEERING

Diploma Thesis

«Development of a simplified version of the multi-player board-game Dominion in Python and implementation of an unsupervised machine learning algorithm for a player AI agent»

Student: Doulamis Konstantinos
Registration Number: 44040

Supervisor
Metafas Dimitrios, Assist. Professor

ATHENS-EGALEO, OCTOBER 2024

Η Διπλωματική Εργασία έγινε αποδεκτή και βαθμολογήθηκε από την εξής τριμελή επιτροπή:

ΜΕΤΑΦΑΣ ΔΗΜΤΡΙΟΣ, ΕΠΙΚ. ΚΑΘΗΓΗΤΗΣ	ΡΑΓΚΟΥΣΗ ΜΑΡΙΑ, ΚΑΘΗΓΗΤΡΙΑ	ΚΑΧΡΗΣ ΧΡΙΣΤΟΦΟΡΟΣ, ΕΠΙΚ. ΚΑΘΗΓΗΤΗΣ
ΔΗΜΗΤΡΙΟΣ ΜΕΤΑΦΑΣ Digitally signed by ΔΗΜΗΤΡΙΟΣ ΜΕΤΑΦΑΣ Date: 2024.10.20 19:45:33 +03'00' (Υπογραφή)	MARIA RAGKOUSI Digitally signed by MARIA RAGKOUSI Date: 2024.10.17 09:59:00 +03'00' (Υπογραφή)	ΧΡΙΣΤΟΦΟΡΟΣ ΚΑΧΡΗΣ Digitally signed by ΧΡΙΣΤΟΦΟΡΟΣ ΚΑΧΡΗΣ Date: 2024.10.17 11:12:05 +03'00' (Υπογραφή)

ΔΗΛΩΣΗ ΣΥΓΓΡΑΦΕΑ ΔΙΠΛΩΜΑΤΙΚΗΣ ΕΡΓΑΣΙΑΣ

Ο κάτωθι υπογεγραμμένος ΔΟΥΛΑΜΗΣ ΚΩΝΣΤΑΝΤΙΝΟΣ του ΠΑΝΑΓΙΩΤΗ, με αριθμό μητρώου 44040 φοιτητής του Πανεπιστημίου Δυτικής Αττικής, της Σχολής ΜΗΧΑΝΙΚΩΝ, του Τμήματος ΗΛΕΚΤΡΟΛΟΓΩΝ ΚΑΙ ΗΛΕΚΤΡΟΝΙΚΩΝ ΜΗΧΑΝΙΚΩΝ,

δηλώνω υπεύθυνα ότι:

«Είμαι συγγραφέας αυτής της διπλωματικής εργασίας και ότι κάθε βοήθεια την οποία είχα για την προετοιμασία της είναι πλήρως αναγνωρισμένη και αναφέρεται στην εργασία. Επίσης, οι όποιες πηγές από τις οποίες έκανα χρήση δεδομένων, ιδεών ή λέξεων, είτε ακριβώς είτε παραφρασμένες, αναφέρονται στο σύνολό τους, με πλήρη αναφορά στους συγγραφείς, τον εκδοτικό οίκο ή το περιοδικό, συμπεριλαμβανομένων και των πηγών που ενδεχομένως χρησιμοποιήθηκαν από το διαδίκτυο. Επίσης, βεβαιώνω ότι αυτή η εργασία έχει συγγραφεί από μένα αποκλειστικά και αποτελεί προϊόν πνευματικής ιδιοκτησίας τόσο δικής μου, όσο και του Ιδρύματος.
Παράβαση της ανωτέρω ακαδημαϊκής μου ευθύνης αποτελεί ουσιώδη λόγο για την ανάκληση του διπλώματός μου.»

Ο Δηλών
ΔΟΥΛΑΜΗΣ ΚΩΝΣΤΑΝΤΙΝΟΣ

Ευχαριστίες

Θα ήθελα να ευχαριστήσω θερμά τον επιβλέποντα καθηγητή μου, κ. Δημήτριο Μετάφα, για την καθοδήγηση και την υποστήριξη του καθ' όλη τη διάρκεια της διπλωματικής μου εργασίας. Οι συμβουλές του ήταν καθοριστικές για την ολοκλήρωση της έρευνάς μου.

Επίσης, θα ήθελα να ευχαριστήσω την οικογένειά μου και τους φίλους μου για την υπομονή και την υποστήριξή τους καθ' όλη την διάρκεια των σπουδών μου.

Τέλος, εκφράζω την ευγνωμοσύνη μου προς όλους όσους συνέβαλαν με οποιονδήποτε τρόπο στην ολοκλήρωση αυτής της εργασίας.

Περίληψη

Η παρούσα διπλωματική εργασία επικεντρώνεται στην ανάπτυξη ενός ψηφιακού παιχνιδιού σε γλώσσα προγραμματισμού Python. Το παιχνίδι χρησιμοποιεί έναν πράκτορα τεχνητής νοημοσύνης (AI Agent) ο οποίος εκπαιδεύεται με αλγόριθμο μη επιβλεπόμενης μάθησης ώστε να παίζει το παιχνίδι με το βέλτιστο τρόπο και να νικά. Σκοπός της εργασίας είναι να αναδείξει τις δυνατότητες και τις εφαρμογές της μη επιβλεπόμενης μάθησης στα ψηφιακά παιχνίδια και να προσφέρει πρακτικά παραδείγματα για την υλοποίηση παρόμοιων έργων. Στο παιχνίδι που αναπτύχθηκε σε Python, οι παίκτες χρησιμοποιούν κάρτες δράσης, θησαυρού και νίκης. Ο πράκτορας AI χρησιμοποιεί τον αλγόριθμο K-means για την κατηγοριοποίηση των καταστάσεων του παιχνιδιού και ενισχυτική μάθηση (Reinforcement Learning) για να βελτιστοποιήσει τις ενέργειές του. Ο κώδικας περιλαμβάνει τη δημιουργία καρτών, τράπουλας και παικτών, καθώς και τις φάσεις του παιχνιδιού: δράση, αγορά, καθαρισμός και συναλλαγή. Ο πράκτορας AI, μέσω του αλγορίθμου DQN, εκπαιδεύεται («μαθαίνει») μέσω επαναληπτικής μνήμης και νευρωνικών δικτύων, χρησιμοποιώντας παραμέτρους όπως το learning rate, discount factor, exploration rate και batch size. Ο αλγόριθμος DQN χρησιμοποιεί νευρωνικά δίκτυα για να εκτιμήσει τις Q-τιμές και να βελτιστοποιήσει την απόδοση του πράκτορα AI σε περιβάλλοντα με μεγάλη διάσταση και πολυπλοκότητα. Η μεθοδολογία περιλαμβάνει τη χρήση replay memory και target networks για τη βελτίωση της απόδοσης του πράκτορα, ενώ γίνεται εκτενής αναφορά στις παραμέτρους του αλγορίθμου DQN. Οι φάσεις του παιχνιδιού περιλαμβάνουν την action phase, buy phase, cleanup phase, με στόχο την καλύτερη αξιοποίηση των καρτών από τους παίκτες. Η ανάλυση των επιδόσεων του πράκτορα AI γίνεται μέσω γραφήματος. Η εργασία κλείνει με τα Συμπεράσματα και προτάσεις για περαιτέρω έρευνα.

Λέξεις – κλειδιά

Ψηφιακά παιχνίδια, τεχνητός παίκτης, μη επιβλεπόμενη μάθηση, Python, τεχνητή νοημοσύνη, ενισχυτική μάθηση, νευρωνικά δίκτυα

Abstract

This thesis focuses on the development of a digital game in Python programming language. The game uses an AI Agent that is trained with an unsupervised learning algorithm to play the game in an optimal way and maximize the probability to win the game. The aim of the thesis is to highlight the potential and applications of unsupervised learning in games and to provide practical examples for the implementation of similar projects. In the present game, developed in Python, players use action, treasure and victory cards. The AI agent uses the K-means algorithm to categorize game states and employs reinforcement learning to optimize actions and decisions in order to win the game. The code includes card, deck, and player generation, as well as the game phases: action, purchase, cleanup and transaction. The AI agent uses the DQN algorithm to "learn" an optimal behavior, through iterative memory and neural networks, by trimming parameters such as learning rate, discount factor, exploration rate and batch size. The DQN algorithm uses neural networks to estimate Q-values and optimize the agent's performance in environments of high dimensionality and complexity. The methodology includes the use of replay memory and target networks to improve the AI agent performance, while the parameters of the DQN algorithm are extensively discussed. In the aim of making the best use of the cards by the players, the phases of the game include the action phase, buy phase and a cleanup phase. The analysis of the agent's performance is done through a graph. The thesis is completed by the Conclusions and proposed future research directions.

Keywords

Digital games, AI agent, unsupervised learning, Python programming language, artificial intelligence, reinforcement learning, neural networks

Περιεχόμενα

Κατάλογος πινάκων

ΕΙΣΑΓΩΓΗ

Η τεχνητή νοημοσύνη (AI) και οι αλγόριθμοι μηχανικής μάθησης έχουν επηρεάσει τον τρόπο με τον οποίο φτιάχνονται τα σύγχρονα παιχνίδια, προσφέροντας νέες δυνατότητες στην ανάπτυξη και στην εμπειρία των παικτών. Στην παρούσα διπλωματική εργασία, έγινε η ανάπτυξη μια απλοποιημένης μορφής του παιχνιδιού Dominion σε γλώσσα Python, και χρησιμοποιήθηκε ένας πράκτορας τεχνητής νοημοσύνης (AI Agent) βασισμένος σε αλγόριθμο μη επιβλεπόμενης μάθησης. Μέσα από αυτή την εργασία, στόχος είναι να αναδειχθούν οι δυνατότητες και οι εφαρμογές της μη επιβλεπόμενης μάθησης στα παιχνίδια, παρέχοντας πρακτικά παραδείγματα και κατευθύνσεις για την υλοποίηση παρόμοιων έργων.

Αντικείμενο της διπλωματικής εργασίας

Η παρούσα διπλωματική εργασία έχει ως κύριο θέμα την ανάπτυξη μιας απλοποιημένης μορφής του παιχνιδιού Dominion με τη χρήση Python, στο οποίο θα χρησιμοποιηθεί ένας πράκτορας τεχνητής νοημοσύνης (AI Agent) βασισμένος σε αλγόριθμο μη επιβλεπόμενης μάθησης. Το πρόβλημα που αντιμετωπίζεται είναι η ενσωμάτωση αλγορίθμων μηχανικής μάθησης σε περιβάλλον παιχνιδιού, με σκοπό τη βελτίωση της απόδοσης του AI Agent. Η εργασία αυτή είναι επίκαιρη καθώς η τεχνητή νοημοσύνη και η μηχανική μάθηση συνεχίζουν να παίζουν σημαντικό ρόλο στην εξέλιξη των βιντεοπαιχνιδιών, ενισχύοντας την εμπειρία του χρήστη, δίνοντας παράλληλα νέες ιδέες για την ανάπτυξη νέων καινοτόμων εφαρμογών.

Σκοπός και στόχοι

Ο σκοπός της εργασίας είναι να αναπτύξει ένα παιχνίδι στο οποίο ένας AI Agent εκπαιδεύεται μέσω αλγόριθμων μη επιβλεπόμενης μάθησης για να βελτιώνει τις επιδόσεις του στο παιχνίδι. Οι στόχοι της εργασίας αναλύονται ως εξής:

- Πώς έχει εξελιχθεί η χρήση της τεχνητής νοημοσύνης στα βιντεοπαιχνίδια.
- Ποιες είναι οι βασικές αρχές της τεχνητής νοημοσύνης και της μηχανικής μάθησης.
- Ποια είναι τα χαρακτηριστικά των αλγόριθμων μη επιβλεπόμενης μάθησης και πώς μπορούν να εφαρμοστούν σε ένα παιχνίδι.
- Πώς μπορεί να σχεδιαστεί και να υλοποιηθεί ένας AI Agent που χρησιμοποιεί μη επιβλεπόμενη μάθηση για να παίζει ένα παιχνίδι.
- Ποια είναι η απόδοση του AI Agent και πώς μπορεί να βελτιωθεί περαιτέρω.

Μεθοδολογία

Η μεθοδολογία που ακολουθήθηκε περιλαμβάνει τα εξής βήματα:

- **Έρευνα και Ανάλυση Βιβλιογραφίας:** Μελέτη της υπάρχουσας βιβλιογραφίας για την κατανόηση των βασικών αρχών της τεχνητής νοημοσύνης και της μηχανικής μάθησης, καθώς και των εφαρμογών τους στα βιντεοπαιχνίδια.

- **Σχεδιασμός του Παιχνιδιού:** Ανάπτυξη του παιχνιδιού καρτών με κανόνες και διάφορες κάρτες (δράσης, θησαυρού, νίκης).

- **Υλοποίηση του AI Agent:** Δημιουργία του AI Agent χρησιμοποιώντας αλγόριθμους μη επιβλεπόμενης μάθησης, K-means, και ενισχυτική μάθηση μέσω Deep Q-Network (DQN).

- **Εκπαίδευση και Αξιολόγηση:** Εκπαίδευση του AI Agent μέσω πολλαπλών παιχνιδιών και αξιολόγηση της απόδοσής του.

- **Ανάλυση Αποτελεσμάτων:** Οπτικοποίηση των αποτελεσμάτων και ανάλυση της απόδοσης του AI Agent.

Καινοτομία

Η καινοτομία της διπλωματικής εργασίας έγκειται στα εξής:

- **Συνδυασμός Μη Επιβλεπόμενης και Ενισχυτικής Μάθησης:** Χρήση αλγορίθμου K-means για την κατηγοριοποίηση των καταστάσεων του παιχνιδιού και συνδυασμός με ενισχυτική μάθηση μέσω DQN για την εκπαίδευση του AI Agent.

- **Μη επιβλεπόμενη μάθηση:** Ανάπτυξη ενός AI Agent που μαθαίνει και προσαρμόζεται δυναμικά στις συνθήκες του παιχνιδιού χωρίς την ανάγκη εποπτείας.

- **Εφαρμογή σε Παιχνίδια Καρτών:** Εφαρμογή αλγορίθμων μηχανικής μάθησης σε ένα παιχνίδι καρτών, προσφέροντας νέες προοπτικές για την ανάπτυξη παιχνιδιών στρατηγικής και τακτικής.

Δομή

Η διπλωματική εργασία οργανώνεται σε εννέα κεφάλαια. Το κάθε ένα εξετάζει διαφορετικές πτυχές της ανάπτυξης ενός παιχνιδιού με τη χρήση AI Agent που βασίζεται σε αλγόριθμους μη επιβλεπόμενης μάθησης. Αναλυτικότερα τα κεφάλαια :

Κεφάλαιο 1: Ιστορική Εξέλιξη Παιχνιδιών με AI Στο κεφάλαιο αυτό αναφέρεται στην επιρροή της τεχνητής νοημοσύνης στα παιχνίδια, περιγράφοντας τις βασικές εφαρμογές της.

Κεφάλαιο 2: Εισαγωγή στην Τεχνητή Νοημοσύνη (AI) Στο κεφάλαιο αυτό γίνεται η περιγραφή των βασικών εννοιών της τεχνητής νοημοσύνης, της μηχανικής μάθησης και των υπό-πεδίων της, όπως τα νευρωνικά δίκτυα, η αναγνώριση προτύπων, η επεξεργασία φυσικής γλώσσας και η υπολογιστική όραση.

Κεφάλαιο 3: Μηχανική Μάθηση (Machine learning) Στο κεφάλαιο αυτό παρουσιάζονται οι αρχές της μηχανικής μάθησης, κατηγοριοποιώντας τους αλγόριθμους σε εποπτευόμενη, μη εποπτευόμενη και ενισχυτική μάθηση.

Κεφάλαιο 4: Τεχνικες **Μάθησης Σε Τεχνητή Νοημοσύνη** Στο κεφάλαιο αυτό αναλύονται οι τεχνικές και οι αλγόριθμοι της εποπτευόμενης μάθησης, όπως ο αλγόριθμος Naive Bayes, K-Nearest Neighbor, τα νευρωνικά δίκτυα και τα δέντρα αποφάσεων.

Κεφάλαιο 5: Ανάπτυξη Παιχνιδιού με AI Agent Το κεφάλαιο αυτό περιλαμβάνει τον σχεδιασμό και την υλοποίηση του παιχνιδιού, την ανάπτυξη του AI Agent με χρήση αλγόριθμου K-Means και DQN, την ανάλυση του κώδικα, καθώς και την εκπαίδευση και αξιολόγηση της απόδοσης του AI Agent.

Κεφάλαιο 6: Συμπεράσματα και Μελλοντική Εργασία Στο τελευταίο κεφάλαιο συνοψίζονται τα συμπεράσματα της εργασίας, τα αποτελέσματα από το πρακτικό κομμάτι και οι προτάσεις για μελλοντική έρευνα και βελτίωση της απόδοσης του AI Agent.

1 ΚΕΦΑΛΑΙΟ 1ο Ιστορική Εξέλιξη Παιχνιδιών Με AI

Εισαγωγή

Η τεχνητή νοημοσύνη (AI) έχει επηρεάσει τη βιομηχανία των παιχνιδιών, δημιουργώντας πιο έξυπνα και ρεαλιστικά περιβάλλοντα για τα παιχνίδια. Από τον τρόπο με τον οποίο κινούνται και φέρονται οι χαρακτήρες του παιχνιδιού (NPC) μέχρι την εμπειρία των παικτών, η τεχνητή νοημοσύνη έχει βοηθήσει στην ανάπτυξη των παιχνιδιών. Αυτό το κεφάλαιο εξετάζει την ιστορική εξέλιξη της AI στα παιχνίδια και τις διάφορες εφαρμογές της.

1.1 Επιρροή Της Τεχνητής Νοημοσύνης

Η τεχνητή νοημοσύνη σε ένα ευρύ φάσμα εφαρμόζεται με πολλούς τρόπους στον κόσμο των παιχνιδιών. Συγκεκριμένα, συμπεριλαμβάνει:

- **Προγραμματισμός Επιπέδου Εχθρών (NPCs):** Γίνεται χρήση της τεχνητής νοημοσύνης για τον προγραμματισμό της συμπεριφοράς των εχθρών και άλλων NPCs, καθώς και για το πώς εξελίσσονται κατά τη διάρκεια του παιχνιδιού, με σκοπό να προσφέρει στους παίκτες μια εμπειρία πιο δυναμική και ενδιαφέρουσα.

- **Βελτιστοποίηση Απόδοσης:** Η τεχνητή νοημοσύνη στον τομέα των γραφικών έχει μεγάλη συμβολή στο να βελτιστοποιείται η απόδοση, εξασφαλίζοντας την υψηλή ποιότητα της εικόνας και την ομαλή κίνηση δίχως επιβάρυνση του συστήματος.

- **Προσωποποίηση και Προσαρμογή:** Πολλά παιχνίδια κάνουν χρήση της τεχνητής νοημοσύνης με σκοπό να προσαρμοστεί η δυσκολία, η εμπειρία, ακόμη και το περιεχόμενο στις προτιμήσεις και στις επιδόσεις των παικτών.

- **Επεξεργασία Φυσικής:** Μέσω της τεχνητής νοημοσύνης βελτιώνεται η φυσική ακρίβεια και η αληθοφάνεια των παιχνιδιών με πολύπλοκες φυσικές διεργασίες, όπως η φυσική κίνηση, οι συγκρούσεις και η ροή.

- **Προβλέψεις και Αποφάσεις:** Στα παιχνίδια στρατηγικής χρησιμοποιείται η τεχνητή νοημοσύνη για τη λήψη αποφάσεων, προβλέποντας τις κινήσεις των παικτών και προσαρμόζοντας τις στρατηγικές των αντιπάλων αναλόγως.

1.2 Εφαρμογές Της Τεχνητής Νοημοσύνης Στα Παιχνίδια

Οι σύγχρονες εφαρμογές της τεχνητής νοημοσύνης στα παιχνίδια εξελίσσονται συνεχώς και περιλαμβάνουν πολλούς καινοτόμους τρόπους χρήσης. Ανάμεσα σε αυτούς συγκαταλέγονται:

- **Συνεχής Μάθηση (Continuous Learning):** Συστήματα τεχνητής νοημοσύνης που είναι σε θέση να μαθαίνουν κατά τη διάρκεια του παιχνιδιού, προσαρμόζοντας τη στρατηγική τους ανάλογα με την εμπειρία και τις αντιδράσεις του παίκτη.

- **Συστήματα Πρόβλεψης και Προβληματισμού (Prediction and Anticipation Systems):** Χρήση αλγορίθμων μάθησης με σκοπό την πρόβλεψη των κινήσεων του

παίκτη ή την αντίληψη των στρατηγικών του αντιπάλου.

- **Ενισχυτική Μάθηση (Reinforcement Learning):** Αλγόριθμοι που μαθαίνουν τις βέλτιστες ενέργειες σε ένα περιβάλλον μέσω δοκιμών και λαθών, βασιζόμενοι σε ανταμοιβές και ποινές.

- **Εφαρμογές Εικονικής Πραγματικότητας (Virtual Reality - VR) και Επαυξημένης Πραγματικότητας (Augmented Reality - AR):** Χρήση της τεχνητής νοημοσύνης για την αντίληψη, την ανίχνευση και τη διαχείριση των αντικειμένων και των συμπεριφορών στο εικονικό περιβάλλον.

- **Προσαρμοστική Εμπειρία (Adaptive Experience):** Ανάπτυξη παιχνιδιών που προσαρμόζονται δυναμικά στον παίκτη, το επίπεδο δεξιοτήτων του, τις προτιμήσεις του και την πρόοδό του.

- **Υποστήριξη Παίκτη (Player Support):** Χρήση της τεχνητής νοημοσύνης για την παροχή προτάσεων, συμβουλών ή ακόμα και αυτόνομης συμπεριφοράς προκειμένου να βοηθηθεί ο παίκτης να προχωρήσει στο παιχνίδι.

1.3 Τεχνικές Και Εφαρμογές Της Τεχνητής Νοημοσύνης Στα Παιχνίδια

- **Προσαρμοστική Νοημοσύνη (Adaptive AI):** Η προσαρμοστική νοημοσύνη στα βιντεοπαιχνίδια ορίζεται ως η χρήση τεχνικών τεχνητής νοημοσύνης προκειμένου οι χαρακτήρες του παιχνιδιού (NPC) να προσαρμόζουν τη συμπεριφορά τους ανάλογα με τις ενέργειες του παίκτη. Με την προσαρμοστικότητα αυτή από τους χαρακτήρες του παιχνιδιού, οι παίκτες έχουν μια πιο δυναμική εμπειρία παιχνιδιού, αφού αναλόγως με το τι επιλέγουν, υπάρχει μια διαφορετική αντίδραση και εξέλιξη στο παιχνίδι τους.

- **Δημιουργία Περιεχομένου:** Η δημιουργία περιεχομένου μέσω της χρήσης αλγορίθμων και τεχνικών τεχνητής νοημοσύνης για την αυτόματη παραγωγή περιεχομένου στα βιντεοπαιχνίδια παίζει πλέον ένα πολύ σημαντικό ρόλο στη δημιουργία των παιχνιδιών. Ως περιεχόμενο ενός παιχνιδιού ορίζονται οι χαρακτήρες, οι χάρτες, το περιβάλλον του παιχνιδιού, οι αποστολές και οτιδήποτε άλλο προσφέρει μια εμπειρία παιχνιδιού στον παίκτη. Με την είσοδο της τεχνητής νοημοσύνης στη σχεδίαση και την υλοποίηση των παιχνιδιών, η διαδικασία αυτή έχει βοηθήσει τους δημιουργούς των παιχνιδιών, αφού η τεχνητή νοημοσύνη υλοποιεί το μεγαλύτερο μέρος αυτών.

- **Αυτόματη Δοκιμή (Automated Testing):** Με τη χρήση της τεχνητής νοημοσύνης στη δημιουργία των παιχνιδιών, ένας ακόμη τομέας που επηρεάζεται είναι η δοκιμή του παιχνιδιού. Με τη χρήση των αλγορίθμων, η ανίχνευση σφαλμάτων και λαθών στο παιχνίδι έχει επιταχυνθεί. Ως αποτέλεσμα αυτού, εξασφαλίζεται η καλή ποιότητα του τελικού παιχνιδιού πριν δοθεί στην κυκλοφορία.

2 ΚΕΦΑΛΑΙΟ 2ο : Εισαγωγή στην Τεχνητή Νοημοσύνη (AI)

Εισαγωγή

Η Τεχνητή Νοημοσύνη (AI) είναι ένας από τους πιο καινοτόμους τομέας της επιστήμης των υπολογιστών. Η δυνατότητα δημιουργίας συστημάτων που μπορούν να εκτελούν εργασίες που αντιγράφουν την ανθρώπινη νοημοσύνη και τρόπο σκέψης των ανθρώπων, έχει οδηγήσει σε σημαντικές εξελίξεις σε διάφορους τομείς.

2.1 Ορισμός της τεχνητής νοημοσύνης

Η Τεχνητή Νοημοσύνη (AI) είναι ο τομέας της επιστήμης των υπολογιστών που ασχολείται με τη δημιουργία συστημάτων ικανά να εκτελούν εργασίες που απαιτούν ανθρώπινη νοημοσύνη. Αυτές οι εργασίες περιλαμβάνουν την αναγνώριση της ομιλίας, την επεξεργασία της φυσικής γλώσσας, την όραση των υπολογιστών και τη λήψη αποφάσεων. Η Τεχνητή Νοημοσύνη συνδυάζει επιστημονικά πεδία όπως η μηχανική μάθηση, η αναγνώριση προτύπων και η ρομποτική. Οι κύριοι τομείς της τεχνητής νοημοσύνης είναι:

- **Μηχανική Μάθηση:** Τεχνικές που επιτρέπουν στους υπολογιστές να μαθαίνουν από δεδομένα.
- **Νευρωνικά Δίκτυα:** Δίκτυα εμπνευσμένα από τον ανθρώπινο εγκέφαλο για την επεξεργασία και ανάλυση δεδομένων.
- **Αναζήτηση και Βελτιστοποίηση:** Αλγόριθμοι που βρίσκουν βέλτιστες λύσεις σε προβλήματα.

2.2 Σχεδιασμός και Ανάπτυξη Νευρωνικών Δικτύων

Τα νευρωνικά δίκτυα είναι η βάση της σύγχρονης τεχνητής νοημοσύνης και της μηχανικής μάθησης. Αποτελούνται από επίπεδα (layers) νευρώνων που επεξεργάζονται τα δεδομένα εισόδου και τα μετατρέπουν σε έξοδο. Ένα τυπικό νευρωνικό δίκτυο αποτελείται από:

- **Είσοδος (Input Layer):** Το πρώτο επίπεδο που λαμβάνει τα δεδομένα εισόδου.
- **Κρυφά Επίπεδα (Hidden Layers):** Ενδιάμεσα επίπεδα που επεξεργάζονται τα δεδομένα μέσω ζυγών και συναρτήσεων ενεργοποίησης.
- **Έξοδος (Output Layer):** Το τελευταίο επίπεδο που παράγει την τελική έξοδο του δικτύου.

Οι συναρτήσεις ενεργοποίησης, όπως η ReLU (Rectified Linear Unit) και η sigmoid, είναι σημαντικές για την προσθήκη μη γραμμικότητας στα νευρωνικά δίκτυα, επιτρέποντάς τους να μάθουν πολύπλοκα πρότυπα.

2.3 Υποπεδία της Τεχνητής Νοημοσύνης

Η τεχνητή νοημοσύνη περιλαμβάνει μια πληθώρα υποπεδίων, καθένα από τα οποία επικεντρώνεται σε συγκεκριμένες πτυχές της δημιουργίας των συστημάτων.

- **Αναγνώριση Προτύπων (Pattern Recognition):** Το πεδίο αυτό επικεντρώνεται στον εντοπισμό και την αναγνώριση μοτίβων και δομών στα δεδομένα. Χρησιμοποιείται σε εφαρμογές όπως η αναγνώριση εικόνας, η αναγνώριση φωνής και η βιομετρία.
- **Επεξεργασία Φυσικής Γλώσσας (Natural Language Processing):** Αυτό το πεδίο ασχολείται με την αλληλεπίδραση μεταξύ υπολογιστών και ανθρώπινης γλώσσας. Εφαρμογές περιλαμβάνουν συστήματα αυτόματης μετάφρασης, αναγνώριση ομιλίας και ανάλυση συναισθήματος.
- **Υπολογιστική Όραση (Computer Vision):** Η υπολογιστική όραση επικεντρώνεται στην εξαγωγή, ανάλυση και κατανόηση πληροφορίας από εικόνες και βίντεο. Χρησιμοποιείται σε εφαρμογές όπως η ανίχνευση αντικειμένων, η αναγνώριση προσώπων και η πλοήγηση ρομπότ.
- **Συστήματα Συστάσεων (Recommender Systems):** Αυτά τα συστήματα παρέχουν προτάσεις για προϊόντα ή περιεχόμενο βασιζόμενα σε ιστορικά δεδομένα χρηστών.
- **Αυτόνομα Οχήματα (Autonomous Vehicles):** Η τεχνητή νοημοσύνη χρησιμοποιείται για την ανάπτυξη οχημάτων που μπορούν να λειτουργούν χωρίς ανθρώπινη παρέμβαση. Τα αυτόνομα οχήματα χρησιμοποιούν δεδομένα από αισθητήρες και αλγόριθμους μηχανικής μάθησης για να κινούνται και να λαμβάνουν αποφάσεις σε πραγματικό χρόνο.
- **Ρομποτική (Robotics):** Στη ρομποτική, η τεχνητή νοημοσύνη χρησιμοποιείται για να δώσει στα ρομπότ την ικανότητα να αντιλαμβάνονται το περιβάλλον τους, να λαμβάνουν αποφάσεις και να εκτελούν εργασίες αυτόνομα. Οι εφαρμογές περιλαμβάνουν τη βιομηχανία, την ιατρική και την εξερεύνηση του διαστήματος.
- **Ανάλυση Δεδομένων (Data Analytics):** Η τεχνητή νοημοσύνη χρησιμοποιείται για την ανάλυση μεγάλων όγκων δεδομένων με στόχο την εξαγωγή χρήσιμων πληροφοριών και τη λήψη αποφάσεων. Οι τεχνικές περιλαμβάνουν την εξόρυξη δεδομένων (data mining), την πρόβλεψη και την ανίχνευση προτύπων.
- **Υγεία (Healthcare):** Η τεχνητή νοημοσύνη έχει σημαντικές εφαρμογές στην υγεία, όπως η ανάλυση ιατρικών εικόνων, η πρόβλεψη ασθενειών και η ανάπτυξη έξυπνων συστημάτων υποστήριξης για ιατρικές αποφάσεις.

3 ΚΕΦΑΛΑΙΟ 3ο : Μηχανική Μάθηση (Machine Learning).

Εισαγωγή

Η μηχανική μάθηση αποτελεί έναν από τους κύριους κλάδους της τεχνητής νοημοσύνης και έχει γνωρίσει μεγάλη ανάπτυξη τα τελευταία χρόνια. Η δυνατότητα των υπολογιστών να μαθαίνουν από δεδομένα και να βελτιώνουν τις επιδόσεις τους χωρίς ανθρώπινη παρέμβαση έχει ανοίξει νέους ορίζοντες σε πολλούς τομείς.

3.1 Ορισμός της Μηχανικής Μάθησης

Η μηχανική μάθηση σχετίζεται άμεσα με την εκμάθηση από δεδομένα και τη λήψη αποφάσεων ή προβλέψεων. Βασίζεται στην υπόθεση ότι οι μηχανές μπορούν να αποκτήσουν ευφυΐα, μαθαίνοντας από προηγούμενους υπολογισμούς και προσαρμοζόμενες στο περιβάλλον.

Το τυπικό πλαίσιο μηχανικής μάθησης περιλαμβάνει μια διαδικασία εκπαίδευσης και μια διαδικασία δοκιμών. Η πρώτη επιτρέπει στο πλαίσιο μηχανικής μάθησης να ανακαλύψει τις σχέσεις μεταξύ δεδομένων εισόδου και δεδομένων εξόδου. Τα υπάρχοντα μοντέλα μηχανικής μάθησης μπορούν να ταξινομηθούν σε:

- **Μοντέλα Ταξινόμησης:** Χρησιμοποιούνται για την επίλυση δυαδικών ταξινομήσεων ή πολλαπλών προβλημάτων ταξινόμησης.
- **Μοντέλα Παλινδρόμησης:** Μπορούν να χρησιμοποιηθούν για την εκτέλεση προβλέψεων.
- **Δομημένα Μοντέλα Μάθησης:** Χρησιμοποιούνται ευρέως σε πολλούς τομείς, όπως η επεξεργασία φυσικής γλώσσας.

Σύμφωνα με τη μέθοδο εκπαίδευσης, η μηχανική μάθηση μπορεί να κατηγοριοποιηθεί σε:

- **Εποπτευόμενη Μάθηση:** Όπου το σύστημα εκπαιδεύεται με τη χρήση δεδομένων που έχουν ετικέτες.
- **Μη Εποπτευόμενη Μάθηση:** Όπου το σύστημα μαθαίνει από δεδομένα χωρίς ετικέτες.
- **Μάθηση Ενίσχυσης:** Όπου το σύστημα μαθαίνει μέσω δοκιμής και λάθους, βασιζόμενο σε ανταμοιβές και ποινές.

3.2 Αλγόριθμοι Βελτιστοποίησης

Οι αλγόριθμοι βελτιστοποίησης είναι κρίσιμοι για την εκπαίδευση νευρωνικών δικτύων. Ο Adam (Adaptive Moment Estimation) είναι ένας από τους πιο δημοφιλείς αλγόριθμους βελτιστοποίησης που συνδυάζει τις ιδιότητες του RMSprop και του Stochastic Gradient Descent με Momentum

(SGD). Ο Adam χρησιμοποιεί προσαρμοστικούς ρυθμούς μάθησης για κάθε παράμετρο, κάνοντας την εκπαίδευση γρηγορότερη και πιο αποδοτική.

Άλλοι σημαντικοί αλγόριθμοι βελτιστοποίησης περιλαμβάνουν:

- **Stochastic Gradient Descent (SGD):** Χρησιμοποιεί τυχαία δείγματα από το σύνολο δεδομένων για την ενημέρωση των παραμέτρων του μοντέλου, επιτρέποντας ταχύτερη σύγκλιση.

- **RMSprop:** Προσαρμόζει τον ρυθμό μάθησης για κάθε παράμετρο ξεχωριστά, βασισμένος στην πρόσφατη ιστορία των gradients, αποφεύγοντας μεγάλα άλματα κατά την ενημέρωση των παραμέτρων.

3.3 Παραδείγματα Χρήσης Μηχανικής Μάθησης

Οι τεχνικές μηχανικής μάθησης εφαρμόζονται σε διάφορους τομείς και παρέχουν λύσεις σε πολύπλοκα προβλήματα. Ορισμένα από τα σημαντικότερα παραδείγματα χρήσης περιλαμβάνουν:

- **Ιατρική Διάγνωση:** Τα μοντέλα μηχανικής μάθησης χρησιμοποιούνται για την ανάλυση ιατρικών εικόνων και τη διάγνωση ασθενειών. Για παράδειγμα, τα νευρωνικά δίκτυα χρησιμοποιούνται για την ανίχνευση όγκων σε ακτινογραφίες.
- **Χρηματοοικονομικές Προβλέψεις:** Στον τομέα των χρηματοοικονομικών, οι αλγόριθμοι μηχανικής μάθησης χρησιμοποιούνται για την πρόβλεψη τιμών μετοχών και την ανάλυση κινδύνου. Η δυνατότητα να αναγνωρίζουν μοτίβα και να κάνουν προβλέψεις είναι ανεκτίμητη για τη λήψη αποφάσεων στις χρηματοοικονομικές αγορές.
- **Αναγνώριση Φωνής και Γλώσσας:** Οι αλγόριθμοι μηχανικής μάθησης χρησιμοποιούνται στην αναγνώριση φωνής και στην επεξεργασία φυσικής γλώσσας, επιτρέποντας τη δημιουργία έξυπνων βοηθών όπως η Siri και η Alexa. Αυτές οι τεχνολογίες μπορούν να κατανοήσουν και να απαντήσουν σε ανθρώπινες εντολές με υψηλή ακρίβεια.
- **Συστήματα Συστάσεων:** Χρησιμοποιούνται από εταιρείες όπως η Amazon και το Netflix για την παροχή προτάσεων προϊόντων και περιεχομένου στους χρήστες. Οι αλγόριθμοι μηχανικής μάθησης αναλύουν τα δεδομένα χρήστη και προτείνουν αντικείμενα που πιθανώς θα τον ενδιαφέρουν.
- **Ανίχνευση Απάτης:** Οι τράπεζες και οι εταιρείες χρηματοοικονομικών υπηρεσιών χρησιμοποιούν αλγόριθμους μηχανικής μάθησης για την ανίχνευση απάτης στις συναλλαγές. Αναλύουν μοτίβα συναλλαγών και εντοπίζουν ανωμαλίες που μπορεί να υποδηλώνουν απάτη.
- **Πρόβλεψη Συντήρησης:** Οι βιομηχανίες χρησιμοποιούν μηχανική μάθηση για την πρόβλεψη της ανάγκης συντήρησης εξοπλισμού. Αναλύοντας τα δεδομένα από τους

αισθητήρες των μηχανημάτων, μπορούν να προβλέψουν πότε θα χρειαστεί συντήρηση, αποφεύγοντας έτσι τις βλάβες και τις διακοπές στη λειτουργία.

4 ΚΕΦΑΛΑΙΟ 4ο: Τεχνικές Μάθησης σε Τεχνητή Νοημοσύνη

4.1 Εποπτευόμενη Μάθηση (Supervised Learning)

Η εποπτευόμενη μάθηση είναι μια από τις βασικές μεθόδους της μηχανικής μάθησης, η οποία χρησιμοποιείται για την επίλυση προβλημάτων που απαιτούν πρόβλεψη και ταξινόμηση δεδομένων.

Στην εποπτευόμενη μάθηση, απαραίτητη είναι η ύπαρξη ενός επόπτη που θα ορίσει τα δεδομένα εισόδου και εξόδου. Κατά τη διαδικασία της εκπαίδευσης, ο αλγόριθμος εκμάθησης τροφοδοτείται με ένα σύνολο δεδομένων, τα οποία θα χρησιμοποιηθούν ως δεδομένα εκπαίδευσης. Τα δεδομένα περιέχουν γνωστή είσοδο και έξοδο.

Στη φάση δοκιμής, ένα νέο σύνολο δεδομένων δοκιμής (test) τροφοδοτείται στο μοντέλο που έχει εκπαιδευθεί για να παραχθεί η αναμενόμενη έξοδος. Για την χρήση της εποπτευόμενης μάθησης είναι απαραίτητη η ύπαρξη μεγάλου όγκου δεδομένων τα οποία θα χρησιμοποιηθούν για την εκπαίδευση του αλγορίθμου. Χρησιμοποιείται ευρέως σε πολλά πεδία, όπως η αναγνώριση ομιλίας και αντικειμένων.

Τα μοντέλα μηχανικής μάθησης μπορούν να ταξινομηθούν σε μοντέλα ταξινόμησης, μοντέλα παλινδρόμησης και δομημένα μοντέλα μάθησης:

- **Ταξινόμηση:** Χρησιμοποιούνται για να προβλέψουν την κατηγορία ενός δεδομένου με βάση τα χαρακτηριστικά του. Για παράδειγμα, ένα μοντέλο ταξινόμησης μπορεί να προβλέψει αν ένα email είναι spam.
- **Παλινδρόμηση:** Χρησιμοποιούνται για να προβλέψουν μια συνεχή τιμή ή ποσότητα, αντί για κατηγορία. Για παράδειγμα, χρησιμοποιούνται για την πρόβλεψη της τιμής ενός σπιτιού βάσει των χαρακτηριστικών του.
- **Δομημένα Μοντέλα Μάθησης:** Χρησιμοποιούνται για να εκπαιδευτούν δεδομένα με δομική οργάνωση, όπως οι συνδέσεις μεταξύ των χαρακτηριστικών.

Η εποπτευόμενη μάθηση είναι ένα ισχυρό εργαλείο που επιτρέπει στους ερευνητές και τους μηχανικούς να κατασκευάσουν μοντέλα που μπορούν να κάνουν ακριβείς προβλέψεις και να πάρουν αποφάσεις βασισμένες σε δεδομένα. Αυτός ο τύπος μάθησης χρησιμοποιείται ευρέως σε εφαρμογές όπως η αναγνώριση προσώπου, η διάγνωση ασθενειών και η ανίχνευση απάτης.

4.2 Αλγόριθμοι Εποπτευόμενης Μάθησης

- **Naive Bayes**: Χρησιμοποιείται κυρίως για προβλήματα ταξινόμησης.
- **K-Nearest Neighbor (KNN)**: Βασίζεται στην εύρεση των κοντινότερων γειτόνων στο χώρο των χαρακτηριστικών.
- **Νευρωνικά Δίκτυα**: Μιμούνται τον ανθρώπινο εγκέφαλο με στρώματα νευρώνων.

- **Δέντρα Απόφασης**: Δημιουργούν μοντέλα βασισμένα σε ερωτήσεις που οδηγούν σε αποφάσεις.

4.3 Μη Εποπτευόμενη Μάθηση (Unsupervised Learning)

Η μη επιβλεπόμενη μάθηση αποτελεί έναν από τους πιο σημαντικούς κλάδους της μηχανικής μάθησης. Σε αντίθεση με την επιβλεπόμενη μάθηση, η μη επιβλεπόμενη μάθηση δεν απαιτεί ετικέτες ή κατηγοριοποιημένα δεδομένα για την εκπαίδευση του μοντέλου. Αντίθετα, εστιάζει στην αναγνώριση προτύπων και δομών μέσα στα δεδομένα, επιτρέποντας την αποκάλυψη κρυφών σχέσεων και την κατανόηση της εσωτερικής δομής των δεδομένων.

Η μη επιβλεπόμενη μάθηση είναι ένα είδος μηχανικής μάθησης που δεν χρησιμοποιεί ετικέτες ή παραδείγματα για την εκπαίδευση του. Σε αντίθεση με την επιβλεπόμενη μάθηση, το μοντέλο αναγνωρίζει πρότυπα ή δομές χωρίς κάποια καθοδήγηση.

Στην επιβλεπόμενη μάθηση, το μοντέλο προσπαθεί να προβλέψει μια τιμή εξόδου με βάση τις ετικέτες των δεδομένων τα οποία έχει σαν είσοδο. Το μοντέλο προσπαθεί να βγάλει ένα αποτέλεσμα στην έξοδο χωρίς κάποια πληροφορία από τα δεδομένα εισόδου.

4.4 Χαρακτηριστικά Μη Εποπτευόμενη Μάθηση

1. **Απουσία Ετικετών**: Στη μη επιβλεπόμενη μάθηση, τα δεδομένα δεν έχουν ετικέτες. Αντίθετα, το μοντέλο πρέπει να αναγνωρίσει μόνο τα πρότυπα ή τις δομές που υπάρχουν στα δεδομένα.

2. **Εκμάθηση Προτύπων**: Στη μη επιβλεπόμενη μάθηση, το μοντέλο προσπαθεί να εντοπίσει πρότυπα στα δεδομένα, όπως συστάδες (clusters) ή κοινότητες, χωρίς να έχει προηγούμενη γνώση για τη φύση αυτών των προτύπων.

3. **Εφαρμογές**: Οι μη επιβλεπόμενες μέθοδοι μάθησης είναι χρήσιμες σε πολλά προβλήματα, όπως η ομαδοποίηση δεδομένων, η εξαγωγή χαρακτηριστικών και η μείωση της διάστασης. Επίσης, χρησιμοποιούνται συχνά σε πειράματα αναγνώρισης προτύπων ως προετοιμασία για επόμενες διαδικασίες.

4. **Απουσία Ετικετών**: Στη μη επιβλεπόμενη μάθηση, τα δεδομένα δεν έχουν ετικέτες. Αντίθετα, το μοντέλο πρέπει να αναγνωρίσει μόνο τα πρότυπα ή τις δομές που υπάρχουν στα δεδομένα.

4.5 Αλγόριθμοι Μη Εποπτευόμενης Μάθησης :

- **K-means**: Χρησιμοποιείται για την ομαδοποίηση δεδομένων σε κέντρα συστάδων (clusters).

- **PCA (Principal Component Analysis)**: Μειώνει τη διάσταση των δεδομένων διατηρώντας τη μεγαλύτερη δυνατή διασπορά.

- **DBSCAN**: Βασισμένος στην πυκνότητα, μπορεί να εντοπίσει συστάδες οποιουδήποτε σχήματος και να απορρίψει θόρυβο.

4.6 Σύγκριση Εποπτευόμενης Και Μη Εποπτευόμενης Μάθησης

Η εποπτευόμενη μάθηση απαιτεί μεγάλο όγκο δεδομένων με ετικέτες και είναι ιδανική για προβλήματα που απαιτούν πρόβλεψη ή ταξινόμηση. Η μη επιβλεπόμενη μάθηση, από την άλλη, είναι κατάλληλη για την ανακάλυψη μοτίβων και σχέσεων χωρίς να υπάρχει προ απαιτούμενη γνώση των δεδομένων.

4.7 Ενισχυτική Μάθηση (Reinforcement Learning)

Η ενισχυτική μάθηση (Reinforcement Learning) αποτελεί έναν από τους πιο σημαντικούς τομείς της μηχανικής μάθησης. Σε αντίθεση με την εποπτευόμενη και τη μη εποπτευόμενη μάθηση, η ενισχυτική μάθηση βασίζεται στην αλληλεπίδραση ενός πράκτορα με το περιβάλλον του, με σκοπό να μάθει από τις συνέπειες των ενεργειών του και να βελτιώσει την απόδοσή του μέσω επαναλαμβανόμενων δοκιμών και λαθών.

Η ενισχυτική μάθηση αναφέρεται σαν ένα είδος μη επιβλεπόμενης μάθησης, όπου ένας αλγόριθμος προσπαθεί να μάθει ποιες ενέργειες (actions) πρέπει να ακολουθήσει σε ένα περιβάλλον, με σκοπό να μεγιστοποιήσει έναν αριθμητικό καθορισμένο στόχο (reward). Ο αλγόριθμος λαμβάνει αποφάσεις λαμβάνοντας υπόψη τις επιπτώσεις των ενεργειών του στο μέλλον, καθώς οι επιβραβεύσεις ή ποινές που λαμβάνει επηρεάζουν τις μελλοντικές επιλογές του. Ένα από τα πιο διάσημα παραδείγματα είναι ο αλγόριθμος Q-learning .

4.8 DQN (Deep Q-Network)

Ο DQN είναι ένας αλγόριθμος ενισχυτικής μάθησης που χρησιμοποιείται σε περιβάλλοντα με μεγάλη διάσταση, όπως βιντεοπαιχνίδια. Χρησιμοποιεί ένα νευρωνικό δίκτυο για να εκτιμήσει την Q-τιμή των ενεργειών σε κάθε κατάσταση, όπου η Q-τιμή αντιπροσωπεύει τη μελλοντική ανταμοιβή που αναμένεται από μια συγκεκριμένη ενέργεια σε μια κατάσταση. Ο στόχος του DQN είναι να μάθει να επιλέγει τις ενέργειες που θα του επιφέρουν τη μεγαλύτερη μακροπρόθεσμη ανταμοιβή. Στο κομμάτι του κώδικα έχει γίνει χρήση του αλγορίθμου DQN.

4.8.1 Πειραματική Εφαρμογή του DQN

Η εφαρμογή του DQN περιλαμβάνει την εκπαίδευση του νευρωνικού δικτύου σε ένα περιβάλλον ενισχυτικής μάθησης. Το νευρωνικό δίκτυο εκπαιδεύεται να εκτιμά την Q-τιμή για κάθε πιθανή ενέργεια σε μια κατάσταση, και ο πράκτορας επιλέγει την ενέργεια που μεγιστοποιεί τη μακροπρόθεσμη ανταμοιβή. Κατά την εκπαίδευση, χρησιμοποιούνται τεχνικές όπως η επαναληπτική μνήμη (replay memory) και η σταθερή στόχευση (target networks) για τη βελτίωση της απόδοσης του πράκτορα.

4.8.2 Λεπτομέρειες Παραμέτρων Αλγορίθμου DQN

Ο αλγόριθμος DQN (Deep-Q-Network) χρησιμοποιεί τις ακόλουθες παραμέτρους για τη βελτιστοποίηση της μάθησης:

1. **State size**: Το μέγεθος του διανύσματος εισόδου που αντιπροσωπεύει την κατάσταση του περιβάλλοντος. Για το συγκεκριμένο παιχνίδι, το state size είναι 8, καθώς περιλαμβάνει τον αριθμό κάθε τύπου κάρτας στο χέρι του παίκτη.

2. **Action size**: Ο αριθμός των πιθανών ενεργειών που μπορεί να λάβει ο πράκτορας. Στο παιχνίδι αυτό, το action size είναι επίσης 8, που αντιστοιχεί στις ενέργειες που μπορεί να κάνει ο παίκτης.

3. **Learning rate (α)**: Ο ρυθμός μάθησης για την ενημέρωση των βαρών του νευρωνικού δικτύου. Χρησιμοποιήθηκε τιμή 0.0001, που καθορίζει πόσο γρήγορα το μοντέλο προσαρμόζεται στις νέες πληροφορίες.

4. **Discount factor (γ)**: Ο συντελεστής έκπτωσης που καθορίζει τη σημασία των μελλοντικών ανταμοιβών. Χρησιμοποιήθηκε τιμή 0.95, που σημαίνει ότι οι μελλοντικές ανταμοιβές θεωρούνται σημαντικές αλλά όχι όσο οι άμεσες ανταμοιβές.

5. **Exploration rate (ε)**: Ο ρυθμός εξερεύνησης που καθορίζει την πιθανότητα ο πράκτορας να επιλέξει μια τυχαία ενέργεια αντί να ακολουθήσει την πολιτική του. Ξεκινά από 1.0 και μειώνεται κατά 0.995 μετά από κάθε βήμα εκπαίδευσης.

6. **Exploration Decay**: Ο ρυθμός μείωσης του exploration rate. Χρησιμοποιήθηκε τιμή 0.995, που σημαίνει ότι ο πράκτορας γίνεται σταδιακά λιγότερο πιθανό να επιλέξει τυχαίες ενέργειες όσο μαθαίνει από το περιβάλλον.

7. **Replay memory size**: Το μέγιστο μέγεθος της επαναληπτικής μνήμης όπου αποθηκεύονται οι εμπειρίες του πράκτορα. Χρησιμοποιήθηκε τιμή 2000, που επιτρέπει την αποθήκευση 2000 εμπειριών για μελλοντική εκπαίδευση.

8. **Batch size**: Το μέγεθος του δείγματος που χρησιμοποιείται σε κάθε βήμα εκπαίδευσης. Χρησιμοποιήθηκε τιμή 64, που σημαίνει ότι σε κάθε βήμα εκπαίδευσης, ο πράκτορας μαθαίνει από 64 τυχαίες εμπειρίες από την επαναληπτική μνήμη.

Με την ρύθμιση αυτών των παραμέτρων, ο αλγόριθμος DQN μπορεί να βελτιστοποιήσει την απόδοσή του και να επιτύχει υψηλά επίπεδα απόδοσης το περιβάλλον του παιχνιδιού.

5 ΚΕΦΑΛΑΙΟ 5ο : Ανάπτυξη Παιχνιδιού Με AI Agent

5.1 Ανάλυση του παιχνιδιού Dominion

Το **Dominion** είναι ένα δημοφιλές παιχνίδι καρτών στρατηγικής και deck-building, στο οποίο οι παίκτες χτίζουν το δικό τους κατάστρωμα καρτών (deck). Κατά τη διάρκεια του παιχνιδιού σκοπός είναι να συγκεντρώσουν τους περισσότερους πόντους νίκης (Victory Points). Οι παίκτες ξεκινούν με ένα μικρό και αδύναμο κατάστρωμα και σταδιακά το ενισχύουν αγοράζοντας νέες κάρτες από μια κοινή αγορά. Οι κάρτες αυτές μπορεί να είναι θησαυροί, δράσεις ή κάρτες νίκης.

- **Στόχος του παιχνιδιού :**

Να συγκεντρώσεις τους περισσότερους πόντους νίκης (Victory Points) μέχρι το τέλος του παιχνιδιού, το οποίο τελειώνει όταν έχουν εξαντληθεί συγκεκριμένες κάρτες από την αγορά.

- **Προετοιμασία παιχνιδιού:**

- Κάθε παίκτης ξεκινάει με ένα κατάστρωμα (deck) που περιλαμβάνει **7 κάρτες Copper** (χρησιμεύουν για να αγοράζεις άλλες κάρτες) και **3 κάρτες Estate** (κάθε κάρτα Estate δίνει 1 πόντο νίκης).

- Το κεντρικό παιχνίδι περιλαμβάνει μια κοινή **αγορά** καρτών, από την οποία οι παίκτες μπορούν να αγοράσουν κάρτες για να βελτιώσουν το κατάστρωμά τους.

- Στην αγορά υπάρχουν διάφοροι τύποι καρτών: **Treasure (Θησαυροί)**, **Victory (Νίκης)** και **Action (Δράσης)**.

- **Φάσεις του γύρου:**

Κάθε γύρος ενός παίκτη αποτελείται από τις εξής φάσεις:

1. **Action Phase (Φάση Δράσης)**:

    - Ο παίκτης μπορεί να παίξει **μία κάρτα δράσης** από το χέρι του (εφόσον έχει τέτοιες κάρτες). Κάθε κάρτα δράσης έχει κάποιο εφέ που επηρεάζει το παιχνίδι, όπως να τραβήξει επιπλέον κάρτες ή να αυξήσει τις ενέργειές του.

2. **Buy Phase (Φάση Αγοράς)**:

    - Ο παίκτης μπορεί να χρησιμοποιήσει τις **κάρτες θησαυρού** που έχει στο χέρι του για να αγοράσει **νέες κάρτες** από την αγορά. Οι κάρτες χωρίζονται σε :

        - **Θησαυροί (Treasure cards)**: Παρέχουν χρήματα για να αγοράζεις άλλες κάρτες.

        - **Κάρτες Νίκης (Victory cards)**: Δίνουν πόντους νίκης στο τέλος του παιχνιδιού, αλλά είναι "άχρηστες" κατά τη διάρκεια του παιχνιδιού.

- **Κάρτες Δράσης (Action cards)**: Παρέχουν επιπλέον δυνατότητες και στρατηγικές κατά τη διάρκεια του γύρου σου.

3. **Cleanup Phase (Φάση καθαρισμού)**:

   - Ο παίκτης πετάει όλες τις κάρτες από το χέρι του (τόσο αυτές που έπαιξε όσο και όσες του έχουν απομείνει) στη στοίβα απόρριψης (discard pile).

   - Στη συνέχεια τραβάει **5 νέες κάρτες** από το κατάστρωμά του για να ετοιμαστεί για τον επόμενο γύρο.

- **Τέλος του παιχνιδιού:**

Το παιχνίδι τελειώνει όταν εξαντληθεί η στοίβα των **Province** (μια από τις βασικές κάρτες νίκης) είτε εξαντληθούν **τρεις άλλες στοίβες** στην αγορά. Τότε, κάθε παίκτης μετρά τους πόντους νίκης που έχει συγκεντρώσει από τις κάρτες νίκης στο κατάστρωμά του.

- **Στρατηγικές :**

- **Κατασκευή Καταστρώματος (Deck Building)**: Οι παίκτες πρέπει να αγοράζουν κάρτες που θα ενισχύσουν το κατάστρωμά τους, ενώ ταυτόχρονα θα αποφεύγουν τις "άχρηστες" κάρτες που θα επιβραδύνουν τις ενέργειές τους.

- **Ισορροπία**: Οι παίκτες πρέπει να βρουν τη σωστή ισορροπία μεταξύ καρτών θησαυρού, δράσης και νίκης, ώστε να διατηρήσουν την απόδοσή τους καθ' όλη τη διάρκεια του παιχνιδιού.

Το **Dominion** είναι ένα παιχνίδι που επιτρέπει στους παίκτες να προσαρμόζουν τις στρατηγικές τους ανάλογα με τις κάρτες που υπάρχουν στην αγορά, καθιστώντας το ιδιαίτερα ενδιαφέρον για παιχνίδι με φίλους ή σε τουρνουά.

5.2 Σχεδιασμός του παιχνιδιού

Το παιχνίδι αναπτύσσεται χρησιμοποιώντας τη γλώσσα προγραμματισμού Python και περιλαμβάνει τη χρήση ενός πράκτορα τεχνητής νοημοσύνης (AI Agent) που βασίζεται σε αλγόριθμο μη επιβλεπόμενης μάθησης. Το παιχνίδι αποτελείται από κάρτες με διάφορες ιδιότητες, όπως κάρτες δράσης, θησαυρού και νίκης.

5.3 Χρήση Αλγορίθμου Μη Εποπτευόμενης Μάθησης

Ο πράκτορας χρησιμοποιεί τον αλγόριθμο K-Means για την κατηγοριοποίηση των καταστάσεων του παιχνιδιού και την εφαρμογή της ενισχυτικής μάθησης για να βελτιστοποιήσει τις ενέργειές του.

5.4 Ανάλυση του κώδικα

Ο κώδικας υλοποιεί ένα παιχνίδι καρτών χρησιμοποιώντας την Python, με έναν πράκτορα τεχνητής νοημοσύνης που μαθαίνει από τις ενέργειές του μέσω ενισχυτικής μάθησης. Ακολουθεί αναλυτική περιγραφή του κάθε κομματιού του κώδικα:

Εισαγωγές και Αρχικοποίηση:

```
import os
import random
import numpy as np
import matplotlib.pyplot as plt
from sklearn.cluster import KMeans
import pickle
from collections import deque
import torch
import torch.nn as nn
import torch.optim as optim
```

Ο κώδικας ξεκινάει με τις απαραίτητες εισαγωγές βιβλιοθηκών, όπως η **OS** για λειτουργίες συστήματος, η **RANDOM** για τυχαίους αριθμούς, η **NUMPY** για αριθμητικές λειτουργίες, η **MATPLOTLIB** για γραφήματα, η **SKLEARN** για μηχανική μάθηση, η **PICKLE** για αποθήκευση δεδομένων, η **COLLECTION** για δομές δεδομένων, και η **TORCH** για την υλοποίηση του νευρωνικού δικτύου.

5.4.1 Δημιουργία καρτών

```
class Card:
    def __init__(self, name, card_type, cost, treasure_value, victory_points):
        self.name = name
        self.card_type = card_type
        self.cost = cost
        self.treasure_value = treasure_value
        self.victory_points = victory_points
```

```
    def __repr__(self):
        return f"{self.name} ({self.card_type}) with cost {self.cost}, treasure value {self.treasure_value} and
victory points {self.victory_points}"
```

Η κλάση Card αντιπροσωπεύει μια κάρτα με όνομα, τύπο (π.χ., "Actions", "Treasure", "Victory"), κόστος, αξία θησαυρού και πόντους νίκης.

5.4.2 Κάρτες Δράσης (action cards): Village και Smithy

```
class Village(Card):
    def __init__(self):
        super().__init__("Village", "Action", 3, 0, 0)

    def play_effect(self, player):
        print("Playing Village. Draw 1 more card and gain two more actions.")
        player.draw_additional_cards(1)
        player.actions += 2
        print(f"Player actions after Village: {player.actions}")


class Smithy(Card):
    def __init__(self):
        super().__init__("Smithy", "Action", 4, 0, 0)

    def play_effect(self, player):
        print("Playing Smithy. Draw 3 more cards.")
        player.draw_additional_cards(3)
        print(f"Player hand after Smithy: {[card.name for card in player.players_hand]}")
```

Οι κλάσεις Village και Smithy είναι εξειδικευμένες κάρτες που κληρονομούν από την Card και έχουν επιπλέον λειτουργικότητα όταν παίζονται.

5.4.3 Δημιουργία Λεξικού Καρτών

```
# Cards Dictionary
copper = Card("Copper", "Treasure", 0, 1, 0)
silver = Card("Silver", "Treasure", 3, 2, 0)
gold = Card("Gold", "Treasure", 6, 3, 0)
estate = Card("Estate", "Victory", 2, 0, 1)
duchy = Card("Duchy", "Victory", 5, 0, 3)
```

```
province = Card("Province", "Victory", 8, 0, 6)
smithy = Smithy()
village = Village()
```

Αυτό το τμήμα κώδικα δημιουργεί στιγμιότυπα των καρτών που θα χρησιμοποιηθούν στο παιχνίδι, Copper, Silver, Gold, Estate, Duchy, Province, Smithy, και Village.

5.4.4 Πίνακας Ανταμοιβών

```
# Reward table
reward_table = {
    "Copper": 0.5,
    "Silver": 0.5,
    "Gold": 5.0,
    "Estate": 1.0,
    "Duchy": 3.0,
    "Province": 10.0,
    "Smithy": 4.0,
    "Village": 4.0
}
```

Ο πίνακας ανταμοιβών καθορίζει την αξία κάθε κάρτας σε πόντους που θα χρησιμοποιηθούν από τον πράκτορα AI για την εκμάθησή του.

5.4.5 Δημιουργία Τράπουλας

```
class Deck:
    def __init__(self):
        self.stacks = {
            "Copper": [copper] * 46,
            "Silver": [silver] * 40,
            "Gold": [gold] * 30,
            "Estate": [estate] * 24,
            "Duchy": [duchy] * 12,
            "Province": [province] * 12,
            "Smithy": [smithy] * 10,
            "Village": [village] * 10
        }

    def draw_card(self, card_name):
        if self.stacks[card_name]:
            return self.stacks[card_name].pop(0)
```

```
        else:
            print(f"No cards left in stack for {card_name}")
            return None

    def is_game_over(self):

    province_empty = len(self.stacks["Province"]) == 0

    empty_stacks = sum(1 for stack in self.stacks.values() if len(stack) == 0)

    game_over = province_empty or empty_stacks >= 3
    print(f"Checking if game is over: {game_over}")
    return game_over
```

Η κλάση Deck δημιουργεί την τράπουλα του παιχνιδιού με συγκεκριμένο αριθμό καρτών και περιέχει μεθόδους για να τραβάει κάρτες και να ελέγχει αν το παιχνίδι έχει τελειώσει. Η κλάση is_game_over κάνει έλεγχο για το αν, βάση των κανόνων, το παιχνίδι έχει τελειώσει

5.4.6 Δημιουργία Παικτών

```
class Player:
    def __init__(self):
        self.players_deck = []
        self.players_hand = []
        self.discard_pile = []
        self.actions = 1
        self.buys = 1
        self.treasure = 0

        for i in range(7):
            self.players_deck.append(copper)
        for i in range(3):
            self.players_deck.append(estate)
        random.shuffle(self.players_deck)
        self.draw_hand()

    def draw_hand(self):
        while len(self.players_hand) < 5:
            if not self.players_deck:
```

```
            self.players_deck, self.discard_pile = self.discard_pile, []
            random.shuffle(self.players_deck)
        if self.players_deck:
            self.players_hand.append(self.players_deck.pop())
        print(f"Player hand after drawing: {[card.name for card in self.players_hand]}")

    def draw_additional_cards(self, num_cards):
        for i in range(num_cards):
            if not self.players_deck:
                self.players_deck, self.discard_pile = self.discard_pile, []
                random.shuffle(self.players_deck)
            if self.players_deck:
                self.players_hand.append(self.players_deck.pop())
        print(f"Player hand after drawing: {[card.name for card in self.players_hand]}")

    def reset_for_new_turn(self):
        self.actions = 1
        self.buys = 1
        self.treasure = 0
        self.discard_pile.extend(self.players_hand)
        self.players_hand = []
        self.draw_hand()

    def count_gold_cards(self):
        total_gold = self.players_deck.count(gold) + self.players_hand.count(gold) +
self.discard_pile.count(gold)
        return total_gold

    def count_victory_points(self):
        total_victory_points = sum(card.victory_points for card in self.players_deck)
        total_victory_points += sum(card.victory_points for card in self.players_hand)
        total_victory_points += sum(card.victory_points for card in self.discard_pile)
        return total_victory_points
```

Με την κλάση Player γίνεται αρχικοποίηση τον κάθε παίκτη με ένα προσωπικό deck, χέρι και στοίβα απόρριψης. Περιέχει μεθόδους για να τραβάει κάρτες, να ανανεώνει το χέρι για τον επόμενο γύρο, και να μετράει τις κάρτες χρυσού και τους πόντους νίκης.

5.4.7 Δημιουργία Του Πράκτορα DQN

```
class DQNAgent:
    def __init__(self, state_size, action_size, learning_rate=0.0001, discount_factor=0.95, exploration_rate=1.0,
exploration_decay=0.995, replay_memory_size=2000, batch_size=64):
```

```
        self.state_size = state_size
        self.action_size = action_size
        self.learning_rate = learning_rate
        self.discount_factor = discount_factor
        self.exploration_rate = exploration_rate
        self.exploration_decay = exploration_decay
        self.replay_memory = deque(maxlen=replay_memory_size)
        self.batch_size = batch_size
        self.device = torch.device("cuda" if torch.cuda.is_available() else "cpu")
        self.model = self._build_model().to(self.device)
        self.target_model = self._build_model().to(self.device)
        self.optimizer = optim.Adam(self.model.parameters(), lr=self.learning_rate)
        self.kmeans = KMeans(n_clusters=2)

    def _build_model(self):
        return nn.Sequential(
            nn.Linear(self.state_size, 64),
            nn.ReLU(),
            nn.Linear(64, 64),
            nn.ReLU(),
            nn.Linear(64, self.action_size)
        )

    def update_target_model(self):
        self.target_model.load_state_dict(self.model.state_dict())

    def choose_action(self, state):
        state = torch.FloatTensor(state).unsqueeze(0).to(self.device)
        if random.random() < self.exploration_rate:
            return random.randint(0, self.action_size - 1)
        else:
            with torch.no_grad():
                action = torch.argmax(self.model(state)).item()
        print(f"Chosen action: {action}")
        return action

    def store_experience(self, state, action, reward, next_state):
        self.replay_memory.append((state, action, reward, next_state))

    def replay_experience(self):
        if len(self.replay_memory) < self.batch_size:
            return
        batch = random.sample(self.replay_memory, self.batch_size)
        for state, action, reward, next_state in batch:
            state = torch.FloatTensor(state).to(self.device)
            next_state = torch.FloatTensor(next_state).to(self.device)
            action = torch.LongTensor([action]).to(self.device)
```

```
        reward = torch.FloatTensor([reward]).to(self.device)

        current_q = self.model(state)[action]
        next_q = reward + self.discount_factor * self.target_model(next_state).max().item()

        loss = nn.MSELoss()(current_q, next_q)
        self.optimizer.zero_grad()
        loss.backward()
        self.optimizer.step()

    def decay_exploration(self):
        self.exploration_rate *= self.exploration_decay
        print(f"Decayed exploration rate: {self.exploration_rate}")


    def learn(self, state, action, reward, next_state):
        self.store_experience(state, action, reward, next_state)
        self.replay_experience()
        self.exploration_rate *= self.exploration_decay
        self.exploration_rate = max(0.01, self.exploration_rate)
        self.update_target_model()

    def get_state(self, player):
        state = [0] * self.state_size
        card_names = [card.name for card in player.players_hand]
        state[0] = card_names.count("Copper")
        state[1] = card_names.count("Silver")
        state[2] = card_names.count("Gold")
        state[3] = card_names.count("Estate")
        state[4] = card_names.count("Duchy")
        state[5] = card_names.count("Province")
        state[6] = card_names.count("Smithy")
        state[7] = card_names.count("Village")
        return state

    def get_reward(self, player, action):
        total_reward = 0
        for card in player.players_deck + player.players_hand + player.discard_pile:
            total_reward += reward_table.get(card.name, 0)
        return total_reward

    def analyze_performance(self):
        if not self.replay_memory:
            return
        successful_trades = [trade for trade in self.replay_memory if trade[2] > 0]
        gold_trades = [trade for trade in successful_trades if trade[1] == 2]
        province_trades = [trade for trade in successful_trades if trade[1] == 4]
```

```
    print("Analysis of Performance:")
    print(f"Total successful trades: {len(successful_trades)}")
    print(f"Gold trades: {len(gold_trades)}")
    print(f"Province trades: {len(province_trades)}")

def train_kmeans(self):
    if len(self.replay_memory) < self.batch_size:
        return
    states = [experience[0] for experience in self.replay_memory]
    self.kmeans.fit(states)

def save_model(self, filename='dqn_model.pth'):
    torch.save(self.model.state_dict(), filename)

def load_model(self, filename='dqn_model.pth'):
    try:
        self.model.load_state_dict(torch.load(filename))
        self.target_model.load_state_dict(torch.load(filename))
    except FileNotFoundError:
        print(f"No model file found under the name {filename}. Starting fresh.")
```

Η κλάση DQNAgent υλοποιεί έναν πράκτορα ενισχυτικής μάθησης χρησιμοποιώντας Deep Q-Network (DQN). Η κλάση περιλαμβάνει:

- **Αρχικοποίηση**: Καθορίζει το μέγεθος της κατάστασης και της δράσης, τους ρυθμούς μάθησης και έκπτωσης, τον ρυθμό εξερεύνησης, και την επαναληπτική μνήμη.
- **Δημιουργία Μοντέλου**: Δημιουργεί το νευρωνικό δίκτυο που θα χρησιμοποιηθεί για να προβλέψει τις Q-τιμές.
- **Επιλογή Δράσης**: Επιλέγει μια δράση είτε τυχαία (εξερεύνηση) είτε βάσει των προβλέψεων του μοντέλου (εκμετάλλευση).
- **Αποθήκευση Εμπειρίας**: Αποθηκεύει εμπειρίες από τις αλληλεπιδράσεις του πράκτορα με το περιβάλλον.
- **Εκπαίδευση μέσω Εμπειρίας**: Αναπαράγει εμπειρίες από την επαναληπτική μνήμη για να εκπαιδεύσει το νευρωνικό δίκτυο.
- **Μάθηση και Εκπαίδευση**: Εκπαιδεύει το μοντέλο και ενημερώνει τις παραμέτρους του.
- **Υπολογισμός Κατάστασης και Ανταμοιβής**: Υπολογίζει την τρέχουσα κατάσταση και την ανταμοιβή του παίκτη.

- **Ανάλυση Απόδοσης**: Αναλύει τις επιδόσεις του πράκτορα βάσει των εμπειριών του.
- **Αποθήκευση και Φόρτωση Μοντέλου**: Αποθηκεύει και φορτώνει το εκπαιδευμένο μοντέλο.

5.4.8 Δημιουργία Παιχνιδιού

```
class Game:
    def __init__(self, num_players, agent, input_size, output_size):
        if num_players < 2:
            raise ValueError("Number of players must be at least 2.")
        self.players = [Player() for i in range(num_players)]
        self.current_player = 0
        self.deck = Deck()
        self.round_count = 1
        self.agent = agent
        self.input_size = input_size
        self.output_size = output_size
        self.reward_history = []
        self.total_victory_points_history = []

        self.player1_victory_points = []
        self.player2_victory_points = []
```

Με την κλάση Game γίνεται η αρχικοποίηση ένα παιχνίδι με συγκεκριμένο αριθμό παικτών, έναν πράκτορα (agent) και το μέγεθος των εισόδων και εξόδων. Περιέχει επίσης λίστες για την καταγραφή της ιστορίας ανταμοιβών και των συνολικών πόντων νίκης των παικτών.

5.4.9 Λήψη Κατάστασης του Παιχνιδιού

```
    def get_game_state(self):
        player = self.players[self.current_player]
        player_hand_cards = [card.name for card in player.players_hand]
        player_deck_cards = [card.name for card in player.players_deck]
        return player_hand_cards + player_deck_cards
```

Η μέθοδος get_game_state επιστρέφει την τρέχουσα κατάσταση του παιχνιδιού, συμπεριλαμβανομένων των καρτών στο χέρι και στην τράτα του τρέχοντος παίκτη.

5.4.10 Υπολογισμός Κατάστασης

```

```

```
  def calculate_state(self):
      player = self.players[self.current_player]
      return self.agent.get_state(player)
```

Η μέθοδος calculate_state υπολογίζει την κατάσταση του παιχνιδιού χρησιμοποιώντας τη μέθοδο get_state του πράκτορα.

5.4.11 Φάση Δράσης (action phase)

```
def action_phase(self, player):

    print("\n--- ACTION PHASE ---")

    print(f"Player {self.current_player + 1}'s hand: {[card.name for card in player.players_hand]}")

    print(f"Player {self.current_player + 1} has {player.actions} actions left.")

    while player.actions > 0:
      action_cards = [card for card in player.players_hand if card.card_type == "Action"]
    if not action_cards:
      break
    card_to_play = None
    for card in action_cards:
       if card.name == "Village" or card.name == "Smithy":
      card_to_play = card
      break
    if card_to_play is None:
       card_to_play = action_cards[0]
    player.players_hand.remove(card_to_play)
    player.actions -= 1
    print(f"Player {self.current_player + 1} is playing {card_to_play.name}.")
    card_to_play.play_effect(player)
    player.discard_pile.append(card_to_play)
      print(f"Remaining actions: {player.actions}")
      print(f"Current hand after action: {[card.name for card in player.players_hand]}")

    action_cards = [card for card in player.players_hand if card.card_type == "Action"]

    if player.actions <= 0 or not action_cards:
       break
```

Η μέθοδος action_phase επιτρέπει στον παίκτη να παίξει κάρτες δράσης από το χέρι του. Αν ο παίκτης έχει κάρτες τύπου "Village" ή "Smithy", τις παίζει και εκτελεί τα εφέ τους.

5.4.12 Φάση Αγοράς (buy phase)

```
def buy_phase(self, player):
    print("\n--- BUY PHASE ---")


    player.treasure = sum(card.treasure_value for card in player.players_hand if card.card_type == "Treasure")
    print(f"Player {self.current_player + 1} has {player.treasure} treasure to spend.")

    print("Available cards for purchase:")
    for card_name, stack in self.deck.stacks.items():
      if stack:
        print(f"{card_name}: Cost = {stack[0].cost}, Remaining = {len(stack)}")

    while player.buys > 0 and player.treasure > 0:

 affordable_cards = [(name, card[0]) for name, card in self.deck.stacks.items() if card and card[0].cost <=
player.treasure]
      if not affordable_cards:
        print("No affordable cards available.")
        break



      chosen_card_name, chosen_card = random.choice(affordable_cards)
      player.discard_pile.append(chosen_card)
      player.treasure -= chosen_card.cost
      player.buys -= 1



      self.deck.stacks[chosen_card_name].pop(0)
      print(f"Player {self.current_player + 1} bought {chosen_card.name} for {chosen_card.cost} cost.
{len(self.deck.stacks[chosen_card_name])} remaining.")

   print("End of buy phase")
```

Η μέθοδος buy_phase επιτρέπει στον παίκτη να αγοράζει κάρτες από την τράπουλα χρησιμοποιώντας τις κάρτες θησαυρού που έχει στο χέρι του.

5.4.13 Φάση Καθαρισμού (cleanup phase)

```
def cleanup_phase(self, player):
    print("\n--- CLEANUP PHASE ---")
    print(f"Player {self.current_player + 1} discarding hand: {[card.name for card in player.players_hand]}")
    player.reset_for_new_turn()
    print(f"New hand for Player {self.current_player + 1}: {[card.name for card in player.players_hand]}")
```

Η μέθοδος cleanup_phase ανανεώνει το χέρι του παίκτη για τον επόμενο γύρο.

5.4.14 Παίξιμο Γύρου

```
def play_turn(self):
    print(f"\n\n--- ROUND {self.round_count}, Player {self.current_player + 1}'s Turn ---")
    player = self.players[self.current_player]
    player.draw_hand()

    # Action Phase
    self.action_phase(player)

    # Buy Phase
    self.buy_phase(player)

    # Cleanup Phase (this should be the last phase)
    self.cleanup_phase(player)

    # Move to the next player
    self.current_player = (self.current_player + 1) % len(self.players)
    self.round_count += 1

    self.player1_victory_points.append(self.players[0].count_victory_points())
    self.player2_victory_points.append(self.players[1].count_victory_points())
```

Η μέθοδος play_turn εκτελεί όλες τις φάσεις ενός γύρου για τον τρέχοντα παίκτη.

5.4.15 Υπολογισμός Συνολικών Πόντων Νίκης

```
def calculate_total_victory_points(self):
        total_victory_points = []
        for player in self.players:
            victory_points = sum(card.victory_points for card in player.players_deck)
            victory_points += sum(card.victory_points for card in player.players_hand)
            victory_points += sum(card.victory_points for card in player.discard_pile)
            total_victory_points.append(victory_points)
        return total_victory_points
```

Η μέθοδος calculate_total_victory_points υπολογίζει τους συνολικούς πόντους νίκης για κάθε παίκτη, λαμβάνοντας υπόψη τις κάρτες στο deck του παίχτη, στο χέρι και στη στοίβα απόρριψης.

5.4.16 Παίξιμο Παιχνιδιού

```
def play_game(self
        self.player1_victory_points = []
        self.player2_victory_points = []

        while not self.deck.is_game_over():
            self.play_turn()

        total_victory_points = self.calculate_total_victory_points()


        self.reward_history.append(total_victory_points)
        self.total_victory_points_history.append(total_victory_points)
        print(f"\n--- Game Analysis  ---")
        print(f"Final Victory Points: {total_victory_points}")

        for i, player in enumerate(self.players):
            state = self.agent.get_state(player)
            action = self.agent.choose_action(state)
            reward = self.agent.get_reward(player, action)
            next_state = self.agent.get_state(player)
            self.agent.learn(state, action, reward, next_state)

        self.agent.train_kmeans()
        self.agent.analyze_performance()
        self.agent.save_model()
        return total_victory_points
```

Η μέθοδος play_game εκτελεί ένα πλήρες παιχνίδι, το οποίο περιλαμβάνει πολλούς γύρους, μέχρι να ολοκληρωθεί το παιχνίδι. Υπολογίζει τους συνολικούς πόντους νίκης και εκπαιδεύει τον πράκτορα AI.

5.4.17 Αριθμός Καρτών Province

```
    def count_province_cards(self, player):
        total_province = player.players_deck.count(province) + player.players_hand.count(province) +
player.discard_pile.count(province)
        return total_province
```

Η μέθοδος count_province_cards υπολογίζει τον αριθμό των καρτών τύπου Province που έχει ο παίκτης στο προσωπικό του deck, στο χέρι και στη στοίβα απόρριψης.

5.4.18 Εκτέλεση Και Ανάλυση Εκπαίδευσης

Ορισμός Αριθμού Παικτών και Μεγεθών Εισόδου/Εξόδου

```
num_players = 2
input_size = 8
output_size = 8

agent = DQNAgent(input_size, output_size)
agent.load_model()
```

Ορίζονται οι παράμετροι για τον αριθμό των παικτών και τα μεγέθη εισόδου και εξόδου για τον πράκτορα. Στη συνέχεια, δημιουργείται ο πράκτορας DQN και φορτώνεται το αποθηκευμένο μοντέλο του.

5.4.19 Δημιουργία Προσωρινής Παρτίδας Για Εκπαίδευση K-Means

```
# Create a temporary game instance to fit KMeans
temp_game = Game(num_players, agent, input_size, output_size)
initial_states = [agent.get_state(player) for player in temp_game.players]
agent.kmeans.fit(initial_states)
```

5.4.20 Εκτέλεση Παιχνιδιού

```
game = Game(num_players, agent, input_size, output_size)
```

Δημιουργείται μια νέα παρτίδα του παιχνιδιού με τους προκαθορισμένους παίκτες και τον πράκτορα.

5.4.21 Ανάλυση Ανταμοιβών

```
def backtest_rewards(reward_history):
    avg_reward = np.mean(reward_history, axis=0)
    total_reward = np.sum(reward_history, axis=0)
    max_reward = np.max(reward_history, axis=0)
    min_reward = np.min(reward_history, axis=0)
    print("Backtesting Results:")
    print("Average Reward:", avg_reward)
```

Η μέθοδος backtest_rewards αναλύει την ιστορία των ανταμοιβών που συγκεντρώθηκαν κατά τη διάρκεια των παιχνιδιών. Υπολογίζει τη μέση, συνολική, μέγιστη και ελάχιστη ανταμοιβή.

5.4.22 Εκτέλεση Γύρων Εκπαίδευσης

```
num_training_rounds = 30

all_player1_victory_points = []
all_player2_victory_points = []

for i in range(num_training_rounds):
    total_victory_points = game.play_game()
    winner_index = total_victory_points.index(max(total_victory_points))
    print(f"Game {i + 1}/{num_training_rounds}: Player {winner_index + 1} wins with
{max(total_victory_points)} total victory points.")

    all_player1_victory_points.extend(game.player1_victory_points)
    all_player2_victory_points.extend(game.player2_victory_points)
```

Σε αυτό το κομμάτι κώδικα, εκτελούνται 30 γύροι εκπαίδευσης του πράκτορα. Σε κάθε γύρο, εκτελείται ένα πλήρες παιχνίδι και οι συνολικοί πόντοι νίκης κάθε παίκτη καταγράφονται. Εκτυπώνεται ο νικητής κάθε παιχνιδιού.

5.4.23 Εμφάνιση Αποτελεσμάτων Σε Γράφημα

```
# Plotting the combined results for all games
plt.figure()
plt.plot(all_player1_victory_points, label='Player 1')
plt.plot(all_player2_victory_points, label='Player 2')
plt.xlabel('Round Number')
plt.ylabel('Victory Points')
plt.title('Victory Points Over Rounds for All Games')
plt.legend()
plt.show()

backtest_rewards(game.reward_history)
```

Τέλος, δημιουργούνται γραφήματα για να απεικονιστούν οι πόντοι νίκης των παικτών σε όλους τους γύρους του παιχνιδιού. Η ανάλυση ανταμοιβών εκτελείται και τα αποτελέσματα εκτυπώνονται.

5.5 Συνολική Ανάλυση Του Κώδικα

Ο κώδικας υλοποιεί ένα παιχνίδι καρτών με τη χρήση Python, όπου οι παίκτες αγοράζουν κάρτες και προσπαθούν να συλλέξουν πόντους νίκης. Ένας πράκτορας τεχνητής νοημοσύνης, χρησιμοποιώντας αλγόριθμο DQN, μαθαίνει να παίζει το παιχνίδι μέσω ενισχυτικής μάθησης, χρησιμοποιώντας εμπειρίες από τις αλληλεπιδράσεις του με το περιβάλλον για να βελτιώνει τις αποφάσεις του. Κατά τη διάρκεια του παιχνιδιού, οι παίκτες παίζουν κάρτες δράσης και αγοράζουν νέες κάρτες για να αυξήσουν τους πόντους νίκης τους. Οι επιδόσεις του πράκτορα αναλύονται και εμφανίζονται μέσω γραφημάτων.

5.6 Λεπτομέρειες παραμέτρων αλγόριθμου K-Means

Ο αλγόριθμος KMeans χρησιμοποιήθηκε για την κατηγοριοποίηση των καταστάσεων του παιχνιδιού. Οι κύριες παράμετροι είναι:

1. **Number of Clusters (k)**: Ο αριθμός των clusters που θα δημιουργηθούν. Χρησιμοποιήθηκε τιμή 2, καθώς στόχος ήταν να διαχωριστούν οι καταστάσεις του παιχνιδιού σε δύο κύριες κατηγορίες.

2. **Max Iterations**: Ο μέγιστος αριθμός επαναλήψεων που θα εκτελέσει ο αλγόριθμος για να συγκλίνει. Χρησιμοποιήθηκε η προεπιλεγμένη τιμή 300, που είναι επαρκής για τη σύγκλιση σε πολλές περιπτώσεις.

3. **Installation Method**: Η μέθοδος για την αρχικοποίηση των κεντρικών σημείων των clusters. Χρησιμοποιήθηκε η μέθοδος 'KMeans++', που βελτιώνει τη σύγκλιση επιλέγοντας πιο κατάλληλα αρχικά κεντρικά σημεία.

6 Συμπεράσματα

Η παρούσα διπλωματική εργασία επικεντρώθηκε στην ανάπτυξη ενός απλοποιημένου επιτραπέζιου παιχνιδιού τύπου Dominion χρησιμοποιώντας τη γλώσσα προγραμματισμού Python, καθώς και στην υλοποίηση ενός αλγορίθμου τεχνητής νοημοσύνης (AI) για έναν αυτόνομο παίκτη μέσω μη επιβλεπόμενης μάθησης. Τα αποτελέσματα που προέκυψαν από την έρευνα και την ανάπτυξη του παιχνιδιού είναι αξιοσημείωτα και παρέχουν πολύτιμες γνώσεις τόσο στον τομέα των παιχνιδιών όσο και στην εφαρμογή των αλγορίθμων μηχανικής μάθησης.

6.1 Επίτευξη των στόχων

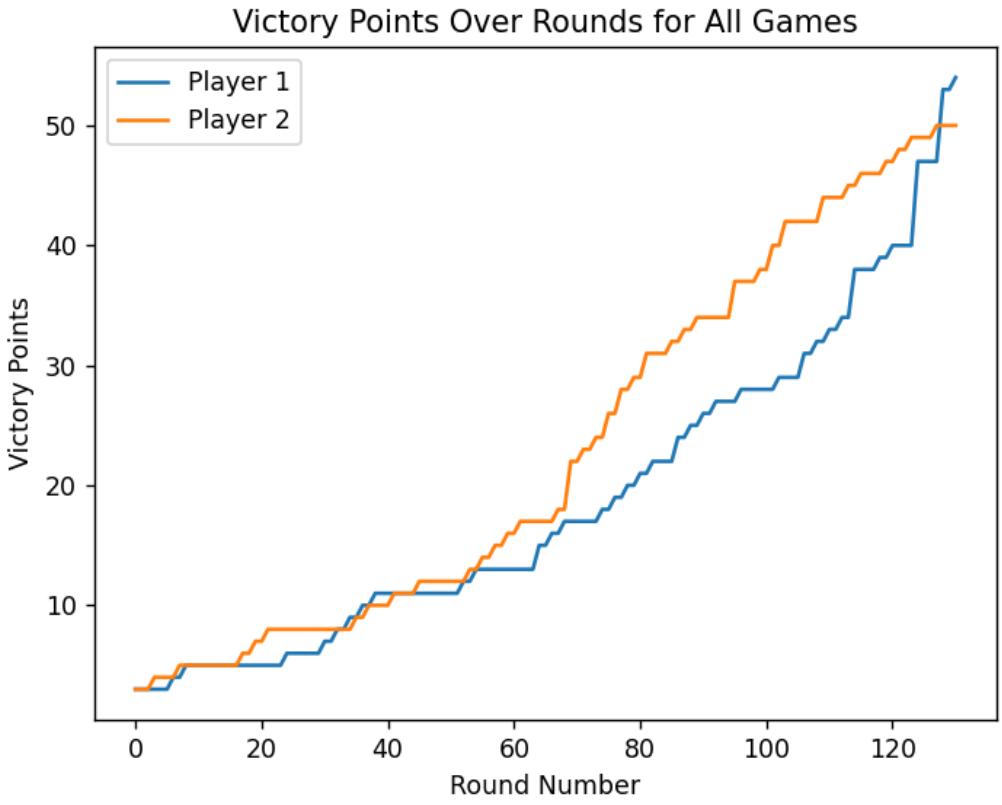
Η εργασία κατάφερε να επιτύχει τους βασικούς στόχους που είχαν τεθεί:

- **Ανάπτυξη Παιχνιδιού σε Python:** Το παιχνίδι δημιουργήθηκε σε γλώσσα περιλαμβάνοντας όλες τις βασικές λειτουργίες ενός επιτραπέζιου παιχνιδιού καρτών. Οι παίκτες μπορούν να παίζουν κάρτες δράσης και να αγοράζουν κάρτες ακολουθώντας τους κανόνες του παιχνιδιού Dominion.
- **Υλοποίηση AI Agent:** Ο πράκτορας τεχνητής νοημοσύνης (AI Agent) υλοποιήθηκε χρησιμοποιώντας τον αλγόριθμο Deep Q-Network (DQN). Ο πράκτορας κατάφερε να μάθει μέσω ενισχυτικής μάθησης, βελτιώνοντας τις στρατηγικές του και τις αποφάσεις του κατά τη διάρκεια του παιχνιδιού.
- **Εφαρμογή Αλγορίθμου KMeans:** Ο αλγόριθμος KMeans χρησιμοποιήθηκε για την κατηγοριοποίηση των καταστάσεων του παιχνιδιού. Αυτό επέτρεψε την αποτελεσματικότερη εκπαίδευση του πράκτορα και την καλύτερη κατανόηση των καταστάσεων του παιχνιδιού.
- **Ανάλυση Απόδοσης του Πράκτορα:** Η απόδοση του πράκτορα αναλύθηκε μέσω πολλαπλών γύρων παιχνιδιού. Τα αποτελέσματα έδειξαν ότι ο πράκτορας βελτίωσε σημαντικά την απόδοσή του, επιτυγχάνοντας υψηλότερα σκορ και καλύτερες στρατηγικές αποφάσεις με την πάροδο του χρόνου.

6.2 Ανάλυση διαγράμματος

Στο διάγραμμα απεικονίζονται οι πόντοι νίκης των δύο παικτών (Player 1 και Player 2) σε σχέση με τον αριθμό των γύρων του παιχνιδιού. Οι καμπύλες δείχνουν την πρόοδο των παικτών καθώς προχωρούν οι γύροι.

Διάγραμμα 1: Πρόοδος Μάθησης



Αυτό το διάγραμμα δείχνει τη συνολική βελτίωση και απόδοση των παικτών κατά τη διάρκεια του παιχνιδιού. Η διακύμανση στους πόντους νίκης και η τελική υπεροχή του Player 1 υποδηλώνουν ότι οι στρατηγικές που εφαρμόστηκαν από τον πράκτορα τεχνητής νοημοσύνης είχαν θετικό αποτέλεσμα

6.3 Ανάλυση γύρων

6.3.1 Αρχικοί γύροι (Rounds 1-6):

--- ROUND 1, Player 1's Turn ---

Player hand after drawing: ['Estate', 'Copper', 'Copper', 'Copper', 'Estate']

--- ACTION PHASE ---

Player 1's hand: ['Estate', 'Copper', 'Copper', 'Copper', 'Estate']

Player 1 has 1 actions left.

--- BUY PHASE ---

Player 1 has 3 treasure to spend.

Available cards for purchase:

Copper: Cost = 0, Remaining = 46

Silver: Cost = 3, Remaining = 40

Gold: Cost = 6, Remaining = 30

Estate: Cost = 2, Remaining = 24

Duchy: Cost = 5, Remaining = 12

Province: Cost = 8, Remaining = 12

Smithy: Cost = 4, Remaining = 10

Village: Cost = 3, Remaining = 10

Player 1 bought Silver for 3 cost. 39 remaining.

--- CLEANUP PHASE ---

Player 1 discarding hand: ['Estate', 'Copper', 'Copper', 'Copper', 'Estate']

Player hand after drawing: ['Copper', 'Copper', 'Copper', 'Estate', 'Copper']

New hand for Player 1: ['Copper', 'Copper', 'Copper', 'Estate', 'Copper']

Checking if game is over: False

--- ROUND 2, Player 2's Turn ---

Player hand after drawing: ['Estate', 'Copper', 'Copper', 'Estate', 'Copper']

--- ACTION PHASE ---

Player 2's hand: ['Estate', 'Copper', 'Copper', 'Estate', 'Copper']

Player 2 has 1 actions left.

--- BUY PHASE ---

Player 2 has 3 treasure to spend.

Available cards for purchase:

Copper: Cost = 0, Remaining = 46

Silver: Cost = 3, Remaining = 39

Gold: Cost = 6, Remaining = 30

Estate: Cost = 2, Remaining = 24

Duchy: Cost = 5, Remaining = 12

Province: Cost = 8, Remaining = 12

Smithy: Cost = 4, Remaining = 10

Village: Cost = 3, Remaining = 10

Player 2 bought Silver for 3 cost 38 remaining.

--- CLEANUP PHASE ---

Player 2 discarding hand: ['Estate', 'Copper', 'Copper', 'Estate', 'Copper']

Player hand after drawing: ['Copper', 'Copper', 'Copper', 'Copper', 'Estate']

New hand for Player 2: ['Copper', 'Copper', 'Copper', 'Copper', 'Estate']

Checking if game is over: False

--- ROUND 3, Player 1's Turn ---

Player hand after drawing: ['Copper', 'Copper', 'Copper', 'Estate', 'Copper']

--- ACTION PHASE ---

Player 1's hand: ['Copper', 'Copper', 'Copper', 'Estate', 'Copper']

Player 1 has 1 actions left.

--- BUY PHASE ---

Player 1 has 4 treasure to spend.

Available cards for purchase:

Copper: Cost = 0, Remaining = 46

Silver: Cost = 3, Remaining = 38

Gold: Cost = 6, Remaining = 30

Estate: Cost = 2, Remaining = 24

Duchy: Cost = 5, Remaining = 12

Province: Cost = 8, Remaining = 12

Smithy: Cost = 4, Remaining = 10

Village: Cost = 3, Remaining = 10

Player 1 bought Silver for 3 cost . 37 remaining.

--- CLEANUP PHASE ---

Player 1 discarding hand: ['Copper', 'Copper', 'Copper', 'Estate', 'Copper']

Player hand after drawing: ['Copper', 'Copper', 'Silver', 'Copper', 'Estate']

New hand for Player 1: ['Copper', 'Copper', 'Silver', 'Copper', 'Estate']

Checking if game is over: False

--- ROUND 4, Player 2's Turn ---

Player hand after drawing: ['Copper', 'Copper', 'Copper', 'Copper', 'Estate']

--- ACTION PHASE ---

Player 2's hand: ['Copper', 'Copper', 'Copper', 'Copper', 'Estate']

Player 2 has 1 actions left.

--- BUY PHASE ---

Player 2 has 4 treasure to spend.

Available cards for purchase:

Copper: Cost = 0, Remaining = 46

Silver: Cost = 3, Remaining = 37

Gold: Cost = 6, Remaining = 30

Estate: Cost = 2, Remaining = 24

Duchy: Cost = 5, Remaining = 12

Province: Cost = 8, Remaining = 12

Smithy: Cost = 4, Remaining = 10

Village: Cost = 3, Remaining = 10

Player 2 bought Village for 3 cost. 9 remaining.

--- CLEANUP PHASE ---

Player 2 discarding hand: ['Copper', 'Copper', 'Copper', 'Copper', 'Estate']

Player hand after drawing: ['Copper', 'Estate', 'Silver', 'Copper', 'Village']

New hand for Player 2: ['Copper', 'Estate', 'Silver', 'Copper', 'Village']

Checking if game is over: False

--- ROUND 5, Player 1's Turn ---

Player hand after drawing: ['Copper', 'Copper', 'Silver', 'Copper', 'Estate']

--- ACTION PHASE ---

Player 1's hand: ['Copper', 'Copper', 'Silver', 'Copper', 'Estate']

Player 1 has 1 actions left.

--- BUY PHASE ---

Player 1 has 5 treasure to spend.

Available cards for purchase:

Copper: Cost = 0, Remaining = 46

Silver: Cost = 3, Remaining = 37

Gold: Cost = 6, Remaining = 30

Estate: Cost = 2, Remaining = 24

Duchy: Cost = 5, Remaining = 12

Province: Cost = 8, Remaining = 12

Smithy: Cost = 4, Remaining = 10

Village: Cost = 3, Remaining = 9

Player 1 bought Estate for 2 cost. 23 remaining.

--- CLEANUP PHASE ---

Player 1 discarding hand: ['Copper', 'Copper', 'Silver', 'Copper', 'Estate']

Player hand after drawing: ['Estate', 'Copper', 'Copper', 'Copper', 'Estate']

New hand for Player 1: ['Estate', 'Copper', 'Copper', 'Copper', 'Estate']

Checking if game is over: False

--- ROUND 6, Player 2's Turn ---

Player hand after drawing: ['Copper', 'Estate', 'Silver', 'Copper', 'Village']

--- ACTION PHASE ---

Player 2's hand: ['Copper', 'Estate', 'Silver', 'Copper', 'Village']

Player 2 has 1 actions left.

Player 2 is playing Village.

Playing Village. Draw 1 more card and gain two more actions.

Player hand after drawing 1 more cards: ['Copper', 'Estate', 'Silver', 'Copper', 'Copper']

Player actions after Village: 2

Remaining actions: 2

Current hand after action: ['Copper', 'Estate', 'Silver', 'Copper', 'Copper']

--- BUY PHASE ---

Player 2 has 5 treasure to spend.

Available cards for purchase:

Copper: Cost = 0, Remaining = 46

Silver: Cost = 3, Remaining = 37

Gold: Cost = 6, Remaining = 30

Estate: Cost = 2, Remaining = 23

Duchy: Cost = 5, Remaining = 12

Province: Cost = 8, Remaining = 12

Smithy: Cost = 4, Remaining = 10

Village: Cost = 3, Remaining = 9

Player 2 bought Copper for 0 cost. 45 remaining.

--- CLEANUP PHASE ---

Player 2 discarding hand: ['Copper', 'Estate', 'Silver', 'Copper', 'Copper']

Player hand after drawing: ['Copper', 'Estate', 'Estate', 'Copper', 'Copper']

New hand for Player 2: ['Copper', 'Estate', 'Estate', 'Copper', 'Copper']

Checking if game is over: False

Οι πρώτοι γύροι δείχνουν ότι και οι δύο παίκτες (ή agents) επιλέγουν σχετικά απλές αγορές και δράσεις. Σε αυτούς τους γύρους, ο agent:

- **Εστιάζει κυρίως στην απόκτηση Silver** (γύροι 1-3), ενισχύοντας έτσι την οικονομική του ικανότητα. Το Silver είναι μια ασφαλής επένδυση στην αρχή για να αυξήσει την αξία των θησαυρών του, χωρίς να ρισκάρει μεγάλες αγορές.
- **Αγορές χαμηλού κόστους** όπως το Village και το Estate, ενισχύουν την ποικιλία στο χέρι του, χωρίς ακόμα να επιδιώκει μεγάλα κέρδη.

Αυτό δείχνει ότι ο agent δεν έχει ακόμα αναπτύξει περίπλοκες στρατηγικές ή συνδυασμούς. Αντίθετα, επικεντρώνεται στο να δημιουργήσει μια σταθερή βάση πόρων.

6.3.2 Μέσο παιχνίδι (Rounds 35-42):

--- ROUND 35, Player 1's Turn ---

Player hand after drawing: ['Estate', 'Silver', 'Copper', 'Smithy', 'Duchy']

--- ACTION PHASE ---

Player 1's hand: ['Estate', 'Silver', 'Copper', 'Smithy', 'Duchy']

Player 1 has 1 actions left.

Player 1 is playing Smithy.

Playing Smithy. Draw 3 more cards.

Player hand after drawing 3 more cards: ['Estate', 'Silver', 'Copper', 'Duchy', 'Copper', 'Silver', 'Estate']

Player hand after Smithy: ['Estate', 'Silver', 'Copper', 'Duchy', 'Copper', 'Silver', 'Estate']

Remaining actions: 0

Current hand after action: ['Estate', 'Silver', 'Copper', 'Duchy', 'Copper', 'Silver', 'Estate']

--- BUY PHASE ---

Player 1 has 6 treasure to spend.

Available cards for purchase:

Copper: Cost = 0, Remaining = 37

Silver: Cost = 3, Remaining = 30

Gold: Cost = 6, Remaining = 30

Estate: Cost = 2, Remaining = 18

Duchy: Cost = 5, Remaining = 10

Province: Cost = 8, Remaining = 12

Smithy: Cost = 4, Remaining = 9

Village: Cost = 3, Remaining = 4

Player 1 bought Village for 3 cost. 3 remaining.

--- CLEANUP PHASE ---

Player 1 discarding hand: ['Estate', 'Silver', 'Copper', 'Duchy', 'Copper', 'Silver', 'Estate']

Player hand after drawing: ['Silver', 'Silver', 'Copper', 'Copper', 'Estate']

New hand for Player 1: ['Silver', 'Silver', 'Copper', 'Copper', 'Estate']

Checking if game is over: False

--- ROUND 36, Player 2's Turn ---

Player hand after drawing: ['Estate', 'Estate', 'Estate', 'Estate', 'Copper']

--- ACTION PHASE ---

Player 2's hand: ['Estate', 'Estate', 'Estate', 'Estate', 'Copper']

Player 2 has 1 actions left.

--- BUY PHASE ---

Player 2 has 1 treasure to spend.

Available cards for purchase:

Copper: Cost = 0, Remaining = 37

Silver: Cost = 3, Remaining = 30

Gold: Cost = 6, Remaining = 30

Estate: Cost = 2, Remaining = 18

Duchy: Cost = 5, Remaining = 10

Province: Cost = 8, Remaining = 12

Smithy: Cost = 4, Remaining = 9

Village: Cost = 3, Remaining = 3

Player 2 bought Copper for 0 cost. 36 remaining.

--- CLEANUP PHASE ---

Player 2 discarding hand: ['Estate', 'Estate', 'Estate', 'Estate', 'Copper']

Player hand after drawing: ['Copper', 'Copper', 'Silver', 'Village', 'Copper']

New hand for Player 2: ['Copper', 'Copper', 'Silver', 'Village', 'Copper']

Checking if game is over: False

--- ROUND 37, Player 1's Turn ---

Player hand after drawing: ['Silver', 'Silver', 'Copper', 'Copper', 'Estate']

--- ACTION PHASE ---

Player 1's hand: ['Silver', 'Silver', 'Copper', 'Copper', 'Estate']

Player 1 has 1 actions left.

--- BUY PHASE ---

Player 1 has 6 treasure to spend.

Available cards for purchase:

Copper: Cost = 0, Remaining = 36

Silver: Cost = 3, Remaining = 30

Gold: Cost = 6, Remaining = 30

Estate: Cost = 2, Remaining = 18

Duchy: Cost = 5, Remaining = 10

Province: Cost = 8, Remaining = 12

Smithy: Cost = 4, Remaining = 9

Village: Cost = 3, Remaining = 3

Player 1 bought Smithy for 4 cost. 8 remaining.

--- CLEANUP PHASE ---

Player 1 discarding hand: ['Silver', 'Silver', 'Copper', 'Copper', 'Estate']

Player hand after drawing: ['Estate', 'Copper', 'Estate', 'Copper', 'Copper']

New hand for Player 1: ['Estate', 'Copper', 'Estate', 'Copper', 'Copper']

Checking if game is over: False

--- ROUND 38, Player 2's Turn ---

Player hand after drawing: ['Copper', 'Copper', 'Silver', 'Village', 'Copper']

--- ACTION PHASE ---

Player 2's hand: ['Copper', 'Copper', 'Silver', 'Village', 'Copper']

Player 2 has 1 actions left.

Player 2 is playing Village.

Playing Village. Draw 1 more card and gain two more actions.

Player hand after drawing 1 more cards: ['Copper', 'Copper', 'Silver', 'Copper', 'Copper']

Player actions after Village: 2

Remaining actions: 2

Current hand after action: ['Copper', 'Copper', 'Silver', 'Copper', 'Copper']

--- BUY PHASE ---

Player 2 has 6 treasure to spend.

Available cards for purchase:

Copper: Cost = 0, Remaining = 36

Silver: Cost = 3, Remaining = 30

Gold: Cost = 6, Remaining = 30

Estate: Cost = 2, Remaining = 18

Duchy: Cost = 5, Remaining = 10

Province: Cost = 8, Remaining = 12

Smithy: Cost = 4, Remaining = 8

Village: Cost = 3, Remaining = 3

Player 2 bought Gold for 6 cost. 29 remaining.

--- CLEANUP PHASE ---

Player 2 discarding hand: ['Copper', 'Copper', 'Silver', 'Copper', 'Copper']

Player hand after drawing: ['Copper', 'Village', 'Copper', 'Silver', 'Duchy']

New hand for Player 2: ['Copper', 'Village', 'Copper', 'Silver', 'Duchy']

Checking if game is over: False

--- ROUND 39, Player 1's Turn ---

Player hand after drawing: ['Estate', 'Copper', 'Estate', 'Copper', 'Copper']

--- ACTION PHASE ---

Player 1's hand: ['Estate', 'Copper', 'Estate', 'Copper', 'Copper']

Player 1 has 1 actions left.

--- BUY PHASE ---

Player 1 has 3 treasure to spend.

Available cards for purchase:

Copper: Cost = 0, Remaining = 36

Silver: Cost = 3, Remaining = 30

Gold: Cost = 6, Remaining = 29

Estate: Cost = 2, Remaining = 18

Duchy: Cost = 5, Remaining = 10

Province: Cost = 8, Remaining = 12

Smithy: Cost = 4, Remaining = 8

Village: Cost = 3, Remaining = 3

Player 1 bought Estate for 2 cost. 17 remaining.

--- CLEANUP PHASE ---

Player 1 discarding hand: ['Estate', 'Copper', 'Estate', 'Copper', 'Copper']

Player hand after drawing: ['Village', 'Silver', 'Silver', 'Duchy', 'Copper']

New hand for Player 1: ['Village', 'Silver', 'Silver', 'Duchy', 'Copper']

Checking if game is over: False

--- ROUND 40, Player 2's Turn ---

Player hand after drawing: ['Copper', 'Village', 'Copper', 'Silver', 'Duchy']

--- ACTION PHASE ---

Player 2's hand: ['Copper', 'Village', 'Copper', 'Silver', 'Duchy']

Player 2 has 1 actions left.

Player 2 is playing Village.

Playing Village. Draw 1 more card and gain two more actions.

Player hand after drawing 1 more cards: ['Copper', 'Copper', 'Silver', 'Duchy', 'Village']

Player actions after Village: 2

Remaining actions: 2

Current hand after action: ['Copper', 'Copper', 'Silver', 'Duchy', 'Village']

Player 2 is playing Village.

Playing Village. Draw 1 more card and gain two more actions.

Player hand after drawing 1 more cards: ['Copper', 'Copper', 'Silver', 'Duchy', 'Copper']

Player actions after Village: 3

Remaining actions: 3

Current hand after action: ['Copper', 'Copper', 'Silver', 'Duchy', 'Copper']

--- BUY PHASE ---

Player 2 has 5 treasure to spend.

Available cards for purchase:

Copper: Cost = 0, Remaining = 36

Silver: Cost = 3, Remaining = 30

Gold: Cost = 6, Remaining = 29

Estate: Cost = 2, Remaining = 17

Duchy: Cost = 5, Remaining = 10

Province: Cost = 8, Remaining = 12

Smithy: Cost = 4, Remaining = 8

Village: Cost = 3, Remaining = 3

Player 2 bought Silver for 3 cost. 29 remaining.

--- CLEANUP PHASE ---

Player 2 discarding hand: ['Copper', 'Copper', 'Silver', 'Duchy', 'Copper']

Player hand after drawing: ['Copper', 'Copper', 'Silver', 'Copper', 'Copper']

New hand for Player 2: ['Copper', 'Copper', 'Silver', 'Copper', 'Copper']

Checking if game is over: False

--- ROUND 41, Player 1's Turn ---

Player hand after drawing: ['Village', 'Silver', 'Silver', 'Duchy', 'Copper']

--- ACTION PHASE ---

Player 1's hand: ['Village', 'Silver', 'Silver', 'Duchy', 'Copper']

Player 1 has 1 actions left.

Player 1 is playing Village.

Playing Village. Draw 1 more card and gain two more actions.

Player hand after drawing 1 more cards: ['Silver', 'Silver', 'Duchy', 'Copper', 'Silver']

Player actions after Village: 2

Remaining actions: 2

Current hand after action: ['Silver', 'Silver', 'Duchy', 'Copper', 'Silver']

--- BUY PHASE ---

Player 1 has 7 treasure to spend.

Available cards for purchase:

Copper: Cost = 0, Remaining = 36

Silver: Cost = 3, Remaining = 29

Gold: Cost = 6, Remaining = 29

Estate: Cost = 2, Remaining = 17

Duchy: Cost = 5, Remaining = 10

Province: Cost = 8, Remaining = 12

Smithy: Cost = 4, Remaining = 8

Village: Cost = 3, Remaining = 3

Player 1 bought Duchy for 5 cost. 9 remaining.

--- CLEANUP PHASE ---

Player 1 discarding hand: ['Silver', 'Silver', 'Duchy', 'Copper', 'Silver']

Player hand after drawing: ['Silver', 'Copper', 'Estate', 'Smithy', 'Estate']

New hand for Player 1: ['Silver', 'Copper', 'Estate', 'Smithy', 'Estate']

Checking if game is over: False

--- ROUND 42, Player 2's Turn ---

Player hand after drawing: ['Copper', 'Copper', 'Silver', 'Copper', 'Copper']

--- ACTION PHASE ---

Player 2's hand: ['Copper', 'Copper', 'Silver', 'Copper', 'Copper']

Player 2 has 1 actions left.

--- BUY PHASE ---

Player 2 has 6 treasure to spend.

Available cards for purchase:

Copper: Cost = 0, Remaining = 36

Silver: Cost = 3, Remaining = 29

Gold: Cost = 6, Remaining = 29

Estate: Cost = 2, Remaining = 17

Duchy: Cost = 5, Remaining = 9

Province: Cost = 8, Remaining = 12

Smithy: Cost = 4, Remaining = 8

Village: Cost = 3, Remaining = 3

Player 2 bought Silver for 3 cost. 28 remaining.

--- CLEANUP PHASE ---

Player 2 discarding hand: ['Copper', 'Copper', 'Silver', 'Copper', 'Copper']

Player hand after drawing: ['Estate', 'Estate', 'Duchy', 'Copper', 'Copper']

New hand for Player 2: ['Estate', 'Estate', 'Duchy', 'Copper', 'Copper']

Checking if game is over: False

Στους μεσαίους γύρους, οι επιλογές του agent αρχίζουν να γίνονται πιο πολύπλοκες και στρατηγικές:

- **Συνδυασμός καρτών δράσης**: Παρατηρούμε ότι ο Player 2 χρησιμοποιεί πιο εξελιγμένες κάρτες όπως το Smithy (για να τραβήξει 3 κάρτες) και το Village (για να κερδίσει επιπλέον δράσεις). Αυτό σημαίνει ότι ο agent μαθαίνει να εκμεταλλεύεται τις κάρτες που του προσφέρουν επιπλέον ενέργειες και κάρτες, κάτι που οδηγεί σε μεγαλύτερη ευελιξία στις μετέπειτα φάσεις.

- **Αύξηση θησαυρών και πόντων**: Ο agent αγοράζει Silver και Estate, συνδυάζοντας θησαυρούς και κάρτες πόντων, βελτιώνοντας την απόδοσή του τόσο στο οικονομικό όσο και στο στρατηγικό επίπεδο. Αυτό δείχνει πως αρχίζει να μαθαίνει να εξισορροπεί τις ανάγκες για θησαυρό και πόντους νίκης.

6.3.3 Τελευταίοι γύροι (Rounds 102-107):

--- ROUND 102, Player 2's Turn ---

Player hand after drawing: ['Estate', 'Smithy', 'Silver', 'Silver', 'Village']

--- ACTION PHASE ---

Player 2's hand: ['Estate', 'Smithy', 'Silver', 'Silver', 'Village']

Player 2 has 1 actions left.

Player 2 is playing Smithy.

Playing Smithy. Draw 3 more cards.

Player hand after drawing 3 more cards: ['Estate', 'Silver', 'Silver', 'Village', 'Silver', 'Silver', 'Copper']

Player hand after Smithy: ['Estate', 'Silver', 'Silver', 'Village', 'Silver', 'Silver', 'Copper']

Remaining actions: 0

Current hand after action: ['Estate', 'Silver', 'Silver', 'Village', 'Silver', 'Silver', 'Copper']

--- BUY PHASE ---

Player 2 has 9 treasure to spend.

Available cards for purchase:

Copper: Cost = 0, Remaining = 28

Silver: Cost = 3, Remaining = 12

Gold: Cost = 6, Remaining = 23

Estate: Cost = 2, Remaining = 8

Duchy: Cost = 5, Remaining = 1

Province: Cost = 8, Remaining = 11

Player 2 bought Estate for 2 cost. 7 remaining.

--- CLEANUP PHASE ---

Player 2 discarding hand: ['Estate', 'Silver', 'Silver', 'Village', 'Silver', 'Silver', 'Copper']

Player hand after drawing: ['Copper', 'Village', 'Estate', 'Silver', 'Gold']

New hand for Player 2: ['Copper', 'Village', 'Estate', 'Silver', 'Gold']

Checking if game is over: False

--- ROUND 103, Player 1's Turn ---

Player hand after drawing: ['Smithy', 'Estate', 'Duchy', 'Duchy', 'Estate']

--- ACTION PHASE ---

Player 1's hand: ['Smithy', 'Estate', 'Duchy', 'Duchy', 'Estate']

Player 1 has 1 actions left.

Player 1 is playing Smithy.

Playing Smithy. Draw 3 more cards.

Player hand after drawing 3 more cards: ['Estate', 'Duchy', 'Duchy', 'Estate', 'Province', 'Silver', 'Copper']

Player hand after Smithy: ['Estate', 'Duchy', 'Duchy', 'Estate', 'Province', 'Silver', 'Copper']

Remaining actions: 0

Current hand after action: ['Estate', 'Duchy', 'Duchy', 'Estate', 'Province', 'Silver', 'Copper']

--- BUY PHASE ---

Player 1 has 3 treasure to spend.

Available cards for purchase:

Copper: Cost = 0, Remaining = 28

Silver: Cost = 3, Remaining = 12

Gold: Cost = 6, Remaining = 23

Estate: Cost = 2, Remaining = 7

Duchy: Cost = 5, Remaining = 1

Province: Cost = 8, Remaining = 11

Player 1 bought Estate for 2 cost. 6 remaining.

--- CLEANUP PHASE ---

Player 1 discarding hand: ['Estate', 'Duchy', 'Duchy', 'Estate', 'Province', 'Silver', 'Copper']

Player hand after drawing: ['Copper', 'Village', 'Copper', 'Estate', 'Silver']

New hand for Player 1: ['Copper', 'Village', 'Copper', 'Estate', 'Silver']

Checking if game is over: False

--- ROUND 104, Player 2's Turn ---

Player hand after drawing: ['Copper', 'Village', 'Estate', 'Silver', 'Gold']

--- ACTION PHASE ---

Player 2's hand: ['Copper', 'Village', 'Estate', 'Silver', 'Gold']

Player 2 has 1 actions left.

Player 2 is playing Village.

Playing Village. Draw 1 more card and gain two more actions.

Player hand after drawing 1 more cards: ['Copper', 'Estate', 'Silver', 'Gold', 'Copper']

Player actions after Village: 2

Remaining actions: 2

Current hand after action: ['Copper', 'Estate', 'Silver', 'Gold', 'Copper']

--- BUY PHASE ---

Player 2 has 7 treasure to spend.

Available cards for purchase:

Copper: Cost = 0, Remaining = 28

Silver: Cost = 3, Remaining = 12

Gold: Cost = 6, Remaining = 23

Estate: Cost = 2, Remaining = 6

Duchy: Cost = 5, Remaining = 1

Province: Cost = 8, Remaining = 11

Player 2 bought Estate for 2 cost. 5 remaining.

--- CLEANUP PHASE ---

Player 2 discarding hand: ['Copper', 'Estate', 'Silver', 'Gold', 'Copper']

Player hand after drawing: ['Estate', 'Silver', 'Smithy', 'Smithy', 'Estate']

New hand for Player 2: ['Estate', 'Silver', 'Smithy', 'Smithy', 'Estate']

Checking if game is over: False

--- ROUND 105, Player 1's Turn ---

Player hand after drawing: ['Copper', 'Village', 'Copper', 'Estate', 'Silver']

--- ACTION PHASE ---

Player 1's hand: ['Copper', 'Village', 'Copper', 'Estate', 'Silver']

Player 1 has 1 actions left.

Player 1 is playing Village.

Playing Village. Draw 1 more card and gain two more actions.

Player hand after drawing 1 more cards: ['Copper', 'Copper', 'Estate', 'Silver', 'Smithy']

Player actions after Village: 2

Remaining actions: 2

Current hand after action: ['Copper', 'Copper', 'Estate', 'Silver', 'Smithy']

Player 1 is playing Smithy.

Playing Smithy. Draw 3 more cards.

Player hand after drawing 3 more cards: ['Copper', 'Copper', 'Estate', 'Silver', 'Silver', 'Estate', 'Silver']

Player hand after Smithy: ['Copper', 'Copper', 'Estate', 'Silver', 'Silver', 'Estate', 'Silver']

Remaining actions: 1

Current hand after action: ['Copper', 'Copper', 'Estate', 'Silver', 'Silver', 'Estate', 'Silver']

--- BUY PHASE ---

Player 1 has 8 treasure to spend.

Available cards for purchase:

Copper: Cost = 0, Remaining = 28

Silver: Cost = 3, Remaining = 12

Gold: Cost = 6, Remaining = 23

Estate: Cost = 2, Remaining = 5

Duchy: Cost = 5, Remaining = 1

Province: Cost = 8, Remaining = 11

Player 1 bought Silver for 3 cost. 11 remaining.

--- CLEANUP PHASE ---

Player 1 discarding hand: ['Copper', 'Copper', 'Estate', 'Silver', 'Silver', 'Estate', 'Silver']

Player hand after drawing: ['Copper', 'Village', 'Copper', 'Estate', 'Smithy']

New hand for Player 1: ['Copper', 'Village', 'Copper', 'Estate', 'Smithy']

Checking if game is over: False

--- ROUND 106, Player 2's Turn ---

Player hand after drawing: ['Estate', 'Silver', 'Smithy', 'Smithy', 'Estate']

--- ACTION PHASE ---

Player 2's hand: ['Estate', 'Silver', 'Smithy', 'Smithy', 'Estate']

Player 2 has 1 actions left.

Player 2 is playing Smithy.

Playing Smithy. Draw 3 more cards.

Player hand after drawing 3 more cards: ['Estate', 'Silver', 'Smithy', 'Estate', 'Village', 'Estate', 'Silver']

Player hand after Smithy: ['Estate', 'Silver', 'Smithy', 'Estate', 'Village', 'Estate', 'Silver']

Remaining actions: 0

Current hand after action: ['Estate', 'Silver', 'Smithy', 'Estate', 'Village', 'Estate', 'Silver']

--- BUY PHASE ---

Player 2 has 4 treasure to spend.

Available cards for purchase:

Copper: Cost = 0, Remaining = 28

Silver: Cost = 3, Remaining = 11

Gold: Cost = 6, Remaining = 23

Estate: Cost = 2, Remaining = 5

Duchy: Cost = 5, Remaining = 1

Province: Cost = 8, Remaining = 11

Player 2 bought Copper for 0 cost. 27 remaining.

--- CLEANUP PHASE ---

Player 2 discarding hand: ['Estate', 'Silver', 'Smithy', 'Estate', 'Village', 'Estate', 'Silver']

Player hand after drawing: ['Copper', 'Copper', 'Silver', 'Estate', 'Copper']

New hand for Player 2: ['Copper', 'Copper', 'Silver', 'Estate', 'Copper']

Checking if game is over: False

--- ROUND 107, Player 1's Turn ---

Player hand after drawing: ['Copper', 'Village', 'Copper', 'Estate', 'Smithy']

--- ACTION PHASE ---

Player 1's hand: ['Copper', 'Village', 'Copper', 'Estate', 'Smithy']

Player 1 has 1 action left.

Player 1 is playing Village.

Playing Village. Draw 1 more card and gain two more actions.

Player hand after drawing 1 more cards: ['Copper', 'Copper', 'Estate', 'Smithy', 'Silver']

Player actions after Village: 2

Remaining actions: 2

Current hand after action: ['Copper', 'Copper', 'Estate', 'Smithy', 'Silver']

Player 1 is playing Smithy.

Playing Smithy. Draw 3 more cards.

Player hand after drawing 3 more cards: ['Copper', 'Copper', 'Estate', 'Silver', 'Estate', 'Copper', 'Silver']

Player hand after Smithy: ['Copper', 'Copper', 'Estate', 'Silver', 'Estate', 'Copper', 'Silver']

Remaining actions: 1

Current hand after action: ['Copper', 'Copper', 'Estate', 'Silver', 'Estate', 'Copper', 'Silver']

--- BUY PHASE ---

Player 1 has 7 treasure to spend.

Available cards for purchase:

Copper: Cost = 0, Remaining = 27

Silver: Cost = 3, Remaining = 11

Gold: Cost = 6, Remaining = 23

Estate: Cost = 2, Remaining = 5

Duchy: Cost = 5, Remaining = 1

Province: Cost = 8, Remaining = 11

Player 1 bought Duchy for 5 cost. 0 remaining.

--- CLEANUP PHASE ---

Player 1 discarding hand: ['Copper', 'Copper', 'Estate', 'Silver', 'Estate', 'Copper', 'Silver']

Player hand after drawing: ['Copper', 'Village', 'Silver', 'Smithy', 'Duchy']

New hand for Player 1: ['Copper', 'Village', 'Silver', 'Smithy', 'Duchy']

Checking if game is over: True

Στο τέλος του παιχνιδιού ο πράκτορας ακολουθεί στρατηγική:

- **Συνδυασμοί πολλαπλών καρτών**: Ο agent αρχίζει να παίζει το Village και το Smithy συνδυαστικά. Παίζει το Village για να αποκτήσει επιπλέον ενέργειες και μετά το Smithy

για να τραβήξει επιπλέον κάρτες, επιτρέποντάς του να έχει μια μεγαλύτερη και πιο παραγωγική φάση αγοράς.

- **Στενευμένες αγορές για νίκη**: Ο Player 1 αγοράζει Duchy στον γύρο 107, μια κάρτα με υψηλούς πόντους νίκης. Σε αυτό το σημείο, ο agent δείχνει ότι κατανοεί πως πρέπει να επενδύσει σε κάρτες με πόντους νίκης για να κερδίσει το παιχνίδι, ακόμη και αν δεν του προσφέρουν άμεσα οφέλη κατά τη διάρκεια των γύρων.

6.4 Ανάλυση αποτελεσμάτων

Αρχικά rounds :

- **Αρχικά Στάδια του Παιχνιδιού:** Και οι δύο παίκτες ξεκινούν με χαμηλούς πόντους νίκης, αλλά η πρόοδος είναι σχετικά αργή και παρόμοια για τους δύο παίκτες.
- **Μεσαία Στάδια του Παιχνιδιού:** Από τον 40ό γύρο και μετά, παρατηρούμε μια σημαντική αύξηση στους πόντους νίκης και για τους δύο παίκτες. Ο Player 2 φαίνεται να αποκτά πλεονέκτημα, καθώς οι πόντοι νίκης αυξάνονται πιο απότομα συγκριτικά με τον Player 1.
- **Τελικά Στάδια του Παιχνιδιού:** Στους τελευταίους γύρους (από τον 100ό και μετά), η αύξηση στους πόντους νίκης συνεχίζεται, με τον Player 1 να επιτυγχάνει τελικά υψηλότερη αύξηση στους πόντους νίκης, ξεπερνώντας τον Player 2 στους τελευταίους γύρους του παιχνιδιού.

6.5 Μαθησιακή διαδικασία:

Από τα δεδομένα των γύρων, φαίνεται ότι ο agent μαθαίνει προοδευτικά:

1. **Σταθερή βάση πόρων**: Στην αρχή επικεντρώνεται στη σταθεροποίηση των πόρων του με ασφαλείς αγορές (Silver).
2. **Εκμάθηση συνδυασμών**: Μαθαίνει πώς να χρησιμοποιεί τις κάρτες δράσης για να εκμεταλλεύεται συνδυασμούς που του επιτρέπουν να αυξήσει την απόδοσή του.
3. **Εξισορρόπηση θησαυρών και πόντων**: Κατανοεί σταδιακά τη σημασία της ισορροπίας ανάμεσα στην αγορά καρτών πόντων νίκης και καρτών που αυξάνουν την οικονομική του δύναμη.

4. **Αύξηση πολυπλοκότητας**: Χρησιμοποιεί σύνθετους συνδυασμούς καρτών για να μεγιστοποιήσει τις ενέργειες και τους πόρους του, όπως φαίνεται στους τελευταίους γύρους.

6.6 Παραμετροποίηση Παραμέτρων Νευρωνικού Δικτύου

Διαφορετικοί συνδυασμοί για τους δυο agent.

Συνδυασμός 1: Χαμηλός ρυθμός μάθησης και αργή απόσβεση εξερεύνησης

- Agent 1:
    - Learning Rate: 0.0001
    - Discount Factor: 0.95
    - Exploration Decay: 0.995
    - Replay Memory Size: 2000
    - Batch Size: 64
- Agent 2:
    - Learning Rate: 0.0002
    - Discount Factor: 0.94
    - Exploration Decay: 0.994
    - Replay Memory Size: 2500
    - Batch Size: 64

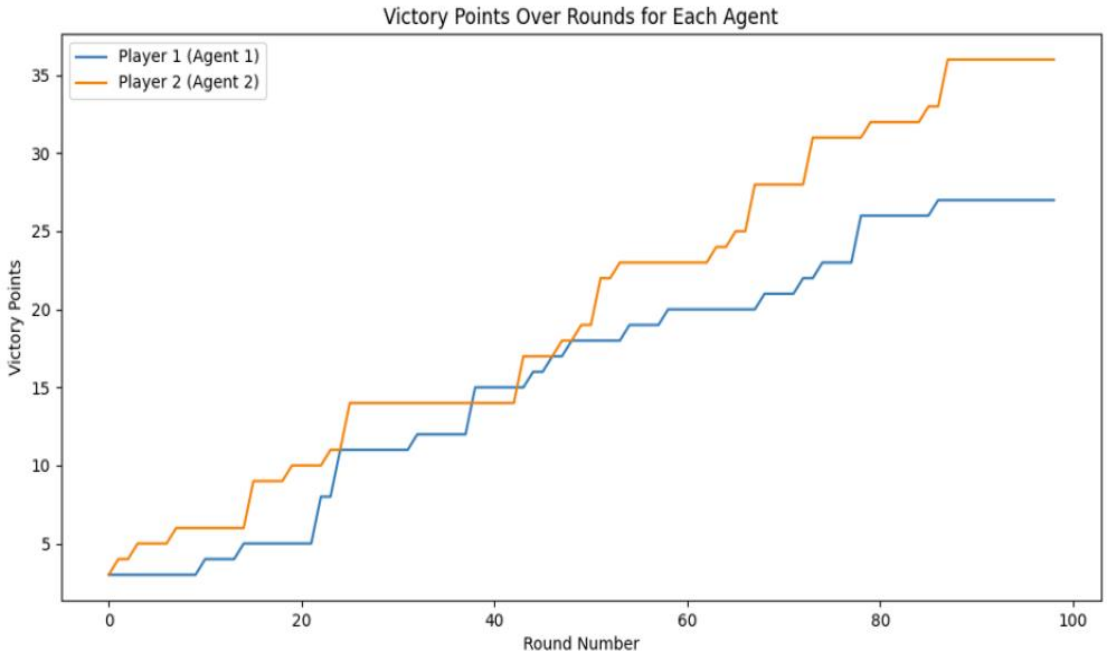
Συνδυασμός 2: Υψηλός ρυθμός μάθησης και γρήγορη απόσβεση εξερεύνησης

- Agent 1:
    - Learning Rate: 0.001
    - Discount Factor: 0.99
    - Exploration Decay: 0.99
    - Replay Memory Size: 10000
    - Batch Size: 32
- Agent 2:
    - Learning Rate: 0.0015
    - Discount Factor: 0.98
    - Exploration Decay: 0.985
    - Replay Memory Size: 12000
    - Batch Size: 32

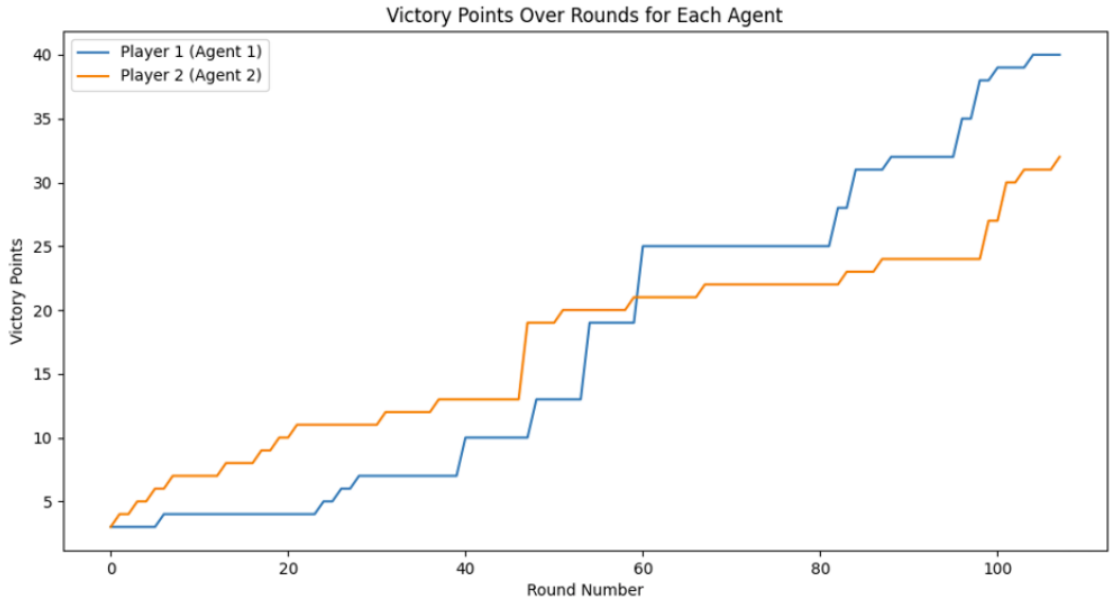
Συνδυασμός 3: Μέτριος ρυθμός μάθησης και μέτρια απόσβεση εξερεύνησης

- Agent 1:
    - Learning Rate: 0.0005
    - Discount Factor: 0.97
    - Exploration Decay: 0.992
    - Replay Memory Size: 5000
    - Batch Size: 48
- Agent 2:
    - Learning Rate: 0.0006
    - Discount Factor: 0.96
    - Exploration Decay: 0.991
    - Replay Memory Size: 5500
    - Batch Size: 48

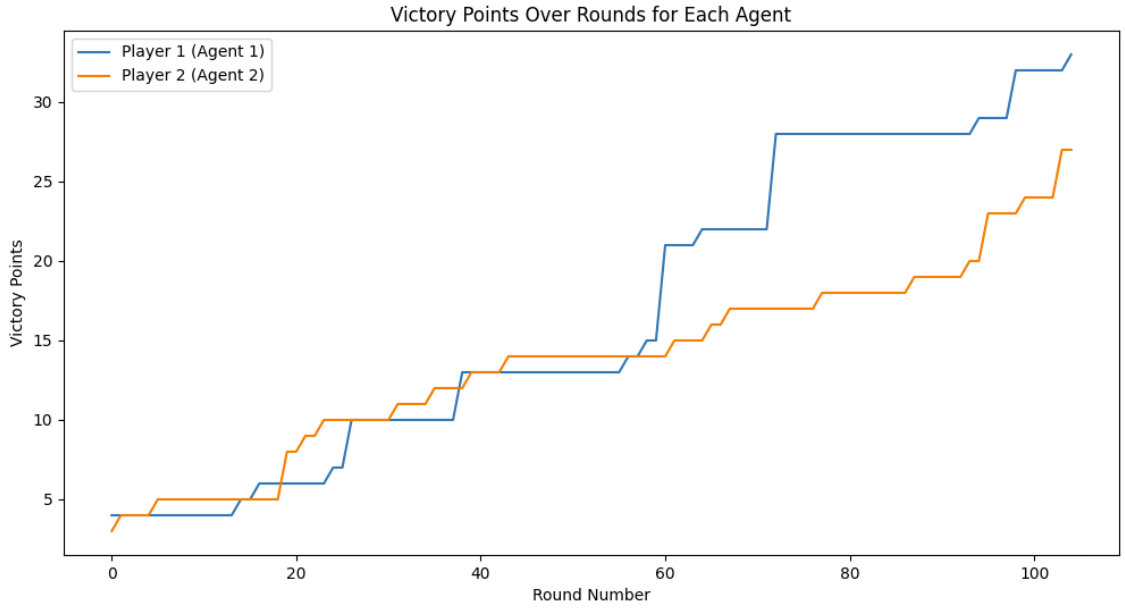
Διάγραμμα 2 : Συνδυασμός 1



Διάγραμμα 3 : Συνδυασμός 2



Διάγραμμα 4 : Συνδυασμός 3



Τα αποτελέσματα από τα τρία διαγράμματα συνδυασμών δείχνουν την διαφορά στον τρόπο με τον οποίο μαθαίνουν οι δυο διαφορετικοί agent με διαφορετικούς συνδυασμούς στις παραμέτρους του νευρωνικού δικτύου.

Από τα διαγράμματα φαίνεται ότι ο δεύτερος συνδυασμός είναι πιο αποτελεσματικός, καθώς ο Agent 1 μαθαίνει πιο γρήγορα και πιο σταθερά. Συγκεκριμένα:

- Στον συνδυασμό 2, ο Agent 1 μαθαίνει ταχύτερα και συγκεντρώνει τους περισσοτέρους πόντους νίκης.

- Συμπέρασμα :

Ο agent μαθαίνει βήμα-βήμα να αναγνωρίζει και να εφαρμόζει πιο εξελιγμένες στρατηγικές, βελτιώνοντας συνεχώς την απόδοσή του. Στα πρώτα στάδια, επικεντρώνεται στην ενίσχυση των θησαυρών και σταδιακά μαθαίνει πώς να χρησιμοποιεί κάρτες δράσης για να κερδίζει στρατηγικά πλεονεκτήματα. Στους τελευταίους γύρους, έχει μάθει να στοχεύει σε κάρτες με πόντους νίκης για να εξασφαλίσει την τελική του νίκη. Σε όλη αυτή την διαδικασία μάθησης σημαντικό ρόλο παίζουν οι παράμετροι του νευρωνικού δικτύου.

7 Βιβλιογραφία – Αναφορές - Διαδικτυακές Πηγές

[1.] Pang-Ning Tan, Michael Steinbach and Vipin Kumar, "Introduction to Data Mining".

[2.] Andreas C. Müller and Sarah Guido, "Introduction to Machine Learning with Python".

[3.] Yannakakis, G. N., & Togelius, J. (2018). *Artificial Intelligence and Games*. Springer.

[4.] Géron, A. (2019). Hands-On Machine Learning with Scikit-Learn, Keras, and TensorFlow. O'Reilly Media.

[5.] Müller, A. C., & Guido, S. (2016). Introduction to Machine Learning with Python. O'Reilly Media.

[6.] Goodfellow, I., Bengio, Y., & Courville, A. (2016). Deep Learning. MIT Press.

[7.] Bishop, C. M. (2006). Pattern Recognition and Machine Learning. Springer.

[8.] Sutton, R. S., & Barto, A. G. (2018). Reinforcement Learning: An Introduction. MIT Press.

[9.] Nayak et al., (2021) Reinforcement Learning – RL.

[10.] Russell, S., & Norvig, P. (2016). *Artificial Intelligence: A Modern Approach* (3rd ed.). Pearson.

[11.] Mnih, V., Kavukcuoglu, K., Silver, D., Rusu, A. A., Veness, J., Bellemare, M. G., ... & Hassabis, D. (2015). Human-level control through deep reinforcement learning. *Nature*, 518(7540), 529-533.

[12.] Silver, D., Schrittwieser, J., Simonyan, K., Antonoglou, I., Huang, A., Guez, A., ... & Hassabis, D. (2017). Mastering the game of Go without human knowledge. *Nature*, 550(7676), 354-359.

[13.] Scikit-learn: *Machine Learning in Python*. Scikit-learn

[14.] TensorFlow: *An end-to-end open-source machine learning platform*. TensorFlow

[15.] PyTorch: *An open-source machine learning framework*. PyTorch