ΠΑΝΕΠΙΣΤΗΜΙΟ ΠΑΤΡΩΝ
ΤΜΗΜΑ ΗΛΕΚΤΡΟΛΟΓΩΝ ΜΗΧΑΝΙΚΩΝ ΚΑΙ ΤΕΧΝΟΛΟΓΙΑΣ ΥΠΟΛΟΓΙΣΤΩΝ
ΕΡΓΑΣΤΗΡΙΟ ΕΝΣΥΡΜΑΤΗΣ ΤΗΛΕΠΙΚΟΙΝΩΝΙΑΣ
ΟΜΑΔΑ ΤΕΧΝΗΤΗΣ ΝΟΗΜΟΣΥΝΗΣ

ΑΝΑΠΤΥΞΗ ΣΚΑΚΙΣΤΙΚΗΣ ΜΗΧΑΝΗΣ

ΔΙΠΛΩΜΑΤΙΚΗ ΕΡΓΑΣΙΑ
ΔΑΣΟΥΛΑ ΙΩΑΝΝΗ

ΕΠΙΒΛΕΠΩΝ: Κ. ΣΓΑΡΜΠΑΣ

ΠΑΤΡΑ – ΣΕΠΤΕΜΒΡΙΟΣ 2021

ΠΙΣΤΟΠΟΙΗΣΗ

Πιστοποιείται ότι η διπλωματική εργασία με θέμα:

ΑΝΑΠΤΥΞΗ ΣΚΑΚΙΣΤΙΚΗΣ ΜΗΧΑΝΗΣ

Του φοιτητή του τμήματος Ηλεκτρολόγων Μηχανικών

και Τεχνολογίας Υπολογιστών:

Δασούλα Ιωάννη (Α.Μ – 1053711)

Παρουσιάστηκε δημόσια και εξετάστηκε στο Τμήμα Ηλεκτρολόγων Μηχανικών και Τεχνολογίας Υπολογιστών στις __ /__ /__

Ο επιβλέπων και διευθυντής του Τομέα Τηλεπικοινωνιών & Τεχνολογίας Πληροφορίας

Κ. Σγάρμπας

Καθηγητής

Ευχαριστίες

Η παρούσα διπλωματική εργασία με τίτλο «Ανάπτυξη Σκακιστικής Μηχανής» εκπονήθηκε στο διάστημα μεταξύ Μαρτίου 2021 και Αυγούστου 2021 στα πλαίσια ολοκλήρωσης του Προπτυχιακού προγράμματος του τμήματος Ηλεκτρολόγων Μηχανικών και Τεχνολογίας Υπολογιστών του Πανεπιστημίου Πατρών. Σε αυτό το σημείο, θα ήθελα να ευχαριστήσω όσους συνέλαβαν στην ανάπτυξή της.

Αρχικά, τον επιβλέποντα καθηγητή, κ. Κυριάκο Σγάρμπα, για την εμπιστοσύνη του και την συνεργασία του σε ότι χρειάστηκα όσον αφορά το περιεχόμενο της εργασίας και τα διαδικαστικά της.

Επίσης, την οικογένειά μου και τους φίλους μου για την στήριξη και την βοήθειά τους σε όλο αυτό το διάστημα.

Δασούλας Ιωάννης
Αύγουστος 2021

Τίτλος:

Ανάπτυξη Σκακιστικής Μηχανής

Συγγραφέας:

Δασούλας Ιωάννης (Α.Μ 1053711)

Περίληψη:

Στην παρούσα διπλωματική εργασία παρουσιάζεται η διαδικασία ανάπτυξης μιας σκακιστικής μηχανής (Python Machine Learning Chess Engine) βασισμένη στην μηχανική μάθηση, και πιο συγκεκριμένα στην ενισχυτική και στην επιβλεπόμενη μάθηση, καθώς και μιας διαδικτυακής εφαρμογής που επιτρέπει στον χρήστη να αντιμετωπίσει τις σκακιστικές μηχανές, αλλά και να τις παρακολουθεί να παίζουν μεταξύ τους, αλλά και με την διάσημη σκακιστική μηχανή Stockfish. Η σκακιστικές μηχανές αναπτύχθηκαν στην γλώσσα προγραμματισμού Python. Συνολικά, δημιουργήθηκαν τρία διαφορετικά μοντέλα παιχνιδιού, ένα μοντέλο που εκπαιδεύτηκε παίζοντας παιχνίδια με τον εαυτό του, ένα μοντέλο που εκπαιδεύτηκε με δεδομένα από επαγγελματικά παιχνίδια του παρελθόντος αποθηκευμένα σε βάσεις δεδομένων στο διαδίκτυο και ένα συνδυαστικό μοντέλο που εκπαιδεύτηκε και με τις δύο μεθόδους. Η επιλογή της κίνησης από τις μηχανές γίνεται με την χρήση του Monte Carlo Tree Search, που αποτελεί έναν ευρετικό αλγόριθμο αναζήτησης για ορισμένα είδη διαδικασιών λήψης αποφάσεων. Επιπλέον, δημιουργήθηκε διαδικτυακή εφαρμογή παιχνιδιού έναντι των μηχανών, αλλά και παρακολούθησης παιχνιδιού μεταξύ των μηχανών. Τα αποτελέσματα αναδεικνύουν την χρησιμότητα της μηχανικής μάθησης σε περίπλοκα προβλήματα, χωρίς καμία πρώιμη γνώση της στρατηγικής και των χαρακτηριστικών του προβλήματος, αλλά και χωρίς ανάγκη για ανθρώπινη ανάλυση των παραμέτρων τους.

Abstract:

This dissertation presents the process of developing a chess engine (Python Machine Learning Chess Engine) based on machine learning, and more specifically on reinforcement and supervised learning, as well as an online application that allows the user to play against the chess engines, but also to watch them play against each other, along with the famous Stockfish chess engine. The chess engines were developed using Python programming language. In total, three different game models were created, a model that was trained by playing games against itself, a model that was trained with data from past professional games stored in online databases and a combinational model that was trained using both methods. The engines' choice of move is made using Monte Carlo Tree Search, which is a heuristic algorithm for certain types of decision-making processes. In addition, a web application was created that allows the user to play against the engines, but also to watch the engines play against

each other. The results highlight the usefulness of machine learning in complex problems, without any prior knowledge of the strategy and characteristics of the problem, but also without the need for human analysis of their parameters.

Περιεχόμενα

Κατάλογος Σχημάτων

1 Εισαγωγή

1.1 Σκάκι – Ιστορική Αναδρομή

Το σκάκι είναι ένα ψυχαγωγικό και ανταγωνιστικό επιτραπέζιο παιχνίδι στο οποίο αναμετρούνται δύο παίκτες. Συχνά αναφέρεται ως δυτικό ή διεθνές σκάκι για να διακρίνεται από παρόμοια παιχνίδια που παίζονται κυρίως στην Ασία, όπως το xiangqi. Η τρέχουσα μορφή του παιχνιδιού εμφανίστηκε στην νότια Ευρώπη κατά το δεύτερο μισό του 15ου αιώνα, αφού εξελίχτηκε από συναφή, πολύ παλαιότερα παιχνίδια Ινδικής και Περσικής καταγωγής. Σήμερα, το σκάκι είναι ένα από τα πιο δημοφιλή παιχνίδια στον κόσμο, παίζεται από εκατομμύρια ανθρώπους σε σπίτια, σε συλλόγους, διά αλληλογραφίας και σε διαγωνισμούς, ερασιτεχνικούς και μη [1].

Το σκάκι είναι ένα παιχνίδι αφηρημένης στρατηγικής και δεν περιλαμβάνει κρυφές πληροφορίες. Παίζεται σε τετράγωνη σκακιέρα με 64 τετράγωνα διατεταγμένα σε πλέγμα οκτώ προς οκτώ. Στην αρχή, κάθε παίκτης (ένας ελέγχει τους λευκούς πεσσούς, ο άλλος τους μαύρους πεσσούς) ελέγχει 16 πεσσούς: έναν βασιλιά, μία βασίλισσα, δύο πύργους, δύο ίππους, δύο αξιωματικούς και οκτώ πιόνια (στρατιώτες). Ο στόχος του παιχνιδιού είναι να κάνει «ρουά ματ» (checkmate) στον βασιλιά του αντιπάλου. Το «ρουά ματ» είναι η κατάσταση κατά την οποία ο βασιλιάς δέχεται άμεση επίθεση («ματ» ή “mate”) και, επιπλέον, δεν υπάρχει τρόπος ο βασιλιάς να διαφύγει και επομένως το παιχνίδι τελειώνει. Υπάρχουν, επίσης, διάφοροι τρόποι με τους οποίους ένα παιχνίδι μπορεί να τελειώσει με ισοπαλία.

Σχήμα 1.1: Παραδοσιακή σκακιέρα

Το οργανωμένο σκάκι εμφανίστηκε τον 19ο αιώνα. Ο διαγωνισμός σκακιού διοργανώνεται σήμερα διεθνώς από την FIDE (International Chess Federation). Η FIDE είναι το οργανωτικό σώμα του αθλήματος του σκακιού [2]. Ο πρώτος παγκοσμίως αναγνωρισμένος παγκόσμιος πρωταθλητής σκακιού, ο Wilhelm Steinitz, κατέκτησε τον τίτλο του το 1886. Ο Magnus Carlsen είναι ο σημερινός παγκόσμιος πρωταθλητής. Από την έναρξη του παιχνιδιού μέχρι σήμερα, έχει αναπτυχθεί μεγάλος όγκος θεωρίας γύρω από το παιχνίδι και την στρατηγική του. Στην σύνθεση του σκακιού συναντώνται συχνά πτυχές της τέχνης, αλλά και το σκάκι με τη σειρά του επηρέασε τη δυτική κουλτούρα και την τέχνη και έχει συνδέσεις με άλλους τομείς, όπως τα μαθηματικά, η πληροφορική και η ψυχολογία.

Ως άθλημα, το σκάκι αποτελεί ένα συναρπαστικό τρόπο δοκιμής των πνευματικών δυνάμεων των παικτών, που παίζουν με ίσους όρους, εφαρμόζοντας με ακρίβεια και επιδεξιότητα τους κανόνες του παιχνιδιού. Στην ουσία, αποτελεί ένα στρατιωτικό παιχνίδι που διεξάγεται «επί χάρτου» (σκακιέρας). Ανά την ιστορία, πολλοί γνωστοί Βασιλείς και Πάπες ήταν λάτρεις του παιχνιδιού, καθώς και εξέχουσες προσωπικότητες των γραμμάτων, επιστημών και τεχνών όπως ο Θερβάντες, ο Βολταίρος, ο Ντιντερό, ο Γκαίτε, ο Βενιαμίν Φραγκλίνος.

Το σημαντικότερο του παιγνιδιού αυτού δεν είναι τόσο η νίκη επί του αντιπάλου όσο η ποιότητα της νίκης. Ο ενθουσιασμός έγκειται στις κινήσεις εκείνες που αποβλέπουν σε επόμενες κινήσεις επί σχεδίου, οι έξυπνοι συνδυασμοί, οι λεπτοί ελιγμοί κ.λπ. που εκτιμώνται στη «σκακιστική σκέψη» ίσως με μεγαλύτερη απόλαυση που δίνουν τα «στατικά» βιβλία, πίνακες ζωγραφικής, και άλλα καλλιτεχνικά έργα. Εξ αυτού οι ωραιότερες παρτίδες που έχουν σημειωθεί τυπώνονται σε βιβλία και περιοδικά του είδους με αναλύσεις και άλλες δυνατότητες χαρίζοντας ένα ιδιαίτερο θαυμασμό αυτών. Το σκάκι καλλιεργεί και αναπτύσσει σπουδαία προτερήματα όπως πειθαρχία, διαπαιδαγώγηση της θέλησης και της αντοχής, ανάπτυξη μνημονικού, ετοιμότητα, εξυπνάδα και λογική σκέψη [3].

1.2 Κανόνες παιχνιδιού

Οι κανόνες του σκακιού είναι δημοσιευμένοι από την FIDE, στο επίσημο εγχειρίδιό της. Οι κανόνες που δημοσιεύονται από εθνικά κυβερνητικά όργανα ή από μη συνδεδεμένους οργανισμούς σκακιού, εμπορικούς κανόνες κ.λπ., ενδέχεται να διαφέρουν σε ορισμένες λεπτομέρειες. Η τελευταία φορά που αναθεωρήθηκαν οι κανόνες της FIDE ήταν το 2018.

1.2.1 Αρχική τοποθέτηση

Οι πεσσοί του σκακιού χωρίζονται σε δύο διαφορετικά χρωματισμένα σετ. Ενώ τα σετ μπορεί να μην είναι κυριολεκτικά λευκά και μαύρα (π.χ το ανοιχτόχρωμο σετ μπορεί να έχει κιτρινωπή ή υπόλευκη απόχρωση, το σκουρόχρωμο σετ μπορεί να είναι καφέ ή κόκκινο), αναφέρονται πάντα ως «λευκό» και «μαύρο». Οι παίκτες

αναφέρονται αντίστοιχα ως «λευκός» και «μαύρος». Κάθε σετ αποτελείται από τα 16 κομμάτια που αναφέρθηκαν.

Το παιχνίδι παίζεται σε έναν τετράγωνο πίνακα οκτώ σειρών και οκτώ στηλών. Συνήθως, τα 64 τετράγωνα εναλλάσσονται στο χρώμα και αναφέρονται ως λευκά και μαύρα τετράγωνα.

Οι πεσσοί τοποθετούνται αρχικά στην σκακιέρα όπως φαίνεται στο σχήμα 1.2. Έτσι, στην μεριά του λευκού, από τα αριστερά προς τα δεξιά, οι πεσσοί τοποθετούνται με την ακόλουθη σειρά: πύργος, ίππος, αξιωματικός, βασίλισσα, βασιλιάς, αξιωματικός, ίππος, πύργος στην πρώτη σειρά. Στη δεύτερη σειρά τοποθετούνται οκτώ στρατιώτες (πιόνια). Η τοποθέτηση του μαύρου αντικατοπτρίζει την αντίστοιχη του λευκού, όπως φαίνεται στο σχήμα, με τους αντίστοιχους πεσσούς στις αντίστοιχες θέσεις.

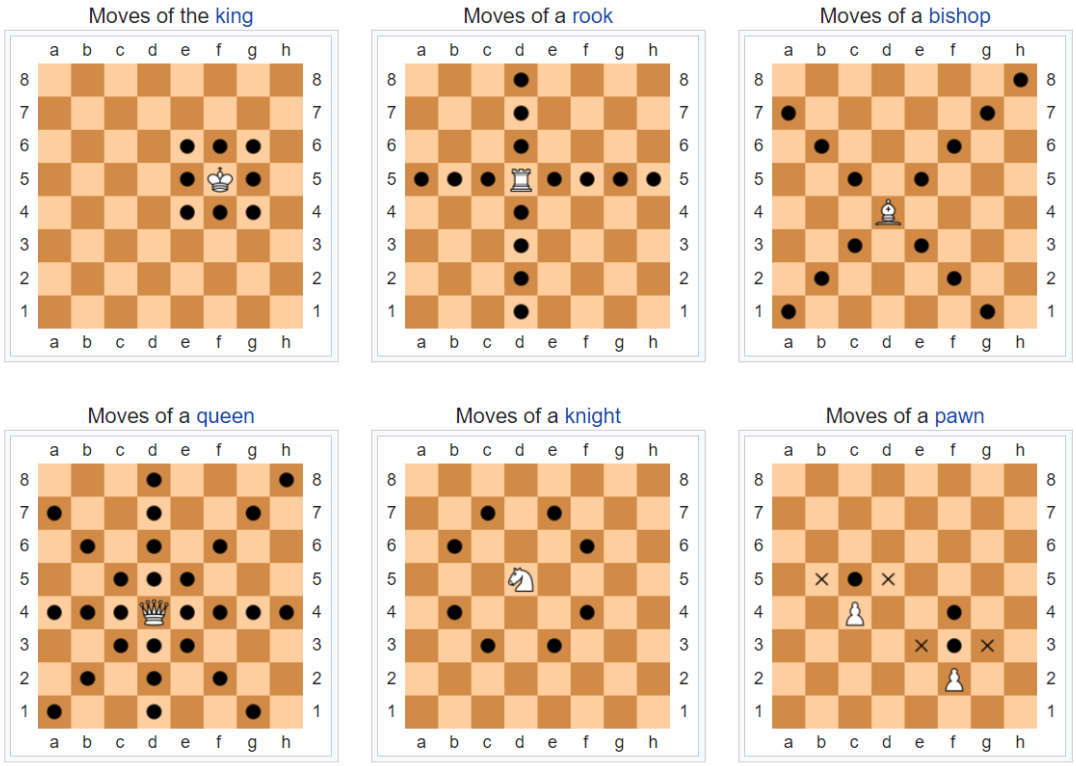
Σχήμα 1.2: Αρχική τοποθέτηση πεσσών στην σκακιέρα (Πηγή: ichess.com)

1.2.2 Κίνηση πεσσών

Το χρώμα που κάθε παίκτης θα έχει καθορίζεται τυχαία με κάποια μορφή κλήρωσης ή ορίζεται από τους διοργανωτές σε ορισμένα επίσημα τουρνουά. Ο λευκός παίκτης ξεκινάει πρώτος και έπειτα οι παίκτες παίζουν εναλλάξ, μετακινώντας έναν πεσσό ανά γύρο (εκτός από την κίνηση ροκέ, όπου μετακινούνται δύο πεσσοί). Ένας πεσσός μετακινείται είτε σε κενό τετράγωνο, είτε σε τετράγωνο κατεχόμενο από έναν πεσσό του αντιπάλου, ο οποίος και «συλλαμβάνεται» και αφαιρείται από το

παιχνίδι. Με τη μοναδική εξαίρεση της «εν διελεύσει» σύλληψης, όλοι οι πεσσοί συλλαμβάνουν μετακινούμενοι σε τετράγωνο που καταλαμβάνεται από πεσσό του αντιπάλου, το οποίο και αντικαθιστούν. Σε κάθε γύρο, η μετακίνηση είναι υποχρεωτική. Δεν μπορεί να παραληφθεί ένας γύρος, ακόμα κι αν σημαίνει ότι πρέπει να γίνει μια επιζήμια κίνηση.

Κάθε πεσσός κινείται με τον δικό του ξεχωριστό τρόπο. Στο σχήμα 1.3 φαίνονται οι επιτρεπόμενες κινήσεις, ξεχωριστά για κάθε είδος πεσσού. Οι κουκκίδες σηματοδοτούν τα τετράγωνα στα οποία μπορεί να κινηθεί ο πεσσός εάν δεν υπάρχουν παρεμβαλλόμενοι πεσσοί από οποιοδήποτε χρώμα (εκτός από τον ίππο, ο οποίος πηδά πάνω από τυχόν παρεμβαλλόμενους πεσσούς). Όλοι οι πεσσοί, εκτός από το πιόνι, μπορούν να συλλάβουν εχθρικό κομμάτι εάν βρίσκεται σε ένα τετράγωνο στο οποίο θα μπορούσαν να κινούνται εάν το τετράγωνο ήταν κενό. Τα τετράγωνα στα οποία τα πιόνια μπορούν να συλλάβουν εχθρικά κομμάτια σημειώνονται με μαύρους σταυρούς στο σχήμα.



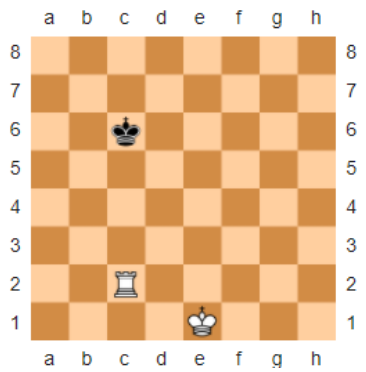
Σχήμα 1.3: Επιτρεπόμενες κινήσεις πεσσών (Πηγή: wikipedia.com)

Αναλυτικότερα, οι κινήσεις για κάθε πεσσό είναι:

- Ο βασιλιάς μετακινείται σε ένα γειτονικό τετράγωνο προς οποιαδήποτε κατεύθυνση. Υπάρχει, επίσης μια ειδική κίνηση που ονομάζεται «ροκέ» (“castling”) που περιλαμβάνει την κίνηση του βασιλιά και ενός πύργου. Ο βασιλιάς είναι ο πιο πολύτιμος πεσσός. Οι επιθέσεις εναντίον του βασιλιά πρέπει να αντιμετωπίζονται άμεσα, αλλιώς το παιχνίδι τελειώνει με ήττα.
- Ο πύργος μπορεί να μετακινείται οποιοδήποτε αριθμό τετραγώνων σε μια στήλη ή γραμμή, χωρίς να μπορεί να υπερπηδήσει άλλους πεσσούς. Μαζί με τον βασιλιά, ο πύργος συμμετέχει στο «ροκέ».
- Ο αξιωματικός μπορεί να μετακινείται οποιοδήποτε αριθμό τετραγώνων διαγώνια, χωρίς να μπορεί να υπερπηδήσει άλλους πεσσούς.
- Η βασίλισσα συνδυάζει τις κινήσεις του πύργου και του αξιωματικού, μπορώντας να μετακινείται οποιοδήποτε αριθμό τετραγώνων διαγώνια, σε στήλη και σε σειρά, χωρίς να μπορεί να υπερπηδήσει άλλους πεσσούς.
- Ο ίππος μετακινείται σε οποιοδήποτε από τα πλησιέστερα τετράγωνα που δεν βρίσκονται στην ίδια στήλη ή στην ίδια γραμμή. Έτσι, η κίνηση σχηματίζει ένα «Γ» (ή “L”), δύο τετράγωνα κάθετα και ένα τετράγωνο οριζόντια, ή δύο τετράγωνα οριζόντια και ένα τετράγωνο κάθετα. Ο ίππος είναι ο μόνος πεσσός που μπορεί να υπερπηδήσει άλλους πεσσούς.
- Το πιόνι μπορεί να μετακινηθεί προς τα εμπρός στο άδειο τετράγωνο ακριβώς μπροστά του, στην ίδια στήλη, ή κατά την πρώτη του κίνηση, μπορεί να προωθηθεί δύο τετράγωνα κατά μήκος της στήλης, αν και τα δύο τετράγωνα είναι κενά. Ένα πιόνι μπορεί να συλλάβει πεσσό αντιπάλου σε ένα τετράγωνο διαγώνια μπροστά του. Το πιόνι έχει δύο ειδικές κινήσεις: την «εν διελεύσει» σύλληψη και την προαγωγή, που εξηγούνται στη συνέχεια.

1.2.3 Ματ και Ρουά Ματ

Όταν ένας βασιλιάς δέχεται άμεση επίθεση, λέγεται ότι είναι σε ματ (“check”). Μια κίνηση-απάντηση σε ματ είναι επιτρεπτή μόνο εάν έχει ως αποτέλεσμα την διαφυγή του βασιλιά από το ματ. Η κίνηση «ροκέ» δεν επιτρέπεται όταν γίνεται ματ. Ένα παράδειγμα ματ φαίνεται στο σχήμα 1.4.



Σχήμα 1.4: Παράδειγμα ματ από τον άσπρο πύργο (Πηγή: wikipedia.com)

Ο σκοπός του παιχνιδιού είναι να γίνει ρουά ματ (“checkmate”) στον αντίπαλο. Αυτό συμβαίνει όταν ο βασιλιάς του αντιπάλου είναι σε ματ και δεν υπάρχει καμία επιτρεπτή κίνηση για να διαφύγει. Δεν επιτρέπεται ένας παίκτης να κάνει μια κίνηση που θέτει τον βασιλιά του σε ματ. Ένα παράδειγμα ρουά ματ φαίνεται στο σχήμα 1.5.

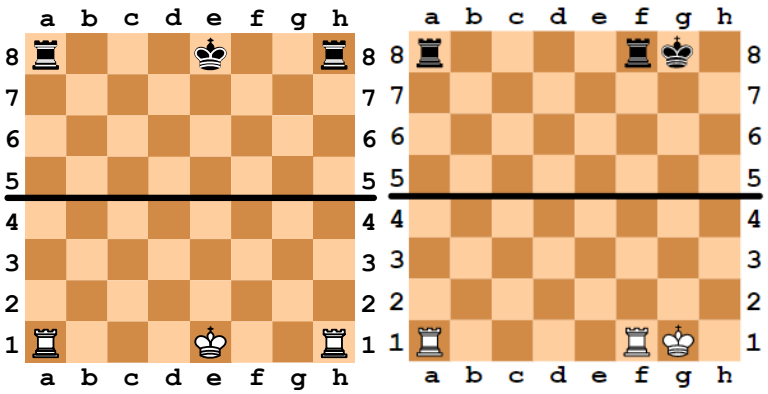
Σχήμα 1.5: Παράδειγμα ρουά ματ από τον μαύρο παίκτη (Πηγή: wikipedia.com)

1.2.4 Κίνηση ροκέ

Μία φορά σε κάθε παιχνίδι, ο βασιλιάς του κάθε παίκτη μπορεί να κάνει μια ειδική κίνηση που καλείται «ροκέ» (“castling”). Η κίνηση ροκέ γίνεται μετακινώντας τον βασιλιά δύο τετράγωνα κατά μήκος της πρώτης γραμμής του παίκτη προς έναν πύργο της ίδιας γραμμής, και στη συνέχεια ο πύργος τοποθετείται στο τελευταίο τετράγωνο που διέσχισε ο βασιλιάς.

Η κίνηση ροκέ επιτρέπεται αν πληρούνται οι ακόλουθες προϋποθέσεις:

- Ούτε ο βασιλιάς, ούτε ο πύργος δεν έχουν μετακινηθεί ακόμα, κατά τη διάρκεια του παιχνιδιού.
- Δεν υπάρχουν πεσσοί μεταξύ του πύργου και του βασιλιά.
- Ο βασιλιάς δεν είναι σε ματ και δεν θα περάσει από τετράγωνα που απειλούνται από αντίπαλους πεσσούς.



Σχήμα 1.6: Παράδειγμα επιτρεπτού ροκέ (Πηγή: wikipedia.com)

1.2.5 Εν διελεύσει σύλληψη

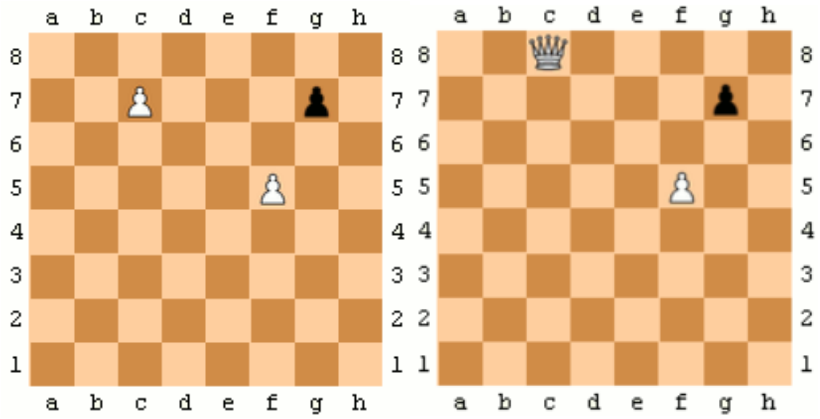
Όταν ένα πιόνι προωθείται δύο τετράγωνα μπροστά από την αρχική του θέση και υπάρχει ένα πιόνι του αντιπάλου σε ένα τετράγωνο δίπλα στο τετράγωνο του προορισμού του πιονιού, σε παρακείμενη στήλη, τότε το πιόνι του αντιπάλου μπορεί να το συλλάβει εν διελεύσει, μετακινούμενο προς το τετράγωνο που διέσχισε το πιόνι. Αυτό μπορεί να γίνει μόνο κατά τη στροφή αμέσως μετά την προώθηση του εχθρικού πιονιού, διαφορετικά χάνεται το δικαίωμα για αυτή τη σύλληψη.

Σχήμα 1.7: Παράδειγμα εν διελεύσει σύλληψης (Πηγή: wikipedia.com)

1.2.6 Προαγωγή

Όταν ένα πιόνι προχωράει στην τελευταία γραμμή, παίρνει προαγωγή και πρέπει να αντικατασταθεί με άλλο πεσσό τον οποίο ο παίκτης επιλέγει. Μπορεί να αντικατασταθεί με ίππο, αξιωματικό, πύργο ή βασίλισσα του ίδιου χρώματος.

Συνήθως, επιλέγεται η βασίλισσα, αφού αποτελεί τον πιο ισχυρό πεσσό. Αλλιώς, η κίνηση ονομάζεται underpromotion. Δεν υπάρχει περιορισμός στον αριθμό των πεσσών που είναι στην σκακιέρα, δηλαδή είναι επιτρεπτό να υπάρχουν 2 βασίλισσες, για παράδειγμα, έπειτα από προαγωγή.



Σχήμα 1.8: Παράδειγμα προαγωγής (Πηγή: wikipedia.com)

1.2.7 Τέλος παιχνιδιού

Ένα παιχνίδι μπορεί να κερδηθεί με τους ακόλουθους τρόπους:

- Με ρουά ματ
- Με την παραίτηση του αντιπάλου
- Σε παιχνίδια με χρόνο, αν τελειώσει ο διαθέσιμος χρόνος του αντιπάλου
- Σε περίπτωση που ο αντίπαλος παίκτης δεν τηρεί τους κανόνες, μπορεί να αποβληθεί από το εκάστοτε τουρνουά

Ένα παιχνίδι μπορεί να καταλήξει σε ισοπαλία με τους ακόλουθους τρόπους:

- Με «πατ» (“stalemate”). Αν ο παίκτης που είναι η σειρά του να παίξει δεν έχει καμία επιτρεπτή κίνηση, το παιχνίδι θεωρείται ότι είναι σε αδιέξοδο και το παιχνίδι σταματάει ως ισόπαλο.
- Με «νεκρή θέση» (“dead position”). Αν κανένας παίκτης δεν είναι σε θέση να κάνει ματ με οποιαδήποτε επιτρεπτή ακολουθία κινήσεων, το παιχνίδι σταματάει ως ισόπαλο.
- Με συμφωνημένη ισοπαλία. Αν και οι δύο παίκτες συμφωνήσουν ότι θέλουν το παιχνίδι να λήξει ισόπαλο, το παιχνίδι σταματάει.
- Με τριπλή επανάληψη θέσης. Όταν κανένας παίκτης δεν μπορεί να αποφύγει τις επαναλαμβανόμενες κινήσεις χωρίς να υφίσταται μειονέκτημα, και οι δύο παίκτες μπορούν να ζητήσουν το παιχνίδι να λήξει ισόπαλο από τον διαιτητή.

- Με τον κανόνα των 50 κινήσεων. Εάν κατά τη διάρκεια των προηγούμενων 50 κινήσεων δεν έχει μετακινηθεί κανένα πιόνι και δεν έχει γίνει κάποια σύλληψη, οποιοσδήποτε παίκτης μπορεί να ζητήσει ισοπαλία από τον διαιτητή. Από το 2014 και μετά, έχει τεθεί σε ισχύ και ο κανόνας των 75 κινήσεων, κατά τον οποίο ο διαιτητής πρέπει να παρέμβει και να κηρύξει ισόπαλο το παιχνίδι, αν στις τελευταίες 75 κινήσεις δεν έχει μετακινηθεί κανένα πιόνι και δεν έχει γίνει κάποια σύλληψη.
- Λόγω χρόνου. Σε παιχνίδια με χρονόμετρο, το παιχνίδι τελειώνει ως ισοπαλία εάν ένας παίκτης δεν έχει άλλο διαθέσιμο χρόνο και καμία ακολουθία επιτρεπτών κινήσεων δεν επιτρέπει στον αντίπαλα να κάνει ματ στον παίκτη.

1.2.8 Παιχνίδι ελεγχόμενο από ρολόι

Στο ανταγωνιστικό σκάκι, τα παιχνίδια παίζονται ελεγχόμενα από ρολόι. Εάν ο χρόνος ενός παίκτη τελειώσει πριν ολοκληρωθεί το παιχνίδι, το παιχνίδι χάνεται αυτόματα (με την προϋπόθεση ότι ο αντίπαλος έχει αρκετούς πεσσούς στη διάθεσή του για να φέρει το παιχνίδι σε ματ). Η διάρκεια ενός παιχνιδιού κυμαίνεται από μεγάλα (ή κλασσικά) παιχνίδια, τα οποία μπορεί να διαρκέσουν έως και 7 ώρες, έως το σκάκι «σφαίρα» (“bullet”), κατά το οποίο κάθε παίκτης έχει κάτω από 3 λεπτά για ολόκληρο το παιχνίδι.

Επίσης, υπάρχει η εκδοχή του παιχνιδιού με αύξηση χρόνου (time increment). Μετά από κάθε κίνηση προστίθεται ένα προκαθορισμένο χρονικό διάστημα στον χρόνο του παίκτη, εκτός αν ο χρόνος του παίκτη έληξε πριν ολοκληρώσει την κίνησή του.

1.3 Σκακιστικές μηχανές

Στο ηλεκτρονικό σκάκι, που παίζεται στους υπολογιστές και σε άλλες συσκευές, η σκακιστική μηχανή είναι ένα πρόγραμμα υπολογιστή που αναλύει θέσεις σκακιού ή παραλλαγής σκακιού και επιστρέφει μια κίνηση ή μια λίστα κινήσεων που θεωρεί ως ισχυρότερη. Η σκακιστική μηχανή είναι συνήθως ένα back end με μια διεπαφή γραμμής εντολών χωρίς γραφικά ή κάποιο front end. Οι σκακιστικές μηχανές χρησιμοποιούνται συνήθως με ένα ξεχωριστό front end, μια γραφική διεπαφή χρήσης όπως το Chessbase ή το WinBoard με το οποίο ο χρήστης μπορεί να αλληλοεπιδράσει μέσω πληκτρολογίου, οθόνης αφής ή ποντικιού. Αυτό επιτρέπει στον χρήστη να παίζει ενάντια σε πολλές μηχανές, χωρίς να χρειάζεται να μαθαίνει ένα νέο περιβάλλον εργασίας για κάθε μια μηχανή και επιτρέπει σε διάφορες μηχανές να παίζουν μεταξύ τους.

1.3.1 Ιστορική αναδρομή

Η έννοια του όρου «σκακιστική μηχανή» εξελίχτηκε με την πάροδο του χρόνου. Το 1986, η Linda και ο Tony Sherzer παρουσίασαν το πρόγραμμά τους Bebe στο 4ο Παγκόσμιο Πρωτάθλημα Σκακιού Υπολογιστών, όπου το έτρεξαν στο “Chess Engine”, το brand name για το hardware υπολογιστή σκακιού που κατασκευάστηκε και διατέθηκε στην αγορά από την εταιρεία τους Sys-10. Μέχρι το 1990, οι προγραμματιστές των Deep Blue, Feng-hsiung Hsu και Murray Campbell, αναφέρονταν στο πρόγραμμά τους ως «μηχανή αναζήτησης», δίνοντας, έτσι, περισσότερο βάρος στο software και όχι στο hardware.

Τον Δεκέμβριο του 1991, η εταιρεία Computer-schach & Spiele αναφέρθηκε στην Fritz, που είχε κυκλοφορήσει πρόσφατα από την Chessbase, ως “Schach-motor” που στα γερμανικά σημαίνει σκακιστική μηχανή. Στις αρχές του 1993, ο Marty Hirsch έκανε διάκριση μεταξύ εμπορικών προγραμμάτων σκακιού, όπως το Chessmaster 3000 ή το Battle Chess, αφενός, και «σκακιστικές μηχανές», όπως το ChessGenius ή το δικό του MChess Pro, αφετέρου. Στον χαρακτηρισμό του, τα εμπορικά προγράμματα σκακιού ήταν χαμηλής τιμής, είχαν φανταχτερά γραφικά, αλλά δεν τοποθετήθηκαν ψηλά στις λίστες αξιολόγησης SSDF (Swedish Chess Computer Association) ενώ οι μηχανές ήταν πιο ακριβές και είχαν υψηλές βαθμολογίες.

Το 1994, ο Shay Bushinsky δούλευε σε μια πρώιμη έκδοση του προγράμματος του: “Junior”. Ήθελε να επικεντρωθεί στο παιχνίδι του σκακιού παρά στα γραφικά, και έτσι ρώτησε τον Tim Mann πώς θα μπορούσε να κάνει τον Junior να επικοινωνήσει με το Winboard. Η απάντηση του Tim αποτέλεσε τη βάση για αυτό που έγινε γνωστό ως Chess Engine Communication Protocol ή Winboard μηχανές, αρχικά ένα υποσύνολο της διεπαφής γραμμής εντολών GNU Chess.

Επίσης το 1994, ο Stephen J. Edwards κυκλοφόρησε την Portable Game Notation (PGN). Αναφέρει ότι τα προγράμματα ανάγνωσης PGN δεν χρειάζεται να διαθέτουν "πλήρη μηχανή σκακιού". Αναφέρει επίσης τρία "γραφικά περιβάλλοντα χρήστη" (GUI): XBoard, pgnRead και Slappy.

Μέχρι τα μέσα της δεκαετίας του 2000, οι μηχανές είχαν γίνει τόσο δυνατές που κατάφεραν να νικήσουν ακόμη και τους καλύτερους παίκτες. Το 2005, ο Μάικλ Άνταμς, ένας από τους κορυφαίους 10 παίκτες στον κόσμο εκείνη την εποχή, ηττήθηκε συνολικά 5 φορές από την ΄Υδρα, ενώ μόνο ένα παιχνίδι έληξε ισόπαλο. Οι αγώνες ανθρώπου εναντίον μηχανής είναι πλέον σπάνιοι, λόγω της ανωτερότητας των μηχανών, και οι μηχανές θεωρούνται όλο και περισσότερο εργαλεία ανάλυσης και όχι αντίπαλοι.

1.3.2 Πρωτόκολλο διεπαφής

Ένα πρωτόκολλο είναι μια επίσημη περιγραφή των μορφών ψηφιακών μηνυμάτων και των κανόνων για την ανταλλαγή αυτών των μηνυμάτων εντός ή μεταξύ υπολογιστικών συστημάτων. Ένα πρωτόκολλο περιγράφει τη σύνταξη, τη σημασιολογία και το συγχρονισμό της επικοινωνίας. Η φύση της επικοινωνίας, τα πραγματικά δεδομένα που ανταλλάσσονται και τυχόν συμπεριφορές που εξαρτώνται από την κατάσταση καθορίζονται από μια προδιαγραφή πρωτοκόλλου, οι κανόνες μπορούν να εκφραστούν με αλγόριθμους και δομές δεδομένων.

Ο στόχος των πρωτοκόλλων σκακιού υπολογιστή είναι να ορίσουν ένα πρότυπο για να επιτρέψουν σε μια μηχανή σκακιού να επικοινωνεί με διεπαφές χρήστη ή γραφικών (GUI), για να επιτρέπεται στις μηχανές να παίζουν αυτόματα σε έναν υπολογιστή, μέσα σε ένα δίκτυο υπολογιστών ή στο διαδίκτυο. Οι μηχανές σκακιού, οι οποίες συνήθως υποδηλώνονται ως θυγατρική διαδικασία μιας εφαρμογής GUI, χρησιμοποιούν τυπικές ροές ή αγωγούς για να λαμβάνουν και να αποκρίνονται συμβολοσειρές ASCII ως μηνύματα [4].

Το 1995, η Chessbase κυκλοφόρησε μια έκδοση του προγράμματος βάσης δεδομένων που περιλαμβάνει το Fritz 4 ως ξεχωριστή μηχανή. Αυτή ήταν η πρώτη εμφάνιση του πρωτοκόλλου Chessbase. Λίγο αργότερα, πρόσθεσαν τις μηχανές Junior και Shredder στη σειρά προϊόντων τους, συμπεριλαμβανομένων μηχανών στο πρωτόκολλο CB ως ξεχωριστά προγράμματα που θα μπορούσαν να εγκατασταθούν στο πρόγραμμα Chessbase ή σε ένα από τα άλλα GUI τύπου Fritz. Η Fritz 1-14 εκδόθηκε μόνο ως μηχανή Chessbase, ενώ οι Hiarcs, Nimzo, Chess Tiger και Crafty έχουν μεταφερθεί σε μορφή Chessbase παρόλο που ήταν μηχανές UCI ή Winboard. Πρόσφατα, η Chessbase άρχισε να συμπεριλαμβάνει μηχανές Universal Chess Interface (UCI) στα προγράμματα της, όπως Komodo, Houdini, Fritz 15–16 και Rybka, αντί να τις μετατρέπει σε μηχανές Chessbase.

Το 2000, οι Stefan Meyer-Kahlen και Franz Huber κυκλοφόρησαν το Universal Chess Interface, ένα πιο λεπτομερές πρωτόκολλο που εισήγαγε ένα ευρύτερο σύνολο χαρακτηριστικών. Το Chessbase αμέσως μετά σταμάτησε την υποστήριξη για μηχανές Winboard και πρόσθεσε υποστήριξη για UCI στα προγράμματα GUI και Chessbase του κινητήρα τους. Οι περισσότεροι από τους κορυφαίους κινητήρες είναι UCI αυτές τις μέρες: Stockfish, Komodo, Leela Chess Zero, Houdini, Fritz 15-16, Rybka, Shredder, Fruit, Critter, Ivanhoe και Ruffian [5].

1.3.3 Λειτουργία σκακιστικής μηχανής

Οι σκακιστικές μηχανές είναι περίπλοκα προγράμματα και μπορούν να χωριστούν σε παραδοσιακές σκακιστικές μηχανές και μηχανές που αξιοποιούν μεθόδους μηχανικής μάθησης, οι οποίες άρχισαν να χρησιμοποιούνται μόλις τα

τελευταία χρόνια. Η κύρια λειτουργία τους είναι αφενός η αξιολόγηση μιας θέσης και αφετέρου η αναζήτηση για την καλύτερη δυνατή κίνηση.

Οι παραδοσιακές μηχανές, αρχικά, εξετάζουν μεμονωμένες θέσεις και αξιολογούν ποια θέση είναι καλύτερη. Σχεδόν όλες οι μηχανές σκακιού εμφανίζουν έναν αριθμό αξιολόγησης, ή «eval», με βάση την ίδια βαθμολογία που χρησιμοποιούν οι περισσότεροι παίκτες σκακιού. Κάθε μηχανή σκακιού το κάνει διαφορετικά, αλλά οι περισσότερες μηχανές βλέπουν πράγματα όπως πεσσοί σε κάθε πλευρά, όλες τις απειλές στο ταμπλό, την ασφάλεια του βασιλιά και τη δομή του πιόνι.

Η πιο απλή μορφή αξιολόγησης μιας θέσης είναι η απλή καταμέτρηση των πεσσών του άσπρου και του μαύρου, πολλαπλασιάζοντας τον αριθμό των πεσσών με ένα βάρος, ανάλογα με την ισχύ του πεσσού. Για παράδειγμα, η αξιολόγηση μπορεί να προέρχεται από την γραμμική σχέση:

$$f(x)= w1*f1(x) + w2*f2(x)+ w3*f3(x)+ ... \text{ , όπου:}$$

$$f1(x) = \Sigma(\text{άσπρες βασίλισσες}) - \Sigma(\text{μαύρες βασίλισσες})$$

$$f2(x) = \Sigma(\text{άσπροι πύργοι}) - \Sigma(\text{μαύροι πύργοι})$$

$$f3(x) = \Sigma(\text{άσπροι αξιωματικοί}) - \Sigma(\text{μαύροι αξιωματικοί})$$

$$f4(x) = \Sigma(\text{άσπροι ίπποι}) - \Sigma(\text{μαύροι ίπποι})$$

$$f5(x) = \Sigma(\text{άσπρα πιόνια}) - \Sigma(\text{μαύρα πιόνια})$$

$$w1=9$$

$$w2=5$$

$$w3=3$$

$$w4=3$$

$$w5=1$$

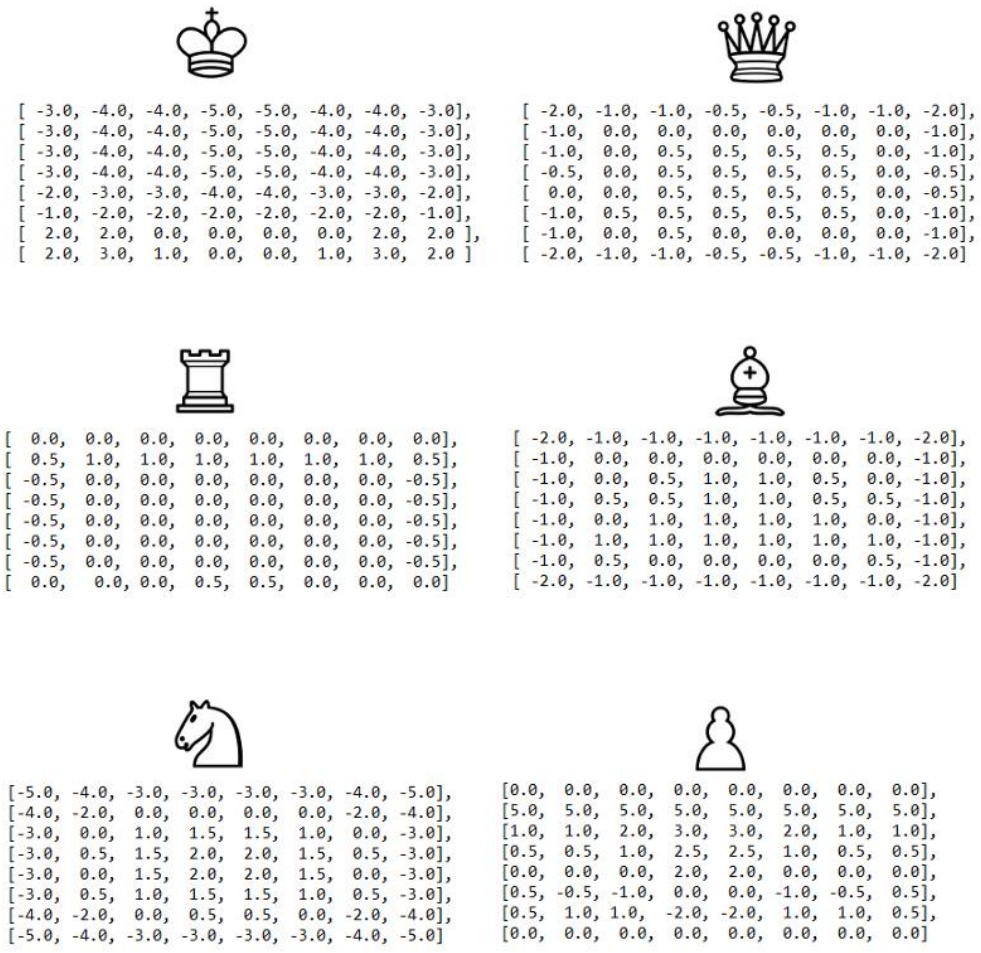
Τα βάρη παραδοσιακά είναι έτσι για τους πεσσούς, με κάποιες μικρές παραλλαγές να υπάρχουν ανά τον κόσμο.

Παρόλα αυτά, ενώ είναι σημαντική η καταμέτρηση των πεσσών, η πολυπλοκότητα του παιχνιδιού και η μεγάλη σημασία της θέσης τους στο ταμπλό, και όχι μόνο της παρουσίας τους, καθιστούν ανεπαρκείς τέτοιες απλές συναρτήσεις αξιολόγησης για τις σκακιστικές μηχανές. Για το λόγο αυτό οι μηχανές χρησιμοποιούν μια προηγμένη αξιολόγηση όπου χρησιμοποιούνται διαφορετικές τεχνικές:

- Αξιολόγηση ανά φάση παιχνιδιού. Είναι χρήσιμο να διαφοροποιούνται τα βάρη των συναρτήσεων ανάλογα με την τρέχουσα φάση του παιχνιδιού. Για

παράδειγμα, είναι καλύτερο ο βασιλιάς να απέχει πολύ από το κέντρο στη μέση της διάρκειας του παιχνιδιού. Ωστόσο, είναι ένα βασικό κομμάτι στο τέλος και, στη συνέχεια, είναι καλύτερο να βρίσκεται στο κέντρο. Για να μετρηθεί η τρέχουσα φάση του παιχνιδιού, οι μηχανές μπορούν, για παράδειγμα, να χρησιμοποιούν τον αριθμό των πεσσών που παραμένουν στο ταμπλό. Λίγοι πεσσοί σημαίνει πως το παιχνίδι βρίσκεται στο τελικό στάδιό του.

- Ζευγάρι αξιωματικών. Μπορεί να προστεθεί ένα μικρό μπόνους αν και οι δύο αξιωματικοί είναι στο ταμπλό (και ένα επιπλέον εάν οι αξιωματικοί καλύπτουν πολλά τετράγωνα στο ταμπλό).
- Πίνακες αξιολόγησης θέσης για κάθε είδος πεσσού. Υπάρχουν εύκολοι τρόποι εκχώρησης τιμών σε συγκεκριμένους πεσσούς σε συγκεκριμένα τετράγωνα. Για παράδειγμα, στα αρχικά πιόνια μπορεί να δίνεται ένα μικρό μπόνους αν καταλαμβάνουν κεντρικά τετράγωνα. Παρακάτω, φαίνεται ένα παράδειγμα πινάκων αξιολόγησης για τους άσπρους πεσσούς, για τις ανάλογες θέσεις στην σκακιέρα. Θετικός αριθμός σημαίνει καλή αξιολόγηση και αρνητικός κακή αξιολόγηση.



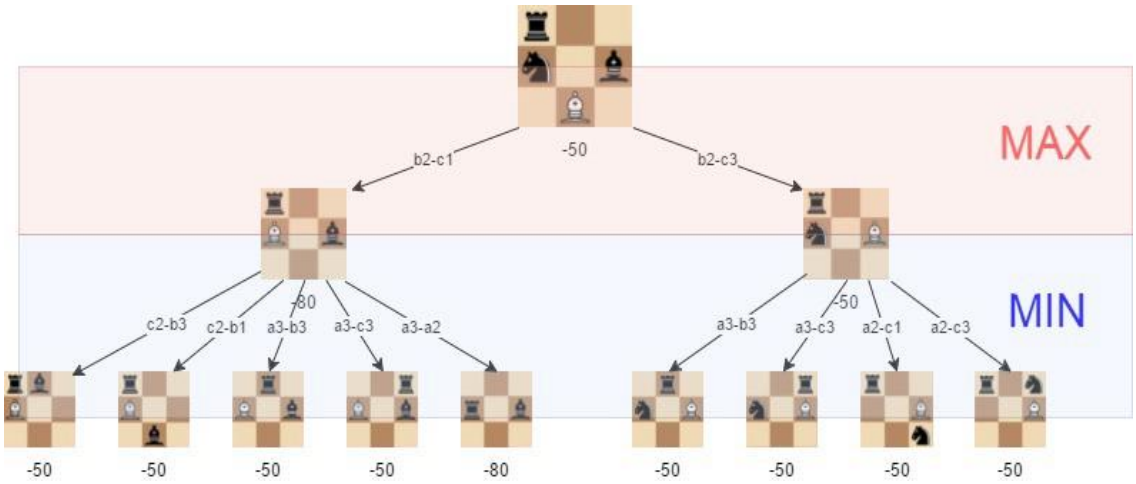
Σχήμα 1.9: Πίνακες αξιολόγησης άσπρων πεσσών (Πηγή: https://www.freecodecamp.org/news/simple-chess-ai-step-by-step-1d55a9266977/)

- Ασφάλεια του βασιλιά. Αυτό είναι πολύ σημαντικό. Για παράδειγμα, μπορεί να μετρηθεί υπολογίζοντας τον αριθμό των πεσσών που περιβάλλουν τον βασιλιά ή αν υπάρχει ένας πύργος κοντά.
- Κινητικότητα. Συνήθως προτιμώνται θέσεις στις οποίες υπάρχουν περισσότερες επιλογές, για παράδειγμα με ανοιχτές διαγώνιες για τους αξιωματικούς. Αυτό μπορεί να μετρηθεί, χρησιμοποιώντας τον αριθμό των επιτρεπτών κινήσεων που διατίθενται σε μια θέση ως βαθμολογία κινητικότητας.
- Δομή πιονιών. Τα πιόνια που καταλήγουν στην ίδια στήλη με άλλα πιόνια μπορούν να τιμωρηθούν ελαφρώς, όπως και τα απομονωμένα πιόνια στο τέλος του παιχνιδιού.
- Πύργοι σε ανοιχτές γραμμές. Είναι επιθυμητό ένας πύργος να είναι σε μία ανοιχτή γραμμή, καθώς μπορεί να κινηθεί παντού εκεί.

Υπάρχουν πολλοί περισσότεροι παράγοντες που μπορούν να ληφθούν υπόψη, αλλά αυτοί οι επτά δίνουν μια ιδέα της προηγμένης αξιολόγησης που μπορεί να προστεθεί στη λειτουργία αξιολόγησης.

Το επόμενο βήμα μιας σκακιστικής μηχανής είναι η αναζήτηση της καλύτερης δυνατής επόμενης θέσης για τον παίκτη. Όπως και οι καλοί παίκτες σκακιού, οι μηχανές προσπαθούν να αναλύσουν όσο καλύτερα μπορούν τη θέση τους. Όσο πιο μπροστά μπορούν να δουν, τόσο καλύτερη είναι η κίνηση που μπορούν να κάνουν, καθώς μπορούν να αξιολογήσουν θέσεις που θα προκύψουν μετά τις καλύτερες δυνατές κινήσεις στο μέλλον. Κάθε μεμονωμένη κίνηση σκακιού ονομάζεται "ply" (ένα επίπεδο), και το βάθος αναζήτησης εξηγείται σε πόσα ply βαθιά έγινε η αναζήτηση. Σε 20 ply (10 λευκές κινήσεις και 10 μαύρες κινήσεις), οι περισσότερες μηχανές αξιολογούν ήδη πολύ βαθύτερα και ισχυρότερα από τους ανθρώπους. Ανάλογα με τον επιτρεπόμενο χρόνο και την πολυπλοκότητα της θέσης, οι κινητήρες μπορούν να έχουν βάθος πάνω από 50 επίπεδα.

Παραδοσιακά, στο σκάκι χρησιμοποιείται ο Minimax αλγόριθμος για την ανάλυση του δέντρου θέσεων. Ο Minimax είναι ένας κανόνας απόφασης που χρησιμοποιείται στην τεχνητή νοημοσύνη, στη θεωρία αποφάσεων, στη θεωρία παιχνιδιών, στη στατιστική και στη φιλοσοφία για την ελαχιστοποίηση της πιθανής απώλειας για ένα σενάριο χειρότερης περίπτωσης (μέγιστη απώλεια). Σε αυτό που ονομάζεται αναζήτηση βάθους 1, δηλαδή όπου εξετάζεται μόνο μία κίνηση μπροστά, το παιχνίδι κοιτάζει απλώς την αξιολόγηση μετά από κάθε πιθανή κίνηση. Επιλέγεται η κίνηση με την καλύτερη αξιολόγηση. Ωστόσο, σε μια αναζήτηση δύο βάθους 2, όπου κινείται και ο αντίπαλος (ο min player), είναι πιο περίπλοκη. Αυτός ή αυτή επιλέγει επίσης μια κίνηση σύμφωνα με την καλύτερη αξιολόγηση. Αυτή η διαδικασία συνεχίζεται ανάλογα με το βάθος που είναι επιθυμητό το δέντρο να φτάσει. Κάθε φορά επιστρέφεται είτε τη μικρότερη είτε τη μεγαλύτερη τιμή του κόμβου-παιδιού στον γονικό κόμβο, ανάλογα με το αν είναι η σειρά του λευκού ή του μαύρου να παίξει. Δηλαδή, γίνεται προσπάθεια ελαχιστοποίησης ή μεγιστοποίησης του αποτελέσματος σε κάθε επίπεδο.

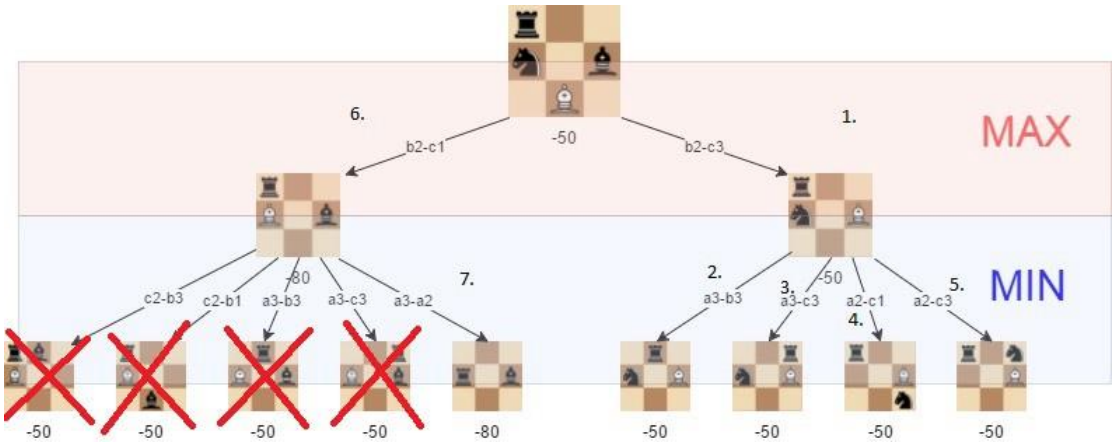


Σχήμα 1.10: Αλγόριθμος Minimax στο σκάκι (Πηγή: https://www.freecodecamp.org/news/simple-chess-ai-step-by-step-1d55a9266977/)

Οι πολλαπλές πιθανές κινήσεις που κάθε θέση έχει, δημιουργούν ένα δέντρο με πολλούς κόμβους που μεγαλώνει με μεγάλο ρυθμό ανάλογα με το βάθος του δέντρου. Αυτό έχει ως αποτέλεσμα η ανάλυση του δέντρου σε μικρό χρόνο να γίνεται πολύ δύσκολη, λόγω των πολλών υπολογισμών που απαιτεί.

Για αυτόν τον λόγο, χρησιμοποιούνται βελτιώσεις του Minimax αλγορίθμου, που επιτρέπουν την παραγωγή αποτελεσμάτων σε λιγότερο χρόνο. Η πιο διαδεδομένη μέθοδος είναι το κλάδεμα α-β (“α-β pruning”). Το κλάδεμα α-β είναι μια μέθοδος βελτιστοποίησης για τον αλγόριθμο minimax που επιτρέπει να αγνοούνται ορισμένοι κλάδοι στο δέντρο αναζήτησης. Αυτό βοηθάει να αξιολογηθεί το ελάχιστο δέντρο αναζήτησης πολύ βαθύτερα, ενώ χρησιμοποιούνται οι ίδιοι πόροι. Βασίζεται στην κατάσταση όπου μπορεί να σταματήσει η αξιολόγηση ενός μέρους του δέντρου αναζήτησης εάν εντοπιστεί μια κίνηση που οδηγεί σε χειρότερη κατάσταση από μια κίνηση που ανακαλύφθηκε προηγουμένως. Το κλάδεμα α-β δεν επηρεάζει το αποτέλεσμα του αλγορίθμου Minimax - το κάνει πιο γρήγορο. Είναι, επίσης, πιο αποτελεσματικό εάν τυχαίνει να γίνει επίσκεψη πρώτα στα μονοπάτια που οδηγούν σε καλές κινήσεις.

Η εξοικονόμηση κατά τη χρήση αυτού του αλγορίθμου είναι σημαντική. Ας υποθέσουμε ότι ένα δέντρο αναζήτησης Minimax έχει x κόμβους. Οι κόμβοι του αλγορίθμου α-β σε καλογραμμένο κώδικα μπορεί να είναι η τετραγωνική ρίζα του x. Η απόδοση εξαρτάται από το πόσο καλά ταξινομημένο είναι το δέντρο αναζήτησης, αλλά και την υπολογιστική δύναμη της συσκευής που η σκακιστική μηχανή τρέχει.



Σχήμα 1.11: Αλγόριθμος Minimax με α-β κλάδεμα στο σκάκι (Πηγή: https://www.freecodecamp.org/news/simple-chess-ai-step-by-step-1d55a9266977/)

Ένα άλλο είδος σκακιστικής μηχανής που είναι ιδιαίτερα δημοφιλές τα τελευταία χρόνια είναι οι σκακιστικές μηχανές βασισμένες στα νευρωνικά δίκτυα. Για δεκαετίες, τα νευρωνικά δίκτυα θεωρούνταν κατώτερα σε σχέση με τις παραδοσιακές μηχανές brute-force αξιολόγησης, επειδή η εκπαίδευση ενός ισχυρού νευρωνικού δικτύου απαιτούσε υπερβολική υπολογιστική ισχύ και τα νευρωνικά δίκτυα δεν μπορούσαν να αναλύσουν τόσες θέσεις τόσο γρήγορα. Αυτό επιβεβαιώθηκε στην πράξη, καθώς οι παραδοσιακές μηχανές όπως η Stockfish ή η Komodo ήταν πολύ ισχυρότερες από τις διάφορες μηχανές που βασίζονταν σε νευρωνικά δίκτυ, όπως η Giraffe.

Ωστόσο, το 2015 η DeepMind ανακοίνωσε μια νέα μηχανή με το όνομα AlphaGo, η οποία αντιπροσώπευε ένα μνημειώδες άλμα προς τα εμπρός για τις μηχανές τεχνητής νοημοσύνης. Η AlphaGo ήταν η πρώτη μηχανή που κέρδισε ποτέ τον Παγκόσμιο Πρωταθλητή του Go και απέδειξε την επιτυχία των μηχανών που βασίζονται σε νευρωνικά δίκτυα.

Το 2017, η DeepMind ανακοίνωσε μια νέα μηχανή που ονομάζεται AlphaZero, η οποία βελτιώθηκε σημαντικά σε σχέση με την AlphaGo. Ενώ η AlphaGo μπορούσε μόνο να παίξει go, και εκπαιδεύτηκε χρησιμοποιώντας ανθρώπινα παιχνίδια, η AlphaZero εκπαιδεύτηκε από το μηδέν και μπορούσε να παίξει σκάκι, shogi και go, όλα σε επίπεδο αιχμής.

Η μηχανή AlphaZero και οι υπόλοιπες μηχανές αυτού του είδους λειτουργούν ελαφρώς διαφορετικά από τις παραδοσιακές σκακιστικές μηχανές. Ολόκληρη η διαδικασία αξιολόγησης καθοδηγείται από ένα βαθύ νευρωνικό δίκτυο. Συγκεκριμένα, στην AlphaZero, το αρχικό δίκτυο αρχικοποιείται με εντελώς τυχαίες παραμέτρους, που σημαίνει ότι δεν υπάρχει καμία γνώση του παιχνιδιού εκτός από τους βασικούς κανόνες.

Έπειτα, το δίκτυο αυτό παίζει εναντίον του εαυτού του ή τροφοδοτείται με καταγεγραμμένα παιχνίδια από βάσεις δεδομένων. Κάθε φορά που μια νέα θέση αναλύεται από τη μηχανή, τα αποτελέσματα τροφοδοτούνται στο νευρωνικό δίκτυο. Εναλλακτικά, γίνεται να δημιουργούνται διάφορα δίκτυα με τυχαίες αρχικές τιμές, να παίζουν μεταξύ τους, και κάθε φορά να διατηρείται το καλύτερο, ώστε μακροχρόνια να έχει μείνει ένα πολύ ισχυρό δίκτυο. Το δίκτυο επιτρέπει να υπολογίσουμε μια "τιμή κεφαλής" ή και "κεφαλίδα πολιτικής" για οποιαδήποτε δεδομένη κατάσταση. Η τιμή κεφαλής αξιολογεί την θέση και τη τιμή πολιτικής δίνει μια πιθανότητα μια δεδομένη κίνηση να επιλεχτεί. Μέσω της οπισθοδιάδοσης, το δίκτυο ενημερώνει τα βάρη του και βελτιώνει τις επιδόσεις αξιολόγησης.

Κλειδί στην λειτουργία είναι και ο αλγόριθμος Monte Carlo Tree Search. Ο MCTS είναι μια διαδικασία τεσσάρων βημάτων για τη δημιουργία ενός δέντρου. Αρχικά, η μηχανή θα επιλέξει έναν νέο κόμβο στο δέντρο. Αυτή η διαδικασία επιλογής σταθμίζει τόσο την εξερεύνηση όσο και την εκμετάλλευση χρησιμοποιώντας τον αλγόριθμο PUCT, πράγμα που σημαίνει ότι οι πιο υποσχόμενοι κόμβοι (οι κόμβοι που φαίνονται καλύτεροι μέχρι τώρα) τείνουν να διερευνώνται συχνότερα, αλλά οι κόμβοι που δεν έχουν εξερευνηθεί μέχρι τώρα έχουν επίσης την ευκαιρία για επέκταση.

Μόλις επιλεχθεί ένας νέος κόμβος, τότε αξιολογείται επεκτείνοντας τους θυγατρικούς κόμβους. Τέλος, τα αποτελέσματα από την προσομοίωση του παιχνιδιού επεκτείνονται σε ολόκληρο το δέντρο. Στη συνέχεια, η διαδικασία επαναλαμβάνεται ξανά, χρησιμοποιώντας τις ενημερωμένες τιμές στο δέντρο.

1.4 Αντικείμενο εργασίας

Για την παρούσα διπλωματική εργασία, τέθηκε ως στόχος η δημιουργία μιας σκακιστικής μηχανής της σύγχρονης μορφής, χρησιμοποιώντας νευρωνικά δίκτυα για την αξιολόγηση των θέσεων και τον καθορισμό των πιθανοτήτων κίνησης. Η επόμενη κίνηση αποφασίζεται μέσω της δημιουργίας του Monte-Carlo δέντρου αναζήτησης. Για την σκακιστική μηχανή δημιουργήθηκαν τρία διαφορετικά μοντέλα αξιολόγησης θέσεων και, αντίστοιχα, τρία διαφορετικά μοντέλα πιθανότητας επόμενης κίνησης.

Το πρώτο μοντέλο δημιουργήθηκε μέσω ενισχυτικής μάθησης, με την μηχανή να παίζει εναντίον του εαυτού της και να ανανεώνονται τα βάρη του νευρωνικού δικτύου, ανάλογα με τα αποτελέσματα. Το δεύτερο μοντέλο δημιουργήθηκε με εκπαίδευση μέσω καταγεγραμμένων παιχνιδιών από ανοικτές βάσεις δεδομένων στο διαδίκτυο. Τέλος, το τρίτο μοντέλο είναι ένας συνδυασμός των δύο καθώς εκπαιδεύτηκε και με τους δύο παραπάνω τρόπους ταυτόχρονα.

Η δημιουργία έγινε στην γλώσσα προγραμματισμού Python και τα νευρωνικά δίκτυα κατασκευάστηκαν με τη βοήθεια του Keras API της Python. Ο λόγος επιλογής της Python ήταν η ευκολία που προσφέρει σε προβλήματα τεχνητής νοημοσύνης και νευρωνικών δικτύων, μέσω των βιβλιοθηκών της.

Στόχος της εργασίας είναι η ανάδειξη της δύναμης της τεχνητής νοημοσύνης, δημιουργώντας μια σκακιστική μηχανή, χωρίς καμία γνώση αξίας πεσσών και στρατηγικής, παρά μόνο την γνώση των επιτρεπτών κινήσεων. Χωρίς καμία ανθρώπινη παρέμβαση στην αξιολόγηση και στην επιλογή κινήσεων η σκακιστική μηχανή παίζει σε υψηλό επίπεδο.

Επίσης, τέθηκε ως στόχος η δυνατότητα χρήσης της μηχανής από οποιοδήποτε χρήστη, χωρίς την ανάγκη κάποιας εγκατάστασης. Για αυτό τον σκοπό, δημιουργήθηκε μια διαδικτυακή εφαρμογή σκακιού που στεγάζει την σκακιστική μηχανή. Η εφαρμογή δημιουργήθηκε με HTML, CSS και Javascript, αλλά και το Bootstrap API και επιτρέπει στον χρήστη να παίξει εναντίον των εκδόσεων της μηχανής, αλλά και να δει την μηχανή να παίζει με τον εαυτό της. Η εφαρμογή επικοινωνεί με τη σκακιστική μηχανή στο back end μέσω της Python και συγκεκριμένα μέσω του Flask API.

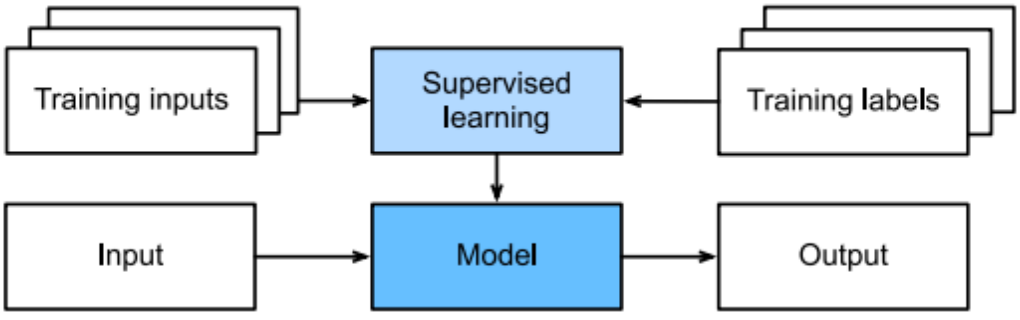
2 Μηχανική Μάθηση

Η μηχανική μάθηση (machine learning) είναι μια κατηγορία αλγορίθμων που επιτρέπουν στις εφαρμογές λογισμικού να γίνουν πιο ακριβείς στην πρόβλεψη των αποτελεσμάτων χωρίς να προγραμματίζονται ρητά. Αποτελεί κλάδο του τομέα της τεχνητής νοημοσύνης. Η βασική προϋπόθεση της μηχανικής μάθησης είναι η δημιουργία αλγορίθμων που μπορούν να λαμβάνουν δεδομένα εισόδου και να χρησιμοποιούν στατιστική ανάλυση για να προβλέψουν μια έξοδο, ενώ ενημερώνουν τις εξόδους καθώς γίνονται διαθέσιμα νέα δεδομένα.

2.1 Είδη μηχανικής μάθησης

Στη μηχανική μάθηση, τα προβλήματα ταξινομούνται γενικά σε ευρείες κατηγορίες. Αυτές οι κατηγορίες βασίζονται στον τρόπο αντίληψης της μάθησης ή στον τρόπο με τον οποίο παρέχεται ανατροφοδότηση σχετικά με τη μάθηση στο σύστημα που αναπτύχθηκε. Η μηχανική μάθηση μπορεί να ταξινομηθεί σε 3 τύπους αλγορίθμων.

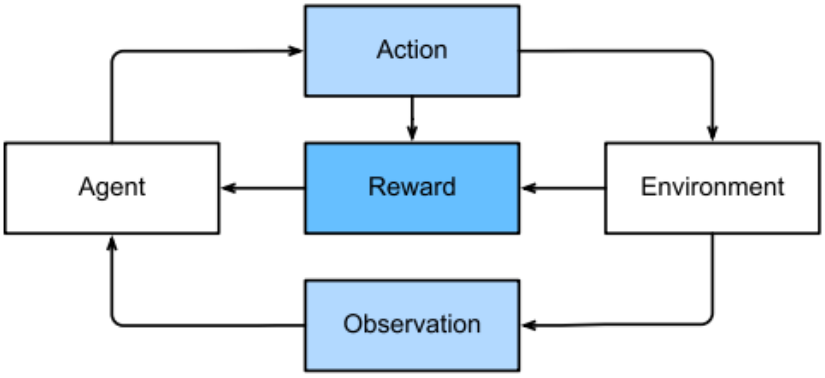
Ο πρώτος είναι η εποπτευόμενη μάθηση. Στην εποπτευόμενη εκμάθηση, ένα σύστημα τεχνητής νοημοσύνης παρουσιάζεται με δεδομένα που φέρουν ετικέτα, πράγμα που σημαίνει ότι κάθε δεδομένο επισημαίνεται με τη σωστή ετικέτα. Ο στόχος είναι η προσέγγιση της λειτουργίας χαρτογράφησης τόσο καλά ώστε όταν υπάρχουν νέα δεδομένα εισόδου (X) να μπορεί να γίνει πρόβλεψη των μεταβλητών εξόδου (Y) για αυτά τα δεδομένα [6]. Σε πιθανοτικούς όρους, συνήθως πρέπει να εκτιμηθεί η συμβατική πιθανότητα μιας ετικέτας με δεδομένη είσοδο. Η εποπτευόμενη μάθηση μπορεί να διακριθεί κυρίως σε δύο είδη, στην κατηγοριοποίηση και στην παλινδρόμηση. Ένα πρόβλημα κατηγοριοποίησης είναι όταν η μεταβλητή εξόδου είναι μια κατηγορία, όπως «κόκκινο» ή «μπλε» ή «ασθένεια» και «καμία ασθένεια». Ένα πρόβλημα παλινδρόμησης είναι όταν η μεταβλητή εξόδου είναι πραγματική τιμή, όπως "δολάρια" ή "βάρος". Άλλα προβλήματα εποπτευόμενης μάθησης αποτελούν η προσθήκη ετικετών, η αναζήτηση, τα συστήματα συστάσεων και η εκμάθηση ακολουθιών. [7].



Σχήμα 2.1: Λειτουργία μοντέλου εποπτευόμενης μάθησης

Ο δεύτερος τύπος είναι η μη εποπτευόμενη μάθηση. Στην μη εποπτευόμενη μάθηση, ένα σύστημα τεχνητής νοημοσύνης παρουσιάζεται με μη επισημασμένα, μη κατηγοριοποιημένα δεδομένα και οι αλγόριθμοι του συστήματος δρουν στα δεδομένα χωρίς προηγούμενη εκπαίδευση. Η έξοδος εξαρτάται από τους κωδικοποιημένους αλγόριθμους. Η υποβολή ενός συστήματος σε μη εποπτευόμενη μάθηση είναι ένας τρόπος δοκιμής της τεχνητής νοημοσύνης. Το μη εποπτευόμενο μοντέλο εξετάζει τον τύπο δεδομένων και μοντέλα της υποκείμενης δομής ή διανομής στα δεδομένα, προκειμένου να μάθει περισσότερα σχετικά με αυτά. Η μη εποπτευόμενη μάθηση μπορεί να χωριστεί κυρίως σε δύο μέρη, στην συσταδοποίηση και στην συσχέτιση. Ένα πρόβλημα συσταδοποίησης είναι η ανακάλυψη των εγγενών ομάδων στα δεδομένα, όπως η ομαδοποίηση πελατών με συμπεριφορά αγοράς. Ένα πρόβλημα συσχέτισης είναι, για παράδειγμα ένα πρόβλημα εκμάθησης κανόνων όπου ανακαλύπτονται κανόνες που περιγράφουν μεγάλα τμήματα των δεδομένων, όπως άτομα που αγοράζουν Χ τείνουν επίσης να αγοράζουν Υ.

Ο τρίτος τύπος αλγορίθμων μηχανικής μάθησης είναι η ενισχυμένη μάθηση. Ένας αλγόριθμος ενισχυμένης μάθησης, ή πράκτορας, μαθαίνει αλληλοεπιδρώντας με το περιβάλλον του. Ο πράκτορας λαμβάνει ανταμοιβές εκτελώντας σωστές δράσεις και κυρώσεις για εσφαλμένη απόδοση. Ο πράκτορας μαθαίνει χωρίς παρέμβαση από έναν άνθρωπο μεγιστοποιώντας την ανταμοιβή του και ελαχιστοποιώντας την ποινή του. Είναι ένας τύπος δυναμικού προγραμματισμού που εκπαιδεύει αλγόριθμους χρησιμοποιώντας ένα σύστημα ανταμοιβής και τιμωρίας. Αναλυτικότερα, η ενισχυμένη μάθηση δίνει μια πολύ γενική δήλωση ενός προβλήματος, στο οποίο ένας πράκτορας αλληλοεπιδρά με ένα περιβάλλον για μια σειρά χρονικών βημάτων. Σε κάθε βήμα, ο πράκτορας λαμβάνει κάποια παρατήρηση από το περιβάλλον και πρέπει να επιλέξει μια ενέργεια που μεταδίδεται στη συνέχεια πίσω στο περιβάλλον μέσω κάποιου μηχανισμού (μερικές φορές ονομάζεται ενεργοποιητής). Τέλος, ο πράκτορας λαμβάνει μια ανταμοιβή από το περιβάλλον. Ο πράκτορας τότε λαμβάνει μια επόμενη παρατήρηση, και επιλέγει μια επόμενη ενέργεια, και ούτω καθεξής. Η συμπεριφορά του πράκτορα ενισχυτικής μάθησης διέπεται από μια πολιτική. Εν ολίγοις, μια πολιτική είναι απλώς μια λειτουργία που καθορίζει, από τις παρατηρήσεις του περιβάλλοντος, τις επόμενες δράσεις. Ο στόχος της ενισχυτικής μάθησης είναι η παραγωγή μιας καλής πολιτικής.



Σχήμα 2.2: Λειτουργία μοντέλου ενισχυτικής μάθησης

Το πρόβλημα της ενισχυτικής μάθησης είναι ένα πολύ γενικό περιβάλλον. Οι ενέργειες επηρεάζουν τις επόμενες παρατηρήσεις. Οι ανταμοιβές παρατηρούνται μόνο σαν απάντηση στις επιλεγμένες ενέργειες. Το περιβάλλον μπορεί να είναι πλήρως παρατηρίσιμο ή εν μέρει. Παίρνοντας υπόψιν όλη αυτή την πολυπλοκότητα ταυτόχρονα μπορεί να είναι πολύ μεγάλη δουλειά για τους ερευνητές. Επιπλέον, δεν παρουσιάζει κάθε πρακτικό πρόβλημα όλη αυτή την πολυπλοκότητα. Σαν αποτέλεσμα, οι ερευνητές έχουν μελετήσει μια σειρά ειδικών περιπτώσεων προβλημάτων ενισχυτικής μάθησης. Όταν το περιβάλλον είναι πλήρως παρατηρίσιμο, το πρόβλημα ενισχυτικής μάθησης χαρακτηρίζεται ως διαδικασία απόφασης Markov. Όταν η κατάσταση δεν εξαρτάται από τις προηγούμενες ενέργειες, το πρόβλημα ονομάζεται “contextual bandit problem”. Όταν δεν υπάρχει κατάσταση, απλά ένα σύνολο διαθέσιμων ενεργειών με αρχικά άγνωστες ανταμοιβές, αυτό το πρόβλημα είναι το “classic multi-armed bandit problem”.

Στην παρούσα εργασία χρησιμοποιούνται δύο από τα είδη της μηχανικής μάθησης, η εποπτευόμενη μάθηση και η ενισχυμένη μάθηση. Η εποπτευόμενη μάθηση χρησιμοποιείται στην δημιουργία του μοντέλου που εκπαιδεύτηκε με παιχνίδια του παρελθόντος, επαγγελματικά και μη από βάσεις δεδομένων στο διαδίκτυο. Η είσοδος του μοντέλου είναι η σκακιέρα και το ποιος παίκτης παίζει και ως ετικέτα ορίζεται το τελικό αποτέλεσμα του παιχνιδιού (-1.0 για νίκη του μαύρου και 1.0 για νίκη του λευκού). Η ενισχυτική μάθηση χρησιμοποιήθηκε για την δημιουργία του μοντέλου που εκπαιδεύτηκε παίζοντας με τον εαυτό του και αξιολογώντας κινήσεις βάσει του τελικού αποτελέσματος. Επίσης, οι δύο μέθοδοι συνδυάστηκαν για τη δημιουργία ενός μοντέλου εκπαιδευμένου και με τους δύο τρόπους. Αντίστοιχα, χρησιμοποιήθηκαν και οι δύο μέθοδοι για την δημιουργία των πιθανοτικών μοντέλων πρόβλεψης κινήσεων.

2.2 Χειρισμός δεδομένων

Για να επιτευχθεί οτιδήποτε στην μηχανική μάθηση, χρειάζεται κάποιος τρόπος αποθήκευσης και χειρισμού δεδομένων. Γενικά, είναι δύο σημαντικά πράγματα που πρέπει να γίνουν με τα δεδομένα: να αποκτηθούν και να τα επεξεργαστούν κατάλληλα μόλις βρεθούν στον υπολογιστή.

Για να εφαρμοστεί η μηχανική μάθηση στην επίλυση προβλημάτων στον πραγματικό κόσμο, συχνά είναι απαραίτητη η προεπεξεργασία των πρωτογενών δεδομένων, καθώς δεν είναι όλα τα δεδομένα, που μπορεί να έχουν αποκτηθεί, χρήσιμα. Επομένως, πρέπει να απομονωθεί το χρήσιμο μέρος των δεδομένων για να χρησιμοποιηθεί για την εκπαίδευση. Τα πραγματικά δεδομένα είναι συχνά μη-καθαρά. Πολλές φορές υπάρχει έλλειψη τιμών των χαρακτηριστικών, έλλειψη ορισμένων χαρακτηριστικών ενδιαφέροντος, ή περιέχουν μόνο συναθροιστικά δεδομένα. Άλλες φορές περιέχουν σφάλματα ή ακραίες τιμές ή είναι ασυνεπή, δηλαδή περιέχουν ασυνέπειες στους κωδικούς ή στα ονόματα. Μη ποιοτικά δεδομένα σημαίνει χειρότερη απόδοση στα αποτελέσματα του μοντέλου, για αυτό και οι βάσεις δεδομένων χρειάζονται συνεπή ενοποίηση ποιοτικών δεδομένων. Τα δεδομένα πρέπει να είναι όσο πιο ακριβή, πλήρη, συνεπή και προσβάσιμα γίνεται.

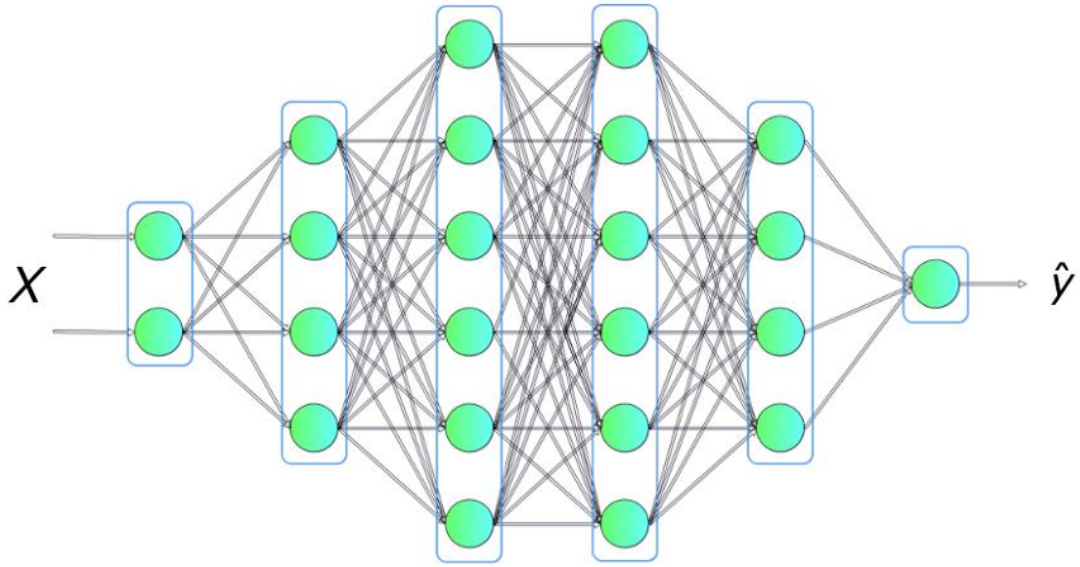
Υπάρχουν κάποιες βασικές εργασίες που μπορούν να γίνουν για την προεπεξεργασία των δεδομένων. Η μία είναι ο καθαρισμός των δεδομένων, δηλαδή η συμπλήρωση ελλιπών τιμών, η εξομάλυνση δεδομένων που έχουν θόρυβο, η αναγνώριση ή απομάκρυνση των ακραίων τιμών και η επίλυση ασυνεπειών. Μια άλλη είναι η ενοποίηση δεδομένων συνδυάζοντας πολλαπλές βάσεις δεδομένων, κύβων δεδομένων, αρχείων ή σημειώσεων. Πολύ συχνά χρησιμοποιείται ο μετασχηματισμός των δεδομένων, όπως η κανονικοποίησή τους (κλιμάκωση σε ένα συγκεκριμένο εύρος) και η συνάθροιση τους (aggregation). Επίσης, χρήσιμη μπορεί να είναι η μείωση των δεδομένων. Για παράδειγμα, η απόκτηση μειωμένων αναπαραστάσεων σε όγκο αλλά παραγωγή των ίδιων ή παρόμοιων αναλυτικών αποτελεσμάτων μπορεί να παρουσιάσει μεγάλη βελτίωση στον χρόνο που απαιτείται για την εκπαίδευση των μοντέλων. Επιπρόσθετα, μπορεί να γίνει διακριτοποίηση δεδομένων με συγκεκριμένη σημασία, ειδικά για αριθμητικά δεδομένα, αλλά και συνάθροιση δεδομένων, μείωση διαστατικότητας, συμπίεση δεδομένων και γενίκευσή τους.

Για την συγκεκριμένη εργασία και το σκάκι, το εποπτευόμενο μόνο μοντέλο απαιτούσε την εύρεση και αποθήκευση δεδομένων. Τα δεδομένα βρέθηκαν στο διαδίκτυο σε βάσεις δεδομένων που είναι ανοικτές για το κοινό και αποτελούνται από κινήσεις παλιών αγώνων σκακιού και το τελικό αποτέλεσμα του εκάστοτε παιχνιδιού. Παρόλα αυτά, τα δεδομένα απαιτούσαν προεπεξεργασία. Η προεπεξεργασία που έγινε έχει να κάνει με μείωση δεδομένων και συγκεκριμένα δεν λήφθηκαν υπόψη τα παιχνίδια με ισοπαλία ως τελικό αποτέλεσμα, καθώς οι κινήσεις που οδηγούν σε ισοπαλία δεν περιέχουν σημαντική πληροφορία για το μοντέλο, σε σχέση με τις κινήσεις που οδηγούν σε νίκη του άσπρου ή σε νίκη του μαύρου. Αντίστοιχα, και στο μοντέλο ενισχυτικής μάθησης, επειδή το μοντέλο στην εκπαίδευση αντιμετωπίζει τον εαυτό του, άρα παίζουν συνέχεια δύο ισάξιοι αντίπαλοι, τα περισσότερα παιχνίδια λήγουν ισόπαλα, οπότε δεν έχει νόημα η χρησιμοποίηση των ισόπαλων παιχνιδιών καθώς απλά θα αναιρούσαν την εκπαίδευση που γίνεται με τα υπόλοιπα παιχνίδια.

2.3 Νευρωνικά δίκτυα

Τα τεχνητά νευρωνικά δίκτυα (ANNs), συνήθως απλά ονομάζονται νευρωνικά δίκτυα (NNs), είναι υπολογιστικά συστήματα εμπνευσμένα από τα βιολογικά νευρωνικά δίκτυα που αποτελούν τον εγκέφαλο των ζώων. Ένα νευρωνικό δίκτυο βασίζεται σε μια συλλογή συνδεδεμένων μονάδων ή κόμβων που ονομάζονται τεχνητοί νευρώνες, οι οποίοι προσεγγιστικά μοντελοποιούν τους νευρώνες σε έναν βιολογικό εγκέφαλο. Κάθε σύνδεση, όπως οι συνάψεις σε έναν βιολογικό εγκέφαλο, μπορεί να μεταδώσει ένα σήμα σε άλλους νευρώνες. Ένας τεχνητός νευρώνας που λαμβάνει ένα σήμα στη συνέχεια το επεξεργάζεται και μπορεί να σηματοδοτήσει νευρώνες που συνδέονται με αυτό. Το "σήμα" σε μια σύνδεση είναι ένας πραγματικός αριθμός και η έξοδος κάθε νευρώνα υπολογίζεται από κάποια μη γραμμική συνάρτηση του αθροίσματος των εισόδων του. Οι νευρώνες και τα άκρα έχουν συνήθως ένα βάρος που προσαρμόζεται καθώς προχωρά η μάθηση. Το βάρος αυξάνει ή μειώνει την ισχύ

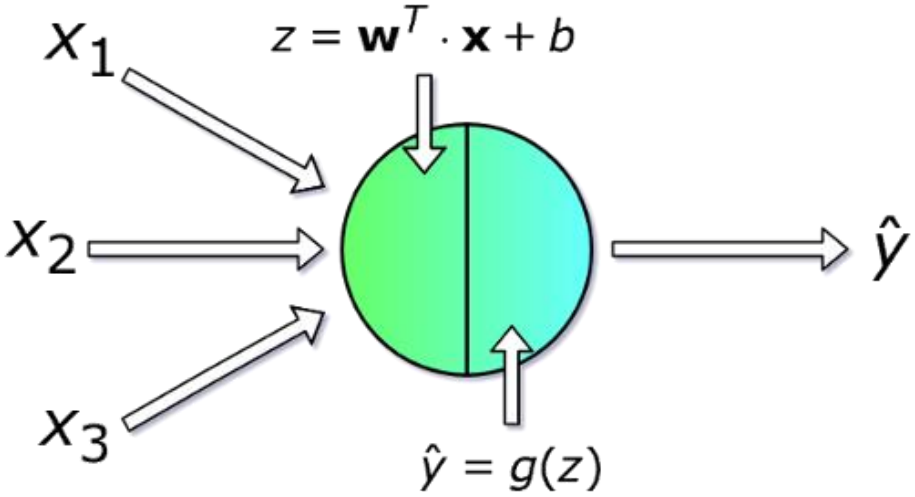
του σήματος σε μια σύνδεση. Οι νευρώνες μπορεί να έχουν ένα κατώφλι τέτοιο ώστε ένα σήμα να αποστέλλεται μόνο εάν το συνολικό σήμα ξεπεράσει αυτό το όριο. Τυπικά, οι νευρώνες συγκεντρώνονται σε επίπεδα. Διαφορετικά επίπεδα μπορούν να πραγματοποιήσουν διαφορετικούς μετασχηματισμούς στις εισόδους τους. Τα σήματα μετακινούνται από το πρώτο επίπεδο (το επίπεδο εισόδου), στο τελευταίο επίπεδο (το επίπεδο εξόδου), πιθανώς αφού περάσουν άλλα επίπεδα πολλές φορές [8].



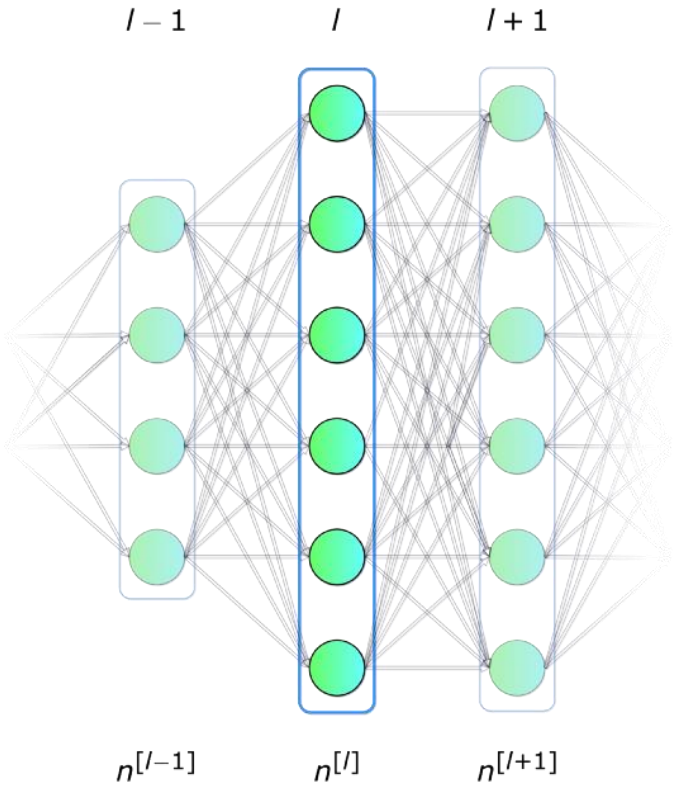
Σχήμα 2.3: Παράδειγμα δομής νευρωνικού δικτύου (Πηγή: https://towardsdatascience.com/https-medium-com-piotr-skalski92-deep-dive-into-deep-networks-math-17660bc376ba)

2.3.1 Λειτουργία νευρώνων και επιπέδων

Κάθε νευρώνας λαμβάνει ένα σύνολο τιμών x (αριθμημένες από 1 έως n) ως είσοδο και υπολογίζει την προβλεπόμενη τιμή του y. Το διάνυσμα x περιέχει, στην πραγματικότητα, τις τιμές των χαρακτηριστικών σε ένα από τα παραδείγματα m από το σετ εκπαίδευσης. Επιπλέον, κάθε μονάδα έχει το δικό της σύνολο παραμέτρων, που συνήθως αναφέρονται ως w (διάνυσμα στήλης βαρών) και b (biases - σταθερές) που αλλάζει κατά τη διάρκεια της διαδικασίας της εκπαίδευσης. Σε κάθε επανάληψη, ο νευρώνας υπολογίζει έναν σταθμισμένο μέσο όρο των τιμών του διανύσματος x, με βάση το τρέχον βάρος του φορέα w και προσθέτει το bias. Τέλος, το αποτέλεσμα αυτού του υπολογισμού περνά μέσω μιας μη γραμμικής συνάρτησης ενεργοποίησης g.



Σχήμα 2.4: Νευρώνας δικτύου (Πηγή: https://towardsdatascience.com/https-medium-com-piotr-skalski92-deep-dive-into-deep-networks-math-17660bc376ba)



Σχήμα 2.5: Επίπεδο δικτύου (Πηγή: https://towardsdatascience.com/https-medium-com-piotr-skalski92-deep-dive-into-deep-networks-math-17660bc376ba)

Οι νευρώνες ενός επιπέδου παρουσιάζουν παρόμοια συμπεριφορά. Ας θεωρηθεί ότι η συνάρτηση a συμβολίζει την συνάρτηση ενεργοποίησης για το αντίστοιχο επίπεδο δικτύου. Το διάνυσμα x είναι επομένως η ενεργοποίηση για το επίπεδο 0, το επίπεδο εισόδου. Κάθε νευρώνας στο επίπεδο εκτελεί έναν παρόμοιο υπολογισμό σύμφωνα με τις ακόλουθες εξισώσεις:

$$z_i^{[l]} = w_i^T \cdot a^{[l-1]} + b_i \, , \; a_i^{[l]} = g^{[l]}(z_i^{[l]})$$

Για κάθε ένα από τα επίπεδα πρέπει να εκτελεστεί ένας αριθμός παρόμοιων λειτουργιών. Ενώνοντας μαζί τα οριζόντια διανύσματα βαρών w (transposed) δημιουργείται η μήτρα W. Ομοίως, ενώνοντας μαζί το bias κάθε νευρώνα στο επίπεδο δημιουργείται το κάθετο διάνυσμα b. Έτσι, η παραπάνω σχέση μπορεί να εκφράσει όλους τους νευρώνες ενός επιπέδου γραμμένη με αυτόν τον τρόπο:

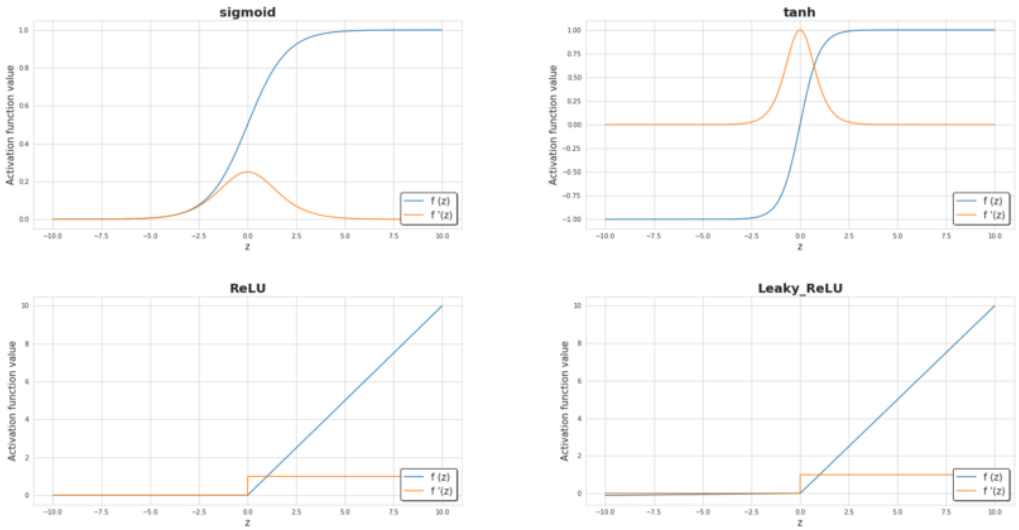
$$z^{[l]} = w^{[l]} \cdot a^{[l-1]} + b^{[l]} \, , \; a^{[l]} = g^{[l]}(z^{[l]})$$

Το επόμενο βήμα είναι η διανυσματικοποίηση σε πολλά παραδείγματα. Έστω ότι το σύνολο δεδομένων έχει καταχωρήσεις m με λειτουργίες nx. Αρχικά, συγκεντρώνονται τα κάθετα διανύσματα x, a και z κάθε επιπέδου δημιουργώντας τους πίνακες X, A και Z, αντίστοιχα. Στη συνέχεια διαμορφώνεται ξανά η προηγούμενη εξίσωση, λαμβάνοντας υπόψη τους νέους πίνακες που δημιουργήθηκαν.

$$Z^{[l]} = W^{[l]} \cdot A^{[l-1]} + b^{[l]} \, , \; A^{[l]} = g^{[l]}(Z^{[l]})$$

2.3.2 Συναρτήσεις ενεργοποίησης

Οι συναρτήσεις ενεργοποίησης είναι ένα από τα βασικά στοιχεία του νευρωνικού δικτύου. Χωρίς αυτά, το νευρωνικό δίκτυο θα γινόταν ένας συνδυασμός γραμμικών συναρτήσεων, οπότε θα ήταν απλώς μια γραμμική συνάρτηση. Το μοντέλο θα είχε περιορισμένη δυνατότητα επέκτασης, όχι μεγαλύτερη από την λογιστική παλινδρόμηση (logistic regression). Το στοιχείο της μη γραμμικότητας επιτρέπει μεγαλύτερη ευελιξία και δημιουργία σύνθετων λειτουργιών κατά τη διάρκεια της μαθησιακής διαδικασίας. Η συνάρτηση ενεργοποίησης έχει επίσης σημαντικό αντίκτυπο στην ταχύτητα εκμάθησης, η οποία είναι ένα από τα κύρια κριτήρια για την επιλογή της. Το σχήμα 2.5 δείχνει μερικές από τις συνήθεις συναρτήσεις ενεργοποίησης. Επί του παρόντος, η πιο δημοφιλής, για τα κρυφά επίπεδα, είναι πιθανώς η ReLU. Ακόμα, μερικές φορές χρησιμοποιείται η sigmoid, ειδικά στο επίπεδο εξόδου, όταν πρόκειται για δυαδική ταξινόμηση και πρέπει οι τιμές που επιστρέφονται από το μοντέλο να είναι στην περιοχή από 0 έως 1.



Σχήμα 2.5: Συνήθεις συναρτήσεις ενεργοποίησης (Πηγή: https://towardsdatascience.com/https-medium-com-piotr-skalski92-deep-dive-into-deep-networks-math-17660bc376ba)

2.3.3 Συνάρτηση σφάλματος

Η βασική πηγή πληροφοριών για την πρόοδο της μαθησιακής διαδικασίας είναι η αξία της συνάρτησης σφάλματος. Σε γενικές γραμμές, η συνάρτηση σφάλματος έχει σχεδιαστεί για να δείξει πόσο μακριά το μοντέλο βρίσκεται από την «ιδανική» λύση. Κατά την εκτέλεση εποπτευόμενης εκπαίδευσης, η πραγματική παραγωγή ενός νευρωνικού δικτύου πρέπει να συγκριθεί με την ιδανική τιμή που καθορίζεται στα δεδομένα εκπαίδευσης. Η διαφορά μεταξύ πραγματικής και ιδανικής εξόδου είναι το σφάλμα του νευρωνικού δικτύου.

Ο υπολογισμός σφάλματος πραγματοποιείται σε δύο επίπεδα. Αρχικά, υπάρχει το τοπικό σφάλμα. Αυτή είναι η διαφορά μεταξύ της πραγματικής εξόδου ενός μεμονωμένου νευρώνα και της ιδανικής εξόδου που είναι η αναμενόμενη. Το τοπικό σφάλμα υπολογίζεται χρησιμοποιώντας μια συνάρτηση σφάλματος. Τα τοπικά σφάλματα συνυπολογίζονται για να σχηματίσουν ένα ολικό σφάλμα. Το ολικό σφάλμα είναι το μέτρο της απόδοσης ενός νευρωνικού δικτύου σε ολόκληρο το σετ εκπαίδευσης. Υπάρχουν πολλά διαφορετικά μέσα με τα οποία μπορεί να υπολογιστεί ένα ολικό σφάλμα. Οι μέθοδοι που χρησιμοποιούνται συχνότερα για τον υπολογισμό του είναι:

- Σφάλμα αθροίσματος τετραγώνων (ESS)
- Σφάλμα μέσου τετραγώνου (MSE)
- Σφάλμα μέσης ρίζας (RMS)

Το τοπικό σφάλμα προέρχεται από τη συνάρτηση σφάλματος. Η συνάρτηση σφάλματος τροφοδοτείται με την πραγματικές και ιδανικές εξόδους για έναν μόνο νευρώνα εξόδου και στη συνέχεια παράγει έναν αριθμό που αντιπροσωπεύει το σφάλμα αυτού του νευρώνα εξόδου. Οι μέθοδοι εκπαίδευσης φροντίζουν να ελαχιστοποιηθεί αυτό το σφάλμα.

2.3.4 Εκπαίδευση

Η μαθησιακή διαδικασία αφορά την αλλαγή των τιμών των παραμέτρων W και b έτσι ώστε η συνάρτηση σφάλματος να ελαχιστοποιείται. Προκειμένου να επιτευχθεί αυτός ο στόχος, χρησιμοποιείται ο αλγόριθμος απότομης καθόδου (gradient descent) για να βρεθεί μια ελάχιστη τιμή της συνάρτησης. Σε κάθε επανάληψη υπολογίζονται οι τιμές των μερικών παραγώγων της συνάρτησης σφάλματος σε σχέση με καθεμία από τις παραμέτρους του νευρικού δικτύου.

Η οπισθοδιάδοση (backpropagation) είναι ένας αλγόριθμος που επιτρέπει τον υπολογισμό μιας πολύ περίπλοκης κλίσης. Οι παράμετροι του νευρωνικού δικτύου προσαρμόζονται σύμφωνα με τους ακόλουθους τύπους:

$$W^{[l]} = W^{[l]} - \alpha dW^{[l]}$$

$$b^{[l]} = b^{[l]} - \alpha db^{[l]}$$

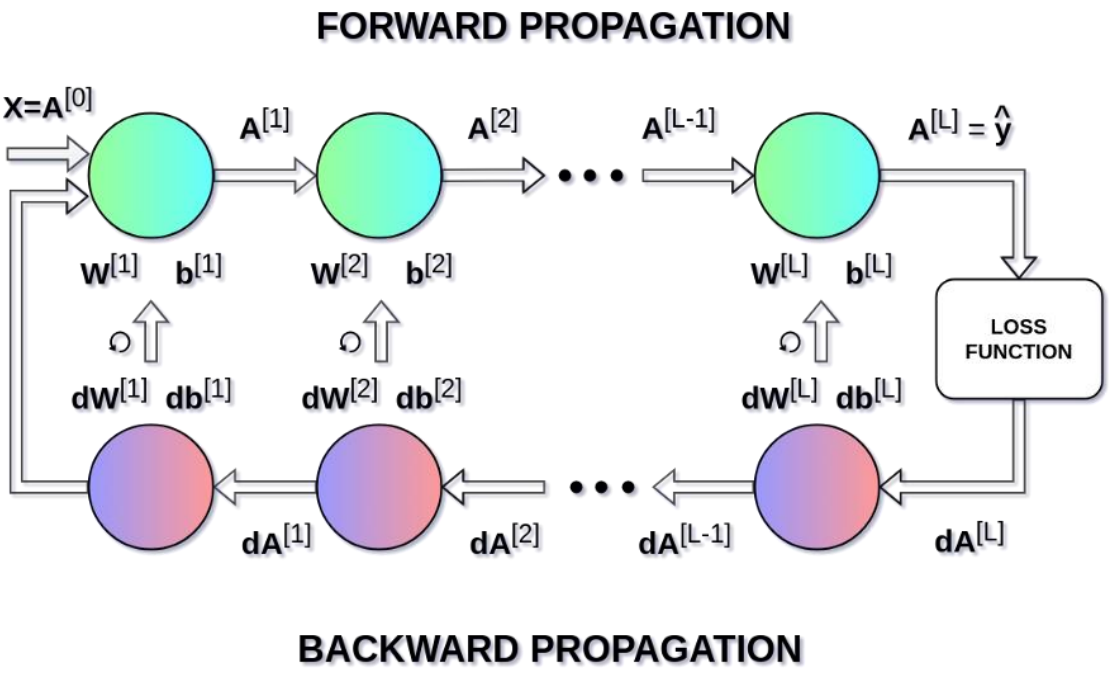
Στις παραπάνω εξισώσεις, το α αντιπροσωπεύει το ρυθμό εκμάθησης (learning rate), μία υπέρ-παράμετρος που επιτρέπει τον έλεγχο της τιμής της εκτελεσμένης προσαρμογής. Η επιλογή του ρυθμού εκμάθησης είναι ζωτικής σημασίας, αν είναι πολύ χαμηλός, το νευρωνικό δίκτυο θα μαθαίνει πολύ αργά, αν είναι πολύ υψηλός δεν θα μπορεί να προσεγγισθεί το ελάχιστο της συνάρτησης σφάλματος. Τα dW και db υπολογίζονται χρησιμοποιώντας τον κανόνα αλυσίδας στις μερικές παραγώγους της συνάρτησης σφάλματος σε σχέση με τα W και b. Το μέγεθος των dW και db είναι το ίδιο με αυτό των W και b αντίστοιχα. Το σχήμα 2.6 δείχνει την ακολουθία των λειτουργιών εντός του νευρικού δικτύου.

$$dW^{[l]} = \frac{1}{m} dZ^{[l]} A^{[l-1]T}$$

$$db^{[l]} = \frac{1}{m} \sum_{i=1}^{m} dZ^{[I]}$$

$$dA^{[I-1]} = W^{[I]T} dZ^{[I]}$$

$$dZ^{[I]} = dA^{[I]}\, g'(Z^{[I]})$$



Σχήμα 2.6: Διαδικασία υπολογισμών εξόδων και οπισθοδιάδοσης σφάλματος (Πηγή: https://towardsdatascience.com/https-medium-com-piotr-skalski92-deep-dive-into-deep-networks-math-17660bc376ba)

Τα νευρωνικά δίκτυα αποτελούν βασικό κομμάτι της εργασίας. Η αξιολόγηση των θέσεων γίνεται με βάση νευρωνικό δίκτυο που εκπαιδεύεται είτε μέσω αντιμετώπισης του εαυτού του, είτε με δεδομένα από το διαδίκτυο είτε και με τα δύο. Είσοδος του νευρωνικού δικτύου είναι η θέση της σκακιέρας σε κωδικοποιημένη μορφή και έξοδος είναι το τελικό αποτέλεσμα, έτσι ώστε οι θέσεις που οδηγούν σε ήττα ή νίκη για έναν παίκτη να αξιολογούνται αρνητικά ή θετικά, αντίστοιχα. Επίσης, αναπτύχθηκε νευρωνικό δίκτυο πρόβλεψης των πιο πιθανών κινήσεων για μια θέση, χωρίς κάποια αξιολόγηση, για να επιταχυνθεί η ανάλυση του δέντρου του παιχνιδιού κατά τη διαδικασία αναζήτησης νέας κίνησης.

2.4 Συνελικτικά νευρωνικά δίκτυα

Σε πολλά προβλήματα τεχνητής νοημοσύνης είναι αναγκαία η ανάλυση εικόνας, για την οποία κάθε παράδειγμα αποτελείται από ένα δισδιάστατο πλέγμα από pixels. Ανάλογα με το αν πρόκειται για ασπρόμαυρη ή έγχρωμη φωτογραφία, κάθε pixel μπορεί να σχετίζεται είτε με μία είτε με πολλές αριθμητικές τιμές, αντίστοιχα. Σε ένα τυπικό νευρωνικό δίκτυο, όπως αυτό που περιεγράφηκε στην προηγούμενη ενότητα, απλά απορρίπτεται η χωρική δομή κάθε εικόνας μετατρεπόμενη σε μονοδιάστατα διανύσματα, για να τροφοδοτηθούν σε ένα δίκτυο. Επειδή αυτά τα δίκτυα είναι αμετάβλητα στη σειρά των χαρακτηριστικών, πιθανώς να προκύψουν παρόμοια αποτελέσματα ανεξάρτητα από το αν διατηρείται μια τάξη που αντιστοιχεί σε χωρική δομή των pixels ή αν αντικατασταθούν οι στήλες της μήτρας σχεδιασμού πριν την προσαρμογή των παραμέτρων του δικτύου. Ιδανικά, όμως, πρέπει να αξιοποιηθεί η εκ των προτέρων γνώση ότι τα κοντινά pixels σε μια εικόνα τυπικά σχετίζονται μεταξύ τους, για τη δημιουργία αποτελεσματικών μοντέλων για μάθηση από δεδομένα εικόνας.

Τα συνελικτικά νευρωνικά δίκτυα (CNNs – Convolutional Neural Networks) είναι μια ισχυρή οικογένεια νευρωνικών δικτύων που έχουν σχεδιαστεί για αυτόν ακριβώς το σκοπό. Οι αρχιτεκτονικές που βασίζονται στα συνελικτικά νευρωνικά δίκτυα είναι πλέον πανταχού παρούσες στο πεδίο μηχανικής όρασης (computer vision), και έχουν γίνει τόσο κυρίαρχες που σχεδόν κανείς σήμερα δεν θα μπορούσε να αναπτύξει μια εμπορική εφαρμογή ή να συμμετάσχει σε διαγωνισμό που σχετίζεται με την αναγνώριση εικόνας, την ανίχνευση αντικειμένων ή σημασιολογική κατάτμηση, χωρίς να βασίζεται σε αυτήν την προσέγγιση.

Τα συνελικτικά νευρωνικά δίκτυα αποτελούν κατάλληλη επιλογή, γενικότερα, όταν τα δεδομένα έχουν μορφή πίνακα (όπως μια εικόνα που έχει γραμμές και στήλες από pixels). Ως πίνακας, σημαίνει ότι τα δεδομένα αποτελούνται από σειρές που αντιστοιχούν σε παραδείγματα και στήλες που αντιστοιχούν σε χαρακτηριστικά. Με πίνακες δεδομένων, μπορεί να αναμένεται ότι τα πρότυπα που αναζητούνται θα μπορούσαν να περιλαμβάνουν αλληλεπιδράσεις μεταξύ των χαρακτηριστικών, αλλά δεν γίνεται κάποια υπόθεση δομής τους εκ των προτέρων σχετικά με τον τρόπο αλληλεπίδρασης των χαρακτηριστικών.

Για πίνακες λίγων διαστάσεων, ένα τυπικό πλήρως συνδεδεμένο νευρωνικό δίκτυο, μπορεί να κάνει καλή δουλειά στην πρόβλεψη, αλλά για δεδομένα με πολλές διαστάσεις, τέτοια δίκτυα μπορεί να γίνουν υπερβολικά μεγάλα σε μέγεθος επιπέδων και νευρώνων, άρα και πιο απαιτητικά υπολογιστικά, καθώς και από άποψη αποθηκευτικού χώρου.

2.4.1 Λειτουργία συνελικτικού επιπέδου

Έστω ότι πρέπει να εντοπιστεί ένα αντικείμενο σε μια εικόνα. Φαίνεται λογικό ότι οποιαδήποτε μέθοδος και να χρησιμοποιείται για την αναγνώριση αντικειμένων, δεν πρέπει να παίζει μεγάλο ρόλο η ακριβής θέση του αντικειμένου στην εικόνα. Τα

συνελικτικά δίκτυα συστηματοποιούν αυτήν την ιδέα της χωρικής απόκλισης, αξιοποιώντας την για να μάθουν χρήσιμες αναπαραστάσεις με λιγότερες παραμέτρους.

Αρχικά, έστω ένα πολύ-επίπεδο νευρωνικό δίκτυο με δισδιάστατες εικόνες X ως εισόδους και τις κρυφές αναπαραστάσεις τους H που αντιπροσωπεύονται με παρόμοιο τρόπο ως πίνακες στα μαθηματικά και ως τρισδιάστατοι τανυστές στον κώδικα, όπου και το X και το H έχουν το ίδιο σχήμα. Πλέον, θεωρείται ότι έχουν χωρική δομή και οι κρυφές αναπαραστάσεις, εκτός από τις εισόδους.

Έστω ότι τα $X_{i,j}$ και $H_{i,j}$ να δηλώνουν το pixel στη θέση (i, j) στην εικόνα εισόδου και την κρυφή παράσταση, αντίστοιχα. Κατά συνέπεια, για να λάβει κάθε μια από τις κρυφές μονάδες είσοδο από κάθε pixel εισόδου, θα γινόταν η αλλαγή από τη χρήση πινάκων βάρους (όπως γινόταν προηγουμένως στα πολύ-επίπεδα νευρωνικά δίκτυα) για να αντιπροσωπευτούν οι παράμετροι ως τέταρτης τάξης τανυστές βαρών W. Ας γίνει η υπόθεση ότι το U περιέχει τα biases, θα μπορούσε να εκφραστεί το πλήρως συνδεδεμένο επίπεδο ως:

$$H_{i,j} = U_{i,j} + \sum_k \sum_l W_{i,j,k,l} X_{k,l}$$
$$= U_{i,j} + \sum_a \sum_b V_{i,j,a,b} X_{i+a,j+b}$$

, όπου η αλλαγή από W σε V είναι εντελώς αισθητική για τώρα, καθώς υπάρχει αντιστοιχία μεταξύ των συντελεστών και στους δύο τανυστές τέταρτης τάξης. Οι δείκτες a και b διατρέχουν θετικά και αρνητικά offset, καλύπτοντας ολόκληρη την εικόνα. Για οποιαδήποτε δεδομένη τοποθεσία (i, j) στην κρυφή αναπαράσταση $H_{i,j}$ υπολογίζεται η τιμή της αθροίζοντας τα pixels του x, συντετριμμένα γύρω από τα (i, j) και σταθμισμένα με τα βάρη $V_{i,j,a,b}$.

Μια αλλαγή στην είσοδο X υποθετικά πρέπει να οδηγεί σε αλλαγή της κρυφής αναπαράστασης H. Αυτό είναι μόνο πιθανό εάν τα V και U δεν εξαρτώνται πραγματικά από τα (i, j), για παράδειγμα αν $V_{i,j,a,b} = V_{a,b}$ και το U είναι μια σταθερά u. Ως αποτέλεσμα, μπορεί να απλοποιηθεί ο ορισμό για το H ως:

$$H_{i,j} = u + \sum_a \sum_b V_{a,b} X_{i+a,j+b}$$

Αυτό αποτελεί μια συνέλιξη. Το $V_{a,b}$ χρειάζεται πολύ λιγότερους συντελεστές από το $V_{i,j,a,b}$, καθώς δεν εξαρτάται πλέον από τη θέση στην εικόνα, το οποίο αποτελεί μεγάλη πρόοδο ήδη.

Μια άλλη αρχή που διέπει τις εικόνες είναι η αρχή της τοποθεσίας (locality). Για να ληφθεί σχετική πληροφορία ώστε να αποφασιστεί τι συμβαίνει στο $H_{i,j}$, χρειάζεται να εξεταστεί η περιοχή γύρω από το i,j και όχι ολόκληρη η εικόνα. Αυτό σημαίνει ότι εκτός μιας περιοχής $|a| > \Delta$ ή $|b| > \Delta$, μπορεί να τεθεί $V_{a,b} = 0$. Επομένως, το $H_{i,j}$ μπορεί να ξαναγραφτεί ως:

$$H_{i,j} = u + \sum_{a=-\Delta}^{\Delta} \sum_{b=-\Delta}^{\Delta} V_{a,b} X_{i+a,j+b}$$

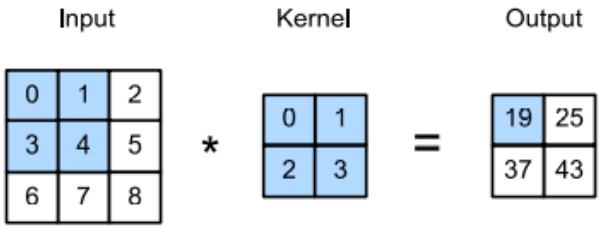
Ουσιαστικά, αυτό είναι ένα επίπεδο συνέλιξης. Τα συνελικτικά νευρωνικά δίκτυα (CNNs) είναι μια ειδική οικογένεια νευρωνικών δικτύων που περιέχουν συνελικτικά επίπεδα. Το V αναφέρεται ως πυρήνας συνέλιξης (convolution kernel), φίλτρο ή απλώς ως τα βάρη του επιπέδου που είναι συχνά παράμετροι. Όταν η τοπική περιοχή είναι μικρή, η διαφορά συγκρινόμενη με ένα πλήρως συνδεδεμένο δίκτυο μπορεί να είναι δραματική. Ενώ, προηγουμένως, ίσως χρειαζόταν δισεκατομμύρια παράμετροι για μόνο ένα επίπεδο σε ένα δίκτυο επεξεργασίας εικόνας, συνήθως τώρα χρειάζονται μόνο μερικές εκατοντάδες, χωρίς να αλλάξει η διαστατικότητα είτε των εισόδων είτε των κρυφών αναπαραστάσεων. Το τίμημα που καταβάλλεται για αυτή τη δραστική μείωση των παραμέτρων είναι ότι τώρα το επίπεδο μπορεί να ενσωματώνει μόνο τοπικές πληροφορίες, όταν ορίζει την τιμή κάθε κρυφής ενεργοποίησης. Όλη η μάθηση εξαρτάται από την επιβολή επαγωγικών biases. Όταν αυτά τα biases συμφωνούν με την πραγματικότητα, προκύπτουν μοντέλα αποδοτικά προς το δείγμα που γενικεύονται σε αόρατα δεδομένα. Φυσικά, εάν αυτά τα biases δεν συμφωνούν με την πραγματικότητα, τα μοντέλα μπορεί να δυσκολευτούν ακόμη και να προσαρμόσουν στα δεδομένα εκπαίδευσης.

Παρόλα αυτά, πολλά δεδομένα σε μορφή πίνακα αποτελούνται από διάφορα κανάλια, για παράδειγμα οι εικόνες αποτελούνται από τρία κανάλια, το κόκκινο, το πράσινο και το μπλε (rgb). Οι εικόνες δεν είναι δισδιάστατα αντικείμενα αλλά μάλλον τανυστές τρίτης τάξης, που χαρακτηρίζονται από ύψος, πλάτος και κανάλι, παραδείγματος χάριν με σχήμα 1024 × 1024 × 3 pixels. Ενώ οι δύο πρώτοι από αυτούς τους άξονες αφορούν χωρικές σχέσεις, ο τρίτος μπορεί να θεωρηθεί ως εκχώρηση πολυδιάστατης αναπαράστασης σε κάθε θέση pixel. Έτσι, το X μπορεί να εκφραστεί ως $X_{i, j, k}$. Το φίλτρο συνέλιξης πρέπει να προσαρμοστεί ανάλογα. Αντί για $V_{a, b}$, τώρα ορίζεται ως $V_{a, b, c}$.

2.4.2 Διασταυρούμενη συσχέτιση (Cross-correlation)

Σε ένα επίπεδο συνελικτικού δικτύου ένας τανυστής εισόδου και ένας τανυστής πυρήνα συνδυάζονται για να παράγουν έναν τανυστή εξόδου μέσω μιας λειτουργίας διασταυρούμενης συσχέτισης (cross-correlation). Για παράδειγμα, έστω ότι υπάρχουν δισδιάστατα δεδομένα και δισδιάστατες κρυφές αναπαραστάσεις. Στο σχήμα 2.7, η είσοδος είναι ένας δισδιάστατος τανυστής με ύψος 3 και πλάτος 3. Το σχήμα του τανυστή συμβολίζεται ως 3×3 ή (3, 3). Το ύψος και το πλάτος του πυρήνα είναι και τα δύο 2. Το σχήμα του παραθύρου του πυρήνα (ή του παραθύρου περιστροφής) δίνεται από το ύψος και το πλάτος του πυρήνα (εδώ είναι 2×2). Τα σκιασμένα τμήματα είναι

τα στοιχεία της εισόδου και του kernel που χρησιμοποιηθήκαν για τον υπολογισμό της αντίστοιχης σκιασμένης εξόδου με την πράξη: 0 × 0 +1 × 1 + 3 × 2 + 4 × 3 = 19.



Σχήμα 2.7: Δισδιάστατη διασταυρούμενη συσχέτιση

Οι υπόλοιπες έξοδοι προκύπτουν με τους ανάλογους υπολογισμούς. Κατά μήκος κάθε άξονα, το μέγεθος εξόδου είναι ελαφρώς μικρότερο από το μέγεθος εισόδου. Αφού ο πυρήνας έχει πλάτος και ύψος μεγαλύτερο από ένα, μπορεί μόνο να υπολογιστεί σωστά η διασταυρούμενη συσχέτιση για τοποθεσίες όπου ο πυρήνας ταιριάζει πλήρως μέσα στην εικόνα. Το μέγεθος εξόδου δίνεται από το μέγεθος εισόδου $n_h \times n_w$ μείον το μέγεθος του πυρήνα συνέλιξης $k_h \times k_w$ μέσω της σχέσης:

$$(n_h - k_h + 1) \times (n_w - k_w + 1)$$

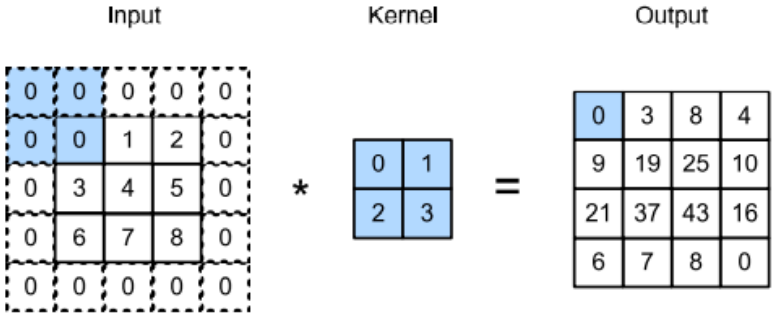
Ένα επίπεδο συνέλιξης συσχετίζει την είσοδο και τον πυρήνα και προσθέτει μια σταθερά (bias) για να παράγει μια έξοδο. Οι δύο παράμετροι ενός συνελικτικού επιπέδου είναι ο πυρήνας και το bias. Όταν εκπαιδεύονται μοντέλα βασισμένα σε συνελικτικά επίπεδα, συνήθως αρχικοποιούνται οι πυρήνες τυχαία, όπως θα γινόταν με ένα πλήρως συνδεδεμένο επίπεδο.

Σε αρκετές περιπτώσεις, είναι απαραίτητη η ενσωμάτωση τεχνικών, συμπεριλαμβανομένων του γεμίσματος (padding) και του διασκελισμού (striding), που επηρεάζουν το μέγεθος της εξόδου. Ο λόγος είναι ότι αφού οι πυρήνες έχουν γενικά πλάτος και ύψος μεγαλύτερο από 1, μετά την εφαρμογή πολλών διαδοχικών συνελίξεων, οι έξοδοι τείνουν να είναι σημαντικά μικρότερες από τις εισόδους. Αν η αρχική εικόνα είναι μεγέθους 240 × 240 pixels, 10 επίπεδα των 5 × 5 συνελίξεων μειώνουν την εικόνα στα 200 × 200 pixels, κόβοντας το 30% της εικόνας και διαγράφοντας κάθε ενδιαφέρουσα πληροφορία σχετικά με τα όρια της. Το padding είναι το πιο δημοφιλές εργαλείο για τον χειρισμό αυτού του ζητήματος.

Σε άλλες περιπτώσεις, μπορεί να χρειάζεται να μειωθεί δραστικά η διάσταση, για παράδειγμα αν κρίνεται ότι η ανάλυση της εικόνας είναι δύσκολο να διαχειριστεί. Οι συνελίξεις με striding είναι μια διαδεδομένη τεχνική που βοηθάει σε τέτοιες περιπτώσεις.

2.4.3 Γέμισμα (Padding)

Όσον αφορά το padding, όταν χρησιμοποιούνται συνελικτικά επίπεδα, υπάρχει η τάση να χάνονται pixels στην περίμετρο της εικόνας. Δεδομένου ότι συνήθως χρησιμοποιούνται μικροί πυρήνες, για κάθε συνέλιξη, μπορεί να χαθούν μόνο μερικά pixels, αλλά αυτό μπορεί να γίνει μεγάλος αριθμός ενόσω εφαρμόζονται πολλά διαδοχικά συνελικτικά επίπεδα. Μια απλή λύση σε αυτό το πρόβλημα είναι η πρόσθεση επιπλέον pixels, συμπληρωματικού χαρακτήρα, στο όριο της εικόνας εισόδου, αυξάνοντας έτσι το πραγματικό μέγεθος της εικόνας. Τυπικά, ορίζονται οι τιμές των επιπλέον pixels στο μηδέν. Στο σχήμα 2.8, υπάρχει μια είσοδος 3 × 3, της οποίας το μέγεθός αυξάνεται σε 5 × 5 χρησιμοποιώντας padding. Η αντίστοιχη έξοδος αυξάνεται στη συνέχεια σε μια μήτρα 4 × 4.



Σχήμα 2.8: Δισδιάστατη διασταυρούμενη συσχέτιση με padding

Γενικεύοντας, αν προστεθεί ένα σύνολο p_h σειρών γεμίσματος (περίπου μισές από πάνω και μισές από κάτω) και ένα σύνολο από p_w στηλών γεμίσματος (περίπου οι μισές στα αριστερά και οι μισές στα δεξιά), το μέγεθος της εξόδου θα είναι:

$$(n_h - k_h + p_h + 1) \times (n_w - k_w + p_w + 1)$$

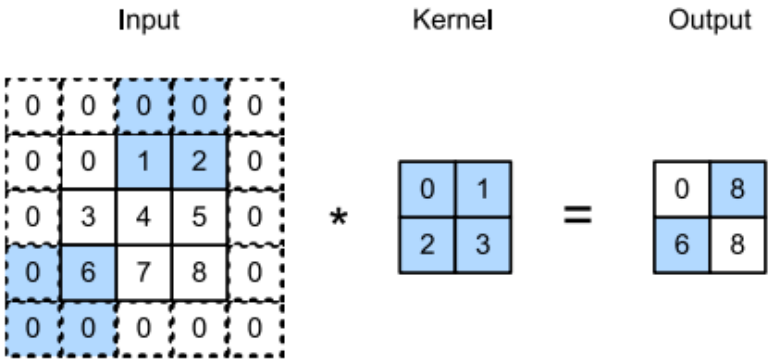
Αυτό σημαίνει ότι το ύψος και το πλάτος της εξόδου θα αυξηθούν κατά p_h και p_w, αντίστοιχα. Σε πολλές περιπτώσεις, είναι επιθυμητό να οριστεί ph = kh - 1 και pw = kw - 1 για να δοθεί στην είσοδο και στην έξοδο ίδιο ύψος και πλάτος. Αυτό διευκολύνει την πρόβλεψη του σχήματος εξόδου κάθε επιπέδου όταν κατασκευάζεται το δίκτυο.

Τα συνελικτικά νευρωνικά δίκτυα χρησιμοποιούν συνήθως πυρήνες συνέλιξης με περιττές τιμές ύψους και πλάτους, όπως 1, 3, 5 ή 7. Η επιλογή περιττών μεγεθών πυρήνα έχει το πλεονέκτημα ότι μπορεί να διατηρηθεί η χωρική διαστατικότητα, ενώ ταυτόχρονα να γίνεται padding με τον ίδιο αριθμό γραμμών στο επάνω και κάτω μέρος και τον ίδιο αριθμό στηλών αριστερά και δεξιά.

2.4.4 Βηματισμός (Stridding)

Ο υπολογισμός του cross-correlation ξεκινάει με το παράθυρο συνέλιξης στην επάνω αριστερά πλευρά του τανυστή εισόδου και προχωράει σε όλες τις θέσεις τόσο προς τα κάτω όσο και προς τα δεξιά. Μερικές φορές, είτε για υπολογιστική αποδοτικότητα είτε επειδή πρέπει να γίνει δειγματοληψία, μετακινείται το παράθυρό περισσότερα από ένα στοιχείο κάθε φορά, παρακάμπτοντας ενδιάμεσες τοποθεσίες.

Ο αριθμός των γραμμών και στηλών που διανύονται ανά ολίσθηση του παραθύρου συνέλιξης αναφέρεται ως βήμα (stride). Μερικές φορές, χρειάζεται να χρησιμοποιείται βήμα μεγαλύτερο του 1. Το σχήμα 2.9 δείχνει μια δισδιάστατη διασταυρούμενη συσχέτιση με ένα βήμα 3 κάθετα και 2 ορίζοντα. Τα σκιασμένα τμήματα είναι τα στοιχεία εξόδου καθώς και τα στοιχεία εισόδου και πυρήνα που χρησιμοποιούνται για τον υπολογισμό εξόδου: 0 × 0 + 0 × 1 + 1 × 2 + 2 × 3 = 8, 0 × 0 + 6 × 1 + 0 × 2 + 0 × 3 = 6. Είναι προφανές ότι όταν εξάγεται το δεύτερο στοιχείο της πρώτης στήλης, το παράθυρο συνέλιξης μετακινείται προς τα κάτω τρεις σειρές. Το παράθυρο συνέλιξης ολισθαίνει δύο στήλες προς τα δεξιά όταν το εξάγεται το δεύτερο στοιχείο της πρώτης σειράς. Όταν το παράθυρο συνέλιξης συνεχίζει να ολισθαίνει δύο στήλες δεξιά στην είσοδο, δεν υπάρχει έξοδος επειδή το στοιχείο εισόδου δεν μπορεί να γεμίσει το παράθυρο (εκτός αν προστεθεί άλλη στήλη γεμίσματος μέσω padding).



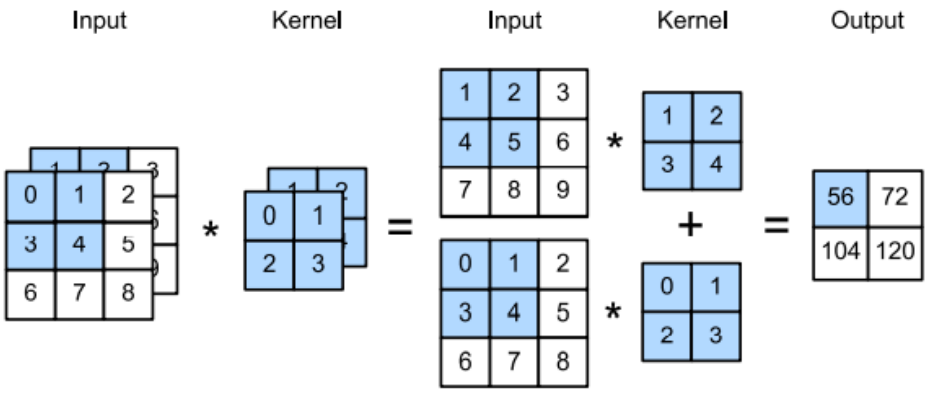
Σχήμα 2.9: Διασταυρούμενη συσχέτιση με βήματα 3 για ύψος και 2 για πλάτος, αντίστοιχα

Γενικεύοντας, όταν το βήμα για το ύψος είναι s_h και το βήμα για το πλάτος είναι s_w, το μέγεθος της εξόδου είναι:

$$[(n_h - k_h + p_h + s_h) / s_h] \times [(n_w - k_w + p_w + s_w) / s_w]$$

2.4.5 Πολλαπλά κανάλια εισόδου

Φυσικά, όλα τα παραπάνω παραδείγματα cross-correlation ισχύουν για ένα κανάλι εισόδου κι ένα κανάλι εξόδου. Όταν υπάρχουν περισσότερα κανάλια, οι είσοδοι και οι κρυφές αναπαραστάσεις γίνονται πολυδιάστατοι τανυστές. Όταν τα δεδομένα εισόδου περιέχουν πολλαπλά κανάλια, πρέπει να κατασκευαστεί ένας πυρήνας συνέλιξης με τον ίδιο αριθμό καναλιών εισόδου, όσα και τα κανάλια των δεδομένων, ώστε να γίνει σωστά η διασταυρούμενη συσχέτιση. Έστω c_i το πλήθος των καναλιών των δεδομένων εισόδου. Αν το c_i είναι μεγαλύτερο του 1, χρειάζεται ένας πυρήνας που περιέχει έναν τανυστή μεγέθους $k_h \times k_w$ για κάθε κανάλι, δηλαδή ένας πυρήνας μεγέθους $k_h \times k_w \times c_i$. Για κάθε κανάλι, το αποτέλεσμα της συνέλιξης αθροίζεται στο τέλος για να προκύψει ο τελικός τανυστής, όπως στο σχήμα 2.10 όπου παρουσιάζεται παράδειγμα με δυο κανάλια εισόδου.



Σχήμα 2.10: Διασταυρούμενη συσχέτιση με δύο κανάλια εισόδου

2.4.6 Συγκέντρωση (Pooling)

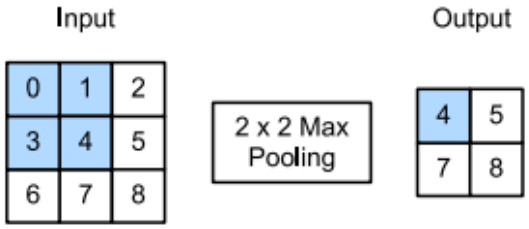
Συχνά, κατά την επεξεργασία εικόνων, πρέπει να μειώνεται σταδιακά η χωρική ανάλυση των κρυφών αναπαραστάσεων, καθώς συγκεντρώνονται πληροφορίες έτσι ώστε όσο πιο ψηλά προχωράνε οι υπολογισμοί στο δίκτυο, τόσο μεγαλύτερο το πεδίο αντίληψης (στην είσοδο) από το οποίο εξαρτάται κάθε κρυμμένος κόμβος. Τυπικά οι νευρώνες του τελικού επιπέδου θα πρέπει να εξαρτώνται από ολόκληρη την είσοδο. Τοποθετώντας πλήρως συνδεδεμένα επίπεδα μετά τα συνελικτικά και τροφοδοτώντας το δίκτυο με όλο και περισσότερη πληροφορία, επιτυγχάνεται η εκμάθηση μιας καθολικής αναπαράστασης, ενώ ταυτόχρονα διατηρούνται τα πλεονεκτήματα των συνελικτικών επιπέδων στο ενδιάμεσο της επεξεργασίας.

Όταν εντοπίζονται χαμηλότερου επιπέδου χαρακτηριστικά, όπως γωνίες, χρειάζεται να λαμβάνονται υπόψη, χωρίς να δίνεται μεγάλη σημασία στο σημείο που

εντοπίστηκαν, καθώς αποτελούν σημαντικό χαρακτηριστικό αναγνώρισης. Άλλωστε, ακόμα και όταν εξετάζονται εικόνες ίδιων αντικειμένων, είναι σχεδόν απίθανο τα διακριτά τους χαρακτηριστικά να εμφανίζονται στο ίδιο ακριβώς σημείο. Για αυτόν τον λόγο χρησιμοποιούνται τα επίπεδα pooling (συγκέντρωσης). Τα επίπεδα pooling μετριάζουν την ευαισθησία των συνελικτικών επιπέδων στη θέση και υποδειγματοληπτούν χωρικά τις αναπαραστάσεις.

Όπως στα συνελικτικά επίπεδα, η λειτουργία pooling αποτελείται από ένα παράθυρο σταθερού σχήματος που ολισθαίνει σε όλες περιοχές της εισόδου σύμφωνα με το βήμα της, υπολογίζοντας μία μόνο έξοδο για κάθε τοποθεσία που διασχίζεται από το παράθυρο σταθερού σχήματος (μερικές φορές γνωστό ως παράθυρο pooling). Ωστόσο, σε αντίθεση με τον υπολογισμό διασταυρούμενου συσχετισμού των εισόδων και των πυρήνων στο συνελικτικό επίπεδο, το επίπεδο pooling δεν περιέχει παραμέτρους (δεν υπάρχει πυρήνας). Αντ' αυτού, η λειτουργία pooling είναι ντετερμινιστική, συνήθως υπολογίζοντας είτε τη μέγιστη είτε τη μέση τιμή των στοιχείων στο παράθυρο pooling. Αυτές οι λειτουργίες ονομάζονται μέγιστο pooling και μέσο pooling, αντίστοιχα.

Και στις δύο περιπτώσεις, όπως και με τον τελεστή διασταυρούμενης συσχέτισης, το παράθυρο συγκέντρωσης ξεκινά από πάνω αριστερά του τανυστή εισόδου και ολισθαίνει κατά τον τανυστή εισόδου από αριστερά προς τα δεξιά και από πάνω προς τα κάτω. Σε κάθε τοποθεσία που το παράθυρο pooling συναντά, υπολογίζει τη μέγιστη ή μέση τιμή του υπό-τανυστή εισόδου στο παράθυρο, ανάλογα με το αν πρόκειται για μέγιστη ή μέση συγκέντρωση.



Σχήμα 2.11: Maximum pooling με παράθυρο pooling μεγέθους 2 × 2

Στο σχήμα 2.11 αποτυπώνεται παράδειγμα μέγιστου pooling με παράθυρο pooling μεγέθους 2 × 2. Τα 4 στοιχεία της εξόδου προκύπτουν από τη μέγιστη τιμή σε κάθε pooling παράθυρο:

$$\max(0, 1, 3, 4) = 4,$$

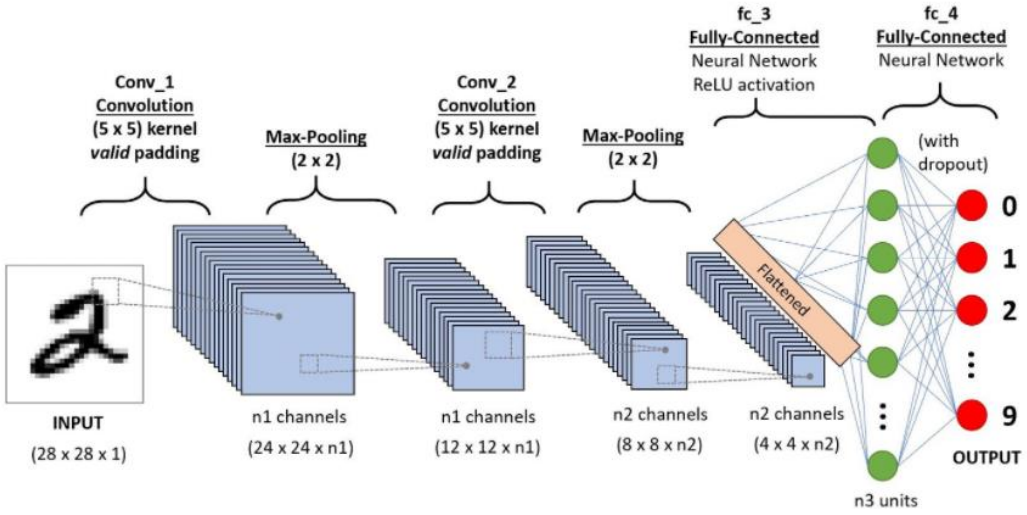
$$\max(1, 2, 4, 5) = 5,$$

$$\max(3, 4, 6, 7) = 7,$$

$$\max(4, 5, 7, 8) = 8$$

2.4.7 Χρήση των συνελικτικών επιπέδων σε μοντέλα

Όλες οι παραπάνω λειτουργίες συντελούν στην κατασκευή ενός πλήρως λειτουργικού νευρωνικού δικτύου συνέλιξης. Συνήθως, μετά τα συνελικτικά επίπεδα, προστίθετο ένα πλήρως συνδεδεμένο μπλοκ στο οποίο τροφοδοτούνται τα δεδομένα αφού μετατρέπονται σε δισδιάστατα. Ουσιαστικά, τα συνελικτικά επίπεδα φιλτράρουν τα δεδομένα εισόδου, συγκρατώντας τις πιο σημαντικές πληροφορίες τους, χρησιμοποιώντας σχετικά μικρό αριθμό παραμέτρων, κι έπειτα τα πλήρως συνδεδεμένα επίπεδα επεξεργάζονται τα δεδομένα για να παράγουν μια έξοδο, όπως περιεγράφηκε στην ενότητα των νευρωνικών δικτύων. Ένα παράδειγμα τέτοιου δικτύου φαίνεται στο σχήμα 2.12.



Σχήμα 2.12: Νευρωνικό δίκτυο που χρησιμοποιεί συνελικτικά επίπεδα (Πηγή: https://towardsdatascience.com/a-comprehensive-guide-to-convolutional-neural-networks-the-eli5-way-3bd2b1164a53)

Στην παρούσα εργασία, παρόλο που δεν αναλύεται κάποια εικόνα, η σκακιέρα αποτελείται από έναν δισδιάστατο πίνακα δεδομένων (8 σειρές και 8 στήλες), άρα υπάρχει νόημα για χρησιμοποίηση των συνελικτικών επιπέδων. Όπως σε μια εικόνα, έτσι και στην σκακιέρα, σκοπός του νευρωνικού δικτύου είναι ο εντοπισμός μοτίβων και λεπτομερειών που, σε αυτή την περίπτωση, μπορεί να οδηγούν σε ήττα ή νίκη ενός παίκτη. Κατά μια έννοια, η σκακιέρα μπορεί να αντιμετωπιστεί ως μια εικόνα, όπου κάθε τετράγωνο αναπαριστά ένα pixel και κάθε pixel αποτελείται από 6 κανάλια, ένα για κάθε διαφορετικό είδος πεσσού. Η ύπαρξη ενός πεσσού σε ένα τετράγωνο συμβολίζεται με άσσο στο αντίστοιχο κανάλι (αρνητικό για μαύρο πεσσό και θετικό για λευκό πεσσό, για παράδειγμα) σε μια μορφή one-hot encoding. Περισσότερες λεπτομέρειες δίνονται στην περιγραφή υλοποίησης παρακάτω. Στα μοντέλα της

εργασίας, δεν χρησιμοποιείται pooling, διότι σε μια σκακιέρα δεν παίζει ρόλο απλά ο εντοπισμός ενός ιδιαίτερου χαρακτηριστικού, αλλά είναι πολύ σημαντική και η θέση στην οποία εντοπίζεται. Στο σκάκι, μια παραμικρή διαφορά στις θέσεις μπορεί να σημάνει εντελώς διαφορετικό αποτέλεσμα. Χρησιμοποιείται 3 × 3 πυρήνας και padding τέτοιο ώστε το μέγεθος εξόδου να είναι ίσο με το μέγεθος εισόδου.

3 Αναζήτηση Δέντρου Monte Carlo

Στην επιστήμη των υπολογιστών, η αναζήτηση δέντρου Monte Carlo (MCTS) είναι ένας ευρετικός αλγόριθμος αναζήτησης για ορισμένα είδη διαδικασιών λήψης αποφάσεων, κυρίως εκείνων που χρησιμοποιούνται σε λογισμικό που παίζει επιτραπέζια παιχνίδια. Σε αυτό το πλαίσιο, η MCTS χρησιμοποιείται για την επίλυση του δέντρου παιχνιδιών.

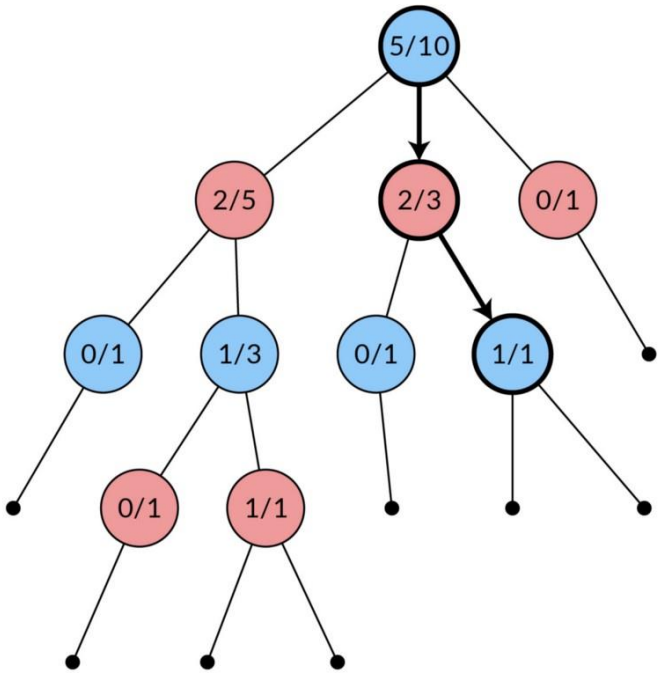
Η MCTS συνδυάστηκε με νευρωνικά δίκτυα το 2016 για το παιχνίδι Go [9] και στη συνέχεια για το σκάκι και το shogi [10] από την DeepMind της Google. Επίσης, έχει χρησιμοποιηθεί σε παιχνίδια με ελλιπείς πληροφορίες, όπως το μπριτζ και το πόκερ [11], καθώς και σε βιντεοπαιχνίδια πολλών ειδών [12].

3.1 Αρχές λειτουργίας

Το επίκεντρο της αναζήτησης δέντρου Monte Carlo είναι η ανάλυση των πιο ελπιδοφόρων κινήσεων, επεκτείνοντας το δέντρο αναζήτησης βάσει τυχαίας δειγματοληψίας του χώρου αναζήτησης. Η εφαρμογή της στα παιχνίδια βασίζεται σε πολλά παιχνίδια, που ονομάζονται roll-outs (προσομοιώσεις). Σε κάθε προσομοίωση, το παιχνίδι παίζεται μέχρι το τέλος επιλέγοντας κινήσεις τυχαία. Το τελικό αποτέλεσμα του παιχνιδιού κάθε παιχνιδιού χρησιμοποιείται στη συνέχεια για να σταθμίσει τους κόμβους στο δέντρο του παιχνιδιού, έτσι ώστε να είναι πιο πιθανό να επιλεγούν καλύτεροι κόμβοι σε μελλοντικές αναπαραγωγές.

Ο πιο βασικός τρόπος για την χρησιμοποίηση των προσομοιώσεων είναι η εφαρμογή του ίδιου αριθμού παιχνιδιών μετά από κάθε νόμιμη κίνηση του τρέχοντος παίκτη και στη συνέχεια η επιλογή της κίνησης που οδήγησε στις περισσότερες νίκες [13]. Η αποτελεσματικότητα αυτής της μεθόδου, που ονομάζεται αγνή αναζήτηση δέντρου Monte Carlo, συχνά αυξάνεται με την πάροδο του χρόνου, καθώς περισσότερα παιχνίδια ανατίθενται στις κινήσεις που έχουν οδηγήσει συχνά στη νίκη του τρέχοντος παίκτη σύμφωνα με προηγούμενες προσομοιώσεις. Κάθε γύρος αναζήτησης δέντρου Monte Carlo αποτελείται από τέσσερα βήματα.

Το πρώτο βήμα είναι η φάση της επιλογής. Ξεκινώντας από τη ρίζα R του δέντρου, επιλέγονται διαδοχικοί θυγατρικοί κόμβοι μέχρι να τερματίσει σε έναν κόμβο L. Η ρίζα είναι η τρέχουσα κατάσταση παιχνιδιού και ένα φύλλο είναι κάθε κόμβος που έχει πιθανό παιδί από τον οποίο δεν έχει ξεκινήσει ακόμη προσομοίωση. Εάν μία, περισσότερες ή όλες οι νόμιμες κινήσεις σε έναν κόμβο δεν έχουν αντίστοιχο κόμβο στο δέντρο αναζήτησης, σταματάει η επιλογή.



Σχήμα 3.1: 1ο βήμα αλγόριθμου αναζήτησης δέντρου Monte-Carlo: Επιλογή (Πηγή: https://medium.com/@quasimik/monte-carlo-tree-search-applied-to-letterpress-34f41c86e238)

Η επιλογή της πορείας θα πρέπει να επιτύχει δύο στόχους: Θα πρέπει να διερευνηθούν νέοι δρόμοι για την απόκτηση πληροφοριών και θα πρέπει να χρησιμοποιηθούν οι υπάρχουσες πληροφορίες για να αξιοποιηθούν διαδρομές που είναι γνωστές ως καλές. Για να επιτευχθούν αυτούς οι δύο στόχοι, πρέπει να επιλεχθούν θυγατρικοί κόμβοι χρησιμοποιώντας μια λειτουργία επιλογής που εξισορροπεί την εξερεύνηση και την εκμετάλλευση.

Ένας τρόπος για να γίνει αυτό είναι η επιλογή τυχαίων διαδρομών. Η τυχαία επιλογή σίγουρα εξερευνά καλά, αλλά δεν εκμεταλλεύεται καθόλου τις ήδη γνωστές διαδρομές. Ένας άλλος τρόπος είναι η χρησιμοποίηση του μέσου ποσοστού κέρδους κάθε κόμβου. Αυτό επιτυγχάνει καλή εκμετάλλευση, αλλά έχει κακή βαθμολογία στην εξερεύνηση. Μια καλή λειτουργία επιλογής που εξισορροπεί την εξερεύνηση με την εκμετάλλευση, ονομάζεται UCB1 (Upper Confidence Bound 1), που δημιουργήθηκε από τον Levente Kocsis και τον Csaba Szepesvari [14]. Όταν εφαρμόζεται στην MCTS, ο συνδυασμένος αλγόριθμος ονομάζεται UCT (Upper Confidence Bound 1 εφαρμόζεται σε δέντρα). Η συνάρτηση επιλογής UCB1 έχει την μορφή:

$$\frac{w_i}{s_i} + c\sqrt{\frac{\ln s_p}{s_i}}$$

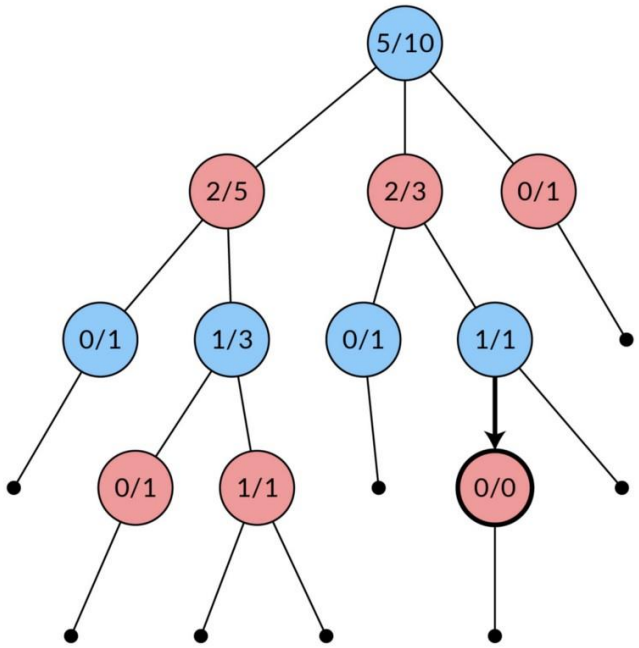
όπου w_i είναι ο αριθμός των προσομοιώσεων για αυτόν τον κόμβο που είχαν ως αποτέλεσμα τη νίκη, s_i ο συνολικός αριθμός προσομοιώσεων αυτού του κόμβου, s_p ο συνολικός αριθμός προσομοιώσεων του κόμβου-γονέα αυτού του κόμβου και c η παράμετρος εξερεύνησης.

Ο αριστερός όρος ($\frac{w_i}{s_i}$) είναι ο όρος εκμετάλλευσης. Είναι απλώς το μέσο ποσοστό νίκης, μεγαλώνοντας όσο καλύτερα έχει επιτύχει ιστορικά ένας κόμβος. Ο δεξιός όρος ($\sqrt{\frac{\ln s_p}{s_i}}$) είναι ο όρος εξερεύνησης. Μεγαλώνει όσο λιγότερο συχνά επιλέγεται ένας κόμβος για προσομοίωση. Η παράμετρος εξερεύνησης c είναι ένας αριθμός που μπορεί να επιλεχθεί αυθαίρετα και επιτρέπει τον έλεγχο της εύνοιας της εξίσωσης για την εξερεύνηση έναντι της εκμετάλλευσης. Ο συνήθης αριθμός που επιλέγεται είναι $c = \sqrt{2}$.

Οι αριθμοί στο εσωτερικό των κόμβων στο σχήμα 3.1 είναι τα στατιστικά στοιχεία για αυτόν τον κόμβο, που αντιστοιχούν σε αριθμό νικών και συνολικό αριθμό προσομοιώσεων (w_i και s_i). Κάθε φορά που πρέπει να επιλεχθεί ένας μεταξύ των πολλών θυγατρικών κόμβων, χρησιμοποιείται η συνάρτηση επιλογής UCB1 για την λήψη μιας τιμής UCB1 για κάθε θυγατρικό κόμβο και επιλέγεται ο θυγατρικός κόμβος με τη μέγιστη τιμή.

Το πρόβλημα της ταυτόχρονης εξισορρόπησης ενός πράκτορα μεταξύ της εξερεύνησης και της εκμετάλλευσης μεταξύ πολλών επιλογών, όταν το αποτέλεσμα κάθε επιλογής είναι άγνωστο, ονομάζεται multi-armed bandit problem και είναι ένα γνωστό πρόβλημα στον προγραμματισμό και στην κατανομή πόρων.

Αφού σταματήσει η επιλογή, θα υπάρχει τουλάχιστον μία κίνηση που δεν έχει εξερευνηθεί στο δέντρο αναζήτησης (εφεξής αναφερόμενες ως μη διευρυμένες κινήσεις). Τώρα, επιλέγεται τυχαία μία μη διευρυμένη κίνηση και στη συνέχεια δημιουργείται ο θυγατρικός κόμβος που αντιστοιχεί σε αυτήν την κίνηση (σημειώνεται με έντονο περίγραμμα στο σχήμα 3.2). Προστίθεται αυτός ο κόμβος ως παιδί στον τελευταίο επιλεγμένο κόμβο στη φάση επιλογής, διευρύνοντας το δέντρο αναζήτησης. Οι πληροφορίες στατιστικών στον κόμβο αρχικοποιούνται με 0 νίκες από 0 προσομοιώσεις ($w_i = 0$, $s_i = 0$).

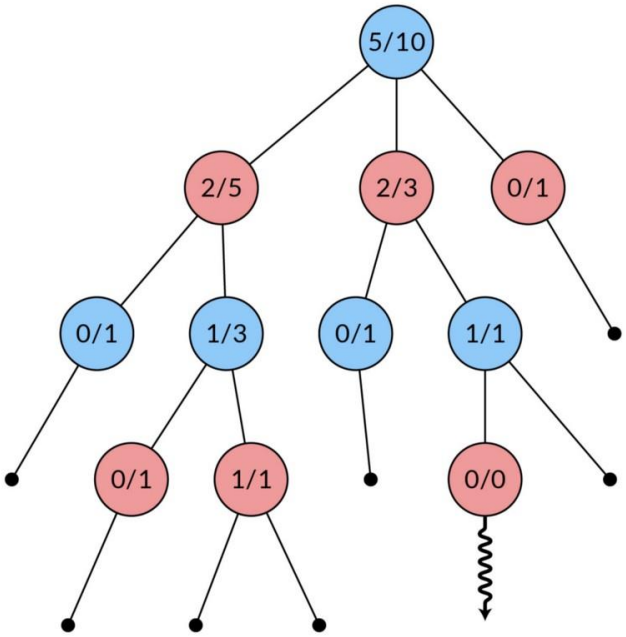


Σχήμα 3.2: 2ο βήμα αλγόριθμου αναζήτησης δέντρου Monte-Carlo: Επέκταση (Πηγή: https://medium.com/@quasimik/monte-carlo-tree-search-applied-to-letterpress-34f41c86e238)

Ορισμένες υλοποιήσεις επιλέγουν να επεκτείνουν το δέντρο κατά πολλούς κόμβους ανά προσομοίωση, αλλά η πιο αποδοτική στη μνήμη εφαρμογή είναι η δημιουργία μόνο ενός κόμβου ανά προσομοίωση. Το δέντρο αναζήτησης μπορεί να γίνει πολύ μεγάλο, ειδικά για παιχνίδια με μεγάλους παράγοντες διακλάδωσης.

Συνεχίζοντας από τον κόμβο που μόλις δημιουργήθηκε στη φάση επέκτασης, οι κινήσεις επιλέγονται τυχαία και η κατάσταση του παιχνιδιού προωθείται επανειλημμένα. Αυτή είναι η τρίτη φάση, η φάση της προσομοίωσης. Αυτό επαναλαμβάνεται μέχρι να τελειώσει το παιχνίδι και να βγει νικητής. Δεν δημιουργούνται νέοι κόμβοι σε αυτή τη φάση.

Σε αυτή τη φάση, απλώς εφαρμόζονται οι κανόνες του παιχνιδιού για να βρίσκονται επανειλημμένα όλες οι νόμιμες κινήσεις στην τρέχουσα κατάσταση του παιχνιδιού, να επιλέγεται μια νόμιμη κίνηση τυχαία και στη συνέχεια να προωθείται η κατάσταση του παιχνιδιού. Κανένα μέρος αυτής της διαδικασίας δεν αποθηκεύεται. Αυτή η φάση τελειώνει όταν βρεθεί μια κατάσταση όπου το παιχνίδι έχει τελειώσει.

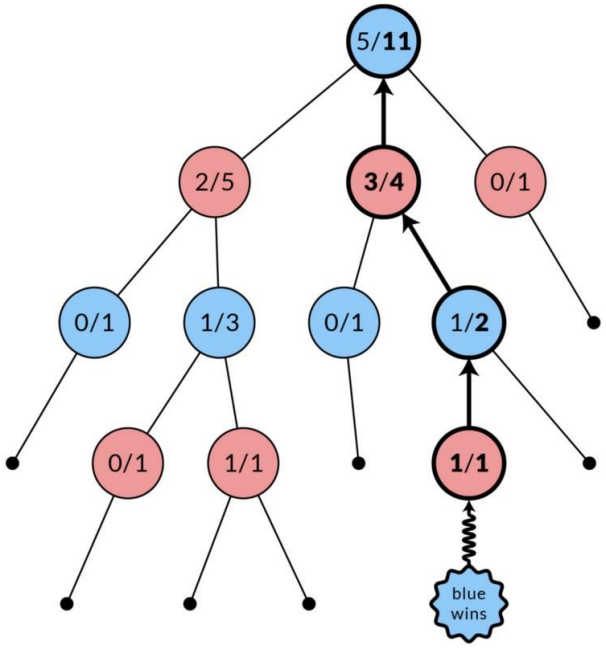


Σχήμα 3.3: 3ο βήμα αλγόριθμου αναζήτησης δέντρου Monte-Carlo: Προσομοίωση (Πηγή: https://medium.com/@quasimik/monte-carlo-tree-search-applied-to-letterpress-34f41c86e238)

Μετά τη φάση της προσομοίωσης, τα στατιστικά στοιχεία για όλους τους κόμβους που συμμετείχαν (με έντονο περίγραμμα στο σχήμα 3.4) ενημερώνονται. Αυτό αποτελεί την τέταρτη φάση, την φάση της οπισθοδρόμησης. Κάθε κόμβος που επισκέπτεται αυξάνει τον αριθμό προσομοίωσης του s_i. Ανάλογα με το ποιος παίκτης κερδίζει, ο αριθμός νικών του w_i μπορεί επίσης να αυξηθεί. Στο σχήμα 3.4, κερδίζει το μπλε, οπότε ο αριθμός νικών κάθε κόκκινου κόμβου που επισκέπτεται αυξάνεται. Αυτή η αντιστροφή οφείλεται στο γεγονός ότι τα στατιστικά στοιχεία κάθε κόμβου χρησιμοποιούνται για την επιλογή του γονικού του κόμβου του και όχι για τη δική του.

Για ένα παιχνίδι δύο παικτών όπως το σκάκι, η MCTS προσπαθεί να βρει τις καλύτερες κινήσεις σε ένα μονοπάτι για κάθε παίκτη, αντίστοιχα. Έτσι, όπως βρίσκει την καλύτερη κίνηση του μπλε για τους κόμβους όπου το μπλε πρόκειται να κινηθεί, βρίσκει επίσης την καλύτερη κίνηση του κόκκινου για τους κόμβους όπου το κόκκινο είναι επόμενο να κινηθεί.

Υπάρχει, βέβαια, και η έκδοση του αλγορίθμου όπου το τελικό αποτέλεσμα διαδίδεται σε όλους τους ενδιάμεσους κόμβους, ανεξαρτήτως του παίκτη που πρόκειται να παίξει. Η επιλογή έχει να κάνει με τον τρόπο βαθμολόγησης της νίκης. Για παράδειγμα, αν το τελικό αποτέλεσμα είναι αριθμός αρνητικός για την νίκη του ενός παίκτη και θετικός για την νίκη του άλλου, μπορεί να διαδίδεται σε όλες τις ενδιάμεσες καταστάσεις. Αν είναι απλός μετρητής νικών του ενός, μπορεί να διαδίδεται μόνο στους κόμβους του αντιπάλου, όπως αναφέρθηκε προηγουμένως.



Σχήμα 3.4: 4ο βήμα αλγόριθμου αναζήτησης δέντρου Monte-Carlo: Οπισθοδρόμηση (Πηγή: https://medium.com/@quasimik/monte-carlo-tree-search-applied-to-letterpress-34f41c86e238)

3.2 Πλεονεκτήματα και μειονεκτήματα χρήσης

Αν και έχει αποδειχθεί ότι η αξιολόγηση των κινήσεων στην αναζήτηση δέντρου Monte Carlo συγκλίνει στην minimax, [15] η βασική έκδοση της αναζήτησης δέντρου Monte Carlo συγκλίνει μόνο στα λεγόμενα παιχνίδια "Τέλεια Monte Carlo" [16]. Ωστόσο, η αναζήτηση δέντρου Monte Carlo προσφέρει σημαντικά πλεονεκτήματα έναντι του κλαδέματος α-β και παρόμοιων αλγορίθμων που ελαχιστοποιούν τον χώρο αναζήτησης.

Συγκεκριμένα, η βασική αναζήτηση δέντρου Monte Carlo δεν χρειάζεται μια ρητή συνάρτηση αξιολόγησης. Η απλή εφαρμογή των κανόνων του παιχνιδιού είναι αρκετή για να εξερευνηθεί ο χώρος αναζήτησης (δηλαδή τη παραγωγή επιτρεπόμενων κινήσεων σε μια δεδομένη θέση και των συνθηκών λήξης του παιχνιδιού). Ως εκ τούτου, μπορεί να χρησιμοποιηθεί σε παιχνίδια χωρίς ανεπτυγμένη θεωρία και γενικά στα παιχνίδια.

Το δέντρο του παιχνιδιού στην αναζήτηση δέντρου Monte Carlo μεγαλώνει ασύμμετρα καθώς η μέθοδος επικεντρώνεται στα πιο πολλά υποσχόμενα υπό-δέντρα. Έτσι, επιτυγχάνει καλύτερα αποτελέσματα από τους κλασικούς αλγόριθμους σε παιχνίδια με υψηλό συντελεστή διακλάδωσης.

Ένα μειονέκτημα είναι ότι, σε μια κρίσιμη θέση έναντι ενός έμπειρου παίκτη, μπορεί να υπάρχει ένας μόνο κλάδος που οδηγεί σε ήττα. Επειδή αυτό δεν βρίσκεται εύκολα τυχαία, η αναζήτηση μπορεί να μην το "δει" και δεν θα το λάβει υπόψη. Πιστεύεται ότι αυτό μπορεί να ήταν μέρος της αιτίας της ήττας του AlphaGo στο τέταρτο παιχνίδι του με τον Lee Sedol, τον παγκόσμιο πρωταθλητή στο Go. Ουσιαστικά, η αναζήτηση επιχειρεί να κλαδέψει ακολουθίες που είναι λιγότερο σχετικές. Σε ορισμένες περιπτώσεις, ένα παιχνίδι μπορεί να οδηγήσει σε μια πολύ συγκεκριμένη γραμμή παιχνιδιού που είναι σημαντική, αλλά η οποία παραβλέπεται όταν κλαδεύεται το δέντρο και αυτό το αποτέλεσμα είναι "εκτός ραντάρ αναζήτησης".

Στην εργασία χρησιμοποιήθηκε ο αλγόριθμος αναζήτησης δέντρου Monte Carlo σε συνδυασμό με το νευρωνικό δίκτυο πιθανοτήτων κινήσεων και το νευρωνικό δίκτυο αξιολόγησης θέσεων, με σκοπό την εύρεση της καλύτερης θέσης, δεδομένου ενός χρονικού ορίου. Το νευρωνικό δίκτυο πιθανοτήτων κίνησης χρησιμοποιείται στον UCB1 αλγόριθμο, κατά την διαδικασία της επιλογής, ώστε να βρεθεί μια κίνηση με χαμηλό αριθμό, έως εκείνο το σημείο, επισκέψεων και υψηλή πιθανότητα πραγματοποίησης, σύμφωνα με το δίκτυο. Το νευρωνικό δίκτυο αξιολόγησης θέσεων χρησιμοποιείται στην φάση της προσομοίωσης. Λόγω του μεγάλου βάθους του παιχνιδιού, οι προσομοιώσεις δεν προχωράνε πάντα μέχρι να καταλήξουν σε τελικό αποτέλεσμα, αλλά προχωράνε ένα συγκεκριμένο αριθμό κινήσεων, λόγω χρόνου. Στο σημείο που σταματάει η προσομοίωση, χρησιμοποιείται το νευρωνικό δίκτυο αξιολόγησης για να κρίνει την συγκεκριμένη θέση τερματισμού και στην συνέχεια γίνεται η οπισθοδιάδοση της αξιολόγησης αυτής.

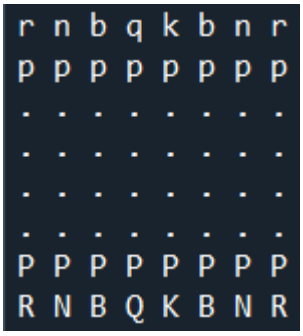
4 Υλοποίηση Σκακιστικής Μηχανής

Για την υλοποίηση της σκακιστικής μηχανής χρησιμοποιήθηκε η γλώσσα προγραμματισμού Python και συγκεκριμένα η έκδοση 3.7.1. Η υλοποίηση έγινε στο περιβάλλον προγραμματισμού Spyder της Anaconda. Ο λόγος που επιλέχτηκε η Python είναι οι μεγάλες ευκολίες που παρέχει στην ανάπτυξη εφαρμογών μηχανικής μάθησης και διαχείρισης δεδομένων. Το μειονέκτημα της χρήσης της είναι οι χαμηλότερες επιδόσεις σε δεδομένο χρόνο, σε σχέση με γλώσσες χαμηλού επιπέδου. Για την βελτίωση των επιδόσεων χρησιμοποιείται κάρτα γραφικής επεξεργασίας (NVIDIA GeForce GTX 1660 Ti with Max-Q Design) στις προβλέψεις των νευρωνικών δικτύων.

4.1 Δημιουργία περιβάλλοντος παιχνιδιού

Πρώτο βήμα για την υλοποίηση της σκακιστικής μηχανής είναι η δημιουργία του παιχνιδιού, των κανόνων που το αποτελούν και των απαραίτητων μηνυμάτων ώστε να μπορεί ένας χρήστης να παίξει. Για την δημιουργία του περιβάλλοντος παιχνιδιού χρησιμοποιήθηκε η βιβλιοθήκη “python-chess” της Python [17]. Η python-chess είναι μια βιβλιοθήκη που υποστηρίζει πολλές παραλλαγές του σκακιού, δημιουργεί την σκακιέρα σε μορφή ASCII, εντοπίζει τα ματ, τα ρουά ματ, τις ισοπαλίες και τις επαναλήψεις. Επίσης, μπορεί να δεχτεί και να επιστρέψει SAN και UCI αναπαραστάσεις κινήσεων και FENs. Η SAN αναπαράσταση κινήσεων είναι συγκεκριμένη μέθοδος που γράφονται οι κινήσεις σε προγράμματα [18]. Το FEN είναι ένας τυπικός συμβολισμός για την περιγραφή μιας συγκεκριμένης σκακιστικής θέσης [19]. Επιπλέον, η python-chess επιτρέπει την επικοινωνία με μηχανές πρωτοκόλλου UCI και Xboard.

Στην συγκεκριμένη περίπτωση, η βιβλιοθήκη χρησιμοποιείται για την αναπαράσταση της σκακιέρας, για την παραγωγή των νόμιμων κινήσεων, για την αναγνώριση του παίκτη που είναι η σειρά του να παίξει και για τον έλεγχο του αν ένα παιχνίδι έχει λήξει με κάποιον παίκτη να κερδίζει ή με ισοπαλία. Για την προώθηση των κινήσεων στην σκακιέρα χρησιμοποιείται ο UCI συμβολισμός.



Σχήμα 4.1: Αναπαράσταση σκακιέρας από τη βιβλιοθήκη python-chess

4.2 Κωδικοποίηση σκακιέρας

Αφού δημιουργείται μια αναπαράσταση της σκακιέρας, επόμενο βήμα είναι η κωδικοποίησή της ώστε να μπορεί να χρησιμοποιηθεί για την εκπαίδευση των νευρωνικών δικτύων. Αμφότερα, το νευρωνικό δίκτυο αξιολόγησης θέσεων και το νευρωνικό δίκτυο εύρεσης πιθανοτήτων κινήσεων δέχονται τη θέση της σκακιέρας ως είσοδο. Για να πραγματοποιηθεί αυτό, είναι απαραίτητη η κατάλληλη μετατροπή της αναπαράστασης της σκακιέρας σε αριθμητικά δεδομένα.

Η σκακιέρα αποτελείται από 8 στήλες και 8 γραμμές. Σε κάθε τετράγωνο μπορεί να βρίσκεται έως ένας πεσσός κάθε φορά. Οι πεσσοί, όμως, είναι διαφορετικοί μεταξύ τους και, συγκεκριμένα, υπάρχουν 6 διαφορετικά είδη πεσσών: ο βασιλιάς, η βασίλισσα, ο πύργος, ο ίππος, ο αξιωματικός και το πιόνι. Για τέτοια δεδομένα που έχουν να κάνουν με διαφορετικές κατηγορίες χρησιμοποιείται η μέθοδος one-hot encoding για την κωδικοποίησή τους [20] διότι η απλή ανάθεση ενός αριθμού σε κάθε είδος πεσσού θα σήμαινε σύγχυση για το μοντέλο υπονοώντας μαθηματική σχέση μεταξύ των πεσσών, κάτι που δεν υφίσταται. Για παράδειγμα, αν συμβολιζόταν ο ίππος με τον αριθμό 1, ο αξιωματικός με το 2 και η βασίλισσα με το 3, θα σήμαινε ότι ο αξιωματικός αποτελεί τον μέσο όρο του ίππου και της βασίλισσας, πράγμα που δεν βγάζει νόημα. Επομένως, χρειάζεται μια διαφορετική αναπαράσταση, εξ ου και η χρησιμοποίηση της μεθόδου one-hot.

Στα ψηφιακά κυκλώματα και στη μηχανική εκμάθηση, το one-hot είναι μια ομάδα δυαδικών ψηφίων μεταξύ των οποίων οι νόμιμοι συνδυασμοί τιμών είναι μόνο αυτοί με ένα μόνο υψηλό (1) bit και όλα τα άλλα χαμηλά (0) [21]. Μια παρόμοια υλοποίηση στην οποία όλα τα bit είναι "1" εκτός από ένα "0" ονομάζεται μερικές φορές “one-cold”. Στα στατιστικά στοιχεία, οι εικονικές μεταβλητές αντιπροσωπεύουν μια παρόμοια τεχνική για την αναπαράσταση κατηγορικών δεδομένων.

Επομένως, η σκακιέρα αναπαρίσταται στα μοντέλα ως μια εικόνα 8 × 8 με 6 κανάλια, δηλαδή 8 × 8 × 6, όπου οι δύο πρώτες διαστάσεις αποτελούν τις διαστάσεις της σκακιέρας 8 στηλών και 8 γραμμών και τα 6 κανάλια αντιστοιχούν στα 6 είδη

πεσσών. Μια διαφορετική αναπαράσταση χρησιμοποιείται για κάθε πεσσό, αφού η αξία του είναι ξεχωριστής σημασίας. Η διαφορά μεταξύ δύο πεσσών δεν είναι συνεχής, οπότε δεν θα ήταν σωστό να μετρηθεί στο ίδιο κανάλι. Για την αναπαράσταση των χρωμάτων, οι λευκοί πεσσοί συμβολίζονται με +1 και οι μαύροι πεσσοί με -1, καθώς θα αποτελούσε υπερβολή η ύπαρξη 12 καναλιών μιας και τα δεδομένα θα ήταν πολύ αραιά, δυσκολεύοντας, έτσι, τη διαδικασία της εκπαίδευσης.

Για παράδειγμα, ένα λευκό πιόνι σε ένα τετράγωνο συμβολίζεται με τον πίνακα [1, 0, 0, 0, 0, 0] κι ένας μαύρος ίππος με τον πίνακα [0, -1, 0, 0, 0, 0]. Ολόκληρη η σκακιέρα αναπαρίσταται με 64 (8 × 8) τέτοιους πίνακες.

Αντίστοιχα, κωδικοποιείται και ο παίκτης που είναι επόμενος να παίξει. Αν είναι η σειρά του λευκού, τα νευρωνικά δίκτυα τροφοδοτούνται με έναν επιπλέον αριθμό, ένα +1, αλλιώς με ένα -1. Διότι δεν έχει σημασία μόνο η θέση των πεσσών της σκακιέρας, αλλά και ποιος παίκτης πρόκειται να παίξει επόμενος. Δύο σκακιέρες, όπου οι πεσσοί τους βρίσκονται στις ίδιες ακριβώς θέσεις, έχουν διαφορετική αξιολόγηση, ανάλογα με το ποιος μπορεί να παίζει στην συνέχεια.

Επομένως, όσον αφορά τα νευρωνικά δίκτυα, η σκακιέρα αναπαρίσταται με μια κωδικοποίηση της θέσης της, δηλαδή των τοποθεσιών που βρίσκονται οι πεσσοί, και με μια κωδικοποίηση του παίκτη που παίζει επόμενος.

4.3 Αρχιτεκτονική δικτύων

Για την εργασία, δημιουργήθηκαν δύο διαφορετικά μοντέλα νευρωνικών δικτύων. Το πρώτο αφορά την αξιολόγηση θέσεων, δέχεται ως είσοδο την κωδικοποίηση της σκακιέρας και τον αριθμό που δείχνει ποιον παίκτη αφορά η πρόβλεψη και επιστρέφει έναν αριθμό στο διάστημα [-1, 1], όπου -1 σημαίνει ότι η σκακιέρα είναι σε μια τέλεια θέση για τον μαύρο παίκτη και 1 ότι η σκακιέρα είναι σε μια τέλεια θέση για τον λευκό, αντίστοιχα. Το δεύτερο δίκτυο αφορά την εύρεση πιθανοτήτων για το ποια είναι η επόμενη κίνηση, στην συγκεκριμένη θέση. Και αυτό το δίκτυο έχει την ίδια είσοδο, δηλαδή δέχεται ως είσοδο την κωδικοποίηση της σκακιέρας και τον αριθμό που δείχνει ποιον παίκτη αφορά η πρόβλεψη και επιστρέφει τις πιθανότητες που έχει κάθε κίνηση να παιχτεί επόμενη. Έχει 128 εξόδους, 64 για τα τετράγωνα από τα οποία ξεκινάει η κίνηση και 64 για τα τετράγωνα στα οποία τερματίζει η κίνηση. Οι πιθανότητες φιλτράρονται βάσει των επιτρεπόμενων κινήσεων κάθε φορά και γίνεται η απαραίτητη κανονικοποίηση.

Τα μοντέλα για τα νευρωνικά δίκτυα δημιουργήθηκαν με την βοήθεια του Keras API [22]. Το Keras API είναι ένα API βαθιάς μάθησης που τρέχει πάνω στην πλατφόρμα μηχανικής μάθησης Tensorflow. Αναπτύχθηκε με στόχο να δίνει τη δυνατότητα για γρήγορο πειραματισμό στους χρήστες. Το Keras επιλέχθηκε λόγω της αμεσότητάς του και των πολλών δυνατοτήτων που παρέχει. Μέσω του Keras

δημιουργήθηκαν τα μοντέλα των νευρωνικών δικτύων, ρυθμίζοντας όλες τις απαραίτητες λεπτομέρειες και έγινε η εκπαίδευσή τους.

Για τα δύο δίκτυα χρησιμοποιείται παρόμοια αρχιτεκτονική, με εξαίρεση το επίπεδο εξόδου. Αρχικά, χρησιμοποιείται ένα συνελικτικό νευρωνικό δίκτυο τριών επιπέδων, με παράθυρο συνέλιξης 3 × 3 σε κάθε επίπεδο και padding τέτοιο ώστε η εικόνα εξόδου από το επίπεδο να έχει ίδιες διαστάσεις με την εικόνα εισόδου. Μετά από κάθε επίπεδο συνέλιξης εφαρμόζεται κανονικοποίηση στις εξόδους μέσω ενός επιπέδου Batch Normalization του Keras API, που εφαρμόζει έναν μετασχηματισμό που διατηρεί τη μέση έξοδο κοντά στο 0 και την τυπική απόκλιση εξόδου κοντά στο 1. Έπειτα, σε κάθε επίπεδο συνέλιξης εφαρμόζεται η συνάρτηση ενεργοποίησης Relu, διατηρώντας τις εξόδους στο διάστημα [0, 1]. Ο αριθμός των τριών επιπέδων συνέλιξης επιλέχτηκε διότι με λιγότερα από τρία, η εκπαίδευση δεν γινόταν το ίδιο αποδοτικά λόγω των λιγότερων παραμέτρων σε ένα περίπλοκο πρόβλημα, αλλά και σε προσπάθειες με περισσότερα (πέντε, για παράδειγμα) η εκπαίδευση πάλι δεν είχε επιτυχία όσον αφορά την αλλαγή των παραμέτρων. Βρέθηκε ότι τα τρία επίπεδα οδηγούν σε πιο αποδοτική εκπαίδευση, με μικρότερες διαφορές. Ο μεγάλος αριθμός παραμέτρων χρησιμοποιείται, κυρίως, όταν πρόκειται για λίγα και αραιά δεδομένα, συνθήκη που καθιστά δύσκολη την εκπαίδευση.

Η αρχικοποίηση των βαρών έγινε με τυχαίο τρόπο από το Keras, ξεκινώντας από πολύ μικρές τιμές. Επίσης, δεν χρησιμοποιήθηκε pooling, διότι στο σκάκι συγκεκριμένα, δεν παίζει ρόλο απλά η αναγνώριση των χαρακτηριστικών, αλλά και το μέρος όπου αυτά βρέθηκαν. Χωρίς το pooling, διατηρείται ο μέγιστος αριθμός χρήσιμων δεδομένων κατά την εκπαίδευση. Οποιαδήποτε μετατροπή της εικόνας του σκακιού έχει τεράστιο αντίκτυπο στην αξιολόγησή, εξ ου και η αποφυγή του pooling.

Μετά τα τρία συνελικτικά επίπεδα και αφού η έξοδος από το τρίτο μετατρέπεται σε μονοδιάστατο διάνυσμα, τροφοδοτείται σε πλήρως συνδεδεμένο δίκτυο δύο επιπέδων. Το πλήρως συνδεδεμένο δίκτυο χρησιμοποιείται ώστε τα χαμηλού επιπέδου χαρακτηριστικά που βρέθηκαν στα συνελικτικά επίπεδα να επεξεργαστούν με ισχυρότερες λογικές πράξεις. Με αυτό τον τρόπο, αποφεύγεται και ο κορεσμός των συνελικτικών επιπέδων. Στο πλήρως συνδεδεμένο δίκτυο προστίθεται μια επιπλέον είσοδος, ένας ακέραιος αριθμός που συμβολίζει τον παίκτη που παίζει επόμενος, -1 αν επόμενος παίζει ο μαύρος και 1 αν επόμενος παίζει ο λευκός. Αυτό γίνεται διότι έχει μεγάλη σημασία στην εξέταση μιας σκακιστικής θέσης το ποιος πρόκειται να παίξει, όχι απλά η τοποθέτηση των πεσσών πάνω στην σκακιέρα. Για το μοντέλο αξιολόγησης της σκακιέρας χρησιμοποιείται πλήρως συνδεδεμένο νευρωνικό δίκτυο δύο επιπέδων με το τελευταίο επίπεδο να έχει μία έξοδο και συνάρτηση ενεργοποίησης “tanh”. Με αυτό τον τρόπο προκύπτει μια αξιολόγηση στο διάστημα [-1, 1], όπου -1 σημαίνει τέλεια θέση για τον μαύρο και 1 σημαίνει τέλεια θέση για τον λευκό. Για το μοντέλο εύρεσης πιθανοτήτων επόμενης κίνησης χρησιμοποιείται πλήρως συνδεδεμένο νευρωνικό δίκτυο δύο επιπέδων με το τελευταίο επίπεδο να έχει 128 εξόδους, 64 για το τετράγωνο εκκίνησης της κίνησης και 64 για το τετράγωνο τερματισμού της. Χρησιμοποιείται η συνάρτηση ενεργοποίησης softmax που διατηρεί τις εξόδους στο διάστημα [0, 1], μιας και αναζητούνται πιθανότητες κίνησης από ένα τετράγωνο σε ένα άλλο. Στα αντίστοιχα τετράγωνα της κίνησης δίνεται η τιμή 0.5 και στα υπόλοιπα η τιμή 0 σε μια παραλλαγή της μεθόδου one-hot encoding.

Για την εκπαίδευση χρησιμοποιείται stochastic gradient descent, και η συνάρτηση σφάλματος είναι η mean squared error. Η ταχύτητα μάθησης τέθηκε πολύ μικρή (0,005 για το μοντέλο αξιολόγησης και 0,0001 για το μοντέλο πρόβλεψης κινήσεων) διότι παρατηρήθηκε καλύτερη συμπεριφορά στην εκπαίδευση. Με μεγαλύτερες ταχύτητες μάθησης, η εκπαίδευση δεν ήταν το ίδιο αποδοτική, πιθανότατα λόγω των πολλών διαφορετικών τρόπων που ένα παιχνίδι μπορεί να καταλήξει σε ένα κοινό αποτέλεσμα. Η ποικιλομορφία του παιχνιδιού και οι πολλές πτυχές του καθιστούν αδύνατη την εκπαίδευση με μεγάλη ταχύτητα εκμάθησης, καθώς δύο παρτίδες παιχνιδιού που φαίνεται ότι μοιάζουν πολύ μπορεί να καταλήξουν σε διαφορετικό αποτέλεσμα από μια μόνο διαφορετική μεταξύ τους κίνηση.

4.4 Εκπαίδευση μοντέλων

Χρησιμοποιώντας τα δύο μοντέλα που αναφέρθηκαν, δημιουργήθηκαν τρία αντίγραφα από το καθένα, δύο για τη μηχανή σκακιού αυτό-εκπαίδευσης, δηλαδή την μηχανή ενισχυτικής μάθησης που μαθαίνει παίζοντας παρτίδες με τον εαυτό της, δύο για την μηχανή σκακιού που εκπαιδεύεται με δεδομένα από παιχνίδια του παρελθόντος και δύο για την συνδυαστική μηχανή, που συνδυάζει τις δύο μεθόδους και εκπαιδεύεται και με τους δυο τρόπους.

4.4.1 Εκπαίδευση μηχανής ενισχυτικής μάθησης

Για την πρώτη έκδοση της μηχανής, χρησιμοποιήθηκε ενισχυτική μάθηση για να εκπαιδευτούν τα νευρωνικά δίκτυα. Αρχικά, αρχικοποιούνται τα μοντέλα με τυχαία βάρη και αποθηκεύονται τοπικά, ώστε να είναι δυνατή η επαναχρησιμοποίησή τους. Έπειτα, σε μια επαναληπτική διαδικασία, το μοντέλο αξιολόγησης τίθεται αντιμέτωπο με τον εαυτό του. Αφού δημιουργηθεί ένα παιχνίδι με τη βοήθεια της βιβλιοθήκης python-chess, βρίσκονται όλες οι επιτρεπτές κινήσεις στην τρέχουσα θέση της σκακιέρας και οι θέσεις που αντιστοιχούν σε κάθε κίνηση. Οι θέσεις αυτές κωδικοποιούνται σε μορφή 8 × 8 × 6 και τροφοδοτούνται στο νευρωνικό δίκτυο αξιολόγησης θέσεων. Αν είναι η σειρά του λευκού να παίξει, κρατούνται οι μεγαλύτερες αξιολογήσεις, αυτές που προσεγγίζουν πιο πολύ το 1 δηλαδή, αλλιώς οι μικρότερες, αυτές που προσεγγίζουν πιο πολύ το -1. Δεν επιλέγεται η καλύτερη αξιολόγηση γιατί αυτό θα σήμαινε ότι θα γινόταν συνεχώς οι ίδιες κινήσεις κατά την εκπαίδευση, όσο η μηχανή παίζει χωρίς να λαμβάνει νέα δεδομένα εκπαίδευσης. Αντ’ αυτού, μέσω μιας κατανομής πιθανοτήτων, επιλέγεται κάθε φορά μία από τις καλύτερες κινήσεις, καθιστώντας κάθε παρτίδα ξεχωριστή. Η κατανομή αυτή δίνει από ένα βάρος σε κάθε κίνηση, το οποίο είναι μεγαλύτερο όσο καλύτερη θεωρείται η κίνηση από το μοντέλο αξιολόγησης. Με αυτόν τον τρόπο δίνεται μεγαλύτερη πιθανότητα στην επιλογή των καλύτερων κινήσεων, διατηρώντας έναν χαρακτήρα τυχαιότητας, ώστε να μην ακολουθείται συνεχώς το ίδιο μονοπάτι παιχνιδιού.

Αυτή η διαδικασία συνεχίζεται, εναλλάξ για τον λευκό και τον μαύρο, μέχρι να τελειώσει το παιχνίδι. Επειδή η μηχανή ανταγωνίζεται τον εαυτό της, τα περισσότερα παιχνίδια καταλήγουν ισόπαλα. Οι παρτίδες που καταλήγουν ισόπαλες δεν χρησιμοποιούνται στην εκπαίδευση διότι δεν φέρουν σημαντική πληροφορία για την αξιολόγηση των θέσεων. Τα παιχνίδια που χρησιμοποιούνται είναι αυτά που καταλήγουν με έναν από τους δύο παίκτες νικητή. Αν σε μια παρτίδα έχει νικήσει κάποιος, αποθηκεύονται όλες οι θέσεις της παρτίδας για να εκπαιδεύσουν το δίκτυο αξιολόγησης. Αφού κωδικοποιούνται οι θέσεις τροφοδοτούνται στο δίκτυο, με ετικέτα 1 αν πρόκειται για νίκη του λευκού και ετικέτα -1 αν πρόκειται για νίκη του μαύρου. Με αυτή τη διαδικασία το δίκτυο αξιολόγησης αντιλαμβάνεται τις καλές και κακές θέσεις για τον κάθε παίκτη σταδιακά. Η φύση του σκακιού είναι τέτοια που και η παραμικρή κίνηση μπορεί να έχει τεράστιο αντίκτυπο στο τελικό αποτέλεσμα, για αυτό και χρησιμοποιούνται όλες οι ενδιάμεσες θέσεις για την εκπαίδευση. Επίσης, στο σκάκι δεν υφίσταται το άμεσο κέρδος για έναν παίκτη, δηλαδή δεν μπορεί να αξιολογηθεί θετικά μια κίνηση που εκείνη τη στιγμή φαίνεται καλή επειδή μπορεί μακροπρόθεσμα να είναι κακή για έναν παίκτη και το αντίστροφο. Στο σκάκι η μόνη πραγματική αξιολόγηση για το αν μια κίνηση είναι καλή ή κακή είναι το τελικό αποτέλεσμα, οποιαδήποτε άλλη προσπάθεια αξιολόγησης μπορεί να αποδειχτεί μη έγκυρη.

Με αντίστοιχο τρόπο εκπαιδεύεται και το μοντέλο πρόβλεψης πιθανοτήτων κινήσεων, μόνο που εδώ δεν παίζει ρόλο το τελικό αποτέλεσμα, οπότε μπορούν να χρησιμοποιηθούν όλες οι κινήσεις από όλες τις παρτίδες. Αποθηκεύονται όλες οι θέσεις, ο επόμενος παίκτης κάθε φορά ως είσοδος και η κίνηση που έγινε ως ετικέτα και τροφοδοτούνται στο μοντέλο για εκπαίδευση.

Αυτή η διαδικασία επαναλαμβάνεται συνεχώς, ανά 50 παρτίδες κάθε φορά προτού γίνει η εκπαίδευση. Στα 50 παιχνίδια, περίπου στα 2 με 4 παρατηρείται νικητής. Ο αριθμός 50 επιλέχτηκε διότι σε λιγότερα παιχνίδια, συχνά δεν παρατηρούνταν κανένας νικητής οπότε δεν εκπαιδευόταν γρήγορα το μοντέλο αξιολόγησης. Αντίστοιχα, σε περισσότερα από 50 παιχνίδια, 100 για παράδειγμα, παρατηρήθηκε ένα φαινόμενο υπεροχής του ενός παίκτη έναντι του άλλου μετά από ένα χρονικό διάστημα. Επειδή η διαδικασία έχει στοχαστικό χαρακτήρα λόγω της προσθήκης τυχαιότητας στην επιλογή των κινήσεων και των αρχικά τυχαίων βαρών, είναι πιθανό μετά από ένα σημείο ο ένας από τους δύο παίκτες να κερδίζει πολλά περισσότερα παιχνίδια. Αν αυτό επαναλαμβάνεται και το δίκτυο τροφοδοτείται με όλο και περισσότερα παιχνίδια στα οποία κερδίζει ο ίδιος παίκτης, αυξάνεται περισσότερο η ισχύς του έναντι του άλλου, καθώς λαμβάνει εγκυρότερες αξιολογήσεις. Ο κίνδυνος υπεροχής του ενός παίκτη ναι μεν επιβεβαιώνει ότι η εκπαίδευση λειτουργεί σωστά, αφού φαίνεται ότι όσο περισσότερη εκπαίδευση ένας παίκτης λαμβάνει, τόσο καλύτερα παίζει, αλλά αποτελεί κίνδυνο για τη διαδικασία της εκπαίδευσης και μπορεί να την εκτροχιάσει πλήρως. Αν κερδίζει συνεχώς ένας παίκτης, όλες οι τροφοδοτούμενες αξιολογήσεις θα είναι υπέρ του που σημαίνει ότι, μετά από ένα σημείο, το μοντέλο θα θεωρεί όλες τις θέσεις καλές για αυτόν, άρα θα μειώνεται η εγκυρότητα της συνάρτησης αξιολόγησης που το μοντέλο προσπαθεί να προσεγγίσει.

Για να αντιμετωπιστεί ο κίνδυνος της υπεροχής αυτής υπάρχουν πολλοί τρόποι. Ο ένας είναι ο περιορισμός της δύναμης του παίκτη που υπερέχει, αλλάζοντας την κατανομή πιθανότητας σύμφωνα με την οποία επιλέγει μια κίνηση, ώστε να κάνει λίγο χειρότερες επιλογές έως ότου χάσει μερικά παιχνίδια και εξισορροπιστεί η διαφορά. Αντίστοιχα, μπορεί να ενισχυθεί ο χειρότερος παίκτης με την αντίστροφη διαδικασία. Ένας άλλος τρόπος είναι η χρησιμοποίηση μόνο των παιχνιδιών που ο υπερέχων παίκτης χάνει για ένα χρονικό διάστημα, ώστε να βελτιωθεί ο πιο αδύναμος παίκτης μέσω της εκπαίδευσης, όσο ο άλλος μένει στάσιμος.

Στην επιλογή των κινήσεων, κατά τη διαδικασία του παιχνιδιού της μηχανής εναντίον του εαυτού της, δεν χρησιμοποιήθηκε η αναζήτηση δέντρου Monte Carlo, αν και αυτή θα ήταν η πιο σωστή επιλογή. Ο λόγος είναι ότι δεν υπάρχουν οι απαραίτητοι υπολογιστικοί πόροι για αυτή τη διαδικασία, καθώς είναι πολύ πιο απαιτητική και χρειάζεται πολύ περισσότερο χρόνο για να παρουσιάσει αποτελέσματα. Αντ’ αυτού, χρησιμοποιήθηκε μόνο η αξιολόγηση του μοντέλου αξιολόγησης.

4.4.2 Εκπαίδευση μηχανής εποπτευόμενης μηχανικής μάθησης

Για την μηχανή εποπτευόμενης μηχανικής χρησιμοποιούνται υπάρχοντα δεδομένα από πραγματικούς αγώνες. Τα δεδομένα αντλήθηκαν από τον ιστότοπο ficsgames.org [23], μια μεγάλη σκακιστική βάση δεδομένων. Προσφέρει τα παιχνίδια που έλαβαν μέρος στον ελεύθερο διαδικτυακό server σκακιού: Free Internet Chess Server [24]. Η βάση δεδομένων ανανεώνεται σε πραγματικό χρόνο, ώστε να είναι δυνατή η πρόσβαση στις παρτίδες αμέσως μετά την ολοκλήρωσή τους. Η βάση περιλαμβάνει όλα τα παιχνίδια που έχουν παιχτεί στην FICS μετά τον Νοέμβριο του 1999.

Τα δεδομένα δίνονται σε αρχεία PGN (.pgn). Το PGN (Portable Game Notation) ή Φορητή Γραφή Παιχνιδιού είναι ένα format κειμένου που χρησιμοποιείται για την αποθήκευση σκακιστικών παρτίδων, αφενός των κινήσεων που γίνονται και αφετέρου άλλων δεδομένων που αφορούν τον αγώνα. Τα αρχεία είναι εύκολα αναγνώσιμα από ανθρώπους και υποστηρίζονται από πολλά λογισμικά που αφορούν το σκάκι [25]. Στο σχήμα 4.2 φαίνεται ένα παράδειγμα μορφής αρχείου pgn. Στο πάνω μέρος υπάρχουν στοιχεία για το παιχνίδι, όπως η τοποθεσία και η μέρα που έλαβε χώρα, οι παίκτες που ήρθαν αντιμέτωποι και το τελικό αποτέλεσμα. Στο κάτω μέρος αναγράφονται όλες οι κινήσεις που έγιναν σε κάθε γύρο, ξεκινώντας πάντοτε από τον λευκό παίκτη. Για την εκπαίδευση απομονώθηκε το τελικό αποτέλεσμα και οι κινήσεις που έγιναν σε κάθε αγώνα. Η υπόλοιπες πληροφορίες δεν αξιοποιήθηκαν κάπως.

```
[Event "F/S Return Match"]
[Site "Belgrade, Serbia JUG"]
[Date "1992.11.04"]
[Round "29"]
[White "Fischer, Robert J."]
[Black "Spassky, Boris V."]
[Result "1/2-1/2"]

1. e4 e5 2. Nf3 Nc6 3. Bb5 a6 {This opening is called the Ruy Lopez.}
4. Ba4 Nf6 5. O-O Be7 6. Re1 b5 7. Bb3 d6 8. c3 O-O 9. h3 Nb8 10. d4 Nbd7
11. c4 c6 12. cxb5 axb5 13. Nc3 Bb7 14. Bg5 b4 15. Nb1 h6 16. Bh4 c5 17. dxe5
Nxe4 18. Bxe7 Qxe7 19. exd6 Qf6 20. Nbd2 Nxd6 21. Nc4 Nxc4 22. Bxc4 Nb6
23. Ne5 Rae8 24. Bxf7+ Rxf7 25. Nxf7 Rxe1+ 26. Qxe1 Kxf7 27. Qe3 Qg5 28. Qxg5
hxg5 29. b3 Ke6 30. a3 Kd6 31. axb4 cxb4 32. Ra5 Nd5 33. f3 Bc8 34. Kf2 Bf5
35. Ra7 g6 36. Ra6+ Kc5 37. Ke1 Nf4 38. g3 Nxh3 39. Kd2 Kb5 40. Rd6 Kc5 41. Ra6
Nf2 42. g4 Bd3 43. Re6 1/2-1/2
```

Σχήμα 4.2: Καταγεγραμμένος αγώνας σε μορφή PGN (Πηγή: https://en.wikipedia.org/wiki/Portable_Game_Notation)

Τα μοντέλα που χρησιμοποιήθηκαν και σε αυτή την μηχανή παρουσιάζουν ίδια αρχιτεκτονική με τα μοντέλα ενισχυτικής μάθησης. Η διαφορά έγκειται στον τρόπο εκπαίδευσης. Κατά τη διαδικασία της εκπαίδευσης, διαβάζονται με τη σειρά τα παιχνίδια του αρχείου και αποθηκεύεται το τελικό αποτέλεσμα, οι θέσεις και οι κινήσεις κάθε παιχνιδιού, ώστε να χρησιμοποιηθούν στην τροφοδότηση του μοντέλου πρόβλεψης πιθανοτήτων κινήσεων. Έπειτα, αν το αποτέλεσμα είναι νικηφόρο για κάποιο παίκτη, τα δεδομένα του παιχνιδιού χρησιμοποιούνται και στην εκπαίδευση του μοντέλου αξιολόγησης. Όπως και στο μοντέλο ενισχυτικής μάθησης, δημιουργείται μια σκακιέρα με τη βιβλιοθήκη python-chess και κάθε κίνηση που διαβάζεται από τα αρχεία προωθείται και στην σκακιέρα. Έτσι, εκτυλίσσεται κάθε παιχνίδι και αποθηκεύονται οι θέσεις που αντιστοιχούν στις κινήσεις που σημειώνονται στο αρχείο.

4.4.3 Εκπαίδευση συνδυαστικής μηχανής

Η τρίτη έκδοση της μηχανής, η συνδυαστική μηχανή αποτελείται, επίσης, από μοντέλα ίδιας αρχιτεκτονικής. Η μηχανή εκπαιδεύτηκε και με τους δύο τρόπους που προαναφέρθηκαν. Αρχικά, μέσω ενισχυτικής μάθησης και έπειτα με εποπτευόμενη μάθηση με τις διαδικασίες που αναλύθηκαν. Ο λόγος που δημιουργήθηκε είναι για να συγκριθεί με τις άλλες μηχανές και να βρεθεί αν έχει νόημα ο συνδυασμός των δύο μεθόδων στην εκπαίδευση των μοντέλων, όσων αφορά την απόδοση. Η δημιουργία της, δηλαδή, έχει ερευνητικό χαρακτήρα.

4.5 Χρήση αναζήτησης δέντρου Monte Carlo

Για την εξερεύνηση του δέντρου θέσεων του παιχνιδιού χρησιμοποιείται ο αλγόριθμος αναζήτησης δέντρου Monte Carlo. Ο λόγος που επιλέχτηκε η χρήση του αλγορίθμου αυτού σε σχέση με τον αλγόριθμο Minimax με α-β κλάδεμα που παραδοσιακά χρησιμοποιείται σε τέτοια προβλήματα, είναι αφενός τα καλά αποτελέσματα που παρουσιάζει σε προβλήματα με μεγάλο παράγοντα διακλάδωσης όπως το σκάκι, και αφετέρου ο πειραματισμός με έναν καινούργιο, σχετικά, αλγόριθμο που χρησιμοποιείται όλο και περισσότερο τα τελευταία χρόνια στον τομέα της τεχνητής νοημοσύνης και παρουσιάζει μεγάλες προοπτικές και περιθώρια βελτίωσης.

Αυτή η επιλογή έχει, φυσικά, ένα σημαντικό μειονέκτημα. Σε καταστάσεις όπου ο αλγόριθμος τρέχει για πολύ λίγες επαναλήψεις, για λίγο χρόνο, δηλαδή, παρουσιάζει πολύ χειρότερα αποτελέσματα από τον αλγόριθμο Minimax. Σε χρόνο ενός δευτερολέπτου, για παράδειγμα, ο Minimax υπερέχει κατά πολύ. Παρόλα αυτά, όσο περισσότερος χρόνος δίνεται τόσο καλύτερα αποτελέσματα παρουσιάζει και πλησιάζει σε απόδοση τον αλγόριθμο Minimax.

Από την στιγμή, όμως, που γίνεται λόγος για ανταγωνιστικά παιχνίδια, η έννοια του χρόνου είναι πολύ σημαντική. Ο παραδοσιακός αλγόριθμος αναζήτησης δέντρου Monte Carlo δεν μπορεί να ανταγωνιστεί τον Minimax με κλάδεμα α-β. Για αυτόν τον λόγο υλοποιήθηκε μια βελτιωμένη έκδοση του αλγορίθμου, χρησιμοποιώντας το μοντέλο αξιολόγησης για την αντικατάσταση των προσομοιώσεων του παιχνιδιού έως το τέλος τους, καθώς και το μοντέλο πρόβλεψης πιθανοτήτων κινήσεων για το κλάδεμα του δέντρου, αποκλείοντας κινήσεις που το μοντέλο θεωρεί ότι δεν πραγματοποιούνται συχνά.

4.5.1 Υλοποίηση αλγορίθμου

Ο αλγόριθμος ξεκινάει έχοντας ως είσοδο τα μοντέλα που χρησιμοποιούνται, το αν πρόκειται για κίνηση του λευκού ή του μαύρου παίκτη, το παιχνίδι του σκακιού για το οποίο θα γίνει η κίνηση, το χρονικό όριο που έχει στη διάθεσή του και τους γύρους που προσομοιώνονται στη φάση της προσομοίωσης.

Η τρέχουσα θέση του παιχνιδιού αποτελεί την ρίζα του δέντρου αναζήτησης. Στο δέντρο προστίθενται και όλες οι πιθανές θέσεις από την ρίζα. Όλοι οι κόμβοι-θέσεις του δέντρου χαρακτηρίζονται από έναν μετρητή επισκέψεων του αλγορίθμου, έναν ακέραιο αριθμό που συμβολίζει ποιος παίκτης παίζει επόμενος (1 για τον λευκό και -1 για τον μαύρο), ένα σύνολο σκορ που συγκεντρώνει ο κόμβος από το σύνολο των οπισθοδρομήσεων, μια λίστα με τους κόμβους-παιδιά του κόμβου (που αρχικοποιείται κενή), ένας αριθμός που συμβολίζει την πιθανότητα να γίνει πραγματική η τρέχουσα κατάσταση από την προηγούμενη και ένα αλφαριθμητικό στο οποίο αναγράφεται η

θέση που συμβολίζει ο κόμβος σε μορφή FEN. Έπειτα, ξεκινάει η επαναληπτική διαδικασία του αλγορίθμου που εξηγήθηκε στην ενότητα 3, για δεδομένο χρονικό διάστημα που έχει τεθεί ως είσοδος. Αρχικά, γίνεται η επιλογή κόμβου-παιδιού με το μεγαλύτερο σκορ αν πρόκειται για κόμβο που επόμενος παίζει ο λευκός ή με το μικρότερο σκορ αν πρόκειται για κόμβο που επόμενος παίζει ο μαύρος. Αυτό επειδή το μοντέλο αξιολόγησης επιστρέφει αρνητικές τιμές για θέσεις ευνοϊκές για τον μαύρο και θετικές τιμές για θέσεις ευνοϊκές για τον λευκό παίκτη.

Η διαδικασία της επιλογής βασίζεται στη μέθοδο UCB1. Κάθε κόμβος του δέντρου σχετίζεται με μία κατάσταση-θέση s. Για κάθε επιτρεπτή κίνηση m από τη θέση s, υπάρχει ένας κόμβος (s, m) στον οποίο αποθηκεύονται μια σειρά στατιστικών, όπως αναφέρθηκαν. Κάθε προσομοίωση ξεκινάει από μία κατάσταση s^o και τελειώνει όταν βρίσκεται ένας κόμβος-φύλλο s^n. Όταν χρησιμοποιείται το μοντέλο πρόβλεψης πιθανότητας κινήσεων, σε κάθε βήμα του αλγορίθμου, η κίνηση του λευκού παίκτη επιλέγεται με την σχέση:

$$m^n = argmax_n \left[\begin{cases} +\infty, N_b = 0 \\ \frac{V_a}{N_b} + c_1 P(a,b) \sqrt{\frac{log(N_a)}{N_b}}, N_b > 0 \end{cases} \right]$$

Το a συμβολίζει τον κόμβο-γονέα από τον οποίο πρέπει να επιλεχτεί μια κίνηση m προς τον κόμβο b. Το V συμβολίζει το σύνολο των αξιολογήσεων που έχει λάβει ο κόμβος από τη διαδικασία της οπισθοδιάδοσης. Το N συμβολίζει το σύνολο των επισκέψεων του κόμβου από τον αλγόριθμο, για τον κόμβο-γονέα και τους κόμβους-παιδιά για a και b αντίστοιχα. Η σταθερά c_1 τέθηκε σε $c_1 = 1,5$, μετά από διάφορους πειραματισμούς. Με τη σταθερά c_1 γίνεται η εξισορρόπηση εξερεύνησης και εκμετάλλευσης, οπότε και προσαρμόζεται ανάλογα. Η τιμή P(a,b) συμβολίζει έναν αριθμό που προέρχεται από το μοντέλο πρόβλεψης πιθανότητας κινήσεων και αντιπροσωπεύει έναν αριθμό-πιθανότητα να συμβεί η κίνηση από τη θέση a στη b σύμφωνα με το νευρωνικό δίκτυο. Το νευρωνικό δίκτυο επιστρέφει 128 τιμές, 64 για την πιθανότητα να ξεκινήσει η κίνηση από ένα τετράγωνο και 64 για την πιθανότητα να τερματίσει η κίνηση στο τετράγωνο αυτό. Για κάθε νόμιμη κίνηση, οι πιθανότητες προστίθενται για να προκύψει η αντίστοιχη πιθανότητα κίνησης. Η χρήση των πιθανοτήτων έχει τον ρόλο του κλαδέματος του δέντρου καταστάσεων, κάτι πολύ χρήσιμο λόγω του μεγάλο παράγοντα διακλάδωσης που το σκάκι παρουσιάζει. Αυτό συμβαίνει λόγω του γεγονότος ότι οι κινήσεις που είναι πιο πιθανό να συμβούν, θα επιλεχτούν πρώτες από τον αλγόριθμο και θα αναλυθούν οι θέσεις που προκύπτουν από αυτές, παρόμοια με τον τρόπο που το κλάδεμα α-β απομονώνει τις κινήσεις που αξιολογούνται ως καλύτερες, αγνοώντας κινήσεις που δεν θεωρούνται καλές. Αυτό θα μπορούσε να γίνει χρησιμοποιώντας το μοντέλο αξιολόγησης αντί το μοντέλο πρόβλεψης πιθανοτήτων, όμως θα χρειαζόταν να καλείται πολλαπλές φορές ανά προσομοίωση, κάτι που θα αύξανε κατά πολύ τον χρόνο του αλγορίθμου, καθώς η

χρήση των μοντέλων είναι η λειτουργία με το μεγαλύτερο κόστος χρόνου για τον αλγόριθμο. Στην παρούσα μορφή, καλείται μόνο μια φορά κάποιο μοντέλο, το μοντέλο πιθανοτήτων, κάθε φορά που γίνεται η προσθήκη ενός κόμβου στο δέντρο των καταστάσεων.

Αντίστοιχα, οι κινήσεις για τον μαύρο παίκτη επιλέγονται μέσω της σχέσης:

$$m^n = argmin_n \left[\begin{cases} -\infty, N_b = 0 \\ \frac{V_a}{N_b} - c_1 P(a,b) \sqrt{\frac{log(N_a)}{N_b}}, N_b > 0 \end{cases} \right]$$

Η επιλογή γίνεται με αντίστοιχο τρόπο, μόνο που αυτή τη φορά επιλέγεται η μικρότερη τιμή. Αυτή η αλλαγή γίνεται επειδή το μοντέλο αξιολόγησης επιστρέφει αρνητικές τιμές για τις καλές θέσεις του μαύρου παίκτη, οπότε πρέπει να επιλεχτούν οι θέσεις με όσο μικρότερη αξιολόγηση είναι δυνατή. Για αυτό και γίνεται η αλλαγή πρόσημου στο εσωτερικό της σχέσης αφού ισχύει η σχέση:

$$c_1 P(a,b) \sqrt{\frac{log(N_a)}{N_b}} > 0$$

επειδή $N_a \geq 1$ αφού ο κόμβος-γονέας έχει ήδη προστεθεί στο δέντρο αναζήτησης, άρα έχει εξερευνηθεί από τον αλγόριθμο τουλάχιστον μία φορά, και αντίστοιχα c_1, $P(a,b) > 0$.

Αν δεν χρησιμοποιείται το μοντέλο πρόβλεψης πιθανοτήτων, οι σχέσεις για τον λευκό και τον μαύρο παίκτη γίνονται αντίστοιχα:

$$m^n = argmax_n \left[\begin{cases} +\infty, N_b = 0 \\ \frac{V_a}{N_b} + c_2 \sqrt{\frac{log(N_a)}{N_b}}, N_b > 0 \end{cases} \right]$$

$$m^n = argmin_n \left[\begin{cases} -\infty\,, N_b = 0 \\ \frac{V_a}{N_b} - c_2\sqrt{\frac{log(N_a)}{N_b}}\,, N_b > 0 \end{cases} \right]$$

Σε αυτή την περίπτωση η σταθερά c_2 παίρνει μια μικρότερη συνήθως τιμή, αλλά φυσικά αυτό εξαρτάται και από το πρόβλημα, από το πεδίο τιμών της συνάρτησης αξιολόγησης, αλλά και την επιθυμητή ισορροπία εξερεύνησης-εκμετάλλευσης κάθε φορά. Ο αριθμός που προτείνεται τις περισσότερες φορές σε πολλές πηγές είναι το $\sqrt{2}$.

Σε κάθε προσομοίωση του αλγορίθμου, επιλέγεται ένας κόμβος του δέντρου ανά εξερευνημένο επίπεδο από την παραλλαγή της UCB1 μεθόδου. Όταν φταστεί το τελευταίο επίπεδο που έχει εξερευνηθεί, ο κόμβος που επιλέγεται προστίθεται στο μονοπάτι εξερεύνησης, καθώς και όλοι οι κόμβοι-παιδιά του κόμβου αυτού, ώστε να αξιολογηθούν στην επόμενη προσομοίωση.

Ακολουθεί η διαδικασία του rollout ή αλλιώς της προσομοίωσης του υπόλοιπου παιχνιδιού. Παραδοσιακά, σε αυτή την διαδικασία, εκτυλίσσεται το υπόλοιπο παιχνίδι με τυχαίες κινήσεις μέχρι να βρεθεί ένα τελικό αποτέλεσμα το οποίο αποθηκεύεται με τη μορφή της οπισθοδιάδοσης στο δέντρο. Οι κόμβοι που συναντιούνται κατά τις τυχαίες κινήσεις δεν αποθηκεύονται στο δέντρο. Συγκεκριμένα για το σκάκι, όμως, επειδή μια τέτοια διαδικασία θα μπορούσε να έχει πολύ μεγάλη διάρκεια, λόγω των πολλών πιθανών κινήσεων μέχρι το τέλος ενός παιχνιδιού, αυτή η διαδικασία παρακάμπτεται ή γίνεται για μικρό αριθμό προσομοιώσεων (ανάλογα με την μηχανή, θα εξηγηθεί στη συνέχεια) και αντικαθίσταται από το μοντέλο αξιολόγησης θέσης. Αντί να παιχτεί με τυχαίο τρόπο το παιχνίδι μέχρι το τέλος, αξιολογείται η θέση που η αναζήτηση έχει φτάσει και η αξιολόγηση διαδίδεται στο υπόλοιπο μονοπάτι. Εναλλακτικά, είναι δυνατό να γίνει η προσομοίωση, όχι με τυχαίο τρόπο, αλλά χρησιμοποιώντας την καλύτερη κίνηση βάσει του μοντέλου αξιολόγησης κάθε φορά, αλλά αυτά η μέθοδος είναι πολύ χρονοβόρα και καταλήγει να έχει χειρότερη απόδοση λόγω των πολλών φορών που απαιτείται να χρησιμοποιείται το μοντέλο.

Όταν προκύψει η αξιολόγηση από το μοντέλο, διαδίδεται σε όλους τους κόμβους του μονοπατιού της συγκεκριμένης προσομοίωσης και μία νέα προσομοίωση ξεκινάει. Επειδή το συγκεκριμένο μοντέλο έχει μικρό πεδίο τιμών, συγκεκριμένα το [0,1], η αξιολόγηση πολλαπλασιάζεται με μία σταθερά c_3, για να αυξηθεί η σημαντικότητά της και να έχει μεγαλύτερη επιρροή στη μέθοδο UCB1. Σε κάθε οπισθοδιάδοση, οι κόμβοι που αποτελούν το μονοπάτι ενημερώνονται με τον εξής τρόπο. Έστω κόμβος a, μέρος του μονοπατιού της προσομοίωσης. Το σύνολο αξιολόγησης V και το σύνολο επισκέψεων N ενημερώνονται σύμφωνα με τις σχέσεις:

$$V_a = V_a + c_3 * EM(a)$$

$$N_a = N_a + 1$$

όπου EM το μοντέλο αξιολόγησης θέσης (evaluation model) που επιστρέφει μια αξιολόγηση για τη θέση a. Η σταθερά c_3 τέθηκε ίση με 22,5, μετά από πειραματισμούς. Το μέγεθός της εξαρτάται από το πόσο επιθετική στρατηγική θέλει κάποιος να δώσει στην σκακιστική μηχανή. Μικρές τιμές της σταθεράς ευνοούν την εξερεύνηση του αλγορίθμου, ενώ μεγάλες τιμές ευνοούν την επιθετική τακτική διότι η άμεση αξιολόγηση του μοντέλου θα επιστρέψει καλή αξιολόγηση για ευθεία επίθεση σε πεσσό του αντιπάλου και ίσως όχι τόσο καλή για αργή και οργανωμένη επίθεση.

Όταν τελειώσει το χρονικό όριο που έχει δοθεί στον αλγόριθμο για να εκτελέσει τις προσομοιώσεις, πρέπει να παρθεί μια απόφαση για το ποια κίνηση τελικά θα επιλεγεί να γίνει από την αρχική θέση. Από τις πιθανές κινήσεις, επιλέγεται αυτή που ο αντίστοιχος κόμβος της έχει δεχθεί τις περισσότερες επισκέψεις από τον αλγόριθμο, δηλαδή ο κόμβος που συμμετείχε στις περισσότερες προσομοιώσεις. Εναλλακτικά, μπορεί να επιλεχθεί ο κόμβος με τον καλύτερο μέσο όρο αξιολόγησης ανά επίσκεψη, μέθοδος που επίσης παρουσιάζει καλά αποτελέσματα. Με αυτόν τον τρόπο προκύπτει κάθε φορά, η επόμενη κίνηση της μηχανής.

4.6 Πειραματισμοί και βελτίωση απόδοσης

Αφού δημιουργήθηκε η βασική δομή του αλγορίθμου και εκπαιδεύτηκαν σε επαρκές σημείο τα μοντέλα αξιολόγησης, ακολούθησε μια σειρά πειραμάτων για να δοκιμαστούν διάφορες μέθοδοι που αφορούν κυρίως την αναζήτηση δέντρου Monte Carlo και την ρύθμιση των παραμέτρων που την αφορούν, για τις οποίες δεν υπάρχει θεωρητικό υπόβαθρο, κυρίως εξαρτώνται από το πρόβλημα και τις απαιτήσεις της υλοποίησης.

4.6.1 Βελτίωση απόδοσης με πειράματα στον αλγόριθμο MCTS

Αρχικά, δοκιμάστηκε η απόδοση του 3ου βήματος της αναζήτησης δέντρου Monte Carlo, αυτό της προσομοίωσης. Συγκεκριμένα, δοκιμάστηκε η χρησιμοποίηση της μεθόδου, κάνοντας κάποιες τυχαίες κινήσεις και στην συνέχεια χρησιμοποιώντας το μοντέλο αξιολόγησης έναντι της ίδιας μηχανής που χρησιμοποιούσε, όμως, απευθείας το μοντέλο αξιολόγησης παρακάμπτοντας την διαδικασία της

προσομοίωσης. Παρατηρήθηκε ότι η μηχανή που δεν πραγματοποιούσε τις τυχαίες κινήσεις απέδιδε καλύτερα, οπότε και το βήμα της προσομοίωσης παρακάμφθηκε. Αυτό δεν θα μπορούσε να είναι δυνατό χωρίς το μοντέλο ή κάποια συνάρτηση αξιολόγησης, που επιτρέπει την αξιολόγηση μιας θέσης, χωρίς την ανάγκη προσομοίωσης της παρτίδας μέχρι να υπάρχει τελικό αποτέλεσμα.

Αντίστοιχα, δοκιμάστηκε η χρήση του 3ου βήματος χωρίς, όμως, τυχαίες κινήσεις, αλλά χρησιμοποιώντας το μοντέλο αξιολόγησης για τη διαδικασία της προσομοίωσης και τις κινήσεις που γίνονται. Η μέθοδος αυτή, ενώ είναι πιο αποδοτική από την τυχαία προσομοίωση, είναι πολύ πιο χρονοβόρα, λόγω των πολλών χρήσεων του μοντέλου αξιολόγησης, οπότε δεν ενδείκνυται η χρήση της.

Πειραματικά ρυθμίστηκε και η παράμετρος c_3 του αλγορίθμου, ο συντελεστής της αξιολόγησης του μοντέλου, δηλαδή. Μηχανές με όμοιο μοντέλο αξιολόγησης τέθηκαν αντιμέτωπες μεταξύ τους με το αποτέλεσμα να συστήνει μικρές θετικές τιμές για τον συντελεστή όταν χρησιμοποιείται το μοντέλο εύρεσης πιθανοτήτων, ο οποίος τέθηκε ίσος με 10,0 για λειτουργία της μηχανής χωρίς το μοντέλο πρόβλεψης πιθανοτήτων και 5,0 για λειτουργία με το μοντέλο.

Με τον ίδιο τρόπο ρυθμίστηκε και η παράμετρος c_1. Όμοιες μηχανές με διαφορετικούς συντελεστές και ίδια μοντέλα τέθηκαν αντιμέτωπες μεταξύ τους. Τα καλύτερα αποτελέσματα παρουσιάστηκαν ξανά για μικρές θετικές τιμές. Η σταθερά τέθηκε ίση με 1,5.

Επιπλέον, στο 2ο βήμα του αλγορίθμου, αυτό της επέκτασης, δοκιμάστηκαν δύο εκδόσεις της μεθόδου, αυτή της μονής επέκτασης τυχαίου κόμβου-παιδιού και αυτή της επέκτασης σε όλους τους κόμβους-παιδιά του τελευταίου κόμβου που επιλέχτηκε. Παρόλο που στην περισσότερη βιβλιογραφία προτείνεται η πρώτη έκδοση, η δεύτερη απέδωσε καλύτερα και υιοθετήθηκε. Σε αυτό πιθανώς συνέβαλε το μοντέλο πιθανοτήτων κινήσεων, το οποίο αποδίδει καλύτερα αν είναι γνωστοί όλοι οι κόμβοι-παιδιά μιας θέσης.

Επίσης, δημιουργήθηκε και μια νέα έκδοση του αλγορίθμου αναζήτησης που, με την χρήση της βιβλιοθήκης pandas, αποθήκευε τα στατιστικά των κόμβων και τα επαναχρησιμοποιούσε την επόμενη φορά που επρόκειτο ο παίκτης να παίξει. Με αυτόν τον τρόπο, το δέντρο δεν δομούνταν από την αρχή κάθε φορά, αλλά οι προσομοιώσεις γινόταν με τη βοήθεια των υπαρχόντων στατιστικών. Παρόλο που και αυτή η μέθοδος παρουσίαζε καλή απόδοση, μετά από έναν αριθμό κινήσεων το δέντρο του παιχνιδιού γινόταν πάρα πολύ μεγάλο, με αποτέλεσμα ο αλγόριθμος να καθυστερεί πάρα πολύ, για αυτό και τελικά δεν χρησιμοποιείται από την τελική έκδοση της μηχανής.

Επιπλέον, παρατηρήθηκε ότι η μηχανή ενισχυόμενης μάθησης απέδιδε καλύτερα χωρίς τη χρησιμοποίηση του μοντέλου πιθανοτήτων, οπότε και το μοντέλο αυτό δεν χρησιμοποιήθηκε για αυτή την μηχανή. Τα υπόλοιπα βήματα του αλγορίθμου παραμένουν ίδια. Ο λόγος της χειρότερης απόδοσης είναι η έλλειψη επαρκούς εκπαίδευσης στην μηχανή ενισχυτικής μάθησης λόγω του πολύ αργού ρυθμού που αυτή είχε, αλλά και λόγω της απουσίας της MCTS στην διαδικασία αυτό-εκπαίδευσης.

4.6.2 Βελτίωση απόδοσης χρησιμοποιώντας κάρτα γραφικών

Βασικό ζήτημα του αλγορίθμου είναι η ταχύτητα. Η αναζήτηση δέντρου Monte Carlo θα πρέπει να ολοκληρώνει όσες περισσότερες προσομοιώσεις μπορεί σε όσο το λιγότερο δυνατό χρονικό διάστημα. Η πιο χρονοβόρα λειτουργία του αλγορίθμου είναι η κλήση των μοντέλων νευρωνικών δικτύων και συγκεκριμένα του μοντέλου αξιολόγησης και του μοντέλου πρόβλεψης πιθανοτήτων κινήσεων. Από άποψη κώδικα, ο αλγόριθμος είναι βελτιστοποιημένος ως προς τα μοντέλα, έτσι ώστε να καλείται μόνο μια φορά σε κάθε προσομοίωση το κάθε μοντέλο, που είναι ο μικρότερος δυνατός αριθμός χρήσης τους, σε σχέση με άλλες μεθόδους. Παρόλα αυτά, η χρήση των μοντέλων εξακολουθεί να απαιτεί αρκετό χρόνο, σχετικά με τα υπόλοιπα βήματα.

Για αυτόν τον λόγο, αξιοποιήθηκε κάρτα γραφικής επεξεργασίας (GPU) , για την επιτάχυνση της λειτουργίας των μοντέλων. Για την χρησιμοποίηση της κάρτας γραφικής επεξεργασίας, χρησιμοποιήθηκε η βιβλιοθήκη tensorflow-gpu της Python. Συγκεκριμένα, χρησιμοποιείται η κάρτα γραφικής επεξεργασίας NVIDIA GeForce GTX 1660 Ti με Max-Q Design, η οποία υποστηρίζει CUDA (Compute Unified Device Architecture) σχεδίαση, μια πλατφόρμα παράλληλου προγραμματισμού σχεδιασμένη από την Nvidia [26]. Η πλατφόρμα CUDA είναι ένα επίπεδο λογισμικού που δίνει απευθείας πρόσβαση στο εικονικό σύνολο εντολών και στα στοιχεία παράλληλου προγραμματισμού της κάρτας γραφικής επεξεργασίας, για την εκτέλεση πυρήνων υπολογισμού [27]. Επίσης, χρησιμοποιείται η βιβλιοθήκη cuDNN (Deep Neural Network library) της Nvidia, μια βιβλιοθήκη που η λειτουργία της επιταχύνεται από την κάρτα γραφικής επεξεργασίας σχεδιασμένη για βαθιά νευρωνικά δίκτυα. Η βιβλιοθήκη προσφέρει λεπτομερώς ρυθμισμένες υλοποιήσεις για συνήθεις ρουτίνες όπως η συνέλιξη, το pooling, η κανονικοποίηση και τα επίπεδα ενεργοποίησης [28]. Η έκδοση της βιβλιοθήκης cuDNN που χρησιμοποιήθηκε είναι η νεότερη (8.1) και αποκτήθηκε από την Nvidia για εκπαιδευτικούς λόγους.

Η βιβλιοθήκη tensorflow-gpu αξιοποιεί τον σχεδιασμό αυτόν και την βιβλιοθήκη cuDNN, με αποτέλεσμα να παρουσιάζει γρηγορότερα αποτελέσματα στην εκπαίδευση αλλά και στην πρόβλεψη. Η χρήση της κάρτας γραφικής επεξεργασίας προσφέρει γρηγορότερα αποτελέσματα από τα δύο μοντέλα στον αλγόριθμο αναζήτησης δέντρου Monte Carlo. Συγκεκριμένα, παρατηρήθηκε αύξηση 39,3% των προσομοιώσεων του αλγορίθμου, σε δεδομένο χρονικό διάστημα, σε σχέση με την εκτέλεση του χρησιμοποιώντας μόνο την μονάδα κεντρικής επεξεργασίας (CPU). Η επιτάχυνση κατά σχεδόν 40% του αλγορίθμου καθιστά πολύ μεγάλη βελτίωση της απόδοσης των σκακιστικών μηχανών και αποδεικνύει την αναγκαιότητα τέτοιας τεχνολογίας σε προβλήματα όπου χρησιμοποιούνται μοντέλα νευρωνικών δικτύων. Η αναγκαιότητα αυτή επιβεβαιώνεται και από την ομάδα Deep Mind της Google, η οποία, κατά την εκπαίδευση της μηχανής AlphaZero, χρησιμοποίησε 5.000 TPUs πρώτης γενιάς (Tensor Processing Units [29]) για την εκτέλεση των παιχνιδιών της μηχανής κατά τα οποία αντιμετωπίζει τον εαυτό της, χρησιμοποιώντας την αναζήτηση δέντρου Monte Carlo, και 64 TPUs δεύτερης γενιάς για την εκπαίδευση των νευρωνικών δικτύων [9].

4.7 Αξιολόγηση απόδοσης σκακιστικών μηχανών

Αφού, ρυθμίστηκαν οι παράμετροι των μηχανών πειραματικά, βελτιστοποιήθηκε ο αλγόριθμος και έγινε επαρκής εκπαίδευση των μοντέλων, ακολούθησε το στάδιο της αξιολόγησης της απόδοσης τους. Η αξιολόγηση των μοντέλων έγινε με το μοντέλο ενισχυτικής μάθησης να έχει εκπαιδευτεί 8.914 παιχνίδια εναντίον του εαυτού του, το μοντέλο επιβλεπόμενης μάθησης με 233.229 επαγγελματικά παιχνίδια που βρέθηκαν στο διαδίκτυο και το συνδυαστικό μοντέλο με 85.081 παιχνίδια που προήλθαν και από τους δύο παραπάνω τρόπους.

Για την μέτρηση της απόδοσής τους, οι μηχανές τέθηκαν αντιμέτωπες με εκδόσεις τις μηχανής Stockfish, μέσω της ιστοσελίδας lichess.org [33]. Ο λόγος που επιλέχτηκε η συγκεκριμένη ιστοσελίδα είναι επειδή παρέχει τη δυνατότητα για παιχνίδι έναντι πολλών διαφορετικών εκδόσεων της μηχανής Stockfish, με διαβάθμιση στην ισχύ τους, σε ένα φιλικό προς τον χρήστη περιβάλλον. Τα παιχνίδια ήταν απεριόριστου χρόνου και στις σκακιστικές μηχανές δόθηκε χρόνος 15 δευτερολέπτων για να επιστρέψουν μια κίνηση.

Αρχικά, οι τρεις σκακιστικές μηχανές τέθηκαν αντιμέτωπες με την πιο αδύναμη έκδοση της μηχανής Stockfish, με βαθμολογία 800 ELO. Και οι 3 κέρδισαν όλα τα παιχνίδια που έπαιξαν. Έπειτα, ακολούθησαν παρτίδες με την αμέσως ισχυρότερη έκδοση, με βαθμολογία 1100 ELO. Η σκακιστική μηχανή που εκπαιδεύτηκε με δεδομένα πραγματικών αγώνων και η συνδυαστική σκακιστική μηχανή κέρδισαν με ευκολία τα περισσότερα παιχνίδια, με λίγες μόνο ισοπαλίες. Η σκακιστική μηχανή που εκπαιδεύτηκε με ενισχυτική μάθηση είχε μοιρασμένο σκορ, κερδίζοντας και φέρνοντας ισοπαλία σε αρκετές παρτίδες, αλλά και χάνοντας μερικές. Στην συνέχεια, έπαιξαν εναντίον της έκδοσης με βαθμολογία 1400 ELO. Η σκακιστική μηχανή ενισχυτικής μάθησης έχασε τα περισσότερα παιχνίδια, φέρνοντας λίγες ισοπαλίες. Οι υπόλοιπες δύο είχαν μοιρασμένο σκορ, φέρνοντας και νίκες και ήττες και ισοπαλίες, με την μηχανή εποπτευόμενης να αποδίδει λίγο καλύτερα από την συνδυαστική σκακιστική μηχανή. Τέλος, οι τρεις σκακιστικές μηχανές αντιμετώπισαν την έκδοση της μηχανής Stockfish, βαθμολογημένης στα 1700 ELO, χάνοντας όλα τα παιχνίδια.

Με βάση την απόδοση σε αυτά τα παιχνίδια, η βαθμολογία κάθε μηχανής υπολογίζεται προσεγγιστικά όπως φαίνεται στον παρακάτω πίνακα.

Σκακιστική μηχανή ενισχυτικής μάθησης	Σκακιστική μηχανή επιβλεπόμενης μάθησης	Σκακιστική μηχανή συνδυαστικής μάθησης
1100 – 1200 ELO	1350 – 1450 ELO	1300 -1400 ELO

Σχήμα 4.3: Προσεγγιστική αξιολόγηση σκακιστικών μηχανών

Τα αποτελέσματα είναι αρκετά ικανοποιητικά, δεδομένου ότι πρόκειται για σκακιστικές μηχανές ανεπτυγμένες σε μια πιο αργή γλώσσα προγραμματισμού υψηλού επιπέδου, όπως η Python, και δεδομένου ότι δεν ακολουθήθηκε η παραδοσιακή μέθοδος ανάπτυξης σκακιστικής μηχανής αλλά επιλέχθηκε η λιγότερο δοκιμασμένη μέθοδος της μηχανικής μάθησης.

Η χειρότερη απόδοση της σκακιστικής μηχανής ενισχυτικής μάθησης οφείλεται στα λιγότερα παιχνίδια εκπαίδευσης που είχε, λόγω της πολύ πιο αργής διαδικασίας εκπαίδευσης, μιας που η μηχανή έπρεπε και να παράγει τα παιχνίδια και να τα χρησιμοποιεί για εκπαίδευση. Επίσης, λόγω έλλειψης των απαραίτητων πόρων, η παραγωγή των παιχνιδιών δεν έγινε χρησιμοποιώντας την αναζήτηση δέντρου Monte Carlo, καθώς θα έπαιρνε πολύ περισσότερο χρόνο, αλλά χρησιμοποιήθηκε μόνο η αξιολόγηση του μοντέλου αξιολόγησης για να γίνουν οι κινήσεις. Αυτό είχε ως αποτέλεσμα την παραγωγή λιγότερο ποιοτικών κινήσεων, οπότε και λιγότερο ποιοτική εκπαίδευση των μοντέλων. Για τον ίδιο λόγο, δεν μπορούσε να εκπαιδευτεί σωστά και το μοντέλο πρόβλεψης πιθανοτήτων για την σκακιστική μηχανή ενισχυτικής μάθησης.

Επίσης, η αξιολόγηση αυτή έγινε σε έναν προσωπικό υπολογιστή στο λειτουργικό σύστημα Windows. Οι σκακιστικές μηχανές σε κάποιο άλλο λειτουργικό ή αν υπάρχει η υποστήριξη καλύτερου hardware, με περισσότερες GPUs ή TPUs, σίγουρα πρόκειται να παρουσιάσουν πολύ καλύτερη απόδοση. Με τη βοήθεια μιας μόνο κάρτας γραφικής επεξεργασίας που ο προσωπικός υπολογιστής διαθέτει, η απόδοση των μηχανών αυξήθηκε κατά 39%, που σημαίνει ότι με περισσότερη βοήθεια από hardware, θα υπήρχε πολύ μεγαλύτερη βελτίωση στην απόδοση.

4.8 Μελλοντικές βελτιώσεις σκακιστικών μηχανών

Παρά τη σχετικά καλή απόδοση, υπάρχουν πολλές βελτιώσεις που μπορούν να γίνουν μελλοντικά για να ενισχυθούν οι σκακιστικές μηχανές. Αρχικά, μπορεί να επιχειρηθεί πιο αποδοτική εκπαίδευση της μηχανής ενισχυτικής μάθησης, αξιοποιώντας τον αλγόριθμο αναζήτησης δέντρου Monte Carlo στην επιλογή κινήσεων. Για να γίνει κάτι τέτοιο, θα πρέπει οι σκακιστικές μηχανές να λειτουργήσουν σε πιο ισχυρό hardware, που θα επιτρέπει γρηγορότερες παρτίδες, άρα και γρηγορότερη εκπαίδευση. Γενικότερα, η λειτουργία σε ισχυρότερο hardware που διαθέτει περισσότερες κάρτες γραφικής επεξεργασίας ή TPUs θα αύξανε πολύ την απόδοση των μηχανών σε δεδομένο χρονικό διάστημα.

Ένας άλλος τρόπος βελτίωσης είναι η μεταφορά κομματιών του κώδικα της λειτουργίας των σκακιστικών μηχανών σε γλώσσες προγραμματισμού χαμηλότερου επιπέδου. Προς το παρόν, οι σκακιστικές μηχανές λειτουργούν αποκλειστικά στην Python γλώσσα, η οποία, παρόλο που παρέχει πολλές ευκολίες ως προς τη μηχανική μάθηση, είναι πολύ πιο αργή από άλλες γλώσσες χαμηλότερου επιπέδου. Θεωρητικά, όσο το δυνατόν περισσότερα μέρη της μηχανής μεταφερθούν σε γλώσσα χαμηλότερου επιπέδου, τόσο γρηγορότερη, άρα και πιο αποδοτική θα γίνει.

Επιπλέον, στο μέλλον μπορούν να γίνουν πολλές βελτιώσεις όσον αφορά τα νευρωνικά δίκτυα. Μπορεί να γίνει περισσότερος πειραματισμός ως προς τη δομή των νευρωνικών δικτύων και των διάφορων παραμέτρων που τα αποτελούν, ίσως αξιοποιώντας και τη βοήθεια κάποιου άλλου API πέρα από το Keras API που χρησιμοποιήθηκε μόνο του έως αυτό το σημείο.

Επίσης, στόχος είναι να εκπαιδεύονται συνεχώς στη συνέχεια οι μηχανές με νέα δεδομένα. Ιδίως η μηχανή ενισχυτικής μάθησης απαιτεί περισσότερα δεδομένα εκπαίδευσης στο μέλλον, αλλά και με καλύτερη διαδικασία αποφάσεων όπως εξηγήθηκε.

5 Υλοποίηση διαδικτυακής εφαρμογής σκακιού

Εκτός από τις σκακιστικές μηχανές, επιλέχτηκε να δημιουργηθεί και μια διεπαφή που θα φιλοξενεί τις μηχανές, ένα γραφικό περιβάλλον, δηλαδή, όπου ένας χρήστης μπορεί να αντιμετωπίσει τις μηχανές αλλά και να βλέπει τις μηχανές να τίθενται αντιμέτωπες μεταξύ τους. Σκοπός της υλοποίησης ήταν η δημιουργία ενός εύκολου για τον χρήστη περιβάλλον, το οποίο δεν θα απαιτεί εγκατάσταση ή αποθήκευση από τον χρήστη, παρά μόνο σύνδεση στο διαδίκτυο. Λόγω έλλειψης απαραίτητων πόρων, η εφαρμογή δεν είναι διαθέσιμη ακόμα στο διαδίκτυο, αλλά είναι πλήρως υλοποιημένη και λειτουργική.

5.1 Τεχνολογίες υλοποίησης εφαρμογής

Η εφαρμογή αποτελείται από το back-end , δηλαδή την μεταφορά πληροφοριών από και προς τον χρήστη, και το front-end, δηλαδή την γραφική διεπαφή με την οποία ο χρήστης αλληλοεπιδρά.

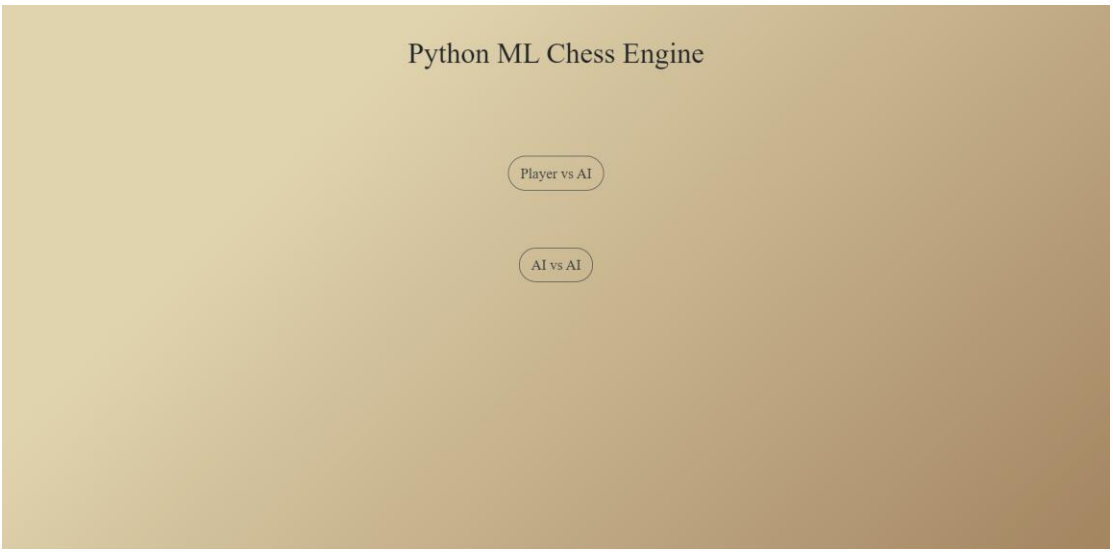
Το back-end δημιουργήθηκε με Python, ώστε να επικοινωνεί χωρίς προβλήματα με τις σκακιστικές μηχανές. Η επικοινωνία με το front-end επιτυγχάνεται με την χρήση της βιβλιοθήκης Flask της Python. Η Flask αποτελεί web framework της Python που διευκολύνει τη δημιουργία πλήρως δομημένων διαδικτυακών εφαρμογών [30]. Μέσω της Flask γίνεται η ανταλλαγή δεδομένων με το front-end, εκτελείται η κατάλληλη δρομολόγηση των σελίδων και καλούνται οι σκακιστικές μηχανές όποτε χρειάζεται. Η ανταλλαγή πληροφοριών γίνεται μέσω της μεθόδου “request” της βιβλιοθήκης. Συγκεκριμένα, χρησιμοποιείται η μέθοδος request ‘GET’ με την οποία αποστέλλονται μηνύματα στον δρομολογητή και ο δρομολογητής επιστρέφει πληροφορίες και η μέθοδος request ‘POST’ με την οποία αποστέλλονται HTML δεδομένα με την μορφή φόρμας στον δρομολογητή.

Το front-end, δημιουργήθηκε με HTML5, CSS3 και JavaScript. Αποτελείται από μια σειρά σελίδων που επιτρέπουν στον χρήστη να αλληλοεπιδράσει με τις σκακιστικές μηχανές, είτε να παίξει εναντίον τους, είτε να τις δει να παίζουν μεταξύ τους. Επίσης, χρησιμοποιήθηκε το toolkit Bootstrap, ένα χρήσιμο framework για front-end εργασίες που διευκολύνει και επιταχύνει την ανάπτυξη εφαρμογών στο διαδίκτυο [31].

5.2 Λειτουργίες εφαρμογής

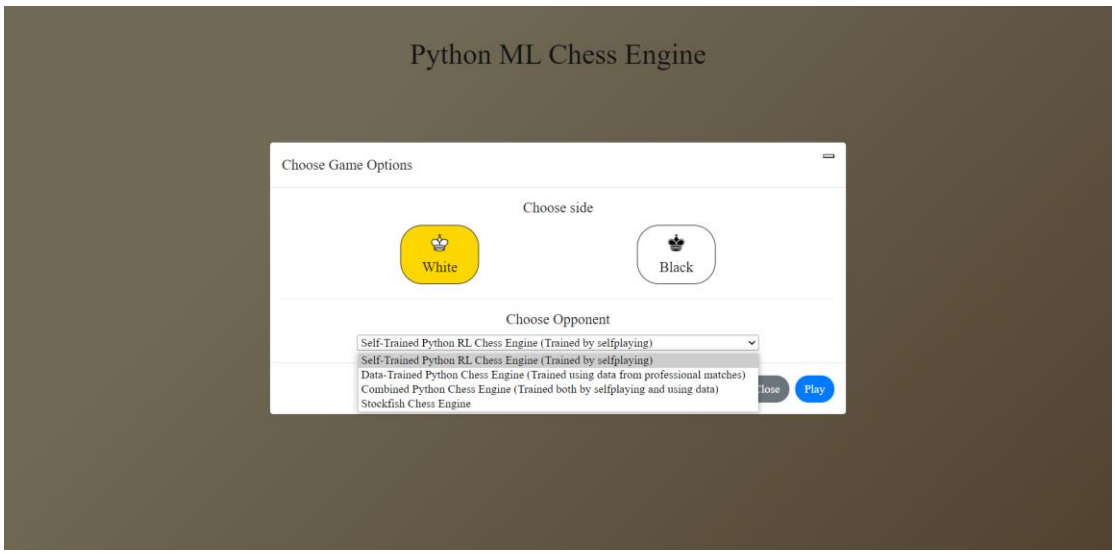
Η διαδικτυακή εφαρμογή έχει ως σκοπό της την εύκολη πρόσβαση του χρήστη στις σκακιστικές μηχανές που δημιουργήθηκαν. Σχεδιάστηκε ώστε να δίνει δύο επιλογές στον χρήστη, να παίξει αντιμέτωπος με τις μηχανές, ή να τις δει να αναμετρούνται μεταξύ τους.

Η αρχική σελίδα της εφαρμογής δίνει στον χρήστη ακριβώς αυτές τις δύο επιλογές, στις οποίες έχει πρόσβαση με το πάτημα του αντίστοιχου κουμπιού.



Σχήμα 5.1: Αρχική σελίδα εφαρμογής

Αν ο χρήστης επιλέξει την πρώτη επιλογή, αυτή του παιχνιδιού εναντίων των σκακιστικών μηχανών, του παρουσιάζονται κάποιες επιλογές για το παιχνίδι, όπως φαίνεται στο σχήμα 5.2. Ο χρήστης καλείται να επιλέξει το χρώμα του στο παιχνίδι και την αντίπαλη μηχανή, ανάμεσα στην σκακιστική μηχανή ενισχυμένης μάθησης, στην σκακιστική μηχανή που εκπαιδεύτηκε με δεδομένα πραγματικών αγώνων, στην συνδυαστική σκακιστική μηχανή που εκπαιδεύτηκε και με τους δύο τρόπους, αλλά και στην διάσημη σκακιστική μηχανή Stockfish, που φορτώθηκε στην σελίδα.



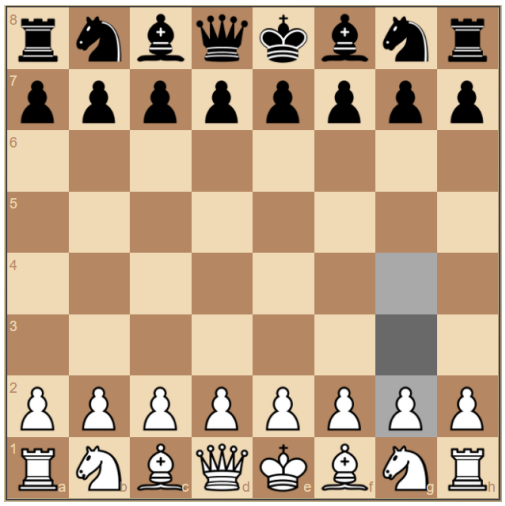
Σχήμα 5.2: Επιλογή χρώματος και αντιπάλου για το παιχνίδι του χρήστη έναντι της σκακιστικής μηχανής

Αφού γίνουν αυτές οι επιλογές από τον χρήστη και με το πάτημα του κουμπιού “Play”, ο χρήστης μεταφέρεται στη σελίδα παιχνιδιού, που φαίνεται στο σχήμα 5.3. Η σελίδα αποτελείται από την σκακιέρα και το πλαίσιο στα δεξιά που περιέχει τα ονόματα των παικτών, το χρονόμετρο του κάθε παίκτη και το ιστορικό των κινήσεων. Το παιχνίδι που παίζεται είναι παιχνίδι 10 λεπτών με 10 δευτερόλεπτα bonus για κάθε παίκτη, κάθε φορά που ολοκληρώνει μια κίνηση. Ο ένας από τους δυο παίκτες κερδίζει αν οδηγήσει το παιχνίδι σε ρουά ματ υπέρ του ή τελειώσει ο χρόνος του αντιπάλου. Επιπλέον, δίνεται η επιλογή για επανεκκίνηση του παιχνιδιού μέσω του κουμπιού “Reset Game” και η επιλογή τερματισμού του παιχνιδιού με παραίτηση και ήττα του χρήστη μέσω του κουμπιού “Surrender”.

Η σκακιέρα δημιουργήθηκε με τη βοήθεια του API chessboard.js, που προσφέρει την σκακιέρα, τα πιόνια και κάποιες έτοιμες λειτουργίες που μπορούν να ρυθμιστούν ανάλογα με τις απαιτήσεις της εργασίας. Οι επιλογές του χρήστη πάνω στην σκακιέρα στέλνονται στο back-end για να επιβεβαιωθεί η εγκυρότητά τους και επιστρέφεται το αντίστοιχο μήνυμα, η νέα θέση της σκακιέρας και το αν το παιχνίδι έχει τελειώσει ή όχι. Η νέα θέση που στέλνεται από την Python φορτώνεται στην σκακιέρα κάθε φορά και το παιχνίδι συνεχίζει [32].

Σχήμα 5.3: Σελίδα παιχνιδιού του παίκτη έναντι της σκακιστικής μηχανής της επιλογής του

Το παιχνίδι παίζεται με το στυλ ‘drag and drop’, δηλαδή ο χρήστης επιλέγει έναν πεσσό και τον ‘σέρνει’ στο τετράγωνο που θέλει να καταλήξει. Αν η κίνηση είναι έγκυρη, εγκρίνεται και το παιχνίδι συνεχίζει, αλλιώς απορρίπτεται και ο πεσσός επιστρέφει αυτόματα στην προηγούμενή του θέση. Οι επιτρεπτές κινήσεις κάθε πεσσού εμφανίζουν γκρι χρώμα όταν ο χρήστης περνάει το ποντίκι του πάνω από τον αντίστοιχο πεσσό, ώστε ο χρήστης να κατανοεί καλύτερα ανάμεσα σε ποιες κινήσεις έχει να επιλέξει, όπως φαίνεται στο σχήμα 5.4.

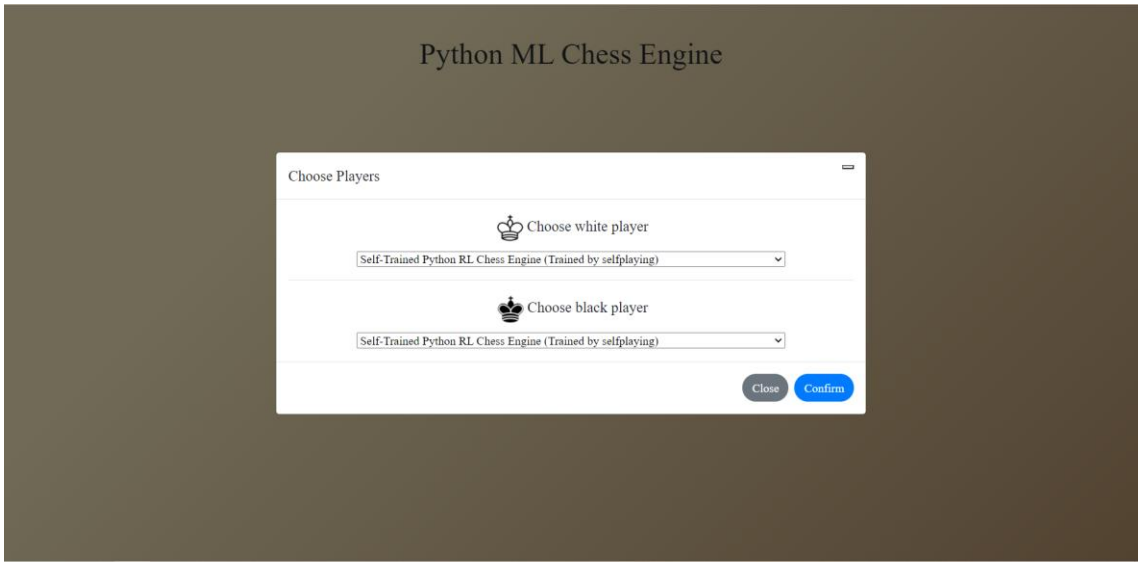


Σχήμα 5.4: Προβολή επιτρεπτών κινήσεων για κάθε πεσσό περνώντας το ποντίκι πάνω από το τετράγωνό του

Στο πλαίσιο δεξιά αναγράφονται οι δύο παίκτες, ενώ επίσης υπάρχει και ένα χρονόμετρο για τον καθένα, το οποίο μετράει αντίστροφα κάθε φορά που είναι η σειρά του αντίστοιχου παίκτη. Αν ο χρόνος ενός παίκτη τελειώσει, χάνει το παιχνίδι. Επίσης, καταγράφεται το ιστορικό όλων των κινήσεων του παιχνιδιού στο πλαίσιο “Move History”, καθώς και το τελικό αποτέλεσμα, όταν αυτό υπάρξει, στο κάτω μέρος του πλαισίου, όπως φαίνεται στο σχήμα 5.5.

Σχήμα 5.5: Ιστορικό κινήσεων και προβολή τελικού αποτελέσματος

Έτσι συνοψίζονται οι λειτουργίες της σελίδας παιχνιδιού του χρήστη εναντίον μιας από τις σκακιστικές μηχανές. Αντίστοιχα, αν ο χρήστης επιλέξει στην αρχική σελίδα το δεύτερο κουμπί, αυτό για το παιχνίδι μεταξύ των σκακιστικών μηχανών, εμφανίζεται ένα παράθυρο μέσω του οποίου ο χρήστης καλείται να επιλέξει ποιες σκακιστικές μηχανές θα αναμετρηθούν, όπως φαίνεται στο σχήμα 5.6.



Σχήμα 5.6: Επιλογή αντιπάλων για το παιχνίδι μεταξύ σκακιστικών μηχανών

Αφού ο χρήστης επιλέξει αντιπάλους και πατήσει το κουμπί “Play” μεταφέρεται στη σελίδα παιχνιδιού των μηχανών. Η σελίδα έχει αντίστοιχες λειτουργίες με την σελίδα παιχνιδιού εναντίον της μηχανής, μόνο που τώρα ο χρήστης δεν παίζει, αλλά παρακολουθεί τις σκακιστικές μηχανές να παίζουν. Το παιχνίδι εκκινεί με την επιλογή “Start Game”, και μπορεί να αρχικοποιηθεί με την επιλογή “Reset Game”. Στο πλαίσιο στα δεξιά εμφανίζονται ξανά οι αναμετρούμενες μηχανές, ο χρόνος που τους απομένει και το ιστορικό των κινήσεων.

Σχήμα 5.7: Σελίδα παιχνιδιού μεταξύ των σκακιστικών μηχανών

5.3 Μελλοντικές βελτιώσεις εφαρμογής

Κύριος στόχος για την εφαρμογή είναι μελλοντικά να φορτωθεί στον παγκόσμιο ιστό, ώστε όλοι οι χρήστες να έχουν πρόσβαση στις σκακιστικές μηχανές, να μπορούν να παίζουν αντιμέτωποι με αυτές αλλά και να τις βλέπουν να αναμετρούνται μεταξύ τους καθώς και με άλλες διάσημες μηχανές.

Επίσης, πρόκειται να γίνουν πολλές άλλες βελτιώσεις στην εφαρμογή, όπως η επιλογή του τρόπου παιχνιδιού, για παράδειγμα η επιλογή του αν θα είναι χρονομετρημένο το παιχνίδι και με πόσο χρόνο ή όχι. Επιπλέον, μπορεί να προστεθεί η δυνατότητα δημιουργίας λογαριασμού και καταγραφής στατιστικών μεταξύ του παίκτη και των μηχανών.

Η σχεδίαση της εφαρμογής αυτή τη στιγμή εξυπηρετεί ερευνητικούς σκοπούς περισσότερο. Για να γίνει πιο εύκολη προς χρήση πρόκειται να γίνουν περισσότερες αλλαγές στη σχεδίαση, με σκοπό τη δημιουργία ενός πιο φιλικού προς τον χρήστη περιβάλλον και μιας πιο διασκεδαστικής εμπειρίας.

Βιβλιογραφία - Πηγές

[1]:https://en.wikipedia.org/wiki/Chess
[2]:https://www.fide.com/
[3]:https://el.wikipedia.org/wiki/%CE%A3%CE%BA%CE%AC%CE%BA%CE%B9
[4]:https://www.chessprogramming.org/Protocols
[5]:https://en.wikipedia.org/wiki/Chess_engine
[6]:https://towardsdatascience.com/introduction-to-machine-learning-for-beginners-eed6024fdb08
[7]: Dive into Deep Learning, Aston Zhang, Zachary C. Lipton, Mu Li, and Alexander J. Smola
[8]:https://en.wikipedia.org/wiki/Artificial_neural_network
[9]: Silver, David; Huang, Aja; Maddison, Chris J.; Guez, Arthur; Sifre, Laurent; Driessche, George van den; Schrittwieser, Julian; Antonoglou, Ioannis; Panneershelvam, Veda; Lanctot, Marc; Dieleman, Sander; Grewe, Dominik; Nham, John; Kalchbrenner, Nal; Sutskever, Ilya; Lillicrap, Timothy; Leach, Madeleine; Kavukcuoglu, Koray; Graepel, Thore; Hassabis, Demis (28 January 2016). "Mastering the game of Go with deep neural networks and tree search". Nature. 529 (7587): 484–489. Bibcode:2016Natur.529..484S. doi:10.1038/nature16961. ISSN 0028-0836. PMID 26819042. S2CID 515925
[10]: Silver, David (2017). "Mastering Chess and Shogi by Self-Play with a General Reinforcement Learning Algorithm". arXiv:1712.01815v1
[11]: Jonathan Rubin; Ian Watson (April 2011). "Computer poker: A review" (PDF). Artificial Intelligence. 175 (5–6): 958–987. doi:10.1016/j.artint.2010.12.005. Archived from the original (PDF) on 2012-08-13
[12]: "Monte-Carlo Tree Search in TOTAL WAR: ROME II's Campaign AI". AI Game Dev. Archived from the original on 13 March 2017. Retrieved 25 February 2017.
[13]: Brügmann, Bernd (1993). Monte Carlo Go (PDF). Technical report, Department of Physics, Syracuse University.
[14]: Kocsis, Levente; Szepesvári, Csaba (2006). "Bandit based Monte-Carlo Planning". In Fürnkranz, Johannes; Scheffer, Tobias; Spiliopoulou, Myra (eds.). Machine Learning: ECML 2006, 17th European Conference on Machine Learning, Berlin, Germany, September 18–22, 2006, Proceedings. Lecture Notes in Computer Science. 4212. Springer. pp. 282–293. CiteSeerX 10.1.1.102.1296. doi:10.1007/11871842_29. ISBN 3-540-45375-X.
[15]: Bouzy, Bruno. "Old-fashioned Computer Go vs Monte-Carlo Go" (PDF). IEEE Symposium on Computational Intelligence and Games, April 1–5, 2007, Hilton Hawaiian Village, Honolulu, Hawaii
[16]: Althöfer, Ingo (2012). "On Board-Filling Games with Random-Turn Order and Monte Carlo Perfectness". Advances in Computer Games. Lecture Notes in Computer Science. 7168. pp. 258–269. doi:10.1007/978-3-642-31866-5_22. ISBN 978-3-642-31865-8.
[17]: python-chess: a chess library for Python — python-chess 1.6.1 documentation
[18]: https://en.wikipedia.org/wiki/Algebraic_notation_(chess)

[19]: https://en.wikipedia.org/wiki/Forsyth%E2%80%93Edwards_Notation
[20]: https://en.wikipedia.org/wiki/One-hot
[21]: Harris, David and Harris, Sarah (2012-08-07). Digital design and computer architecture (2nd ed.). San Francisco, Calif.: Morgan Kaufmann. p. 129. ISBN 978-0-12-394424-5.
[22]: https://keras.io/
[23]: https://www.ficsgames.org/
[24]: https://www.freechess.org/
[25]: https://en.wikipedia.org/wiki/Portable_Game_Notation
[26]: https://en.wikipedia.org/wiki/CUDA
[27]: Abi-Chahla, Fedy (June 18, 2008). "Nvidia's CUDA: The End of the CPU?". Tom's Hardware. Retrieved May 17, 2015.
[28]: https://developer.nvidia.com/cudnn
[29]: https://en.wikipedia.org/wiki/Tensor_Processing_Unit
[30]: https://flask.palletsprojects.com/en/2.0.x/
[31]: https://getbootstrap.com/
[32]: https://chessboardjs.com/
[33]: https://lichess.org/

Παράρτημα – Κώδικας υλοποίησης

Μέρος Α – Κώδικας σκακιστικών μηχανών

Αρχείο: board.py

```
import chess


class Board():

    def __init__(self):
        self.board = chess.Board()

    def move(self, uci_move):
        return
self.board.push_uci(uci_move)

    def turn_str(self, player):
        return "White" if player ==
chess.WHITE else "Black"

    def turn_int(self):
        return 1.0 if self.board.turn
== True else -1.0

    def turn_num(self):
        return
len(self.board.move_stack)

    def fen(self):
        return self.board.fen()

    def legal_moves(self):
        return list(move.uci() for move
in self.board.legal_moves)

    def legal_fens(self):

        current_fen = self.fen()
        fens = []
        moves = self.legal_moves()

        for i in range(len(moves)):
            self.move(moves[i])
            next_fen = self.fen()
            fens.append(next_fen)

self.reset_to_specific(current_fen)

        return fens

    def state_reward(self):

        result = None
        if self.board.is_checkmate():
            result = not
self.board.turn

        if result == True:
            return 1

        elif result == False:
            return -1

        else:
            return None

def reset_to_specific(self, fen):
     self.board = chess.Board(fen)

def reset(self):
    self.__init__()
```

Αρχείο: datatrain.py

```
import chess.pgn
import numpy as np
from board import Board
from fen_transformation import
fen_transform
from tensorflow.keras.models import
load_model
from move_transformation import
move_transform


def get_pgn_data(pgn, batch_size,
datagames, combined_games, parsed):

    training_set = []
    training_turn_set = []
    training_labels = []
    probability_fens = []
    probability_turns = []
    probability_labels = []
    eof = False

    for match in range(batch_size):
        print("Game:", match+1)
        game = chess.pgn.read_game(pgn)
        if game == None:
            eof = True
            print("End of file!!!")
            break
        parsed += 1

        game_copy = game
        board = game_copy.board()
        turn = 1.0
        for move in
game_copy.mainline_moves():
            fen_table =
fen_transform(board.fen())


probability_fens.append(fen_table)

probability_turns.append(turn)

probability_labels.append(move_transfor
m(str(move)))
            board.push(move)
            turn = -1.0*turn

        if(game.headers["Result"] ==
"1/2-1/2"):
            continue
        elif(game.headers["Result"] ==
"1-0"):
            result = 1.0
        elif(game.headers["Result"] ==
"0-1"):
            result = -1.0
        else:
            print("\nNo result
reported!")
            eof = True
            break


        board = game.board()
        turn = -1.0
        for move in
game.mainline_moves():
            board.push(move)
            fen_table =
fen_transform(board.fen())

training_set.append(fen_table)

training_labels.append(result)

training_turn_set.append(turn)
            turn = -1.0*turn


        datagames += 1
        combined_games += 1


    training_set =
np.asarray(training_set)
    training_turn_set =
np.asarray(training_turn_set)
    training_labels =
np.asarray(training_labels)
    probability_labels =
np.asarray(probability_labels)
    probability_fens =
np.asarray(probability_fens)
    probability_turns =
np.asarray(probability_turns)

    return (training_set,
training_turn_set, training_labels,
            probability_fens,
probability_turns, probability_labels,
            datagames, combined_games,
parsed, eof)


if __name__ == "__main__":

    pgn = open("ficsgamesdb_2004.pgn")
    eof = False
    batch_size = 50
    batch = 1

    file =
open("games_parsed.txt","r+")
    parsed = int(file.read())
    file.close()

    print("Skipping already analyzed
games...")
    for match in range(parsed):
        game = chess.pgn.read_game(pgn)

    print("Examining rest of
file...\n")
    while eof == False:

        print("\nBatch:", batch)
        file =
open("datagames.txt","r+")
        datagames = int(file.read())
        file.close()

        file =
open("combinedgames.txt","r+")
        combined_games =
int(file.read())
        file.close()

        (training_set,
training_turn_set, training_labels,

          probability_fens,
probability_turns, probability_labels,
```

```
        new_datagames,
new_combined_games, new_parsed, eof) =
get_pgn_data(
        pgn, batch_size, datagames,
combined_games, parsed)

        model1 =
load_model("datatrain_model.h5")
        model2 =
load_model("combined_model.h5")

        probability_model1 =
load_model("datatrain_probability_model
.h5")
        probability_model2 =
load_model("combined_probability_model.
h5")

        if (len(training_set) > 0):
            print("\nModel 1:")
            model1.fit([training_set,
training_turn_set], training_labels,
              batch_size=32, epochs=15,
verbose=1)
             print("\nModel 2:")
             model2.fit([training_set,
training_turn_set], training_labels,
               batch_size=32,
epochs=15, verbose=1)
            print("\nProbability Model
1:")

probability_model1.fit([probability_fen
s, probability_turns],
probability_labels,
              batch_size=100,
epochs=40, verbose=1)
             print("\nProbability Model
2:")

probability_model2.fit([probability_fen
s, probability_turns],
probability_labels,
               batch_size=100,
epochs=15, verbose=1)


model1.save("datatrain_model.h5")

#model2.save("combined_model.h5")


probability_model1.save("datatrain_prob
ability_model.h5")

#probability_model2.save("combined_prob
ability_model.h5")

        file =
open("datagames.txt","w+")
        file.write(str(new_datagames))
        file.close()

         file =
open("combinedgames.txt","w+")

file.write(str(new_combined_games))
         file.close()

        file =
open("games_parsed.txt","w+")
        file.write(str(new_parsed))
        file.close()

        batch += 1
        parsed = new_parsed
pgn.close()
```

Αρχείο: fen_transformation.py

```
game = Board()

pieces = ['k', 'q', 'r', 'b', 'n', 'p',
'P', 'N', 'B', 'R', 'Q', 'K']
pawn_values = [0.0, 0.0, 0.0, 0.0, 0.0,
-1.0, 1.0, 0.0, 0.0, 0.0, 0.0, 0.0]
knight_values = [0.0, 0.0, 0.0, 0.0, -
1.0, 0.0, 0.0, 1.0, 0.0, 0.0, 0.0, 0.0]
bishop_values = [0.0, 0.0, 0.0, -1.0,
0.0, 0.0, 0.0, 0.0, 1.0, 0.0, 0.0, 0.0]
rook_values = [0.0, 0.0, -1.0, 0.0,
0.0, 0.0, 0.0, 0.0, 0.0, 1.0, 0.0, 0.0]
queen_values = [0.0, -1.0, 0.0, 0.0,
0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 1.0, 0.0]
king_values = [-1.0, 0.0, 0.0, 0.0,
0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 1.0]
empty_position = [0.0, 0.0, 0.0, 0.0,
0.0, 0.0]


def fen_transform(fen):

    position_list = []
    row_list = []
    chessboard_list = []

    fen = fen.split(" ")[0]
    fen = fen.split("/")
    for row in fen:
        for char in row:
            if char.isalpha():

position_list.append(pawn_values[pieces
.index(char)])

position_list.append(knight_values[piec
es.index(char)])

position_list.append(bishop_values[piec
es.index(char)])

position_list.append(rook_values[pieces
.index(char)])

position_list.append(queen_values[piece
s.index(char)])

position_list.append(king_values[pieces
.index(char)])
                #position_list =
piece_lists[pieces.index(char)]

row_list.append(position_list)

            else:
                position_list =
empty_position
                for i in range
(int(char)):

row_list.append(position_list)
            position_list = []


chessboard_list.append(row_list)
        row_list = []

    fen_table =
np.asarray(chessboard_list)

    return fen_table
```

Αρχείο: mcts.py

```
import numpy as np
from fen_transformation import
fen_transform
from move_transformation import
predict_probability
import random
import time
import tensorflow as tf


class Node:

    def __init__(self, player, state,
prob):
        self.visit_count = 0
        self.player = player
        self.value_sum = 0
        self.children = []
        self.state = state
        self.prob = prob

    def expanded(self):
        return len(self.children) > 0

    def value(self, parent, player,
prob_mul):

        if self.visit_count == 0:
            return player*np.inf

        node_score = self.value_sum/
self.visit_count
        prior_score =
prob_mul*self.prob* np.sqrt(

np.log(parent.visit_count)/self.visit_c
ount)
        return node_score +
player*prior_score


    def select_child(self, prob_mul):

        best_score = -
self.player*np.inf
        best_action = 0
        best_child = self.children[0]

        index = 0

        if self.player==1:
            for child in self.children:
                score =
child.value(self, 1, prob_mul)

                if score > best_score:
                    best_score = score
                    best_action = index
                    best_child = child

                index += 1

        elif self.player==-1:
            for child in self.children:
                score =
child.value(self, -1, prob_mul)

                if score < best_score:
                    best_score = score
                    best_action = index
                    best_child = child

                index += 1

        return best_action, best_child

    def expand(self, game, player,
prob_model):

        self.player = player

        legal_fens = game.legal_fens()
        probs_list =
predict_probability(game, prob_model)
        for i in
range(len(legal_fens)):
            #if legal_fens[i] in
visited_df['fen'].values:
            #
self.children.append(visited_df.loc[vis
ited_df['fen'] ==
legal_fens[i]]['node'].values[0])
            #else:


self.children.append(Node(self.player*-
1, legal_fens[i], probs_list[i]))


        return


class MCTS:

    def __init__(self, model,
prob_model):
        self.model = model
        self.prob_model = prob_model

    def run(self, game, time_limit):

        turns_simed = 0
        ev_mul = 5
        prob_mul = 1.5
        player = game.turn_int()

        fen = game.fen()
        #visited_df =
pd.DataFrame(columns = ['node', 'fen'])
        time_start = time.time() +
time_limit
        sims = 0

        prob_list =
predict_probability(game,
self.prob_model)
        root = Node(player, fen,
prob_list)

#visited_df.loc[len(visited_df.index)]
= (root, fen)
        root.expand(game, player,
self.prob_model)

        while(time.time() <
time_start):
        #for i in range(sims):
            #print("sims", i+1)
            prediction_set = []
            turn_set = []
            node = root
            path = [node]

            #Select
            while node.expanded():
                #print()
                #print("row:", row)
                action, node =
node.select_child(prob_mul)
```

```
game.move(game.legal_moves()[action])
                #print(action)
                path.append(node)

            parent = path[-2]
            reward =
game.state_reward()
            #If game has not ended:
Expand
            if reward is None:
                node.expand(game,
parent.player * -1, self.prob_model)
                #if node.state not in
visited_df['fen'].values:

#visited_df.loc[len(visited_df.index)]
= (node, game.fen())

                #Simulation
                reward =
self.simulate(game, turns_simed)

                if reward is None:
                    fen_table =
fen_transform(game.fen())

prediction_set.append(fen_table)
                    prediction_set =
np.asarray(prediction_set)

turn_set.append(game.turn_int())
                    turn_set =
np.asarray(turn_set)
                    reward =
tf.keras.backend.get_value(self.model([
prediction_set, turn_set]))[0][0]
                    #print()
                    #print(reward)

            #Backpropagate
            self.backpropagate(path,
reward, ev_mul)
            game.reset_to_specific(fen)
            sims += 1

        best_move =
self.get_best_move(root, game)

        return best_move

    def backpropagate(self, path,
reward, ev_mul):

        for node in reversed(path):
            node.value_sum +=
ev_mul*reward
            node.visit_count += 1

        return


    def simulate(self, game,
turns_simed):

        reward = game.state_reward()
        for i in range(turns_simed):
            reward =
game.state_reward()
            if reward is None:
                legal_moves =
game.legal_moves()
                if len(legal_moves) >
0:

game.move(legal_moves[random.randint(0,
len(legal_moves)-1)])
                else:
                    reward = 0
                    return reward
            else:
                return reward

        return reward

    def get_best_move(self, root,
game):

        best_avg_value = -np.inf
        best_node = 0
        index = 0

        for node in root.children:
            avg_value =
node.visit_count
            if
avg_value>best_avg_value:
                best_avg_value =
avg_value
                best_node = index
            index += 1

        return
game.legal_moves()[best_node]
```

Αρχείο: mcts_no_prob.py

```
import numpy as np
from fen_transformation import
fen_transform
import random
import time
import tensorflow as tf


class Node:

    def __init__(self, player, state):
        self.visit_count = 0
        self.player = player
        self.value_sum = 0
        self.children = []
        self.state = state

    def expanded(self):
        return len(self.children) > 0

    def value(self, parent, player):

        if self.visit_count == 0:
            return player*np.inf

        node_score = self.value_sum/
self.visit_count
        prior_score = 1.4*np.sqrt(

np.log(parent.visit_count)/self.visit_c
ount)
        return node_score +
player*prior_score


    def select_child(self):

        best_score = -
self.player*np.inf
        best_action = 0
        best_child = self.children[0]

        index = 0

        if self.player==1:
            for child in self.children:
                score =
child.value(self, 1)

                if score > best_score:
                    best_score = score
                    best_action = index
                    best_child = child

                index += 1

        elif self.player==-1:
            for child in self.children:
                score =
child.value(self, -1)

                if score < best_score:
                    best_score = score
                    best_action = index
                    best_child = child

                index += 1


        return best_action, best_child

    def expand(self, game, player):

        self.player = player

        legal_fens = game.legal_fens()
        for i in
range(len(legal_fens)):


self.children.append(Node(self.player*-
1, legal_fens[i]))


        return


class MCTS:

    def __init__(self, model):
        self.model = model

    def run(self, game, time_limit,
turns_simed, ev_mul):

        player = game.turn_int()

        fen = game.fen()
        time_start = time.time() +
time_limit
        sims = 0

        root = Node(player, fen)
        root.expand(game, player)

        while(time.time() <
time_start):
            prediction_set = []
            turn_set = []
            node = root
            path = [node]

            #Select
            while node.expanded():
                action, node =
node.select_child()

game.move(game.legal_moves()[action])
                #print(action)
                path.append(node)

            parent = path[-2]
            reward =
game.state_reward()
            #If game has not ended:
Expand
            if reward is None:
                node.expand(game,
parent.player * -1)
                #if node.state not in
visited_df['fen'].values:

#visited_df.loc[len(visited_df.index)]
= (node, game.fen())

                #Simulation
                reward =
self.simulate(game, turns_simed)

                if reward is None:
                    fen_table =
fen_transform(game.fen())

prediction_set.append(fen_table)
                    prediction_set =
np.asarray(prediction_set)

turn_set.append(game.turn_int())
                    turn_set =
np.asarray(turn_set)
```

```
                    reward =
tf.keras.backend.get_value(self.model([
prediction_set, turn_set]))[0][0]
                    #print()
                    #print(reward)

            #Backpropagate
            self.backpropagate(path,
reward, ev_mul)
            game.reset_to_specific(fen)
            sims += 1

        best_move =
self.get_best_move(root, game)

        return best_move

    def backpropagate(self, path,
reward, ev_mul):

        for node in reversed(path):
            node.value_sum +=
ev_mul*reward
            node.visit_count += 1

        return


    def simulate(self, game,
turns_simed):

        reward = game.state_reward()
        for i in range(turns_simed):
            reward =
game.state_reward()
            if reward is None:
                legal_moves =
game.legal_moves()
                if len(legal_moves) >
0:

game.move(legal_moves[random.randint(0,
len(legal_moves)-1)])
                else:
                    reward = 0
                    return reward
            else:
                return reward

        return reward

    def get_best_move(self, root,
game):

        best_avg_value = -np.inf
        best_node = 0
        index = 0

        for node in root.children:
            avg_value =
node.visit_count
            if
avg_value>best_avg_value:
                best_avg_value =
avg_value
                best_node = index
            index += 1

        return
game.legal_moves()[best_node]
```

Αρχείο: model.py

```
import keras
from tensorflow.python.keras import
layers, activations, models
import tensorflow as tf


def neural_net():

    model = keras.Sequential()

    model.add(layers.Conv2D(128,
kernel_size = (3, 3),
                     input_shape =
(8,8,6), padding = 'same'))

model.add(layers.BatchNormalization())

model.add(layers.Activation(activations
.relu))

    model.add(layers.Conv2D(128,
kernel_size = (3, 3), padding =
'same'))

model.add(layers.BatchNormalization())

model.add(layers.Activation(activations
.relu))

    model.add(layers.Conv2D(64,
kernel_size = (3, 3), padding =
'same'))

model.add(layers.BatchNormalization())

model.add(layers.Activation(activations
.relu))

    model.add(layers.Flatten())

    model.add(layers.Dense(64,
activation = 'relu'))

    model.add(layers.Dense(1,
activation = 'tanh'))

    opt =
tf.keras.optimizers.SGD(learning_rate=0
.01, momentum = 0.3)
    model.compile(loss =
"mean_squared_error",
              optimizer = opt,
              metrics=['mse'])

    return model

def refined_model():

    chessboard_input =
layers.Input(shape = (8,8,6))
    turn_input = layers.Input( shape =
(1,))

    conv1 = layers.Conv2D(128,
kernel_size = (3, 3), padding =
'same')(chessboard_input)
    bn1 =
layers.BatchNormalization()(conv1)
    act1 =
layers.Activation(activations.relu)(bn1
)

    conv2 = layers.Conv2D(128,
kernel_size = (3, 3), padding =
'same')(act1)
    bn2 =
layers.BatchNormalization()(conv2)
    act2 =
layers.Activation(activations.relu)(bn2
)

    conv3 = layers.Conv2D(64,
kernel_size = (3, 3), padding =
'same')(act2)
    bn3 =
layers.BatchNormalization()(conv3)
    act3 =
layers.Activation(activations.relu)(bn3
)

    flatten = layers.Flatten()(act3)
    merge =
layers.concatenate([flatten,
turn_input])

    d1 = layers.Dense(64, activation =
'relu')(merge)
    out = layers.Dense(1, activation =
'tanh')(d1)

    model = models.Model(inputs =
[chessboard_input, turn_input], outputs
= out)
    opt =
tf.keras.optimizers.SGD(learning_rate=0
.005, momentum = 0.2)
    model.compile(loss =
"mean_squared_error",
              optimizer = opt,
              metrics=['mse'])

    return model

def probability_model():

    chessboard_input =
layers.Input(shape = (8,8,6))
    turn_input = layers.Input( shape =
(1,))

    conv1 = layers.Conv2D(128,
kernel_size = (3, 3), padding =
'same')(chessboard_input)
    bn1 =
layers.BatchNormalization()(conv1)
    act1 =
layers.Activation(activations.relu)(bn1
)

    conv2 = layers.Conv2D(128,
kernel_size = (3, 3), padding =
'same')(act1)
    bn2 =
layers.BatchNormalization()(conv2)
    act2 =
layers.Activation(activations.relu)(bn2
)

    conv3 = layers.Conv2D(64,
kernel_size = (3, 3), padding =
'same')(act2)
    bn3 =
layers.BatchNormalization()(conv3)
    act3 =
layers.Activation(activations.relu)(bn3
)

    flatten = layers.Flatten()(act3)
```

```
    merge =
layers.concatenate([flatten,
turn_input])

    d1 = layers.Dense(64, activation =
'relu')(merge)
    out = layers.Dense(128, activation
= 'softmax')(d1)

    model = models.Model(inputs =
[chessboard_input, turn_input], outputs
= out)
    opt =
tf.keras.optimizers.SGD(learning_rate=0
.001, momentum = 0.2)
    model.compile(loss =
"categorical_crossentropy",
                  optimizer = opt,
                  metrics=['accuracy'])

    return model
```

Αρχείο: move_transformation.py

```
import tensorflow as tf
from fen_transformation import
fen_transform
import numpy as np

squares = ['a1', 'a2', 'a3', 'a4',
'a5', 'a6', 'a7', 'a8',
           'b1', 'b2', 'b3', 'b4',
'b5', 'b6', 'b7', 'b8',
           'c1', 'c2', 'c3', 'c4',
'c5', 'c6', 'c7', 'c8',
           'd1', 'd2', 'd3', 'd4',
'd5', 'd6', 'd7', 'd8',
           'e1', 'e2', 'e3', 'e4',
'e5', 'e6', 'e7', 'e8',
           'f1', 'f2', 'f3', 'f4',
'f5', 'f6', 'f7', 'f8',
           'g1', 'g2', 'g3', 'g4',
'g5', 'g6', 'g7', 'g8',
           'h1', 'h2', 'h3', 'h4',
'h5', 'h6', 'h7', 'h8']

def move_transform(move):

    prob_labels = [0]*128
    start = move[0:2]
    end = move[2:4]

    prob_labels[squares.index(start)] =
0.5
    prob_labels[64 +
squares.index(end)] = 0.5

    return prob_labels

def predict_probability(game, model):

    fen_table =
fen_transform(game.fen())

    prediction_set = []
    prediction_set.append(fen_table)
    prediction_set =
np.asarray(prediction_set)

    turn_set = []
    turn_set.append(game.turn_int())
    turn_set = np.asarray(turn_set)

    probs =
tf.keras.backend.get_value(model([predi
ction_set, turn_set]))[0]

    probs_list = []
    for move in game.legal_moves():
        start = move[0:2]
        end = move[2:4]

probs_list.append(probs[squares.index(s
tart)] + probs[64 +
squares.index(end)])

    return probs_list
```

Αρχείο: selftrain.py

```
from board import Board
import numpy as np
from fen_transformation import
fen_transform
from move_transformation import
move_transform
from tensorflow.keras.models import
load_model
import random
import tensorflow as tf

def selfplay(games_simed, model,
distributions, selfgames,
combined_games):

    sys_random = random.SystemRandom()
    training_set = []
    training_turn_set = []
    training_labels = []
    probability_labels = []
    probability_fens = []
    probability_turns = []

    wins = 0
    draws = 0
    losses = 0


    for match in range(games_simed):

        print("Match:", match+1)
        game = Board()
        moves_set = []
        moves = 0

        while not
game.board.is_game_over(claim_draw =
True):

            legal_moves =
game.legal_moves()
            legal_fens =
game.legal_fens()
            prediction_set = []
            turn_set = []

            if moves%2 == 0:
                next_turn = -1.0
            else:
                next_turn = 1.0

            for i in
range(len(legal_fens)):
                fen_table =
fen_transform(legal_fens[i])

prediction_set.append(fen_table)

turn_set.append(next_turn)

            prediction_set =
np.asarray(prediction_set)
            turn_set =
np.asarray(turn_set)
            #predictions =
model.predict([prediction_set,turn_set]
).reshape(len(prediction_set))
            predictions =
tf.keras.backend.get_value(model([predi
ction_set, turn_set])).flatten()

            if moves%2 == 0:
                best_indices =
predictions.argsort()[-
(int(len(predictions)/8) + 1):][::-1]
            else:
                best_indices = (-
predictions).argsort()[-
(int(len(predictions)/8) + 1):][::-1]

            weights =
distributions[len(best_indices) - 1]

            stohastic_choice =
sys_random.choices(best_indices,
weights=weights)[0]


moves_set.append(prediction_set[stohast
ic_choice])

probability_fens.append(fen_transform(g
ame.fen()))

probability_labels.append(move_transfor
m(legal_moves[stohastic_choice]))

probability_turns.append(game.turn_int(
))

game.move(legal_moves[stohastic_choice]
)
            moves += 1

        result = "draw"
        if game.board.is_checkmate():
            result = not
game.board.turn

        if result=="draw":
            draws += 1

        elif result==False:
            losses += 1
            turn = -1.0
            for i in range(moves):

training_labels.append(-1.0)

training_set.append(moves_set[i])

training_turn_set.append(turn)
                turn = -1.0*turn

        else:
            wins += 1
            turn = -1.0
            for i in range(moves):

training_labels.append(1.0)

training_set.append(moves_set[i])

training_turn_set.append(turn)
                turn = -1.0*turn



    print("Wins:", wins, "Draws:",
draws, "Losses:", losses, "\n")


    training_set =
np.asarray(training_set)
    training_turn_set =
np.asarray(training_turn_set)
    training_labels =
np.asarray(training_labels)
```

```
    probability_labels =
np.asarray(probability_labels)
    probability_fens =
np.asarray(probability_fens)
    probability_turns =
np.asarray(probability_turns)
    new_selfgames = selfgames + losses
+ wins
    new_combined = combined_games +
losses + wins

    return (training_set,
training_turn_set, training_labels,
probability_fens,
            probability_turns,
probability_labels, new_selfgames,
new_combined)

def create_distributions():

    distributions = []
    for length in range(1,50):
        weights = [100]
        prob = 100
        for i in range(1, length):
            prob = prob - int(prob/2.3)
            weights.append(prob)
        distributions.append(weights)

    return distributions



if __name__ == "__main__":

    distributions =
create_distributions()
    games_simed = 100

    for i in range(100):
        print("\nBatch:", i+1, "\n")

        file =
open("selfgames.txt","r+")
        selfgames = int(file.read())
        file.close()

        file =
open("combinedgames.txt","r+")
        combined_games =
int(file.read())
        file.close()

        model1 =
load_model("selftrain_model.h5")
        #model2 =
load_model("combined_model.h5")
        probability_model1 =
load_model("selftrain_probability_model
.h5")
        #probability_model2 =
load_model("combined_probability_model.
h5")

        (training_set,
training_turn_set, training_labels,
probability_fens,
            probability_turns,
probability_labels, new_selfgames,
new_combined) = (
        selfplay(games_simed, model1,
distributions, selfgames,
combined_games))

        if (len(training_set) > 0):
            model1.fit([training_set,
training_turn_set], training_labels,
              batch_size=64, epochs=15,
verbose=2)
            print("\n")
            # model2.fit([training_set,
training_turn_set], training_labels,
            #   batch_size=64,
epochs=15, verbose=2)
            # print("\n")

probability_model1.fit([probability_fen
s, probability_turns],
probability_labels,
              batch_size=164,
epochs=40, verbose=2)
        # print("\n")
        #
probability_model2.fit([probability_fen
s, probability_turns],
probability_labels,
        #       batch_size=164,
epochs=40, verbose=2)


model1.save("selftrain_model.h5")

#model2.save("combined_model.h5")

probability_model1.save("selftrain_prob
ability_model.h5")

#probability_model2.save("combined_prob
ability_model.h5")

        file =
open("selfgames.txt","w+")
        file.write(str(new_selfgames))
        file.close()

        # file =
open("combinedgames.txt","w+")
        # file.write(str(new_combined))
        # file.close()
```

Μέρος Β – Κώδικας διαδικτυακής εφαρμογής

Αρχείο: init.py

```
from flask import Flask, render_template,
request, jsonify, url_for, redirect,
session
import players
from board import Board
import time

app = Flask(__name__)
app.secret_key = "prlce_secret_key"

# mcts1 = None
# mcts2 = None
# session["mode"]= None
# session["opponent_model"]= None
# session["selfplay_model1"]= None
# session["selfplay_model2"]= None
# session["time_limit1"]= 5
# session["time_limit2"]= 5
# session["turns_simed"]= 10
player_move = "None"
game = Board()
# color = None
# opp_color = None

def run_app():
    app.run(debug = True)


@app.route('/', methods = ["POST", "GET"])
def index():
    #global mode, color, opp_color,
opponent_model, selfplay_model1,
selfplay_model2
    session["reset"] = False

    # game = Board()
    # with open('data.pkl', 'wb') as
output:
    #     pickle.dump(game, output,
pickle.HIGHEST_PROTOCOL)
    session["time_limit1"]= 20
    session["time_limit2"]= 20

    if request.method == "POST":
        session["mode"] =
request.form["mode"]
        session["color"] =
request.form["color"]

        session["opponent_model"] =
request.form["opp_model"]
        session["selfplay_model1"] =
request.form["sp_model1"]
        session["selfplay_model2"] =
request.form["sp_model2"]

        if session["color"] == "White":
            session["opp_color"] = "Black"
        else:
            session["opp_color"] = "White"

        return
redirect(url_for(session["mode"]))
    else:
        return
render_template('index.html')


@app.route('/quit')
def quit():
    func =
request.environ.get('werkzeug.server.shutdo
wn')
    func()
    return "Quitting..."

@app.route('/selfplay')
def selfplay():
    #global session["mcts1"],
selfplay_model1, session["mcts2"],
selfplay_model2
    # global mcts1, mcts2
    # with open('data.pkl', 'rb') as input:
    #     game = pickle.load(input)

    # mcts1 =
players.initialize_mcts(session["selfplay_m
odel1"])
    # mcts2 =
players.initialize_mcts(session["selfplay_m
odel2"])
    game.reset()
    #
players.initialize_engine1(session["selfpla
y_model1"])
    #
players.initialize_engine2(session["selfpla
y_model2"])

    # with open('data.pkl', 'wb') as
output:
    #     pickle.dump(game, output,
pickle.HIGHEST_PROTOCOL)
    time.sleep(5)

    return render_template('selfplay.html',
                starting_fen =
game.board.fen(),
                model1 =
session["selfplay_model1"],
                model2 =
session["selfplay_model2"])

@app.route('/versusplay')
def versusplay():
    #global session["mcts1"], color,
opponent_model
    # global mcts1
    # with open('data.pkl', 'rb') as input:
    #     game = pickle.load(input)
    # mcts1 =
players.initialize_mcts(session["opponent_m
odel"])
    game.reset()

#players.initialize_engine1(session["oppone
nt_model"])
    # with open('data.pkl', 'wb') as
output:
    #     pickle.dump(game, output,
pickle.HIGHEST_PROTOCOL)
    time.sleep(5)

    return
render_template('versusplay.html',
        starting_fen = game.board.fen(),
        first_moves = list(move.uci() for
move in game.board.legal_moves),
        player_color =
session["color"].lower(),
        opponent =
session["opponent_model"])
```

```
@app.route('/selfmove', methods=['GET',
'POST'])
def selfmove():
    #global session["mcts1"], time_limit1,
session["mcts2"], time_limit2, turns_simed
    #global mcts1, mcts2

    # with open('data.pkl', 'rb') as input:
    #     game = pickle.load(input)

    while not
game.board.is_game_over(claim_draw = True):
        turn =
game.turn_str(game.board.turn)
        if turn == "White":
            #uci =
players.engine_move(game, mcts1,
session["time_limit1"],
session["turns_simed"])
            uci = players.engine_move(game,
session["time_limit1"],
session["selfplay_model1"])
        else:
            #uci =
players.engine_move(game, mcts2,
session["time_limit2"],
session["turns_simed"])
            uci = players.engine_move(game,
session["time_limit2"],
session["selfplay_model2"])

        game.move(uci)
        # with open('data.pkl', 'wb') as
output:
        #     pickle.dump(game, output,
pickle.HIGHEST_PROTOCOL)

        # GET request
        if request.method == 'GET':
            message = {'move':uci,
'player': turn,

'fen':game.board.fen(),
'isgameover':game.board.is_game_over()}
            return jsonify(message)

    result, msg = check_result()
    if request.method == 'GET':
        message = {'result': result, 'msg':
msg}
        return jsonify(message)

    game.reset()
    return

@app.route('/get_player_move',
methods=['GET', 'POST'])
def get_player_move():
    #global player_move, color, opp_color
    global player_move
    # with open('data.pkl', 'rb') as input:
    #     game = pickle.load(input)
    #while game.turn_str(game.board.turn)
== session["opp_color"]:
        #pass
    while player_move=="None":
        continue
    game.move(player_move)
    isgameover = game.board.is_game_over()

    # with open('data.pkl', 'wb') as
output:
    #     pickle.dump(game, output,
pickle.HIGHEST_PROTOCOL)

    # GET player's move
    if request.method == 'GET':
        message = {'move': player_move,
'player': session["color"],
                   'fen':game.board.fen(),
'isgameover': isgameover}
        return jsonify(message)

    return

@app.route('/get_engine_move',
methods=['GET', 'POST'])
def get_engine_move():
    #global session["mcts1"], time_limit1,
turns_simed, color, opp_color
    #global mcts1
    # with open('data.pkl', 'rb') as input:
    #     game = pickle.load(input)
    #while game.turn_str(game.board.turn)
== session["color"]:
        #pass
    isgameover = game.board.is_game_over()
    if not isgameover:
        #engine_move =
players.engine_move(game, mcts1,
session["time_limit1"],
session["turns_simed"])
        engine_move =
players.engine_move(game,
session["time_limit1"],
session["opponent_model"])
        game.move(engine_move)

        # with open('data.pkl', 'wb') as
output:
        #     pickle.dump(game, output,
pickle.HIGHEST_PROTOCOL)
        next_moves = list(move.uci() for
move in game.board.legal_moves)

        # GET engine's move
        if request.method == 'GET':
            message = {'move':engine_move,
'player': session["opp_color"],
                       'fen':game.board.fen(),
'isgameover': isgameover,
                       'legal_moves':
next_moves}
            return jsonify(message)

    return

@app.route('/get_result', methods=['GET',
'POST'])
def get_result():
    # with open('data.pkl', 'rb') as input:
    #     game = pickle.load(input)

    isgameover = game.board.is_game_over()
    if isgameover:
        result, msg = check_result()
        if request.method == 'GET':
            message = {'result': result,
'msg': msg}
            return jsonify(message)

    return


@app.route('/reset',
methods=['GET','POST'])
def reset():
    #global opponent_model,
session["mcts1"], selfplay_model1,
session["mcts2"], selfplay_model2, mode
    #global mcts1, mcts2
    # with open('data.pkl', 'rb') as input:
```

```
    #     game = pickle.load(input)

    if request.method == "POST":

        # if session["mode"] ==
"versusplay":
        #     #mcts1 =
players.initialize_mcts(session["opponent_m
odel"])
        #
players.initialize_engine1(session["opponen
t_model"])

        # else:
        #     # mcts1 =
players.initialize_mcts(session["selfplay_m
odel1"])
        #     # mcts2 =
players.initialize_mcts(session["selfplay_m
odel2"])
        #
players.initialize_engine1(session["selfpla
y_model1"])
        #
players.initialize_engine2(session["selfpla
y_model2"])
        # time.sleep(3)

        req = request.get_json()
        if req.get("reset") == "true":
            game.reset()
            time.sleep(5)
            # with open('data.pkl', 'wb')
as output:
            #     pickle.dump(game, output,
pickle.HIGHEST_PROTOCOL)
        return req.get("reset"), 200



@app.route('/post_move',
methods=['GET','POST'])
def post_move():
    global player_move

    if request.method == "POST":
        req = request.get_json()
        player_move = req.get("move")
        return player_move, 200


def check_result():

    # with open('data.pkl', 'rb') as input:
    #     game = pickle.load(input)

    result = "draw"
    if game.board.is_checkmate():
        msg = "Checkmate: " +
game.turn_str(not game.board.turn) + "
wins!"
        result = not game.board.turn
    elif game.board.is_stalemate():
        msg = "Draw: Stalemate"
    elif
game.board.is_fivefold_repetition():
        msg = "Draw: 5-fold repetition"
    elif
game.board.is_insufficient_material():
        msg = "Draw: Insufficient material"
    elif game.board.can_claim_draw():
        msg = "Draw: Claim"

    return result, msg

if __name__ == "__main__":
    # with open('data.pkl', 'wb') as
output:
    #     game = Board()
    #     pickle.dump(game, output,
pickle.HIGHEST_PROTOCOL)
    run_app()
```

Αρχείο: players.py

```
import random
import mcts
import mcts_no_prob
import chess
import chess.engine
from tensorflow.keras.models import
load_model

def engine_move(game, time_limit,
model_type):
    #global mcts2

    if model_type == "1":
       ev_model =
load_model("selftrain_model.h5")
       mcts_tree =
mcts_no_prob.MCTS(ev_model)
       move = mcts_tree.run(game,
time_limit, 0, 10)
       return move
    elif model_type == "2":
       ev_model =
load_model("datatrain_model.h5")
       prob_model =
load_model("datatrain_probability_model.h5"
)
    elif model_type == "3":
       ev_model =
load_model("combined_model.h5")
       prob_model =
load_model("combined_probability_model.h5")
    elif model_type == "4":
       engine =
chess.engine.SimpleEngine.popen_uci("stockf
ish")
       board = game.board
       result = engine.play(board,
chess.engine.Limit(time=0.5))
       return str(result.move)

    mcts_tree = mcts.MCTS(ev_model,
prob_model)
    move = mcts_tree.run(game, time_limit)

    return move
```

Αρχείο: base.html

```
<!DOCTYPE html>
<html lang="en">
<head>
    <meta charset="UTF-8">
    <meta name="viewport"
content="width=device-width, initial-
scale=1.0">
    <meta http-equiv="X-UA-Compatible"
content="ie=edge">

    <!-- Bootstrap CSS -->
    <link rel="stylesheet"
href="https://stackpath.bootstrapcdn.com/bo
otstrap/4.3.1/css/bootstrap.min.css"
integrity="sha384-
ggOyR0iXCbMQv3Xipma34MD+dH/1fQ784/j6cY/iJTQ
UOhcWr7x9JvoRxT2MZw1T"
crossorigin="anonymous">
    <link rel="shortcut icon" href="#">

    {% block head %}{% endblock %}

</head>
<body>

    <!-- Optional JavaScript -->
    <!-- jQuery first, then Popper.js, then
Bootstrap JS -->
    <script
src="https://code.jquery.com/jquery-
3.3.1.slim.min.js" integrity="sha384-
q8i/X+965DzO0rT7abK41JStQIAqVgRVzpbzo5smXKp
4YfRvH+8abtTE1Pi6jizo"
crossorigin="anonymous"></script>
    <script
src="https://cdnjs.cloudflare.com/ajax/libs
/popper.js/1.14.7/umd/popper.min.js"
integrity="sha384-
UO2eT0CpHqdSJQ6hJty5KVphtPhzWj9WO1clHTMGa3J
DZwrnQq4sF86dIHNDz0W1"
crossorigin="anonymous"></script>
    <script
src="https://stackpath.bootstrapcdn.com/boo
tstrap/4.3.1/js/bootstrap.min.js"
integrity="sha384-
JjSmVgyd0p3pXB1rRibZUAYoIIy6OrQ6VrjIEaFf/nJ
GzIxFDsf4x0xIM+B07jRM"
crossorigin="anonymous"></script>

    {% block body %}{% endblock %}

</body>
</html>
```

Αρχείο: index.html

```
{% extends 'base.html' %}

{% block head %}
<link rel="stylesheet" href="{{
url_for('static',
filename='css/chess_main.css') }}"
     type= "text/css">

{% endblock %}

{% block body %}

<div class="container-fluid  mine">

    <div class="row">
        <h1> Python ML Chess Engine </h1>
    </div>

    <!--
    <div class="row btn-up">
        <button id="versusplay"
type="button" class="btn btn-outline-dark
btn-lg"

onclick="window.location.href='/versusplay'
">Player vs AI</button>
    </div> -->

    <div class="row btn-up">
        <button id="versusplay"
type="button" class="btn btn-outline-dark
btn-lg"
        data-bs-toggle="modal" data-bs-
target="#modal1"
onclick="show_vsmodal()">Player vs
AI</button>
    </div>



    <div class="row btn-up">
        <button id="selfplay" type="button"
class="btn btn-outline-dark btn-lg"
        data-bs-toggle="modal" data-bs-
target="#modal2"
onclick="show_spmodal()">AI vs AI</button>
    </div>

    <form method="POST" id="form">
        <div class="modal fade" id="modal1"
tabindex="-1" aria-labelledby="modal1Label"
aria-hidden="true">
          <div class="modal-dialog modal-
dialog-centered modal-lg" >
            <div class="modal-content">
              <div class="modal-header">
                <h5 class="modal-title"
id="modal1Label">Choose Game Options</h5>
                <button type="button"
class="btn-close" data-bs-dismiss="modal"
aria-label="Close"
onclick="hide_vsmodal()"></button>
              </div>
              <div class="modal-body">
                <h5>Choose side </h5>
                <div class="row ">
                    <div class="col-10
justify-content-around row ">
                    <div class="profile_div
white col-2 btn" role="button"
onclick="choose_white()">
                        <div class="icon-
div row">
                            <img
src="../static/img/chesspieces/wikipedia/wK
.png">
                        </div>
                        <div class="span-
div row">

<span>White</span>
                        </div>
                    </div>

                    <div class="profile_div
black col-2 btn" role="button"
onclick="choose_black()">
                        <div class="icon-
div row">
                            <img
src="../static/img/chesspieces/wikipedia/bK
.png">
                        </div>
                        <div class="span-
div row">

<span>Black</span>
                        </div>
                    </div>
                    </div>
                </div>

                <hr>
                <h5>Choose Opponent</h5>
                <select class="form-select"
name="opp_model">
                    <option selected
value="1">Self-Trained Python RL Chess
Engine (Trained by selfplaying)</option>
                    <option value="2">Data-
Trained Python Chess Engine (Trained using
data from professional matches)</option>
                    <option
value="3">Combined Python Chess Engine
(Trained both by selfplaying and using
data)</option>
                    <option
value="4">Stockfish Chess Engine </option>
                </select>
              </div>
              <div class="modal-footer">
                <button type="button"
class="btn btn-secondary" data-bs-
dismiss="modal"
onclick="hide_vsmodal()">Close</button>
                <button type="submit"
class="btn btn-primary">Play</button>
              </div>
            </div>
          </div>
        </div>

        <div class="modal fade" id="modal2"
tabindex="-1" aria-labelledby="modal2Label"
aria-hidden="true">
          <div class="modal-dialog modal-
dialog-centered modal-lg" >
            <div class="modal-content">
              <div class="modal-header">
                <h5 class="modal-title"
id="modal2Label">Choose Players</h5>
                <button type="button"
class="btn-close" data-bs-dismiss="modal"
aria-label="Close"
onclick="hide_spmodal()"></button>
              </div>
              <div class="modal-body">
                <h5><img
src="../static/img/chesspieces/wikipedia/wK
```

```
.png" style="width:45px;">Choose white
player</h5>
                <select class="form-select"
name="sp_model1">
                    <option selected
value="1">Self-Trained Python RL Chess
Engine (Trained by selfplaying)</option>
                    <option value="2">Data-
Trained Python RL Chess Engine (Trained
using data from professional
matches)</option>
                    <option
value="3">Combined Python Chess Engine
(Trained both by selfplaying and using
data)</option>
                    <option
value="4">Stockfish Chess Engine </option>
                </select>
                <hr>
                <h5><img
src="../static/img/chesspieces/wikipedia/bK
.png" style="width:45px;">Choose black
player</h5>
                <select class="form-select"
name="sp_model2">
                    <option selected
value="1">Self-Trained Python RL Chess
Engine (Trained by selfplaying)</option>
                    <option value="2">Data-
Trained Python RL Chess Engine (Trained
using data from professional
matches)</option>
                    <option
value="3">Combined Python Chess Engine
(Trained both by selfplaying and using
data)</option>
                    <option
value="4">Stockfish Chess Engine </option>
                </select>
              </div>
              <div class="modal-footer">
                <button type="button"
class="btn btn-secondary" data-bs-
dismiss="modal"
onclick="hide_spmodal()">Close</button>
                <button type="submit"
class="btn btn-primary">Confirm</button>
              </div>
            </div>
          </div>
        </div>

        <div class="row">
            <input id="send_color"
type="hidden" name="color" value="White">
        </div>

        <div class="row">
            <input id="send_page"
type="hidden" name="mode">
        </div>

    </form>

</div>

<script type="text/javascript" src="{{
url_for('static',
filename='js/chess_main.js') }}"></script>
{% endblock %}
```

Αρχείο: selfplay.html

```
{% extends 'base.html' %}

{% block head %}
<link rel="stylesheet" href="{{
url_for('static',
filename='css/chessboard.css') }}"
     type= "text/css">



{% endblock %}

{% block body %}

<div class="container-fluid  mine">


    <div class="row justify-content:
left;">
        <h3> Engine VS Engine </h3>
    </div>
    <div class="row justify-content:
left;">
        <h6> 10|10 Game - 10 Minute Game
with 10 Seconds Increment per Move  </h6>
    </div>

    <div class="row justify-content-end">
        <div id="board" style="width:
500px"></div>
        <div class="col-3 log-box justify-
content-center">
            <div class="row p1-box"> <span
id="player1"> Black </span> </div>
            <div class="row timer-box"
id="t1box">
                <span
id="minutes1">10</span>
                <span>:</span>
                <span
id="seconds1">00</span>
            </div>
            <br>
            <div class="overflow-auto"  id
= "scrollbox" style="max-width: 420px; max-
height: 220px;">
                <p id="history">Move
History: <p>
            </div>
            <br>
            <br>
            <div class="row timer-box
timer-box-bottom" id="t2box">
                <span
id="minutes2">10</span>
                <span>:</span>
                <span
id="seconds2">00</span>
            </div>
            <div class="row p2-box"> <span
id="player2"> White </span> </div>
        </div>
    </div>

    <br>

        <div class="row">
            <button type="button"
class="btn btn-outline-dark"
            id="startBtn"
onclick="fetch_move()">Start Game</button>
            <div class = "col-1"></div>
            <button type="button"
class="btn btn-outline-dark"
            id="resetBtn"
onclick="reset_board()">Reset Game</button>

        </div>

    <span id="starting_fen"
style="display:none">{{starting_fen}}</span
>
    <span id="model1"
style="display:none">{{model1}}</span>
    <span id="model2"
style="display:none">{{model2}}</span>
    <span id="reset_text"
style="display:none">Resetting
game...</span>
    <audio id="myAudio">
      <source src="{{ url_for('static',
filename='chess_move.wav') }}"
type="audio/ogg">
    </audio>

</div>

<script type="text/javascript" src="{{
url_for('static',
filename='js/jquery.min.js') }}"></script>
<script type="text/javascript" src="{{
url_for('static', filename='js/chessboard-
1.0.0.js') }}"></script>
<script type="text/javascript" src="{{
url_for('static',
filename='js/selfplay.js') }}"></script>


{% endblock %}
```

Αρχείο: versusplay.html

```
{% extends 'base.html' %}

{% block head %}
<link rel="stylesheet" href="{{
url_for('static',
filename='css/chessboard.css') }}"
     type= "text/css">


{% endblock %}

{% block body %}

<div class="container-fluid  mine">


    <div class="row justify-content:
left;">
        <h3> Player VS Engine </h3>
    </div>
    <div class="row justify-content:
left;">
        <h6> 10|10 Game - 10 Minute Game
with 10 Seconds Increment per Move  </h6>
    </div>

    <div class="row justify-content-end">
        <div id="board" style="width:
500px"></div>
        <div class="col-3 log-box justify-
content-center">
            <div class="row p1-box"> <span
id="player1"> Black </span> </div>
            <div class="row timer-box"
id="t1box">
                <span
id="minutes1">10</span>
                <span>:</span>
                <span
id="seconds1">00</span>
            </div>
            <br>
            <div class="overflow-auto"
id="scrollbox" style="max-width: 420px;
max-height: 220px;">
                <p id="history">Move
History: <p>
            </div>
            <br>
            <br>
            <div class="row timer-box
timer-box-bottom" id="t2box">
                <span
id="minutes2">10</span>
                <span>:</span>
                <span
id="seconds2">00</span>
            </div>
            <div class="row p2-box"> <span
id="player2"> You </span> </div>
        </div>
    </div>

    <br>

    <div class="row">

        <button type="button" class="btn
btn-outline-dark"
        id="resetBtn"
onclick="reset_board()">Reset Game</button>

        <div class = "col-1"></div>

        <button type="button" class="btn
btn-outline-dark"
        id="surrenderBtn"
onclick="surrender()">Surrender</button>
    </div>

    <span id="starting_fen"
style="display:none">{{starting_fen}}</span
>
    <span id="first_moves"
style="display:none">{{first_moves}}</span>
    <span id="player_color"
style="display:none">{{player_color}}</span
>
    <span id="opponent"
style="display:none">{{opponent}}</span>
    <span id="reset_text"
style="display:none">Resetting
game...</span>
    <audio id="myAudio">
      <source src="{{ url_for('static',
filename='chess_move.wav') }}"
type="audio/ogg">
    </audio>

</div>

<script type="text/javascript" src="{{
url_for('static',
filename='js/jquery.min.js') }}"></script>
<script type="text/javascript" src="{{
url_for('static', filename='js/chessboard-
1.0.0.js') }}"></script>
<script type="text/javascript" src="{{
url_for('static',
filename='js/versusplay.js') }}"></script>


{% endblock %}
```

Αρχείο: chess_main.css

```
body, html{
    margin: 0;
    padding: 0;
    font-family: 'Playfair Display', serif;
}

h1 {
    margin: 40px;
    text-align: center;
}

.mine{
    background-color: #a38560;
    background-image: linear-
gradient(315deg, #a38560 0%, #e0d4ae 74%);
    background-position: center;
    width: 100%;
    min-height: 100vh;
    text-align: center;
}

.row{
    justify-content: center;
    font-size: 2em;
    padding: 2px ;
}

.btn-up {
    margin: 75px 50px 25px;
}

.btn-down {
    margin: 25px 50px 75px;
}

.btn{
    border-radius: 30px;
}

span{
    font-size: 20px;
}

.icon-div img{
    width: 30px;
}

.profile_div{
    border:1px solid black;
    background-color: white;
    cursor: pointer;
}
```

Αρχείο: chessboard.css

```
body, html{
    margin: 0;
    padding: 0;
    font-family: 'Playfair Display', serif;
}

.mine{
    background-color: #a38560;
    background-image: linear-
gradient(315deg, #a38560 0%, #e0d4ae 74%);
    background-position: center;
    width: 100%;
    min-height: 100vh;
    text-align: center;
}

.row{
    justify-content: center;
    font-size: 2em;
    padding: 2px ;
}

h3 {
    margin: 20px;
    text-align: left;
    justify-content: left;
}

h6{
    margin: 10px;
}

span{
    font-size: 20px;
}

.log-box{
    border-radius: 30px;
    margin: 0px 66px 0px 66px;
    background-color: #c3cbdc;
    background-image: linear-
gradient(147deg, #c3cbdc 0%, #edf1f4 74%);
}

.p1-box{
    background-color: #d2d8d6;
    background-image: linear-
gradient(315deg, #d2d8d6 0%, #dce8e0 74%);
    border-radius: 40px;
    border-bottom: solid black;
}

.p1-box:hover, .p2-box:hover{
    background-color: #e4eee9;
    background-image: linear-
gradient(315deg, #e4eee9 0%, #93a5ce 74%);

}

.p2-box{
    background-color: #d2d8d6;
    background-image: linear-
gradient(315deg, #d2d8d6 0%, #dce8e0 74%);
    border-radius: 40px;
    border-top: solid black;
    position:absolute;
    bottom: 0;
    width: 100%;
}

p{
    font-size: 16px;
}

#scrollbox{
    padding: 10px;
    background-color: inherit;
    border-radius: 10px;
}

.timer-box{
    background-color: #C8C8C8;
    border-radius: 5px;
    margin-left: 30%;
    margin-right: 30%;
    box-shadow: none;
}

.timer-box-bottom{
    position:absolute;
    bottom: 37px;
    right: 3px;
    width: 37%;
}

.clearfix-7da63 {
  clear: both;
}

.board-b72b1 {
  border: 2px solid #404040;
  box-sizing: content-box;
}

.square-55d63 {
  float: left;
  position: relative;

  /* disable any native browser
highlighting */
  -webkit-touch-callout: none;
    -webkit-user-select: none;
     -khtml-user-select: none;
       -moz-user-select: none;
        -ms-user-select: none;
            user-select: none;
}

.white-1e1d7 {
  background-color: #f0d9b5;
  color: #b58863;
}

.black-3c85d {
  background-color: #b58863;
  color: #f0d9b5;
}

.highlight1-32417, .highlight2-9c5d2 {
  box-shadow: inset 0 0 3px 3px yellow;
}

.notation-322f9 {
  cursor: default;
  font-family: "Helvetica Neue", Helvetica,
Arial, sans-serif;
  font-size: 14px;
  position: absolute;
}

.alpha-d2270 {
  bottom: 1px;
  right: 3px;
}

.numeric-fc462 {
  top: 2px;
  left: 2px;
}
```

Αρχείο: chessboard-1.0.0.min.css ((c) 2019 Chris Oakman | MIT License chessboardjs.com)

```
.clearfix-7da63{clear:both}.board-b72b1{border:2px solid #404040;box-sizing:content-box}.square-55d63{float:left;position:relative;-webkit-touch-callout:none;-webkit-user-select:none;-khtml-user-select:none;-moz-user-select:none;-ms-user-select:none;user-select:none}.white-1e1d7{background-color:#f0d9b5;color:#b58863}.black-3c85d{background-color:#b58863;color:#f0d9b5}.highlight1-32417,.highlight2-9c5d2{box-shadow:inset 0 0 3px 3px #ff0}.notation-322f9{cursor:default;font-family:"Helvetica Neue",Helvetica,Arial,sans-serif;font-size:14px;position:absolute}.alpha-d2270{bottom:1px;right:3px}.numeric-fc462{top:2px;left:2px}
```

Αρχείο: chess_main.js

```
let white =
document.querySelector(".white");
let black =
document.querySelector(".black");
let color =
document.querySelector("#send_color");
let mode =
document.querySelector("#send_page");
white.style.backgroundColor =
"rgb(255,215,0)";

function show_vsmodal(){
    $('#modal1').modal("show")
    mode.value = "versusplay"
}

function hide_vsmodal(){
    $('#modal1').modal("hide")
}

function show_spmodal(){
    $('#modal2').modal("show")
    mode.value = "selfplay"
}

function hide_spmodal(){
    $('#modal2').modal("hide")
}

function choose_white() {
    color.value = "White"
    white.style.backgroundColor =
"rgb(255,215,0)";
    black.style.backgroundColor = "white";
}

function choose_black() {
    color.value = "Black"
    black.style.backgroundColor =
"rgb(255,215,0)";
    white.style.backgroundColor = "white";
}
```

Αρχείο: selfplay.js

```
let starting_fen =
document.getElementById("starting_fen").inn
erHTML;
let reset_text =
document.getElementById("reset_text")
let start_button =
document.getElementById("startBtn")
let model1 =
document.getElementById("model1").innerHTML
;
let model2 =
document.getElementById("model2").innerHTML
;
let player1 =
document.getElementById("player1");
let player2 =
document.getElementById("player2");
let history =
document.getElementById("history");
let scrollbox =
document.getElementById("scrollbox");
let minutes1 =
document.getElementById("minutes1");
let minutes2 =
document.getElementById("minutes2");
let seconds1 =
document.getElementById("seconds1");
let seconds2 =
document.getElementById("seconds2");
let t1box =
document.getElementById("t1box");
let t2box =
document.getElementById("t2box");
let timer1;
let timer2;
let time_increment = 10;
let snd =
document.getElementById("myAudio");

let board = ChessBoard('board', {
   position: starting_fen,
   draggable: true,
   showNotation: true,
   moveSpeed: 'fast',
   snapbackSpeed: 500,
   snapSpeed: 100
});
let isgameover = false;
let reset = false;

set_names();
let moves = 0;
let turn = 1;

function reset_board(){

    start_button.disabled = true
    reset_text.style.display = "block";
    reset = true;
    isgameover = false;
    board.position('start')
    clearInterval(timer1);
    t1box.style.backgroundColor =
"#C8C8C8";
    t1box.style.boxShadow = "none";
    clearInterval(timer2);
    t2box.style.backgroundColor =
"#C8C8C8";
    t2box.style.boxShadow = "none";

    fetch('/reset', {

     headers: {'Content-Type':
'application/json'},
     method: 'POST',
     body: JSON.stringify({"reset":
"true"})
     }).then(function (response) {
         return response.text();
     }).then(function (text) {
         console.log('POST response: ');
         console.log(text);
     });

     history.innerHTML = "Move History:";
     moves = 0;
     minutes1.innerHTML = "10";
     minutes2.innerHTML = "10";
     seconds1.innerHTML = "00";
     seconds2.innerHTML = "00";
     setTimeout(() => {
start_button.disabled = false;
reset_text.style.display = "none"; },
5000);
     clearInterval(timer1);
     clearInterval(timer2);
     history.innerHTML = "Move History:";

}

async function fetch_move(){
    reset = false;
    isgameover = false;
    start_button.disabled = true

    if(reset==true){
            board.position('start')
            console.log("reset1")
            reset_board();
            return;
    }

    if (turn === 1){
        timer2 = setInterval(() => {
start_timer(seconds2, minutes2, 1); },
1000);
        t2box.style.backgroundColor =
"white";
        t2box.style.boxShadow = "2px 3px
#696969";
    }
    else{
        timer1 = setInterval(() => {
start_timer(seconds1, minutes1, -1); },
1000);
        t1box.style.backgroundColor =
"white";
        t1box.style.boxShadow = "2px 3px
#696969";
    }

    if (isgameover === false && reset ===
false){

       await fetch('/selfmove')
           .then(async function (response)
{
               return await
response.json();
           }).then(function (text) {
               if(text.isgameover == false
&& reset==false){
                   console.log('GET
response:');
                   console.log(text.move);

console.log(text.player);
                   console.log(text.fen);

console.log(text.isgameover)
                   board.position(text.fen)
```

```
                    snd.play();
                }

            if (text.player === "White"){
                moves = moves + 1
                history.innerHTML =
history.innerHTML.concat("<br>" + moves +
". " + text.move)
                scrollbox.scrollTop =
scrollbox.scrollHeight;
                clearInterval(timer2);
                add_increment(seconds2,
minutes2, time_increment)
                t2box.style.backgroundColor
= "#C8C8C8";
                t2box.style.boxShadow =
"none";
            }
            else{
                history.innerHTML =
history.innerHTML.concat("  " + text.move)
                clearInterval(timer1);
                add_increment(seconds1,
minutes1, time_increment)
                t1box.style.backgroundColor
= "#C8C8C8";
                t1box.style.boxShadow =
"none";
            }

            turn = turn*(-1);
            isgameover = text.isgameover;
            if (isgameover === false &&
reset === false){
                fetch_move();
            }
            //else{
            //    reset_board();
            //}
    });
    }

    if (isgameover === true){
        await fetch('/selfmove')
         .then(async function (response) {
             return await response.json();
         }).then(function (text) {
             console.log('GET response:');
             console.log(text.result);
             console.log(text.msg);

             history.innerHTML =
history.innerHTML.concat("<br> <b>" +
text.msg + "</b>")
             scrollbox.scrollTop =
scrollbox.scrollHeight;
             clearInterval(timer1);
             clearInterval(timer2);
    });
    }

}

function set_names(){

    let player1_title = "PCE Self-Trained
Model"
    let player2_title = "PCE Self-Trained
Model"

    if (model1 == "2"){
        player1_title = "PCE Data-Trained
Model";
    }
    else if (model1 == "3"){
        player1_title = "PCE Combined
Model";
    }
    else if (model1 == "4"){
        player1_title = "Stockfish Chess
Engine";
    }
    if (model2 == "2"){
        player2_title = "PCE Data-Trained
Model";
    }
    else if (model2 == "3"){
        player2_title = "PCE Combined
Model";
    }
    else if (model2 == "4"){
        player2_title = "Stockfish Chess
Engine";
    }

    player2.innerHTML = player1_title;
    player1.innerHTML = player2_title;
}

function start_timer( secs, mins, color){
    s = secs.innerHTML - 1;
    m = mins.innerHTML;

    if (s>9){
        secs.innerHTML = secs.innerHTML -
1;
    }
    else if(s>-1 && s<10){
        secs.innerHTML = "0" +
(secs.innerHTML - 1);
    }
    else{
        if(m>1 && m<10){
            mins.innerHTML = "0" +
(mins.innerHTML - 1);
            secs.innerHTML = 59;
        }
        else if(m>1){
            mins.innerHTML = mins.innerHTML
 - 1;
            secs.innerHTML = 59;
        }
        else if( m == 1){
            mins.innerHTML = "00";
            secs.innerHTML = 59;
        }
        else{
            mins.innerHTML = "00";
            secs.innerHTML = "00";
            clearInterval(timer1);
            clearInterval(timer2);
            t2box.style.backgroundColor =
"#C8C8C8";
            t2box.style.boxShadow = "none";
            t1box.style.backgroundColor =
"#C8C8C8";
            t1box.style.boxShadow = "none";

            if(color === 1){
                history.innerHTML =
history.innerHTML.concat("<br> <b>White out
of time. Black Wins!</b>")
                scrollbox.scrollTop =
scrollbox.scrollHeight;
            }
            else if(color === -1){
                history.innerHTML =
history.innerHTML.concat("<br> <b>Black out
of time. White Wins!</b>")
                scrollbox.scrollTop =
scrollbox.scrollHeight;
```

```
            }

            isgameover = true;
            clearInterval(timer1);
            clearInterval(timer2);
        }
    }

}

function add_increment(seconds, minutes,
increment){

    s = parseInt(seconds.innerHTML);
    if ((s + increment) < 60){
        seconds.innerHTML = parseInt(s) +
increment;
    }
    else{
        diff = parseInt(s + increment -
60);
        if (minutes.innerHTML < 9){
            minutes.innerHTML = "0" +
parseInt(parseInt(minutes.innerHTML) + 1);
        }
        else{
            minutes.innerHTML =
parseInt(parseInt(minutes.innerHTML) + 1);
        }
        seconds.innerHTML = "0" +
parseInt(diff);
    }
}
```

Αρχείο: versusplay.js

```
let starting_fen =
document.getElementById("starting_fen").inn
erHTML;
let first_legal_moves =
document.getElementById("first_moves").inne
rHTML

.replace("[","").replace("]","").replace(/'
/gi, "").split(", ");
let orientation =
document.getElementById("player_color").inn
erHTML;
let legal_moves = get_first_moves();
let reset_button =
document.getElementById("resetBtn");
let sur_button =
document.getElementById("surrenderBtn");
let opponent =
document.getElementById("opponent").innerHT
ML;
let player1 =
document.getElementById("player1");
let history =
document.getElementById("history");
let scrollbox =
document.getElementById("scrollbox");
let minutes1 =
document.getElementById("minutes1");
let minutes2 =
document.getElementById("minutes2");
let seconds1 =
document.getElementById("seconds1");
let seconds2 =
document.getElementById("seconds2");
let t1box =
document.getElementById("t1box");
let t2box =
document.getElementById("t2box");
let timer1;
let timer2;
let time_increment = 10;
let snd =
document.getElementById("myAudio");

let moves = 0;
let isgameover = false;
let turn = true;
let reset = false;

let board = ChessBoard('board', {
   position: starting_fen,
   draggable: true,
   showNotation: true,
   moveSpeed: 'fast',
   snapbackSpeed: 500,
   snapSpeed: 100,
   onDragStart: onDragStart,
   onDrop: onDrop,
   orientation: orientation,
   onMouseoverSquare: onMouseoverSquare,
   onMouseoutSquare: onMouseoutSquare
});

set_names();
white_move_first();


function reset_board(){
   reset = true;
   reset_button.disabled = true;
   reset_text.style.display = "block";
   isgameover = false;
   turn = true;
   clearInterval(timer1);
   t1box.style.backgroundColor = "#C8C8C8";
   t1box.style.boxShadow = "none";
   clearInterval(timer2);
   t2box.style.backgroundColor = "#C8C8C8";
   t2box.style.boxShadow = "none";

   fetch('/reset', {
    headers: {'Content-Type':
'application/json'},
    method: 'POST',
    body: JSON.stringify({"reset": "true"})
    }).then(function (response) {
        return response.text();
    }).then(function (text) {
        console.log('POST response: ');
        console.log(text);
    });

    setTimeout(() => { start_new_game(); },
5000);
    moves = 0;
}

function start_new_game(){

    board.position(starting_fen)
    reset_text.style.display = "none";
    reset = false;
    reset_button.disabled = false;
    if(orientation === "white"){
        legal_moves = first_legal_moves;
    }
    else{
        legal_moves = [];
        white_move_first();
    }

    minutes1.innerHTML = "10";
    minutes2.innerHTML = "10";
    seconds1.innerHTML = "00";
    seconds2.innerHTML = "00";

}

function surrender(){

    if (orientation === "white"){
        history.innerHTML =
history.innerHTML.concat("<br> <b> White
surrendered. Black Wins! </b>")
    }

    else if (orientation === "black"){
        history.innerHTML =
history.innerHTML.concat("<br> <b> Black
surrendered. White Wins! </b>")
    }

    scrollbox.scrollTop =
scrollbox.scrollHeight;
    reset_board();
}


function onDragStart(source, piece,
position, orientation){

    if ((orientation === "white" && turn
=== false) || (orientation === "black" &&
turn === true)){
        console.log('snapback2')
        return
    }
    if(isgameover === true){
        return false}
    if((orientation === "white" &&
piece.search(/^w/) === -1) ||
```

```
       (orientation === "black" &&
piece.search(/^b/) === -1)){
       console.log('snapback3')
        return false
    }
}

function onDrop(source, target, piece,
newPos, oldPos, orientation){

    removeGreySquares();
    if(reset==true){
        board.position(starting_fen)
        return;
    }

    let move = source.concat(target);

    if(orientation === "white" &&
piece.search(/P/) === 1 &&
target.search(/8/) === 1){
        move = move + 'q';
    }

    if(orientation === "black" &&
piece.search(/P/) === 1 &&
target.search(/1/) === 1){
        move = move + 'q';
    }

    if(!legal_moves.includes(move) ||
isgameover || (orientation === "white" &&
turn === false) ||
      (orientation === "black" && turn ===
true)){
      console.log('snapback1')
        return 'snapback'
    }


    console.log("Move: ", move)

    fetch('/post_move', {
        headers: {'Content-Type':
'application/json'},
        method: 'POST',
        body: JSON.stringify({"move":
move})
        }).then(function (response) {
            return response.text();
        }).then(function (text) {
            console.log('POST response: ');
            console.log(text);
            if(orientation === "white"){
                turn = false;
            }
            else{
                turn = true;
            }


    })
    clearInterval(timer2);
    add_increment(seconds2, minutes2,
time_increment)
    t2box.style.backgroundColor =
"#C8C8C8";
    t2box.style.boxShadow = "none";

    GET_data();

}

async function GET_data(){

    reset_button.disabled = true;
    [isgameover, turn] = await GET_board();
    if(isgameover===false){
        [isgameover, turn] = await
GET_engine_move();
    }

    if(isgameover===true ||
legal_moves.length == 0){
        GET_result();
    }
    reset_button.disabled = false;
}

function onMouseoverSquare (square, piece)
{

    if ((orientation === "white" && turn
=== false) || (orientation === "black" &&
turn === true)){
        return
    }

    let square_moves = [];
    for(i=0; i<legal_moves.length; i++){
        source =
legal_moves[i].substring(0,2);
        if(source === square){

square_moves.push(legal_moves[i])
        }
    }

    if (square_moves.length === 0) return

    greySquare(square)
    for (let i = 0; i <
square_moves.length; i++) {

greySquare(square_moves[i].substring(2,4))
    }

}

let whiteSquareGrey = '#a9a9a9'
let blackSquareGrey = '#696969'

function greySquare (square) {
    var $square = $('#board .square-' +
square)

    var background = whiteSquareGrey
    if ($square.hasClass('black-3c85d')) {
      background = blackSquareGrey
    }

    $square.css('background', background)
}

function removeGreySquares () {
  $('#board .square-
55d63').css('background', '')
}

function onMouseoutSquare (square, piece) {
    removeGreySquares()
}

async function GET_board(){

    if(reset==true){
        board.position(starting_fen)
        return;
    }

    await fetch('/get_player_move')
        .then(async function (response) {
            return await response.json();
```

```
        }).then(function (text) {
            if(text.isgameover == false &&
reset==false){
                board.position(text.fen)
                snd.play();
                console.log('GET
response:');
                console.log(text.move);
                console.log(text.player);
                console.log(text.fen);

console.log(text.isgameover)
            }

        isgameover = text.isgameover;
        if(moves === 0){
            history.innerHTML = "Move
History:"
        }
        if(orientation === "white"){
            moves = moves + 1
            turn = false;
            history.innerHTML =
history.innerHTML.concat("<br>" + moves +
". " + text.move)
            scrollbox.scrollTop =
scrollbox.scrollHeight;
        }
        else{
            turn = true;
            history.innerHTML =
history.innerHTML.concat("  " + text.move)
        }

    });

    return [isgameover, turn] ;

}

async function GET_engine_move(){

    if(reset==true){
        board.position(starting_fen)
        return;
    }

    if(orientation === "white"){
        timer1 = setInterval(() => {
start_timer(seconds1, minutes1, "black");
}, 1000);
    }
    else{
        timer1 = setInterval(() => {
start_timer(seconds1, minutes1, "white");
}, 1000);
    }
    t1box.style.backgroundColor = "white";
    t1box.style.boxShadow = "2px 3px
#696969";



    await fetch('/get_engine_move')
         .then(async function (response) {
             return response.json();
         }).then(await function (text) {
             if(text.isgameover == false &&
reset==false){
                 board.position(text.fen)
                 snd.play();
                 console.log('GET
response:');
                 console.log(text.move);
                 console.log(text.player);
                 console.log(text.fen);

console.log(text.isgameover)
             }

             isgameover = text.isgameover;
             clearInterval(timer1);
             add_increment(seconds1,
minutes1, time_increment)
             t1box.style.backgroundColor =
"#C8C8C8";
             t1box.style.boxShadow =
"none";
             legal_moves =
text.legal_moves;

             if(moves === 0){
                 history.innerHTML = "Move
History:"
             }
             if(orientation === "white"){
                 turn = true;
                 history.innerHTML =
history.innerHTML.concat("  " + text.move)
             }
             else{
                 moves = moves + 1;
                 turn = false;
                 history.innerHTML =
history.innerHTML.concat("<br>" + moves +
". " + text.move)
                 scrollbox.scrollTop =
scrollbox.scrollHeight;

             }
             timer2 = setInterval(() => {
start_timer(seconds2, minutes2,
orientation); }, 1000);
             t2box.style.backgroundColor =
"white";
             t2box.style.boxShadow = "2px
3px #696969";
         });

    return [isgameover, turn] ;

}

async function GET_result(){

    await fetch('/get_result')
         .then(async function (response) {
             return await response.json();
         }).then(function (text) {
             console.log('GET response:');
             console.log(text.result);
             console.log(text.msg);

             history.innerHTML =
history.innerHTML.concat("<br> <b>" +
text.msg + "</b>")
             scrollbox.scrollTop =
scrollbox.scrollHeight;
             clearInterval(timer1);
             clearInterval(timer2);
        });

    return;
}


function get_first_moves(){

    let legal_moves = [];
    if(orientation === "white"){
        legal_moves = first_legal_moves;
    }

    return legal_moves;
}
```

```
async function white_move_first(){

    if (reset==true){
        return;
    }

    reset_button.disabled = true;
    if(orientation === "black"){
        [isgameover, turn] = await
GET_engine_move();
    }
    reset_button.disabled = false;

}

function set_names(){
    let opponent_title = "PCE Self-Trained
Model"

    if (opponent == "2"){
        opponent_title = "PCE Data-Trained
Model";
    }
    else if (opponent == "3"){
        opponent_title = "PCE Combined
Model";
    }
    else if (opponent == "4"){
        opponent_title = "Stockfish Chess
Engine";
    }
    player1.innerHTML = opponent_title;
}

function start_timer( secs, mins, color){
    s = secs.innerHTML - 1;
    m = mins.innerHTML;

    if (s>9){
        secs.innerHTML = secs.innerHTML -
1;
    }
    else if(s>-1 && s<10){
        secs.innerHTML = "0" +
(secs.innerHTML - 1);
    }
    else{
        if(m>1 && m<10){
            mins.innerHTML = "0" +
(mins.innerHTML - 1);
            secs.innerHTML = 59;
        }
        else if(m>1){
            mins.innerHTML = mins.innerHTML
- 1;
            secs.innerHTML = 59;
        }
        else if( m == 1){
            mins.innerHTML = "00";
            secs.innerHTML = 59;
        }
        else{
            mins.innerHTML = "00";
            secs.innerHTML = "00";
            clearInterval(timer1);
            clearInterval(timer2);
            t2box.style.backgroundColor =
"#C8C8C8";
            t2box.style.boxShadow = "none";
            t1box.style.backgroundColor =
"#C8C8C8";
            t1box.style.boxShadow = "none";

            if(color === "white"){
                history.innerHTML =
history.innerHTML.concat("<br> <b>White out
of time. Black Wins!</b>")
                scrollbox.scrollTop =
scrollbox.scrollHeight;

            }
            else if(color === "black"){
                history.innerHTML =
history.innerHTML.concat("<br> <b>Black out
of time. White Wins!</b>")
                scrollbox.scrollTop =
scrollbox.scrollHeight;
            }

            isgameover = true;
            clearInterval(timer1);
            clearInterval(timer2);
        }
    }
}

function add_increment(seconds, minutes,
increment){

    s = parseInt(seconds.innerHTML);
    if ((s + increment) < 60){
        seconds.innerHTML = parseInt(s) +
increment;
    }
    else{
        diff = parseInt(s + increment -
60);
        if (minutes.innerHTML < 9){
            minutes.innerHTML = "0" +
parseInt(parseInt(minutes.innerHTML) + 1);
        }
        else{
            minutes.innerHTML =
parseInt(parseInt(minutes.innerHTML) + 1);
        }
        seconds.innerHTML = "0" +
parseInt(diff);
    }
}
```

Σημείωση: Για την δημιουργία της σκακιέρας επιπλέον χρησιμοποιήθηκαν τα υπάρχοντα αρχεία: chessboard-1.0.0.min.js (https://github.com/oakmac/chessboardjs/, Copyright (c) 2019, Chris OakmanReleased under the MIT license https://github.com/oakmac/chessboardjs/blob/master/LICENSE.md) και jquery.min.js ((c) JS Foundation and other contributors | jquery.org/license */)