



ΠΑΝΕΠΙΣΤΗΜΙΟ ΠΕΙΡΑΙΩΣ  
ΣΧΟΛΗ ΤΕΧΝΟΛΟΓΙΩΝ, ΠΛΗΡΟΦΟΡΙΚΗΣ  
ΚΑΙ ΕΠΙΚΟΙΝΩΝΙΩΝ  
Τμήμα Πληροφορικής

ΤΕΛΙΚΗ ΕΡΓΑΣΙΑ ΜΑΘΗΜΑΤΟΣ  
ΕΥΦΥΕΙΣ ΠΡΑΚΤΟΡΕΣ  
Εξάμηνο: 8<sup>ο</sup>

**Ομάδα υλοποίησης Εργασίας:**

- Καλογήρου Στυλιανή, Α.Μ. Π18181, [stelakalogirou@gmail.com](mailto:stelakalogirou@gmail.com)
- Καπερναράκος Δημοσθένης, Α.Μ. Π18058, [dimostheniskap13@gmail.com](mailto:dimostheniskap13@gmail.com)

ΑΥΓΟΥΣΤΟΣ 2022

## ΠΕΡΙΕΧΟΜΕΝΑ

Εκφώνηση Εργασίας.....	3
Περιγραφή Εργασίας.....	5
Περιγραφή προβλήματος Block world:.....	5
Περιγραφή προβλήματος Water jug (water jug problem):.....	5
Περιγραφή θεωρητικής βάσης εφαρμογής.....	6
Περιγραφή σημαντικών σχεδιαστικών αποφάσεων και στοιχείων υλοποίησης.....	7
Επεξήγηση Λειτουργίας.....	9
1.Water Pouring Puzzle.....	9
2.Block world .....	11
Παραδείγματα από στιγμιότυπα οθόνης .....	12
Περιγραφή διαδικασίας εγκατάστασης .....	13
Επεκτάσεις/Εργαλεία .....	13
Python 3.10.....	13
PyCharm 2021.3 .....	13
Βιβλιογραφία .....	13

## Εκφώνηση Εργασίας

### 3 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 σενεργειών 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

[https://en.wikipedia.org/wiki/Blocks\\_world](https://en.wikipedia.org/wiki/Blocks_world)

[https://en.wikipedia.org/wiki/Water\\_pouring\\_puzzle](https://en.wikipedia.org/wiki/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.**

### **Παραδοτέα**

1. Η εφαρμογή σε εκτελέσιμη μορφή (π.χ. αν είναι σε Unity πρέπει να έχει project και να έχει γίνει build)
2. Πηγαίος κώδικας για το σύνολο της εφαρμογής
3. Τεκμηρίωση του planner και των αναπαραστάσεων των καταστάσεων και των ενεργειών για τα 2 προβλήματα
4. Τα αποτελέσματα του planner για τα 2 προβλήματα
5. video + powerpoint

### **ΣΗΜΕΙΩΣΗ :**

**Η επιλογή γλωσσών και πλατφόρμας υλοποίησης είναι ελεύθερη.**

## Περιγραφή Εργασίας

### **Ανάπτυξη generic planner**

Η κατασκευή ενός 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.

### **Περιγραφή προβλήματος Block world:**

Ο κόσμος των μπλοκ είναι ένας από τους πιο διάσημους τομείς σχεδιασμού στην τεχνητή νοημοσύνη. Ο αλγόριθμος είναι παρόμοιος με ένα σύνολο από ξύλινα μπλοκ διαφόρων σχημάτων και χρωμάτων που κάθετα σε ένα τραπέζι. Ο στόχος είναι να οικοδομήσουμε μία ή περισσότερες κάθετες στοίβες μπλοκ. Μόνο ένα μπλοκ μπορεί να μετακινηθεί κάθε φορά: μπορεί είτε να τοποθετηθεί στο τραπέζι είτε να τοποθετηθεί πάνω σε ένα άλλο μπλοκ. Εξαιτίας αυτού, τυχόν μπλοκ που βρίσκονται, σε μια δεδομένη στιγμή, κάτω από ένα άλλο μπλοκ δεν μπορούν να μετακινηθούν. Επιπλέον, ορισμένα είδη μπλοκ δεν μπορούν να στοιβάζουν άλλα μπλοκ πάνω τους.

Η απλότητα αυτού του κόσμου παιχνιδιών προσφέρεται εύκολα σε κλασικές προσεγγίσεις τεχνητής νοημοσύνης, στις οποίες ο κόσμος διαμορφώνεται ως σύνολο αφηρημένων συμβόλων που μπορεί να είναι λογικά.

### **Περιγραφή προβλήματος Water jug (water jug problem):**

Ο γρίφος water jug (ονομάζονται επίσης προβλήματα με την κανάτα νερού, τα προβλήματα απόφραξης, τα παζλ μέτρησης) είναι μια κατηγορία παζλ που περιλαμβάνει μια πεπερασμένη συλλογή κανατών νερού γνωστών ακέραιων δυνατοτήτων (σε όρους υγρού μέτρου όπως λίτρα ή γαλόνια). Αρχικά κάθε κανάτα περιέχει έναν γνωστό ακέραιο όγκο υγρού, όχι απαραίτητα ίσο με την χωρητικότητά του. Οι γρίφοι αυτού του τύπου ρωτούν πόσα βήματα ρίχνουν νερό από τη μία κανάτα στην άλλη (έως ότου η μία κανάτα αδειάσει ή η άλλη γεμίσει) για να φτάσει σε μια κατάσταση στόχου, που καθορίζεται σε όρους όγκου υγρού που πρέπει να υπάρχει κάποια κανάτα ή κανάτες.

### Περιγραφή θεωρητικής βάσης εφαρμογής

Ένας σχεδιαστής(planner) δημιουργεί μια ακολουθία ενεργειών έτσι ώστε όταν εκτελούνται οι ενέργειες, ο κόσμος του πράκτορα βρίσκεται από την αρχική έως την τελική κατάσταση. Ο σχεδιαστής θεωρείται γενικός όταν δεν αλλάζουμε τίποτα στον κώδικά του για να λύσουμε διαφορετικά προβλήματα.

Για να δημιουργήσει την ακολουθία ενεργειών, ο σχεδιαστής χρησιμοποιεί έναν αλγόριθμο αναζήτησης για να κατασκευάσει ένα δέντρο αναζήτησης.

Η θεωρητική βάση της εφαρμογής έχει ως στόχο την επίλυση 2 διαφορετικών προβλημάτων το Blocks World και το Water jug. Για την επίλυση των δύο αυτών προβλημάτων χρησιμοποιήσαμε τον αλγόριθμο (A\* algorithm). Ο A\* (προφέρεται "A-star") είναι ένας αλγόριθμος διασταύρωσης γραφήματος και αναζήτησης διαδρομής, ο οποίος χρησιμοποιείται συχνά σε πολλούς τομείς της επιστήμης των υπολογιστών λόγω της πληρότητάς του, της βελτιστοποίησης και της βέλτιστης απόδοσής του.

Όπως ο αλγόριθμος Dijkstra, ο A\* λειτουργεί δημιουργώντας ένα δέντρο διαδρομής χαμηλότερου κόστους από τον κόμβο έναρξης έως τον κόμβο προορισμού. Αυτό που κάνει ο A\* διαφορετικό και καλύτερο για πολλές αναζητήσεις είναι ότι για κάθε κόμβο, το A\* χρησιμοποιεί μια συνάρτηση  $f(n)$  που δίνει μια εκτίμηση του συνολικού κόστους μιας διαδρομής που χρησιμοποιεί αυτόν τον κόμβο.

Επίσης για την επίλυση του προβλήματος Block World χρησιμοποιήθηκε και ο αλγόριθμος Best First Search Algorithm in AI. Ουσιαστικά η χρήση του είναι εξής:

Δημιουργεί 2 κενές λίστες: OPEN και CLOSE

Ξεκινάει από τον αρχικό κόμβο (ας πούμε N) και βάζει τον στη λίστα "OPEN". Επαναλαμβάνει τα επόμενα βήματα έως ότου επιτευχθεί ο κόμβος GOAL. Εάν η λίστα OPEN είναι κενή, τότε EXIT ο βρόχος επιστρέφει "False". Επιλέγει τον πρώτο / κορυφαίο κόμβο (π. N) στη λίστα ΑΝΟΙΧΤΟ και τον μετακινεί στη λίστα ΚΛΕΙΣΤΟ. Καταγράφει επίσης τις πληροφορίες του γονικού κόμβου. Αν το N είναι κόμβος GOAL, μετακινεί τον κόμβο στη λίστα κλειστών και βγαίνει από τον βρόχο επιστρέφοντας «True». Η λύση μπορεί να βρεθεί ακολουθώντας το μονοπάτι. Εάν

το N δεν είναι ο κόμβος GOAL, αναπτύσσει τον κόμβο N για να δημιουργεί τους "άμεσους" επόμενους κόμβους που συνδέονται με τον κόμβο N και προσθέτει όλους αυτούς στη λίστα OPEN. Αναδιάταξη των κόμβων στη λίστα OPEN σε αύξουσα σειρά σύμφωνα με τη συνάρτηση αξιολόγησης  $f(n)$

Αυτός ο αλγόριθμος θα διασχίσει τη συντομότερη διαδρομή πρώτα στην ουρά. Η χρονική πολυπλοκότητα του αλγορίθμου δίνεται από το  $O(n * \log n)$ .

Η σχεδίαση έγινε με γνώμονα τη λύση του προβλήματος σε γλώσσα python αφού αρχικά έγινε μια αναλυτική μελέτη στο διαδίκτυο για τα προβλήματα Water jug και Block World. Από εκεί έγινε κατανοητό ότι η επίλυση των προβλημάτων μπορεί να γίνει με τη χρήση και του αλγορίθμου Best First Search Algorithm όπως και του A\* algorithm. Επίσης η γλώσσα προγραμματισμού python βοηθάει περισσότερο λόγω των πολλών βιβλιοθηκών που μπορούν να γίνουν import.

### Περιγραφή σημαντικών σχεδιαστικών αποφάσεων και στοιχείων υλοποίησης

Αρχικά με την εκκίνηση του προγράμματος μας, κάνουμε run την main. Από την Main καλούμε από το αρχείο menu.py την συνάρτηση showMenu() που είναι το μενού του προγράμματος μας. Ο χρήστης αν επιλέξει 1 είναι για το jugs problem και αν επιλέξει 2 είναι για το blockworld και Q για έξοδο από το πρόγραμμα. Με την επιλογή 1 ο χρήστης βάζει την χωρητικότητα των κανατών και τον τελικό προορισμό και ελέγχουμε να μην είναι μηδενικά. Στην συνέχεια, στέλνουμε τα δεδομένα αυτά στην συνάρτηση bfs() του αρχείου SearchAlgorithms.py και τυπώνουμε το αποτέλεσμα. Στην συνάρτηση bsf() υλοποιείται ο αλγόριθμος πρώτα κατά πλάτος (Breadth First Search).

Αναζήτηση κατά πλάτος είναι ένας [αλγόριθμος](#) για διάσχιση ή αναζήτηση σε δομές δεδομένων τύπου δέντρου ή γράφου. Η αναζήτηση ξεκινά από τη ρίζα του δέντρου (ή από κάποιο αυθαίρετο κόμβο του γραφήματος, που μερικές φορές αναφέρεται ως "κλειδί αναζήτησης του") και διερευνά πρώτα τους γειτονικούς κόμβους, προτού μεταβεί στους γείτονες του επόμενου επίπεδο.

Η αναζήτηση κατά πλάτος επιτυγχάνεται με χρήση μιας βοηθητική ουράς. Οι κόμβοι των κάθε επιπέδων του δέντρου εισάγονται διαδοχικά στη βοηθητική ουρά. Ο κάτωθι επαναληπτικός αλγόριθμος διακρίνει δύο περιπτώσεις:

- Περίπτωση ρίζας: Η ουρά είναι αρχικά κενή, συνεπώς ο κόμβος της ρίζας προστίθεται στη βοηθητική ουρά.
- Γενική περίπτωση: Όσο η βοηθητική ουρά δεν είναι κενή, εξαγωγή και επεξεργασία του επόμενου κόμβου από την ουρά. Στη συνέχεια, εισαγωγή στη βοηθητική ουρά όλων των κόμβων παιδιών του τρέχοντος κόμβου. Εάν ο τρέχων κόμβος είναι φύλλο, δεν εισάγεται τίποτα.

Με την επιλογή 2 ο χρήστης βάζει τον αριθμό των μπλοκ. Μετά γίνεται δημιουργία τυχαίων καταστάσεων για την αρχική και τελική κατάσταση, ελέγχοντας αν δεν είναι ίδιες. Μετά καλούμε την συνάρτηση `BlocksWorldH2`, που σε αυτή την συνάρτηση βρίσκουμε τον αριθμό κινήσεων που πρέπει να γίνουν για να φτάσει το κάθε μπλοκ στην σωστή θέση. Στην συνέχεια, καλούμε την συνάρτηση `a_star()` από το αρχείο `SearchAlgorithms.py` που μέσω αυτής εκτελείται ο αλγόριθμος  $A^*$ . Ο αλγόριθμος  $A^*$  βρίσκει μόνο τη συντομότερη διαδρομή από μια καθορισμένη πηγή σε έναν καθορισμένο στόχο και όχι το δέντρο της συντομότερης διαδρομής από μια καθορισμένη πηγή προς όλους τους πιθανούς στόχους. Αυτός είναι ένας απαραίτητος συμβιβασμός για τη χρήση μιας ευρετικής κατευθυνόμενης προς συγκεκριμένο στόχο.

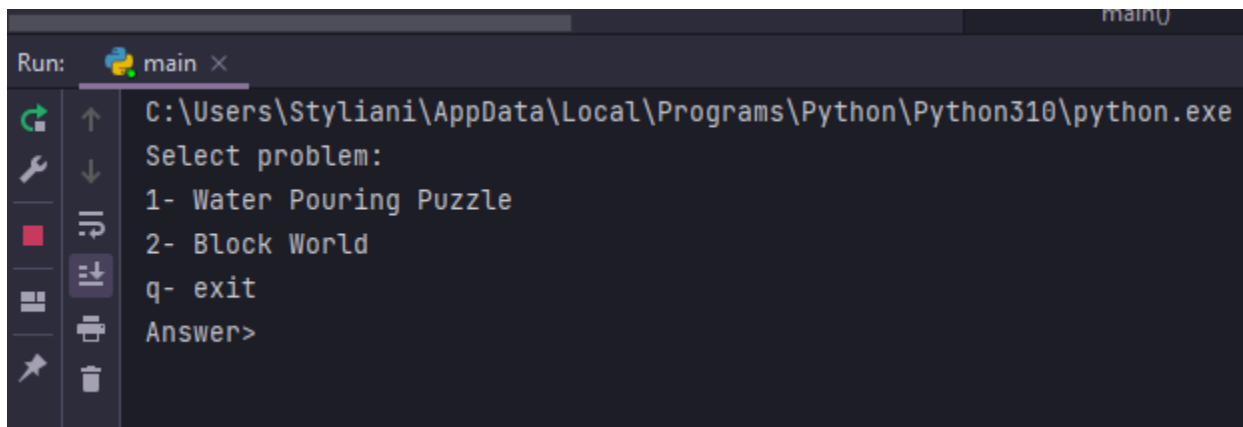
Στην συνέχεια τυπώνουμε τις καταστάσεις που προέκυψαν μέχρι τον τελικό μας στόχο.



## Επεξήγηση Λειτουργίας

Αρχικά επιλέγουμε πιο πρόβλημα θα τρέξουμε

1. Water Pouring Puzzle (water jug problem)
2. Block world

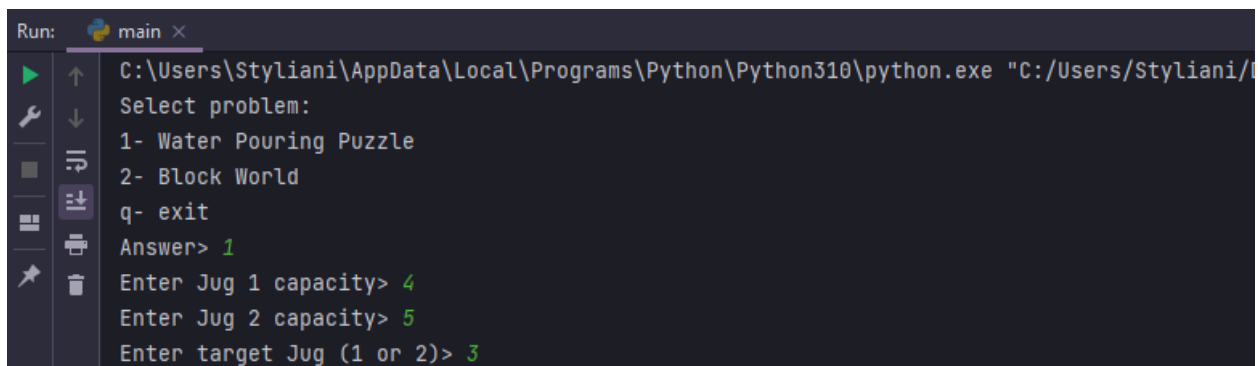


```
Run: main x
C:\Users\Styliani\AppData\Local\Programs\Python\Python310\python.exe
Select problem:
1- Water Pouring Puzzle
2- Block World
q- exit
Answer>
```

### 1. Water Pouring Puzzle

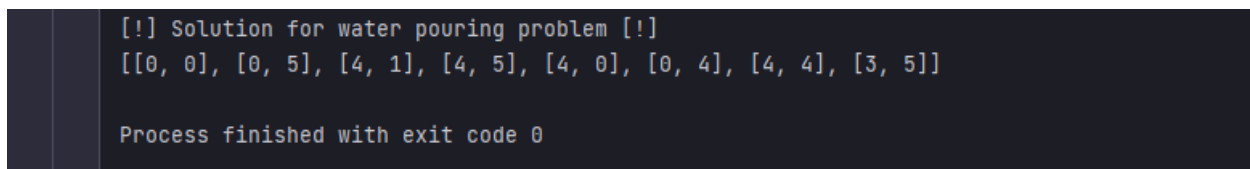
Έστω πως ο χρήστης του προγράμματος επιλέγει το 1. Water pouring puzzle.

Στην συνέχεια θα πρέπει να εισάγει την χωρητικότητα για τα Jugs.



```
Run: main x
C:\Users\Styliani\AppData\Local\Programs\Python\Python310\python.exe "C:/Users/Styliani/
Select problem:
1- Water Pouring Puzzle
2- Block World
q- exit
Answer> 1
Enter Jug 1 capacity> 4
Enter Jug 2 capacity> 5
Enter target Jug (1 or 2)> 3
```

Αποτέλεσμα:



```
[!] Solution for water pouring problem [!]
[[0, 0], [0, 5], [4, 1], [4, 5], [4, 0], [0, 4], [4, 4], [3, 5]]

Process finished with exit code 0
```

Για τη λύση του έχουμε ένα path από 6 rules

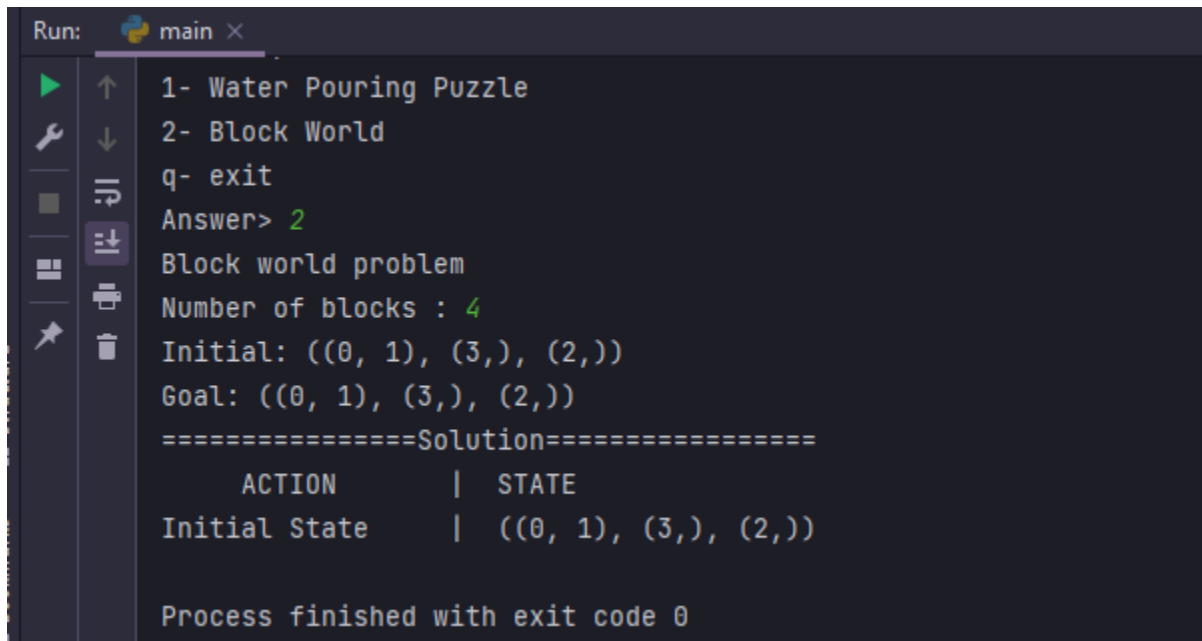
<b># rule 1</b>
if current[0] < x_capacity and ([x_capacity, current[1]] not in visited):
<b># rule 2</b>
if current[1] < y_capacity and ([current[0], y_capacity] not in visited):
<b># rule 3</b>
if current[0] > x_capacity and ([0, current[1]] not in visited):
<b># rule 4</b>
if current[1] > y_capacity and ([x_capacity, 0] not in visited):
<b># rule 5</b>
# (x, y) -> (min(x + y, x_capacity), max(0, x + y - x_capacity)) if y > 0
if current[1] > 0 and ([min(x + y, x_capacity), max(0, x + y - x_capacity)] not in visited):
<b># rule 6</b>
# (x, y) -> (max(0, x + y - y_capacity), min(x + y, y_capacity)) if x > 0
if current[0] > 0 and ([max(0, x + y - y_capacity), min(x + y, y_capacity)] not in visited):

Στη συνέχεια δημιουργείτε μια κατάσταση έναρξης:  $x = 0$ ,  $y = 0$  και μετά η προϋπόθεση για την εξεύρεση λύσης: ο όγκος στόχου «τέλος» θα πρέπει να είναι πολλαπλάσιο του gcd (a, b).

## 2. Block world

Έστω πως ο χρήστης του προγράμματος επιλέγει το 2. Block World.

Στην συνέχεια θα πρέπει να εισάγει τον αριθμό των blocks.



```
Run: main x
1- Water Pouring Puzzle
2- Block World
q- exit
Answer> 2
Block world problem
Number of blocks : 4
Initial: ((0, 1), (3,), (2,))
Goal: ((0, 1), (3,), (2,))
=====Solution=====
      ACTION      |  STATE
Initial State     |  ((0, 1), (3,), (2,))
Process finished with exit code 0
```

Για αρχή γίνεται δημιουργία τυχαίων καταστάσεων για αρχική κατάσταση και κατάσταση στόχου.

```
Initial: ((0, 1), (3,), (2,))
Goal: ((0, 1), (3,), (2,))
```

Η υλοποίηση έγινε με την χρήση των αλγορίθμων A\* και BFS(best first search) και καταλήγουν στην τελική μας κατάσταση στόχου (goal).

## Παραδείγματα από στιγμιότυπα οθόνης

```
Select problem:
1- Water Pouring Puzzle
2- Block World
q- exit
Selection: 1
Enter Jug 1 capacity: 3
Enter Jug 2 capacity: 6
Enter target Jug: 3
[!] Solution for water pouring problem [!]
[[0, 0], [0, 6], [3, 3]]

Process finished with exit code 0
```

```
Select problem:
1- Water Pouring Puzzle
2- Block World
q- exit
Selection: 2
Block world problem
Number of blocks : 10
Initial: ((2, 3), (1, 5), (0, 4, 6), (7, 9), (8,))
Goal: ((2, 3), (1, 5), (0, 4, 6), (7, 9), (8,))
=====Solution=====
  ACTION      | STATE
Initial State | ((2, 3), (1, 5), (0, 4, 6), (7, 9), (8,))

Process finished with exit code 0
```

## Περιγραφή διαδικασίας εγκατάστασης

1. Κατεβάζουμε compiler για python(Στη περίπτωση μας συνίσταται το **PyCharm 2022.1.2**)  
<https://www.jetbrains.com/pycharm/download/#section=windows>
2. Κατεβάζουμε την python 3.10  
<https://www.python.org/downloads/>
3. Αφού έχουμε εγκαταστήσει τα πιο πάνω στο Python IDE μας θα ανοίξουμε τα αρχεία του προγράμματος μας open το φάκελο Intelligent\_Agents\_Generic\_Planner
4. Τρέχουμε το αρχείο **main** που είναι το κεντρικό μας αρχείο όπου εκεί γίνεται χρήση των άλλων αρχείων για την λύση των προβλημάτων μας.

## Επεκτάσεις/Εργαλεία

Python 3.10

PyCharm 2021.3

## Βιβλιογραφία

- IntelligentAgents\_Vlahavas(σημειώσεις από το μάθημα gunet2)
- <https://www.geeksforgeeks.org/water-jug-problem-using-bfs/>
- <https://www.geeksforgeeks.org/a-search-algorithm/>
- <https://towardsdatascience.com/a-star-a-search-algorithm-eb495fb156bb>
- [https://en.wikipedia.org/wiki/Blocks\\_world#:~:text=The%20blocks%20world%20is%20one,more%20vertical%20stacks%20of%20blocks.](https://en.wikipedia.org/wiki/Blocks_world#:~:text=The%20blocks%20world%20is%20one,more%20vertical%20stacks%20of%20blocks.)
- <https://apoorvdixit619.medium.com/goal-stack-planning-for-blocks-world-problem-41779d090f29>