src/Tests/BenchMarkScript.java

```
  1 packageimportimportimportimportimportimportimportimportimportimportimport
importimportimportimportpublicclassBenchMarkScriptpublicstaticvoidmain
(String[] args)throwsFileWriterwriter=newFileWriter"BenchMarkOutput.txt"
PrintStreamoriginalOut=ByteArrayOutputStreambos=newByteArrayOutputStreamnew
PrintStreamnewArrayListnewArrayListnewArrayListnewArrayListnewArrayListnew
ArrayListnewArrayListnewArrayListnewArrayListnewArrayListnewArrayListnew
ArrayListRandomrand=newRandomintplanetSize=15intmemorySize=3forinti=0100int
goal_d=1intstart_d=1intgoal_a=845intstart_a=845Nodegoal=newNodenull0nullNode
start=newNodenull0longstartTime=longendTime=// startTime = System.nanoTime();
// PartC_IDS.iterativeDeepeningSearch(start, goal, planetSize);// endTime =
System.nanoTime();// idsTimes.add(endTime - startTime);//
idsCosts.add(parseCost(bos.toString()));// bos.reset();"BFS Times: ""\nBFS
Costs: ""\n""DFS Times: ""\nDFS Costs: ""\n""AStar Times: ""\nAStar Costs: "
"\n""BestF Times: ""\nBestF Costs: ""\n"// writer.write("IDS Times: " +
idsTimes + "\nIDS Costs: " + idsCosts + "\n");"SMAStar Times: ""\nIDS Costs: "
"\n"privatestaticdoubleparseCost(String output)Patternpattern="\\((\\d+:\\d+)\
\)\\n(\\d+\\.\\d+)"Matchermatcher=ifreturn2// Get the cost from the regex
groupreturn0.0// Return 0 or some error value if not found Tests;
  2
  3  General.Node;
  4  Algorithms.PartA_BFS;
  5  Algorithms.PartA_DFS;
  6  Algorithms.PartB_AStar;
  7  Algorithms.PartB_BestF;
  8  Algorithms.PartB_SMAStar;
  9  Algorithms.PartC_IDS;
 10
 11  java.io.ByteArrayOutputStream;
 12  java.io.FileWriter;
 13  java.io.IOException;
 14  java.io.PrintStream;
 15  java.util.ArrayList;
 16  java.util.List;
 17  java.util.Random;
 18  java.util.regex.Pattern;
 19
 20    {
 21
 22          IOException {
 23             ();
 24            System.out;
 25             ();
 26         System.setOut( (bos));
 27
 28         List<Long> bfsTimes =  <>();
 29         List<Double> bfsCosts =  <>();
 30
 31         List<Long> dfsTimes =  <>();
 32         List<Double> dfsCosts =  <>();
 33
 34         List<Long> astarTimes =  <>();
 35         List<Double> astarCosts =  <>();
 36
 37         List<Long> bestfTimes =  <>();
 38         List<Double> bestfCosts =  <>();
 39
 40         List<Long> smastarTimes =  <>();
 41         List<Double> smastarCosts =  <>();
```

```
42
43          List<Long> idsTimes =  <>();
44          List<Double> idsCosts =  <>();
45
46              ();
47             ;
48             planetSize * ;
49
50           (   ; i < ; i++) {
51                rand.nextInt(planetSize) + ;
52                rand.nextInt(planetSize) + ;
53                rand.nextInt() * ;
54                rand.nextInt() * ;
55
56                 (goal_d, goal_a, , , );
57                 (start_d, start_a, , , goal);
58
59                System.nanoTime();
60              PartA_BFS.bfs(start, goal, planetSize);
61                 System.nanoTime();
62              bfsTimes.add(endTime - startTime);
63              bfsCosts.add(parseCost(bos.toString()));
64              bos.reset();
65
66              startTime = System.nanoTime();
67              PartA_DFS.dfs(start, goal, planetSize);
68              endTime = System.nanoTime();
69              dfsTimes.add(endTime - startTime);
70              dfsCosts.add(parseCost(bos.toString()));
71              bos.reset();
72
73              startTime = System.nanoTime();
74              PartB_AStar.AStar(start, goal, planetSize);
75              endTime = System.nanoTime();
76              astarTimes.add(endTime - startTime);
77              astarCosts.add(parseCost(bos.toString()));
78              bos.reset();
79
80              startTime = System.nanoTime();
81              PartB_BestF.BestF(start, goal, planetSize);
82              endTime = System.nanoTime();
83              bestfTimes.add(endTime - startTime);
84              bestfCosts.add(parseCost(bos.toString()));
85              bos.reset();
86
87
88
89
90
91
92
93
94              startTime = System.nanoTime();
95              PartB_SMAStar.smaStar(start, goal, planetSize, memorySize);
96              endTime = System.nanoTime();
97              smastarTimes.add(endTime - startTime);
98              smastarCosts.add(parseCost(bos.toString()));
99              bos.reset();
100         }
101         System.setOut(originalOut);
102         writer.write( + bfsTimes +  + bfsCosts + );
```

```
103         writer.write( + dfsTimes +  + dfsCosts + );
104         writer.write( + astarTimes +  + astarCosts + );
105         writer.write( + bestfTimes +  + bestfCosts + );
106 
107         writer.write( + smastarTimes +  + smastarCosts + );
108         writer.close();
109     }
110 
111         {
112            Pattern.compile();
113         java.util.regex.   pattern.matcher(output);
114          (matcher.find()) {
115              Double.parseDouble(matcher.group());
116         }
117          ;
118     }
119 }
120 
```