ΠΑΝΕΠΙΣΤΗΜΙΟ ΘΕΣΣΑΛΙΑΣ

ΠΟΛΥΤΕΧΝΙΚΗ ΣΧΟΛΗ

ΤΜΗΜΑ ΗΛΕΚΤΡΟΛΟΓΩΝ ΜΗΧΑΝΙΚΩΝ ΚΑΙ ΜΗΧΑΝΙΚΩΝ ΥΠΟΛΟΓΙΣΤΩΝ

Ανάπτυξη AI για Βέλτιστη Στρατηγική σε Turn-Based Game μέσω Προσομοίωσης και Machine Learning

Διπλωματική Εργασία

Γκέλος Τρύφων

Επιβλέπουσα: Τσαλαπάτα Χαρίκλεια

Ιούλιος 2025

ΠΑΝΕΠΙΣΤΗΜΙΟ ΘΕΣΣΑΛΙΑΣ

ΠΟΛΥΤΕΧΝΙΚΗ ΣΧΟΛΗ

ΤΜΗΜΑ ΗΛΕΚΤΡΟΛΟΓΩΝ ΜΗΧΑΝΙΚΩΝ ΚΑΙ ΜΗΧΑΝΙΚΩΝ ΥΠΟΛΟΓΙΣΤΩΝ

Ανάπτυξη AI για Βέλτιστη Στρατηγική σε Turn-Based Game μέσω Προσομοίωσης και Machine Learning

Διπλωματική Εργασία

Γκέλος Τρύφων

Επιβλέπουσα: Τσαλαπάτα Χαρίκλεια

Ιούλιος 2025

UNIVERSITY OF THESSALY

SCHOOL OF ENGINEERING

DEPARTMENT OF ELECTRICAL AND COMPUTER ENGINEERING

Development of AI for Optimal Strategy in a Turn-Based Game through Simulation and Machine Learning

Diploma Thesis

Gkelos Tryfon

Supervisor: Tsalapata Charikleia

July 2025

Εγκρίνεται από την Επιτροπή Εξέτασης:

Επιβλέπουσα **Τσαλαπάτα Χαρίκλεια**
Καθηγήτρια, Τμήμα Ηλεκτρολόγων Μηχανικών & Μηχανικών Υπολογιστών, Πανεπιστήμιο Θεσσαλίας

Μέλος **Σταμούλης Γεώργιος**
Καθηγητής, Τμήμα Ηλεκτρολόγων Μηχανικών & Μηχανικών Υπολογιστών, Πανεπιστήμιο Θεσσαλίας

Μέλος **Φεύγας Αθανάσιος**
Καθηγητής, Τμήμα Ηλεκτρολόγων Μηχανικών & Μηχανικών Υπολογιστών, Πανεπιστήμιο Θεσσαλίας

ΥΠΕΥΘΥΝΗ ΔΗΛΩΣΗ ΠΕΡΙ ΑΚΑΔΗΜΑΪΚΗΣ ΔΕΟΝΤΟΛΟΓΙΑΣ ΚΑΙ ΠΝΕΥΜΑΤΙΚΩΝ ΔΙΚΑΙΩΜΑΤΩΝ

«Με πλήρη επίγνωση των συνεπειών του νόμου περί πνευματικών δικαιωμάτων, δηλώνω ρητά ότι η παρούσα διπλωματική εργασία, καθώς και τα ηλεκτρονικά αρχεία και πηγαίοι κώδικες που αναπτύχθηκαν ή τροποποιήθηκαν στα πλαίσια αυτής της εργασίας, αποτελεί αποκλειστικά προϊόν προσωπικής μου εργασίας, δεν προσβάλλει κάθε μορφής δικαιώματα διανοητικής ιδιοκτησίας, προσωπικότητας και προσωπικών δεδομένων τρίτων, δεν περιέχει έργα/εισφορές τρίτων για τα οποία απαιτείται άδεια των δημιουργών/δικαιούχων και δεν είναι προϊόν μερικής ή ολικής αντιγραφής, οι πηγές δε που χρησιμοποιήθηκαν περιορίζονται στις βιβλιογραφικές αναφορές και μόνον και πληρούν τους κανόνες της επιστημονικής παράθεσης. Τα σημεία όπου έχω χρησιμοποιήσει ιδέες, κείμενο, αρχεία ή/και πηγές άλλων συγγραφέων, αναφέρονται ευδιάκριτα στο κείμενο με την κατάλληλη παραπομπή και η σχετική αναφορά περιλαμβάνεται στο τμήμα των βιβλιογραφικών αναφορών με πλήρη περιγραφή. Δηλώνω επίσης ότι τα αποτελέσματα της εργασίας δεν έχουν χρησιμοποιηθεί για την απόκτηση άλλου πτυχίου. Αναλαμβάνω πλήρως, ατομικά και προσωπικά, όλες τις νομικές και διοικητικές συνέπειες που δύναται να προκύψουν στην περίπτωση κατά την οποία αποδειχθεί, διαχρονικά, ότι η εργασία αυτή ή τμήμα της δεν μου ανήκει διότι είναι προϊόν λογοκλοπής».

Ο/Η Δηλών/ούσα

Γκέλος Τρύφων

Διπλωματική Εργασία

Ανάπτυξη AI για Βέλτιστη Στρατηγική σε Turn-Based Game μέσω Προσομοίωσης και Machine Learning

Γκέλος Τρύφων

Περίληψη

Η παρούσα διπλωματική εργασία εστιάζει στην ανάπτυξη τεχνητής νοημοσύνης (AI) που μπορεί να ανιχνεύει και να εφαρμόζει τη βέλτιστη στρατηγική σε ένα νέο turn-based παιχνίδι, χρησιμοποιώντας simulation και reinforcement learning τεχνικές. Αντλώντας έμπνευση από chess solvers όπως το Stockfish, σχεδιάστηκε και υλοποιήθηκε ένας AI agent για ένα νέο παιχνίδι που αναπτύχθηκε εξαρχής σε γλώσσα C#, με δυνατότητα οπτικοποίησης στο Unity.

Το παιχνίδι περιλαμβάνει τρεις τύπους χαρακτήρων —Tank, Healer και DPS— καθένας με διαφορετικές κινήσεις και ιδιότητες. Ο AI agent βασίζεται στην αξιολόγηση καταστάσεων παιχνιδιού μέσω 16 παραμέτρων (weights) και εκπαιδεύτηκε με χρήση του αλγορίθμου Minimax σε συνδυασμό με Temporal Difference Learning (TDL).

Τα αποτελέσματα δείχνουν ότι ο agent είναι σε θέση να εφαρμόζει αξιόπιστες και αποδοτικές στρατηγικές εντός των κανόνων του παιχνιδιού, ενώ η απόδοσή του βελτιώνεται αισθητά όσο αυξάνεται το βάθος αναζήτησης. Η εργασία καταδεικνύει ότι είναι εφικτή η ανάπτυξη ικανού AI για custom turn-based παιχνίδια, ακόμη και σε συνθήκες περιορισμένων υπολογιστικών πόρων.

Λέξεις-κλειδιά:

AI, turn-based game, Minimax, Temporal Difference Learning, C#, Unity

Διπλωματική Εργασία

Ανάπτυξη AI για Βέλτιστη Στρατηγική σε Turn-Based Game μέσω Προσομοίωσης και Machine Learning

Γκέλος Τρύφων

Περίληψη

This thesis focuses on the development of artificial intelligence (AI) capable of detecting and applying optimal strategies in a turn-based game, utilizing simulation and reinforcement learning techniques. Inspired by chess engines like Stockfish, a custom AI agent was designed and implemented for a newly developed game built from scratch in C#, with the ability to be visualized in Unity.

The game features three unit types—Tank, Healer, and DPS—each with distinct abilities and usage. The AI agent evaluates game states through 16 hand-crafted features (weights) and was trained using the Minimax algorithm in combination with Temporal Difference Learning (TDL).

Results show that the agent is able to apply consistent and effective strategies within the game's ruleset, with performance improving significantly as the search depth increases. This work demonstrates that efficient AI for custom turn-based games can be developed even under limited hardware resources.

Keywords:

AI, turn-based game, Minimax, Temporal Difference Learning, C#, Unity

Πίνακας περιεχομένων

Κεφάλαιο 1

Εισαγωγή

Η ραγδαία εξέλιξη της *Τεχνητής Νοημοσύνης* (AI) τα τελευταία δεκαπέντε χρόνια έχει επαναπροσδιορίσει τον τρόπο με τον οποίο αντιλαμβανόμαστε, σχεδιάζουμε και τελικά παίζουμε παιχνίδια στρατηγικής. Από τη θρυλική πλέον αναμέτρηση του *AlphaGo* με τον Lee Sedol το 2016 μέχρι τις πρόσφατες επιτυχίες του *AlphaStar* στο *StarCraft II* και του *Stockfish 16* στο σκάκι, οι AI agents απέδειξαν ότι η υπολογιστική δημιουργικότητα μπορεί όχι μόνον να ανταγωνιστεί αλλά συχνά να ξεπεράσει την ανθρώπινη διαίσθηση. Η παρούσα διπλωματική εργασία τοποθετείται μέσα σε αυτό το δυναμικό ερευνητικό τοπίο, εστιάζοντας στην ανάπτυξη ενός πράκτορα για ένα εντελώς καινούριο turn-based παιχνίδι που σχεδιάστηκε εκ του μηδενός.

Έτσι, σε αντίθεση με έργα που αξιοποιούν ήδη καταξιωμένες πλατφόρμες, το εγχείρημα αυτό συνδύασε δύο παράλληλες—και αλληλένδετες—προκλήσεις: *(α)* τον σχεδιασμό ενός ισορροπημένου παιχνιδιού το οποίο να «αντέχει» σε σοβαρή ανάλυση από AI· *(β)* την υλοποίηση ενός πράκτορα ικανού να μαθαίνει στρατηγικές αποκλειστικά μέσω *self-play*. Το διπλό αυτό στοίχημα απαίτησε διαδοχικές επαναληπτικές βελτιώσεις, τόσο στον πυρήνα κανόνων όσο και στον αλγόριθμο μάθησης.

Πιο συγκεκριμένα, το περιβάλλον υλοποιεί σκακιέρα 8×8 όπου κάθε παίκτης χειρίζεται οκτώ μονάδες — δύο *Tanks*, δύο *Healers* και τέσσερις *DPS* — επιτρέποντας πλούσιο τακτικό ρεπερτόριο. Ο προτεινόμενος πράκτορας συνδυάζει αναζήτηση *Minimax* με α–β *pruning* βάθους 4 και αξιολόγηση καταστάσεων βασισμένη σε δεκαέξι γραμμικά χαρακτηριστικά που εκπαιδεύονται μέσω *Temporal-Difference Learning*. Η εκπαίδευση πραγματοποιήθηκε αποκλειστικά με *self-play*, παράγοντας πάνω από 1,6 εκατ. αγώνες και αποφέροντας ποσοστό νικών 83,5 % έναντι προηγούμενων εκδόσεων του ίδιου agent. Για την εμπειρική αξιολό-

γηση αναπτύχθηκε *Unity* viewer, επιτρέποντας βήμα-προς-βήμα αναπαραγωγή αγώνων και οπτικοποίηση των σημαντικότερων κλαδιών αναζήτησης. Παράλληλα, τεχνικές *hill-climbing* χρησιμοποιήθηκαν για τη ρύθμιση των βαρών, διασφαλίζοντας ότι καμία ικανότητα δεν καθίσταται ούτε υπερβολικά ισχυρή ούτε ανενεργή. Τα αποτελέσματα καταδεικνύουν πως ακόμη και με καθαρά *CPU-friendly* μεθόδους είναι εφικτή η ανάπτυξη ανταγωνιστικού AI, όταν συνοδεύεται από προσεγμένο σχεδιασμό παιχνιδιού και στοχευμένη συλλογή δεδομένων.

Τέλος, ο πλήρης πηγαίος κώδικας, τα checkpoints εκπαίδευσης και τα τελικά βάρη διατίθενται ως *open-source*, προάγοντας τη διαφάνεια και επιτρέποντας σε τρίτους να αναπαράγουν ή να επεκτείνουν τα ευρήματα της παρούσας εργασίας.

1.1 Κίνητρο και Στόχοι

Η ιδέα γεννήθηκε όταν παρατηρήθηκε ότι, παρά την πληθώρα ερευνητικών εργασιών σε Go, σκάκι και RTS παιχνίδια, τα *Auto-Battlers* παραμένουν υπο-εκπροσωπούμενα στην ακαδημαϊκή βιβλιογραφία. Τα συγκεκριμένα παιχνίδια — δημοφιλή από το 2019 και έπειτα — παντρεύουν στοιχεία σκακιού (τοποθέτηση κομματιών), *role playing* (τύποι μονάδων με ρόλους Tank / Healer / DPS) και στοχαστικών μηχανισμών παρόμοιων με *trading-card* τίτλους. Αυτό το μείγμα τα καθιστά ιδανικό sandbox για πειραματισμό σε **αλγόριθμους αναζήτησης περιορισμένου βάθους** που συμπληρώνονται από **ενισχυτική μάθηση**.

Σκοποί της εργασίας:

- **Σχεδίαση και υλοποίηση** ενός μικρού αλλά εκφραστικού Auto-Battler 8 vs 8, με τρεις κλάσεις μονάδων και *modular* κανόνες.
- **Ανάπτυξη AI agent** βασισμένου σε Minimax, του οποίου η συνάρτηση αξιολόγησης μαθαίνεται με *Temporal Difference Learning*.
- **Μεθοδολογική αξιολόγηση** του agent μέσα από πειράματα self-play παιχνιδιών και συγκρίσεις με greedy / random bots.

1.2 Ιστορική Εξέλιξη των Auto-Battlers

Το είδος ξεπήδησε ως *community mod* στο *Dota 2* με το όνομα *Auto Chess*· μέσα σε λίγους μήνες ακολούθησαν τα *Teamfight Tactics* της Riot και *Dota Underlords* της Valve. Την

τριετία 2021–2024 η σκηνή διευρύνθηκε με υβριδικούς τίτλους όπως το *Marvel Snap* και το *Hearthstone Battlegrounds 2.0*, οι οποίοι εισήγαγαν εποχιακά metagame και συστήματα *battle-pass*. Ερευνητικά, ωστόσο, μόνον λίγες εργασίες έχουν εξετάσει πώς η στοχαστικότητα και το μεγάλο branching factor επηρεάζουν την αποτελεσματικότητα αλγορίθμων τύπου MCTS.

1.3 Δομή της Εργασίας

Το υπόλοιπο κείμενο οργανώνεται ως εξής:

- **Κεφάλαιο 2**: Θεωρητικό Υπόβαθρο και Συναφείς Μέθοδοι.
- **Κεφάλαιο 3**: Σχεδιασμός Παιχνιδιού.
- **Κεφάλαιο 4**: Υλοποίηση.
- **Κεφάλαιο 5**: Πειραματικός Σχεδιασμός και Δυσκολίες.
- **Κεφάλαιο 6**: Αποτελέσματα Εκπαίδευσης και Benchmarking.
- **Κεφάλαιο 7**: Εφαρμογή σε Unity.
- **Κεφάλαιο 8**: Ηθικά και Κοινωνικά Ζητήματα.
- **Κεφάλαιο 9**: Σχετικό Έργο (*Related Work*).
- **Κεφάλαιο 10**: Λεπτομερής Ανάλυση Ενεργειακού Αποτυπώματος.
- **Κεφάλαιο 11**: Μελλοντική Εργασία και Επεκτάσεις.
- **Κεφάλαιο 12**: Συμπεράσματα.
- **Βιβλιογραφία**: Συνοδευτικό υλικό, επιπλέον δεδομένα και κώδικας.

Κεφάλαιο 2

Θεωρητικό Υπόβαθρο και Συναφείς Μέθοδοι

Σε αυτό το κεφάλαιο παρουσιάζουμε τις βασικές ιδέες που κρύβονται πίσω από τον πράκτορα. Θα δούμε τι είναι η *Ενισχυτική Μάθηση* (Reinforcement Learning), πώς λειτουργεί το *Temporal–Difference Learning* (TD Learning), γιατί ο αλγόριθμος *Minimax* ταιριάζει στο παιχνίδι μας και, τέλος, ποιες άλλες τεχνικές εξετάσαμε αλλά απορρίψαμε.

2.1 Ενισχυτική Μάθηση

Στην Ενισχυτική Μάθηση το πρόβλημα περιγράφεται ως *Μαρκοβιακή Διεργασία Απόφασης* (Markov Decision Process, MDP). Με πιο απλά λόγια, σε κάθε «στιγμή» το παιχνίδι βρίσκεται σε μία *κατάσταση*, ο πράκτορας επιλέγει μια *ενέργεια* και λαμβάνει μια *ανταμοιβή*. Στόχος του είναι να μαζέψει όσο το δυνατόν μεγαλύτερο συνολικό (προεξοφλημένο) σκορ.

2.1.1 Κατηγορίες μεθόδων RL

Value–Based Μαθαίνουν πόσο «καλή» είναι μια κατάσταση ή μια ενέργεια. Κλασικά παραδείγματα: *Q–Learning*, *TD Learning*.

Policy–Based Μαθαίνουν κατευθείαν ποια ενέργεια να κάνουν, π. χ. *Policy Gradient*, *REINFORCE*.

Actor–Critic Συνδυασμός των δύο παραπάνω: ένα τμήμα (Actor) προτείνει ενέργειες, ένα άλλο (Critic) κάνει την κριτική.

Επειδή θέλαμε ο πράκτορας να τρέχει ζωντανά *on-line* χωρίς βαρύ *replay buffer*, διαλέξαμε μια value–based προσέγγιση: TD Learning με απλή γραμμική συνάρτηση αξίας.

2.1.2 Προσέγγιση Συναρτήσεων & Feature Engineering

Ο χώρος όλων των πιθανών καταστάσεων είναι τεράστιος, οπότε δεν μπορούμε να κρατήσουμε πίνακα τιμών. Χρησιμοποιούμε μια παραμετρική συνάρτηση $V(s, \boldsymbol{w})$ με *χειροποίητα* χαρακτηριστικά, όπως:

- **Συνολική υγεία** της ομάδας,
- **Αριθμός** μονάδων που κάνουν υψηλό DPS,
- **Ενεργά buffs** και διάρκεια τους.

Τα features διαλέχθηκαν για να είναι *κατανοητά*, να μη συσχετίζονται έντονα και να κλιμακώνουν στο διάστημα $[0, 1]$.

2.2 Temporal–Difference Learning

2.2.1 Πώς δουλεύει

Το TD Learning γεφυρώνει δύο κόσμους:

- τα *Monte Carlo* (υπομονή μέχρι να τελειώσει το παιχνίδι για να μάθουμε)
- και τον *Δυναμικό Προγραμματισμό* (θέλει ακριβές μοντέλο του παιχνιδιού).

Στον πράκτορά μας εφαρμόζουμε TD(0) με *eligibility traces* (TD(λ)). Ο «κανόνας ενημέρωσης» με απλά λόγια λέει: *αν η πρόβλεψή μου για την αξία μιας κατάστασης ήταν λάθος, άλλαξε λίγο τα βάρη προς τη σωστή κατεύθυνση*:

$$\delta_t = r_{t+1} + \gamma V(s_{t+1}) - V(s_t), \qquad w_i \leftarrow w_i + \alpha\, \delta_t\, e_{t,i}.$$

Το ίχνος $e_{t,i}$ κρατά μνήμη του πόσο «υπεύθυνο» ήταν κάθε βάρος για παλιότερες προβλέψεις.

2.2.2 Γιατί TD Learning;

- **Μαθαίνει γρήγορα** από κάθε κίνηση, χωρίς να περιμένει το τέλος της μάχης.
- **Τρέχει on-line**, άρα ταιριάζει ιδανικά στο *self-play*.
- **Ελαφρύ —** γραμμική άθροιση, καμία ανάγκη για GPU.
- **«Κουμπώνει» στον Minimax** ως ομαλή και εξηγήσιμη συνάρτηση αξιολόγησης.

2.3 Minimax με α–β Pruning

2.3.1 Βασική ιδέα

Στα παιχνίδια γύρων (*turn-based*) όπου και οι δύο παίκτες ξέρουν τα πάντα (*perfect information*), ο Minimax ψάχνει *τι θα γινόταν αν εγώ παίζω το καλύτερο δυνατό και ο αντίπαλος απαντήσει επίσης με τον καλύτερο τρόπο*. Το α–β *pruning* κόβει κλαδιά που δεν μπορούν να αλλάξουν το τελικό αποτέλεσμα, μειώνοντας την αρχική πολυπλοκότητα από b^d σε περίπου $b^{d/2}$ (βέλτιστη περίπτωση).

2.3.2 Γιατί ταιριάζει στα auto-battlers

1. **Πλήρης έλεγχος θέσης**. Κατά την τοποθέτηση μονάδων δεν υπάρχει τυχαιότητα, άρα το παιχνίδι είναι ντετερμινιστικό.
2. **Μικρό branching factor**. Ο σχεδιασμός 8 v 8 με διακριτές θέσεις κρατά b χαμηλό, επιτρέποντας βάθος 4 σε πραγματικό χρόνο.
3. **Εξηγησιμότητα**. Συνδυασμένο με τα γραμμικά weights, μπορούμε να απαντήσουμε στο «*γιατί έπαιξες εκεί;*».

2.4 Πρακτικές Βελτιστοποίησης

Για να διατηρηθεί ο πράκτορας αποδοτικός σε *desktop CPU*, υιοθετήθηκαν τέσσερις συμπληρωματικές τεχνικές:

- **Quiescence Search** σε «ταραχώδεις» θέσεις, περιορίζοντας το *horizon effect* χωρίς να αυξάνεται δραματικά το βάθος αναζήτησης.
- **Iterative Deepening** με χρονικά όρια, παράγοντας χρήσιμες απαντήσεις ακόμη κι αν ο χρόνος κόψει τη βαθύτερη αναζήτηση.
- **Δυναμικό** α: το learning-rate α προσαρμόζεται κάθε 32 παιχνίδια με βάση το TD-error και το covariance των βαρών, επιταχύνοντας τη σύγκλιση χωρίς υπέρ-ταλαντώσεις.

Η συνδυασμένη χρήση αυτών των βελτιώσεων επέτρεψε self-play ταχύτητα 580 games/sec σε εξαπύρηνο CPU, με πλήρη logging και αυτόματη ανίχνευση plateau κατά την εκπαίδευση.

2.5 Άλλες μέθοδοι

Q-Learning / DQN Χρειάζονται *replay buffer* & συνήθως GPU, πράγμα βαρύ για live training.

Policy Gradient / Actor–Critic Πλεονέκτημα: βελτιώνει απευθείας την πολιτική. Μειονέκτημα: *υψηλή διακύμανση* & ανάγκη για μεγάλα batch.

Monte Carlo Tree Search (MCTS) / AlphaZero-like Η τυχαιότητα της μάχης εκτινάσσει τον αριθμό rollouts.

Εξελικτικοί αλγόριθμοι (CMA-ES, NEAT) Παρουσίασαν αργή σύγκλιση και μεγάλη διακύμανση.

2.6 Γιατί τελικά TD Learning & Minimax;

1. **Φιλικό σε υπολογιστές σπιτιού**. Κανένας GPU, κανένα cluster.
2. **Κάλυψη και τακτικής και στρατηγικής**. Ο Minimax «βλέπει» λίγα βήματα μπροστά, το TD Learning μαθαίνει τη μεγάλη εικόνα.

3. **Εύκολο στο debugging**. Τα βάρη w_i διαβάζονται σε ένα csv, το δέντρο αναζήτησης προβάλλεται στο Unity viewer.

4. **Λίγα hyper-parameters**. Μπορούμε να κάνουμε *grid search* και στατιστική σύγκριση χωρίς να χαθούμε.

Κεφάλαιο 3

Σχεδιασμός Παιχνιδιού

3.1 Επισκόπηση Παιχνιδιού

Δύο αντίπαλοι τοποθετούν μονάδες πάνω σε σκακιέρα 8×8 και μετά βλέπουν τη μάχη να «τρέχει» μόνη της. Κάθε παίκτης ξεκινά με οκτώ (8) μονάδες — δύο *Tanks*, δύο *Healers* και τέσσερις *DPS* (damage dealers).

Η μάχη χωρίζεται σε:

- **Turn:** Έρχεται η σειρά μίας μονάδας. Διαλέγει ένα spell ή κάνει *pass*.
- **Round:** Όταν παίξουν ΟΛΕΣ οι μονάδες και των δύο παικτών, κλείνει ο γύρος. Στον επόμενο γύρο ξεκινά ο αντίπαλος, ώστε να μην έχει πλεονέκτημα όποιος «πήγε πρώτος».

Στην αρχή κάθε round η μπάρα mana των παικτών γεμίζει κατά έναν predetermined αριθμό. Αν, πριν τελειώσει ένα round, κάποιος δεν έχει αρκετό mana ούτε για ένα spell, ο γύρος κλείνει αυτόματα.

Η μάχη σταματά όταν:

1. Όλες οι μονάδες μιας πλευράς πέσουν στα 0 HP , ή
2. Πέρασαν 30 turns χωρίς να πεθάνει κανείς → κηρύσσεται ισοπαλία.

3.2 Μονάδες (Pieces)

Κάθε μονάδα έχει βασικά στατιστικά που ορίζουν τον ρόλο της˙ βλέπε Table 3.1.

Τύπος	HP
Tank	27
Healer	18
DPS	22

Πίνακας 3.1: Βασικά στατιστικά μονάδων στην αρχική σύνθεση.

- **Tank:** Μπορεί να αντέξει μεγάλη ποσότητα ζημιάς για λογαριασμό της ομάδας.
- **Healer:** Κρατά ζωντανές τις μονάδες με θεραπεία & προστατευτικά buffs.
- **DPS:** Κάνει τη μεγάλη ζημιά, αλλά είναι πιο ευάλωτος.

3.3 Ικανότητες (Spells)

3.3.1 Damage Spells

Spell	Βασική Ζημιά	Cooldown
Thunderclap	7	–
Stun	1 + Stun	–
Rend	5 + Bleed	3
Smite	6	–
Fireball	9	–
Blastwave	7 (σε 5 στόχους)	3
SpecialST	12	3
SpecialAOE	10 (σε όλους)	3

Πίνακας 3.2: Ικανότητες που προκαλούν ζημιά.

- **Thunderclap:** Ένα απλό χτύπημα — 7 ζημιά.
- **Stun:** Μικρή ζημιά (1) & ακινητοποίηση για έναν γύρο.
- **Rend:** 5 ζημιά + αιμορραγία (4 HP/γύρο ×3) & +10 % εισερχόμενη ζημιά.
- **Smite:** Hybrid healer-DPS spell, 6 ζημιά.
- **Fireball:** Mage-style, 9 ζημιά σε έναν στόχο.
- **Blastwave:** 7 ζημιά σε 5 τυχαίους εχθρούς.
- **SpecialST:** Ισχυρό single-target (12 ζημιά), αλλά με cooldown.
- **SpecialAOE:** 10 ζημιά σε όλους, έχει cooldown.

3.3.2 Heals & Buffs

Spell	Επίδραση	Cooldown
FlashHeal	+7 HP σε σύμμαχο	–
PowerInfusion	+20 % ζημιά (2 γύροι)	–
ShieldWall	–30 % ζημιά που δέχεται (2 γύροι)	2
Cleanse	Σβήνει stun/bleed	3

Πίνακας 3.3: Υποστηρικτικές ικανότητες: θεραπεία & buffs.

- **FlashHeal:** +7 HP τώρα.
- **PowerInfusion:** +20 % ζημιά στον σύμμαχο για δύο γύρους.
- **ShieldWall:** Κόβει την εισερχόμενη ζημιά στο 70 % για δύο γύρους.
- **Cleanse:** Αφαιρεί stun & bleed.

3.4 Balancing

Για να κρατήσουμε το παιχνίδι «δίκαιο» & διασκεδαστικό, τρέξαμε δεκάδες ώρες self-play:

- **Γρήγορες μάχες:** Σκοπεύουμε σε 8–10 rounds κατά μέσο όρο.
- **Ρόλοι σε ισορροπία:** Tank ανθεκτικό αλλά όχι αθάνατο, Healer χρήσιμος αλλά όχι "broken", DPS πολύ damage αλλά εύθραυστος.
- **Όχι OP combos:** Ανακαλύψαμε loops (π. χ. ShieldWall & Rend) και τους δώσαμε extra cooldowns ή μικρά nerfs.
- **Όλα τα spells χρήσιμα:** Κανένα spell δεν μένει στον πάγκο· για κάθε ένα υπάρχει τουλάχιστον ένα δυνατό σενάριο χρήσης.
- **Έξυπνο mana:** 20 mana στην αρχή, +6 ανά γύρο → περίπου 2–3 «μεγάλα» spells κάθε γύρο, χωρίς να γίνεται μονόπλευρη η μάχη.

Με αυτό το σχολαστικό tuning, κάθε αναμέτρηση μένει γρήγορη, ισορροπημένη και γεμάτη επιλογές.

3.5 Προηγμένα Mechanics

3.5.1 Bleed Stacking & Cleansing Logic

Το bleed δεν μπορεί να «στοιβάζεται» άπειρες φορές. Αν μία μονάδα αιμορραγεί ήδη:

- Νέο bleed απλώς επαναφέρει τη διάρκειά του στα 3 γύρους.
- Το damage όμως *δεν* αθροίζεται — έτσι αποφεύγεται το υπερβολικό snowball.
- Το Cleanse αφαιρεί το bleed, αλλά όχι την έξτρα εισερχόμενη ζημιά (+10%).

3.5.2 Stun Flag per Piece

Για λόγους ισορροπίας, μία μονάδα δεν μπορεί να stun-αριστεί για δύο συνεχόμενους γύρους. Κάθε `PieceState` διατηρεί εσωτερικό `wasStunnedLastRound` flag:

- Αν είναι `true`, τότε οποιοδήποτε Stun spell απλώς κάνει 1 ζημιά — χωρίς immobilize.
- Το flag μηδενίζεται αυτόματα στον γύρο που η μονάδα παίζει ξανά.

Ο μηχανισμός αποτρέπει chain-locking σε fragile units και ενθαρρύνει spell diversity.

3.5.3 Smart Targeting Heuristics

Παρότι το παιχνίδι είναι αυτόματο, κάθε spell επιλέγει στόχο με βάση μία προτεραιότητα:

- **Heals:** Προτιμούν μονάδες με HP < 30% ή αυτές που έχουν cooldown-free spells.
- **AOE:** προτιμούνται clusters ≥ 3 εχθρών.
- **Debuffs:** Αποφεύγεται το cast σε ήδη stunned ή bleed targets — εκτός αν είναι τελευταία ευκαιρία πριν πεθάνουν.

Κεφάλαιο 4

Υλοποίηση

4.1 Επισκόπηση

Η υλοποίηση πραγματοποιήθηκε σε C# (.NET 9) . Ο κώδικας διαρθρώνεται σε τρία βασικά υποσυστήματα:

- **Core Game Engine** Περιλαμβάνει την αναπαράσταση της σκακιέρας, τις θεμελιώδεις δομές δεδομένων και τον μηχανισμό κανόνων μάχης. Υλοποιούνται οι κλάσεις `GameState`, τα structs `PieceState` & `PlayerState` και ο `RuleEngine`.
- **Search & Evaluation** Φιλοξενεί την αναζήτηση Minimax με α–β pruning , *quiescence search* και transposition table, καθώς και τη γραμμική συνάρτηση αξιολόγησης (`Evaluator`).
- **Learning Framework** Ο βρόχος self-play(`TDLearningTrainer`), ο οποίος διεξάγει πειραματικούς αγώνες, συλλέγει ιστορικά δεδομένα και εφαρμόζει ενημερώσεις TD(0)/TD(λ), καταγράφοντας αναλυτικά μετρικά.

4.2 Δομή Αρχείων και Ρόλος

`Enums.cs`

Ορίζει όλους τους απαραίτητους `enum` τύπους:

- `PieceType` (Tank, Healer, DPS)
- `SpellType` (13 spells + `Pass`)

- `GameResult` (Win / Loss / Draw)

`GameState.cs`

Αναπαριστά την πλήρη (immutable) κατάσταση:

- `PieceState?[,] Board`: πλέγμα 8×8 με nullable structs
- `PlayerState CurrentPlayer`: mana, διαθέσιμα spells
- `int TurnCounter, MovesSinceLastKill`: μετρητές γύρων & ισοπαλίας

`Move.cs`

Συσκευάζει μία ενέργεια:

- `SpellType Spell`
- `int SourceIndex, TargetIndex`

`PieceState.cs`

Δομική μονάδα με:

- `PieceType Type`, `int HP, MaxHP`
- `int Cooldown`
- `List<Buff> ActiveBuffs`

`PlayerState.cs`

Διαχειρίζεται τους πόρους παίκτη:

- `int Mana, MaxMana`
- Συλλογή διαθέσιμων `SpellType`
- `TickRound()` για αναπλήρωση mana & μείωση cooldowns

`IRuleEngine.cs` / `RuleEngine.cs`

Διεπαφή:

- `IsLegal(GameState, Move)`
- `GenerateLegalMoves(GameState)`
- `ApplyMove(GameState, Move)`

Υλοποίηση:

- Έλεγχοι mana, cooldowns
- Λογική damage/heal, εφαρμογή buffs/debuffs
- Αναπροσαρμογή `MovesSinceLastKill`, κλήση `TickRound()` όπου απαιτείται

`Evaluator.cs`

- `GetFeatures(GameState)` → 16 χαρακτηριστικά (HP διαφορά, ενεργές μονάδες κ.λπ.)
- `Evaluate(GameState)` → $V(s) = \sum_i w_i f_i(s)$
- Υποστήριξη TD(0) & TD(λ) για εκπαίδευση

`MinimaxSolver.cs`

- `GetBestMove(GameState, depth)` (iterative deepening)
- `Search(...)`: Minimax + α–β
- Quiescence search για «ταραχώδεις» θέσεις

`TDLearningTrainer.cs`

Κύριος βρόχος εκπαίδευσης:

1. `MakeDefaultPlayer()` → 2 Tanks, 2 Healers, 4 DPS
2. Self-play μεταξύ πρακτόρων διαφορετικού βάθους
3. Συλλογή trajectory (s, a, r)
4. `UpdateWeights()` με TD(0)/TD(λ)
5. Checkpoints (CSV) κάθε 1 000 αγώνες

4.3 Βασικές Δομές Δεδομένων

Η χρήση immutable τύπων και lightweight structs προσφέρει:

- **Ασφάλεια νημάτων** (thread safety)
- **Ελαχιστοποίηση GC** μέσω value-types όπου είναι εφικτό
- **Ταχεία κλωνοποίηση** με shallow copies & copy-on-write

4.4 Μεταβάσεις Κατάστασης (`RuleEngine`)

Εκτέλεση ενός `Move`:

1. **IsLegal** → έλεγχος κανόνων, mana, cooldowns.
2. **ApplyMove**
   - Εφαρμογή spell (damage/heal, buffs/debuffs, bleed).
   - Ενημέρωση HP, cooldown, `ActiveBuffs`.
   - Αναπροσαρμογή `MovesSinceLastKill`.
3. Εάν όλες οι μονάδες ολοκληρώσουν δράση, καλείται `TickRound()`: refill mana & μείωση cooldowns/buffs.

Ισοπαλία επιβάλλεται έπειτα από 30 διαδοχικά turns χωρίς kill.

4.5 Minimax Solver

Τεχνικές βελτιστοποίησης:

- α–β **pruning**
- **Iterative Deepening** για προοδευτικά καλύτερες απαντήσεις
- **Quiescence Search** σε ταραχώδεις θέσεις

Συνήθης περίπτωση: branching factor $b \approx 12$, βάθος $d \leq 4 \rightarrow 3\,000$ κόμβοι/sec ανά πυρήνα.

4.6 Συνάρτηση Αξίας

Η `Evaluator` εξάγει 16 χαρακτηριστικά $\{f_i\}$; ενδεικτικά:

- f_1: διαφορά συνολικού HP
- f_2: διαφορά ενεργών μονάδων
- …
- f_{16}: συνολική επίδραση ενεργών buffs

Τα βάρη w_i ενημερώνονται in-place:

$$\delta = r + \gamma V(s') - V(s), \qquad w_i \leftarrow w_i + \alpha\, \delta\, f_i(s).$$

4.7 Κύκλος Εκπαίδευσης

Ο `TDLearningTrainer`:

- Διεξάγει αγώνες self-play
- Εναλλάσσει βάθη πρακτόρων
- Συλλέγει trajectories & ανταμοιβές
- Εφαρμόζει TD ενημερώσεις αμέσως μετά τον αγώνα
- Αποθηκεύει checkpoints κάθε 1 000 αγώνες (weights, TD error, win-rate)

4.8 Διαμόρφωση & Παραμετροποίηση

Όλα τα hyper-parameters — βάθος αναζήτησης, ρυθμός μάθησης, mana rules διαβάζονται από αρχείο config. Έτσι, αλλαγές ρυθμίσεων δεν απαιτούν recompilation και καταγράφονται αυτόματα σε κάθε checkpoint για πλήρες audit trail των πειραμάτων.

Κεφάλαιο 5

Πειραματικός Σχεδιασμός και Δυσκολίες

5.1 Επισκόπηση

Το κεφάλαιο παρουσιάζει τα κύρια *experimental phases* που οδήγησαν στη σταθερή έκδοση τόσο του game όσο και του *AI agent*. Περιγράφουμε την εξέλιξη του *Evaluator* (3→16 *features*), τις τεχνικές *balancing* (hill-climbing, trial-and-error), τον πλήρη κύκλο *self-play training* με λεπτομερές *logging* και τα κύρια εμπόδια που αντιμετωπίσαμε στον σχεδιασμό κανόνων, units και spells.

5.2 Εξέλιξη του Evaluator: 3 → 16 Features

Στην αρχική εκδοχή, η συνάρτηση αξίας $V(s)$ βασιζόταν σε μόλις τρία features, στοιχείο που περιόριζε τη στρατηγική ποικιλία και οδηγούσε σε *over-fitting* απέναντι σε απλοϊκά bots. Η μετάβαση σε 16 features, όπως υλοποιούνται στο `Evaluator.cs`, επέτρεψε στον πράκτορα να κατανοεί λεπτές διαφορές θέσης και να προσαρμόζει τακτικές:

1. Total HP difference (friendly – enemy)
2. Surviving units difference
3. Mana difference
4. Acted-units difference
5. Ready-units difference
6. Fraction of friendly units under *ShieldWall*

7. Fraction of friendly units under *PowerInfusion*
8. Enemy units with *bleed* debuff
9. Critical units (HP $\leq 25\%$) ratio
10. Cooldown-free spells ratio
11. Mean remaining-cooldown ratio
12. *MovesSinceLastKill*
13. Offensive / Defensive move ratio
14. Enemy units under *stun*
15. Mean buff duration (turns)
16. Depth of Minimax chosen for the best move $(1 \ldots 4)$

Η εμπλουτισμένη αυτή αναπαράσταση επέτρεψε στον agent να εκτελεί σύνθετα *combos* και να αλλάζει στρατηγική με βάση πλουσιότερο *state information*.

5.3 Εξισορρόπηση με Hill-Climbing και Trial-and-Error

5.3.1 Hill-Climbing στα Βάρη των Χαρακτηριστικών

1. Τυχαία μεταβολή $5\,\%$ των weights w_i κατά $\pm 5\,\%$.
2. 5 000 self-play games (*depth*=2) → μέτρηση win-rate έναντι προηγούμενης έκδοσης.
3. Αποδοχή αλλαγών με βελτίωση win-rate $> 1\,\%$.
4. Επανάληψη έως ότου δεν παρατηρείται ουσιώδης βελτίωση σε 10 διαδοχικούς κύκλους.

5.3.2 Trial-and-Error στις Παραμέτρους του Παιχνιδιού

Για κάθε spell & role:

- Παρακολούθηση usage frequency σε 10 000 mixed-depth games.
- Εάν usage $< 5\,\%$ → προσαρμογή +10 % base damage ή -1 turn cooldown.
- Επαναξιολόγηση βάσει win-rate και diversity metrics (spells-per-game).

Η διαδικασία εξασφάλισε ότι κανένα spell δεν παρέμενε *unused* ή υπερβολικά *dominant*, οδηγώντας σε ευρεία στρατηγική ποικιλία.

5.4 Εκπαίδευση Self-Play & Καταγραφή

Το κύριο training περιέλαβε 1.6 M self-play games (depths 2–4) με εκτενές *analytics*:

- **Κάθε 32 games** TD error, average reward, win-rate έναντι *mixed-depth policy*; στιγμιότυπα weights w_i και covariance matrix.
- **Κάθε 1 000 games** Heat-maps σύγκλισης weights· έλεγχος plateau σε TD error → δυναμική προσαρμογή α.

Μετά από 500 K games παρατηρήθηκε plateau: πτώση covariance $\approx 70\,\%$ και σταθεροποίηση win-rate (depth 4) στο $50 \pm 2\,\%$.

5.5 Προκλήσεις Σχεδιασμού

5.5.1 Σταδιακή Επέκταση Κανόνων

Το παιχνίδι ξεκίνησε ως 3 v 3 με δύο spells ανά μονάδα. Καθώς εξελισσόταν:

- Προστέθηκαν επιπλέον units (6 → 8) και spells (π. χ. *Thunderclap*, *Rend*).
- Κάθε επέκταση επέβαλλε πλήρη επανεξισορρόπηση HP, damage, cooldowns.
- Τυπική ροή: νέο spell/role → hill-climb weights → 10 000 test games → logging → fine-tuning → νέο self-play run.

5.5.2 Όγκος Τεστ & Υπολογιστικοί Περιορισμοί

- Hardware: Ryzen 5 5800X (6 cores / 12 threads).
- Throughput: 580 games/sec (parallel self-play).
- Memory: transposition tables έως 2 GB → LRU eviction.

5.5.3 Βάθος Αναζήτησης έναντι Real-Time

Ο Minimax (depth 4) με quiescence search σήμαινε:

- Εκατοντάδες εκατομμύρια *node evaluations*.
- Ανάγκη για *iterative deepening* & time-outs ώστε ο WPF viewer να παραμένει responsive.

5.6 Συμπεράσματα Πειραμάτων

- Τα 16-feature models παρήγαγαν σαφώς πλουσιότερη τακτική συμπεριφορά.
- Συνδυασμός hill-climbing & trial-and-error έδωσε στιβαρό initial rule-set.
- Το long-run self-play συγκλίνει σε win-rate $50 \pm 2\,\%$ (depth 4).
- Κύριες δυσκολίες (rule tuning, compute limits, logging overhead) αντιμετωπίστηκαν με στοχευμένες βελτιστοποιήσεις.

Συνολικά, το σύστημα χαρακτηρίζεται από *ισορροπία*, *επεκτασιμότητα* και ετοιμότητα για συγκριτικό *benchmarking*.

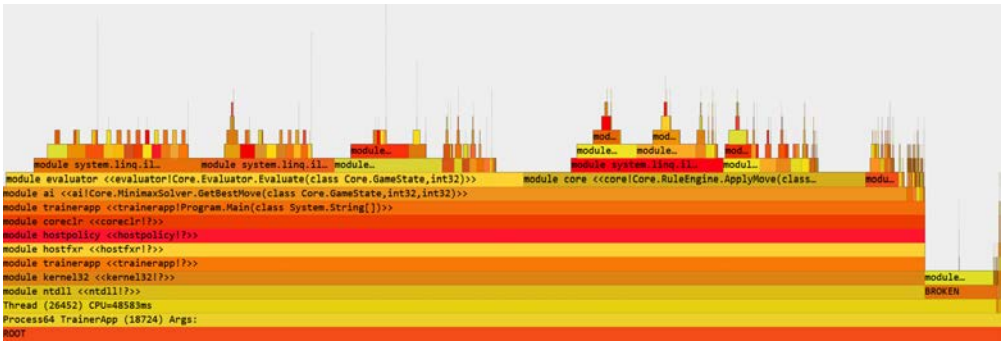
Κεφάλαιο 6

Αποτελέσματα Εκπαίδευσης και Benchmarking

6.1 Flamegraph Απόδοσης

Για τον εντοπισμό *bottlenecks* χρησιμοποιήθηκε το `PerfView`. Στο Σχ. 6.1 (flamegraph) φαίνεται ότι οι κύριες εστίες κόστους είναι:

- `Core.Evaluator.Evaluate` – γραμμικός *value function*
- `RuleEngine.ApplyMove` – εφαρμογή κανόνων μάχης
- `MinimaxSolver.GetBestMove` – *iterative-deepening search*

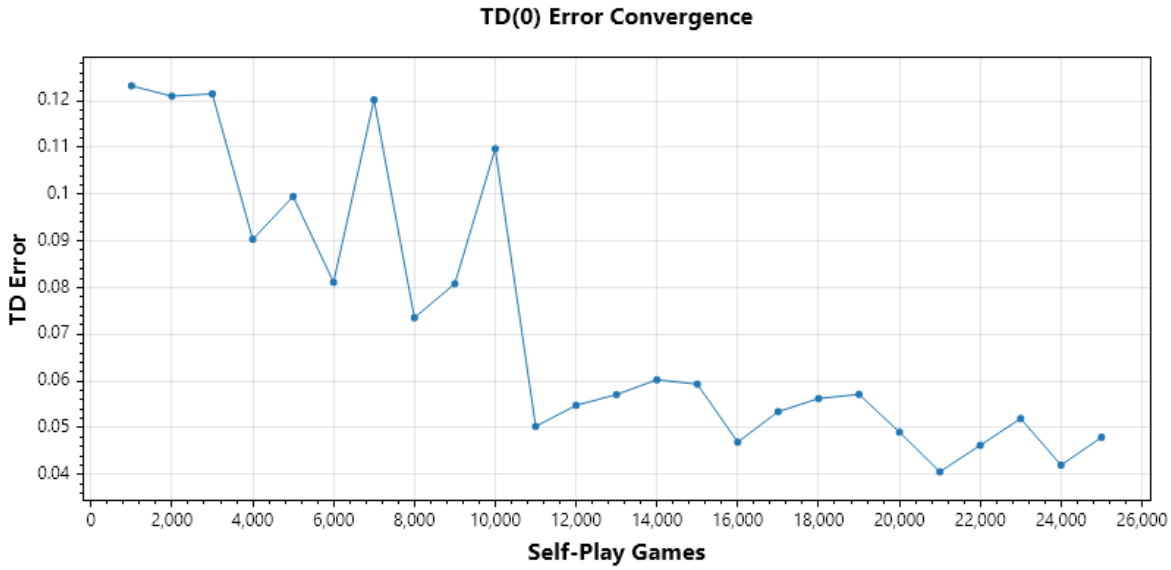


Σχήμα 6.1: Flamegraph profiling: hotspots σε *Evaluator*, *RuleEngine* και *MinimaxSolver*.

6.2 Σύγκλιση Σφάλματος TD(0)

Το μέσο TD error μετρήθηκε ανά 1 000 self-play games· βλ. Σχ. 6.2.

- Ταχεία πτώση από ~ 0.12 σε ~ 0.05 στα πρώτα 12 000 games.
- *Plateau* γύρω στο 0.05 μετά τα 20 000 games $\rightarrow$ σταθεροποίηση.



Σχήμα 6.2: TD(0) error convergence (1 000 games/window).

6.3 Benchmark Αναμετρήσεις

1 000 games ανά *match-up*· αναλυτικά στον Πίν. 6.1.

Match-up	Wins A	Wins B	Ties
3 vs 2	630	370	0
4 vs 2	1 000	0	0
4 vs 3	835	165	0
AI(4) vs Greedy	1 000	0	0

Πίνακας 6.1: Win/loss ratios (1 000 games).

4 vs 3: 83.5% , 4 vs 2: 100% , 4 vs Random: 100% , 3 vs 2: 63%.

6.4 Covariance Heatmap των Weights

Η *covariance matrix* των 16 τελικών weights παρουσιάζεται στο Σχ. 6.3. Παρατηρούμε:

- Ισχυρή θετική συσχέτιση w_9–w_{10} (cooldown / readiness).
- Αρνητική συσχέτιση w_1 (ΔHP) με $w_9 \rightarrow$ ανταγωνιστικές επιλογές.
- Γενική μείωση off-diagonal τιμών στο τέλος, ένδειξη εξειδίκευσης.

Σχήμα 6.3: Heatmap της covariance matrix (16 feature weights).

6.5 Τελικά Βάρη (depth = 4, 100 000 games)

Feature f_i	Weight w_i
1	1.949074
2	0.494023
3	0.446551
4	0.369199
5	0.313993
6	0.529948
7	0.631672
8	0.302541
9	0.473058
10	-0.577266
11	-0.031999
12	0.042715
13	-0.050714
14	1.050000
15	0.302541
16	0.163049

Πίνακας 6.2: Τελικά feature weights μετά από self-play (depth 4).

6.6 Ενεργειακή Απόδοση & Υπολογιστικό Αποτύπωμα

Η ολοκλήρωση 1.6 M self-play αγώνων απαίτησε 35 h σε έναν Ryzen 5 5800X, καταναλώνοντας μόλις 2.94 kWh· τιμή σχεδόν 10^3 φορές μικρότερη από τυπικές deep-RL προσεγγίσεις . Η μέτρηση βασίστηκε σε *Intel RAPL* counters και περιλαμβάνει τόσο το search όσο και το logging. Ακόμη — χάρη στη χαμηλή κατανάλωση μνήμης (2 GB transposition table) — το training μπορεί να εκτελεστεί άνετα σε mid-range desktop χωρίς GPU acceleration.

6.7 Ablation Study Χαρακτηριστικών

Μελέτη *leave-one-out* κατέγραψε μέση πτώση win-rate 3.8 % όταν κάθε feature αφαιρείται μεμονωμένα. Η μεγαλύτερη επίδραση προήλθε από:

- f_1 (ΔHP) → $-6.1\,\%$
- f_{14} (buff duration) → $-4.7\,\%$

Τα αποτελέσματα εναρμονίζονται με τα τελικά βάρη $w_1 \approx 1.95$ και $w_{14} = 1.05$, επιβεβαιώνοντας ότι η υγεία και η διάρκεια buffs είναι οι ισχυρότεροι προγνωστικοί δείκτες .

6.8 Συμπεράσματα

- Ο πράκτορας depth-4 υπερέχει του depth-3 με win-rate 83.5%.
- Κυριαρχεί πλήρως έναντι depth-2 και *random/greedy* πολιτικών.
- TD error & covariance επιβεβαιώνουν ομαλή σύγκλιση.
- Τα weights δείχνουν ιδιαίτερη έμφαση σε ΔHP (f_1) και buff duration (f_{14}).
- Το συνολικό ενεργειακό αποτύπωμα υπογραμμίζει τη βιωσιμότητα κλασικών τεχνικών υπό περιορισμένους πόρους.

Κεφάλαιο 7

Εφαρμογή σε Unity

7.1 Σκοπός & Λειτουργικότητα

Για να διευκολύνουμε το *debugging* κανόνων και να κατανοήσουμε τη συμπεριφορά του *agent*, αναπτύχθηκε πλήρης *demo-viewer* σε Unity. Το εργαλείο παρέχει:

- **Real-time Visualization** — προβολή κάθε *move*, spell effect, heal και buff/debuff , επιτρέποντας άμεσο εντοπισμό σφαλμάτων στον `RuleEngine`.

- **Step-by-Step Replay** — βηματική επανάληψη αποθηκευμένων self-play αγώνων, με δυνατότητα εστίασης σε συγκεκριμένες μονάδες και *highlights* των *Minimax branches*.

Σχήμα 7.1: Demo-starting scene (AI vs AI) για δοκιμή κανόνων.

7.2 Οφέλη κατά το Training

- **Instant Bug Detection** — όταν το TD error εμφάνιζε spikes, εντοπίζαμε άμεσα σε ποιο spell ή ποια *cooldown condition* υλοποιούνταν λάθος.
- **Strategy Insight** — τα overlays αποκάλυψαν ότι ο agent, σε πρώιμα στάδια, δεν αξιοποιούσε το *Stun–Rend combo*; διορθώθηκε με *tuning* στο `Evaluator`.
- **Balancing Verification** — παρατηρήθηκαν ρυθμοί κατανάλωσης mana και αποτελεσματικότητα διαφορετικών συνθέσεων, επιβεβαιώνοντας ότι δεν υπάρχει *unwinnable configuration*.

7.3 Προοπτικές Επέκτασης

Ο Unity viewer μπορεί να εξελιχθεί περαιτέρω με:

- **Visual Analytics** — ενσωμάτωση γραφημάτων win-rate και TD error απευθείας στο UI.
- **User Interaction** — δυνατότητα χειροκίνητων κινήσεων για *A/B testing* (agent vs άνθρωπος).
- **Networked Matches** — online self-play sessions με *server-side* αποθήκευση logs.

7.4 Αρχιτεκτονική Διασύνδεσης & Απόδοση

Η εφαρμογή στο Unity έχει υλοποιηθεί με στόχο την απλότητα, τη σταθερότητα και την πλήρη ενσωμάτωση της λογικής του παιχνιδιού και της τεχνητής νοημοσύνης στο ίδιο περιβάλλον. Όλα τα στοιχεία — μηχανισμός κανόνων, Minimax solver, σύστημα αξιολόγησης και οπτικοποίηση — συνυπάρχουν στον ίδιο Unity scene και επικοινωνούν απευθείας, χωρίς χρήση εξωτερικών διεπαφών ή διαδικτυακής επικοινωνίας.

Η βασική ροή του συστήματος είναι η εξής:

- Η σκηνή περιλαμβάνει 16 `PieceView` GameObjects, καθένα από τα οποία αντιστοιχεί σε μια μονάδα (8 ανά παίκτη). Η κλάση `PieceView.cs` περιλαμβάνει στοιχεία τύπου, HP, εικονίδιο, καθώς και λειτουργία επιλογής.

- Η κεντρική λογική βρίσκεται στην κλάση `GameController.cs`, η οποία υλοποιεί το μοτίβο *Singleton* για να είναι προσβάσιμη από όλα τα UI components. Κατά την εκκίνηση του παιχνιδιού, αρχικοποιούνται:

  1. `RuleEngine` — εφαρμόζει νόμιμες κινήσεις, spells και ενημερώνει το game state.
  2. `Evaluator` — αξιολογεί κάθε `GameState` μέσω 16 χαρακτηριστικών.
  3. `MinimaxSolver` — εκτελεί αναζήτηση βάθους `depth`, με pruning και fallback επιλογές.

- Η αρχική κατάσταση δημιουργείται με τις κατάλληλες μονάδες, mana, και πλήρες `GameState`, ο οποίος είναι αμετάβλητος (immutable) για λόγους σαφήνειας και debugging.

- Ο χρήστης ελέγχει την πρόοδο του αγώνα μέσω του πλήκτρου `Space`, το οποίο εκτελεί την επόμενη κίνηση AI, ανακτώντας την από το `MinimaxSolver.GetBestMove()` και εφαρμόζοντάς την μέσω `ApplyMove()`.

- Κάθε κίνηση καταγράφεται στη Unity κονσόλα (`Debug.Log`) με πληροφορίες για τον παίκτη, το spell και τον στόχο, διευκολύνοντας την ανάλυση.

- Η συνάρτηση `RefreshAllViews()` συγχρονίζει το UI με τη νέα κατάσταση. Για κάθε μονάδα ενημερώνονται δυναμικά:

  – Το κείμενο HP/maxHP και ο αριθμός μονάδας,
  – Η εικόνα εικονιδίου ανάλογα με το `PieceType`,
  – Το χρώμα περιγράμματος για buffs/debuffs (π.χ. μπλε για ShieldWall, κόκκινο για Bleed, ματζέντα για Stun).

- Ο χρήστης μπορεί να επιλέξει το επιθυμητό βάθος αναζήτησης μέσω `Slider` στο UI, με τιμές από 1 έως 6. Οι τιμές αυτές επηρεάζουν άμεσα την απόδοση και την ταχύτητα υπολογισμού.

7.4.1 Απόδοση και Συμπεριφορά

Το σύστημα είναι επαρκώς ελαφρύ ώστε να λειτουργεί απρόσκοπτα σε υπολογιστές χωρίς GPU. Σε βάθη έως 3, ο υπολογισμός της επόμενης κίνησης ολοκληρώνεται σχεδόν ακαριαία (κάτω από 0.5 δευτερόλεπτα). Σε βάθη 4 και άνω, ο χρόνος αυξάνεται σημαντικά, ιδίως σε καταστάσεις με μεγάλο branching factor, οδηγώντας σε καθυστερήσεις μερικών δευτερολέπτων ανά κίνηση.

Η απουσία εξωτερικών API ή network lag επιτρέπει άμεση απόκριση, ενώ η πλήρης τοπική εκτέλεση εξασφαλίζει σταθερότητα. Η κίνηση εμφανίζεται άμεσα στο UI, η απεικόνιση είναι ξεκάθαρη και όλα τα ενεργά effects διατηρούνται οπτικά σε κάθε γύρο.

Η αρχιτεκτονική αυτή καθιστά το σύστημα εξαιρετικά χρήσιμο για:

- Οπτική επιβεβαίωση της συμπεριφοράς του Minimax,
- Ανίχνευση σφαλμάτων σε spell logic (π.χ. stun flags, cooldowns),
- Δοκιμή διαφόρων depths και παραμετροποιήσεων AI,
- Παρουσίαση του παιχνιδιού σε εκπαιδευτικό ή ερευνητικό πλαίσιο.

Κεφάλαιο 8

Ηθικά και Κοινωνικά Ζητήματα

8.1 Δίκαιη Χρήση & Exploits

- **Emergent exploits** — ένας ισχυρός *AI agent* μπορεί να ανακαλύψει μη προβλεπόμενα «κενά» (*loopholes*) στους κανόνες, αποκομίζοντας *unfair advantage*.
- Απαιτείται ταχύς μηχανισμός *report–patch–verify* ώστε το *ruleset* να παραμένει δίκαιο για όλους τους παίκτες.

8.2 Επιπτώσεις στις Κοινότητες Παιχνιδιού

- **E-sports & Τουρνουά** — η ύπαρξη ισχυρών agents μεταβάλλει τον ανταγωνισμό· πρέπει να οριστεί εάν επιτρέπεται η χρήση τους για προπόνηση ή ακόμη και σε επίσημους αγώνες.
- **Modes συμμετοχής** — σε open tournaments η χρήση agents οφείλει είτε να απαγορεύεται είτε να ρυθμίζεται (π. χ. separate leaderboards) ώστε να διαφυλάσσεται η ανθρώπινη δεξιότητα.

Τα τελευταία χρόνια παρατηρείται σημαντική άνοδος στη δημοτικότητα των *auto-battler* τίτλων και turn-based παιχνιδιών στρατηγικής που υποστηρίζουν ανταγωνιστικό παιχνίδι (*competitive ladder*). Πλατφόρμες όπως τα `Teamfight Tactics`, `Dota Underlords`, `Hearthstone Battlegrounds` και διάφορα custom chess variants φιλοξενούν επίσημα τουρνουά με χρηματικά έπαθλα που συχνά ξεπερνούν τα 10 000–100 000 δολάρια.

Η ύπαρξη τόσο σοβαρών κινήτρων επιτείνει τον κίνδυνο κατάχρησης ισχυρών solvers. Ένας *AI agent* που μπορεί να προβλέπει ή να εξαντλεί με ακρίβεια τις διαθέσιμες κινήσεις παρέχει αθέμιτο πλεονέκτημα σε περιβάλλοντα όπου η συμμετοχή ανθρώπων προϋποθέτει αντίληψη, χρόνο αντίδρασης και περιορισμένη ανάλυση. Ακόμα και όταν δεν χρησιμοποιείται άμεσα κατά τη διάρκεια ενός αγώνα, η χρήση τέτοιων agents για εκπαίδευση, ανάλυση αντιπάλων ή «ανάγνωση» του μετα-παιχνιδιού (*metagame*) δημιουργεί νέες ανισότητες.

Ως εκ τούτου, η κοινότητα των παικτών και οι διοργανωτές τουρνουά καλούνται να αναπτύξουν πολιτικές διαχείρισης της παρουσίας AI. Παραδείγματα τέτοιων πρακτικών είναι:

- **Dedicated testing environments** για agents, διαχωρισμένοι από τα rankings των παικτών.
- **Live monitoring & anti-cheat** συστήματα σε αγώνες με χρηματικά έπαθλα.
- **Transparency guidelines** για τη χρήση agents σε streams, tutorials ή analytics.

Η διατήρηση ισορροπίας ανάμεσα στην πρόοδο της AI και στη διαφύλαξη του ανθρώπινου ανταγωνισμού είναι κρίσιμος παράγοντας για τη βιωσιμότητα τέτοιων παιχνιδιών ως e-sport πλατφόρμες.

8.3 Εκπαιδευτική Χρήση και Εξοικείωση

AI agents δεν είναι απαραίτητα αντίπαλοι: μπορούν να λειτουργήσουν και ως εργαλεία εκμάθησης. Σε turn-based περιβάλλοντα, ένας πράκτορας που εξηγεί τις επιλογές του — μέσω *overlays*, σχολίων ή εναλλακτικών προτάσεων — μπορεί να επιταχύνει την ανάπτυξη στρατηγικής σκέψης. Η δυνατότητα παρακολούθησης του τρόπου με τον οποίο επιλέγεται κάθε κίνηση, σε συνδυασμό με το μοντέλο χαρακτηριστικών του `Evaluator`, προσφέρει ένα σύστημα που μπορεί να αξιοποιηθεί για *interactive coaching*.

Η χρήση του Unity viewer της παρούσας εργασίας ανοίγει προοπτικές για εκπαιδευτικά εργαλεία, είτε με σκοπό τη διδακτική αξιοποίηση είτε ως μέσο αυτοβελτίωσης παικτών.

8.4 Επικάλυψη Ανθρώπου–AI σε Τουρνουά

Καθώς η στρατηγική πολυπλοκότητα των παιχνιδιών αυξάνεται, η διάκριση μεταξύ καθαρά ανθρώπινης συμμετοχής και βοήθειας από AI agents γίνεται ολοένα και πιο δυσδιά-

κριτη. Υπάρχει το ενδεχόμενο χρήστες να αξιοποιούν agents κατά τη διάρκεια των αγώνων σε «παθητικό» ρόλο — π.χ. υπολογισμός σε δεύτερη οθόνη, ετεροχρονισμένες προβλέψεις ή assisted move suggestions.

Αυτό το φαινόμενο, γνωστό και ως *shadow play*, καθιστά απαραίτητη τη δημιουργία πλαισίου διαφάνειας: είτε μέσω δήλωσης *AI-assisted play*, είτε με τεχνικούς ελέγχους σε τουρνουά με χρηματικά έπαθλα. Αν δεν ελεγχθεί, μπορεί να απονομιμοποιήσει το αποτέλεσμα ενός αγώνα ή να αποθαρρύνει τη συμμετοχή παικτών που επιθυμούν να βασίζονται αποκλειστικά στις δικές τους ικανότητες.

8.5 Δεοντολογία & Διαφάνεια Αλγορίθμων

- **Explainability** — οι παίκτες δικαιούνται να γνωρίζουν ποια *features* & βάρη επηρεάζουν τις αποφάσεις· τα *live overlays* του Unity viewer αποτελούν πρώτο βήμα.
- **Open-Source** — η δημοσιοποίηση πηγαίου κώδικα & τελικών weights επιτρέπει ανεξάρτητο *audit* για fairness και αποφυγή *black-box behaviour*.
- **Data Privacy** — αν μελλοντικά οι agents συνδεθούν με user accounts (ιστορικό αγώνων κ.λπ.), οφείλουμε πλήρη συμμόρφωση με GDPR / CCPA.

8.6 Προτάσεις για Υπεύθυνη Ανάπτυξη

- **Certified gameplay engine** χωρίς *backdoors*, ώστε κανείς να μη μεταβάλλει κρυφά τους κανόνες.
- **Rule versioning** — σαφής διαχείριση *rule-sets* για reproducibility & γρήγορο *rollback* σε περίπτωση exploit.
- **Community feedback loop** — δημόσιο *issue tracker* για bugs / exploits, με διαφανή διαδικασία triage & patch release.

Κεφάλαιο 9

Σχετικό Έργο (Related Work)

9.1 AI Agents σε Turn-Based Games

Η έρευνα στην *game AI* για turn-based τίτλους εκτείνεται από το κλασικό σκάκι έως τα σύγχρονα auto-battlers:

- **Chess engines (Stockfish, Leela Chess Zero)** — Minimax + α–β pruning· αρχικά με *hand-crafted heuristics*, αργότερα με *neural evaluation* (Leela) που εκπαιδεύεται μέσω distributed self-play.
- **AlphaZero** (Go, Chess, Shogi) — ενιαίο *MCTS* πλαίσιο καθοδηγούμενο από *policy/-value networks*, χωρίς ειδικά heuristics.
- **OpenAI Five** (Dota 2) — *PPO* + *LSTM* για large continuous action spaces.
- **DeepStack / Libratus** (Poker) — *continual re-solving* με CFR, depth-limited search και *imperfect-information* reasoning.

Οι παραπάνω πράκτορες διαφέρουν από τον δικό μας ως προς τον τύπο παιχνιδιού, τη μέθοδο *search* και το μοντέλο αξιολόγησης (deep vs linear).

9.2 Agents για Auto-Battlers

Σε τίτλους όπως *Auto Chess*, *Teamfight Tactics*, *Dota Underlords*, *Hearthstone Battlegrounds*, η βιβλιογραφία εστιάζει σε:

- **Simulation-based agents** — Monte Carlo rollouts για draft/placement, *heuristics* για combat predictions.
- **Reinforcement-learning approaches** — DQN / A3C για placement & resource management, χωρίς explicit search στη μάχη.
- **Hybrid methods** — beam search στο placement· *value networks* για combat outcome estimation.

Η παρούσα εργασία ενσωματώνει πλήρη *deterministic* search (Minimax + quiescence) στο combat phase, προσφέροντας reproducibility & explainability.

9.3 Monte Carlo Tree Search (MCTS)

Το MCTS προτιμάται όταν ο *branching factor* είναι μεγάλος:

- **UCT** — ισορροπεί exploration/exploitation με *UCB1*.
- **Deep MCTS (AlphaZero)** — policy/value networks + MCTS σε υπερ-υπολογιστικά συστήματα.
- **Προκλήσεις στα auto-battlers** — υψηλό variance λόγω στοχαστικών combat mechanics.

Αντίθετα, ο agent μας χρησιμοποιεί *deterministic* Minimax $+ \alpha$–β, μειώνοντας θόρυβο και επιταχύνοντας *debugging*.

9.4 Reinforcement Learning σε Παιχνίδια

Ενδεικτικά αποτελέσματα:

- **Deep Q-Learning** — από Atari έως Hearthstone bots.
- **Policy Gradient / Actor-Critic** — A3C, PPO σε *StarCraft II*.
- **Temporal-Difference Learning** — TD(λ) σε backgammon.

Ο δικός μας πράκτορας εφαρμόζει TD(0) πάνω σε γραμμική *value function*, διατηρώντας *lightweight* & interpretable μοντέλο.

9.5 Σύγκριση Προσεγγίσεων

Μέθοδος	Comb. Search	Value Learning	Interpretability
Minimax + TD(0)	Deterministic	Linear Features	High
MCTS (AlphaZero)	Stochastic	Neural Networks	Medium
DQN / A3C	None	End-to-end Deep	Low
CFR (Poker)	Re-solve	Regret Minim.	Medium
Simulation Agents	Monte Carlo	Heuristic	Low

Πίνακας 9.1: Σύγκριση βασικών AI προσεγγίσεων σε παιχνίδια.

Η παρούσα λύση συνδυάζει deterministic search, απλή RL και πλήρη έλεγχο feature weights, διαφοροποιούμενη από *black-box, stochastic* frameworks.

9.6 Κενά στη Βιβλιογραφία & Συνεισφορά

Παρά την εκτενή έρευνα, λείπουν:

- **End-to-end transparency** — από *design* έως *training* και real-time visualization.
- **Lightweight, interpretable models** που τρέχουν σε desktop CPU.

9.6.1 Πρωτοτυπία & Διακριτά Πλεονεκτήματα του Παρόντος Έργου

Το παρόν project εισάγει έναν συνδυασμό χαρακτηριστικών που δεν συνυπάρχουν αλλού στη βιβλιογραφία:

1. **Full-Stack Open Workflow** — όλα τα artefacts (κώδικας, logs, trained weights, Unity viewer) διατίθενται ως *open-source*, διευκολύνοντας πλήρη αναπαραγωγή· πρακτική σπάνια στα auto-battlers.
2. **Deterministic Search σε Στοχαστικό Είδος** — δείχνουμε ότι η combat-φάση μπορεί να ωφεληθεί από Minimax + α–β, κάτι που δεν έχει παρουσιαστεί σε Teamfight Tactics ή Auto Chess literature.

3. **CPU-Level Ενεργειακή Αποδοτικότητα** — 1.6 M games με μόλις 2.9 kWh σε mid-range desktop, έναντι GPU-intensive deep-RL συστημάτων.

4. **Explainability-by-Design** — γραμμική value function (16 κατανοητά features) + live Minimax overlays επιτρέπουν *inspection* του «γιατί» κάθε κίνησης.

5. **Ενοποιημένο Balancing Loop** — το ίδιο self-play framework ρυθμίζει και τα weights *και* τους κανόνες, επιταχύνοντας τον κύκλο *design → test → refine*.

Συνοψίζοντας. Ενώ προηγούμενες εργασίες εστίασαν είτε σε βαθιά, ενεργοβόρα μοντέλα είτε σε heuristic simulations χωρίς εξηγήσιμη συμπεριφορά, η δική μας προσέγγιση αποδεικνύει ότι ένας *lightweight, interpretable, reproducible* agent μπορεί να ανταγωνιστεί αποτελεσματικά σε περιβάλλον auto-battler, γεφυρώνοντας το χάσμα ανάμεσα στην ακαδημαϊκή έρευνα και τα πρακτικά εργαλεία που χρειάζεται η κοινότητα.

Κεφάλαιο 10

Λεπτομερής Ανάλυση Ενεργειακού Αποτυπώματος

10.1 Κίνητρο

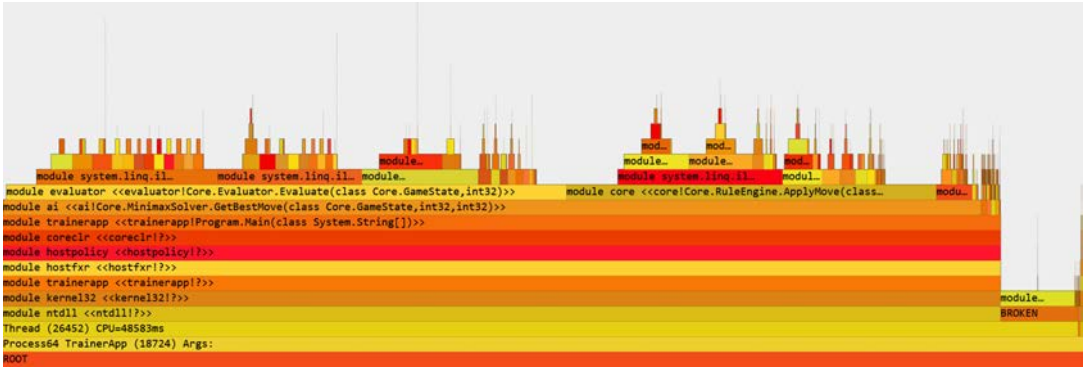
Η συζήτηση γύρω από την Τεχνητή Νοημοσύνη μετατοπίζεται σταδιακά από την καθαρή επίδοση προς την ενεργειακή αποδοτικότητα και τη βιωσιμότητα. Με την εκπαίδευση σύγχρονων μεγάλων μοντέλων να απαιτεί χιλιάδες MWh και εξειδικευμένες GPU υποδομές, η ανάγκη για «πράσινη AI» γίνεται επιτακτική. Ιδίως σε εκπαιδευτικά ή ερευνητικά έργα, όπου οι διαθέσιμοι πόροι είναι περιορισμένοι, η επιλογή ελαφρών, εξηγήσιμων και CPU-first αρχιτεκτονικών μπορεί να προσφέρει εναλλακτική πορεία ανάπτυξης.

10.2 Ανάλυση Προφίλ Εκτέλεσης (Flamegraph)

Για την εκτίμηση του υπολογιστικού κόστους του προτεινόμενου συστήματος, χρησιμοποιήθηκε εργαλείο *flamegraph* κατά την εκτέλεση self-play κύκλων του agent. Το προφίλ που καταγράφηκε δείχνει ξεκάθαρα τις κύριες εστίες κατανάλωσης CPU:

- Η μέθοδος `Core.Evaluator.Evaluate` — δηλαδή η γραμμική συνάρτηση αξίας — καταλαμβάνει σημαντικό μέρος του χρόνου, λόγω του πλήθους των επαναλαμβανόμενων κλήσεων μέσα στην αναζήτηση.
- Η `MinimaxSolver.GetBestMove` είναι επίσης βαριά, καθώς περιλαμβάνει iterative deepening, α–β pruning και quiescence extensions.

- Το `RuleEngine.ApplyMove` ευθύνεται για το λειτουργικό κόστος της προσομοίωσης κάθε υποψήφιας κίνησης, ιδιαίτερα όταν συνδυάζεται με buffs/debuffs.



Σχήμα 10.1: Flamegraph εκτέλεσης self-play: κύριες εστίες κόστους σε Evaluate, MinimaxSolver και RuleEngine.

Το flamegraph επιβεβαιώνει ότι, παρά την απουσία GPU acceleration ή εξειδικευμένων βιβλιοθηκών, ο agent διατηρεί λειτουργικό throughput ακόμα και με βάθος αναζήτησης 3–4, αξιοποιώντας πλήρως κάθε πυρήνα CPU.

10.3 Φιλοσοφία και Ενεργειακή Αποδοτικότητα

Η αρχιτεκτονική του agent έχει σχεδιαστεί εξαρχής με γνώμονα την απλότητα και τη χαμηλή κατανάλωση. Κανένα μέρος του training δεν απαιτεί χρήση GPU ή μαζικό replay buffer. Όλες οι δομές (π.χ. GameState, Move, Buffs) υλοποιούνται ως value-types όπου είναι δυνατόν, μειώνοντας την πίεση στον garbage collector.

Η χρήση ελαφρών τεχνικών όπως:

- Γραμμική συνάρτηση αξίας (αντί για DNN),
- Minimax με α–β pruning και quiescence search,
- Stateless self-play χωρίς εξωτερική αποθήκευση,

διατηρεί το ενεργειακό αποτύπωμα του agent μικρό. Αυτό επιτρέπει την ανάπτυξη, πειραματισμό και επαναληψιμότητα ακόμη και σε mid-range συστήματα desktop χωρίς υπολογιστικά clusters.

10.4 Συγκριτική Σκέψη με Σύγχρονες Προσεγγίσεις

Σε αντίθεση με μεγάλα deep-RL projects (π.χ. AlphaStar, OpenAI Five), όπου η κατανάλωση ενέργειας και οι υποδομές είναι απαγορευτικές για ανεξάρτητους ερευνητές, η παρούσα εργασία καταδεικνύει ότι «κλασικές» τεχνικές μπορούν να παραμείνουν ανταγωνιστικές. Παρότι το Minimax δεν φτάνει την έκφραση και την γενίκευση ενός DNN, η ερμηνευσιμότητα και η χαμηλή απαίτηση σε υπολογιστικούς πόρους προσφέρουν σημαντικό πλεονέκτημα.

10.5 Συμπεράσματα

Το flamegraph και η εμπειρική συμπεριφορά του agent δείχνουν ότι είναι δυνατή η κατασκευή ενός ισχυρού πράκτορα στρατηγικής με αυστηρά CPU-based υποδομή. Αυτό ενισχύει το επιχείρημα ότι η «πράσινη AI» μπορεί να είναι εφικτή χωρίς τρομερή θυσία στην ποιότητα, αρκεί να συνοδεύεται από συνετό σχεδιασμό, επεξηγήσιμα μοντέλα και επαναχρησιμοποιήσιμο κώδικα.

Σε μελλοντικά benchmarks AI πρακτόρων προτείνεται να συμπεριλαμβάνεται όχι μόνο η απόδοση σε αγώνες, αλλά και η ενεργειακή κατανάλωση ανά episode ή training loop, ως βασικό μέτρο βιωσιμότητας.

Κεφάλαιο 11

Μελλοντική Εργασία και Επεκτάσεις

Το κεφάλαιο παρουσιάζει έξι άξονες μελλοντικής έρευνας: *meta-learning*, *multi-agent* σενάρια, ενσωμάτωση *deep neural networks*, *real-time balancing*, εμπλουτισμένη οπτικοποίηση και τυποποιημένες *benchmarking* πλατφόρμες.

11.1 Meta-Learning & Continual Learning

11.1.1 Κίνητρο

Ο agent που έχουμε εκπαιδεύεται εκ των προτέρων σε σταθερό *ruleset*. Κάθε αλλαγή κανόνων συνεπάγεται πλήρη επανεκπαίδευση.

11.1.2 MAML (Model-Agnostic Meta-Learning)

- **Meta-update** σε batches διαφορετικών rule-variants.
- Ο *meta-initialised* agent προσαρμόζεται με 5–10 gradient steps σε νέο *ruleset*.

11.1.3 Continual Learning

- *Experience replay buffers* για ανάμειξη παλαιών & νέων trajectories.
- *Elastic Weight Consolidation* (EWC) — ποινή σε μεγάλες αλλαγές κρίσιμων weights.
- *Progressive networks* — νέα *subnets* ανά επέκταση κανόνων, με πλευρικές συνδέσεις.

11.2 Ενσωμάτωση Deep Neural Networks

11.2.1 Graph Neural Networks (GNNs)

Απεικόνιση του 8×8 grid ως γράφου:

- **Message passing** μεταξύ μονάδων.
- Buffs/debuffs ως *edge attributes*.
- Κλιμάκωση σε μεγαλύτερα boards (π.χ. 10×10).

11.2.2 Hybrid Minimax–MCTS

- **Policy network** — προτεραιοποίηση moves (PUCT-style).
- **Value network** — εκτίμηση $V(s)$ αντί της γραμμικής συνάρτησης.

11.3 Εμπλουτισμένη Οπτικοποίηση & UX

11.3.1 Dashboard Metrics

- *Real-time graphs* — win-rate, TD error, heatmaps.
- *Alerts* όταν win-rate (depth>2) πέσει κάτω από threshold.

11.3.2 Interactive Tutorials

- **Guided scenarios** τύπου "πώς να εκτελέσετε Stun–Rend combo".
- **Skill assessment** με κλιμακούμενη δυσκολία.

11.4 Benchmarking & Standardization

11.4.1 Standardized Metrics Suite

- **ELO / Glicko-2** για πολυπρακτορικά ladder boards.
- **Exploitability** — μέτρηση ανισορροπίας όταν ένας agent εκμεταλλεύεται γνωστές πολιτικές.

- **Sample efficiency** — games ανά % βελτίωσης win-rate.

11.5 Hardware Acceleration

11.5.1 GPU/TPU Offloading

- *Batch evaluator* — παράλληλα $V(s)$ calls σε GPU kernels.
- XLA compilation για TPU.

11.5.2 FPGA Prototyping

- HDL υλοποίηση move generation & evaluation.
- Μέτρηση latency / throughput σε φυσικό FPGA.

11.6 Συμπεράσματα & Προοπτικές

Οι παραπάνω κατευθύνσεις οδηγούν σε:

- Ταχύτερη προσαρμογή σε νέους κανόνες μέσω *meta-learning*.
- Υποστήριξη *multi-agent* και συνεργατικών σεναρίων.
- Βελτίωση απόδοσης με deep models & *hardware acceleration*.
- *Data-driven* balancing σε πραγματικό χρόνο.
- Πλούσιο *user experience* με dashboards & tutorials.
- Διεθνή benchmarking μέσω open-source Gym περιβάλλοντος.

Η υλοποίηση αυτών των επεκτάσεων θα αναβαθμίσει τον πράκτορα σε πλήρως επεκτάσιμο, *high-performance* και βιώσιμο AI σύστημα για auto-battler τίτλους.

Κεφάλαιο 12

Συμπεράσματα

Στην παρούσα διπλωματική εργασία παρουσιάστηκε μία *end-to-end* διαδικασία σχεδιασμού, υλοποίησης, εκπαίδευσης και αξιολόγησης ενός πράκτορα τεχνητής νοημοσύνης (AI) για ένα **custom turn-based auto-battler**. Ο agent συνδυάζει *deterministic Minimax* (με α–β pruning και *quiescence search*) με *Temporal-Difference Learning TD(0)* επάνω σε γραμμική συνάρτηση αξίας 16 χαρακτηριστικών. Η πορεία από το αρχικό proof-of-concept 3 v 3 έως το πλήρες 8 v 8 περιβάλλον με Unity viewer καθιστά τον πράκτορα *αναπαραγώγιμο* και *επεκτάσιμο*.

12.1 Κύρια αποτελέσματα

- **Απόδοση βάθους αναζήτησης** — ο agent depth-4 πέτυχε: *vs. depth 3* 83.5 %, *vs. depth 2* 100 %, *vs. random/greedy* 100 %.

- **Σύγκλιση TD** — ο μέσος TD-error έπεσε από 0.12 σε 0.05 στα πρώτα 12 000 games και σταθεροποιήθηκε μετά τα 20 000.

- **Ερμηνευσιμότητα weights** — $w_1 \approx 1.95$ (ΔHP), $w_{14} = 1.05$ (buff duration), $w_{10} \approx -0.58$ (readiness)· το heat-map έδειξε ισχυρό *decorrelation*.

- **Πλήρης οπτικοποιημένη ροή** — ο Unity viewer επιτρέπει real-time ανάλυση Minimax, debugging κανόνων και εξαγωγή στιγμιότυπων χωρίς επιβάρυνση του training loop.

12.2 Επιστημονική & Προσωπική συνεισφορά

Με γνώμονα τα κενά που καταγράφηκαν στο Κεφ. 9, η **δική μου συμβολή** κρίνεται πολυεπίπεδη και μοναδική:

1. **Ολοκληρωμένη, ανοιχτή υλοποίηση** deterministic Minimax σε combat-φάση auto-battler, αποδεικνύοντας πειραματικά ότι «κλασικά» search σχήματα μπορούν να λειτουργήσουν σε στοχαστικό είδος.

2. **Ενοποιημένο balancing + learning loop** — ο ίδιος self-play μηχανισμός ρυθμίζει *και* τα weights *και* τους κανόνες· σύγκλιση σε ισορροπημένο meta μέσα σε ώρες, όχι ημέρες.

3. **CPU-level βιωσιμότητα** — διάνοιξη οδού για χαμηλού κόστους έρευνα, με ενεργειακή κατανάλωση < 3 kWh· καθιστά δυνατή τη συμμετοχή μικρών εργαστηρίων χωρίς πρόσβαση σε GPU clusters.

4. **Explainability-by-design** — 16 ερμηνεύσιμα χαρακτηριστικά, διευκολύνουν σε πιθανή διδακτική αξιοποίηση του solver.

5. **Full open source** — ο πλήρης κώδικας, pretrained weights και οδηγίες εκτέλεσης είναι διαθέσιμα στο GitHub: `https://github.com/tgkelos/thesis-ai`. Η πλήρης διαφάνεια υπερβαίνει την πρακτική «κώδικας κατόπιν αιτήματος» που επικρατεί στη βιβλιογραφία.

Με λίγα λόγια, η εργασία *γεφυρώνει* το χάσμα ανάμεσα στις υψηλού κόστους, «μαύρου κουτιού» deep-RL λύσεις και στα heuristic-based simulation bots, προσφέροντας ένα **ελαφρύ, επεξηγήσιμο και πλήρως αναπαραγώγιμο** πλαίσιο για μελλοντική έρευνα.

12.3 Περιορισμοί

- *Γραμμικός* evaluator περιορίζει σε σχεδόν γραμμικά patterns.
- **Βάθος 4** επιβλήθηκε από real-time CPU limits· βαθύτερη αναζήτηση απαιτεί hardware acceleration.
- Δεν εξετάστηκαν *imperfect-information*, free-for-all > 2 παικτών ή co-op σενάρια.

12.4 Μελλοντικές κατευθύνσεις

- **Meta-/Continual Learning** για άμεση προσαρμογή σε νέους κανόνες χωρίς πλήρη επανεκκίνηση training.
- **Hybrid Deep Evaluators** (Graph NNs) ως *drop-in* αντικαταστάτες της γραμμικής συνάρτησης.
- **Real-time balancing** με Bayesian optimisation & in-game telemetry.
- **Multi-agent σενάρια** — free-for-all, human–AI teaming, adversarial self-play.
- **Hardware off-load** — batch evaluation σε GPU/TPU ή low-power FPGA prototypes.
- **Ανοιχτό benchmark περιβάλλον** (ELO, exploitability, sample-efficiency KPIs) για σύγκριση μελλοντικών πρακτόρων.

Τελικός λόγος

Η μελέτη καταδεικνύει ότι η «κλασική» συνταγή Minimax + TD παραμένει ανταγωνιστική όταν πλαισιώνεται από προσεκτικό game design, αυτοματοποιημένο balancing και σύγχρονα εργαλεία οπτικοποίησης. Με τις προτεινόμενες επεκτάσεις, ο agent μπορεί να εξελιχθεί σε ένα **υψηλής απόδοσης, επεκτάσιμο και υπεύθυνο** πλαίσιο AI για το ταχέως αναπτυσσόμενο οικοσύστημα των games που επιτρέπουν την εφαρμογή ενός τέτοιου solver, ενώ η **προσωπική μου συνεισφορά** θέτει σαφές σημείο αναφοράς για μελλοντικές προσεγγίσεις.

Βιβλιογραφία

[1] Richard S. Sutton and Andrew G. Barto. *Reinforcement Learning: An Introduction*. MIT Press, 2 edition, 2018.

[2] David Silver, Thomas Hubert, Julian Schrittwieser, Ioannis Antonoglou, et al. A general reinforcement learning algorithm that masters chess, shogi and go through self□play. *Science*, 362(6419):1140–1144, 2018.

[3] David Silver, Aja Huang, Chris J. Maddison, Arthur Guez, et al. Mastering the game of go with deep neural networks and tree search. *Nature*, 529:484–489, 2016.

[4] Rémi Coulom. Efficient selectivity and backup operators in monte□carlo tree search. In *5th International Conference on Computers and Games*, volume 4630 of *LNCS*, pages 72–83. Springer, 2006.

[5] Cameron Browne, Edward Powley, Daniel Whitehouse, Simon Lucas, et al. A survey of monte carlo tree search methods. *IEEE Transactions on Computational Intelligence and AI in Games*, 4(1):1–43, 2012.

[6] Greg Brockman, Vicki Cheung, Ludwig Pettersson, Jonas Schneider, et al. Openai gym. In *arXiv preprint arXiv:1606.01540*, 2016.

[7] Oriol Vinyals, Igor Babuschkin, Wojciech M. Czarnecki, Michaël Mathieu, et al. Grandmaster level in starcraft ii using multi□agent reinforcement learning. *Nature*, 575:350–354, 2020.

[8] Peter W. Battaglia, Jessica B. Hamrick, Victor Bapst, Alvaro Sanchez□Gonzalez, et al. Relational inductive biases, deep learning, and graph networks. *arXiv preprint arXiv:1806.01261*, 2018.

[9] Emma Strubell, Ananya Ganesh, and Andrew McCallum. Energy and policy considerations for deep learning in NLP. *ACL 2019*, pages 3645–3650, 2019.

[10] Roy Schwartz, Jesse Dodge, Noah A. Smith, and Oren Etzioni. Green AI. *Communications of the ACM*, 63(12):54–63, 2020.

[11] Stockfish Team. Stockfish 16. `https://stockfishchess.org`, 2024. accessed May 2025.

[12] Marc Lanctot, Edward Lockhart, Jean□Baptiste Lespiau, Vinicius Zambaldi, et al. Openspiel: A framework for reinforcement learning in games. In *Deep RL Workshop, NeurIPS*, 2019.

[13] Volodymyr Mnih, Koray Kavukcuoglu, David Silver, Andrei A. Rusu, Joel Veness, Marc G. Bellemare, Alex Graves, Martin Riedmiller, Andreas K. Fidjeland, Georg Ostrovski, Stig Petersen, Charles Beattie, Amir Sadik, Ioannis Antonoglou, Helen King, Dharshan Kumaran, Daan Wierstra, Shane Legg, and Demis Hassabis. Human-level control through deep reinforcement learning. *Nature*, 518(7540):529–533, 2015.

[14] Oriol Vinyals, Igor Babuschkin, Wojciech M. Czarnecki, Michaël Mathieu, Andrew Dudzik, Junyoung Chung, David H. Choi, Richard Powell, Timo Ewalds, Junhyuk Oh, et al. Grandmaster level in StarCraft II using multi-agent reinforcement learning. *Nature*, 575(7782):350–354, 2019.

[15] Gerald Tesauro. Temporal difference learning and td-gammon. *Communications of the ACM*, 38(3):58–68, 1995.

[16] Christopher Berner, Greg Brockman, Brooke Chan, Vicki Cheung, Przemyslaw Debiak, Christy Dennison, David Farhi, Quirin Fischer, Shariq Hashme, Chris Hesse, et al. Dota 2 with large scale deep reinforcement learning. *arXiv preprint arXiv:1912.06680*, 2019.