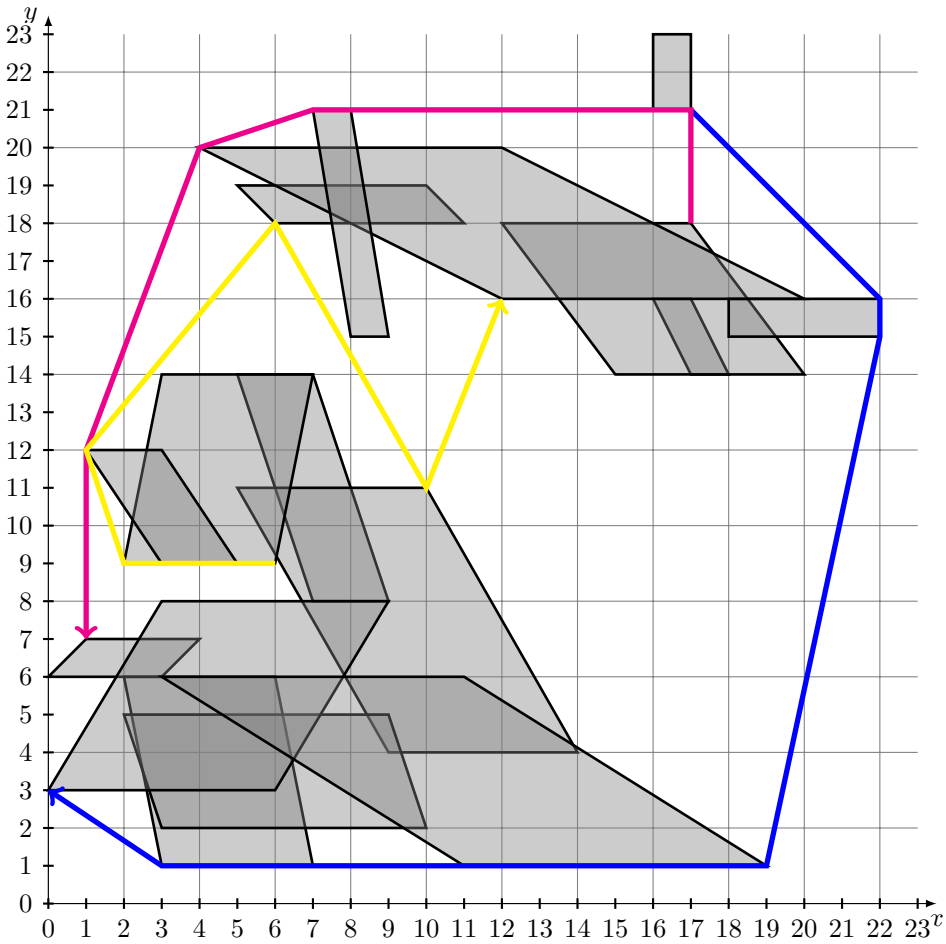
Navigating around Rhombuses

Consider an integer grid with coordinates (x, y) where $0 \leq x < 24$ and $0 \leq y < 24$ which is populated by 16 rhombuses as illustrated:



Now consider the problem of traveling from coordinate $(7, 21)$ to coordinate $(0, 3)$, subject to the constraints that the journey is made up of one or more straight-line segments, and the journey can change direction only at a vertex of a rhombus. Moreover, none of the line segments are permitted to overlap with the interior of any of the rhombuses. The interior of a rhombus are those points contained with the rhombus which are not on its perimeter. Thus it is permissible for a line segment to pass along the edge of a rhombus, or through a vertex of the rhombus, since both are contained within its perimeter.

The blue line segments illustrate one solution to the problem of navigating from $(17, 21)$ to $(0, 2)$. This solution can be described by the following sequence of vertices: $(17, 21)$, $(22, 16)$, $(22, 15)$, $(19, 1)$, $(3, 1)$, $(0, 3)$. The magenta line segments illustrate a solution for navigating from $(17, 18)$ to $(1, 7)$ and the yellow line the problem of travelling from $(6, 9)$ to $(12, 16)$.

Problem statement

You will receive an automatic e-mail which contains a pdf document, that is individual to you, which includes a table like that given below to the left:

rhombus	vertex_1	vertex_2	vertex_3	vertex_4
1	(16, 21)	(17, 21)	(17, 23)	(16, 23)
2	(7, 8)	(9, 8)	(7, 14)	(5, 14)
3	(9, 4)	(14, 4)	(10, 11)	(5, 11)
4	(3, 1)	(7, 1)	(6, 6)	(2, 6)
5	(2, 9)	(6, 9)	(7, 14)	(3, 14)
6	(6, 18)	(11, 18)	(10, 19)	(5, 19)
7	(3, 9)	(5, 9)	(3, 12)	(1, 12)
8	(17, 14)	(18, 14)	(17, 16)	(16, 16)
9	(0, 6)	(3, 6)	(4, 7)	(1, 7)
10	(15, 14)	(20, 14)	(17, 18)	(12, 18)
11	(18, 15)	(22, 15)	(22, 16)	(18, 16)
12	(12, 16)	(20, 16)	(12, 20)	(4, 20)
13	(3, 2)	(10, 2)	(9, 5)	(2, 5)
14	(8, 15)	(9, 15)	(8, 21)	(7, 21)
15	(0, 3)	(6, 3)	(9, 8)	(3, 8)
16	(11, 1)	(19, 1)	(11, 6)	(3, 6)

puzzle	start	finish
1	(8, 21)	(17, 14)
2	(5, 9)	(15, 14)
3	(3, 9)	(8, 15)
4	(3, 8)	(7, 14)
5	(22, 15)	(5, 9)
6	(17, 23)	(2, 9)
7	(17, 21)	(0, 3)
8	(17, 18)	(1, 7)
9	(12, 20)	(0, 6)
10	(6, 9)	(12, 16)
11	(22, 16)	(6, 9)
12	(20, 16)	(3, 9)

This table will give the vertices of your 16 rhombuses (actually the ones illustrated on the previous page). The document will also contain 12 navigation puzzles, as in the table given above to the right. Your mission, should you decide to accept it[1], is to generate solutions for each of the 12 problems. Each solution should be written to a separate output file and placed in the submission directory detailed below (not a subdirectory). For example, for problem 7, you will need to generate `7.txt` which contains either:

```
(17, 21) (22, 16) (22, 15) (19, 1) (3, 1) (0, 3)
```

or any other sequence of line segments which connects coordinate $(17, 21)$ to $(0, 3)$.

Note that a single space separates each coordinate and the coordinates are themselves formatted with a single space following the comma. Since part of the assessment will be automatically marked, it is vital that you conform to this format otherwise 0 marks will be awarded automatically. To de-risk formatting, you are advised to check each of your solutions using `FormatChecker.java` which is supplied on the course webpage. You should check your solutions before and after copying them to the submission directory (not a subdirectory). You are also advised to check your solutions using the diagram provided in the pdf document.

You are free to make use of `Vertex.java` which is available on the webpage. `Vertex` provides the function `linesIntersect(Vertex u1, Vertex u2, Vertex v1, Vertex v2)` for checking whether the line between two vertices `u1` and `u2` intersects with the line between `v1` and `v2`. The class also provides the function `vertexIntersect(Vertex u, Vertex v1, Vertex v2)` to check whether `u` occurs on the line between `v1` and `v2`. `Vertex` is specifically designed to support this assessment; you are advised to read and carefully consider the comments given in this source code.

Where to start

To help you a start, `JourneyPlanner.java` illustrates how iterative deepening can be applied to plan a journey between two locations. Observe how `List<String> nextConfigs(String state)` enumerates all the towns that are adjacent to given town `state`. You are free to build on and adapt this code. The challenge of the assessment is the design of `nextConfigs` method, which should have the signature:

[1] This is not meant to imply that the assignment is impossible, or that I will disavow any knowledge of it.

```
List<Vertex> nextConfigs(Vertex state)
```

You are advised to keep `nextConfigs` side-effect free so that it does no more than enumerate all the vertices which are connected with a single straight-line segment to the given state `state`. These adjacent states are those vertices which can be reached from `state` in a single line-segment without passing through the interior of any rhombus. A vertex of one rhombus can arise on the edge of another rhombus so your `nextConfigs` method will have to perform a careful case analysis (which is typical of planning problems). You should check these code very carefully before plugging it into an off-the-shelf search algorithm. Finally, there are a number of tactical errors that you should avoid on this assessment:

- Adding a déjà va (been here before) check to `nextConfigs` – the role of `nextConfigs` is merely to define the adjacent states;
- Deploying a complicated or space-hungry search algorithm without first implementing a simple one;
- Not devoting enough time to considering how a journey can navigate around the obstacles (and so end up with an `nextConfigs` method which is incomplete or buggy);
- Implement a search algorithm which has poor space behaviour such as Dijkstra's algorithm or breadth-first search; another mistake would be to implement A* search without a clear (documented) explanation as to why a heuristic is admissible.

Mark scheme

Twelve marks will be awarded for a correction solution to each problem and a further twelve marks will be awarded for optimal solutions, that is, solutions that minimise the number of line segments. In addition to placing the 12 solutions in the submission directory, you will also need to submit the source code used to generate your solutions; without providing the code itself no marks be will be awarded for the solutions themselves.

This code must be written in Java and should be clearly (but not overly) documented. All source files need to be marked with your login and surname. The code should include a `main()` method so that it can be executed without using BlueJ. Also, for testing, your code should only use language features supported in Java 1.8. The source code should be submitted as one or more .java files (not a .jar file, nor a .zip file, nor one or more .class files). Code that is clear, elegant and succinct will attract the highest marks. Minor changes to the code that significantly improve efficiency are encouraged; major changes that have a minor impact on efficiency are discouraged. Particular attention should be paid to your `nextConfigs` method. Twelve marks will be awarded for the code overall.

Deadline

You need to place your solutions in: `/proj/co528c/rhombuses/`xyz on raptor where xyz is your own personal login. Permissions have been set so that only xyz can access files in the directory xyz. The deadline for submission is *23.55* on the Tuesday of week 20 (March 3rd). Section 2.2.1.1 of Annex 9 of the Credit Framework states that, "Academic staff may not accept coursework submitted after the applicable deadline except in concessionary circumstances".

A Frequently Asked Questions document on Plagiarism and Collaboration is available at: `www.cs.kent.ac.uk/teaching/student/assessment/plagiarism.local`. The work you submit must be your own, except where its original author is clearly referenced. Checks will be run on submitted work in an effort to identify possible plagiarism, and take disciplinary action against anyone found to have committed plagiarism. When you use other peoples' material, you must clearly indicate the source of the material.