Tuc Ants

Δελατόλας Θάνος 2016030074

Βουρτσάνης Μάνος 2016030076

Εισαγωγή

Για την υλοποίηση ενός agent που παίζει ικανοποιητικά το παιχνίδι TUC Ants υλοποιήθηκαν οι τεχνικές: minimax , alpha-beta pruning , quiescence search , transposition table, MTDF και iterative deepening. Θα τις αναλύσουμε όλες καθώς επίσης θα αναλύσουμε το λόγο που εντάχθηκε η κάθε μια στον agent. Οι κώδικες των : minimax , alpha-beta pruning MTDF και iterative deepening είναι βασισμένοι στους αντίστοιχους ψευδοκώδικες της Wikipedia.

Minimax

Αρχικά υλοποιήθηκε ο απλός minimax. Μέσα σε λογικά πλαίσια χρόνου έφτανε μέχρι βάθος 4. Επιπλέον έχανε καμία φορά από τον random αλγόριθμο. Λόγω της χαμηλής επίδοσης του, τον αναπτύξαμε σε alpha-beta pruning.

Alpha – Beta pruning

Αποτελεί το πρώτο στάδιο κλαδέματος που υλοποιήθηκε για να πετύχει ο agent αναζήτηση σε μεγαλύτερο βάθος. Ο agent σταματάει να ερευνά μια κίνηση εάν αποδεδειγμένα αυτή η κίνηση θα είναι χειρότερη από μια άλλη που έχει ήδη αναλυθεί. Αυτό υλοποιείται με τις μεταβλητές alpha και beta οι οποίες έχουν το χαμηλότερο score του maximizer και το μικρότερο score του minimizer αντίστοιχα. Καθώς η αναζήτηση προχωράει όποια κίνηση δεν βρίσκεται σε αυτό το διάστημα κλαδεύεται. Η συνάρτηση alpha-beta δέχεται εκτός από τα γνωστά ορίσματα και έναν δείκτη σε Position και έναν δείκτη σε Move. Η Position αποτελεί το παιχνίδι την δεδομένη στιγμή και χρησιμοποιείται από την evaluate_function, που θα αναλυθεί στη συνέχεια. Στον δείκτη Move γίνεται return by reference η απόφαση του agent. Σε λογικά πλαίσια χρόνου ο αλγόριθμος έφτανε σε βάθος 5-6. Αξίζει να σημειωθεί πως στο δεδομένο παιχνίδι το μέγιστο βάθος είναι 11.

Evaluate Function και Quiescence search

Η evaluate function είναι απλή και η λογική της είναι:

- Κερδίζονται πόντοι για κάθε μυρμήγκι.
- Χάνονται πόντοι για κάθε μυρμήγκι του αντιπάλου.
- Κερδίζονται πόντοι ανάλογα με το πόσο βαθιά (τόσοι όσοι το νούμερο της γραμμής στο χάρτη) βρίσκεται το κάθε μυρμήγκι.
- Χάνονται πόντοι ανάλογα με το πόσο βαθιά είναι τα μυρμήγκια του αντιπάλου.
- Κερδίζονται τόσοι το σκορ με συντελεστή 90.
- Χάνονται όσοι το σκορ του αντιπάλου με συντελεστή 90.

Το quiescence search εξασφαλίζει πως η Position που θα αξιολογηθεί είναι stable δηλαδή κανένα μυρμήγκι του agent δεν κινδυνεύει από αντίπαλο μυρμήγκι. Αυτό έχει ως αποτέλεσμα να γίνεται καλύτερη αξιολόγηση στην Position.

Ο agent ενθαρρύνει τα μυρμήγκια να πηγαίνουν βαθιά χωρίς να κινδυνεύουν, δεν κάθεται να περιμένει τον αντίπαλο και έτσι είναι λίγο πιο επιθετικός. Έχουμε συντελεστή 90 στο σκορ για να του δώσουμε μεγάλη βαρύτητα εφόσον περιέχει τα φαγητά και τις βασίλισσες. Τέλος προστατεύονται τα μυρμήγκια χάρη στο quiescence και στους πόντους που κερδίζονται για κάθε μυρμήγκι.

Transposition Table

Αποφασίσαμε να δημιουργήσαμε έναν transposition table επειδή μια Position μπορεί να προκύψει από δυο ή και παραπάνω σύνολα κινήσεων. Έτσι ψάχνουμε πρώτα να δούμε αν μια Position υπάρχει στο transposition table και αν δεν υπάρχει εκτελείται κανονικά ο minimax. Αν υπάρχει η Position τότε γίνεται return η αξιολόγηση της ή περιορίζεται το παράθυρο a, b.

Δεν είναι δυνατόν να κρατήσουμε όλες τις Position αλλά κρατάμε αρκετές λόγω της μεθόδου Zobrist hashing. Αντί να αποθηκεύουμε έναν 2-D πίνακα που αναπαριστά την Position αποθηκεύομε έναν unsigned long αριθμό. Καταλαβαίνουμε πως η εξοικονόμηση μνήμης είναι τεράστια. Επιπλέον η σύγκριση δυο long είναι πολύ πιο γρήγορη από τη σύγκριση δυο 2-D πινάκων.

Η Position μετατρέπεται σε unsigned long κάνοντας bitwise xor διάφορους τυχαίους αριθμούς με τον board και με το score της Position (οι τυχαίοι αριθμοί δημιουργούνται και αποθηκεύονται σε έναν πίνακα). Ο unsigned long που δημιουργείται λέγεται Zobrist key. Είναι σχεδόν απίθανο δυο διαφορετικές Position να δημιουργήσουν το ίδιο Zobrist key.

Μαζί με το Zobrist key αποθηκεύονται οι μεταβλητές : lowerBound, upperBound, lowerDepth, upperDepth, type. Οι μεταβλητές αυτές είναι για τον MTDF, αρχικά ο transposition table έσωζε μόνο το Zobrist key το βάθος της Position και την evaluation της.

Τέλος το transposition table δεσμεύει 1.9 gb για να μπορεί να αποθηκεύσει αρκετές Position. Αν δεν καταφέρει να δεσμευτεί αυτός ο χώρος ο agent θα τερματίσει. Μπορείτε να μειώσετε το μέγεθος του table από το αρχείο transposition.h αλλάζοντας τιμή στη σταθερά TABLE_SIZE. Για να βελτιωθεί παραπάνω η υλοποίηση μας θα μπορούσαμε να είχαμε φτιάξει μια καλή replacement strategy.

Αλλαγή στον alpha beta

Σε αυτό το στάδιο αλλάξαμε τον alpha beta. Αρχικά ψάχνει αν υπάρχει αποθηκευμένη στον transposition table η Position, αν υπάρχει και το βάθος της είναι μεγαλύτερο ή ίσο από το βάθος που βρίσκεται ο alpha beta τότε γυρνάμε την evaluation που έχει η αποθηκευμένη Position. Αν δεν ισχύει η παραπάνω συνθήκη εκτελείται κανονικά ο αλγόριθμος και σώζεται η Position μαζί με την evaluation και το βάθος της στον table. Έτσι ο αλγόριθμος γίνεται πολύ πιο γρήγορος από τον κλασσικό alpha beta.

MTD(f)

Ο MTD(f) είναι ένας αρκετά πιο γρήγορος και αποδοτικός αλγόριθμος από τον alpha-beta. Είναι πιο αποδοτικός γιατί κάνει zero window alpha beta searches γύρω από τις τιμές που εκτιμά πως βρίσκεται το αποτέλεσμα. Χρησιμοποιεί δύο μεταβλητές το lowerBound και το upperBound που αρχικά έχουν τιμές : -∞ και +∞ αντίστοιχα. Όσο το lowerBound είναι μικρότερο γίνονται alpha beta searches στο παράθυρο b-1, b. Αν η τιμή που επιστρέφεται από τον alpha-beta είναι μικρότερη του beta τότε το upperBound παίρνει την επιστρεφόμενη τιμή αλλιώς το loweBound.

Επειδή ο MTD(f) κάνει αναζητήσεις σε τόσο μικρό παράθυρο γίνονται πολλά pruning και έτσι ο αλγόριθμος είναι αρκετά πιο γρήγορος. Εφόσον το παράθυρο είναι σωστό είναι και πολύ πιο αποδοτικός, στη περίπτωση που το παράθυρο είναι λανθασμένο ο αλγόριθμος θα έχει πολύ κακό αποτέλεσμα (διότι θα κλαδεύονται κινήσεις που είναι καλές).

Για την υλοποιήσει του απαιτείται transposition table. Εφόσον είχαμε transposition table αποφασίσαμε να προσθέσουμε MTD(f) στον agent.

Χρειάστηκε να αλλάξουμε τον transposition table γιατί έπρεπε να αποθηκεύουμε τις μεταβλητές lowerBound, upperBound, lowerDepth, upperDepth, type.

Οι μεταβλητές αναπαριστούν:

- lowerBound: Το κάτω όριο από τον alpha beta.
- upperBound: Το πάνω όριο από τον alpha beta.
- lowerDepth: Το βάθος που βρέθηκε το κάτω όριο στον alpha beta.
- upperDepth: Το βάθος που βρέθηκε το πάνω όριο στον alpha beta.
- type: Αν η απόφαση του alpha beta:
    - είναι μικρότερη ή ίση του alpha τότε: type = 0x5 (101) (upper transposition)

- είναι μεγαλύτερη του alpha και μικρότερη του beta τότε: type = 0x7 (111) (exact transposition)
- είναι μεγαλύτερη ή ίση του beta τότε: type = 0x3 (011) (lower transposition)

(type = 0 όταν είναι ελεύθερη η θέση στον transposition table)

Αναπαριστούμε το type έτσι (αντί να έχουμε ένα enum) για να κάνουμε γρηγορότερα τις συγκρίσεις, παίρνοντας παράδειγμα από τον Zobrist.

Με βάση το type προκύπτει :

- type & 0x2 => not upper
- type & 0x4 => not lower
- type & 0x1 => not empty

Αλλαγή στον alpha beta

Στο στάδιο αυτό προσαρμόσαμε την αλλαγή που έγινε στον transposition table. Αν δεν υπάρχει αποθηκευμένη η Position στον transposition table τότε σώζεται ως upper ή lower ή exact.

Αν υπάρχει ή περιορίζεται το παράθυρο ή επιστρέφεται κατευθείαν η evaluation της ή εκτελείτε κανονικά ο alpha beta αν το βάθος της είναι μικρότερο από αυτό που βρίσκεται ο alpha beta.

- Αν δεν είναι upper και το lowerBound είναι μεγαλύτερο ή ίσο του beta τότε επιστρέφεται το lowerBound. Το beta έχει πάρει τιμή από τον MTD(f) και αφού είναι μεγαλύτερο, ο MTD(f) θα βάλει στο beta τη τιμή loweBound με αποτέλεσμα να μειωθεί το παράθυρο που ψάχνει ο MTD(f).
- Αντίστοιχα, αν δεν είναι lower και το upperBound είναι μικρότερο του alpha, επιστρέφεται το upperBound διότι στον MTD(f) το upperBound θα γίνει alpha και το παράθυρο θα μειωθεί.
- Αν δεν είναι upper αλλά το lowerBound είναι μικρότερο του beta δεν υπάρχει λόγος να επιστραφεί κάτι. Μπορούμε όμως να μειώσουμε το παράθυρο που ψάχνει ο alpha beta. Το παράθυρο μειώνεται αναθέτοντας στο alpha τη μεγαλύτερη τιμή ανάμεσα στα alpha και lowerBound.
- Αντίστοιχα, αν δεν είναι lower αλλά το upperBound είναι μεγαλύτερο του alpha τότε μειώνουμε το παράθυρο που ψάχνει ο alpha beta, αναθέτοντας στο beta τη μικρότερη τιμή μεταξύ των beta και upperBound.

Iterative deepening

Τέλος για να γίνει πιο αποδοτικός ο MTD(f) αλλά και για να ελέγχουμε το βάθος που θα φτάσει ο agent με βάση το χρόνο υλοποιήθηκε ένα απλό iterative deepening. Όσο υπάρχει διαθέσιμος χρόνος τρέχει τον MTD(f) με αυξανόμενο (κατά 1) βάθος αναθέτοντας στη μεταβλητή f του MTD την επιστρεφόμενη τιμή από τον προηγούμενο MTD. Η μεταβλητή f έχει σημαντικό ρόλο στην απόδοση του MTD και είναι μια εκτίμηση (first guess) της αξιολόγηση της πρώτης Position. Χάρη στο iterative deepening δεν μαντεύουμε αλλά αναθέτουμε μια αποδεδειγμένα καλή τιμή στο f μεγιστοποιώντας την απόδοση του MTD(f).

Επιπλέον ιδέες που δεν υλοποιήσαμε

- Καλύτερη evaluate function.
- Opening library. Αν το παιχνίδι ήταν σκάκι θα μπορούσαμε να βρούμε μια αποδεδειγμένα καλή αρχή.
- Move ordering. Θεωρήσαμε πως δεν χρειάζεται εφόσον οι πιθανές κινήσεις δεν είναι τόσο πολλές.
- Machine learning. Δεν γνωρίζουμε τόσο καλά και δεν είχαμε περιθώριο χρόνου να μάθουμε αρκετά καλά.
- Καλύτερη υλοποίηση του transposition table. Καλύτερο replacement strategy αλλά και καλύτερη hash function. Η τωρινή hash function κάνει απλά mod το Zobrist key με το table size.

Compile

Μέσω του makefile, όπως γινόταν compile και ο δοσμένος κώδικας. Κάναμε αλλαγές στο makefile αλλά ο τρόπος που γίνεται compile είναι ίδιος.