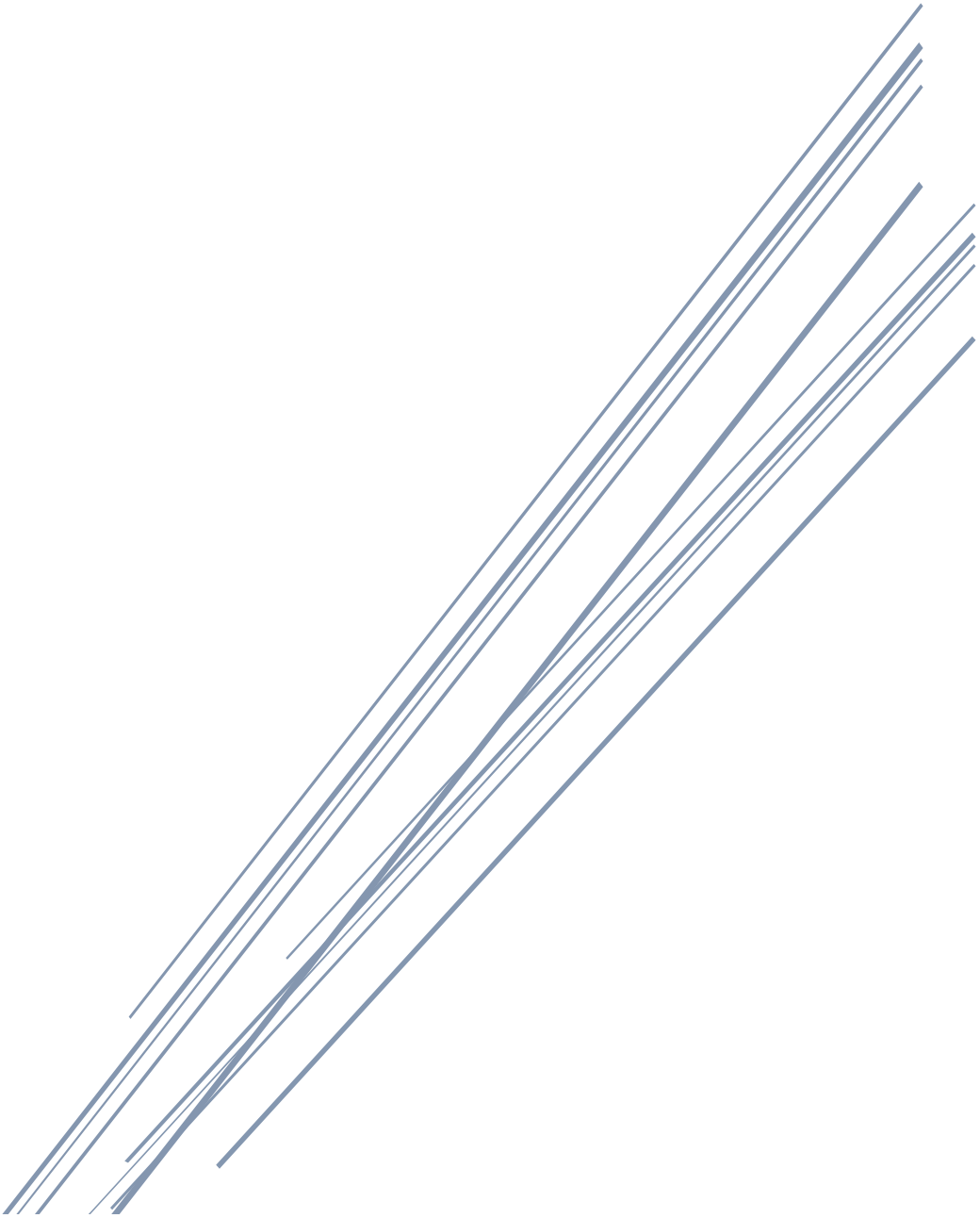
Ευφυείς Πράκτορες

Απαλλακτική Εργασία

Ακαδημαϊκό Έτος 2020 - 2021



ΠΑΝΕΠΙΣΤΗΜΙΟ ΠΕΙΡΑΙΩΣ
UNIVERSITY OF PIRAEUS

Παναγιώτης Αποστολόπουλος, Π17007
Δημήτρης Ματσαγγάνης, Π17068
Παύλος Ρουμελιώτης, Π17112
Σκαρπέλος Αλέξανδρος, Π17122

Περιεχόμενα

1. Εκφώνηση Εργασίας 2
2. Εισαγωγή 4
3. Blocks World 6
   3.1 Χώρος Καταστάσεων 7
   3.2 Συναρτήσεις 7
      3.2.1 Συνάρτηση Κόστους 7
      3.2.2 Ευρυστική Συνάρτηση 8
   3.3 Δομές Δεδομένων 9
      3.3.1 Κλάση BlocksNode 9
4. Water Jug 10
   4.1 Χώρος Καταστάσεων 10
   4.2 Συναρτήσεις 11
      4.2.1 Συνάρτηση Κόστους 11
      4.2.2 Ευρυστική Συνάρτηση 11
   4.3 Δομές Δεδομένων 12
      4.3.1 Κλάση WaterNode 12
5. A-Star (A*) 14
   5.1 Κλάση PriorityHeap 15
   5.2 Εφαρμογή του A* 16
6. Παραδείγματα Εκτέλεσης 17
   6.1 Blocks World 17
   6.2 Water Jug 20
7. Μελλοντικές Επεκτάσεις – Χειρισμός Εξαιρέσεων 23
8. Απαιτήσεις συστήματος και εκτέλεση 26
9. Βιβλιογραφία 31

1. Εκφώνηση Εργασίας

Ανάπτυξη generic planner

Περιγραφή

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 στιγμιότυπου 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.

1. Αναζητείστε κώδικα ενός generic planner στο διαδίκτυο, ή αναπτύξτε τον δικό σας κώδικα.
2. Τρέξτε τον planner για 2 διαφορετικά προβλήματα : το Blocks World και το Water jug

Δείτε τις παρακάτω πηγές:

- Blocks World
- Water Pouring Puzzle

Τεκμηρίωση

Η τεκμηρίωση της εφαρμογής θα περιλαμβάνει τα εξής:

1. Περιγραφή του προβλήματος

2. Περιγραφή της θεωρητικής βάσης της εφαρμογής, συμπεριλαμβανομένων των δομών δεδομένων και αναπαράστασης γνώσης που υιοθετήθηκαν, των αλγορίθμων και μεθοδολογιών που χρησιμοποιήθηκαν, καθώς και των προσαρμογών και μεταβολών που

έγιναν στα παραπάνω προκειμένου να είναι δυνατή η εφαρμογή τους στο συγκεκριμένο πρόβλημα

3. Περιγραφή σημαντικών σχεδιαστικών αποφάσεων και στοιχείων υλοποίησης

4. Ολοκληρωμένη περιγραφή μίας παραδειγματικής εκτέλεσης και των αποτελεσμάτων της τόσο για το Blocks World όσο και για το Water Jug

5. Αναλυτική περιγραφή της διαδικασίας εγκατάστασης

6. Περιγραφή πρόσθετων δυνατοτήτων της εφαρμογής, εάν υπάρχουν

8. Αναλυτική περιγραφή ανοικτών θεμάτων, ανεπίλυτων προβλημάτων και πιθανοτήτων εμφάνισης σφαλμάτων κατά την εκτέλεση

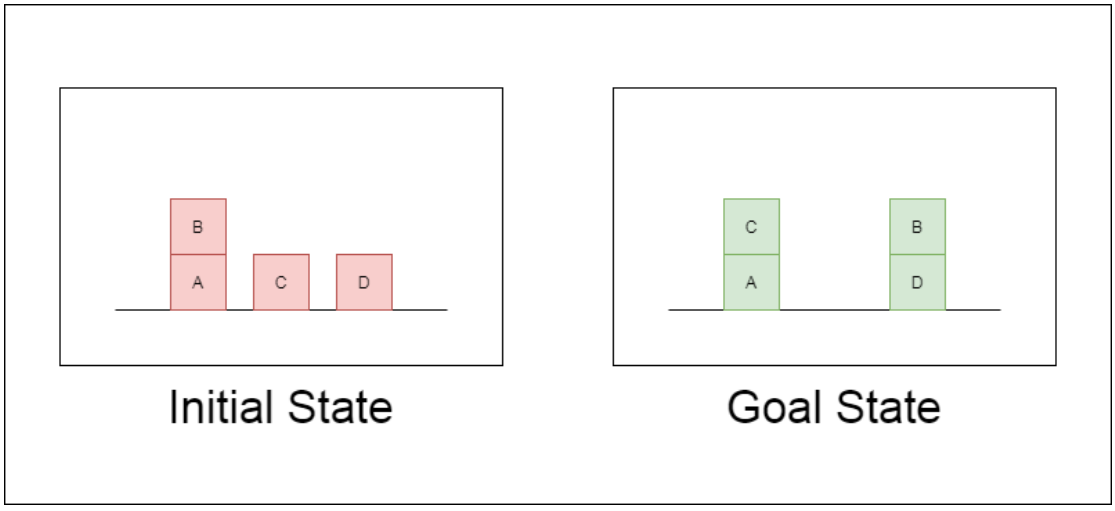
Είναι σημαντικό να υπάρχουν αναλυτικά και επεξηγημένα screenshots από την εκτέλεση της εφαρμογής.

Η εφαρμογή θα διαθέτει άμεσα εκτελέσιμο πρόγραμμα που δεν θα απαιτεί να κατέβουν διάφορα περιβάλλοντα για να εκτελεστεί.

2. Εισαγωγή

Ζητούμενο της συγκεκριμένης εργασίας είναι η υλοποίηση ενός *Generic Planner*, ο οποίος θα μπορεί να επιλύει τα προβλήματα του Blocks World και του Water Jug.

Αρχικά, το πρόβλημα του *Blocks World* είναι ένας από τα πιο γνωστά προβλήματα της τεχνητή νοημοσύνης. Ο στόχος του συγκεκριμένου αλγορίθμου είναι να οικοδομήσουμε μία ή περισσότερες κάθετες στοίβες από μπλοκ. Οι κανόνες του προβλήματος είναι ότι μόνο ένα μπλοκ μπορεί να μετακινηθεί κάθε φορά, δηλαδή μπορεί είτε να τοποθετηθεί στο τραπέζι, είτε να τοποθετηθεί πάνω σε κάποιο άλλο μπλοκ. Εξαιτίας των παραπάνω, στην περίπτωση που ένα μπλοκ μια δεδομένη στιγμή, βρίσκεται κάτω από ένα άλλο μπλοκ, τότε δεν μπορεί να μετακινηθεί.



Εικόνα 1: Παράδειγμα Blocks World

Προκειμένου να επιλυθεί το παρών πρόβλημα, θα πρέπει να ευρεθεί το σύνολο των πιθανών καταστάσεων, οι οποίες αξιολογούνται από μια ευρυστική συνάρτηση, με τη βοήθεια της οποίας υπολογίζεται το κόστος μετάβασης. Συνεπώς, με τη βοήθεια του *Generic Planner*, αναζητούμε την κατάλληλη αλληλουχία βημάτων, ώστε να καταλήξουμε με βέλτιστο τρόπο στην τελική κατάσταση, την οποία έχει ορίσει ο χρήστης κατά την εκτέλεση του προγράμματος.

Ένα άλλο πρόβλημα, το οποίο αποτελεί ένα από τα πλέον γνωστά στον τομέα της τεχνητής νοημοσύνης είναι το *Water Pouring Puzzle (Water Jug)*.Μέσω του *Γεννήτορα Σχεδίων (Generic Planner),* που επιλύει και το

πρόβλημα του *Blocks World*, παράγει μία ακολουθία ενεργειών, η οποία μας οδηγεί στη βέλτιστη λύση.

Πιο συγκεκριμένα, θεωρούμε ότι κατά την εκκίνηση του προβλήματος οι δύο κανάτες είναι άδειες. Ο χρήστης μπορεί τόσο να ορίσει τη χωρητικότητα (σε λίτρα), όσο και την τελική κατάσταση για την κάθε κανάτα. Οι επιτρεπτές κινήσεις του προβλήματος είναι το γέμισμα των κανατών, το άδειασμα αυτών, αλλά και το γέμισμα της μιας από την άλλη.

Ακολούθως, προκειμένου να εκτελεστεί, με τρόπο ορθολογικό, ο *Generic Planner*, ενδείκνυται να ορίσουμε χώρο – σύνολο των καταστάσεων.

Έπειτα, μέσω του αλγορίθμου *A Star (A*)*, μίας συνάρτησης κόστους, η οποία θα υλοποιεί μία ευρυστικής συνάρτηση, επιλέγουμε την βέλτιστη κίνηση του αλγορίθμου μέχρις ότου το πρόγραμμα να φτάσει στην τελική κατάσταση που έχει ορισθεί από τον χρήστη.

Μετά την ολοκλήρωση του αλγορίθμου, θα εκτυπωθεί στην οθόνη ο αριθμός των απαιτούμενων βημάτων, καθώς και η λίστα με τη αλληλουχία των κινήσεων - ενεργειών που θα πρέπει να ακολουθηθούν για να οδηγηθεί και ο χρήστης στη λύση.

Σημαντικό ρόλο στη δημιουργία του *Generic Planner*, διαδραματίζει ο αλγόριθμος *A Star (A*)*. Ο εν λόγω αλγόριθμος βασίζεται στη δημιουργία ελαχίστων μονοπατιών μέσα από μια ακολουθία από συγκεκριμένες ενέργειες. Πρόκειται για έναν αλγόριθμο αναζήτησης, ο οποίος διερευνά το πρόβλημα με σκοπό την εύρεση της ελάχιστης διαδρομής μεταξύ της αρχικής και της τελικής κατάστασης. Οι εφαρμογές του συγκεκριμένου αλγορίθμου δεν περιορίζονται μόνο στην επίλυση των παρόντων προβλημάτων, αλλά αποτελούν ένα χρήσιμο εργαλείο σε τομείς όπως η αστρονομία, η δημιουργία αυτόνομων ρομπότ τεχνητής νοημοσύνης, αυτοκαθοδηγούμενων αυτοκινήτων αλλά και σε πολλές εκφάνσεις της ιατρικής.

3. Blocks World

Με τον συγκεκριμένο *Γεννήτορα Σχεδίων (Generic Planner),* παράγεται μία ακολουθία ενεργειών, η οποία δύναται να επιλύσει το πρόβλημα του *Blocks World*.

Πιο συγκεκριμένα, κατά την εκκίνηση του προβλήματος, ο χρήστης ορίζει την αρχική και την τελική κατάσταση, η οποία αποτελεί τον τρόπο με τον οποίο είναι τοποθετημένα τα blocks στο τραπέζι. Άξιο αναφοράς αποτελεί το γεγονός, ότι κατά τη διαδικασία εκτέλεσης η αρχική και η τελική κατάσταση πρέπει να πληρούν κάποιες προϋποθέσεις:

- ***[A-Z,a-z,0-9]*[,]([A-Z,a-z,0-9]*[,]*)***, δηλαδή να είναι τουλάχιστον δύο στοίβες, όπου η καθεμία μπορεί να έχει blocks που ορίζονται με κάποιο κεφαλαίο ή πεζό γράμμα του Αγγλικού αλφαβήτου, ή με κάποιο ψηφίο.
- Ο διαχωρισμός των στοιβών γίνεται με το κόμμα (“**,**”).
- Η κενή στοίβα ορίζεται με κενό (“” ή “ ”).
- Ο αριθμός των στοιβών της τελικής κατάστασης δεν πρέπει να είναι μικρότερος από τον αριθμό των στοιβών της αρχικής κατάστασης.

Στη συνέχεια, προκειμένου να εκτελεστεί επιτυχώς ο αλγόριθμος, θα πρέπει να ορίσουμε το σύνολο των καταστάσεων. Ειδική αναφορά για το εν λόγω ζήτημα θα γίνει στη σχετική ενότητα.

Έπειτα, μέσω του αλγορίθμου *A Star (A*)*, μίας συνάρτησης κόστους, η οποία θα υλοποιεί μία ευρυστική συνάρτηση, επιλέγουμε την αλληλουχία των βέλτιστων κινήσεων του *Generic* Planner μέχρις ότου το πρόγραμμα να φτάσει στην τελική κατάσταση, όπως αυτή έχει ορισθεί από τον χρήστη. Σαν βέλτιστη κίνηση, ορίζουμε την κίνηση με το μικρότερο κόστος, όπως αυτό υπολογίζεται από την αντίστοιχη μέθοδο.

Όταν ο αλγόριθμός ολοκληρωθεί, θα εκτυπωθεί στην οθόνη τόσο τον αριθμό των απαιτούμενων βημάτων, αλλά και τη σχετική λίστα με την αλληλουχία των κινήσεων - ενεργειών που θα πρέπει να ακολουθηθούν για να οδηγηθεί και ο χρήστης στη λύση με το μικρότερο κόστος.

3.1 Χώρος Καταστάσεων

Στην ενότητα θα αναφερθούμε κατά βάθος στο χώρο των καταστάσεων του προβλήματος.

Πιο συγκεκριμένα, η αναπαράσταση του χώρου καταστάσεων σε συνδυασμό με τον υπολογισμό του κόστους μετάβασης αξιοποιούνται από τον *Generic Planner* για την επίλυση του συγκεκριμένου προβλήματος.

Ως εκ τούτου, για την εύρεση των καταστάσεων των παιδιών επιλέγεται το μπλοκ που βρίσκεται στην κορυφή κάθε στοίβας και τοποθετώντας το στην κορυφή των υπόλοιπων στοιβών. Με αυτό τον τρόπο, δημιουργείται μία λίστα που εμπεριέχει όλες τις δυνατές καταστάσεις που μπορούν να προκύψουν από την παρούσα.

Αυτή η διαδικασία πραγματοποιείται μέσω της συνάρτησης *getChildren.*

3.2 Συναρτήσεις

Στη παρούσα ενότητα θα δοθεί ιδιαίτερη σημασία στις συναρτήσεις που υλοποιούνται μέσα στην υλοποίησή μας.

3.2.1 Συνάρτηση Κόστους

Η πλέον σημαντική συνάρτηση στην υλοποίησή μας είναι η συνάρτηση του κόστους, η οποία μας οδηγεί στην επιλογή του μονοπατιού (path) με το μικρότερο κόστος για το εκάστοτε βήμα. Η διαδικασία στη πράξη γίνεται μέσω της συνάρτησης *getCost,* που βρίσκεται μέσα στη κλάση *BlocksNode.*

Αναλυτικότερα, το κόστος ενός βήματος – μίας κίνησης ορίζεται ως το κόστος του μονοπατιού μέχρι εκείνο το βήμα, συνδυαστικά με το αποτέλεσμα της ευρυστικής συνάρτησης για τη συγκεκριμένη κατάσταση (κίνηση).

3.2.2 Ευρυστική Συνάρτηση

Στη παρούσα ενότητα θα γίνει ειδική μνεία στην ευρυστική συνάρτηση, η οποία σε συνδυασμό με την συνάρτηση του κόστους μας οδηγεί στην επιλογή της βέλτιστης κίνησης για το κάθε βήμα.

Ως ευρυστική συνάρτηση ορίζουμε το άθροισμα των μπλοκ που βρίσκονται σε λανθασμένη θέση συνδυαστικά με τον αριθμό των μπλοκ, όπου το από κάτω τους μπλοκ βρίσκεται σε λάθος θέση.

Ειδικότερα, για την εκάστοτε στοίβα μίας κατάστασης αρχικά ελέγχει εάν η στοίβα είναι κενή, τότε ο έλεγχος συνεχίζεται στην επόμενη στοίβα. Επιπλέον, στην περίπτωση που ο αριθμός των μπλοκ της στοίβας στη παρούσα κατάσταση είναι μικρότερος ή ίσος με τον αριθμό των μπλοκ της αντίστοιχης στοίβας στην τελική κατάστασης, τότε γίνεται έλεγχος για την ορθότητα, τόσο της θέσης του κάθε μπλοκ, όσο και για αυτά που βρίσκονται από κάτω του. Ακόμη, στην περίπτωση που ο αριθμός των μπλοκ της στοίβας στη παρούσα κατάσταση είναι μεγαλύτερος από τον αριθμό των μπλοκ της αντίστοιχης στοίβας στην τελική κατάστασης, τότε ελέγχεται εάν η στοίβα στην τελική κατάσταση είναι κενή και σε αυτή την περίπτωση ο αλγόριθμος αντιλαμβάνεται ότι τα μπλοκ που υφίστανται στη συγκεκριμένη στοίβας την παρούσα κατάσταση βρίσκονται σε λανθασμένη θέση, αλλιώς γίνεται έλεγχος με βάση τον αριθμό των μπλοκ της στοίβας στην τελική κατάσταση, στην παρούσα και ανάλογα με τα αποτελέσματα αυτού ο αλγόριθμος συμπεραίνει πόσα μπλοκ είναι σε λανθασμένη θέση και χρειάζονται μετακίνηση.

Η συνάρτηση που υλοποιεί τα παραπάνω είναι η *blocksHeuristic.*

3.3 Δομές Δεδομένων

Στο σημείο αυτό, θα παρουσιαστούν οι δομές δεδομένων που χρησιμοποιήθηκαν για την επίλυση του προβλήματος:

- Κλάση *BlocksNode*
- Γίνεται χρήση λιστών και πλειάδων (tuples), οι οποίες εξυπηρετούν συγκεκριμένους σκοπούς και διευκολύνουν την επίλυση του προβλήματος.

3.3.1 Κλάση BlocksNode

Με την κλάση *BlocksNode* δημιουργείται ο κόμβος (*node*) για την κάθε κατάσταση.

Η παρούσα συνάρτηση δέχεται σαν χαρακτηριστικά την κατάσταση, τους γονείς, το κόστος, και το αποτέλεσμα της ευρυστικής συνάρτησης.

Ειδικότερα, η παρούσα κλάση δέχεται σαν ορίσματα την αρχική κατάσταση, που είναι το όρισμα του χρήστη κατά την εκκίνηση του αλγόριθμου, το αρχικό μονοπάτι – path που είναι μία κενή λίστα, το αρχικό κόστος και το αρχικό heuristic που ισούνται αμφότερα με 0.

Ακόμη, υπάρχει η συνάρτηση *getChildren*, η οποία δημιουργεί ένα *BlocksNode* για το κάθε παιδί που μπορεί να προκύψει από την κατάσταση που βρισκόμαστε και τοποθετεί όλα τα παιδία σε μία λίστα. Τη λειτουργεία της συνάρτησης *getChildren* την έχουμε αναφέρει στην Ενότητα 3.1 .

Η συνάρτηση *getCost*, η οποία υπολογίζει το συνολικό κόστος μίας κατάστασης και έχει γίνει εκτενής αναφορά στην Ενότητα 3.2.1 .

Η συνάρτηση *checkGoal*, ελέγχει εάν βρισκόμαστε στην ορισμένη από το χρήστη, τελική κατάσταση και στην περίπτωση που βρισκόμαστε σε αυτή, τότε εκτυπώνει τον αριθμό των απαιτούμενων κινήσεων και πραγματοποιεί backtracking για την ανάδειξη της κάθε κατάστασης του μονοπατιού.

4. Water Jug

Ο συγκεκριμένος *Γεννήτορας Σχεδίων (Generic Planner),* παράγει μία ακολουθία ενεργειών, η οποία δύναται να επιλύσει το πρόβλημα του Water Jug.

Ειδικότερα, κατά την εκκίνηση του προβλήματος οι δύο κανάτες είναι άδειες. Ο χρήστης έχει τη δυνατότητα τόσο να ορίσει τη χωρητικότητα (σε λίτρα), όσο και την τελική κατάσταση για την κάθε κανάτα.

Στη συνέχεια, προκειμένου να εκτελεστεί, με τρόπο ορθολογικό, ο Generic Planner, ενδείκνυται να ορίσουμε χώρο – σύνολο των καταστάσεων. Ειδική αναφορά για το εν λόγο ζήτημα θα γίνει σε παρακάτω ενότητα.

Έπειτα, μέσω του αλγορίθμου *A Star (A*)*, μίας συνάρτησης κόστους, η οποία θα υλοποιεί μία ευρυστική συνάρτηση, επιλέγουμε την βέλτιστη κίνηση του αλγορίθμου μέχρις ότου το πρόγραμμα να φτάσει στην τελική κατάσταση που έχει ορισθεί από τον χρήστη. Ως βέλτιστή κίνηση, ορίζουμε την κίνηση με το μικρότερο κόστος, όπως αυτό υπολογίζεται από την αντίστοιχη μέθοδο.

Μόλις ο αλγόριθμός ολοκληρωθεί, θα εμφανιστεί ο αριθμός των απαιτούμενων βημάτων, καθώς και μία λίστα με τη αλληλουχία των κινήσεων - ενεργειών που θα πρέπει να ακολουθηθούν για να οδηγηθεί και ο χρήστης στη λύση με το μικρότερο κόστος.

4.1 Χώρος Καταστάσεων

Σε αυτή την ενότητα θα αναφερθούμε εκτενώς στο χώρο των καταστάσεων του προβλήματος.

Η αναπαράσταση του χώρου καταστάσεων σε συνδυασμό με τον υπολογισμό του κόστους μετάβασης αξιοποιούνται από τον *Generic Planner* για την επίλυση του συγκεκριμένου προβλήματος.

Ειδικότερα, στη περίπτωση που η παρούσα ποσότητα νερού της κανάτας 1 (αντίστοιχα κανάτα 2) είναι μικρότερη από την αρχική της, τότε η ενέργεια που πρέπει να πραγματοποιηθεί είναι το γέμισμα της κανάτας 1 (αντίστοιχα κανάτα 2). Εάν η παρούσα ποσότητα νερού της κανάτας 1 (αντίστοιχα κανάτα 2) είναι μεγαλύτερη του μηδενός, τότε αδειάζουμε την κανάτα 1 (αντίστοιχα κανάτα 2). Επιπλέον, στην περίπτωση που το

άθροισμα των ποσοτήτων των δύο κανατών είναι μικρότερο ή ίσο από την αρχική ποσότητα της κανάτας 1 (αντίστοιχα κανάτα 2), τότε αδειάζουμε όλη την κανάτα 2 στην κανάτα 1 (αντίστοιχα την κανάτα 1 στην κανάτα 2). Τέλος, στην περίπτωση που το άθροισμα των ποσοτήτων των δύο κανατών είναι μεγαλύτερο από την αρχική ποσότητα της κανάτας 1 (αντίστοιχα κανάτα 2), τότε γεμίζουμε την κανάτα 1 από την κανάτα 2 (αντίστοιχα την κανάτα 2 από την κανάτα 1).

Αυτή η διαδικασία πραγματοποιείται μέσω της συνάρτησης *getChildren* της κλάσης *WaterNode* και της βοηθητικής συνάρτησης *getWaterChildren.*

4.2 Συναρτήσεις

Στη παρούσα ενότητα θα δοθεί ιδιαίτερη σημασία στις συναρτήσεις που υλοποιούνται μέσα στο πρόγραμμά μας.

4.2.1 Συνάρτηση Κόστους

Καθοριστική σημασία στην υλοποίησή μας διαδραματίζει η συνάρτηση του κόστους, η οποία μας οδηγεί στην επιλογή του μονοπατιού (path) με το μικρότερο κόστος για το εκάστοτε βήμα.

Ειδικότερα, το κόστος ενός βήματος – μίας κίνησης ορίζεται ως το κόστος του μονοπατιού μέχρι εκείνο το βήμα, συνδυαστικά με το αποτέλεσμα της ευρυστικής συνάρτησης για τη συγκεκριμένη κατάσταση (κίνηση).

Η συνάρτηση κόστους είναι η *getCost* της κλάσης *WaterNode.*

4.2.2 Ευρυστική Συνάρτηση

Η ευρυστική συνάρτηση σε συνδυασμό με την συνάρτηση του κόστους μας οδηγεί στην επιλογή της βέλτιστης κίνησης για το κάθε βήμα.

Ως ευρυστική συνάρτηση ορίζουμε:

$$\text{heuristic} = |(CurrentCapacity_{Κανατα1} - GoalCapacity_{Κανατα1}) + (CurrentCapacity_{Κανατα2} - GoalCapacity_{Κανατα2})|$$

Πιο συγκεκριμένα, για την κάθε κανάτα αφαιρούμε την τελική ποσότητα νερού από την παρούσα, όπως ο χρήστης τις έχει ορίσει και προσθέτουμε το αποτέλεσμα αυτής της διαδικασίας. Το αποτέλεσμα πρέπει να είναι

πάντα θετικός αριθμός, για αυτό και η διαδικασία ολοκληρώνεται λαμβάνοντας το αποτέλεσμα μέσω απολύτων τιμών.

Η συνάρτηση αυτή είναι η *waterHeuristic.*

4.3 Δομές Δεδομένων

Στο σημείο αυτό, θα παρουσιαστούν οι δομές δεδομένων που χρησιμοποιήθηκαν για την επίλυση του προβλήματος:

- Κλάση WaterNode
- Επιπλέον, αξίζει να σημειωθεί ότι γίνεται χρήση λιστών και πλειάδων (tuples), οι οποίες εξυπηρετούν συγκεκριμένους σκοπούς και διευκολύνουν την επίλυση του προβλήματος.

4.3.1 Κλάση WaterNode

Μέσω της κλάσης *WaterNode* δημιουργείται ο κόμβος (*node*) για την κάθε κατάσταση.

Αναλυτικότερα, η παρούσα κλάση δέχεται σαν ορίσματα την αρχική κατάσταση, που είναι πάντα (0,0), το αρχικό μονοπάτι – path που είναι μία κενή λίστα, το αρχικό κόστος και το αρχικό heuristic που ισούνται αμφότερα με 0.

Ακόμη, υπάρχει η συνάρτηση *getChildren*, η οποία δημιουργεί ένα *WaterNode* για το κάθε παιδί που μπορεί να προκύψει από την κατάσταση που βρισκόμαστε. Οι διάφορες καταστάσεις αναπαρίστανται ως μία πλειάδα (jug1, jug2), όπου το jug1 αναφέρεται στην ποσότητα του νερού της κανάτας 1, ενώ jug2 στην ποσότητα της δεύτερης. Γενικά ισχύει, ότι:

$$0 \leq jug1 \leq cap1\ ,$$

όπου cap1 η χωρητικότητα της κανάτας 1, όπως την έχει ορίσει ο χρήστης.

Επίσης, ισχύει και το εξής:

$$0 \leq jug2 \leq cap2\ ,$$

όπου cap2 η χωρητικότητα της κανάτας 2, όπως την έχει ορίσει ο χρήστης.

Ως αρχική κατάσταση θεωρείται η (0,0), όπου οι δύο κανάτες είναι άδειες.

Το σύνολο των δυνατών καταστάσεων περιγράφεται αναλυτικά στον παρακάτω πίνακα.

Περιγραφή Κατάστασης	Αρχική Κατάσταση	Νέα Κατάσταση
Γέμισε την κανάτα 1	(jug1, jug2), jug1 < cap1	(cap1, 0)
Γέμισε την κανάτα 2	(jug1, jug2), jug2 < cap2	(0, cap2)
Άδειασε την κανάτα 1	(jug1, jug2), jug1 > 0	(0, cap2)
Άδειασε την κανάτα 2	(jug1, jug2), jug2 > 0	(cap1, 0)
Άδειασε νερό από την κανάτα 2 στην 1	(jug1, jug2), $0 < jug1+jug2 \geq cap1, jug2 > 0$	(cap1, jug2 – (cap1 – jug1))
Άδειασε νερό από την κανάτα 1 στην 2	(jug1, jug2), $0 < jug1+jug2 \geq cap2, jug1 > 0$	(jug1 – (cap2 – jug2), cap2)
Άδειασε την κανάτα 2 στην 1	(jug1, jug2), $0 < jug1+jug2 \leq cap1, jug2 \geq 0$	(jug1 + jug2, 0)
Άδειασε την κανάτα 1 στην 2	(jug1, jug2), $0 < jug1+jug2 \leq cap2, jug1 \geq 0$	(0, jug1 + jug2)

Η συνάρτηση *getWaterChildren* γίνεται κλήση κατά τη διάρκεια της εκτέλεσης της *getChildren* και κύριο μέλημα της είναι η εύρεση όλων των δυνατών ενεργειών που θα οδηγήσουν στην εύρεση των παιδιών της παρούσας κατάστασης.

Η συνάρτηση *getCost,* η οποία υπολογίζει το συνολικό κόστος μίας κατάστασης και έχει γίνει εκτενής αναφορά στην Ενότητα 4.2.1 .

Η συνάρτηση *checkGoal*, ελέγχει εάν βρισκόμαστε στην ορισμένη από το χρήστη, τελική κατάσταση και στην περίπτωση που βρισκόμαστε σε αυτή, τότε εκτυπώνει τον αριθμό των απαιτούμενων κινήσεων και τη λίστα με το path.

5. A-Star (A*)

Ο αλγόριθμος *A** είναι ένα αλγόριθμος αναζήτησης γραφήματος και αναζήτησης διαδρομής, ο οποίο χρησιμοποιείται συχνά σε πολλούς τομείς της επιστήμης των υπολογιστών λόγω της πληρότητας του και της βέλτιστης απόδοσής του.

Πιο αναλυτικά, ξεκινώντας από έναν συγκεκριμένο κόμβο εκκίνησης ενός γραφήματος, έχει ως σκοπό την εύρεση μιας διαδρομής προς τον κόμβο στόχο, με το ελάχιστο κόστος.

Σε κάθε επανάληψη, ο αλγόριθμος πρέπει να καθορίσει σε ποια από τις διαθέσιμες διαδρομές θα επεκταθεί. Στην συγκεκριμένη υλοποίηση αυτό επιτυγχάνεται με το κόστος του μονοπατιού μέχρι το συγκεκριμένο κόμβο και μια εκτίμηση κόστους μέχρι την τελική κατάσταση. Αναγκαία και ικανή συνθήκη για την επιλογή μιας διαδρομής είναι η ελαχιστοποίηση της παρακάτω συνάρτησης

$$f(n) = g(n) + h(n)\ ,$$

όπου *n* είναι ο επόμενος κόμβος του μονοπατιού, *g(n)* το κόστος της διαδρομής από τον αρχικό κόμβο και *h(n)* η τιμή της ευρυστικής συνάρτησης.

Ο *A** τερματίζεται όταν βρεθεί η τελική κατάσταση ή όταν δεν υπάρχει η βέλτιστη αλληλουχία βημάτων για την επίλυση ενός προβλήματος.

5.1 Κλάση PriorityHeap

Η κλάση *PriorityHeap* εμπεριέχει τρεις συναρτήσεις *push*, *pop* και *isEmpty*, μέσω των οποίων δημιουργεί μία στοίβα που ταξινομείται με βάση την τιμή προτεραιότητας, που ορίζουμε για το κάθε αντικείμενο που εισάγεται.

Η *push,* δέχεται ως όρισμα ένα αντικείμενο και την τιμή προτεραιότητάς του και στην συνέχεια ταξινομεί κατάλληλα τη στοίβα (τα αντικείμενα με χαμηλότερη τιμή προτεραιότητας, κατατάσσονται στην αρχή της λίστας και θα επιλεχθούν γρηγορότερα).

Η *pop,* εξάγει το αντικείμενο με τη χαμηλότερη τιμή προτεραιότητας (το πρώτο αντικείμενο της λίστας).

Η *isEmpty,* είναι η συνάρτηση που ελέγχει εάν η λίστα προτεραιοτήτων είναι κενή.

5.2 Εφαρμογή του A*

Αρχικά, για την εφαρμογή του *A** εκτελούμε την συνάρτηση *AStar*, που δέχεται σαν ορίσματα τον αρχικό κόμβο της κλάσης του εκάστοτε προβλήματος (*BlocksNode* ή *WaterNode*) και την αντίστοιχη ευρυστική συνάρτηση (*blocksHeuristic* ή *waterHeuristic*).

Έπειτα, δημιουργείται ένα αντικείμενο της κλάσης *PriorityHeap*, καθώς και μια κενή λίστα που αντιπροσωπεύει το κλειστό σύνολο, δηλαδή τις καταστάσεις που έχουμε ήδη επισκεφτεί. Αφού δεχτεί σαν όρισμα τον αρχικό κόμβο τον εισαγάγει στο *PriorityHeap* και δέχεται σαν τιμή προτεραιότητας το συνολικό κόστος του. Για κάθε επανάληψη ελέγχουμε αν το *PriorityHeap* είναι άδειο. Εάν αυτός ο έλεγχος επαληθευτεί επιστρέφεται μήνυμα για την αδυναμία εύρεσης λύσης με ΄βάση την αρχική και τελική κατάσταση.

Σε διαφορετική περίπτωση, παίρνουμε τον κόμβο με το χαμηλότερο συνολικό κόστος και ελέγχουμε εάν αυτός αποτελεί την τελική κατάσταση. Ακόμα με τον εν λόγω αλγόριθμο, εάν δεν έχουμε επισκεφτεί ξανά τον κόμβο τον εισάγουμε στο κλειστό σύνολο και βρίσκουμε τα παιδιά του. Τέλος, προσθέτουμε στο *PriorityHeap* το κάθε παιδί προσθέτοντας σαν τιμή προτεραιότητας το συνολικό κόστος με την βοήθεια της αντίστοιχης ευρυστικής συνάρτησης.

Η παραπάνω διαδικασία επαναλαμβάνεται έως ότου αδειάσει το *PriorityHeap* ή βρεθεί η τελική κατάσταση.

6. Παραδείγματα Εκτέλεσης

Στην παρούσα ενότητα θα παρουσιαστούν τα αποτελέσματα από την εκτέλεση του προγράμματος.

6.1 Blocks World

Ακολουθούν παραδείγματα για το *Blocks World.*

Στο πρώτο παράδειγμα ο χρήστης εισάγει σαν αρχική κατάσταση την:

AB, C, D. Δηλαδή θα φτιαχτούν τρεις στοίβες, όπου στην πρώτη το block **A** βρίσκεται πάνω στο τραπέζι και το **B** από πάνω του, στη δεύτερη το **C** και στην τρίτη το **D**, τα οποία είναι πάνω στο τραπέζι.

Σαν τελική κατάσταση ορίζουμε την **AC , , DB**. Πιο αναλυτικά, στην πρώτη στήλη βρίσκεται το block **A** και από πάνω του το **C**, η δεύτερη στοίβα είναι κενή και στην τρίτη βρίσκεται ακουμπισμένο στο τραπέζι το block **D** και από πάνω το **B**.

```
Επιλέχθηκε το πρόβλημα Blocks World
Αρχική κατάσταση: ab,c,d
Αρχική κατάσταση: [['a', 'b'], ['c'], ['d']]
Τελική κατάσταση: ac,,db
Τελική κατάσταση: [['a', 'c'], [], ['d', 'b']]

==============================
Βρέθηκε λύση
==============================
Κινήσεις που χρειάστηκαν: 2
==============================
[['a', 'b'], ['c'], ['d']]
[['a'], ['c'], ['d', 'b']]
[['a', 'c'], [], ['d', 'b']]
==============================
```

Εικόνα 2: Παράδειγμα εκτέλεσης Blocks World

Όπως παρατηρούμε, εμφανίζεται ο αριθμός των κινήσεων που χρειάστηκαν, αλλά και η αλληλουχία των βημάτων που εκτέλεσε ο *Generic Planner* προκειμένου να βρεθεί η βέλτιστη λύση στο πρόβλημα.

Ένα άλλο παράδειγμα εκτέλεσης είναι το παρακάτω:

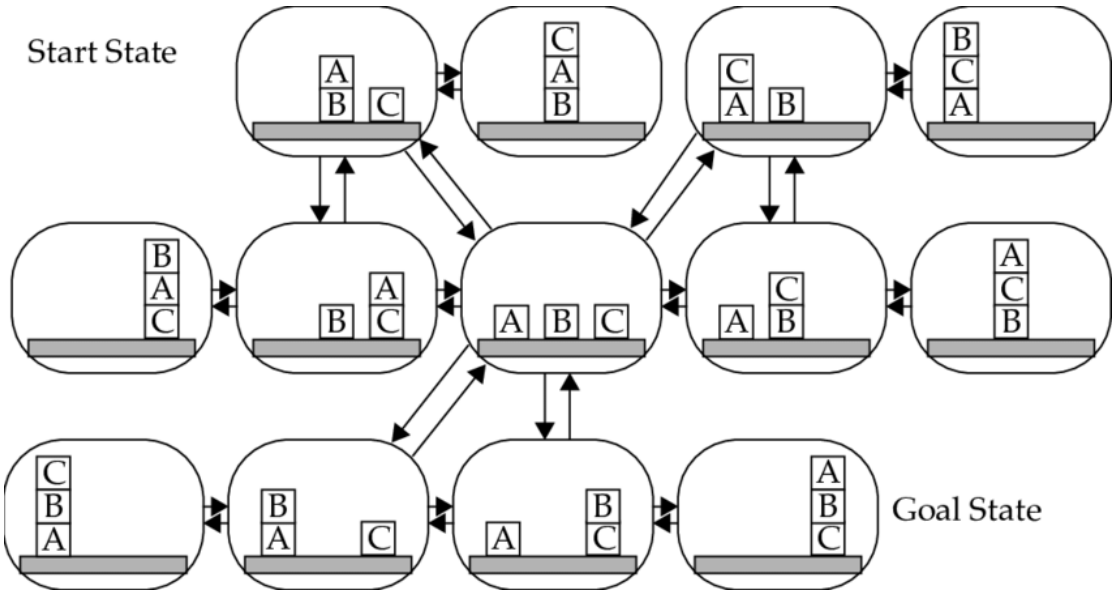
- Η αρχική κατάσταση είναι η **,AB,C**, όπου η πρώτη στοίβα είναι κενή
- Η τελική κατάσταση είναι η **,,CBA**, όπου οι πρώτες δύο στοίβες δεν έχουν κανένα block.

```
Επιλέχθηκε το πρόβλημα Blocks World
Αρχική κατάσταση: ,ab,c
Αρχική κατάσταση: [[], ['a', 'b'], ['c']]
Τελική κατάσταση: ,,cba
Τελική κατάσταση: [[], [], ['c', 'b', 'a']]

==============================
Βρέθηκε λύση
==============================
Κινήσεις που χρειάστηκαν: 2
==============================
[[], ['a', 'b'], ['c']]
[[], ['a'], ['c', 'b']]
[[], [], ['c', 'b', 'a']]
==============================
```

Εικόνα 3: Παράδειγμα εκτέλεσης Blocks World

Μετά και την ολοκλήρωση της αλγοριθμικής διαδικασίας, παρατηρούμε τον αριθμό και την αλληλουχία των κινήσεων προκειμένου να βρεθεί η επιθυμητή τελική κατάσταση με το ελάχιστο δυνατό κόστος.

Οι πιθανές καταστάσεις αλλά και η διαδρομή μέχρι την τελική κατάσταση οπτικοποιούνται από το παρακάτω διάγραμμα.



Εικόνα 4: Διάγραμμα καταστάσεων για το δεύτερο παράδειγμα

6.2 Water Jug

Ακολουθούν παραδείγματα για το *Water Jug.*

Στο πρώτο παράδειγμα ο χρήστης εισάγει την χωρητικότητα των δύο κανατών ως εξής:

- Για την κανάτα 1, η χωρητικότητα ορίζεται στα 4 λίτρα.
- Για την κανάτα 2, η χωρητικότητα ορίζεται στα 3 λίτρα.

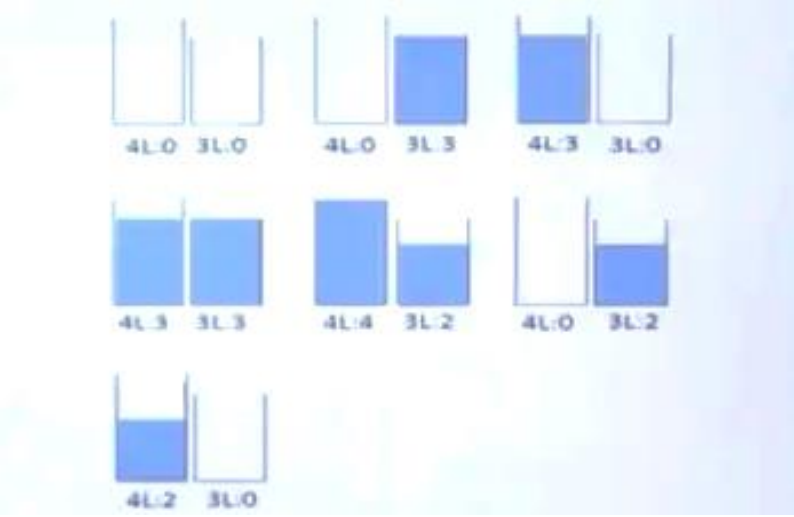
Σαν τελική κατάσταση ορίζουμε την **2,0**. Πιο αναλυτικά, η κανάτα 1 θα έχει δύο λίτρα νερού, ενώ η δεύτερη κανένα.

```
Επιλέχθηκε το πρόβλημα Water Jug
Ποσότητα στην κανάτα 1: 4
Ποσότητα στην κανάτα 2: 3
Τελική ποσότητα στην κανάτα 1: 2
Τελική ποσότητα στην κανάτα 2: 0

==============================
Βρέθηκε λύση
==============================
Κινήσεις που χρειάστηκαν: 6
==============================
['Γέμισε την κανάτα 2', 'Άδειασε όλη την κανάτα 2 στην
κανάτα 1', 'Γέμισε την κανάτα 2', 'Γέμισε την κανάτα 1
από τη κανάτα 2', 'Άδειασε την κανάτα 1', 'Άδειασε όλη
την κανάτα 2 στην κανάτα 1']
==============================
```

Εικόνα 5: Παράδειγμα εκτέλεσης Water Jug

Όπως παρατηρούμε και στην παραπάνω εικόνα, εμφανίζεται τόσο ο αριθμός των κινήσεων που χρειάστηκαν, όσο και η αλληλουχία των βημάτων που εκτέλεσε ο *Generic Planner* προκειμένου να βρεθεί η βέλτιστη λύση στο πρόβλημα.

Οι πιθανές καταστάσεις αλλά και η διαδρομή μέχρι την τελική κατάσταση μπορούν να οπτικοποιηθούν από το παρακάτω διάγραμμα.



Εικόνα 6: Διάγραμμα καταστάσεων για το παράδειγμα 1

Ένα άλλο παράδειγμα εκτέλεσης για το εν λόγω πρόβλημα είναι το παρακάτω:

- Για την κανάτα 1, η χωρητικότητα ορίζεται στα 5 λίτρα.
- Για την κανάτα 2, η χωρητικότητα ορίζεται στα 3 λίτρα.

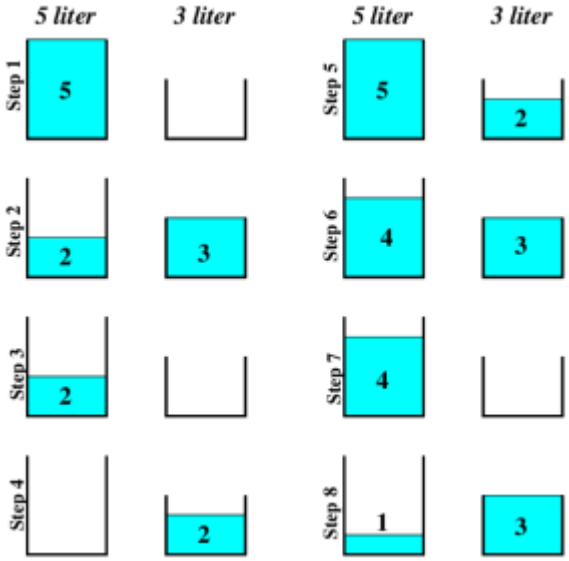
Ως τελική κατάσταση ορίζουμε την **1,3**. Πιο αναλυτικά, η κανάτα 1 θα έχει 1 λίτρο νερού, ενώ η δεύτερη 3.

```
Επιλέχθηκε το πρόβλημα Water Jug
Ποσότητα στην κανάτα 1: 5
Ποσότητα στην κανάτα 2: 3
Τελική ποσότητα στην κανάτα 1: 1
Τελική ποσότητα στην κανάτα 2: 3

==============================
Βρέθηκε λύση
==============================
Κινήσεις που χρειάστηκαν: 8
==============================
['Γέμισε την κανάτα 1', 'Γέμισε την κανάτα 2 από τη κανάτα 1',
'Άδειασε την κανάτα 2', 'Άδειασε όλη την κανάτα 1 στην κανάτα 2
', 'Γέμισε την κανάτα 1', 'Γέμισε την κανάτα 2 από τη κανάτα 1'
, 'Άδειασε την κανάτα 2', 'Γέμισε την κανάτα 2 από τη κανάτα 1'
]
==============================
```

Εικόνα 7: Παράδειγμα εκτέλεσης Water Jug

Μετά και την ολοκλήρωση της αλγοριθμικής διαδικασίας, παρατηρούμε τον αριθμό αλλά και την αλληλουχία των κινήσεων προκειμένου να βρεθεί η επιθυμητή τελική κατάσταση με το ελάχιστο δυνατό κόστος.

Στο παρακάτω διάγραμμα φαίνονται οι πιθανές καταστάσεις αλλά και η διαδρομή μέχρι την τελική κατάσταση.



Εικόνα 8: Διάγραμμα καταστάσεων για το παράδειγμα 2

7. Μελλοντικές Επεκτάσεις – Χειρισμός Εξαιρέσεων

Στο συγκεκριμένο σημείο θα παρουσιαστούν οι τυχόν μελλοντικές επεκτάσεις της εφαρμογής, αλλά και ο χειρισμός των εξαιρέσεων σε περίπτωση που κάποιο από τα προβλήματα δεν μπορεί να επιλυθεί.

Ως προς το πρώτο σημείο, θα πρέπει να επισημανθεί ότι ο κατάλληλος ορισμός ενός νέου προβλήματος θα μπορούσε να επιλυθεί από τον συγκεκριμένο *Generic Planner*.

Όσο αναφορά το δεύτερο σημείο, έχει γίνει πρόβλεψη ώστε στην περίπτωση που ο χρήστης δεν επιλέξει μια από τις διαθέσιμες επιλογές του Menu, να του εμφανίζεται η αντίστοιχη εικόνα που θα τον ενημερώνει.

```
==============================
=============MENU=============
==============================
1. Blocks World
2. Water Jug
3. Έξοδος
==============================
Εισάγετε την επιλογή σας [1-3]: 4
Εισάγετε ένα αριθμό από το 1 έως το 3
```

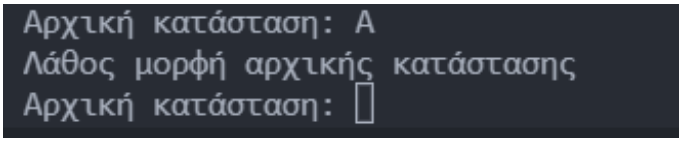
Εικόνα 9: Λάθος επιλογή στο Menu

Επιπρόσθετα, για το πρόβλημα του *Blocks World* έχει μεριμνήθει να γίνεται έλεγχος στην περίπτωση που ο χρήστης εισαγάγει λάθος την αρχική κατάσταση ή την τελική κατάσταση. Για την επίτευξη του συγκεκριμένου σκοπού συγκρίνεται η είσοδος με μια *regular expression* της μορφής:

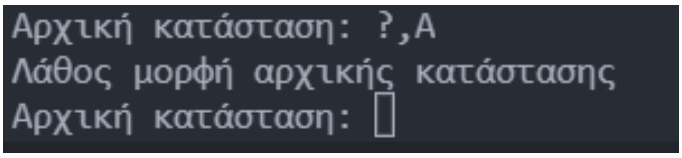
[A-Z,a-z,0-9]*[,]([A-Z,a-z,0-9]*[,]*)

```
Επιλέχθηκε το πρόβλημα Blocks World
Αρχική κατάσταση: 1.25
Λάθος μορφή αρχικής κατάστασης
Αρχική κατάσταση:
```

Εικόνα 10: Λάθος εισαγωγή της αρχικής κατάστασης

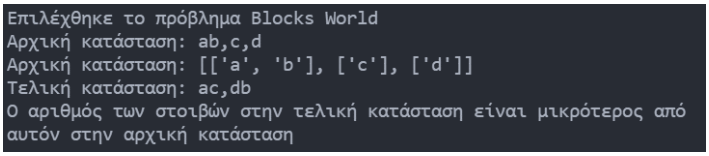


Εικόνα 11: Λάθος εισαγωγή της αρχικής κατάστασης



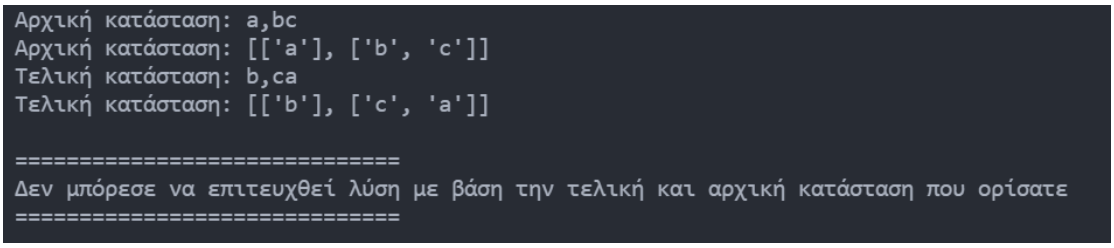
Εικόνα 12: Λάθος εισαγωγή της αρχικής κατάστασης

Ένας άλλος έλεγχος που γίνεται κατά την εισαγωγή της τελικής κατάστασης, είναι αν ο αριθμός των στοιβών σε αυτήν είναι μικρότερος από τον αριθμό των στοιβών της αρχικής κατάστασης.



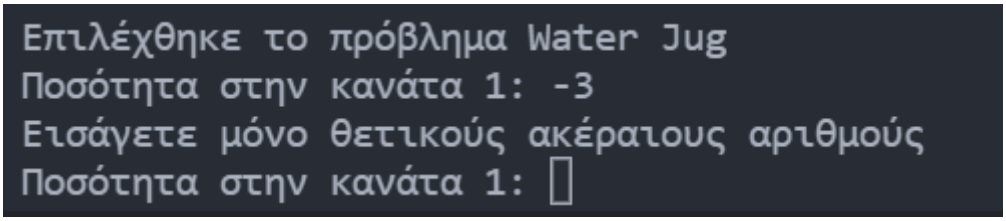
Εικόνα 13: Λάθος εισαγωγή της τελικής κατάστασης

Αξιοσημείωτο είναι, ότι εάν δεν μπορεί να ευρεθεί η λύση του προβλήματος λόγω της αρχικής και τελικής κατάστασης, εμφανίζεται στην οθόνη το αντίστοιχο μήνυμα.

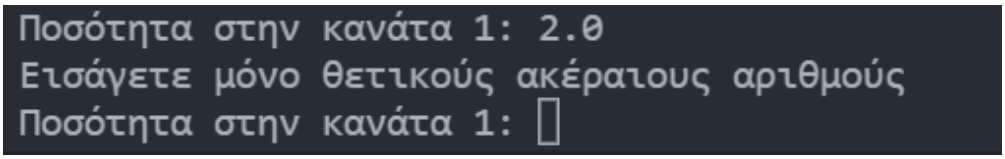


Εικόνα 14: Μη εύρεση της βέλτιστης λύσης

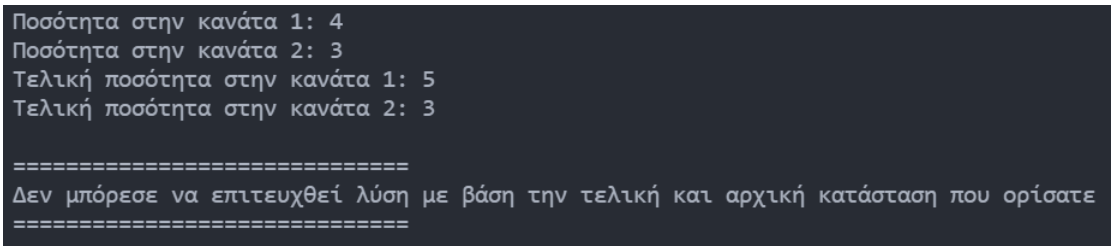
Αντίστοιχα, για το πρόβλημα του *Water Jug*, κατά την εισαγωγή της χωρητικότητας της κανάτας 1 ή 2, αλλά και για την τελική κατάσταση της κάθε μιας, ελέγχεται αν ο χρήστης εισάγει αρνητικές τιμές. Ακόμα, έχει γίνει πρόβλεψη ώστε κατά τον καθορισμό της τελικής κατάστασης η ποσότητα του νερού μιας κανάτας να μην είναι μεγαλύτερη από την χωρητικότητα της.



Εικόνα 15: Εισαγωγή αρνητικής ποσότητας νερού

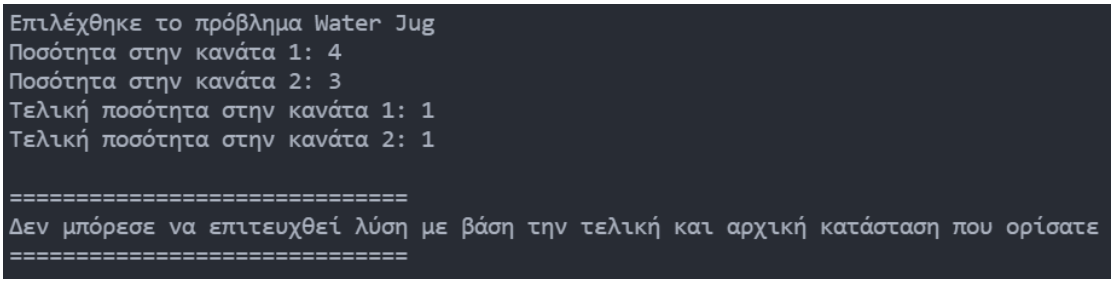


Εικόνα 16: Εισαγωγή λανθασμένης μορφής χωρητικότητας



Εικόνα 17: Ορισμός λανθασμένης τελικής κατάστασης

Τέλος, στην περίπτωση που δεν μπορεί να ευρεθεί η λύση του προβλήματος λόγω της αρχικής και τελικής κατάστασης, εμφανίζεται στην οθόνη του χρήστη το αντίστοιχο μήνυμα.



Εικόνα 18: Μη εύρεση της βέλτιστης λύσης

8. Απαιτήσεις συστήματος και εκτέλεση

Η παραπάνω άσκηση υλοποιήθηκε στην γλώσσα προγραμματισμού ***Python, v3.7.9***.

Οι συναρτήσεις που υλοποιήθηκαν βρίσκονται στο αρχείο **functions.py**. Το αρχείο που καλεί τις συγκεκριμένες συναρτήσεις και είναι υπεύθυνο για την εκτέλεση του προγράμματος είναι το αρχείο **main.py**.

Για να εκτελέσουμε τον κώδικα ακολουθούμε τα παρακάτω βήματα:

1. Ανοίγουμε τη γραμμή εντολών και μεταβαίνουμε στον αντίστοιχο φάκελο**: \Ευφυείς Πράκτορες**. (***Εικόνα 5***)
2. Πληκτρολογούμε το όνομα του αρχείου **main.py**, ώστε να ανοίξει το εκτελέσιμο. (***Εικόνα 6***)

Όταν ανοίξει το πρόγραμμα εμφανίζεται το menu του προγράμματος, όπου ο χρήστης πρέπει να επιλέξει ποιο ερώτημα επιθυμεί να εκτελέσει.(***Εικόνα***).

Σε περίπτωση που επιθυμεί ο χρήστης να επιλύσει το πρώτο πρόβλημα, του *Blocks World*, τότε πληκτρολογεί το: **1**, ενώ πληκτρολογεί το: **2,** εάν επιθυμεί την επίλυση του προβλήματος *Water Jug*. Το **3** πληκτρολογείται για την έξοδο από το πρόγραμμα.

Στην περίπτωση που επιλεχθεί το **1**, ο χρήστης πρέπει να ακολουθήσει την εξής αλληλουχία βημάτων:

1. Εισάγει την αρχική κατάσταση, από την οποία θα ξεκινήσει η επίλυση του προβλήματος. Η αρχική κατάσταση θα πρέπει να έχει την μορφή ***x,yz***, όπου ***x*** είναι ο κύβος που βρίσκεται στην πρώτη στήλη και είναι πάνω στο τραπέζι, το «***,***» ορίζει μια νέα στοίβα από κύβους, ενώ το ***y*** βρίσκεται πάνω στο τραπέζι και το ***z*** πάνω από το ***y***. Για να ορίσουμε μια κενή στοίβα απλά αφήνουμε κενό, δηλαδή είτε γράφουμε «***,,***», είτε «***, ,***», είτε «***x,***», είτε «***,x***».
2. Για τον ορισμό της τελικής κατάστασης ακολουθούμε τους κανόνες του παραπάνω βήματος.
3. Αφού έχουν ολοκληρωθεί τα παραπάνω βήματα, εκκινείται η αλγοριθμική διαδικασία για την επίλυση του προβλήματος.

Εάν ο χρήστης επιλέξει το **2**, τότε θα πρέπει να προβεί στις παρακάτω ενέργειες:

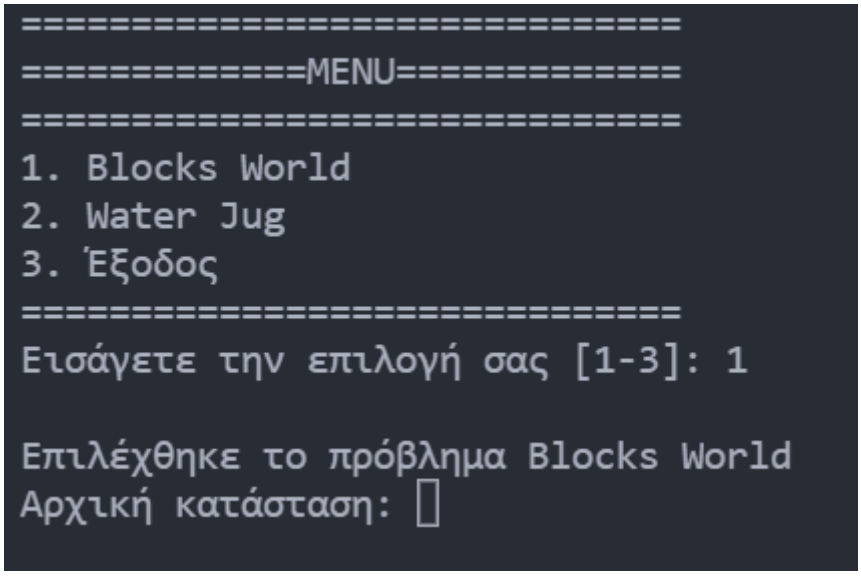
1. Εισάγει την χωρητικότητα της πρώτης κανάτας σε λίτρα.
2. Εισάγει την χωρητικότητα της δεύτερης κανάτα σε λίτρα.
3. Καθορίζει την τελική κατάσταση της κανάτας 1.
4. Καθορίζει την τελική κατάσταση της κανάτας 2.
5. Αφού έχει εισάγει όλα τα παραπάνω στοιχεία εκτελείται ο *Generic Planner* για την επίλυση του προβλήματος, *Water Jug.*

```
Microsoft Windows [Version 10.0.19042.1083]
(c) Microsoft Corporation. Με επιφύλαξη κάθε νόμιμου δικαιώματος.

C:\Users\panap\OneDrive\Υπολογιστής\Ευφυείς Πράκτορες>python main.py
```

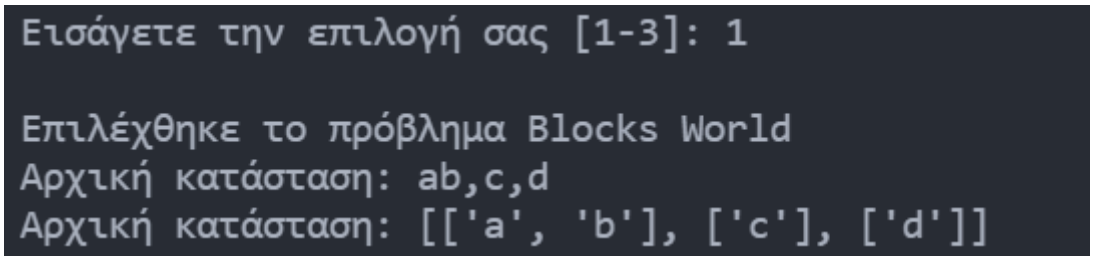
Εικόνα 19: Εισαγωγή του ονόματος αρχείου προς εκτέλεση

```
Microsoft Windows [Version 10.0.19042.1083]
(c) Microsoft Corporation. Με επιφύλαξη κάθε νόμιμου δικαιώματος.

C:\Users\panap\OneDrive\Υπολογιστής\Ευφυείς Πράκτορες>python main.py

==============================
=============MENU=============
==============================
1. Blocks World
2. Water Jug
3. Έξοδος
==============================
Εισάγετε την επιλογή σας [1-3]:
```

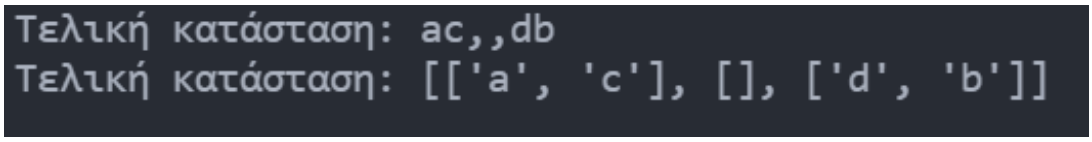
Εικόνα 20: Εμφάνιση Menu



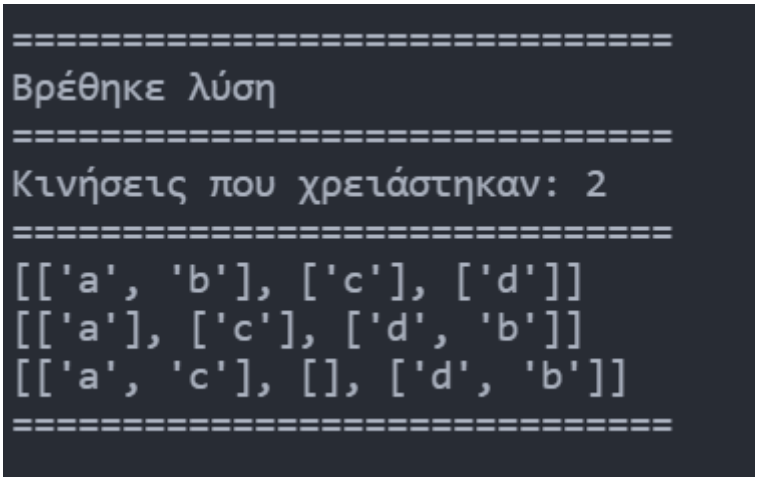
Εικόνα 21: Επιλογή προβλήματος Blocks World



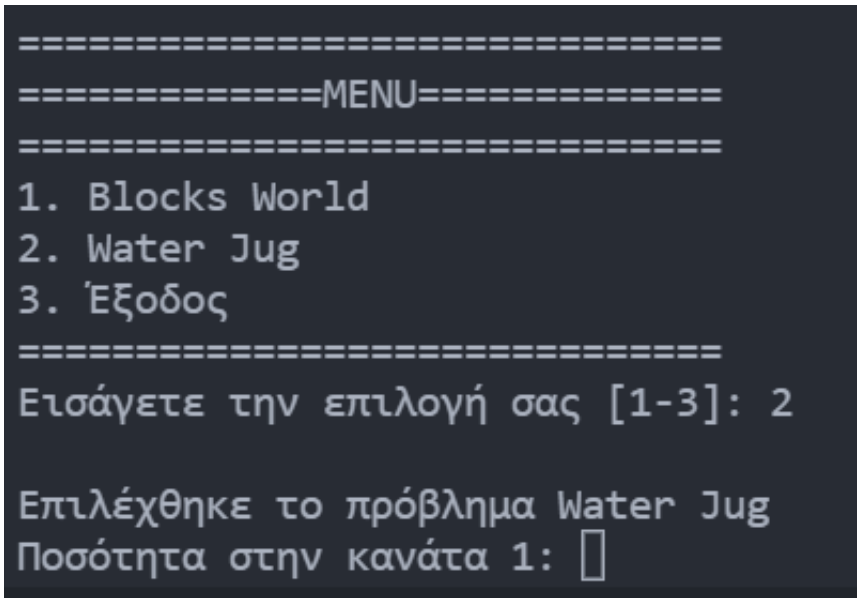
Εικόνα 22: Ορισμός αρχικής κατάστασης



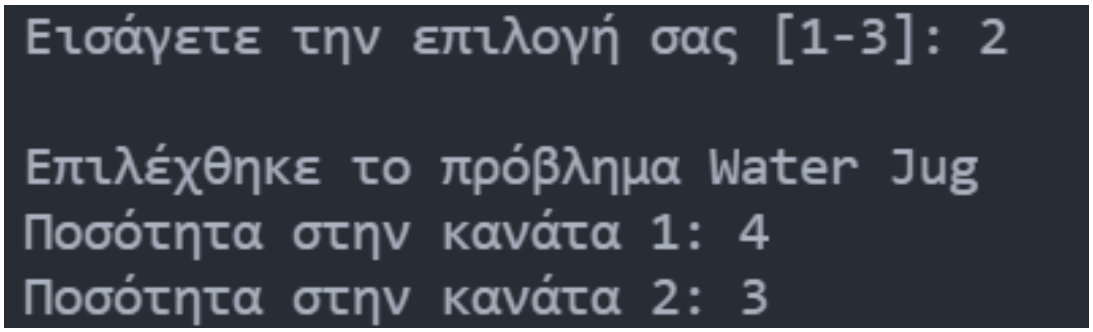
Εικόνα 23: Ορισμός τελικής κατάστασης



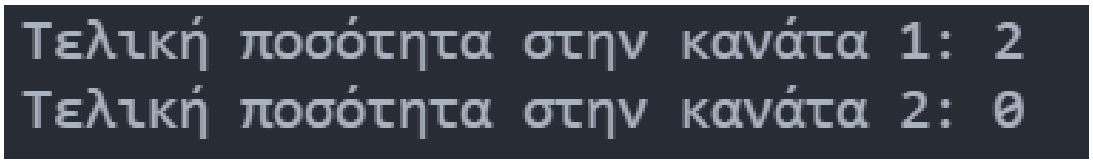
Εικόνα 24: Εύρεση βέλτιστης λύσης



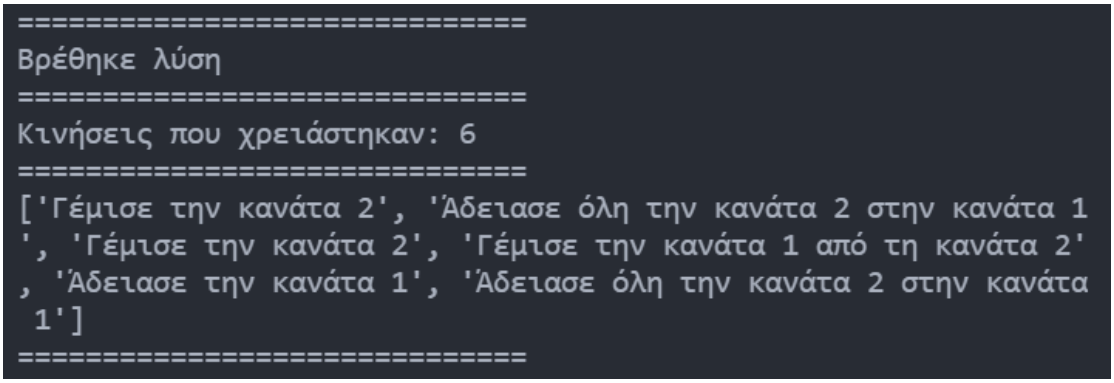
Εικόνα 25: Επιλογή προβλήματος Water Jug



Εικόνα 26: Εισαγωγή της χωρητικότητας των δύο κανατών



Εικόνα 27: Εισαγωγή της τελικής κατάστασης



Εικόνα 28: Εύρεση βέλτιστης λύσης

9. Βιβλιογραφία

Σε αυτή την ενότητα θα αναφερθούν οι βιβλιογραφικές πηγές της εφαρμογής μας:

1. Visual Studio Code Documentation
   (τελευταία προσπέλαση 13/07/2021)
2. Python v.3.7.9 Documentation
   (τελευταία προσπέλαση 15/07/2021)
3. Blocks World
   (τελευταία προσπέλαση 11/07/2021)
4. Water Pouring Puzzle
   (τελευταία προσπέλαση 11/07/2021)
5. Why Heuristics Work – G. Gigerenzer, Michigan State University
   (τελευταία προσπέλαση 12/07/2021)
6. A* search algorithm
   (τελευταία προσπέλαση 12/07/2021)
7. A* Comparison – Stamford University
   (τελευταία προσπέλαση 12/07/2021)
8. Path Planning Using an Improved A-star Algorithm - Chunyu Ju, Qinghua Luo, Xiaozhen Yan
   (τελευταία προσπέλαση 13/07/2021)
9. A Logic Programming Solution to the Water Jug Puzzle
   (τελευταία προσπέλαση 14/07/2021)
10. A Planning System for Blocks-World Domain
    (τελευταία προσπέλαση 14/07/2021)