



ΕΘΝΙΚΟ ΜΕΤΣΟΒΙΟ ΠΟΛΥΤΕΧΝΕΙΟ

ΣΧΟΛΗ ΗΜ&ΜΥ
Τεχνητή Νοημοσύνη
1^η Άσκηση
Ακ. έτος 2011-2012

Γερακάρης Βασίλης Α.Μ.: 03108092
Λύρας Γρηγόρης Α.Μ.: 03109687

9 Φεβρουαρίου 2012

Υλοποίηση αλγορίθμου A*

1.1 Παρουσίαση προβλήματος και σκιαγράφηση λύσης

Στην άσκηση αυτή καλούμαστε να υλοποιήσουμε τον αλγόριθμο αναζήτησης A* με σκοπό να οδηγηθούν με βέλτιστο τρόπο 2 ρομπότ σε ένα προκαθορισμένο σημείο συνάντησης, αποφεύγοντας τις πιθανές συγκρούσεις.

Έχοντας ως δεδομένο ότι τα δύο ρομπότ έχουν γνώση της θέσης και του πλάνου του άλλου ρομπότ, καθώς και πλήρη γνώση της αίθουσας, επιλέξαμε να κάνουμε μια παρατήρηση /παραδοχή που διευκολύνει σημαντικά την δομή του προγράμματός μας:

Αντί τα 2 ρομπότ να πραγματοποιούν τις κινήσεις/επιλογές τους εναλλάξ, θεωρούμε, χωρίς βλάβη της γενικότητας, ότι στις συγκρούσεις το 2ο ρομπότ θα έχει μια 'nice' συμπεριφορά, παραχωρώντας τη βέλτιστη κίνηση στο 1ο ρομπότ. Με τον τρόπο αυτό, μπορούμε να εκτελέσουμε σειριακά τον αλγόριθμο A*, πρώτα για το 1ο ρομπότ και στη συνέχεια για το 2ο, έχοντας ήδη δεδομένες τις κινήσεις του 1ου.

Ο χώρος των καταστάσεων μας δίνεται ως ένα grid με O (στις θέσεις που επιτρέπεται η κίνηση) και X (στις θέσεις όπου βρίσκονται εμπόδια). Η βασική ιδέα πίσω από τον αλγόριθμό μας είναι ότι οι θέσεις με X κωδικοποιούνται ως '-1', οι θέσεις με O με '0', ενώ η θέση στην οποία θα βρίσκεται το 1ο ρομπότ στο k-οστό βήμα με 'k'.

Με τον τρόπο αυτό, μπορούμε να περιορίσουμε τις επιλογές του 2ου ρομπότ σε κάθε βήμα, αναγκάζοντας το να ψάξει εναλλακτικό μονοπάτι ή να μείνει στάσιμο γι'αυτό το βήμα (αν το συμφέρει). Καταφέρνουμε έτσι να αποφύγουμε τις πιθανές συγκρούσεις, ενώ ταυτόχρονα δε θυσιάζουμε τη βελτιστότητα της λύσης μας.

Για την υλοποίηση του παραπάνω αλγορίθμου, επιλέχθηκε ως κατάλληλη γλώσσα η Python, επειδή μας δίνει τη δυνατότητα να επικεντρωθούμε στα πλέον σημαντικά κομμάτια του προβλήματος (αλγόριθμος και στρατηγική αναζήτησης), να γράψουμε πυκνό και ευανάγνωστο κώδικα, ενώ με τη χρήση ενός Just-in-time compiler όπως ο Psycho έχουμε ίδιο χρόνο εκτέλεσης όπως αν γράφαμε σε μια γλώσσα χαμηλότερου επιπέδου.

1.2 Επιλογή Δομών Δεδομένων

Στην υλοποίησή μας επιλέξαμε να χρησιμοποιήσουμε λίστες λιστών της Python, όπου ο κάθε κόμβος στη λίστα του A* είχε 4 στοιχεία:

1. Το άθροισμα ευριστικής και κόστους ($h + c$)
2. Το κόστος (c)
3. Την τετμημένη του σημείου (x)
4. Την τεταγμένη του σημείου (y)

Επιπλέον, χρησιμοποιείται ένα λεξικό (dict) προκειμένου να πραγματοποιηθεί η αναδόμηση της επιλεγμένης διαδρομής ενός ρομπότ. Τα dictionaries είναι associative arrays, ένας τύπος αντικειμένων που μοιάζει με λίστα, αλλά όπου σε μοναδικά κλειδιά αντιστοιχίζονται (όχι απαραίτητα μοναδικές) τιμές.

1.3 Βασικές συναρτήσεις-τελεστές

2

1.4 Ευριστικές μέθοδοι

Δεδομένου ότι δουλεύουμε σε δισδιάστατο χώρο με μόνο οριζόντια και κάθετη κίνηση (όχι διαγώνια), επιλέχθηκε ως πιο έγκυρος και αποδοτικός υποεκτιμητής (admissible heuristic) η απόστατη Manhattan:

$$M_D = |x_1 - x_2| + |y_1 - y_2|$$

Η χρήση υποεκτιμητή μας εγγυάται τη βελτιστότητα της λύσης.

Αν επιλέγαμε να χρησιμοποιήσουμε ένα υπερεκτιμητή (non-admissible heuristic), θα επεκτείναμε πιθανώς πολύ λιγότερους κόμβους κατά την αναζήτηση μας. Το χρονικό κέρδος αυτό όμως θα αντισταθμιζόταν με την πιθανότητα να μην είναι βέλτιστη η λύση που θα προέκυπτε, ουσιαστικά αναιρώντας ένα από τα χαρακτηριστικά του αλγορίθμου του A*.

1.5 Έλεγχος και επίλυση συγκρούσεων

Όπως εξηγήσαμε πριν, ο αλγόριθμός μας εκτελείται πρώτα για το ένα ρομπότ, και στη συνέχεια για το δεύτερο, σε ένα τροποποιημένο grid. Στην περίπτωση που στο μέτωπο αναζήτησης του 2ου για το k-οστό βήμα είναι η θέση που πήγε το 1ο ρομπότ στο k βήμα, ανιχνεύεται πιθανή σύγκρουση. Το 2ο ρομπότ έχει πλέον 2 επιλογές:

1. Να περιμένει (wait) για ένα γύρο, επιτρέποντας στο 1ο να προχωρήσει και έπειτα ακολουθώντας το (μιας και οι επιλογές του 1ου είναι εξ'ορισμού βέλτιστες), δίνοντας στο σημείο της σύγκρουσης κόστος $c + 1$
2. Να αναζητήσει κάποια εναλλακτική πορεία προς το σημείο συνάντησης.

Από τις 2 αυτές επιλογές, τελικά θα διαλέξει αυτή που έχει το λιγότερο κόστος (σε αριθμό βημάτων)

Εκτελώντας τον αλγόριθμο A* 2 φορές, ώστε στη μία το ρομπότ A να παραχωρεί προτεραιότητα στις συγκρούσεις, ενώ στη 2η το B να παραχωρεί προτεραιότητα και επιλέγοντας αυτό που παράγει μικρότερο αριθμό βημάτων τερματισμού, καταλήγουμε στο τελικά βέλτιστο αποτέλεσμα της αναζήτησης.

Algorithm 1 Conflict resolution

```
1: procedure resolve(newGrid[x][y], currStep)
2:   if newGrid[x][y] == currStep then
3:      $cost_1 \leftarrow (len(Astar((x, y), target)) + 1)$ 
4:      $path_1 \leftarrow (path(Astar((x, y), target)))$ 
5:     newgrid[x][y]  $\leftarrow Invalid$ 
6:      $cost_2 \leftarrow len(Astar(currPos, target))$ 
7:      $path_1 \leftarrow (path(Astar((currPos, target)))$ 
8:     if  $cost_1 < cost_2$  then
9:        $cost \leftarrow cost_1$ 
10:       $path \leftarrow path_1$ 
11:    else
12:       $cost \leftarrow cost_2$ 
13:       $path \leftarrow path_2$ 
```

1.6 Χρόνος εκτέλεσης ανά μέγεθος εισόδου

5

1.7 Εκτύπωση βημάτων-συγκρούσεων-εναλλακτικών δρόμων σε παράδειγμα εκτέλεσης

6

1.8 Πηγαίος κώδικας

Here be dragons!