Πανεπιστήμιο Πειραιώς
Σχολή Τεχνολογιών Πληροφορικής και Τηλεπικοινωνίων
Τμήμα Ψηφιακών Συστημάτων

Επίπεδο: Προπτυχιακό Πρόγραμμα Σπουδών

Μάθημα – Συστήματα Ευφυών Πρακτόρων

Τίτλος - 2η ομαδική εργασία 2 ατόμων (2023-2024)

Επιβλέπον Καθηγητής: Γεώργιος Βούρος, Ναταλία Κόλιου

Ονοματεπώνυμο	E-mail	A.M.
Απόστολος Χασιώτης	apostolis10@pm.me	E20183
Σωτήριος Δημητρακουλάκος	sdimitrakoulakos@gmail.com	E20040

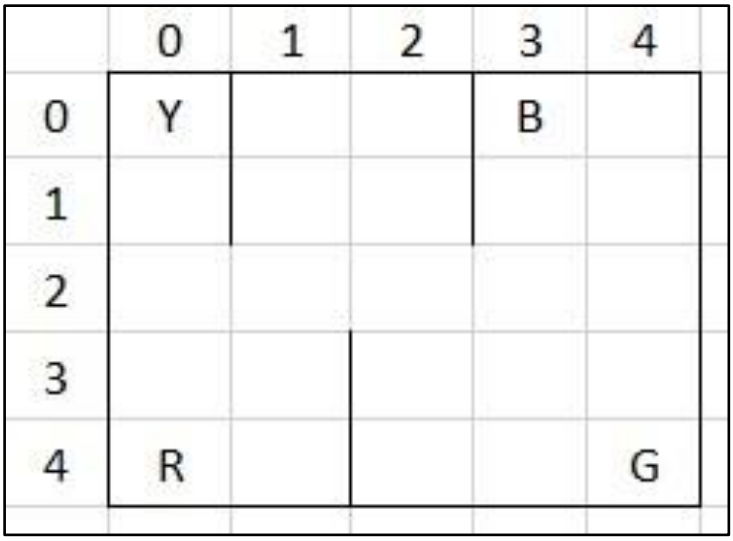
Πειραιάς

11/12/2024

Abstract

Περιεχόμενα

Αρχιτεκτονική



Η παρούσα εργασία αποσκοπεί στην δημιουργία ενός κόσμου, όπου ένας πράκτορας **ταξί**, αναλαμβάνει την μεταφορά πολλαπλών **πελατών** προς τον **προορισμό** τους.
Στόχος μας αποτέλεσε η υλοποίηση ενός συστήματος, στο οποίο ο πράκτορας εφαρμόζει με τον βέλτιστο δυνατό τρόπο (ελαχιστοποίηση του κόστους) την επιλογή του επόμενου πελάτη, καθώς και τη μετακίνηση του στον κόσμο. Επιδιώκει την επιλογή της συντομότερης διαδρομής και παράλληλα μαθαίνει τις βέλτιστες ακολουθίες βημάτων μεταξύ των **τεσσάρων** προκαθορισμένων σημείων (R, Y, G, B). Επίσης, κατά τη μετακίνηση του στον κόσμο, πραγματοποιεί και εκμάθηση των εμποδίων, που έχουν οριστεί (τοίχοι και όρια του κόσμου).
Αφού πραγματοποιηθεί επιτυχώς η φόρτωση και η εκφόρτωση του πελάτη στον προορισμό του, τότε η εξυπηρέτηση του παρόντος πελάτη έχει διεκπεραιωθεί και εκτελείται η όλη διαδικασία εκ νέου. Το επεισόδιο ολοκληρώνεται μετά την εκτέλεση 50 ενεργειών του πράκτορα ταξί, έτσι ώστε να μπορεί να μελετηθεί η δράση του σε πολλαπλές περιπτώσεις και σενάρια.

Στον 5x5 κόσμο στον οποίο έχουμε ορίσει το σύστημα εξυπηρέτησης με ταξί, υπάρχουν καταχωρημένα:

- Όρια του κόσμου
- Εμπόδια-Τοίχοι (όπως παρουσιάστηκαν στην παραπάνω εικόνα)
- Οντότητες για τα κελία (cell)
- Κόστη για κάθε μετάβαση
- Λεξικό, που θα κρατάει τα βέλτιστα μονοπάτια (μεταξύ των 4 διακριτών θέσεων)

Στον κόσμο μας υπάρχουν οι εξής πράκτορες και οντότητες:

- 2 Ταξί (πράκτορες)
- Πελάτες (οντότητα)
- Προορισμός (πράκτορας)
- Τηλεφωνικό Κέντρο (Οντότητα/Συνάρτηση)

Ο **πράκτορας ταξί** εμφανίζεται εξ αρχής σε μία τυχαία θέση από τις 25 θέσεις-κελιά του κόσμου. Μπορεί και μετακινείται καθ' όλη την έκταση του κόσμου μας, προχωρώντας ένα κελί τη φορά.

Από την άλλη, η **οντότητα πελάτης** εμφανίζεται με 50% πιθανότητα, κάθε 0.7 δευτερόλεπτα, σε μία τυχαία από τις 4 διακριτές προκαθορισμένες θέσεις (R, Y, G, B). Συγκεκριμένα, με τη δημιουργία ενός νέου πελάτη, αναπαράγονται μια τιμή τοποθεσίας και μια τιμή προορισμού (τυχαίες από τα R, Y, G, B), οι οποίες αντιστοιχούν στον πελάτη αυτόν. Το σύνολο των πελατών που διατηρείται στο περιβάλλον είναι προσβάσιμο από τον πράκτορα ταξί. Ο μέγιστος αριθμός πελατών κάθε δεδομένη στιγμή δεν ξεπερνά τους τρεις, ενώ παράλληλα ένας πελάτης δεν μπορεί να εμφανιστεί στην τοποθεσία ενός υπάρχοντος πελάτη.

Ο **πράκτορας d** (destination) βρίσκεται κάθε φορά στην τοποθεσία που αποτελεί τον καταχωρημένο προορισμό του τωρινού πελάτη, που εξυπηρετείται.

Πρώτος παράγοντας που επηρεάζει τη επιλογή ενός νέου πελάτη αποτελεί η απόσταση Μανχάτταν, μεταξύ της τοποθεσίας του ταξί και της τοποθεσίας του πελάτη.
Αν βρεθεί μοναδικός πελάτης με την ελάχιστη απόσταση, επιλέγεται αυτός προς εξυπηρέτηση. Όμως, σε περίπτωση εύρεσης ισότιμης ελάχιστης απόστασης, μεταξύ πολλαπλών πελατών, επιλέγεται αυτός με την ελάχιστη απόσταση Μανχάτταν, μεταξύ της τωρινής τοποθεσίας του και του προορισμού του.\

Εύρεση Βέλτιστης Διαδρομής:

- Περίπτωση 1: Το ταξί δεν βρίσκεται σε σημείο εκ των R, Y, G, B:

  Γίνεται χρήση της μεθόδου με την απόσταση Μανχάτταν που προαναφέρθηκε, προσαυξημένη του κόστους ενέργειας μετάβασης, για την επιλογή του επόμενου βήματος, που θα εκτελέσει ο πράκτορας κάθε φορά, κατά τη μετακίνησή του από την τωρινή τοποθεσία του προς τον πελάτη.

- Περίπτωση 2: Όταν το ταξί βρίσκεται σε ένα από τα R, Y, G, B:

  - 2.1: Η βέλτιστη διαδρομή μεταξύ των σημείων είναι γνωστή:

    Ακολουθεί την καταχωρημένη στο λεξικό βέλτιστη διαδρομή, που έχει βρεθεί κατά την εκμάθηση του πράκτορα. (*)

  - 2.2: Το ταξί δεν έχει μάθει ακόμη τη βέλτιστη διαδρομή μεταξύ των σημείων:

    Εκτελεί την ίδια διαδικασία με την **περίπτωση 1**, ενώ ταυτόχρονα γίνεται και εκμάθηση της βέλτιστης διαδρομής. (*)

Γίνεται φόρτωση του πελάτη στο ταξί και ανανεώνεται το αντίστοιχο perception στο περιβάλλον (που δείχνει ότι ο πελάτης βρίσκεται μέσα στο ταξί).
Η μεταφορά του πελάτη γίνεται με τον ίδιο ακριβώς τρόπο και σύμφωνα με τα σενάρια, που υπάρχουν και προαναφέρθηκαν, κατά την διαδικασία **μετακίνησης σε πελάτη**.
Ύστερα, όταν το ταξί φτάσει στον προορισμό d, γίνεται εκφόρτωση του πελάτη στον προορισμό του, ανανέωση του αντίστοιχου perception στο περιβάλλον (που δείνχει ότι ο πελάτης πια βρίσκεται στο προορισμό) και τέλος ο πελάτης αφαιρείται από το περιβάλλον, ενώ το ταξί μπαίνει στη διαδικασία να επιλέξει επόμενο πελάτη.

Κατά τη μετακίνηση του από σημείο σε σημείο (μεταξύ των R, Y, G, B), ο πράκτορας μαθαίνει την βέλτιστη διαδρομή – ακολουθία βημάτων, που πρέπει να ακολουθήσει, για να ελαχιστοποιήσει το κόστος μετακίνησης του (ελάχιστο πλήθος βημάτων, ελάχιστες συγκρούσεις, κλπ.).
Ο μηχανισμός εκμάθησης έχει σχεδιαστεί έτσι, ώστε όταν μια διαδρομή (π.χ. η R-Y/Y-R) είναι άγνωστη, να βρίσκει και να μαθαίνει την βέλτιστη ακολουθία βημάτων, με την πρώτη μετάβασή του, από το ένα διακριτό σημείο στο άλλο.
Η βέλτιστη ακολουθία κινήσεων, αποθηκεύεται στο λεξικό που προαναφέρθηκε, ως ακολουθία γραμμάτων που αντιπροσωπεύουν τις κινήσεις αυτές (π.χ. «KKDDDEE» για το Y-B). Όταν, λοιπόν, εκτελείται η διαδρομή αυτή, διαβάζονται οι κινήσεις μία-μία από το λεξικό και εκτελούνται. Σε περίπτωση της αντίστροφης διαδρομής (B-Y), οι κινήσεις εκτελούνται αντίθετα (για κάθε κίνηση) και αντίστροφα (από το τέλος στην αρχή).

Ο τρόπος που επιλέγεται κάθε φορά η επόμενη κίνηση, από κελί σε κελί, που θα εκτελεστεί και θα καταχωρηθεί στο λεξικό, είναι αυτή που αναφέρθηκε στην **περίπτωση 2.2** της **μετακίνησης σε πελάτη**, αλλά και στην **μεταφορά πελάτη** (Μανχάτταν + Κόστη).
Κατά την εκτέλεση των βημάτων αυτών, υπάρχει περίπτωση ο πράκτορας ταξί να προσκρούσει σε τοίχο. Σε αυτή την περίπτωση, αφού εκτελείται η κίνηση και πηγαίνει στο επόμενο κελί, τότε λόγω πρόσκρουσης με τοίχο, ανανεώνει τα κόστη των μεταβάσεων με την τιμή 101 (του προηγούμενου κελιού προς το επόμενο και του επόμενου προς το προηγούμενο). Κατόπιν, επιστρέφει στο κελί που ήταν, επιλέγει πια

την καλύτερη κίνηση, μεταξύ των υπολοίπων που έμειναν (η προηγούμενη πια δεν είναι η καλύτερη αφού έχει πια κόστος 101) και πρέπει η κίνηση που προστέθηκε τελευταία στο λεξικό, να αφαιρεθεί.

Στην ουσία, η διαδικασία της εκμάθησης γίνεται μέσω της εγγραφής κινήσεων στο λεξικό των διαδρομών, αλλά και της ανανέωσης των τιμών του κόστους μετάβασης μερικών κελιών. Αφού ο πράκτορας έχει «μάθει», θα εκτελεί πάντα τις βέλτιστες κινήσεις.

Πολυπρακτορικό:

Το Contracct Net Protocol (CNP) στην περίπτωσή μας δουλεύει ως εξής:

Έχουμε 2 ταξί. Τα δύο αυτά ταξί κάνουν register και ως manager για να εκτελούν το δικό τους CNP, αλλά και ως participants σε contracts άλλων managers. Ένας manager που εκτελεί το πρωτόκολλο για έναν πελάτη έχει ως participants το άλλο ταξί και τον εαυτό του. Έτσι το πρωτόκολλο εξελίσσεται ως εξής:

Το τηλεφωνικό κέντρο από το αρχείο java παίρνει την λίστα των πελατών και αναθέτει τη διαχείριση του κάθε πελάτη μία στο ένα ταξί και μία στο άλλο εναλλάξ, με τη χρήση percept και ID για τον customer (τους αριθμούς της τοποθεσίας του). Έτσι κάθε διαχειριστής κάνει CNPs για συγκεκριμένους customers που του αναθέτει το τηλεφωνικό κέντρο, έναν-έναν την φορά.

Ας δούμε όμως την λειτουργία του CNP στο σύστημά μας και ας εξηγήσουμε τις συναρτήσεις του .asl νοηματικά, με ένα παράδειγμα. Ας πούμε ότι ένας πελάτης με ID = 21 (βρίσκεται στο 2,1 κελί του πίνακα δηλαδή και είναι πάντα ξεχωριστό αφού δεν μπορεί να εμφανιστεί παραπάνω από 2 πελάτες σε ένα κελί) ανατέθηκε στο πρώτο ταξί (0) για

να τον διαχειριστεί (το ταξί 0 είναι manager του πελάτη 21). Αρχικά ο manager/initiator λέει στους participants «Ποιος θέλει πελάτη;» (call for proposals / CFP). Μετά οι participants (ο εαυτός του και το άλλο ταξί) θα απαντήσουν ως εξής. Αν έχουν μέσα τους πελάτη θα στείλουν refuse στον manager. Αν δεν έχουν πελάτη θα στείλουν proposal στον manager για το task του να τον μεταφέρουν αυτόν τον πελάτη στον προορισμό του (deliver(cust(ID))). Το proposal για αυτό το task θα έχει ένα Offer του οποίο η τιμή θα αναπαράγεται από την συνάρτηση price. Η συνάρτηση price θα θέτει σαν τιμή του offer την απόσταση Μανχάτταν της τοποθεσίας του ταξί participant που κάνει το Offer, με τον πελάτη που θέλει να εξυπηρετήσει. Αυτό το proposal με αυτό το Offer μέσα, θα στέλνεται στον manager.

Ο manager μετά θα παίρνει όλα τα proposals και θα διαλέγει σαν νικητήριο proposal και Offer (WOf) και αντίστοιχα νικητήριο ταξί (WAg), αυτό με το μικρότερο Offer, δηλαδή το proposal του ταξί που βρίσκεται πιο κοντά στον συγκεκριμένο πελάτη. Σε αυτόν θα κάνει accept proposal και σε όλους τους άλλους reject proposal και θα τους ενημερώνει για τα αποτελέσματα.

Του ταξί που του έγινε accept το proposal και νίκησε το συγκεκριμένο CNP, θα του γίνεται true το κατηγόρημα customer(taxi) και έτσι θα εκτελείται όλη η διαδικασία μετά της μεταφοράς του πελάτη αυτού στον προορισμό του (όπως και στο μονοπρακτορικό σύστημα). Η διαδικασία αυτή θα γίνεται για κάθε πελάτη που ανατίθεται σε manager, παράλληλα για διαφορετικά στιγμιότυπα του multi_taxi.asl.

Λειτουργίες

```
MarsModel consructor
```

Θέτει τις αρχικές τοποθεσίες του agent 0 (taxi), του agent 1 (destination) και κάνει τις διαδικασίες αρχικοποίησης defineCosts(), createWalls(), simulateWalls().

updatePercepts

Κάθε φορά που εκτελείται, προσθέτει perception στο μοντέλο για τις τοποθεσίες των δύο πρακτόρων (taxi/0 και d/1(destination)), αλλά και το αν ισχύουν τα percepts gr1 (customer(taxi) – ο πελάτης είναι στο ταξί) και το percept gd (customer(d) – ο πελάτης είναι στο destination).

generateRandomCustomer

Η συνάρτηση αυτή δημιουργεί κάθε φορά που εκτελείται έναν πελάτη, στον οποίο θα αντιστοιχεί ένα τυχαίο location και ένα τυχαίο destination (τυχαία επιλογή ανάμεσα στις τοποθεσίες R, Y, G, B στον χάρτη). Καλεί την getRandomLocation (με όρισμα 1 για να επιλέξει μία από τις 4 τοποθεσίες-γράμματα / επεξήγηση στο κομμάτι για την αντίστοιχη μέθοδο) και αναθέτει τυχαίες συντεταγμένες λοιπόν στην κάθε μεταβλητή. Ύστερα, ελέγχει από την λίστα με τους υπάρχοντες customers, αν η τοποθεσία καποιανού από αυτούς τους πελάτες, είναι ταυτόσημη με την τοποθεσία που μόλις πήραμε τυχαία. Αν ναι, τότε κάνει return και δεν δημιουργείται πελάτης σε αυτήν την τοποθεσία, αφού δεν πρέπει να υπάρχει παραπάνω από ένας πελάτης τη φορά, σε ένα συγκεκριμένο κελί-τοποθεσία. Αν όμως δεν υπάρχει ήδη πελάτης σε αυτήν την τοποθεσία, απλά βάζει την οντότητα του πελάτη στον χάρτη και καλεί την addCustomerToWorld, η οποία στην ουσία ανανεώνει την λίστα των customers και της προσθέτει τον καινούργιο πελάτη με τις συντεταγμένες του.

checkForWall

Παίρνει από το walls για τον κάθε τοίχο, το πού είναι ο τοίχος και το direction (E, K, A, D) προς στο οποίο υπάρχει τοίχος. Αν σε ένα από τους τοίχους, τα location και direction (μεριά του κελιού που υπάρχει ο τοίχος) ταυτίζονται με τα location και direction τους ταξί, τότε σημαίνει ότι πρόκειται να κοπανήσει το ταξί σε τοίχο. Ελέγχει για ποια ακριβώς κατεύθυνση έχουμε (E, K, A, D) και βάζει στον costs, στις αντίστοιχες συντεταγμένες που βρίσκεται το wall (x, y) και για την αντίστοιχη μεριά d (0,1,2,3 / E, K, A, D), το αντίστοιχο κόστος της κατεύθυνσης αυτού του κελιού, που θα είναι 101. Έτσι, ο πράκτορας μας μαθαίνει πού υπάρχουν τοίχοι. Π.χ. αν το κελί [0][0] έχει τοίχο στα δεξιά του, το κόστος του θα βρίσκεται στον costs σε μορφή int [0][0][3] = 101, όπου το 3 αντιστοιχεί σε D (δεξιά).

createWalls

Η συνάρτηση αυτή πολύ απλά κάνει hard code στη λίστα walls που υπάρχει στο περιβάλλον μας, όλα τα κελιά που έχουν τοίχους γύρω τους, καθώς και την κατεύθυνση στην οποία υπάρχει ο τοίχους (E(επάνω), K(κάτω), D(δεξιά), A(αριστερά)).

defineCosts

Η συνάρτηση αυτή όταν εκτελεστεί, αρχικοποιεί όλα τα elements (ένα για κάθε κελί στον χάρτη).

getRandomLocation

Η συνάρτηση αυτή δέχεται σαν όρισμα έναν αριθμό (0 ή 1) που αντιπροσωπεύει τον τύπο της τυχαίας τοποθεσίας που θα παραχθεί. Αν το type είναι 1, τότε επιλέγεται τυχαία μία από τις 4 τοποθεσίες με γράμματα στον πίνακα (R, Y, G, B), αλλιώς αν είναι 0, επιλέγεται μια τυχαία τοποθεσία γενικά στον κόσμο του ταξί. Αυτή η διάκριση γίνεται διότι την πρώτη φορά που θα εμφανιστεί το ταξί, πρέπει να εμφανιστεί σε μια τυχαία

τοποθεσία γενικά, ενώ η τοποθεσία του customer και του destination του όταν δημιουργηθεί, θα πρέπει να είναι μία από τις 4 τοποθεσίες με γράμματα (R, Y, G, B).

LocationToArray

Μέθοδος που δέχεται ένα όρισμα τύπου Location και το μετατρέπει σε Array ακεραίων, όπου το πρώτο στοιχείο της είναι η συντεταγμένη x και το δεύτερο η συντεταγμένη y. Επιστρέφει το αποτέλεσμα.

manhattan

Συνάρτηση που δέχεται 2 Array ακεραίων (το ένα με συντεταγμένες για το αρχικό position και το άλλο για το τελικό), υπολογίζει την απόσταση Μανχάταν μεταξύ αυτών των δύο κελιών και την επιστρέφει.

getNextBestCustomer

Η μέθοδος αυτή έχει ως σκοπό να βρίσκει ποιος από τους πελάτες βρίσκεται εκείνη την στιγμή πιο κοντά στο ταξί και να τον κάνει current customer. Αρχικά, παίρνει την τωρινή τοποθεσία του πράκτορα ταξί και αρχικοποιεί τα δεδομένα του ελαχίστου (ελάχιστη τοποθεσία και αντίστοιχος πελάτης που την έχει (συντεταγμένες αυτού)). Παίρνοντας έναν-έναν τους πελάτες και υπολογίζοντας την απόσταση Μανχάταν του ταξί από την τοποθεσία του, βρίσκει εν τέλει τον πελάτη με την μικρότερη απόσταση από το ταξί, με τον απλό, κλασσικό αλγόριθμο εύρεσης ελαχίστου. Αν τώρα βρεθούν αποστάσεις που έχουν και οι δύο την ελάχιστη τιμή, ο αλγόριθμος θα κοιτάξει ένα βήμα παραπέρα (αυξάνεται κατά ένα το βάθος της αναζήτησης) και θα λάβει υπόψιν ποιος από τους δύο πελάτες έχει το λιγότερο μακρινό destination, από το location του. Έτσι υπολογίζει καινούργιες Μανχάταν αποστάσεις και διαλέξει σαν min_customer, αυτόν που έχει την μικρότερη. Αν και αυτές αποστάσεις

είναι ίδιες, διαλέγει αυθαίρετα ένα από τους 2. Αφού βρέθηκε ποιος θα είναι ο επόμενος πελάτης που θα εξυπηρετηθεί (και αυτός δεν είναι null σε περίπτωση που δεν υπήρχε καν πελάτης να εξυπηρετηθεί στην customers), τότε θέτεται ως current_customer αυτός ο πελάτης (min_customer) και ο πράκτορας 1 (d/destination), τοποθετείται στις συντεταγμένες του καταχωρημένου προορισμού που έχει ο πελάτης που μόλις επιλέχθηκε. Αν επιλέχθηκε όντως πελάτης προς εξυπηρέτηση, γίνεται return true, αλλιώς false.

performActionToArray

Παίρνει ως όρισμα ένα array με δύο ακέραιους, όπου ο καθένας αντιστοιχεί σε μια από τις συντεταγμένες x, y και έναν χαρακτήρα. Άρα το array περιέχει μια τοποθεσία για ένα κελί (σε νούμερα) πάνω στον χάρτη μας και έναν χαρακτήρα που υποδηλώνει στην ουσία ποιον γείτονα αυτού του κελιού θέλουμε να επιστρέψουμε μετά (τον πάνω (E), τον κάτω (K), τον δεξιά (D) ή τον αριστερά (A)). Έτσι δημιουργεί ένα αντίγραφο αυτής της τοποθεσίας και αν ο χαρακτήρας που δεχτήκαμε είναι E, θα μειώσει το y κατά 1 (η αρίθμηση του πίνακα πάει από 0 έως 4, από πάνω προς τα κάτω), αν δεχτεί K θα το αυξήσει κατά 1, αν δεχθεί D θα αυξήσει το x του κατά 1 και αν δεχθεί A θα το μειώσει. Τέλος, επιστρέφει το μετατοπισμένο αντίγραφο.

nextSlot

Αν δεν υπάρχει current_customer (είναι null), τότε καλείται η συνάρτηση που βρίσκει τον πελάτη που πρέπει να εξυπηρετήσει τώρα και τον κάνει current_customer. Αρχικοποιεί την αρχική τοποθεσία του ταξί στην r1 και την αποθηκεύει σε μια integer list, με όνομα old_location. Καλείται η μέθοδος που επιστρέφει μια λίστα με πρώτο στοιχείο της το σημείο που είναι το βέλτιστο να πάει μετά το ταξί (το βάζει στο res_pos / περιέχει τις

συντεταγμένες του), δεύτερο στοιχείο την κατεύθυνση που βρίσκεται αυτό το σημείο σε σχέση με το προηγούμενο (το βάζει στο move) και σαν τρίτο το κόστος της μετάβασης στο σημείο αυτό. Το αντικείμενο r1 τύπου Location πια παίρνει σαν συντεταγμένες αυτές που περιέχει το res_po και τοποθετείται το ταξί στην τοποθεσία r1 (άρα στο επόμενο καλύτερο κελί σε σχέση με αυτό που ήταν πριν). Καλείται μια συνάρτηση (εξηγείται παραπάνω) για να γίνει ο έλεγχος για το αν ανάμεσα στη μετάβαση αυτή υπήρχε τοίχος. Αν ναι ο πράκτορας επιστρέφει στο old_location.
Καλείται από την executableAction.

pickCust

Η μέθοδος αυτή εκτελεί την ενέργεια «εκφόρτωσης» του πελάτη στο ταξί. Αφού ελέγξει αν ο agent 0 (ταξί) έχει τον πελάτη, θα επιχειρήσει το πολύ MErr φορές και εν τέλει θα πάρει τον πελάτη και η boolean μεταβλητή που δείχνει αν το ταξί έχει τον πελάτη, θα γίνει true.Καλείται από την executableAction.

dropCust

Η μέθοδος αυτή εκτελεί την ενέργεια «αποφόρτισης» του πελάτη από το ταξί. Αφού ελέγξει αν η boolean μεταβλητή που δείχνει αν το ταξί έχει τον πελάτη, είναι true, τότε θα την κάνει false και θα βάλει τον πελάτη θα τοποθετηθεί στην τοποθεσία που βρίσκεται αυτή τη στιγμή το ταξί. Καλείται από την executableAction.

removeCust

Η μέθοδος αυτή, αφού ελέγξει αν το ταξί (taxi agent 0) και το destination (d agent 1), έχουν τις ίδιες συντεταγμένες τοποθεσίας και αν ο agent 1 (d) έχει τον πελάτη, τότε μόνο θα τον κάνει remove από αυτόν και θα εξαφανιστεί. Καλείται από την executableAction.

Βιβλιογραφία

- Συστήματα Ευφυών Πρακτόρων: Διαφάνειες και Διαλέξεις μαθήματος
- Συστήματα Ευφυών Πρακτόρων: Διαφάνειες και Διαλέξεις Εργαστηρίου μαθήματος
- https://github.com/nataliakoliou/AgentSpeak-Programming-using-Jason/tree/main/examples/marsbots
- https://github.com/nataliakoliou/AgentSpeak-Programming-using-Jason/blob/main/slides.pdf
- «Εισαγωγή στα Πολυπρακτορικά συστήματα»: Michael Wooldridge – Εκδόσεις Κλειδάριθμος

ΑΠΟΣΤΟΛΟΣ ΧΑΣΙΩΤΗΣ

ΣΩΤΗΡΙΟΣ ΔΗΜΗΤΡΑΚΟΥΛΑΚΟΣ