Νικόλαος Γιαννόπουλος ΑΜ5199

Δημήτριος Μάριος Σαρηγιάννης ΑΜ5345

Γεώργιος Στρούγγης ΑΜ5357

Για την εκτέλεση του παιχνιδιού 8-puzzle χρειάστηκε να υλοποιήσουμε τις εξής κλάσεις: board, node, puzzle8, UCS, A_star.

board

Η κλάση αυτή περιέχει ένα στιγμιότυπο του πίνακα σε κάποιο βήμα της εκτέλεσης καθώς και ένα αντίγραφο της τελικής κατάστασης με μορφή πίνακα 3x3. Ο constructor του αρχικοποιεί την τωρινή κατάσταση του πίνακα ως ΑΚ και βρίσκει τις συντεταγμένες του κενού στοιχείου στα pos_i, pos_j, καθώς θα μας είναι χρήσιμη στο μέλλον

Η μέθοδος showCurrent μας δείχνει την τωρινή κατάσταση του πίνακα. Η getDifference επιστρέφει τον αριθμό των θέσεων του πίνακα που περιέχουν διαφορετικά στοιχεία από τις αντίστοιχες θέσεις στην τελική κατάσταση. Η copy δημιουργεί και επιστρέφει ένα ακριβές αντίγραφο του πίνακα. Η isSameBoard δέχεται δεύτερο πίνακα και ελέγχει αν είναι ίδιος με τον τωρινό.

Η move δέχεται ως είσοδο την κατεύθυνση στην οποία επιθυμούμε να κινηθεί το κενό στοιχείο και εφόσον αυτή η κίνηση είναι επιτρεπτή και δεν βγαίνει εκτός ορίων την εκτελεί. Ανταλλάσσει την θέση του κενού με τον αριθμό που βρίσκεται στην επιθυμητή κατεύθυνση και ενημερώνει τον πίνακα καθώς και τις συντεταγμένες του κενού. Στο τέλος αν εκτέλεσε την κίνηση επιστρέφει τον πίνακα στην νέα του κατάσταση, αλλιώς επιστρέφει null.

Η μέθοδος surroundings δημιουργεί οκτώ αντίγραφα του πίνακα, ένα για να εκτελέσει κάθε πιθανή κίνηση μέσω move. Κάθε ένα το οποίο εκτελείται επιτυχώς εισέρχεται σε ένα ArrayList εν ονόματι surroundings, το οποίο αποτελεί τους “γείτονες” της τωρινής κατάστασης του πίνακα και εν τέλει επιστρέφει.

Οι μέθοδοι της board είναι απαραίτητες για την εκτέλεση του προγράμματος όσον αφορά την κατάσταση του πίνακα.

node

Το κάθε node περιέχει τέσσερα δεδομένα: Board, level, parent, cost. Το Board περιέχει ένα αντικείμενο board για μια κατάσταση του πίνακα. Το level περιέχει το βάθος της node στο δέντρο αναζήτησης που δημιουργείται από τις μεθόδους A* και UCS. Το parent παρέχει ένα άλλο node από το οποίο το τωρινό προήλθε, το πρώτο node του δέντρου παίρνει parent = null. Το cost χρησιμοποιείται για την μέθοδο A* και αντί να παίρνει είσοδο στον constructor όπως τα προηγούμενα τρία θα υπολογιστεί μέσω της ευρετικής συνάρτησης που επιλέξαμε να υλοποιήσουμε. Πιο πολλές λεπτομέρειες θα το κόστος συμπεριλαμβάνονται στην εξήγηση του A_star.

Ως μέθοδοι της node υπάρχουν μόνο getters για τα δεδομένα της. Σκοπός της node είναι να κρατάει στιγμιότυπα από board καθώς και πληροφορίες που τα συνδέουν

μεταξύ τους και είναι απαραίτητα για την εκτέλεση των A* και UCS. Κατά τα άλλα δεν εκτελούν καμία μέθοδο τα ίδια τα node.

UCS

Η κλήση αυτής της κλάσης με είσοδο έναν πίνακα 3x3 εκτελεί μέθοδο αναζήτησης ομοιόμορφου κόστους. Απαραίτητες βιβλιοθήκες για την εκτέλεση αποτελούν οι ArrayList, PriorityQueue και HashSet.

Η UCS εφαρμόζει δέντρο αναζήτησης μέσω BFS. Για να υλοποιηθεί αυτό χρειαζόμαστε μια priority queue εν ονόματι nodes για να κρατάει τα node κατά την διάρκεια της εκτέλεσης. Το πλεονέκτημα μιας priority queue είναι ότι θα εξερευνήσει τα node πιο κοντά στην ρίζα του δέντρου αναζήτησης. Η παράμετρος (n1, n2) -> n1.getLevel() - n2.getLevel() που χρησιμοποιείται για την αρχικοποίηση της priority queue δηλώνει ότι μεγαλύτερη προτεραιότητα θα έχει ένα node n1 από το n2 αν το level του είναι μικρότερο. Έτσι θα επιλέγει node τα οποία βρίσκονται πιο κοντά στην ρίζα.

Δημιουργεί ένα αντικείμενο board με τον 3x3 πίνακα που δέχεται ως είσοδο και με αυτό το πρώτο node σε επίπεδο 0 και γονέα null, καθώς είναι η ρίζα του δέντρου. Είναι σημαντικό για να μειώσουμε τον αριθμό των επεκτάσεων και να κρατάμε τον χρόνο εκτέλεσης χαμηλό να μην επισκεπτόμαστε τους ίδιους κόμβους δύο φορές, γι' αυτό δημιουργείται ένα HashSet με το όνομα used που θα εγγυάται ότι κάθε κόμβος έχει επισκεφθεί μόνο μια φορά.

Στην συνέχεια εκτελείται ένα while loop το οποίο θα τρέχει μέχρι η σειρά προτεραιότητας αδειάσει ή βρεθεί η τελική κατάσταση. Πρώτα διαλέγει την επόμενη node από την σειρά προτεραιότητας και εκτελεί την μέθοδο surroundings για τον πίνακα της, βρίσκοντας όλους τους γείτονες. Αν κάποιος γείτονας δεν έχει ήδη παρουσιαστεί στην used τότε προστίθεται στα priority queue και hash set, ενώ αν έχει ήδη επισκεφθεί αυτό το node τότε το προσπερνά. Στο τέλος είτε θα έχει βρει την τελική κατάσταση είτε θα εξασθενήσει κάθε πιθανή κίνηση μέχρι να αδειάσει η σειρά προτεραιότητας.

Αφότου βρει την τελική κατάσταση θα την βάλει καθώς και την σειρά από κόμβους που οδήγησαν από την ρίζα σε αυτή σε ένα ArrayList. Αυτό επιτυγχάνεται μέσω των γονέων των κόμβων. Στο τέλος εκτυπώνεται το μονοπάτι από την αρχική στην τελική κατάσταση καθώς και το κόστος του και ο αριθμός τον επεκτάσεων. Για την μέθοδο UCS το κόστος ισχύει η σχέση f(n) = g(n) που σημαίνει ότι θα ισοδυναμεί με το επίπεδο στο οποίο βρέθηκε η τελική κατάσταση.

A_star

Η A* μέθοδος που χρησιμοποιούμε είναι παρόμοια στην υλοποίηση με την UCS, μόνο που η σειρά προτεραιότητας δέχεται είσοδο (n1, n2) -> n1.getCost() - n2.getCost(). Μεγαλύτερη προτεραιότητα θα έχει η node n1 αν το κόστος της είναι μικρότερο από το κόστος της n2.

Για την εύρεση του κόστους για την οποία μιλήσαμε νωρίτερα αξιοποιείται η ευρετική συνάρτηση h(n). Μέσω της μεθόδου getDifference της κλάσης board βρίσκεται η διαφορά μεταξύ την τωρινή και την τελική κατάσταση, αυτό θα αποτελεί την ευρετική μας συνάρτηση καθώς θα προχωράμε στο δέντρο αναζήτησης δίνοντας προτεραιότητα στους κόμβους με την μικρότερη διαφορά από την τελική κατάσταση.

Η σχέση για το κόστος θα είναι f(n) = g(n) + h(n), δηλαδή το επίπεδο της τωρινής κατάστασης + την ‘απόσταση’ του από την τελική.

Αυτή η συνάρτηση είναι αποδεκτή γιατί μας βοηθάει να επικεντρωνόμαστε σε πίνακες που βρίσκονται πιο κοντά στην επιθυμητή κατάσταση για να την φτάνουμε με τα λιγότερα πιθανά βήματα.

Όπως την UCS θα βρίσκει τους γείτονες και θα αποκλείει καταστάσεις τις οποίες έχουμε ήδη επισκεφτεί για να αποφύγει περεταίρω επεκτάσεις. Τέλος εκτυπώνει το μονοπάτι από την αρχική προς την τελική κατάσταση, το κόστος του το οποίο υπολογίστηκε με την παραπάνω συνάρτηση και τον αριθμό των επεκτάσεων.

puzzle8

Περιέχει την main μας, ο τρόπος με τον οποίο δέχεται είσοδο είναι ο χρήστης να πληκτρολογεί έναν αριθμό ή ένα space και να πατάει enter μέχρι να συμπληρώσει τον πίνακα. Σε περίπτωση που ο χρήστης προσπαθήσει να βάλει κάποια διαφορετική είσοδο πέρα από τους αριθμούς 1 έως 8 και το space (πχ γράμμα ή σύμβολο) ή βάλει κάποιο δεκτό στοιχείο το οποίο όμως έχει ήδη εισαχθεί τότε δεν θα το δεχτεί και θα του ζητήσει να βάλει μια αποδεκτή είσοδο.

Με αυτές τις εισόδους δημιουργεί έναν πίνακα ο οποίος θα χρησιμοποιηθεί για την κλήση των μεθόδων A* και UCS, οι οποίες θα φτιάξουν αντικείμενο board και node με αυτό.

Στην συνέχεια ρωτάει τον χρήστη ποια μέθοδο επιθυμεί να εκτελέσει και δρα σύμφωνα.

Για να δείξουμε τις δυνατότητες των μεθόδων καθώς και τις διαφορές τους επιλέξαμε τις ακόλουθες 5 εισόδους. Τα αποτελέσματα και οι επιδόσεις τους σε “εύκολες” και “δύσκολες” περιπτώσεις φαίνονται στις ακόλουθες εικόνες.

1) AK = [[5, ,4],[6,7,3],[8,2,1]]

A*=

```
5


4
6
7
3
8
2
1
Select method to execute: 0 for A* and 1 for UCS
0

5   4
6 7 3
8 2 1

  5 4
6 7 3
8 2 1

6 5 4
  7 3
8 2 1

6 5 4
7   3
8 2 1
```

```
6 5 4
7 2 3
8   1

6 5 4
7 2 3
8 1

6 5 4
7   3
8 1 2
Cost: 6
Number of expansions: 78
```

UCS=

```
5


4
6
7
3
8
2
1
Select method to execute: 0 for A* and 1 for UCS
1

5   4
6 7 3
8 2 1

  5 4
6 7 3
8 2 1

6 5 4
  7 3
8 2 1

6 5 4
7   3
8 2 1
```

```
6 5 4
7 1 3
8 2

6 5 4
7 1 3
8   2

6 5 4
7   3
8 1 2
Cost: 6
Number of expansions: 19260
```

2) AK = [[6,4,5],[7,8,1],[,3,2]]

A*=

```
0
4
5
7
8
1


3
2
Select method to execute: 0 for A* and 1 for UCS
0

6 4 5
7 8 1
  3 2

6 4 5
7   1
8 3 2

6 4 5
7 3 1
8   2

6 4 5
7 3
8 1 2

6 4
7 3 5
8 1 2
```

```
6   4
7 3 5
8 1 2

6 5 4
7 3
8 1 2

6 5 4
7   3
8 1 2
Cost: 7
Number of expansions: 319
```

UCS=

```
6
4
5
7
8
1


3
2
Select method to execute: 0 for A* and 1 for UCS
1

6 4 5
7 8 1
  3 2

6 4 5
7   1
8 3 2

6   5
7 4 1
8 3 2

6 5
7 4 1
8 3 2

6 5 4
7   1
8 3 2
```

```
6 5 4
7 1
8 3 2

6 5 4
7 1 3
8   2

6 5 4
7   3
8 1 2
Cost: 7
Number of expansions: 60292
```

---

3) AK = [[5,4,3],[8,7,],[6,1,2]]

A*=

```
5
4
3
8
7


0
1
2
Select method to execute: 0 for A* and 1 for UCS
0

5 4 3
8 7
6 1 2

5 4
8 7 3
6 1 2

5   4
8 7 3
6 1 2

5 7 4
8   3
6 1 2

5 7 4
8 6 3
  1 2
```

```
5 7 4
  6 3
8 1 2

5   4
7 6 3
8 1 2

  5 4
7 6 3
8 1 2

6 5 4
7   3
8 1 2
Cost: 8
Number of expansions: 794
```

UCS=

```
5
4
3
8
7


6
1
2
Select method to execute: 0 for A* and 1 for UCS
1

5 4 3
8 7
6 1 2

5 4
8 7 3
6 1 2

5   4
8 7 3
6 1 2

5 7 4
8   3
6 1 2

5 7 4
8 6 3
  1 2
```

```
5 7 4
  6 3
8 1 2

5   4
7 6 3
8 1 2

  5 4
7 6 3
8 1 2

6 5 4
7   3
8 1 2
Cost: 8
Number of expansions: 996171
```

4) AK= [[2,6,],[7,1,5],[8,4,3]]

A*=

```
2
6

7
1
5
8
4
3
Select method to execute: 0 for A* and 1 for UCS
0

2 6
7 1 5
8 4 3

2 6 5
7 1
8 4 3

2 6 5
7 1 4
8   3

2 6 5
7   4
8 1 3

  6 5
7 2 4
8 1 3

6   5
7 2 4
8 1 3
```

```
6 5
7 2 4
8 1 3

6 5 4
7 2
8 1 3

6 5 4
7 2 3
8 1

6 5 4
7   3
8 1 2
Cost: 9
Number of expansions: 1630
```

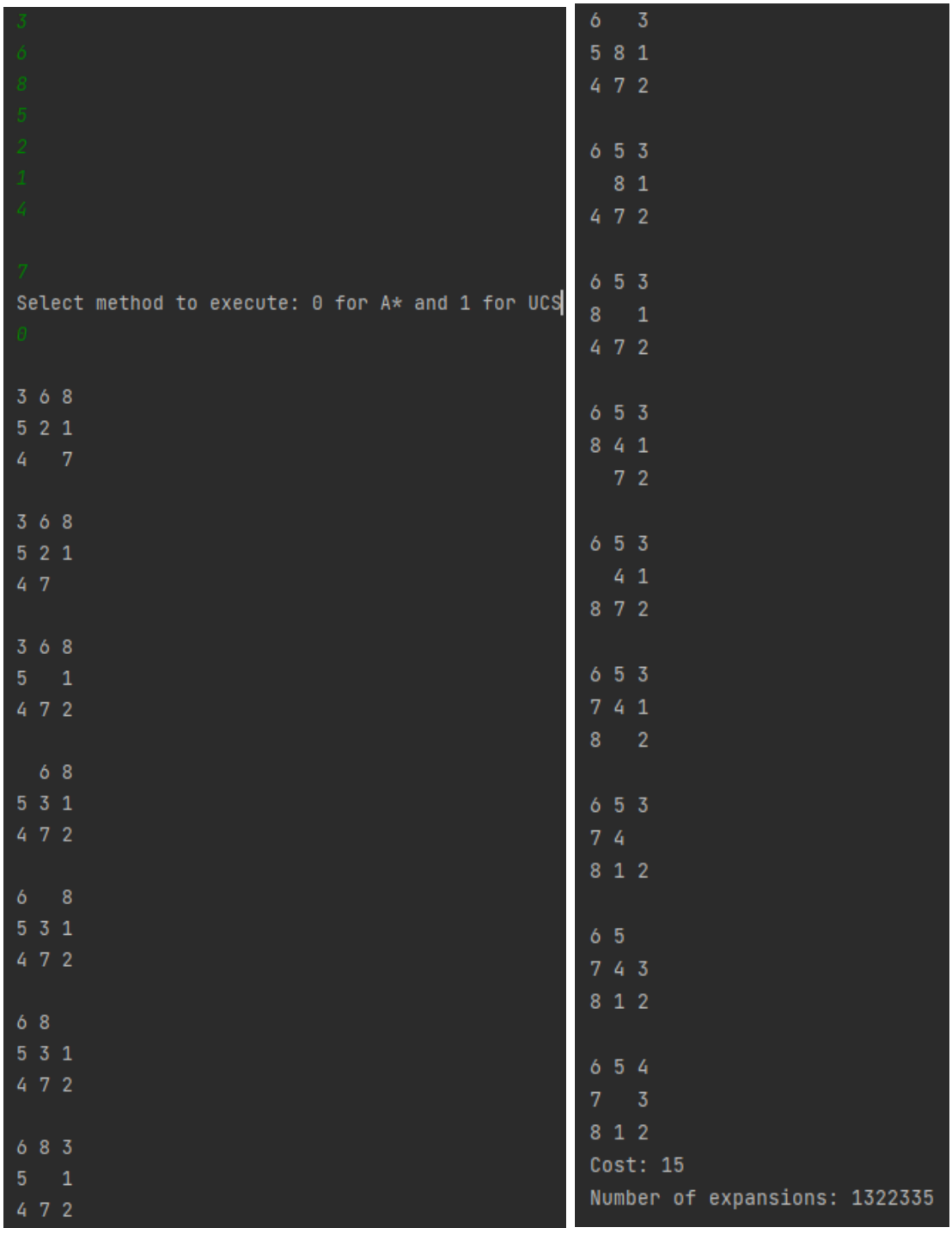
UCS=

```
2
6


7
1
5
8
4
3
Select method to execute: 0 for A* and 1 for UCS
1

2 6
7 1 5
8 4 3

2 6 5
7 1
8 4 3

2 6 5
7 1 4
8   3

2 6 5
7   4
8 1 3

  6 5
7 2 4
8 1 3

6   5
7 2 4
8 1 3
```

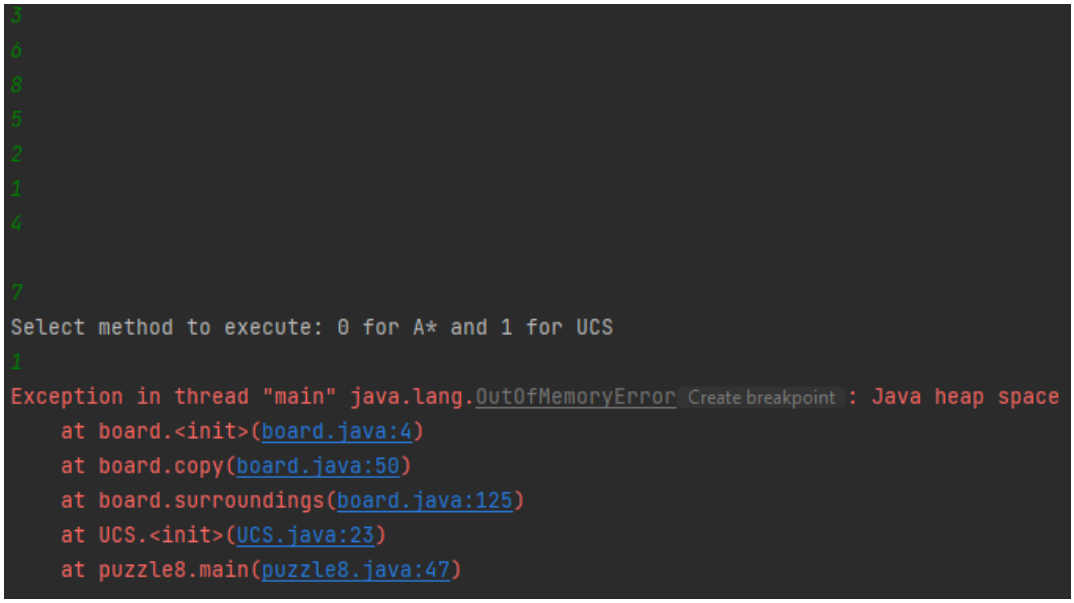
```
6 5
7 2 4
8 1 3

6 5 4
7 2
8 1 3

6 5 4
7 2 3
8 1

6 5 4
7   3
8 1 2
Cost: 9
Number of expansions: 1408487
```

5) AK = [[3,6,8],[5,2,1],[4, ,7]]

A*=



UCS=



Σύγκριση απόδοσης A* με UCS

Από τα πέντε τεστ που πραγματοποιήθηκαν για τις δύο μεθόδους συμπεραίνουμε ότι η A* παρουσιάζει καλύτερες αποδόσεις. Ο υπολογισμός της προτεραιότητας μέσω του κόστους από την ευρετική συνάρτηση αντί για το επίπεδο αποδεικνύεται ότι οδηγεί στην τελική κατάσταση όχι μόνο πιο γρήγορα αλλά και με μειωμένη πιθανότητα για σφάλμα.

Η μέθοδος UCS δημιουργεί πολύ μεγαλύτερο αριθμό επεκτάσεων, κάτι το οποίο ρισκάρει να καταναλώσει όλο τον διαθέσιμο χώρο που θα οδηγήσει σε σφάλμα και πρόωρο τερματισμό χωρίς αποτέλεσμα σε ακραίες περιπτώσεις όπως φαίνεται στο πέμπτο παράδειγμα. Καθώς οι δύο μέθοδοι αναπτύσσουν το δέντρο αναζήτησης με διαφορετικές προτεραιότητες θα εκτελούνται και με διαφορετικούς ρυθμούς, ως αποτέλεσμα η A* επιπλέον να ολοκληρώνει και πιο γρήγορα, αν και στις δύο περιπτώσεις ο χρόνος δεν προσπερνά δύο με τρία δευτερόλεπτα.

Επίσης αξιοσημείωτο είναι ότι το μονοπάτι που παίρνουν οι δύο μέθοδοι εν μέρη διαφέρει ως προς ποιες κινήσεις πραγματοποιήθηκαν στα πρώτα τρία παραδείγματα.

Αν και η τελική κατάσταση πάντα βρίσκετε στο ίδιο επίπεδο ανεξάρτητα από την μέθοδο και οι χρόνοι είναι παρόμοιοι, η τάση της UCS να αποτυγχάνει να βρει αποτέλεσμα με τον διαθέσιμο χώρο την κάνει αναξιόπιστη σε σύγκριση με την A*.