Εργασία Δομημένου Προγραμματισμού

Η βασική ιδέα της εργασίας ηταν να προσομειώνω επομένες κινήσεις στο παιχνίδι με σκόπο να βρίσκω κάθε φορά που ήταν βέλτιστη θέση για να παίξω

- 1η Υλοποίηση-for: (εχει λογικά λάθη δεν θα σας στείλω την πρώτη μορφή του)

  Η πρώτη σκέψη ήταν να δημιουργήσω δυναμικά δεντρά με structs αλλά μάλλον δεν αυτή η υλοποίηση δυνατή στη C οπότε χρησιμοποίησα πολλές for με την μια μέσα στην άλλη. Σε κάθε επανάληψη της πρώτης for αντιστοιχούσε ενα root και κάθε επανάληψη των επόμενων for συνιστούσε ενα node. Τα roots και τα nodes αποτελούν τους συνδυασμούς που θα έπαιζαν οι 2 παίχτες. Έτσι κάθε for είχε board->column επαναλήψεις και η πρώτη αντιστοιχούσε στις κινήσεις του παίχτη , η δεύτερη του αντιπάλου και ούτω καθεξής

  Για να μπορέσω να προσομοιώσω τις κινήσεις δημιούργησα μια συνάρτηση για να δημιουργώ ενα αντίγραφο του board και να παίζω της κινήσεις πάνω σε αυτό αξιοποιώντας τις συναρτήσεις που είχε ηδη το παιχνίδι .

  Σε καθε επάλαψη:

  1. Ελέγχω αν είναι δυνατή η κίνηση αξιοποιώντας την CheckMove
  2. Βάζω την κίνηση στο αντίγραφο του πίνακα
  3. Αποθήκευω τα δεδομένα σε 3 arrays μεγέθους (αριθμός στηλών)^(αριθμό των for)=συνολικος αριθμός των επαναλήψεων-συνδυασμών.
  4. Αναιρώ την κίνηση όταν χρειάζεται για να παίξω την επόμενη

  Τα arrays ηταν τα εξής:

  - score: κρατάει τη διαφορά των σκορ των δυο παικτών σε περίπτωση ισοπαλίας σε κάθε συνδυασμό
  - scoremax: κρατάει τη διαφορά των σκορ των δυο παικτών σε καθε συνδυασμό

- scoreIndex: κράταει τη πρώτη κίνηση-root για τα οποία ισχύουν τα προηγούμενα

Οι επιπλέον συναρτήσεις που έφτιαξα είναι οι εξής:

- col_vs_row: Δεν δουλεύει. Σκοπός της ήταν να βρίσκει τις κένες θέσεις κοντά εκεί που παίζει ο παίχτης
- finder: επιστρέφει 1 αν ο παίχτης έχει περισσότερες κενές θέσεις απο τον αντίπαλο 0 αν αν έχουν ίσες και -1 αν ο αντίπαλος έχει περισσότερες
- free_spaces_0: επιστρέφει 1 αν και οι 2 παίχτες εχουν 0 κενές θέσεις πάνω η δίπλα απο το μέγιστο σκορ τους

Αρχικός σκοπός ήταν να συνδύασω τα δεδομένα που θα παρήχαν οι προηγόυμενες συναρτήσεις και τα σκορ σε κάθε συνδυασμό ώστε να βαθμολογώ την κάθε κίνηση και να παίζω στην καλύτερη θέση. Επίσης θα απέριπτα τις κινήσεις-roots που οδηγούσαν σε πολλές ήττες. Ωστόσο δεν κατάφερα να βρώ κάποιον αποδοτικό τρόπο. Αν και μερικές φορές κέρδιζε ο τρόπος που υπολόγιζε το σκορ θεωρούσε ευνοικές και τις πιθανές ΄΄χαζές΄΄ κινήσεις του αντιπάλου. Έτσι έκανα τον αλγόριθμο κοντόφθαλμο (έχει βάθος 4 , δηλαδή 4 for) καθώς το πλήθος των συνδύασμων αυξάνεται εκθετικά με την αύξηση των for και εφόσον δεν βρήκα κάποιον αποδοτικό τρόπο για να εκμεταλλευτώ τα δεδομένα που είχα για καθε συνδυασμό δεν θεώρησα σκόπιμο να έχω πολλές επαναλήψεις. Ο παίχτης ουσιαστικά παίζει πλεόν εκεί που θα αυξήσει το σκόρ του κατα 1. Μερικές συνθήκες που θα έβαζα επιπλέον αν δούλευε η συνάρτηση για τις κενές θέσεις:

Αν το σκορ του παίχτη στο πραγματικό board είναι ίσο με το σκορ του αντιπάλου και το πλήθος των κενών θέσων του παίχτη ήταν λιγότερες απο του αντιπάλου τοτε να παίξει εκεί που αντίπαλος δεν αυξάνει το σκορ του. Έτσι δεν επιτρέπω στον παίχτη να παίζει σε θέσεις που δεν έχουν ΄΄μέλλον΄΄. Το ίδιο συμβαίνει και ο παίτης έχει πετύχει τόσους πόντους οσου και οι στήλες ή σειρές ώστε να μην αφήσω τον αντίπαλο να φτάσει το σκορ μου.

Επιπλέον αν το σκορ του παίχτη είναι ίσο με το σκορ του αντιπάλου και δεν έχει αλλες κενές θέσεις (δηλαδή το παιχνίδι παει για ισοπαλία) ο παιχτής να παίξει εκεί που αυξάνει το δευτερεύων σκορ

Ο αλγόριθμος χωρίς τις επιπλέον συναρτήσεις επιστέφει κανονικά τιμές αλλα δεν ήταν καθόλου ‘’έξυπνος’’. Πχ καμιά φορα έπαιζε σε σειρές ενώ αντίπαλος σε στήλες αλλά το μέγιστο σκόρ που θα έφτανε θα ήταν μικρότερο απο το σκορ του αντιπάλου.

- 2η Υλοποίση-negamax:

Επειδή ο πρώτος αλγόριθμος δεν εκμεταλλευόταν τη ‘’διορατικότητα’’ του προσπάθησα να βρω βιβλιογραφία για το πως να δημιουργήσω έναν τέτοιο αλγόριθμο. Έτσι βρήκα μια έκδοση ενός min-max αλγορίθμου τον negamax. Ο negamax ουσιαστικά είναι ένας αναδρομικός αλγόριθμος που και αυτός προσομοιώνει τις επόμενες κινήσεις και βασιζόμενος στο πίο είναι το μέγιστο σκορ, τις κινήσεις που εχουν παιχτεί, το σκορ του αντιπάλου και του παίχτη προβλέπει αν στις επόμενες κινήσεις ο παίχτης χάνει, κερδίζει η θα φέρει ισοπαλία και παίζει ανάλογα. Για να τον υλοποιήσω δημιούργησα μία αναδρομική συνάρτηση που προσομοιώνε περιπτώσεις , τις βαθμολογούσε με μια συνάρτηση εκμεταλλευόμενος τα προηγούμενα δεδόμενα και επέλεγε την καλύτερη κίνηση. Αυτό προυπέθετε βέβαια να κάνεις πάρα πολλές προσομοιώσεις και η δυναμικότητα του board δημιουργούσε προβλήματα. Αν και φαίνεται να είναι αποδοτικός σε μικρούς πίνακες σε μεγάλους δεν μπορεί να τρέξει καν το πρόγραμμα. Ακόμα και αν περιόριζα την αναδρομικότητα του για να παίζει σε μεγάλους πίνακες τότε δεν ήταν αποδοτίκος καθώς δεν προλάβαινε να τρέξει όσες φορές χρειαζόταν. Επίσης δεν βρήκα κάποιο τρόπο για να εισάγω την περίπτωση ισοπαλίας στον negamax. Εξαιτίας του γεγονότος οτι δεν καταλάβαινα και πλήρως τον αλγόριθμο δεν ασχόληθηκα περαιτέρω

http://blog.gamesolver.org/solving-connect-four/03-minmax/

- 3η Υλοποίση(Θα σας στείλω τον κώδικα αλλά δεν νομίζω οτι έχει νόημα να τον δείτε, τον έκανα περιττά πολύπλοκο)

Οι δυο προηγούμενοι αλγόριθμοι με οδήγησαν στο συμπέρασμα οτι δύσκολα διαχείριζονται οι πολλές αναδρομικότητες και οι προβλέψεις πόσο μάλλον σε έναν δυναμικό πίνακα και προσπάθησα να φτιάξω έναν πιο απλό αλλα αποτελεσματικό αλγόριθμο χωρίς προβλέψεις που

βασίζεται ουσιαστικά στη συνάρτηση της 1ης υλοποιήσης που υπολόγιζε τις κενές θέσεις. Οι συναρτήσεις αυτή την φορά ήταν οι εξής:

- col_vs_row (spoiler alert: πάλι δεν δουλέυει): είναι η συνάρτηση της πρώτης υλοποίησης αλλά ελαφρώς τροποποιημένη. Αυτή τη φορά επιστέφει 1 αν ο παίχτης έχει το max στις στήλες, 0 αν είναι ίσα και -1 αν είναι στις γραμμές. Παράλληλα κανει call by reference σε δύο μεταβλητές που περιέχουν τα max κενά σε στήλες και σειρές. Η λογική της είναι να υπολογίζει τη μεγαλύτερη σειρά διαδοχικών ψηφίων στα rows, να κρατάει τη θέση του τελευταίου ψηφίου, του πρώτου και το row στο οποίο υπήρχε αυτή. Στη συνέχεια υπολόγιζε στο row που βρήκε πόσα 0 είχε δεξιά και αρίστερα απο το maxrow και τα άθροιζε (αν υπήρχαν). Στα columns όταν βρίσκει μια μέγιστη σειρά κρατάει τη θέση του τελευταίου ψηφίου. Στη συνέχεια ελέγχει της θέση του ψηφίου που βρίσκεται στην ίδια στήλη και στη σειρά που η διαφορά της θέσης του τελευταίου ψηφίου με τη μέγιστη σειρα. Αν είναι είναι 0 τότε οι κενές θέσεις είναι (θέση του τελευταίου ψηφίου +1)-max. Ωστόσο αυτή η συνάρτηση εξακολουθούσε να παρουσίαζει πολλά προβλήματα και τελικά δεν δούλευε

  Παραδείγματα

  Στήλη:

  0

  0

  1

  1

  1

  2

  Ας υποθέσουμε οτι αυτή είναι η στήλη 0. Στη σειρά tempy= 4 βρίσκεται το τελευταία ψηφία της σειράς απο 1 και το maxrow=3. Τώρα ελέγχουμε το ψηφίο board[4-3][0] το οποίο είναι ίσο με 0. Άρα το max_col0=(tempy+1)-maxrow=2

Η συνάρτηση όμως παρουσίαζε πολλά προβλήματα και καθέ φορά που έλυνα κάποιο τα προβλήματα πολλαπλασιάζοταν. Η περιπλοκότητα της συνάρτησης αυξήθηκε δραμάτικα οταν συνειδητοποίησα οτι θα μπορούσαν να υπάρχουν πολλές ίσες σειρές διαδοχικών αριθμών που έχουν το μέγιστο οπότε έπρεπε να επιλέξω τη σείρα που είχε τις μέγιστες κενές θέσεις δίπλα ή απο πάνω της (σε row ή column) και οπότε έπρεπε να αποθηκεύω τα δεδομένα για κάθε σειρά σειρά σε ένα struct*. Το γεγόνος ότι δεν έχω πολύ εμπειρία με τον προγραμματισμό με οδήγησε στην δημιουργία μια εξαιρετικάς πολύπλοκης συνάρτησης που δεν δούλευε, είχε πάρα πολλές μεταβλητές και ακόμα και αν τις εκτύπωνα δεν μπορούσα να βρω το λάθος

- where: χρησιμοποιεί την col_vs_row και επιστρέφει τις μέγιστες κενές θέσεις ανάλογα με το που παίζει μαλλον ο παίχτης. Πχ αν επιστρέφει 1 η col_vs_row αυτή θα επιστρεφεί τα μεγιστα κενά στις στήλες ενώ αν επιστρέφει 0 επιστρέφει το περισσότερα κενά

Δημιουργώ πάλι ενα αντίγραφο του board και δοκιμάζω αρκετές συνθήκες.

- Αν τα μέγιστα κενά του παίχτη μου είναι μεγαλύτερα ή ισα του αντιπάλου χωρίς να είναι 0 μέσω μιας for βλέπω σε ποιά θέση θα μειώθουν τα μέγιστα κενα ώστε να παίξω εκει με την προυπόθεση οτι θα αυξηθεί και το σκορ μου
- Αν τα μέγιστα κενά του παίχτη μου ειναι λιγότερα απο του αντιπάλου τότε παίζω εκεί που θα έπρεπε να παίξει αντίπαλος μου δηλαδή στη θέση με τα περισσότερα κενά και εκεί που θα αυξανόταν το σκορ του αν έπαιζε σε εκείνη τη θεση
- Αν τα μεγίστα κενά του παίχτη μου είναι 0 θα προσπαθήσω να κερδίσω το παιχνίδι σε ισοπαλία. Έτσι μέσω μιας for ελέγχω που θα αυξηθεί το δευτερεύων σκορ σε περίπτωση ισοπαλίας. Βέβαια υπάρχει και μια περίπτωση ο αντίπαλος να πάει να χτίσει και ένα δεύτερο μέγιστο σκορ και να ξεπεράσει το πρώτο και γι αυτό πρώτα ελέγχω τη περίπτωση που πάει να γίνει αυτό για να παίξω σε εκείνη τη θέση

- Επίσης επειδή δεν παίζω διαγώνια και θα ήθελα να αποφύγω το γεγονός να παίξει διαγώνια και να μου στήσει παγίδα αφήνοντας τον να παίζει από πάνω μου οταν του δημιουργώ το κατάλληλο σκαλοπάτι θα προσπαθούσα να αποτρέψω αυτή την περίπτωση(αν υλοποιούσα αυτόν τον αλγόριθμο μάλλον θα ήταν αποδοτικός στον random). Ο τρόπος που σκέφτηκα ήταν ότι αν ο αντίπαλος ήθελε να παίξει στη διαγώνιο θα έπρεπε να παίξει σε μια ακραία θεση και θα περίμενε κάποια στιγμή να παίξω εγώ απο δίπλα του για να του δημιουργούσα το σκαλοπατάκι. Έτσι θα έβαζα προυπόθεση να μην παίζει ποτέ δίπλα στην πρώτη θέση του αντίπαλου μέχρι να αναγκαζόταν να παίξει αυτός

Θεωρώ οτι αν δούλευε η συνάρτηση με τα κενά ο αλγόριθμος θα ήταν αρκετά αποδοτικός. Επίσης επείδη όλα τα return ήταν μέσα σε συνθήκες θα έβαζα στο τέλος σαν δικλείδα ασφαλείας να παίζει ο random. Νομίζω οτι αν είχα ξεκινήσει να δουλεύω την 3^{η} υλοποίηση απο την αρχή και ξαναέφτιαχνα την συνάρτηση με τα κενά χωρίς τις περιττές πολυπλότητες ίσως κατάφερνα να δημιουργήσω έναν ικανοποιητικό αλγόριθμο. Το κυριότερο λάθος μου νομίζω είναι δεν προσπάθησα να δουλέψω πιο απλά και αποτελεσματικά.

- Πίσω στην αρχή

Μετά απο ακόμα μια αποτυχημένη προσπάθεια και λίγες ώρες πριν τη λήξη της προθεσμίας είπα να γυρίσω στον πρώτο αλγόριθμο αφού δούλευε τουλάχιστον χωρίς τις έξτρα συναρτήσεις και να τον τροποποιήσω λίγο. Ειπά να επαναφέρω την αρχική ιδέα περι βαθμολογίας των κινήσεων. Δημιούργησα έναν πίνακα μεγέθους ((αριθμος των for-1)x(στήλες)). Στην τελευταία for βαθμολογούσα την κάθε κατάσταση ως εξής: Στήν κάθε τελευταία κίνηση (που ήταν του αντιπάλου) έβαζα σε -6 αν έχανα -6 αν ήταν ισοπαλία και έχανα +2 αν ήταν ισοπαλία και κέρδιζα και +2 αν κέρδιζα. Στην συνέχεια έβαζα τη βαθμολογία της τελευταίας κίνησης στη σειρά του πίνακα που αντιστοιχούσε στην προτελευταία for και στή στήλη που αντιστοιχούσε η επάναληψη τις προτετευταίας for. Έτσι είχα b->colums βαθμολογίες για κάθε επάναληψης της for που τις έβαζα

στον πίνακα στις αντίστοιχες θέσεις. Στη συνέχεια έβρισκα τό άθροισμα της σειράς και το έβαζα στην κατάλληλη θέση του πίνακα που αντιστοιχούσε στην προηγούμενη for και ούτω καθεξής. Ουσιαστικά κάθε κίνηση οδηγεί σε κάποιες άλλες κινήσεις. Με βάση το αν αυτές οι κινήσεις είναι ωφέλιμες βαθμολογώ την πρώτη κίνηση. Έτσι ξεκινώντας απο τα leaves σιγά σιγά φτάνουμε και σε μια βαθμολογία για τα roots και επιλέγω το root με την καλύτερη βαθμολογία. Σκοπός της βαθμολόγησης ήταν να οδηγούμε στις κινήσεις που έχουν μόνο θετικά αποτελέσματα όποια και ήταν η κίνηση του αντιπάλου για αυτό βαθμόλογησα έτσι και την ήττα.

Αυτός ο αλγόριθμος νικάει αρκετά συχνά ωστόσο δεν πρόλαβα να βγάλω ποσοστά και έχει θέμα στους μεγαλύτερους πίνακες

Χρησιμοποιησα μια συνάρτηση που υπολογίζει το σκορ στον τελευταίο κόμβο. Επειδή είχα αρχικοποίησει τον πινακα με -9999 ήξερα οτι ενα στοιχεία της σειρας(που αναφετεται σε συγκεκριμενη for) του πινακα ειχε αυτη τη τιμη τοτε να μην το λαβω υποψιν. Αν ολα τα στοιχεια της σειράς ηταν -9999 αυτό σήμαινε οτι ήταν τελευταία κίνηση οποτε το loop της προηγούμενης for ηταν leaf και οχι απλά κόμβος αρα ισχύουν για κείνον οι κανόνες που ισχύουν για τα leaves. Όσο έτρεξα τον αλγόριθμο κέρδιζε αλλά λόγω λήξης προθεσμίας δεν πρόλαβα να συλλέξω αρκετά δεδομένα για το καλύτερο scoring system αντε βάλω τις μέγιστες for που μπορούσα.

Σημείωση: Για να αναιρώ την κίνηση καθώς δεν είχα πολύ χρόνο να τροποποιήσω την InsertInto εφτιάχνα ενα αντίγραφο του αντιγράφου (inception) πριν βάλω την κίνηση και μετά όταν χρειαζόταν κατέστρεφε το πρώτο αντίγραφο και το έκανα ίσο με το δεύτερο

Γιακουμόγλου Πασχάλης 10054