1η προγραμματιστική εργασία Τεχνητή Νοημοσύνη

Ονοματεπώνυμα	A.M
Ελένη Ζάνου	p3210049
Μαρία Μπουμπή	p3210135

Κατάλογος αρχείων

```
3210049-3210135
|   README.txt
|   project1_description.pdf
├── src
|   | main.py
|   | person.py
|   | spacesearcher.py
|   | state.py
```

Κλάση Person

Η Person αναπαριστά ένα άτομο, το οποίο καθορίζεται από τα εξής πεδία:

- *_time:* Ο χρόνος που χρειάζεται για να περάσει τη γέφυρα.
- *_fast:* Αν είναι γρήγορος ή όχι.

Σημείωση: Όταν αρχικοποιούνται τα αντικείμενα Person, αυτά με τους 2 πιο μικρούς χρόνους τα θέτουμε *fast = True* ;έτσι ώστε να ξέρουμε ποιοί Persons πρέπει να γυρίζουν πίσω με τον φακό.

Κλάση State

Η State αναπαριστά μια κατάσταση του προβλήματος και καθορίζεται από τα εξής πεδία:

- *_total_time:* Ο χρόνος που έχει διανύσει το παιχνίδι μέχρι και αυτή την κατάσταση.
- *_persons_moved:* Η λίστα με το άτομο ή άτομα που μετακινούνται στη συγκεκριμένη κατάσταση.
- *_f:* Το συνολικό κόστος της κατάστασης.
- *_h:* Η ευρετική συνάρτηση που αξιολογεί την κατάσταση.
- *_g:* Το κόστος από την αρχική κατάσταση μέχρι την τωρινή κατάταση.
- *_left:* Η λίστα με τα άτομα που βρίσκονται στα αριστερά.
- *_right:* Η λίστα με τα άτομα που βρίσκονται στα δεξιά.
- *_father:* Η κατάσταση του πατέρα.
- *_state_duration:* Ο χρόνος που διαρκεί η συγκεκριμένη κατάσταση.
- *_torch:* Η θέση του φακού (δεξιά ή αριστερά).
- *_game_time:* Ο συνολικός χρόνος του παιχνιδιού.

Συνάρτηση evaluate

Προκειμένου να υλοποιήσουμε το πρόγραμμα με τον αλγόριθμο A* πρέπει να κατασκευάσουμε μια ευρετική συνάρτηση *(calculate_H)* η οποία επιστρέφει το εκτιμώμενο κόστος μιας κατάστασης καθώς και το κόστος από την ρίζα ως τον συγκεκριμένο κόμβο *(calculate_G)*. Έτσι το άθροισμα της `calculate_H()` και της `calculate_G()` είναι το συνολικό κόστος μιάς κατάστασης (δηλαδή το πεδίο f).

Συνάρτση calculate_G

Η συνάρτηση αυτή υπολογίζει το συνολικό χρόνο που χρειάζεται από την αρχική κατάσταση μέχρι τον τωρινό κομβο. Αυτό προκύπτει από το άθροισμα του πεδίου g του πατέρα-κόμβου και από τον χρόνο που χρειάζεται να γίνει η μετακίνηση της συγκεκριμένης κατάστασης *(state_duration)*. Μετά ορίζει το πεδίο g ως το άθροισμα αυτό και το επιστρέφει.

Συνάρτηση calculate_H

Η συνάρτηση αυτή υλοποιεί την ευρετική συνάρτηση. Η γενική ιδέα είναι οτι υπολογίζει ένα score για κάθε κατάσταση. Η κατάσταση με το μικρότερο score είναι αυτή στην οποία θέλουμε να επιλέξει ο αλγόριθμος του A*, γι'αυτό η συνάρτηση αυτή θα δίνει χαμηλό score στις επιθυμητές καταστάσεις και υψηλό σε μη επιθυμητές καταστάσεις.

Αρχικά ορίζει την μεταβλητή *individual* ως True αν έχει μετακινηθεί μόνο ένα άτομο, αλλίως False.

- Αμα ο *φακός* στην συγκεκριμένη κατάσταση είναι *δεξιά* αυτο σημαίνει οτι το άτομο ή τα άτομα μόλις μετακινήθηκαν **από αριστερά στα δεξιά**.
    - Δίνει ένα penalty (υψηλό score) στην κατάσταση που έχει μετακινήσει 2 άτομα, καθώς αυτό δεν συμφέρει το παιχνίδι. Αυξάνει το score, δηλαδή, κατά `(10 + game_time) * (not individual)`, οπότε αν βρισκόμαστε σε κατάσταση που μετακινείται ένας το score θα αυξηθεί κατά *0* επειδή `not individual = False` αλλιώς θα αυξηθεί κατά *10 + game_time* `(not individual = True)`.
    - Στη συνέχεια, εμείς θέλουμε να μετακινείται αυτός με τον μικρότερο χρόνο ώστε να μην γίνεται χρονοβόρα η επιστροφή του φακού. Σε αυτή την περίπτωση, η συνάρτηση καλεί την `find_position_in_list()` ώστε να μας επιστρέψει τη θέση του ατόμου στην αριστερή λίστα που μετακινήθηκε και το πολλαπλασιάζει με την χρονο που χρειάζεται για να εκτελεστεί η κατάσταση αυτή (τον χρόνο του ατόμου δηλαδή). Για παράδειγμα, αν βρισκόμαστε σε κατάσταση που μετακινείται ο πιο γρήγορος η `find_position_in_list()` θα επιστρέψει 1 και το score θα διαμορφωθεί σε *1 * state_duration*. Ο παραπάνω υπολογισμός καθώς και το πιθανό penalty ορίζουν το score της κατάστασης στην ευρετική.
- Αμα ο *φακός* στην συγκεκριμένη κατάσταση είναι *αριστερά* αυτό σημαίνει οτι το άτομο ή τα άτομα μόλις μετακινήθηκαν από **δεξιά στα αριστερά**.
    - Αντίστοιχα με πριν δίνουμε ένα penalty στην κατάσταση που έχει μετακινήσει 1 άτομο, καθώς αυτό δεν συμφέρει το παιχνίδι αφού στόχος είναι να πάνε όσο πιο γρήγορα τα μέλη στα αριστερά (θέλουμε δηλαδή από τα δεξιά στα αριστερά να μετακινούνται πάντα 2 άτομα). Αυξάνει, δηλαδή, το score κατά `(10 + game_time) * individual`. Οπότε, αν μετακινούνται 2 άτομα (`individual = False`) και άρα το score αυξάνει κατά *0*. Αν όμως μετακινείται ένας τότε `individual = True` και άρα το score αυξάνει κατά *10 + game_time*.

    - Στη περίπτωση που έχουν μετακινηθεί 2 άτομα τότε καλούμε την `evaluate_left_movement()` έτσι ώστε να αξιολογήσουμε τις περιπτώσεις μετακίνησης 2 ατόμων. Το score της ευρετικής προκύπτει από το penalty στη περίπτωση του ενός ατόμου και από το `evaluate_left_movement()` στην περίπτωση των 2 ατόμων.

Τέλος, το συνολικό score της κατάστασης ορίζει το πεδίο *h* και επιστρέφεται.

Συνάρτηση evaluate_left_movement

Η συνάρτηση αυτή αξιολογεί τις καταστάσεις όπου ο *φακός* βρίσκεται στα *δεξιά* και πρέπει να μετακινηθούν άτομα από **τα δεξιά στα αριστερά**. Η αξιολόγηση αυτή λαμβάνει υπόψην την θέση των 2 πιο γρήγορων άτομων, οι οποίοι θέλουμε να γυρίζουν τον φακό πίσω στην δεξιά όχθη. Έτσι, η συνάρτηση υπολογίζει το πλήθος των γρήγορων ατόμων στην αριστερή λίστα του πατέρα (διότι θέλουμε την κατάσταση των λιστών πρίν γίνει η κίνηση).

- **Αν δεν υπάρχει κανένας γρήγορος στα αριστερά** τότε δεν μπορούμε να στείλουμε τους 2 πιο αργούς (ώστε ο χρόνος του ενός να επικαλυφθεί με του άλλου) γιατί μετά κάποιος θα πρέπει να τον επιστρέψει και εμείς θέλουμε μόνο οι 2 πιο γρήγοροι να το κάνουν αυτό. Οπότε, σε αυτή την περίπτωση δίνει χαμηλό score στη κατάσταση που πηγαίνουν οι 2 πιο γρήγοροι από τα δεξιά στα αριστερά, ώστε η κατάσταση αυτή να επιλεγεί από τον αλγόριθμο και δίνει πιο υψηλό score (ανάλογο του χρόνου διάσχισης και του συνολικού χρόνου του παιχνιδιού, δηλαδή `state_duration * (10 + game_time)`) στις υπόλοιπες καταστάσεις.
- Σε άλλη περίπτωση, **αν υπάρχει τουλάχιστον ένας γρήγορος στα αριστερά** (αριστερή λίστα του πατέρα) τότε πρέπει να στείλουμε τους 2 πιο αργούς στην άλλη όχθη. Έτσι, η συνάρτηση ελέγχει αν η τωρινή κατάσταση είναι η επιθυμητή και βαθμολογεί με χαμηλό score αλλιώς δίνει πιο υψηλό score (ανάλογο του χρόνου διάσχισης και του συνολικού χρόνου του παιχνιδιού, δηλαδή `state_duration * (10 + game_time)`) στις υπόλοιπες καταστάσεις.

Τέλος, επιστρέφει το score.

Συνάρτηση find_position_in_list

Η συνάρτηση αυτή ταξινομεί την αριστερή λίστα του πατέρα-κόμβου και βρίσκει την θέση του ατόμου που μετακινείται ή την μέγιστη θέση του πιο αργού ατόμου όταν μετακινούνται δύο άτομα σε αυτήν και επιστρέφει την θέση αυτή.

Συνάρτηση get_children

Η συνάρτηση αυτή δημιουργεί και επιστρέφει τα παιδιά της τωρινής κατάστασης. Καλεί 2 συναρτήσεις : Την `moveIndividual()` η οποία δημιουργεί τα παιδιά στα οποία μετακινείται ένα άτομο και την `movePairs()` η οποία δημιουργεί τα παιδιά στα οποία μετακινούνται δύο άτομα.

Συνάρτηση create_child_state

Η συνάρτηση αυτή δημιουργεί μια νέα κατάσταση είτε για κίνηση ενός ατόμου είτε για κίνηση δύο ατόμων. Παίρνει ως παραμέτρους: Την δεξιά (right) και την αριστερή λίστα (left), τον φακό (torch) και δύο άτομα (p1 και p2). Η μεταβλητή p2 είναι *optional* σε περίπτωση που κινείται ένα άτομο.

Αφού δημιουργήσει την νέα κατάσταση, η συνάρτηση υπολογίζει τον χρόνο του παιχνιδιού μέχρι και αυτή την κατάσταση και ελέγχει αν αυτός ο χρόνος ξεπερνά τον μέγιστο επιτρεπόμενο. Αν τον ξεπερνά δεν εξερευνά αυτό το παιδί αλλιώς καλεί την `evaluate()` προκειμένου να αξιολογήσει την κατάσταση. Οπότε, είτε επιστρέφει την κατάσταση αυτή είτε επιστρέφει *None*.

Συνάρτηση moveIndividual

Η συνάρτηση αυτή δημιουργεί όλες τις πιθανές καταστάσεις που μπορούν να προκύψουν από μία κατάσταση με κίνηση ενός ατόμου. Η μέθοδος ελέγχει την θέση του φακού. Εάν ο φακός βρίσκεται στη *δεξιά* πλευρά, θα εξετάσει τη μετακίνηση ατόμου από τα δεξιά προς τα αριστερά. Διαφορετικά, αν είναι *αριστερά*, θα εξετάσει τη μετακίνηση ατόμου από τα αριστερά προς τα δεξιά. Για κάθε άτομο της αντίστοιχης λίστας, το διαγράφει από αυτήν και το τοποθετεί στην άλλη, αφού πρώτα πάρει αντίγραφα και των 2 λιστών για να μην τροποποιηθούν οι λίστες της τωρινής κατάστασης. Στη συνέχεια, καλεί την `create_child_state(self,right, left, torch, p1, p2=None)` η οποία δημιουργεί την αντίστοιχη κατάσταση-παιδί και τέλος επιστρέφει την λίστα με τα παιδιά που δημιούργησε.

Συνάρτηση movePairs

Η συνάρτηση αυτή δημιουργεί όλες τις πιθανές καταστάσεις που μπορούν να προκύψουν από μία κατάσταση με κίνηση δύο ατόμων. Η μέθοδος ελέγχει την θέση του φακού. Εάν ο φακός βρίσκεται στη *δεξιά* πλευρά, θα εξετάσει τη μετακίνηση ατόμων από τα δεξιά προς τα αριστερά. Διαφορετικά, αν είναι *αριστερά*, θα εξετάσει τη μετακίνηση ατόμων από τα αριστερά προς τα δεξιά. Για κάθε συνδιασμό δύο ατόμων της αντίστοιχης λίστας, τα διαγράφει από αυτήν και τα τοποθετεί στην άλλη, αφού πρώτα πάρει αντίγραφα και των 2 λιστών για να μην τροποποιηθούν οι λίστες της τωρινής κατάστασης. Στη συνέχεια, καλεί την `create_child_state(self,right, left, torch, p1, p2)` η οποία δημιουργεί την αντίστοιχη κατάσταση-παιδί και τέλος επιστρέφει την λίστα με τα παιδιά που δημιούργησε.

Συνάρτηση is_final

Η συνάρτηση αυτή υπολογίζει αν μια κατάσταση είναι τελική, δηλαδή αν βρήκαμε λύση στο πρόβλημα. Ελέγχει αν η δεξιά λίστα της κατάστασης είναι άδεια, διότι αυτό θα σήμαινε οτι όλα τα μέλη εχουν μετακινηθεί στα αριστερά και άρα ο στόχος του παιχνιδιού επιτεύχθη.

Κλάση spaceSearcher

Η κλάση αυτή εφαρμόζει τον αλγόριθμο αναζήτησης A*, για την εύρεση μιας διαδρομής από μια αρχική κατάσταση σε μια τελική κατάσταση σε ένα μέτωπο αναζήτησης.

Έχει πεδίο μια λίστα *frontier* η οποία χρησιμοποιείται ως ουρά προτεραιότητας και περιέχει τις καταστάσεις που δεν έχουν ακόμη εξερευνηθεί.

Συνάρτηση AStar

Παίρνει ως ορίσμα την αρχική κατάσταση και αφού κάνει έλεγχο αν είναι τελική κατάσταση, την προσθέτει στην ουρά προκείμενου να ξεκινήσει η διαδικασία δημιουργίας νέων κόμβων. Στη συνέχεια, ξεκινάει μια επαναληπτική διαδικασία η οποία σταματάει είτε οταν δεν υπάρχουν άλλες καταστάσεις στην ουρά είτε όταν βρίσκει μια τελική κατάσταση. Κάθε φορά βγάζει την κατάσταση με το μικρότερο κόστος *(currentState)* από την ουρά και ελέγχει αν είναι τελική κατάσταση. Στη συνέχεια, δημιουργεί τα παιδιά της

currentState και τα προσθέτει στην ουρά. Έτσι, η συνάρτηση επιστρέφει είτε *None* όταν δεν βρίσκει τελική κατάσταση είτε επιστρέφει τον τελικό κομβο.

Σημείωση: Έχουμε χρησιμοποιήσει ουρά προτεραιότητας για να εκμεταλλευτούμε την ιδιότητα της να βγάζει τα στοιχεία ταξινομημένα. Έτσι πάντα το πρώτο στοιχείο που θα βγάζει θα είναι αυτό με το μικρότερο κόστος.

Παραδείγματα εισόδου στο πρόγραμμα

1ο παράδειγμα (της περίπτωσης της εκφώνησης)

Run: python main.py 30 5 1 3 6 8 12

Input:

Time	**30**
Number of people	5
Person 1 (time)	1
Person 2 (time)	3
Person 3 (time)	6
Person 4 (time)	8
Person 5 (time)	12

Solution:

Torch Position	**Left Side**	**Right Side**	**Persons Moved**	**Remaining Time**
Right	[]	[1, 3, 6, 8, 12]	Game initialized	30
Left	[1, 3]	[6, 8, 12]	To left side: [1, 3]	27
Right	[1]	[3, 6, 8, 12]	To right side: [3]	24
Left	[1, 8, 12]	[3, 6]	To left side: [8, 12]	12
Right	[8, 12]	[1, 3, 6]	To right side: [1]	11
Left	[1, 3, 8, 12]	[6]	To left side: [1, 3]	8
Right	[3, 8, 12]	[1, 6]	To right side: [1]	7
Left	[1, 3, 6, 8, 12]	[]	To left side: [1, 6]	1

Total Time: 29

Search Time: 0.0015490055084228516 sec.

2ο παράδειγμα

Run: python main.py 45 6 2 4 6 8 10 12

Input:

Time	**45**
Number of people	6
Person 1 (time)	2
Person 2 (time)	4
Person 3 (time)	6
Person 4 (time)	8
Person 5 (time)	10
Person 6 (time)	12

Solution:

Torch Position	**Left Side**	**Right Side**	**Persons Moved**	**Remaining Time**
Right	[]	[2, 4, 6, 8, 10, 12]	Game initialized	45
Left	[2, 4]	[6, 8, 10, 12]	To left side: [2, 4]	41
Right	[2]	[4, 6, 8, 10, 12]	To right side: [4]	37
Left	[2, 10, 12]	[4, 6, 8]	To left side: [10, 12]	25
Right	[10, 12]	[2, 4, 6, 8]	To right side: [2]	23
Left	[2, 4, 10, 12]	[6, 8]	To left side: [2, 4]	19
Right	[2, 10, 12]	[4, 6, 8]	To right side: [4]	15
Left	[2, 6, 8, 10, 12]	[4]	To left side: [6, 8]	7
Right	[6, 8, 10, 12]	[2, 4]	To right side: [2]	5
Left	[2, 4, 6, 8, 10, 12]	[]	To left side: [2, 4]	1

Total Time: 44

Search Time: 0.0031228065490722656 sec.

3ο παράδειγμα

Run: python main.py 70 9 1 3 5 7 9 11 13 15 17

Input:

Time	**70**
Number of people	9
Person 1 (time)	1
Person 2 (time)	3
Person 3 (time)	5
Person 4 (time)	7
Person 5 (time)	9
Person 6 (time)	11
Person 7 (time)	13
Person 8 (time)	15
Person 9 (time)	17

Solution:

Torch Position	**Left Side**	**Right Side**	**Persons Moved**	**Remaining Time**
Right	[]	[1, 3, 5, 7, 9, 11, 13, 15, 17]	Game initialized	70
Left	[1, 3]	[5, 7, 9, 11, 13, 15, 17]	To left side: [1, 3]	67
Right	[1]	[3, 5, 7, 9, 11, 13, 15, 17]	To right side: [3]	64
Left	[1, 15, 17]	[3, 5, 7, 9, 11, 13]	To left side: [15, 17]	47
Right	[15, 17]	[1, 3, 5, 7, 9, 11, 13]	To right side: [1]	46
Left	[1, 3, 15, 17]	[5, 7, 9, 11, 13]	To left side: [1, 3]	43
Right	[3, 15, 17]	[1, 5, 7, 9, 11, 13]	To right side: [1]	42
Left	[3, 11, 13, 15, 17]	[1, 5, 7, 9]	To left side: [11, 13]	29
Right	[11, 13, 15, 17]	[1, 3, 5, 7, 9]	To right side: [3]	26
Left	[1, 3, 11, 13, 15, 17]	[5, 7, 9]	To left side: [1, 3]	23
Right	[1, 11, 13, 15, 17]	[3, 5, 7, 9]	To right side: [3]	20
Left	[1, 7, 9, 11, 13, 15, 17]	[3, 5]	To left side: [7, 9]	11
Right	[7, 9, 11, 13, 15, 17]	[1, 3, 5]	To right side: [1]	10

Torch Position	Left Side	Right Side	Persons Moved	Remaining Time
Left	[1, 3, 7, 9, 11, 13, 15, 17]	[5]	To left side: [1, 3]	7
Right	[3, 7, 9, 11, 13, 15, 17]	[1, 5]	To right side: [1]	6
Left	[1, 3, 5, 7, 9, 11, 13, 15, 17]	[]	To left side: [1, 5]	1

Total Time: 69

Search Time: 0.018772602081298828 sec.