



Πανεπιστήμιο Πελοποννήσου
Τεχνητή Νοημοσύνη
Εαρινό εξάμηνο 2024

Αναφορά Πρώτης Εργασίας

Γιαννόπουλος Γεώργιος
ΑΜ:2022202000039

Γιαννόπουλος Ιωάννης
ΑΜ:2022201900032

Διδάσκοντας Μαθήματος: Ιωάννης Νικολέντζος

Μάιος, 2024

Περιεχόμενα

1	Εισαγωγή	3
2	Λεπτομέρειες Υλοποίησης	3
2.1	Main.py	3
2.2	Game Module	4
2.3	Players Module	4
2.4	Monte Carlo Module	4
2.5	Gamestate Module	5
2.6	Εκτέλεση Προγράμματος	5
3	Τρίλιζα	6
3.1	Τυχαίος Παίκτης	6
3.2	Πριόνισμα α - β	7
3.3	Αναζήτηση Δένδρου Monte Carlo	8
4	Reversi	9
4.1	Αλγόριθμος MiniMax	9
4.2	Τυχαίος παίκτης	9
4.3	Ευρετική συνάρτηση αξιολόγησης	10
4.4	Πριόνισμα α - β με περιορισμό βάθους	11
5	Επίλογος	12

1 Εισαγωγή

Η παρούσα αναφορά αποτελεί την αξιολόγηση μας στην πρώτη εξαμηνιαία εργασία του μαθήματος *Τεχνητή Νοημοσύνη*. Κατά την διάρκεια του εξαμήνου διδαχθήκαμε διάφορους αλγορίθμους αναζήτησης που χρησιμοποιούνται στο πεδίο της Τεχνητής Νοημοσύνης (TN) και ως στόχο για την πρώτη εργασία ήταν η υλοποίηση γνωστών αλγορίθμων για παίγνια δύο αντιπάλων. Πιο συγκεκριμένα υλοποιήσαμε τους αλγορίθμους **MiniMax**, **Πριόνισμα α - β** , **Αναζήτηση δέντρου Monte Carlo**, καθώς και **Πριόνισμα α - β με περιορισμό βάθους** χρησιμοποιώντας ευρετική συνάρτηση που μας δόθηκε σε μορφή αλγορίθμου και υλοποιήθηκε προγραμματιστικά από εμάς.

Ως γλώσσα προγραμματισμού για την υλοποίηση της εργασίας επιλέχθηκε η *Python* καθώς είμαστε εξοικειωμένοι με αυτήν και χρησιμοποιείτε ευρέως στην υλοποίηση εφαρμογών TN. Παρακάτω θα αναλύσουμε την διαδικασία που ακολουθήσαμε για την υλοποίηση της εργασίας καθώς και μία εκτενή αξιολόγηση των αποτελεσμάτων για τον κάθε αλγόριθμο.

Η ανάπτυξη των αλγορίθμων για αντίπαλους παίκτες σε αυτά τα δύο κλασικά παιχνίδια αποτέλεσε μια σημαντική ευκαιρία να εφαρμόσουμε τις γνώσεις που αποκτήσαμε κατά τη διάρκεια του μαθήματος, πειραματιζόμενοι με την ανάπτυξη λύσεων σε πραγματικά σενάρια.

2 Λεπτομέρειες Υλοποίησης

Προτού αρχίσουμε με την αξιολόγηση των αποτελεσμάτων, αξίζει να αναφερθούμε στην δομή της εργασίας μας καθώς δεν είναι υλοποιημένη σε Jupyter Notebook αλλά σε αρχεία *Python(.py)*¹. Η δομή που ακολουθείτε είναι η εξής:

```
AI_Project1
├── game
│   ├── game.py
│   ├── reversi.py
│   └── tic_tac_toe.py
├── gamestate
│   └── gamestate.py
├── monte_carlo
│   └── monte_carlo_tree_search.py
├── players
│   └── players.py
└── main.py
```

2.1 Main.py

Αποτελεί το κυρίως αρχείο της υλοποίησης. Σε αυτό το αρχείο γίνονται τα απαραίτητα `import` οι συναρτήσεις που θα χρησιμοποιήσουμε από τα άλλα αρχεία που έχουμε δημιουργήσει.

¹Ο κώδικας περιέχει εκτενή σχόλια στις υλοποιημένες συναρτήσεις και μεθόδους

2.2 Game Module

Ο φάκελος *game* περιέχει τρία αρχεία. Το αρχείο *game.py* μας έχει δωθεί από τον διδάσκοντα του μαθήματος και αποτελείτε από την αφηρημένη κλάση **Game** η οποία υλοποιείτε στα άλλα δύο αρχεία.

Το αρχείο *tic_tac_toe.py* είναι και αυτό επίσης έτοιμο από τον διδάσκοντα και περιέχει την κλάση **TicTacToe** που υλοποιεί την αφηρημένη κλάση.

Το αρχείο *reversi.py* είναι και αυτό επίσης έτοιμο από τον διδάσκοντα και περιέχει την κλάση **Reversi** που υλοποιεί την αφηρημένη κλάση. Ακόμα έχουμε προσθέσει σε αυτό το αρχείο μεθόδους για τον υπολογισμό της ευρετικής συνάρτησης και την αναζήτηση με περιορισμό βάθους, τα οποία θα εξηγήσουμε στην ενότητα 4.3.

2.3 Players Module

Ο φάκελος *players* περιέχει το αρχείο *players.py* το οποίο έχει υλοποιημένες συναρτήσεις των αλγορίθμων που μας ζητήθηκαν ως παίκτες. Περιέχει τις παρακάτω συναρτήσεις²:

- `manual_player`
- `minmax_player`
- `random_player`
- `alpha_beta_player`
- `mcts_player`
- `alpha_beta_cutoff_player`

2.4 Monte Carlo Module

Ο φάκελος αυτός περιέχει την bonus υλοποίηση της αναζήτησης δένδρου Monte Carlo. Περιέχει την κλάση **MCTNode** καθώς και τις συναρτήσεις

- `ucb`
- `select`
- `expand`
- `simulate`
- `backpropagate`
- `monte_carlo_tree_search`³

²Οι συναρτήσεις `manual_player`, `minmax_player` μας είχαν δοθεί έτοιμες

³Η οποία καλείτε από τον `mcts_player`

2.5 GameState Module

Περιέχει την υλοποίηση του namedtuple **GameState** ώστε να είναι διαθέσιμο σε όλα τα αρχεία που το χρειάζονται.

2.6 Εκτέλεση Προγράμματος

Το πρόγραμμα μας μπορεί να τρέξει εύκολα μέσα από τερματικό⁴

```
cd AI_Project1
python main.py
```

Listing 1: Εντολές Εκτέλεσης Προγράμματος μέσα από τερματικό

Η μόνη βιβλιοθήκη που δεν είναι προεγκατεστημένη είναι η *NumPy* η οποία μπορεί να εγκατασταθεί εύκολα με την παρακάτω εντολή:

```
pip install numpy
```

Listing 2: Εγκατάσταση βιβλιοθήκης NumPy

Έπειτα εμφανίζεται το παρακάτω menu για την εκτέλεση των ερωτημάτων:

Εργασία Μαθήματος Τεχνητής Νοημοσύνης

Τα ερωτήματα εκτελούνται σειριακά εκτός του ερωτήματος 3.1 που είναι το παιχνίδι του manual player VS minimax για το Reversi.

Πληκτρολογήστε

- (1) για τα ερωτήματα της τρίλιζας
 - (2) για τα ερωτήματα του Reversi
 - (3) για το ερώτημα 3.1 και έπειτα `ctrl+c` για την διακοπή του.
- Οτιδήποτε άλλο για έξοδο

Ποιό παιχνίδι θέλεις να παίζεις?

- (1) Tic Tac Toe
- (2) Reversi
- (3) Question 3.1

Πληκτρολόγησε τον αριθμό του παιχνιδιού »

Όταν τελειώνει ένα ερώτημα εμφανίζεται ένα μήνυμα που περιμένει οποιαδήποτε είσοδο, π.χ `enter`, ώστε να μπορούμε να εξετάσουμε τα αποτελέσματα του ερωτήματος. Το ερώτημα 3.1 θα πρέπει να το σταματήσει ο χρήστης πιέζοντας `ctrl+c`.

⁴Υποθέτουμε ότι έχουμε μεταβεί στην τοποθεσία που είναι αποθηκευμένος ο φάκελος και είναι εγκατεστημένη η Python

3 Τρίλιζα

Από εδώ και κάτω ακολουθούν οι απαντήσεις μας στα ερωτήματα που μας ανατέθηκαν για τον σχολιασμό των αποτελεσμάτων που λαμβάνουμε από τους παίκτες που δημιουργήσαμε για το παιχνίδι της Τρίλιζα.

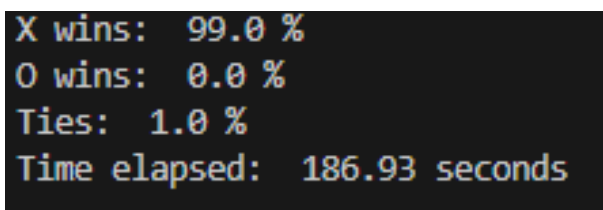
3.1 Τυχαίος Παίκτης

Για την υλοποίηση του τυχαίου παίκτη έχουμε χρησιμοποιήσει το `numpy.random.choice`, χωρίς να βάλουμε δικιά μας τιμή στην παράμετρο `p` που διαθέτει καθώς χρησιμοποιεί από προκαθορισμένη τιμή μία ομοιόμορφη κατανομή των τιμών.

Η `random.choice(a, size=None, replace=True, p=None)` συνάρτηση παράγει ένα τυχαίο δείγμα από ένα 1-Δ πίνακα. Σύμφωνα με το επίσημο εγχειρίδιο για την παράμετρο `p` που αφορά την πιθανότητα επιλογής ισχύει:

`p`: 1-D array-like, optional. The probabilities associated with each entry in `a`. If not given, the sample assumes a uniform distribution over all entries in `a`.

Οπότε και βάζουμε τον *Minimax player* εναντίον του τυχαίου παίκτη για 100 παιχνίδια τρίλιζας.



```
X wins: 99.0 %
O wins: 0.0 %
Ties: 1.0 %
Time elapsed: 186.93 seconds
```

Σχήμα 1: Αποτελέσματα από τα 100 παιχνίδια με νικητή τον *Minimax player* σε 99 από αυτά

Παρατηρούμε από την πιο πάνω εικόνα ότι ο *Minimax player* κατάφερε να κερδίσει σε 99 παιχνίδια ενώ μόλις ένα παιχνίδι ήταν ισοπαλία. Φυσικά και ήταν κάτι αναμενόμενο, καθώς ο ένας παίκτης χρησιμοποιεί αλγόριθμο αναζήτησης, ώστε να επιλέξει την καλύτερη κίνηση ενώ ο άλλος παίκτης επιλέγει μια θέση στην τύχη. Ο *Minimax player* χρειάστηκε τις περισσότερες φορές τρεις με τέσσερις κινήσεις για να κερδίσει. Για τον υπολογισμό του χρόνου χρησιμοποιήθηκε η βιβλιοθήκη *time* της *Python*.

Ενάντια σε έναν τυχαίο παίκτη, ο *minimax* αλγόριθμος θα αποδώσει γενικά καλά, επειδή έχει σχεδιαστεί για να ελαχιστοποιεί την πιθανή απώλεια στο χειρότερο σενάριο. Ένας τυχαίος παίκτης, εξ ορισμού, δεν θα κάνει πάντα τις βέλτιστες κινήσεις, και ο *minimax* θα το εκμεταλλευτεί αυτό επιλέγοντας κάθε φορά την καλύτερη κίνηση.

```

      1 2 3
1 X X X
2 . . .
3 . 0 0
'X' won!

      1 2 3
1 X X X
2 . 0 .
3 . . 0
'X' won!

      1 2 3
1 X X X
2 . 0 .
3 0 . .
'X' won!

```

Σχήμα 2: Στιγμιότυπο των παιχνιδιών του *Minimax player* και του τυχαίου παίκτη

3.2 Πριόνισμα α-β

Τώρα αντί για τον *Minimax player* θα χρησιμοποιήσουμε τον *Alpha-Beta player* εναντίον του τυχαίου παίκτη.

```

X wins: 100.0 %
O wins: 0.0 %
Ties: 0.0 %
Time elapsed: 6.83 seconds

```

Σχήμα 3: Αποτελέσματα από τα 100 παιχνίδια με νικητή τον *Alpha-Beta player* και στα 100, σε σημαντικά λιγότερο χρόνο

Παρατηρούμε ότι έχει κερδίσει σε όλα τα παιχνίδια και αξίζει να αναφέρουμε και τον χρόνο, ο οποίος σε σχέση με πριν είναι αρκετά πιο μικρός.

Ο αλγόριθμος πριονίσματος α-β δεν επιστρέφει πάντα την ίδια ενέργεια που θα επέστρεφε ο αλγόριθμος *Minimax* σε μια αντίστοιχη θέση του παιχνιδιού. Ο στόχος του αλγορίθμου α-β είναι να εξετάσει το λιγότερο δυνατό πλήθος καταστάσεων, αποφεύγοντας τον έλεγχο κινήσεων με αξία μεγαλύτερη από την προηγούμενη. Το πλεονέκτημα του αλγορίθμου

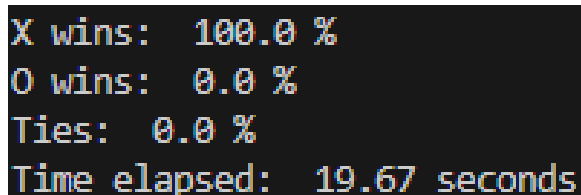
μου πριονίσματος α - β σε σχέση με τον αλγόριθμο Minimax είναι ότι μειώνει σημαντικά τον αριθμό των καταστάσεων που πρέπει να εξετάσει, χωρίς να χάνει την ικανότητα να βρίσκει την βέλτιστη κίνηση. Αυτό συμβάλλει στην απόδοση του αλγορίθμου σε παιχνίδια όπως η τρίλιζα, καθώς μπορεί να εξετάσει λιγότερες πιθανές κινήσεις και να επιλέξει την πιο αποτελεσματική.

3.3 Αναζήτηση Δένδρου Monte Carlo

Η στρατηγική που ακολουθείτε από την αναζήτηση δένδρου Monte Carlo είναι ότι μέσα από ένα πλήθος προσομοιώσεων ολοκληρωμένων παιχνιδιών επιλέγονται οι κινήσεις για τον ένα παίκτη και έπειτα για τον άλλο και αυτό συνεχίζεται μέχρι να φτάσει σε μία κατάσταση τερματισμού.

Στην δική μας υλοποίηση έχουμε στο αρχείο `monte_carlo/monte_carlo_tree_search.py` στο οποίο έχουμε δημιουργήσει μία κλάση που αναπαριστά τους κόμβους στο δένδρο. Έπειτα η κύρια συνάρτηση είναι η `monte_carlo_tree_search()` που παίρνει ως ορίσματα την κατάσταση του παιχνιδιού, το παιχνίδι και τον αριθμό των επαναλήψεων έχει ως τιμή 1000. Αρχικά βάζουμε την κατάσταση του παιχνιδιού ως την ρίζα του δένδρου, χρησιμοποιώντας την κλάση που έχουμε δημιουργήσει. Έπειτα μπαίνουμε στον βρόχο των επαναλήψεων, χρησιμοποιώντας την παράμετρο της συνάρτησης για το πλήθος της. Οπότε και είμαστε έτοιμοι να ακολουθήσουμε τα βήματα του αλγορίθμου της αναζήτησης δένδρου Monte Carlo.

Τοποθετούμε τον παίκτη Monte Carlo, `mcts_player`, να παίζει και αυτός εκατό παιχνίδια εναντίον του τυχαίου παίκτη.



```
X wins: 100.0 %
O wins: 0.0 %
Ties: 0.0 %
Time elapsed: 19.67 seconds
```

Σχήμα 4: Στιγμιότυπο των παιχνιδιών του Monte Carlo player και του τυχαίου παίκτη

Ο αλγόριθμος αναζήτησης Monte Carlo πετυχαίνει και αυτός με την σειρά του παρόμοια αποτελέσματα με τον minimax σε λίγο περισσότερο χρόνο. Καταφέρνει να νικάει τον τυχαίο παίκτη τις περισσότερες φορές με τις ελάχιστες κινήσεις που απαιτούνται ώστε να κερδίσει, δηλαδή τρεις.

Αυτό που κάνει τον αλγόριθμο του Monte Carlo τόσο ξεχωριστό και δημοφιλή είναι το γεγονός ότι βασίζεται στις τυχαίες προσομοιώσεις που πραγματοποιεί για την εξερεύνηση του δένδρου και δεν αναζητά διεξοδικά το δένδρο όπως ο minimax. Ακόμα ο MCTS μπορεί να εφαρμοστεί στα παιχνίδια χωρίς την γνώση των κανόνων του παιχνιδιού και των στρατηγικών του. Μαθαίνει από το μηδέν μέσω επαναλαμβανόμενων προσομοιώσεων και βελτιώνει την απόδοσή του με την πάροδο του χρόνου.

Συνολικά, ο συνδυασμός απλότητας, προσαρμοστικότητας και αποτελεσματικότητας του MCTS στην εξερεύνηση χώρων καταστάσεων τον καθιστά έναν ιδιαίτερο και ευρέως χρησιμοποιούμενο αλγόριθμο σε προβλήματα λήψης αποφάσεων.

4 Reversi

Παρακάτω ακολουθούν οι απαντήσεις και οι παρατηρήσεις για το παιχνίδι Reversi(Othello). Πρόκειται για ένα παιχνίδι πιο στρατηγικό σε σχέση με την τρίλιζα, που απαιτεί πιο προσεκτικές επιλογές κινήσεων. Αυτό μαζί και με το γεγονός ότι κάθε δεδομένη στιγμή ο παίκτης έχει περισσότερες επιλογές κινήσεων, λόγω του μεγαλύτερου ταμπλό, καθιστά την εύρεση της καλύτερης δυνατής κίνησης πιο χρονοβόρα.

4.1 Αλγόριθμος MiniMax

Παίζοντας αντίπαλοι με τον `minimax_player` στο παιχνίδι Reversi παρατηρούμε ότι ο αλγόριθμος καθυστερεί πολύ να πραγματοποιήσει κάποια κίνηση. Αυτό συμβαίνει επειδή ο αλγόριθμος Minimax εξερευνά ολόκληρο το δέντρο του παιχνιδιού μέχρι τις τελικές καταστάσεις, λαμβάνοντας υπόψη όλες τις πιθανές κινήσεις και αντι-κινήσεις. Το Reversi έχει υψηλό συντελεστή διακλάδωσης, που σημαίνει ότι υπάρχουν πολλές πιθανές κινήσεις για κάποια δεδομένη κατάσταση του παιχνιδιού, γεγονός που καθιστά το δέντρο εξαιρετικά μεγάλο και πολύπλοκο.

Αυτό αποτελεί και κάτι που περιμέναμε, καθώς από την θεωρία γνωρίζουμε ότι ο αλγόριθμος `minimax` έχει $O(b^d)$ πολυπλοκότητα χρόνου, όπου:

- b είναι ο παράγοντας διακλάδωσης ο οποίος αντιπροσωπεύει τον μέσο αριθμό πιθανών κινήσεων ανά θέση.
- d είναι το βάθος του δένδρου και αναπαριστά τον αριθμό των κινήσεων που θα εξετάσει ο αλγόριθμος

Ως αποτέλεσμα, ο Minimax θα συνεχίσει να εξερευνά όλο και βαθύτερα το δέντρο μέχρι να φτάσει στις τερματικές καταστάσεις (τέλος του παιχνιδιού), γεγονός που μπορεί να απαιτήσει τεράστιο χρόνο και υπολογιστικούς πόρους, και τελικά μπορεί να μην κάνει κάποια κίνηση μέσα σε εύλογο χρονικό διάστημα, ειδικά καθώς το παιχνίδι εξελίσσεται και ο αριθμός των πιθανών κινήσεων αυξάνεται.

4.2 Τυχαίος παίκτης

Έχοντας υλοποιημένο τον τυχαίο παίκτη από πριν, θα αρχικοποιήσουμε δύο τυχαίους παίκτες ώστε να έρθουν αντιμέτωποι σε εκατό παιχνίδια και να δούμε τα αποτελέσματα τους.

```
Time elapsed: 2.50 seconds  
  
X wins: 21.0 %  
O wins: 39.0 %  
Ties: 40.0 %
```

Σχήμα 5: Αποτελέσματα από τα 100 παιχνίδια του τυχαίου παίκτη εναντίον του τυχαίου παίκτη

Παρατηρούμε ότι τα 100 παιχνίδια έχουν ολοκληρωθεί πολύ γρήγορα, καθώς οι παίκτες απλά επιλέγουν μία έγκυρη κίνηση στην τύχη. Από τα εκατό παιχνίδια νικητής ο πρώτος τυχαίος παίκτης έχει βγει σε 21 από αυτά, ενώ ο δεύτερος τυχαίος παίκτης σε 39 από αυτά. Ακόμα 40 παιχνίδια έληξαν ισοπαλία. Φυσικά κάθε φορά που τρέχουμε το πρόγραμμα τα ποσοστά αυτά αλλάζουν.

4.3 Ευρετική συνάρτηση αξιολόγησης

Η ευρετική συνάρτηση έχει υλοποιηθεί μέσα στην κλάση `Reversi`, η οποία βρίσκεται στην διαδρομή `game/reversi.py`. Η ευρετική συνάρτηση είναι αποτέλεσμα αθροίσματος πολλών ευρετικών συναρτήσεων. Περιέχει τις μεθόδους:

- `countTiles`
- `countCorners`
- `proximityCorners`
- `calcMobility`
- `calcDiscs`
- `heuristic_score`⁵

Η κάθε μέθοδος πραγματοποιεί τον συγκεκριμένο υπολογισμό από την εκφώνηση της εργασίας. Η χρήση της ευρετικής θα πραγματοποιηθεί αργότερα στο κεφάλαιο 4.4.

Τα μειονεκτήματα της χρήσης μιας ευρετικής συνάρτησης σε παιχνίδια συνήθως σχετίζονται με την ποιότητα της εκτίμησης που παρέχει καθώς και τον υπολογιστικό φόρτο που επιβάλλει. Ορισμένα από τα μειονεκτήματα περιλαμβάνουν:

1. **Αύξηση πολυπλοκότητας:** Ορισμένες ευρετικές συναρτήσεις μπορεί να απαιτούν πολύ χρόνο για υπολογισμό, ειδικά αν το παιχνίδι έχει μεγάλο χώρο καταστάσεων ή πολύπλοκους κανόνες.
2. **Δυσκολία Γενίκευσης:** Ο σχεδιασμός μιας ευρετικής συνάρτησης που αποδίδει καλά σε διάφορες καταστάσεις και σενάρια του παιχνιδιού μπορεί να είναι δύσκολος. Οι ευρετικές συναρτήσεις μπορεί να δυσκολεύονται να γενικεύσουν αποτελεσματικά, οδηγώντας σε ασυνεπή απόδοση.

⁵Αποτελεί την μέθοδο υπολογισμού της τελικής τιμής της ευρετικής συνάρτησης

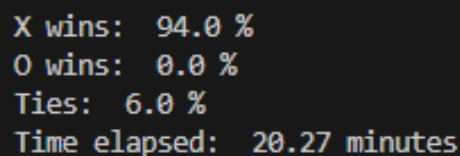
3. **Μεροληψία:** Οι ευρετικές συναρτήσεις βασίζονται σε παραδοχές που ενυπάρχουν στο σχεδιασμό τους. Εάν αυτές οι υποθέσεις δεν αντιστοιχούν στην στρατηγική του παιχνιδιού, η ευρετική συνάρτηση μπορεί να παρέχει μη βέλτιστη καθοδήγηση.
4. **Δυσκολία στον Σχεδιασμό:** ο σχεδιασμός και η υλοποίηση μίας **αποτελεσματικής** ευρετικής συνάρτησης αποτελεί μία περίπλοκη και αρκετά δύσκολη διεργασία.

Η υλοποίηση μίας ευρετικής συνάρτησης χωρίς να έχουμε γνώση των κανόνων του παιχνιδιού, θα οδηγήσει σε μία ευρετική με πολύ περιορισμένη ακρίβεια και ασυνέπεια στην απόδοση της. Δεν καθίσταται αδύνατο, αλλά θα αποτελέσει μία συνάρτηση που θα συνεισφέρει θετικά από καθόλου έως και πάρα πολύ ελάχιστα. Η ευρετική αυτή θα βασίζεται σε γενικές αρχές παρόμοιων παιχνιδιών και σε παραδοχές μας. Για παράδειγμα, η πιο απλή ευρετική συνάρτηση μπορεί να είναι ανάλογα με το πλήθος των πιόνων που έχει ο κάθε παίκτης πάνω στο ταμπλό.

Ωστόσο σε γενικές γραμμές, η υλοποίηση ευρετικής συνάρτησης χωρίς την γνώση των κανόνων του παιχνιδιού, αποτελεί μία λανθασμένη τεχνική και θα πρέπει να αποφεύγεται.

4.4 Πριόνισμα $\alpha - \beta$ με περιορισμό βάθους

Τώρα θα χρησιμοποιήσουμε την ευρετική συνάρτηση για τον υπολογισμό της ωφέλειας της κάθε κίνησης και μέγιστο βάθος δένδρου τρία ώστε να υλοποιήσουμε το πριόνισμα $\alpha - \beta$ με περιορισμό βάθους. Ο παίκτης `alpha_beta_cutoff_player` καλεί την συνάρτηση `alpha_beta_cutoff_search` και χρησιμοποιεί το σύμβολο 'X'.



```
X wins: 94.0 %
O wins: 0.0 %
Ties: 6.0 %
Time elapsed: 20.27 minutes
```

Σχήμα 6: Αποτελέσματα από τα 100 παιχνίδια του $\alpha - \beta$ παίκτη εναντίον του τυχαίου παίκτη στο παιχνίδι Reversi

Όπως παρατηρούμε, για την ολοκλήρωση όλων των παιχνιδιών απαιτείτε πολύ μεγαλύτερος χρόνος σε σχέση με τους τυχαίους παίκτες. Ο αλγόριθμος έχει καταφέρει να κερδίσει 94 από τα 100 παιχνίδια, με τα υπόλοιπα 6 να λήγουν ισόπαλα. Ο παίκτης $\alpha - \beta$ σε κάθε παιχνίδι επιλέγει κινήσεις με βάση την ευρετική και μπορούμε εύκολα να δούμε, στο [Σχήμα 7](#), ότι λειτουργεί καθώς πάντα καταλαμβάνει τα πιο σημαντικά κελιά του ταμπλό που είναι οι γωνίες καθώς και σημαντικά τετράγωνα από τον πίνακα βαρών. Λόγω ότι εμφανίζεται μόνο η τελική κατάσταση του ταμπλό δεν μπορούμε να αξιολογήσουμε εάν επιλέγεται η κίνηση που προσφέρει κάθε φορά την μεγαλύτερη κινητικότητα, αλλά αυτό προκύπτει έμμεσα από τον αριθμό των πλακιδίων που κατέχει στο ταμπλό και είναι πάντα μεγαλύτερο από τον τυχαίο παίκτη.

	1	2	3	4	5	6	7	8
1	X	X	X	X	X	X	X	X
2	X	X	O	O	X	O	O	O
3	X	O	X	X	X	X	O	O
4	X	X	X	O	O	O	O	O
5	X	X	O	O	X	X	X	X
6	X	O	X	X	O	X	X	X
7	X	X	X	X	X	X	O	X
8	X	X	X	X	X	X	X	X

'X' won!

Σχήμα 7: Στιγμιότυπο ενός παιχνιδιού του α - β εναντίον του τυχαίου παίκτη

5 Επίλογος

Εν κατακλείδι, η εξαμηνιαία εργασία στην Τεχνητή Νοημοσύνη, η οποία επικεντρώθηκε στην ανάπτυξη αντιπάλων παικτών αποτέλεσε ένα σημαντικό ταξίδι έρευνας και μάθησης. Η υλοποίηση των αλγορίθμων αναζήτησης σε παίγνια δύο αντιπάλων επέτρεψε την πρακτική εφαρμογή των θεωρητικών γνώσεων που αποκτήθηκαν κατά τη διάρκεια αυτού του μαθήματος.

Επιπλέον, πέρα από τις τεχνικές πτυχές, το εγχείρημα αυτό ανέδειξε την κρίσιμη δια-

σταύρωση της θεωρίας και της πράξης σε πραγματικές εφαρμογές. Υπογράμμισε τη σημασία της σύνδεσης της ακαδημαϊκής γνώσης με την πρακτική εφαρμογή για αποτελεσματικές και αποδοτικές λύσεις.

Καθώς το έργο αυτό φτάνει στο τέλος του, σηματοδοτεί όχι μόνο την κορύφωση των προσπαθειών αυτού του εξαμήνου, αλλά χρησιμεύει και ως βήμα προς περαιτέρω εξερεύνησης του τομέα της Τεχνητής Νοημοσύνης. Αυτή η εμπειρία έχει προκαλέσει μια βαθύτερη εκτίμηση για τις ιδιαιτερότητες της ανάπτυξης αλγορίθμων τεχνητής νοημοσύνης. Περιείχε πολύτιμες γνώσεις για τον τρόπο δημιουργίας και λειτουργίας της τεχνητής νοημοσύνης στο πεδίο των παιχνιδιών και στην λήψη αποφάσεων.