Πολυτεχνείο Κρήτης

2η Προγραμματιστική Εργασία

Τεχνητή Νοημοσύνη (ΠΛΗ 417)

Νταουντάκης Σταύρος **2015030037**

Νικολαίδης Δημήτρης **2015030100**

TUC-Ants

Introduction

Όπως περιγράφεται και απο την εκφώνηση, το TUC-ANTS είναι μια παραλλαγή του γνωστού παιχνιδιού Checkers με κάποιες διαφορές. Αυτές οι διαφορές, όπως θα δούμε και παρακάτω, πρέπει να ληφθούν υπ' όψιν εάν θέλουμε να επιτεύξουμε μια καλύτερη στρατηγική για τον agent μας. Στα πλαίσια της εργασίας έγινε υλοποίηση του minimax αλγορίθμου με την δυνατότητα a-b pruning καθώς και quiescence search για την καλύτερη και πιο αποδοτική αναζήτηση του χώρου για καλές κινήσεις.

Python Implementation

Το project μας υλοποιήθηκε σε γλώσσα Python χτίζοντας πάνω στις ήδη υπάρχουσες συναρτήσεις σε κώδικα C που μας δόθηκαν. Οι C συναρτήσεις αυτές καλούνται απευθείας μέσω Python αφότου κάνουμε compile τον κώδικα C σε μορφή shared object. Με τον τρόπο αυτό εγγυόμαστε πως ο agent μας θα είναι συμβατός με τον server του παιχνιδιού ενώ ταυτόχρονα μπορούμε τον επεκτείνουμε με μεγαλύτερη ταχύτητα. Περισσότερες λεπτομέρειες και οδηγίες εκτέλεσης μπορούν να βρεθούν στο αρχείο README.md.

State Space

Για την εξερεύνηση του χώρου καταστάσεων υλοποιήθηκε μια συνάρτηση get_available_moves() η οποία επιστρέφει όλες τις δυνατές κινήσεις(move struct) σε ένα position struct. Για την βελτίωση της αναζήτηση στον χώρο και συγκεκριμένα για την βελτίωση του a-b pruning παρακάτω η συνάρτηση αυτή επιστρέφει τις κινήσεις από τις καλύτερες προς τις χειρότερες. Συγκεκριμένα πρώτα τοποθετούνται οι αιχμαλωσίες με φθίνουσα σειρά βάση του αριθμού των αιχμαλώτων. Στην συνέχεια τοποθετούνται οι κινήσεις που δημιουργούν βασίλισσα. Μετά οι κινησεις που τελειώνουν σε τετράγωνο φαγητού και τέλος επιστρέφονται οι απλές κινήσεις.

Έχοντας στην διάθεση μας όλα τα διαθέσιμα moves με σειρά προτεραιότητας μένει πλέον να υλοποιήσουμε τα successor states, δηλαδή copies του object gamePosition όπου στο κάθε ένα έχει εφαρμοστεί μια απο τις διαθέσιμες κινήσεις. Η *successor_states()* θα χρησιμοποιηθεί όπως θα δούμε και παρακάτω από τον minimax για την εξερεύνηση του χώρου.

Evaluation Function

Σε πρώτο βήμα υλοποιήσαμε την απλή συνάρτηση αξιολόγησης που μας προτάθηκε απο την εκφώνηση της άσκησης. Έτσι είχαμε ένα πρώτο βήμα για να υλοποιήσουμε και τον αλγόριθμο minimax. Για την την συνάρτηση χρειάστηκε να έχουμε στην διάθεση μας κάποιες πληροφορίες (π.χ. Αριθμό μυρμηγκιών, βασιλισσών) τις οποίες για να τις εξάγουμε παρατηρούμε τις αλλαγές στο board μετά από κάθε πιθανή κίνηση με κατάλληλες συναρτήσεις.

Το evaluation function μας, κατα την πορεία της εργασίας, δέχθηκε δύο "βελτιώσεις". Σε κάθε βήμα βελτίωσης συγκρίναμε την απόδοση του agent σε σχέση με την προηγούμενη "γενιά". Όλα τα στατιστικά παρέχονται σε επόμενη παράγραφο. Οι δύο βασικές βελτιώσεις που δέχθηκε η E.V. ήταν :

- Is in danger : Εξετάζουμε αν μια πιθανή κίνηση θα φέρει ένα μυρμήγκι σε κίνδυνο. Δηλαδή, εάν η κίνηση αυτή θα σκοτώσει το μυρμήγκι μόλις παίξει ο αντίπαλος. Φυσικά, μια τέτοια μέτρηση ρίχνει το evaluation του State.

- Move en Masse : Προτιμάμε κινήσεις οι οποίες θα καλύψουν αβοήθητα μυρμήγκια. Με άλλα λόγια, θέλουμε ένα μυρμήγκι, εφ' όσον είναι ικανό, να πηγαίνει πίσω από ένα άλλο ώστε να μην είναι εκτεθειμένο.

Η στρατηγική προσέγγιση για τον agent μας μπορεί να χαρακτηριστεί ώς **αμυντική.** Απο την στιγμή που ένα μυρμήγκι μετα που γίνει βασίλισσα **δεν** επιστρέφει στο παιχνίδι (όπως στο κανονικό checkers) δεν υπάρχει λόγος να υπάρχει βιασύνη άρα και επιθετικό παίξιμο.

Minimax & a-b pruning

Για την υλοποίηση του αλγορίθμου minimax χρησιμοποιήσαμε ως reference τον ψευδοκώδικα που βρήκαμε online. Το value που μας επιστρέφει δίνει μια αποτίμηση για μια υποψήφια κίνηση. Συγκρίνοντας με όλες τις υποψήφιες κινήσεις επιλέγουμε φυσικά αυτή με την μεγαλύτερη αποτίμηση.

Έπειτα προχωρήσαμε στην προσθήκη της δυνατότητας **a-b pruning** (με παροχή on-off switch). Εξαιτίας της σειράς με την οποία παρέχουμε τις υποψήφιες κινήσεις στον αλγόριθμο το pruning λειτουργεί καλύτερα άρα καταφέραμε να μειώσουμε αρκετά τον χρόνο εκτέλεσης του αλγορίθμου. Περαιτέρω αναφορά στο performance θα γίνει και παρακάτω.

Minimax improvement

Για την βελτίωση της αναζήτησης υλοποιήσαμε **quiescence search**. Ο αλγόριθμος **quiescence search** έχει στόχο να βελτιώσει την αναζήτηση κοιτάζοντας τις κινήσεις του αντιπάλου μετά από τις κινήσεις στα φύλλα του δέντρου μας που φαίνονται καλές πολύ καλές αλλά στην πραγματικότητα επιφέρουν μεγαλύτερες απολαβές στον αντίπαλο σε βάθος χρόνου. Συγκεκριμένα αν μια κίνηση στα φύλλα του

δέντρου είναι αιχμαλώτιση μυρμηγκιου θεωρείται πολύ καλή από τον minimax και θα μας ωθήσει προς αυτό το μονοπάτι. Αυτή η κίνηση όμως μετά μπορεί να στήνει στον αντίπαλο μια αιχμαλώτιση περισσότερων μυρμηγκιών κάνοντας έτσι την αρχική κίνηση στρατηγικά λάθος. Η υλοποίηση μας ελέγχει τέτοιες κινήσεις και έτσι βοηθάει να περιορίσουμε το Horizon Effect από την αναζήτηση συγκεκριμένου βάθους.

Statistics & Performance

Παρακάτω βλέπουμε μερικά Performance χαρακτηριστικά διαφόρων agent. Όλα τα παρακάτω test έγιναν για βαθος 6 με 200 match με αλλαγή χρώματος κάθε φορά. Σε γενικες γραμμες ο agent μας παίρνει απόφαση μέσα 510ms κατά μέσο όρο.

Agent	alphabeta	qsearch	indanger	enmasse
AgeAnt	Yes	Yes	Yes	Yes
basicAgent	Yes	No	No	No
inDangerAgent	Yes	No	Yes	No
nQsearchAgent	Yes	No	Yes	No
nAlphaBetaAgent	No	Yes	Yes	Yes

Πίνακας με breakdown του τι τρέχει ο κάθε agent. alphabeta αναφέρεται στο alpha-beta pruning, qsearch σε quiescence search και indanger και enmasse στα αντίστοιχα evaluation function improvements.

Agent	0	1	2	3	4	5	QSearch	Total
AgeAnt	2586	3517	14391	25266	108544	191822	17814	363942
	0.8%	1%	3.9%	6.9%	29.8%	52.7%	4.9%	100%
basicAgent	2503	3838	17232	30627	140953	251280	0	446436
	0.6%	0.8%	3.8%	6.9%	31.6%	56.3%	0%	100%
inDangerAgent	2499	3554	15359	27164	123100	219035	0	390712
	0.6%	0.9%	3.9%	7%	31.5%	56.1%	0%	100%
nQsearchAgent	2589	3637	15481	27326	122139	216953	0	388127
	0.7%	0.9%	4%	7%	31.5%	55.9%	0%	100%
nAlphaBetaAgent	2591	13914	110440	936251	8267063	75252378	1194251	85776891
	0.003%	0.02%	0.1%	1.1%	9.6%	87.7%	1.4%	100%

Πίνακας με αριθμούς καταστάσεων που κάνει expand ο agent για το κάθε επίπεδο και τα ποσοστά από το σύνολο που αποτελούν.

Matchup	P1 win %	P2 win %	Draw %
basicAgent VS AgeAnt	7.5%	89%	3.5%
inDangerAgent VS AgeAnt	41.5%	57.2%	1%
nQsearchAgent VS AgeAnt	54%	45%	1%

Πίνακας στατιστικών μερικών παιχνιδιών μεταξύ agent.

Conclusion

Από τα παραπάνω αποτελέσματα αρχικά βλέπουμε την απίστευτη βελτίωση που έχει η εφαρμογή alpha-beta pruning στην εξερεύνηση του χώρου κάνοντας την εφαρμογή του αλγόριθμου minimax για παιχνίδια όπως το checkers με μεγάλο state space εφικτή. Επίσης βλέπουμε πως οι βελτιώσεις στο evaluation function και ο καλύτερος καθορισμός και ανάθεση βαθμολογίας σε κάθε κατάσταση βοηθάει στην βελτίωση περαιτέρω της απόδοσης του αλγορίθμου μειώνοντας τις καταστάσεις που χρειάζεται να εξεταστούν και αυξάνοντας αυτές που γίνονται prune μέσω alpha-beta pruning.

Ταυτόχρονα βέβαια βλέπουμε τους κινδύνους της μείωσης του χώρου καταστάσεων. Η εισαγωγή της υλοποίησης της quiescence search μας μειώνει τον χώρο καταστάσεων αλλά ταυτόχρονα προκαλεί τον agent να χάνει ενάντια σε έναν agent που δεν την χρησιμοποιεί. Αυτό ευθύνεται σε πιθανό λάθος στην λογική υλοποίησης το οποίο προκαλεί την qsearch να δίνει χαμηλή εκτίμηση σε καταστάσεις που στην πραγματικότητα ίσως δεν είναι κακές οι οποίες στην συνέχεια θα αφαιρεθούν από την αναζήτηση.