Experiment Setup & Assumptions:

The experiment is a Java application using a pre-define set of coordinates for the vertices of the polygons. The vertices are defined based on the assignment and are as follows:

- Polygon 1: ((220, 616), (220, 666), (251, 670), (272, 647))
- Polygon 2: ((341, 655), (359, 667), (374, 651), (366, 577))
- Polygon 3: ((311, 530), (311, 559), (339, 578), (361, 560), (361, 528), (336, 516))
- Polygon 4: ((105, 628), (151, 670), (180, 629), (156, 577), (113, 587))
- Polygon 5: ((118, 517), (245, 517), (245, 577), (118, 557))
- Polygon 6: ((280, 583), (333, 583), (333, 665), (280, 665))
- Polygon 7: ((252, 594), (290, 562), (264, 538))
- Polygon 8: ((198, 635), (217, 574), (182, 574))

The start and end points are also pre-defined and are (115, 655) and (380, 560), respectively. The robot uses a modified A* algorithm to find the shortest point from the start to the end positions, with the assumption that it always starts at the start point. The robot is not allowed to cross through any polygons, and uses a provided algorithm to check for intersections. Barring intersections, the robot is allowed to move from any two points on the (x,y) plane. Finally, the robot is assumed to have free movement along the (x,y) plane instead of discrete movement.

Data Structures & Algorithms

The program consists of a 3 classes: Vector2, Node, and MainClass. Vector2 is a simple class used to represent the x,y cords of the experiment. Node is used to represent individual points in the experiment. This includes all vertices of the polygons, and the start and end points. Nodes consist of the following fields:

- a Vector2 object (to show its coordinates)
- an ArrayList of other Node objects to represent nodes this node connects to (edges that form a polygon)
- A node id
- A polygon id, used to determine which polygon it is in. Note that nodes with a polygon of id = -1 means that they do not belong to a polygon (this is used to identify the start and end points)
- The f(h) of this node, as determined by the A* algorithm
- The g(h) of this node, as determined by the A* algorithm
- A single Node object called "parent", used for the final pathing so that nodes along the correct "path" can properly chain to one another. The parent node is determined by the A* algorithm

The Node class also implements the Comparable Java interface; this is because it allows the Node class to be sorted. The compareTo() function for the node compares the f(h) values for the nodes in question, and determines which value is greater. The Node class also features a function to calculate the distance between two nodes. The Euclidean distance is used for calculations vs. Manhattan distance because of the movement assumptions mentioned earlier (free movement instead of discrete).

The MainClass contains the main function and is also where many of the algorithm functions reside. The A* function is explained in the homework submission, but for the purposes of this application it is used to return the solution for the path the robot will take, and returns the last node of the robot's path. From there the application uses the last node and its parent nodes to print the path of the robot (automatically formatting the order such that the final list doesn't flip backwards). The A* function makes use of 2 subfunctions:

- checkIntersect() takes in 4 Vector2 coordinates, and creates two line segments based on them (first two coordinates make the first segment, and last two make the second segment). It then checks if any intersection exists between those two segments using the provided algorithm for the assignment. It returns a Boolean whether the intersection exists or not (true and false, respectively)
- canGoTo() takes in a start Node and an end Node as parameters, and sees if the robot can make a direct path form start to end. It uses the above checkIntersect() function to see if all line segments created by the various polygon edges interfere with the path from the provided start to end. Since it is assumed the robot can move from any two points, all edges must be check to see if any edges can interfere with the robot pathing, meaning this function must always iterate through all existing nodes. The following pseudocode outlines how this function works

```
function canGoTo(Node start, Node end)
  for every currentNode in listOfAllNodes
    for every neighbor in currentNodeListOfNeighbors
      if checkInterect(currentNode.x, currentNode.y, neighbor.x, neighbor.y) == true
        return false // intersection detected
  return true // no intersections detected
```

With these subfuctions, the A* function can determine the shortest path of the given vertices. The main functions sets up all the necessary vertices, calls the A* function, and then subsequently prints out the resulting path.