

# SISTEMAS INTELIGENTES

## A\* LABORATORY EXERCISE

### **Group C :**

Cergan Radu-Mihai

Micu Cristian-Mihai

Plaisanu Alexandru-Ciprian

# PROBLEM SUMMARY

In this problem we have a battlefield with an agent which is generated in a initial position and his task is to parse the shortest path to a final position avoiding obstacles (The SittingDuck robots) .

The SittingDucks must be generated in a random position and they occupy 30% of the battlefield. The initial and final positions of the agent are also generated randomly, but the condition is that the tile must be unoccupied by an obstacle.

The battlefield contains multiple tiles by 64x64 pixels. Those can also be viewed as states for A\* algorithm. Each robot is 36x36 pixels in size.

To reach the goal of this problem we use the A\* algorithm to return the path the robot has to follow.




The whole process will be launched by a Java application that uses the robocode.control package to set up a battle according to the above specifications, so that we can see your robot moving through the battlefield according to the sequence of actions found by A\*.

# DESIGN AND IMPLEMENTATION



We have a workspace that is formed from two projects :

- Java application that launches the Robocode through Robocode engine
- The robot project

 .metadata	Date modified: 4/24/2020 6:10 PM
 AppLaunchRobocode	Date modified: 4/21/2020 5:15 PM
 myRobot	Date modified: 4/23/2020 1:41 PM

# DESIGN AND IMPLEMENTATION

- JAVA APPLICATION

- ▶ The java application has only one class which includes the main where is created one battlefield and some arrays for storing all types of information about the robots. All of these are necessary to launch the Robocode through robocode engine.
- ▶ In order to create the battlefield(2D array that contains only Tile objects), we have to set the dimensions of the battlefield(length, width) and in our case those are 640x640.
- ▶ For generating the obstacle map, we used a list that contains all integers between 0 and the product of the number of tiles per row and the number of tiles per column. All these integers represents every position on battlefield.
- ▶ We've used that list to avoid generating more obstacles on the same tile.
- ▶ For inserting the robots(SittingDuck) with their specifications in Robocode we have to convert the generated number of tile per row and the number of tile per column into double(pixels).

# DESIGN AND IMPLEMENTATION

```
routeBot.java RouteFinder.java ✕
1 package searchpractice;
2 import java.util.List;
3
4 public class RouteFinder {
5     public static void main(String[] args) {
6         RobocodeEngine engine = new RobocodeEngine(new java.io.File("C:/robocode"));
7         engine.setVisible(true);
8
9         BattlefieldSpecification battlefield1 = new BattlefieldSpecification(640, 640);
10        int numberOfRounds = 5;
11        long inactivityTime = 10000000;
12        double gunCoolingRate = 1.0;
13        int sentryBorderSize = 50;
14        boolean hideEnemyNames = false;
15        int tileSize = 64;
16        int halfTile = (tileSize / 2);
17        int numTileRows = (int)(640 / tileSize);
18        int numTileCols = (int)(640 / tileSize);
19        int verticalOffset = 640 % 64;
20
21        List<Integer> remainingPossiblePositions = routeBot.generateAllPositions(numTileRows * numTileCols);
22        double sittingDuckPercentage = 0.30;
23        int numObstacles = (int)(numTileRows * numTileCols * sittingDuckPercentage);
24        RobotSpecification[] modelRobots = engine.getLocalRepository("sample.SittingDuck, searchpractice.routeBot*");
25        RobotSpecification[] existingRobots = new RobotSpecification[numObstacles + 1];
26        RobotSetup[] robotSetups = new RobotSetup[numObstacles + 1];
27        Random randomGenerator = new Random();
28        randomGenerator.setSeed(9);
29        for(int it = 0; it < numObstacles; ++it) {
30            int idx1 = randomGenerator.nextInt(remainingPossiblePositions.size());
31            int position1 = remainingPossiblePositions.remove(idx1);
32
33            int initialTileRow1 = (int)(position1 % numTileRows);
34            int initialTileCol1 = (int)(position1 / numTileRows);
35            double initialObstacleRow = (double)(initialTileRow1 * tileSize + halfTile);
36            double initialObstacleCol = (double)(initialTileCol1 * tileSize + halfTile + verticalOffset);
37
38            robotSetups[it] = new RobotSetup(initialObstacleRow, initialObstacleCol, 0.0);
39            existingRobots[it] = modelRobots[0];
40        }
41    }
42 }
```



# DESIGN AND IMPLEMENTATION

- ▶ Also, we need to generate the initial and the final position of the agent. We do that by doing the same thing we did to the obstacle positions (Slide 4 last paragraph).
- ▶ Now, all we have to do is to set the rules of the battle (numberOfRounds, inactivityTime, gunCoolingRate, sentryBorderSize, hideEnemyNames) and run it.

```
routeBot.java RouteFinder.java
37 double InitialObstacleCol = (double)(InitialTileCol1 * TileSize + halfTile + VerticalOffset);
38
39 robotSetups[it] = new RobotSetup(InitialObstacleRow, InitialObstacleCol, 0.0);
40 existingRobots[it] = modelRobots[0];
41 }
42 Random randomGenerator1 = new Random(7);
43 existingRobots[NumObstacles] = modelRobots[1];
44 // Generating the initial position of the agent
45 int idx = randomGenerator1.nextInt(remainingPossiblePositions.size());
46 int position = remainingPossiblePositions.remove(idx);
47
48 int InitialTileRow = position % NumTileRows;
49 int InitialTileCol = position / NumTileRows;
50
51 double InitialAgentRow = InitialTileRow * TileSize + halfTile;
52 double InitialAgentCol = InitialTileCol * TileSize + halfTile + VerticalOffset;
53
54 robotSetups[NumObstacles] = new RobotSetup(InitialAgentRow, InitialAgentCol, 0.0);
55
56 routeBot.startX = InitialTileRow;
57 routeBot.startY = InitialTileCol;
58
59 idx = randomGenerator1.nextInt(remainingPossiblePositions.size());
60 position = remainingPossiblePositions.remove(idx);
61
62 InitialTileRow = position % NumTileRows;
63 InitialTileCol = position / NumTileRows;
64
65 routeBot.endX = InitialTileRow;
66 routeBot.endY = InitialTileCol;
67 BattleSpecification battleSpec = new BattleSpecification(battlefield1, numberOfRounds, inactivityTime, gunCoolingRate, sentryBorderSize, hideEne
68
69 engine.runBattle(battleSpec, true);
70 engine.close();
71 System.exit(0);
72 }
73
74 }
75
```

# DESIGN AND IMPLEMENTATION

- ROBOT PROJECT

- ▶ This project contains only one class called routeBot which extends the Robot class making our class to behave as a robot.
- ▶ This class contains a static nested class(Tile) which represents the states for the A\* algorithm.
- ▶ One tile contains the estimated cost of the cheapest path from the state at node  $n$  to a goal state( heuristic  $h(n)$  ), the real path cost from the initial state to the state at node  $n$ (  $g(n)$  ), the evaluation function(  $f(n)$  ) which is the sum between  $h(n)$  and  $g(n)$ , a boolean variable which checks if a tile was visited, the coordinates of the tile on the battlefield. It also has a previous tile, which is called parent and this helps us to reconstruct the path.

# DESIGN AND IMPLEMENTATION

- ▶ Our robot also has different methods that help us to reach the goal of the problem. We will present those below :
  - ❖ *generateAllPositions(int n)*
    - it is a method that returns a list with all integers between 0 and n
  - ❖ *setBlocked(Tile[][] battlefield, int x, int y)*
    - this method is used to block the tile at x, y coordinates when an obstacle was generated on it
  - ❖ *ManhattanDistance(Tile tile)*
    - this method helps us to compute the estimated cost of the cheapest path from the state at node n to a goal state(heuristic  $h(n)$ )
  - ❖ *updateCost(Tile currentTile, Tile nextTile)*
    - this method computes the evaluation function of the nextTile which represents the neighbor of currentTile and add it in the openSet

!! See the code of the above functions at the [Slide 9](#) !!



# DESIGN AND IMPLEMENTATION

```
182 public static List<Integer> generateAllPositions(int n){
183     List<Integer> list = new ArrayList<Integer>(n);
184     for(int x = 0; x < n; x++){
185         list.add(x);
186     }
187     return list;
188 }
189 public static void setBlocked(Tile[][] battlefield, int x, int y){
190     battlefield[x][y] = null;
191 }
192 public static int MovementCost = 1;
193 public static PriorityQueue<Tile> openSet;
194
195 static int startX, startY;
196 static int endX, endY;
197
198 public static int ManhattanDistance(Tile tile) {
199     return(Math.abs(tile.x - endX) + Math.abs(tile.y - endY));
200 }
201 public static void updateCost(Tile currentTile, Tile nextTile) {
202     if(nextTile == null || nextTile.closed)
203         return;
204     nextTile.h = ManhattanDistance(nextTile);
205     int nextTile_f = nextTile.h + currentTile.g + MovementCost;
206     boolean inOpenSet = openSet.contains(nextTile);
207     if(!inOpenSet || nextTile_f < nextTile.f){
208         nextTile.f = nextTile_f;
209         nextTile.g = currentTile.g + MovementCost;
210         nextTile.parent = currentTile;
211         if(!inOpenSet)
212             openSet.add(nextTile);
213     }
214 }
215 }
```

# DESIGN AND IMPLEMENTATION

❖ *Astar(Tile[][] battlefield, int startX, int startY, int endX, int endY)*

- represents the method that aims to find a path to the final position having the smallest cost starting from a specific initial position
- for implementing this method, we used the pseudocode attached on the next slide (Slide 11), but we adjusted it in order to find a sequence of actions (moves) by adding a suitable heuristic function
- to determine the path that the robot has to parse we have to begin from the coordinates of the final position and determine where is located his parent.
  - if the parent is below the current then we have to push “0” integer(that means the robot has to go **UP**) in the stack
  - if the parent is on the left side of the current then we have to push “1” integer(that means the robot has to go **RIGHT**) in the stack
  - if the parent is above of the current then we have to push “2” integer(that means the robot has to go **DOWN**) in the stack
  - if the parent is on the right side of the current then we have to push “3” integer(that means the robot has to go **LEFT**) in the stack
- in the end of the method, if the final position is not reached it will return NULL(the stack is NULL)

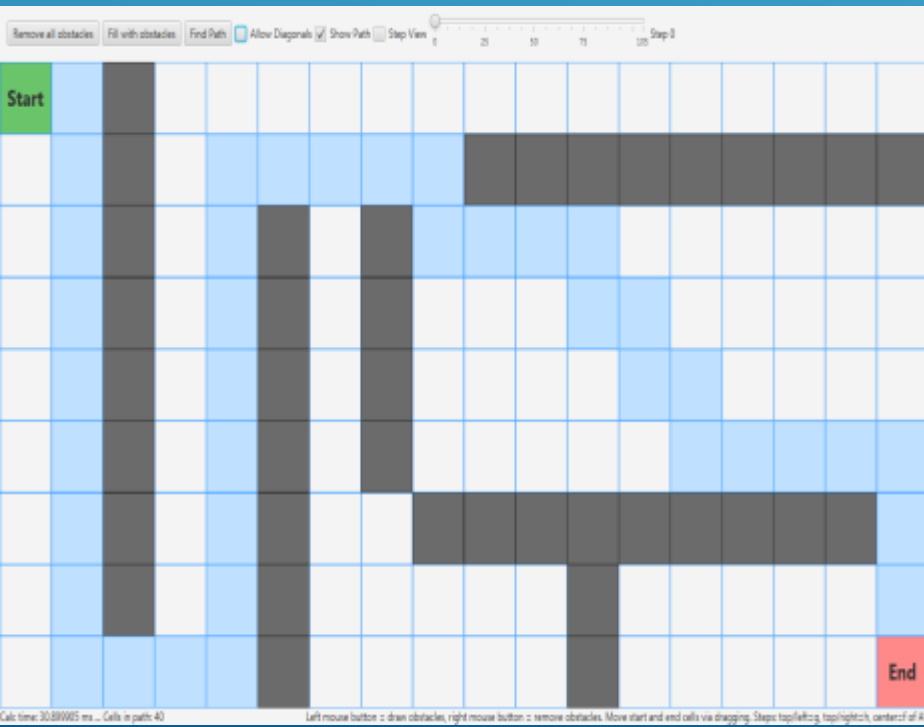
For AStar() method check Slide 12

# DESIGN AND IMPLEMENTATION

```
closedset := the empty set           // The set of nodes already evaluated
openset := {start}                   // The set of tentative nodes to be evaluated
parent := the empty map              // The map of navigated nodes
g[start] := 0                        // Cost from start along best known path
f[start] := g[start] + h(start)       // Estimated total cost from start to nearest goal through y
while openset is not empty
    current := the node in openset having the lowest f[] value
    if is_goal(current)
        return reconstruct_path(parent, current)
    remove current from openset
    add current to closedset
    for each neighbor in neighbor_nodes(current)
        if neighbor in closedset continue
        tentative_g := g[current] + dist_between(current, neighbor)
        if neighbor not in openset or tentative_g < g[neighbor]
            parent[neighbor] := current
            g[neighbor] := tentative_g
            f[neighbor] := g[neighbor] + h(neighbor)
            if neighbor not in openset
                add neighbor to openset
return failure
```



# DESIGN AND IMPLEMENTATION



```

216 public static Stack<Integer> AStar(Tile[][] battlefield, int startX, int startY, int endX, int endY) {
217     openSet = new PriorityQueue<Tile>((Tile t1, Tile t2) -> { if(t1.f < t2.f) return -1;
218     else if (t1.f > t2.f) return 1;
219     else return 0;});
220
221     openSet.add(new Tile(startX, startY));
222     Tile current;
223     while(true) {
224         current = openSet.poll();
225         if(current == null)
226             break;
227         current.closed = true;
228         if(current.x == endX && current.y == endY) {
229             break;
230         }
231         Tile next;
232         if(current.x - 1 >= 0) {
233             next = battlefield[current.x - 1][current.y];
234             updateCost(current, next);
235         }
236         if(current.x + 1 < battlefield.length) {
237             next = battlefield[current.x + 1][current.y];
238             updateCost(current, next);
239         }
240         if(current.y + 1 < battlefield[0].length) {
241             next = battlefield[current.x][current.y + 1];
242             updateCost(current, next);
243         }
244         if(current.y - 1 >= 0){
245             next = battlefield[current.x][current.y - 1];
246             updateCost(current, next);
247         }
248     }
249     Stack<Integer> path = new Stack<Integer>();
250     if(battlefield[endX][endY].closed == true) {
251         Tile current1 = battlefield[endX][endY];
252         while(current1.parent != null) {
253             if(current1.y > current1.parent.y)
254                 path.push(0);
255             else if(current1.x > current1.parent.x)
256                 path.push(1);
257             else if(current1.y < current1.parent.y)
258                 path.push(2);
259             else
260                 path.push(3);
261             current1 = current1.parent;
262         }
263         return path;
264     }
265     else
266         return null;
267 }
268 }
269 }
270 }
    
```

# DESIGN AND IMPLEMENTATION

## ❖ *run()*

- the main method in every robot, we must override this to set up our robot's behavior.
- in order to move our robot we have to store the path generated by A\* algorithm in a stack
- we iterate through the stack and interpret the integers between 0 and 3 as robot movement, like we said at the Slide 10
- at every move of the robot we must test his heading, and if it is not oriented correctly we must turn it with the heading on the next tile of the path
- in case of an empty stack we will print the message "No path found!" in the robot console

For *run()* method, check Slide 14 and Slide 15

```

22 public void run()
23 {
24     int NumTileRows = 10;
25     int NumTileCols = 10;
26     Tile[][] battlefield = new Tile[NumTileRows][NumTileCols];
27     for(int i = 0; i < NumTileRows; ++i)
28     {
29         for(int j = 0; j < NumTileCols; ++j)
30         {
31             battlefield[i][j] = new Tile(i, j);
32         }
33     }
34
35     List<Integer> remainingPossiblePositions = generateAllPositions(NumTileRows * NumTileCols);
36     double SittingDuckPercentage = 0.30;
37     int NumObstacles = (int)(NumTileRows * NumTileCols * SittingDuckPercentage);
38
39     Random randomGenerator = new Random();
40     randomGenerator.setSeed(9);
41     int idx1;
42     int position1;
43     int InitialTileRow1;
44     int InitialTileCol1;
45     for(int it = 0; it < NumObstacles; ++it) {
46         idx1 = randomGenerator.nextInt(remainingPossiblePositions.size());
47         position1 = remainingPossiblePositions.remove(idx1);
48
49         InitialTileRow1 = (int)(position1 % NumTileRows);
50         InitialTileCol1 = (int)(position1 / NumTileRows);
51         setBlocked(battlefield, InitialTileRow1, InitialTileCol1);
52     }
53     Random randomGenerator1 = new Random(7);
54     int idx = randomGenerator1.nextInt(remainingPossiblePositions.size());
55     int position = remainingPossiblePositions.remove(idx);
56
57     int InitialTileRow = (int)(position % NumTileRows);
58     int InitialTileCol = (int)(position / NumTileRows);
59     startX = InitialTileRow;
60     startY = InitialTileCol;
61
62     out.println("x : " + InitialTileRow + "y : " + InitialTileCol);
63     idx = randomGenerator1.nextInt(remainingPossiblePositions.size());
64     position = remainingPossiblePositions.remove(idx);
65
66     int InitialTileRow2 = (int)(position % NumTileRows);
67     int InitialTileCol2 = (int)(position / NumTileRows);
68     endX = InitialTileRow2;
69     endY = InitialTileCol2;
70     out.println("x : " + InitialTileRow2 + "y : " + InitialTileCol2);
71     Stack<Integer> aux = AStar(battlefield, startX, startY, endX, endY);
72     if(!aux.empty())
73     {
74         while(!aux.empty())
75         {
76             Integer it = aux.pop();
77             if(it == 0)
78             {
79                 if(getHeading() == 90)
80                 {
81                     turnLeft(90);
82                     ahead(64);
83                 }
84                 else if(getHeading() == 0)
85                 {
86                     ahead(64);
87                 }
88                 else if(getHeading() == 180)
89                 {
90                     turnLeft(180);
91                     ahead(64);
92                 }
93                 else if(getHeading() == 270)
94                 {
95                     turnRight(90);
96                     ahead(64);
97                 }
98             }
99             else if(it == 1)

```



```

100 {
101     if(getHeading() == 90)
102     {
103         ahead(64);
104     }
105     else if(getHeading() == 0)
106     {
107         turnRight(90);
108         ahead(64);
109     }
110     else if(getHeading() == 180)
111     {
112         turnLeft(90);
113         ahead(64);
114     }
115     else if(getHeading() == 270)
116     {
117         turnRight(180);
118         ahead(64);
119     }
120 }
121 else if(it == 2)
122 {
123     if(getHeading() == 90)
124     {
125         turnRight(90);
126         ahead(64);
127     }
128     else if(getHeading() == 0)
129     {
130         turnRight(180);
131         ahead(64);
132     }
133     else if(getHeading() == 180)
134     {
135         ahead(64);
136     }
137     else if(getHeading() == 270)
138
139
140
141
142
143
144
145
146
147
148
149
150
151
152
153
154
155
156
157
158
159
160
161
162
163
164
165
166
167
168
169
170
171
172
173
174
175
176
177
178
179
180
181
182
183
184
185
186
187
188
189
190
191
192
193
194
195
196
197
198
199
200
201
202
203
204
205
206
207
208
209
210
211
212
213
214
215
216
217
218
219
220
221
222
223
224
225
226
227
228
229
230
231
232
233
234
235
236
237
238
239
240
241
242
243
244
245
246
247
248
249
250
251
252
253
254
255
256
257
258
259
260
261
262
263
264
265
266
267
268
269
270
271
272
273
274
275
276
277
278
279
280
281
282
283
284
285
286
287
288
289
290
291
292
293
294
295
296
297
298
299
300
301
302
303
304
305
306
307
308
309
310
311
312
313
314
315
316
317
318
319
320
321
322
323
324
325
326
327
328
329
330
331
332
333
334
335
336
337
338
339
340
341
342
343
344
345
346
347
348
349
350
351
352
353
354
355
356
357
358
359
360
361
362
363
364
365
366
367
368
369
370
371
372
373
374
375
376
377
378
379
380
381
382
383
384
385
386
387
388
389
390
391
392
393
394
395
396
397
398
399
400
401
402
403
404
405
406
407
408
409
410
411
412
413
414
415
416
417
418
419
420
421
422
423
424
425
426
427
428
429
430
431
432
433
434
435
436
437
438
439
440
441
442
443
444
445
446
447
448
449
450
451
452
453
454
455
456
457
458
459
460
461
462
463
464
465
466
467
468
469
470
471
472
473
474
475
476
477
478
479
480
481
482
483
484
485
486
487
488
489
490
491
492
493
494
495
496
497
498
499
500
501
502
503
504
505
506
507
508
509
510
511
512
513
514
515
516
517
518
519
520
521
522
523
524
525
526
527
528
529
530
531
532
533
534
535
536
537
538
539
540
541
542
543
544
545
546
547
548
549
550
551
552
553
554
555
556
557
558
559
560
561
562
563
564
565
566
567
568
569
570
571
572
573
574
575
576
577
578
579
580
581
582
583
584
585
586
587
588
589
590
591
592
593
594
595
596
597
598
599
600
601
602
603
604
605
606
607
608
609
610
611
612
613
614
615
616
617
618
619
620
621
622
623
624
625
626
627
628
629
630
631
632
633
634
635
636
637
638
639
640
641
642
643
644
645
646
647
648
649
650
651
652
653
654
655
656
657
658
659
660
661
662
663
664
665
666
667
668
669
670
671
672
673
674
675
676
677
678
679
680
681
682
683
684
685
686
687
688
689
690
691
692
693
694
695
696
697
698
699
700
701
702
703
704
705
706
707
708
709
710
711
712
713
714
715
716
717
718
719
720
721
722
723
724
725
726
727
728
729
730
731
732
733
734
735
736
737
738
739
740
741
742
743
744
745
746
747
748
749
750
751
752
753
754
755
756
757
758
759
760
761
762
763
764
765
766
767
768
769
770
771
772
773
774
775
776
777
778
779
780
781
782
783
784
785
786
787
788
789
790
791
792
793
794
795
796
797
798
799
800
801
802
803
804
805
806
807
808
809
810
811
812
813
814
815
816
817
818
819
820
821
822
823
824
825
826
827
828
829
830
831
832
833
834
835
836
837
838
839
840
841
842
843
844
845
846
847
848
849
850
851
852
853
854
855
856
857
858
859
860
861
862
863
864
865
866
867
868
869
870
871
872
873
874
875
876
877
878
879
880
881
882
883
884
885
886
887
888
889
890
891
892
893
894
895
896
897
898
899
900
901
902
903
904
905
906
907
908
909
910
911
912
913
914
915
916
917
918
919
920
921
922
923
924
925
926
927
928
929
930
931
932
933
934
935
936
937
938
939
940
941
942
943
944
945
946
947
948
949
950
951
952
953
954
955
956
957
958
959
960
961
962
963
964
965
966
967
968
969
970
971
972
973
974
975
976
977
978
979
980
981
982
983
984
985
986
987
988
989
990
991
992

```


# CHOICES DURING THE DESIGN AND IMPLEMENTATION PROCESS

- ▶ We choose the size of the battlefield to be 640x640 because we avoid getting half-tiles on the map
- ▶ To avoid generating 2 obstacles on the same tile we store all the possible positions from the battlefield in a list
- ▶ For design, we choose the heading of our robots to be 0.0
- ▶ We choose to create the openset of type PriorityQueue for an easier extract of the tile with the smallest evaluation function (f)
- ▶ We choose that the function Astar to return a stack because for generating the path that the robot has to follow we have to parse from final position to initial position and if we hadn't used a stack we would have to reverse the array(list, vector, set, etc)
- ▶ In order to verify if the obstacle maps coincides between them, we created one battlefield in main for printing the coordinates of all robots from our map

# MISTAKES

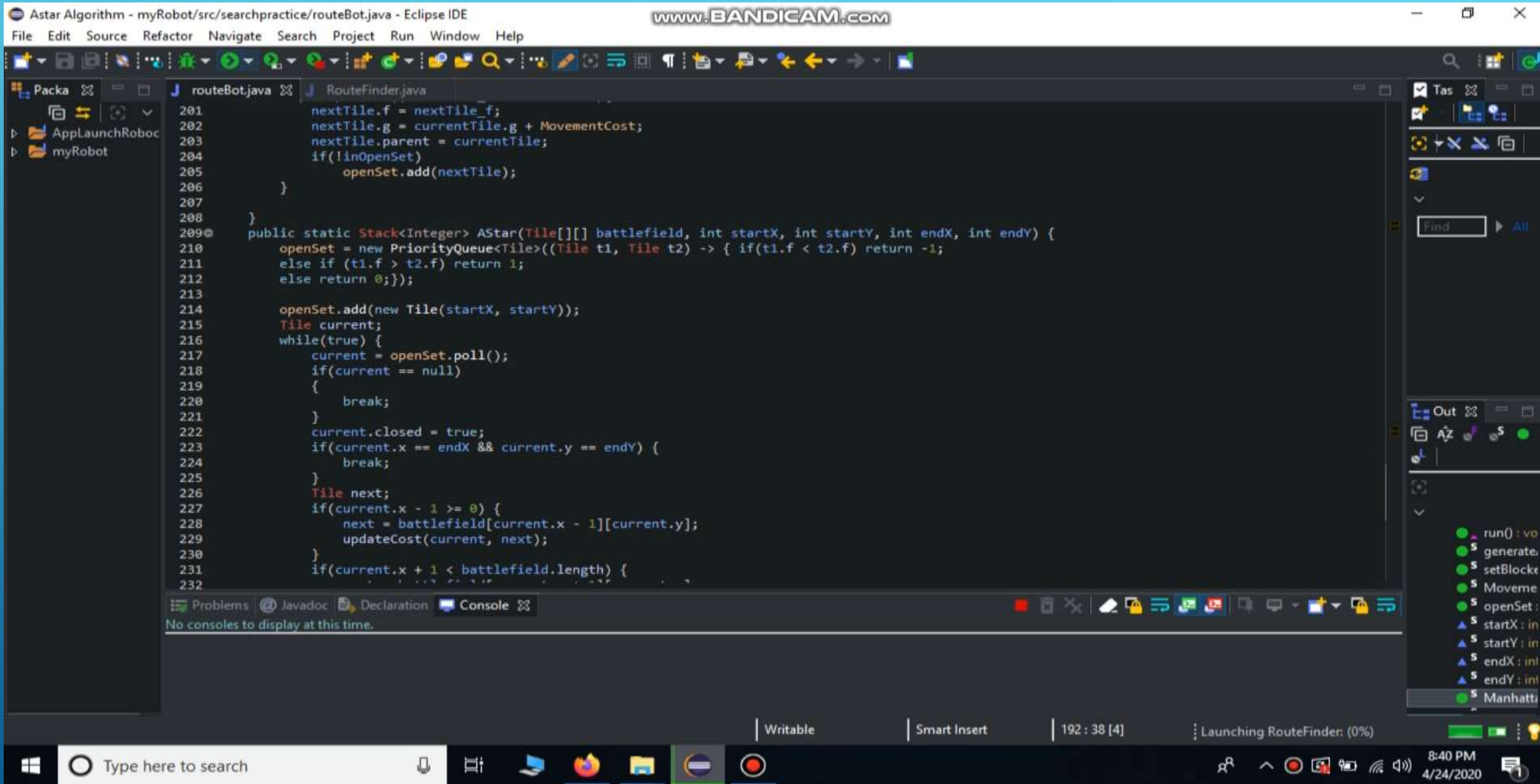
- ▶ One of the mistakes is that we set different seeds for generating the obstacle map on both projects and that made us realize that our robot was hitting obstacles and walls, so we decided to put the same seed for both java application and robot project and after that, it generated the same obstacle map
- ▶ When we thought that our project was finished, we ran the project and we saw that the robots are not placed correctly (on the middle of the tile), so we've seen that this problem could be solved by installing a newer version of Robocode
- ▶ Another mistake we did was putting all the code in a single project and we received an error, after reading the post on the forum, we created a new workspace that contains two different projects, one project for the java application that launches the Robocode through robocode engine and the other one representing our robot
- ▶ We generated the stack containing the path given by A\* algorithm outside the run() method, so there was no way to read the stack inside it. We decided that the only way to do this is to generate the stack inside the run() method.
- ▶ The last mistake was not setting the seeds for generating the positions of the agent, which caused that the positions were different, so setting the same seed solved our final problem and after that we got drunk 😊

# CONCLUSIONS

- ▶ We learned robocode which was a new thing for us because we never worked with graphical engines.
  - ▶ This project was a good introduction in Java for us as well, because we worked only in C, C++ and a little bit of Python. We learned how to manage classes in Java and how to work in Eclipse IDE because we only worked in Visual Studio before
  - ▶ We improved our abilities of team-working, which made us finish the project faster
  - ▶ Another conclusion is that every time you encounter a problem, you should search for the problem on the internet, forums and not blocking on it
- 
- Several white lines of varying lengths and slopes are positioned in the bottom right corner of the slide, creating a modern, abstract graphic element.



Here we attached a video of our robot succeeding his task



Astar Algorithm - myRobot/src/searchpractice/routeBot.java - Eclipse IDE

File Edit Source Refactor Navigate Search Project Run Window Help

routeBot.java

```
201     nextTile.f = nextTile.f;
202     nextTile.g = currentTile.g + MovementCost;
203     nextTile.parent = currentTile;
204     if(!inOpenSet)
205         openSet.add(nextTile);
206     }
207
208 }
209 public static Stack<Integer> AStar(Tile[][] battlefield, int startX, int startY, int endX, int endY) {
210     openSet = new PriorityQueue<Tile>((Tile t1, Tile t2) -> { if(t1.f < t2.f) return -1;
211     else if (t1.f > t2.f) return 1;
212     else return 0;});
213
214     openSet.add(new Tile(startX, startY));
215     Tile current;
216     while(true) {
217         current = openSet.poll();
218         if(current == null)
219         {
220             break;
221         }
222         current.closed = true;
223         if(current.x == endX && current.y == endY) {
224             break;
225         }
226         Tile next;
227         if(current.x - 1 >= 0) {
228             next = battlefield[current.x - 1][current.y];
229             updateCost(current, next);
230         }
231         if(current.x + 1 < battlefield.length) {
232
```

Problems Javadoc Declaration Console

No consoles to display at this time.

Writable Smart Insert 192 : 38 [4] Launching RouteFinder: (0%)

run() : vo  
generate  
setBlocke  
Moveme  
openSet :  
startX : in  
startY : in  
endX : in  
endY : in  
Manhatti

8:40 PM 4/24/2020