Ευφυείς Πράκτορες

8ο Εξάμηνο

Τεκμηρίωση της Εργασίας

Θέμα: Ανάπτυξη Generic Planner

Αριθμός εργασίας – Τίτλος εργασίας	***Εργασία εξαμήνου***
Όνομα φοιτητή	Αναστάσιος Καλλίγερος
Αρ. Μητρώου	Π19253
Ημερομηνία παράδοσης	12/09/2023

Περιγραφή του προβλήματος

Σκοπός της εργασίας είναι η ανάπτυξη ενός generic planner για την επίλυση των προβλημάτων Blocks World και Water Pouring Puzzle.

Ένας γεννήτορας σχεδίων (plan generation, planner) παράγει μια ακολουθία ενεργειών ώστε, όταν εκτελεστούν οι ενέργειες ο κόσμος του

πράκτορα να βρεθεί από μια αρχική σε μια τελική κατάσταση. Ο planner θεωρείται generic αν δεν αλλάζουμε σε τίποτα τον κώδικα του για την λύση διαφορετικών προβλημάτων.

Κάθε πρόβλημα που προσπαθούμε να επιλύσουμε διαφοροποιείται από τα άλλα μόνο από την διαφορετική περιγραφή του αρχικού, του τελικού, αλλά και οποιουδήποτε ενδιάμεσου στιγμιοτύπου του κόσμου. Δηλαδή για κάθε πρόβλημα έχουμε μεν τον ίδιο τρόπο αναπαράστασης των καταστάσεων του κόσμου, αλλά (πιθανώς) διαφορετική αναπαράσταση από την αναπαράσταση σε άλλα προβλήματα.

Επίσης για κάθε πρόβλημα διαθέτουμε ένα διαφορετικό σετ ενεργειών που μπορούμε να εκτελέσουμε.

Το πρόβλημα Blocks World

Το blocks world είναι ένας τομέας σχεδιασμού στην τεχνητή νοημοσύνη. Ο αλγόριθμος είναι παρόμοιος με ένα σετ από ξύλινους κύβους διαφόρων σχημάτων και χρωμάτων που κάθονται πάνω σ'ενα τραπέζι. Ο στόχος είναι να κατασκευαστούν μία ή περισσότερες κάθετες στοίβες από κύβους. Μόνο ένας κύβος μπορεί να μετακινηθεί τη φορά: μπορεί να τοποθετηθεί είτε πάνω στο τραπέζι, είτε πάνω σε έναν άλλο κύβο. Εξαιτίας αυτού, οποιοσδήποτε κύβος βρίσκεται κάτω από έναν άλλο κύβο σε οποιαδήποτε δεδομένη στιγμή, δεν μπορεί να μετακινηθεί. Επιπλέον, κάποια είδη κύβων δεν μπορούν να έχουν άλλους κύβους τοποθετημένους πάνω τους.

Το πρόβλημα Water Pouring Puzzle

Τα Water pouring puzzles είναι μια κατηγορία γρίφων που περιλαμβάνουν μια πεπερασμένη ποσότητα κανατών νερού με γνωστή, ακέραιη χωρητικότητα. Αρχικά κάθε κανάτα περιέχει μια γνωστή, ακέραιη ποσότητα υγρού, όχι απαραίτητα ίση με τη χωρητικότητά της.

Γρίφοι αυτού του τύπου ζητάνε πόσα βήματα χυσίματος νερού απαιτούνται από τη μία κανάτα στην άλλη (είτε μέχρι η μία κανάτα να

αδειάσει είτε μέχρι η άλλη να γεμίσει) για να φτάσουμε σε μία κατάστασηστόχο, που προσδιορίζεται από τον όγκο του υγρού που πρέπει να βρίσκεται σε μια ή περισσότερες από τις κανάτες.

Θεωρητική βάση της εφαρμογής

Ο κώδικας της εφαρμογής είναι βασισμένος σε αυτόν τον κώδικα: https://github.com/PetePrattis/generic-planner-for-minigames

Η εφαρμογή είναι υλοποιημένη σε Python 3.9 και χρησιμοποιεί τις βιβλιοθήκες sys, heapq, string, random, copy και numpy.

Αρχικά ορίζεται μια κλάση PriorityQueue, η οποία υλοποιεί τη δομή δεδομένων της ουράς προτεραιότητας. Κάθε αντικείμενο που εισάγεται στην ουρά έχει μια προτεραιότητα, και το πρόγραμμα-πελάτης συνήθως ενδιαφέρεται να βρει το αντικείμενο της ουράς με τη μικρότερη προτεραιότητα.

Η PriorityQueue περιλαμβάνει τις παρακάτω συναρτήσεις:

__init__: Αρχικοποιεί τον σωρό στον οποίο θα τοποθετηθεί η ουρά, και τον μετρητή count, που δείχνει πόσα στοιχεία βρίσκονται στην ουρά.

lenHeap: Επιστρέφει το μήκος του σωρού

push: Παίρνει ως ορίσματα το αντικείμενο και την προτεραιότητά του. Ορίζει με τη βοήθεια αυτών και της count μια καταχώρηση στον σωρό, και τοποθετεί την καταχώρηση αυτή στο σωρό. Τέλος, αυξάνει το count κατά 1.

pop: Αφαιρεί το μικρότερο στοιχείο από τον σωρό, και το επιστρέφει στην έξοδο.

IsEmpty: Ελέγχει αν ο σωρός είναι άδειος, δηλαδή αν το μήκος του είναι μηδέν, και επιστρέφει True αν είναι άδειος, και False αν δεν είναι.

aStarSearch: Υλοποιεί τον αλγόριθμο A* Search. Ο αλγόριθμος A* είναι ένας αλγόριθμος διάσχισης γραφημάτων και αναζήτησης μονοπατιών. Βρίσκει τη βέλτιστη διαδρομή μεταξύ δύο προκαθορισμένων σημείων. Εδώ, ο αλγόριθμος ψάχνει πρώτα τον κόμβο με το χαμηλότερο συνδυαστικό κόστος και ευριστικό.
Τα ευριστικά είναι τεχνικές για γρηγορότερη επίλυση προβλημάτων, ή για καλύτερη προσέγγιση της λύσης σ'ένα πρόβλημα όπου οι κλασικές τεχνικές δεν μπορούν να δώσουν λύση.

Στον κώδικα, η aStarSearch παίρνει ως ορίσματα τον κόμβο και ένα ευριστικό. Αρχικά ορίζονται:

- μια λίστα με όνομα closed
- ένα αντικείμενο της κλάσης PriorityQueue ονόματι Q
- μια μεταβλητή με όνομα iter, που μετράει πόσες φορές εκτελέστηκε ο αλγόριθμος
- μια μεταβλητή με όνομα visited, που μετράει πόσους κόμβους επισκέφτηκε ο αλγόριθμος

Επίσης, ο κόμβος που περάστηκε ως όρισμα στη συνάρτηση ορίζεται ως ο αρχικός κόμβος (startNode), ο οποίος μετά εισάγεται στον σωρό. Μετά ξεκινάει ένας βρόχος while, ο οποίος θα σταματήσει μόνο αν βρεθεί λύση, ή αν βρεθεί ότι δεν υπάρχει λύση. Αν ο σωρός είναι άδειος, θα εμφανιστεί μήνυμα ότι δεν υπάρχει λύση, και ο βρόχος θα διακοπεί. Αν δεν είναι άδειος, συνεχίζεται κανονικά η εκτέλεσή του. Αφαιρείται τότε το μικρότερο στοιχείο του σωρού, και ο αριθμός των κόμβων που έχουμε επισκεφτεί αυξάνεται κατά 1. Αν βρεθεί λύση στο πρόβλημα, τότε εμφανίζεται σχετικό μήνυμα. Έπειτα αυξάνεται ο αριθμός των φορών που έτρεξε ο αλγόριθμος κατά 1.
Στη συνέχεια γίνεται έλεγχος αν η κατάσταση του προβλήματος η οποία περιέχει τον κόμβο βρίσκεται στη λίστα closed, και αν δεν βρίσκεται, την τοποθετεί. Μετά, κάθε κόμβος-παιδί της τοποθετείται στον σωρό, αφού πρώτα υπολογιστεί η προτεραιότητά του.

Κλάσεις και συναρτήσεις για το πρόβλημα Blocks World

startState: Ο ρόλος της είναι να δημιουργεί την αρχική κατάσταση του προβλήματος. Δέχεται ως ορίσματα τον αριθμό των στοιβών και των κύβων που δίνει ο χρήστης.
Αρχικά ορίζεται μια μεταβλητή l στην οποία τοποθετείται ο αριθμός των στοιβών που έδωσε ο χρήστης, και μια μεταβλητή b στην οποία τοποθετούνται οι αριθμοί από το 0 μέχρι το 9. Στη συνέχεια, δημιουργείται μια λίστα list_blocks που περιέχει τους αριθμούς από το 0 μέχρι τον αριθμό των κύβων-1, η οποία μετά ανακατεύεται. Ορίζεται επίσης μια κενή λίστα problem_state, ώστε να τοποθετηθεί εκεί η κατάσταση του προβλήματος.
Για κάθε κύβο, αν ο κύβος δεν βρίσκεται στη λίστα με τους κύβους, γίνεται έξοδος από τον βρόχο. Αν έχουμε μία μόνο στοίβα, τότε βάζουμε τη λίστα των κύβων στη λίστα problem_state, και βγαίνουμε από τον βρόχο.
Σε άλλη περίπτωση, γεννάμε τυχαία έναν αριθμό από το 1 μέχρι τον αριθμό των κύβων-1, και δημιουργούμε μια λίστα s στην οποία τοποθετούμε τους αριθμούς από το 1 μέχρι τον αριθμό που δημιουργήσαμε-1 της list_blocks. Τοποθετούμε ύστερα την s στη problem_state.
Στη συνέχεια, αφαιρούμε τον αριθμού που δημιουργήσαμε από τον αριθμό των κύβων μειώνουμε των αριθμό των στοιβών κατά 1, και στη λίστα list_blocks κρατάμε τα στοιχεία από τη θέση που αντιστοιχεί στον αριθμό που δημιουργήσαμε και έπειτα.
Τώρα, όσο το μήκος της problem_state είναι μικρότερο από τον αριθμό των στοιβών, προσθέτουμε άδειες στοίβες σε αυτή.
Τέλος, ανακατεύουμε την problem_state, και την επιστρέφουμε στην έξοδο.

finalState: Ο ρόλος της είναι να γεννά την τελική κατάσταση, δοθείσης της αρχικής. Παίρνει ως όρισμα την αρχική κατάσταση του προβλήματος.
Αρχικά ορίζεται μια κενή λίστα final, η οποία θα περιέχει την τελική κατάσταση του προβλήματος.
Καθεμιά από τις στοίβες της αρχικής κατάστασης προστίθεται στην τελική. Έπειτα τα νέα περιεχόμενα της final ταξινομούνται.
Τέλος, για καθένα από τα στοιχεία της αρχικής κατάστασης, προστίθεται μια άδεια στοίβα στη final, και επιστρέφουμε τη final στην έξοδο.

NodeBlocks: Η κλάση NodeBlocks περιγράφει έναν κόμβο του προβλήματος Blocks World. Περιέχει τις παρακάτω συναρτήσεις:

__init__: Η συνάρτηση αυτή παίρνει ως ορίσματα την κατάσταση του προβλήματος, δηλαδή το πώς είναι τοποθετημένοι οι κύβοι, και τον κόμβο-γονιό(από προεπιλογή δεν υπάρχει γονιός). Αρχικοποιεί την κατάσταση του κόσμου, τον κόμβο-γονιό και το κόστος του κόμβου. Αν ο κόμβος είναι γονιός, το κόστος του ορίζεται ως το κόστος του γονιού + 1.

goalTest: Η συνάρτηση αυτή βρίσκει αν το πρόβλημα έφτασε στην τελική του κατάσταση. Ελέγχει αν η τρέχουσα κατάσταση είναι όμοια με την τελική, και αν είναι, γράφει στην κονσόλα “Solution found!”, εμφανίζει τις ενδιάμεσες καταστάσεις του προβλήματος καλώντας την συνάρτηση traceback, και επιστρέφει True στην έξοδο. Σε άλλη περίπτωση επιστρέφει False.

heuristics: Η συνάρτηση αυτή υπολογίζει το ευριστικό του κάθε κόμβου, ώστε με τη βοήθεια αυτού να του ανατεθεί μια προτεραιότητα κατά την εκτέλεση του προγράμματος.
Ορίζουμε αρχικά μια μεταβλητή not_on_stack_zero, που είναι η διαφορά του ύψους της πρώτης στοίβας της τελικής κατάστασης με το ύψος της πρώτης στοίβας της τρέχουσας κατάστασης. Ορίζουμε επίσης μια μεταβλητή wrong_on_stack_zero, και της δίνουμε την τιμή μηδέν. Για καθέναν από τους κύβους της πρώτης στοίβας της τρέχουσας κατάστασης ελέγχει αν είναι ίδιος με τον αντίστοιχό του στην τελική κατάσταση, και αν δεν είναι, η wrong_on_stack_zero αυξάνεται κατά 2. Έπειτα ορίζεται μια μεταβλητή dis_bw_pairs, η οποία αρχικοποιείται επίσης με μηδέν. Για κάθε μεμονωμένο κύβο, γίνεται έλεγχος αν είναι μεγαλύτερος από τον επόμενό του, και αν είναι, το dis_bw_pairs αυξάνεται κατά 1. Στην έξοδο, επιστρέφεται το αποτέλεσμα της παράστασης not_on_stack_zero + 4 * wrong_on_stack_zero – dis_bw_pairs. Αυτό είναι το ευριστικό του κόμβου.

getSuccessors: Η συνάρτηση αυτή επιστρέφει τα παιδιά ενός κόμβου. Παίρνει ως όρισμα το ευριστικό του κόμβου. Αρχικά ορίζουμε μια λίστα children, όπου θα τοποθετηθούν τα παιδιά του κόμβου. Για κάθε στοίβα στο πρόβλημα, παίρνουμε την κάθε στοίβα σε ένα αντίγραφο

ουσιαστικά του προβλήματος. Αν δεν δουλεύουμε στις ίδιες στοίβες, και υπάρχει το αντίγραφο της στοίβας, δουλεύουμε ως εξής:

- Αντιγράφουμε τη στοίβα της πρωτότυπης κατάστασης, και τη βάζουμε στη μεταβλητή temp
- Αντιγράφουμε τον κόμβο και τον τοποθετούμε στη μεταβλητή child
- Αντιγράφουμε τη στοίβα της κατάστασης-αντίγραφο, και τη βάζουμε στη μεταβλητή temp1
- Εισάγουμε το τελευταίο στοιχείο της στοίβας-αντίγραφο στην πρωτότυπη στοίβα, και το διαγράφουμε από τη στοίβα-αντίγραφο
- Ορίζουμε την κατάσταση που αποθηκεύσαμε στη temp ως τη στοίβα του παιδιού της πρωτότυπης κατάστασης
- Ορίζουμε την κατάσταση που αποθηκεύσαμε στη temp1 ως τη στοίβα του παιδιού της κατάστασης-αντιγράφου
- Αποθηκεύουμε τον κόμβο στην τωρινή του κατάσταση ως γονιό
- Εισάγουμε το παιδί στη λίστα children
- Τέλος, επιστρέφουμε στην έξοδο τη λίστα children

traceback: Η συνάρτηση αυτή εμφανίζει τον αριθμό των βημάτων που απαιτείται για να πάμε από την αρχική στην τελική κατάσταση, καθώς και ποια είναι αυτά τα βήματα.
Ορίζουμε αρχικά μια λίστα path_back, όπου θα τοποθετηθούν οι ενδιάμεσες καταστάσεις, και μια μεταβλητή s, όπου αποθηκεύουμε τον κόμβο.
Τοποθετούμε κάθε κατάσταση του προβλήματος στη path_back, και τοποθετούμε κάθε φορά στη μεταβλητή s τον γονιό της s με την οποία μόλις δουλέψαμε.
Στη συνέχεια εμφανίζουμε τον αριθμό των βημάτων που απαιτούνται για να φτάσουμε από την αρχική στην τελική κατάσταση του προβλήματος, καθώς και τα ίδια τα βήματα, αντιστρέφοντας την path_back.
Εμφανίζουμε, παράλληλα, και μερικά επεξηγηματικά κείμενα.

pathCost: Η συνάρτηση αυτή υπολογίζει το κόστος του μονοπατιού από τον έναν κόμβο στον άλλον.
Υπολογίζει και επιστρέφει το άθροισμα της εξόδου της heuristics + το κόστος του κόμβου.

Κλάσεις και συναρτήσεις για το πρόβλημα Water Pouring Puzzle

getSuccessorsWater: Η συνάρτηση αυτή δέχεται ως ορίσματα τις ποσότητες του υγρού που περιέχουν οι κανάτες, τις οποίες έδωσε ο χρήστης. Ο ρόλος της είναι να βρίσκει την επόμενη/επόμενες κατάσταση/καταστάσεις από την τρέχουσα στο πρόβλημα. Αρχικά ορίζεται μια λίστα successors, στην οποία θα τοποθετηθεί/τοποθετηθούν η/οι επόμενη/επόμενες κατάσταση/καταστάσεις. Ορίζουμε στη συνέχεια τις αρχικές ποσότητες που περιέχουν οι κανάτες ως C1 και C2 αντίστοιχα. Τα J1 και J2 αντιμετωπίζονται ως οι ποσότητες που περιέχουν οι κανάτες σε κάθε δεδομένη στιγμή.

- Αν η τρέχουσα ποσότητα που περιέχει η J1 είναι μικρότερη από την αρχική της, η J1 γεμίζει
- Αν η τρέχουσα ποσότητα που περιέχει η J2 είναι μικρότερη από την αρχική της, η J2 γεμίζει
- Αν η J1 περιέχει υγρό, αδειάζει
- Αν η J2 περιέχει υγρό, αδειάζει
- Αν η J1 και η J2 μαζί περιέχουν λιγότερη ή ίση ποσότητα υγρού από την αρχική ποσότητα της J1, η J2 αδειάζει ολόκληρη στη J1
- Αν η J1 και η J2 μαζί περιέχουν λιγότερη ή ίση ποσότητα υγρού από την αρχική ποσότητα της J2, η J1 αδειάζει ολόκληρη στη J2
- Αν η J1 και η J2 μαζί περιέχουν περισσότερο υγρό από την αρχική ποσότητα της J1, η J1 γεμίζει από τη J2
- Αν η J1 και η J2 μαζί περιέχουν περισσότερο υγρό από την αρχική ποσότητα της J2, η J2 γεμίζει από τη J1

Καθεμιά από αυτές τις περιπτώσεις εισάγεται, αν ικανοποιείται φυσικά, στη λίστα successors. Στο τέλος, επιστρέφεται στην έξοδο η successors.

waterHeurestic: Υπολογίζει το ευριστικό του εκάστοτε κόμβου του προβλήματος. Παίρνει ως όρισμα την κατάσταση του προβλήματος. Το ευριστικό του κόμβου είναι ίσο με την απόλυτη τιμή της τιμής του πρώτου κόμβου του προβλήματος, μειωμένη κατά 2.

NodeWater: Η κλάση αυτή περιγράφει έναν κόμβο του προβλήματος Water Pouring Puzzle. Περιλαμβάνει τις παρακάτω συναρτήσεις:

__init__: Παίρνει ως ορίσματα την κατάσταση του προβλήματος, το μονοπάτι από τον αρχικό κόμβο προς τον κόμβο που περιγράφεται από την κλάση, το κόστος του, και το ευριστικό του. Η συνάρτηση αρχικοποιεί τις παραπάνω τιμές.

getSuccessors: Η συνάρτηση αυτή επιστρέφει τους επόμενους κόμβους του τρέχοντα κόμβου. Παίρνει για όρισμα την συνάρτηση ευριστικού. Μπορεί και να μην υπάρχει τέτοια συνάρτηση. Σε αυτή την περίπτωση, αρχικοποιείται με την τιμή None.
Ορίζει αρχικά μια λίστα children, στην οποία θα τοποθετηθούν τα παιδιά του κόμβου. Έπειτα, για καθεμιά από τις επόμενες καταστάσεις που βρέθηκαν μέσω της getSuccessorsWater, ορίζει την κατάσταση του προβλήματος ως την πρώτη τιμή του successor, το μονοπάτι, και τοποθετεί στο μονοπάτι τη δεύτερη τιμή του successor. Στη συνέχεια υπολογίζει το κόστος του κόμβου, προσθέτοντας στο κόστος που ήδη έχει, την τρίτη τιμή του successors. Αν έχει περαστεί συνάρτηση ευριστικού, ορίζεται εκείνη ως το ευριστικό του κόμβου. Αν όχι, αυτό ορίζεται ίσο με μηδέν. Μετά, ορίζει τον κόμβο-παιδί με βάση τις τιμές που υπολογίστηκαν παραπάνω, εισάγει τον κόμβο στη λίστα children, και τέλος επιστρέφει τη children στην έξοδο.

pathCost: Η συνάρτηση αυτή υπολογίζει το κόστος του μονοπατιού από τον έναν κόμβο στον άλλον.
Υπολογίζει και επιστρέφει το άθροισμα της εξόδου του ευριστικού + το κόστος του κόμβου.

goalTest: Ελέγχει αν η τρέχουσα κατάσταση του προβλήματος είναι η τελική. Αν η τιμή του πρώτου κόμβου ισούται με 2, εμφανίζεται στην κονσόλα το μονοπάτι του προβλήματος, και επιστρέφεται True στην έξοδο. Σε άλλη περίπτωση, επιστρέφεται False.

Γενικά, τόσο στο πρόβλημα των κύβων, όσο και στο πρόβλημα των κανατών, υπάρχει μια αρχική κατάσταση και μια τελική κατάσταση οι οποίες είναι γνωστές εκ των προτέρων, και το πρόγραμμα πρέπει να υπολογίσει και να εμφανίσει τα ενδιάμεσα βήματα, ώστε να φτάσουμε από την αρχική στην τελική κατάσταση. Κάθε κόμβος αποτελεί και μια ενδιάμεση κατάσταση στα προβλήματα αυτά.

Αφού ορίστηκαν οι κλάσεις και οι συναρτήσεις που δίνουν λύση στα δύο προβλήματα, ορίζεται μια συνάρτηση με όνομα print_menu, η οποία εμφανίζει στην κονσόλα το μενού της εφαρμογής, δηλαδή τι πρέπει να πατήσει ο χρήστης για να αλληλεπιδράσει μαζί της.
Ορίζεται επίσης μια boolean μεταβλητή με όνομα Loop, ώστε το μενού της εφαρμογής, καθώς και το σχετικό κάλεσμα για είσοδο τιμής από τον χρήστη να εμφανίζονται συνέχεια μετά την επίλυση(ή όχι) του εκάστοτε προβλήματος από την εφαρμογή, μέχρι ο χρήστης να επιλέξει να την τερματίσει.
Ορίζεται επίσης ένα αντικείμενο pq της κλάσης PriorityQueue, ώστε να μπορεί να χρησιμοποιηθεί η συνάρτηση aStarSearch για την επίλυση των προβλημάτων.
Στη συνέχεια, αρχίζει ο βρόχος ο οποίος εμφανίζει το μενού και καλεί τον χρήστη να επιλέξει ποιο πρόβλημα θέλει να λύσει, ή να τερματίσει το πρόγραμμα. Εμφανίζεται το μενού, και καλείται ο χρήστης να εισάγει έναν αριθμό από το 1 έως το 3.
Αν ο χρήστης εισάγει το 1, τότε επιλύεται το Blocks World, και εμφανίζεται μήνυμα στον χρήστη ότι επιλέχθηκε το πρόβλημα αυτό. Ζητείται από τον χρήστη να εισάγει τον αριθμό των στοιβών του προβλήματος, και στη συνέχεια, τον αριθμό των κύβων. Αν μια από τις παραμέτρους αυτές είναι αρνητική, τότε εμφανίζεται μήνυμα στον χρήστη ότι πρέπει να εισάγει μόνο θετικούς αριθμούς, και καλείται να τις εισάγει εκ νέου. Μόλις εισάγει αυτές τις παραμέτρους, το πρόγραμμα γεννάει την αρχική κατάσταση του προβλήματος με την χρήση αυτών, στη συνέχεια γεννάει την τελική κατάσταση του προβλήματος με τη βοήθεια της αρχικής, και χρησιμοποιεί την aStarSearch για να γεννήσει τις ενδιάμεσες.
Αν ο χρήστης εισάγει το 2, τότε επιλύεται το Water Pouring Puzzle, και εμφανίζεται μήνυμα στον χρήστη ότι επιλέχθηκε το πρόβλημα αυτό. Ζητείται από τον χρήστη να εισάγει την ποσότητα υγρού που περιέχει η

πρώτη, και στη συνέχεια, η δεύτερη κανάτα. Αν μια από τις παραμέτρους αυτές είναι αρνητική, τότε εμφανίζεται μήνυμα στον χρήστη ότι πρέπει να εισάγει μόνο θετικούς αριθμούς, και καλείται να τις εισάγει εκ νέου. Μόλις εισάγει αυτές τις παραμέτρους, καλείται η aStarSearch για την επίλυση του προβλήματος με τις δοθείσες παραμέτρους.
Μόλις επιλυθεί οποιοδήποτε από τα δύο προβλήματα, το μενού της εφαρμογής και η προτροπή για κάποια είσοδο από τον χρήστη εμφανίζονται εκ νέου
Αν ο χρήστης εισάγει το 3, εμφανίζεται μήνυμα στον χρήστη ότι επιλέχθηκε να γίνει έξοδος από το πρόγραμμα, και η μεταβλητή Loop παίρνει την τιμή False, ώστε να γίνει έξοδος από τον βρόχο, και κατ'επέκταση, από την εφαρμογή.
Αν ο χρήστης εισάγει οτιδήποτε άλλο, εμφανίζεται μήνυμα, που λέει ότι αυτό που εισήγαγε δεν είναι έγκυρη επιλογή, και παροτρύνεται να πατήσει το Enter, ώστε να επιστρέψει στο μενού και να κάνει μια επιλογή εκ νέου.

Παράδειγμα εκτέλεσης της εφαρμογής για τα δύο προβλήματα

- **Blocks World**

```
------------------------------ MENU ------------------------------
1. Blocks World
2. Water Pouring Puzzle
3. Exit
------------------------------------------------------------------
Enter your choice [1-3]: 1
Blocks World has been selected
Stacks: 3
Blocks: 5
[['0', '4', '3', '2'], ['1'], []]
Solution Found!
Number of MOVES required :  10
--------------------------------------------------
List of nodes forming the path from the root to the goal.
[['0', '4', '3', '2'], ['1'], []]
[['0', '4', '3'], ['1', '2'], []]
[['0', '4'], ['1', '2', '3'], []]
[['0'], ['1', '2', '3'], ['4']]
[['0'], ['1', '2'], ['4', '3']]
[['0'], ['1'], ['4', '3', '2']]
[['0', '1'], [], ['4', '3', '2']]
[['0', '1', '2'], [], ['4', '3']]
[['0', '1', '2', '3'], [], ['4']]
[['0', '1', '2', '3', '4'], [], []]
```

Μόλις ο χρήστης επιλέξει το Blocks World, καλείται να επιλέξει με πόσες στοίβες θα τρέξει ο αλγόριθμος. Μόλις δώσει έναν θετικό αριθμό, και πατήσει Enter, θα κληθεί να δώσει τον αριθμό των κύβων με τους οποίους θα τρέξει ο αλγόριθμος. Μόλις δώσει έναν θετικό αριθμό, και πατήσει Enter, ο αλγόριθμος θα ξεκινήσει να τρέχει. Αρχικά θα εμφανίσει την αρχική κατάσταση του προβλήματος, και το αν βρέθηκε λύση ή όχι. Αν βρεθεί λύση, εμφανίζει τον αριθμό των κινήσεων που απαιτούνται για την επίλυση του προβλήματος, καθώς και τα ενδιάμεσα βήματα-κόμβους για την επίλυσή του.

- Βλέπουμε ότι ξεκινάει με την αρχική κατάσταση που εμφάνισε και πιο πάνω. Στην πρώτη στοίβα υπάρχουν, από κάτω προς τα πάνω, οι κύβοι 0, 4, 3, 2. Στη δεύτερη, ο κύβος 1, και η τρίτη είναι κενή.
- Ο κύβος 2 μετακινείται στη δεύτερη στοίβα
- Ο κύβος 3 μετακινείται στη δεύτερη στοίβα
- Ο κύβος 4 μετακινείται στη τρίτη στοίβα • Ο κύβος 3 μετακινείται στη τρίτη στοίβα
- Ο κύβος 2 μετακινείται στη τρίτη στοίβα
- Ο κύβος 1 μετακινείται στη πρώτη στοίβα • Ο κύβος 2 μετακινείται στη πρώτη στοίβα
- Ο κύβος 3 μετακινείται στη πρώτη στοίβα

- Ο κύβος 4 μετακινείται στη πρώτη στοίβα

- **Water Pouring Puzzle**

```
------------------------------ MENU ------------------------------
1. Blocks World
2. Water Pouring Puzzle
3. Exit
------------------------------------------------------------------
Enter your choice [1-3]: 2
Water Pouring Puzzle has been selected
Liters/Gallons in jug 1: 2
Liters/Gallons in jug 2: 4
```

```
---------------------------------------------------------------------------
TypeError                                 Traceback (most recent call last)
<ipython-input-1-c1aac65a6588> in <module>
    280         t = (data1,data2)#size of jugs
    281
--> 282         pq.aStarSearch(NodeWater((0,0),[],0,0),waterHeurestic)
    283
    284     elif choice=="3":

<ipython-input-1-c1aac65a6588> in aStarSearch(self, node, heuristic)
     63             if node.state not in closed:
     64                 closed.append(node.state)
---> 65                 for childNode in node.getSuccessors(heuristic):
     66                     Q.push(childNode, childNode.pathCost())
     67

<ipython-input-1-c1aac65a6588> in getSuccessors(self, heuristicFunction)
    211             def getSuccessors(self, heuristicFunction=None):
    212                 children = []
--> 213                 for successor in getSuccessorsWater(self.state):
    214                     state = successor[0]
    215                     path = list(self.path)

TypeError: getSuccessorsWater() missing 1 required positional argument: 'J2'
```

Μόλις ο χρήστης επιλέξει το Water Pouring Puzzle, καλείται να επιλέξει την ποσότητα που περιέχει η πρώτη κανάτα. Μόλις δώσει έναν θετικό

αριθμό, και πατήσει Enter, την ποσότητα που περιέχει η δεύτερη κανάτα. Μόλις δώσει έναν θετικό αριθμό, και πατήσει Enter, ο αλγόριθμος θα ξεκινήσει να τρέχει. Δυστυχώς παρουσιάζεται ένα σφάλμα το οποίο δεν μπόρεσα να διορθώσω, και το οποίο αποτρέπει τον αλγόριθμο από το να τρέξει.

Οδηγίες Εγκατάστασης

Προσπάθησα να μετατρέψω το .py αρχείο μου(GenericPlanner.py) σε .exe, αλλά αντιμετώπισα πρόβλημα με το pyinstaller και δεν μπόρεσα να κάνω τη μετατροπή. Οπότε για να τρέξετε το πρόγραμμά μου θα χρειαστεί να έχετε εγκατεστημένη την Python και τη βιβλιοθήκη numpy.

Αν δεν έχετε την Python, μπορείτε να την κατεβάσετε από εδώ: https://www.python.org/downloads/

Επιλέξτε την έκδοση που σας ταιριάζει.

Μόλις τελειώσει η λήψη, κάντε διπλό κλικ στον installer για να ανοίξει.

Στο παράθυρο που θα ανοίξει, κάντε check το κουτάκι στο κάτω μέρος του που λέει “Add Python 3.X to PATH” και μετά πατήστε “Install Now”.

Η εγκατάσταση ξεκίνησε. Περιμένετε να ολοκληρωθεί. Αν εμφανιστεί κάποιο κουτάκι που λέει “Θέλετε να επιτρέψετε στο πρόγραμμα να κάνει αλλαγές στον υπολογιστή σας;”, πατήστε “Ναι/Yes”.

Μόλις ολοκληρωθεί η εγκατάσταση, θα εμφανιστεί ένα παράθυρο που λέει “Setup was successful”. Πατήστε “close” σε αυτό.

Τώρα, ανοίξτε command prompt/γραμμή εντολών. Αναζητήστε το στην αναζήτηση στη μπάρα, ή πατήστε windows key + R, και στο παραθυράκι που θα ανοίξει γράψτε cmd και πατήστε enter. Θα ανοίξει η γραμμή εντολών σε κάθε περίπτωση

Όταν ανοίξει, τρέξτε την παρακάτω εντολή:

python -m pip install numpy

ή

py -m pip install numpy

ο,τι δουλεύει για εσάς

Περιμένετε μέχρι να ολοκληρωθεί η εγκατάσταση της numpy. Μόλις ολοκληρωθεί, θα μπορέσετε να τρέξετε τον κώδικά μου.

Ανεπίλυτα προβλήματα

Το μόνο ανεπίλυτο πρόβλημα της εφαρμογής είναι στο πρόβλημα με τις κανάτες. Ο χρήστης μπορεί κανονικά να εισάγει την ποσότητα του υγρού που περιέχουν οι κανάτες, όμως όταν πάει να τρέξει τον αλγόριθμο, αυτός παθαίνει σφάλμα και διακόπτει τη λειτουργία του.

Το σφάλμα περιγράφεται από το παρακάτω screenshot:

```
---------------------------------------------------------------------------
TypeError                                 Traceback (most recent call last)
<ipython-input-1-c1aac65a6588> in <module>
    280         t = (data1,data2)#size of jugs
    281
--> 282         pq.aStarSearch(NodeWater((0,0),[],0,0),waterHeurestic)
    283
    284     elif choice=="3":

<ipython-input-1-c1aac65a6588> in aStarSearch(self, node, heuristic)
     63             if node.state not in closed:
     64                 closed.append(node.state)
---> 65                 for childNode in node.getSuccessors(heuristic):
     66                     Q.push(childNode, childNode.pathCost())
     67

<ipython-input-1-c1aac65a6588> in getSuccessors(self, heuristicFunction)
    211             def getSuccessors(self, heuristicFunction=None):
    212                 children = []
--> 213                 for successor in getSuccessorsWater(self.state):
    214                     state = successor[0]
    215                     path = list(self.path)

TypeError: getSuccessorsWater() missing 1 required positional argument: 'J2'
```

Πιο συγκεκριμένα, με το που εισάγει τις τιμές ο χρήστης και ο αλγόριθμος ξεκινά να τρέχει, καλείται η aStarSearch, ώστε να αναζητηθεί το μονοπάτι από τον αρχικό προς τον τελικό κόμβο του προβλήματος. Αρχίζει η εκτέλεση της aStarSearch, και φτάνει σ'ενα σημείο, όπου χρειάζεται να γίνει κλήση της getSuccessors, για να επιστραφεί μια τιμή. Αρχίζει τότε να εκτελείται η getSuccessors, και φτάνει σ'ενα σημείο, όπου χρειάζεται να γίνει κλήση της getSuccessorsWater. Η getSuccessorsWater παίρνει δύο κατηγορήματα, και στην κλήση της εδώ, περνιέται μόνο ένα. Το άλλο της κατηγόρημα μένει κενό, και δεν μπορεί να τρέξει αν της λείπει μια τιμή που χρειάζεται. Γι'αυτό παρουσιάζεται το σφάλμα. Δεν μπόρεσα να καταλάβω ποιο θα έπρεπε να είναι το δεύτερο κατηγόρημα εδώ.