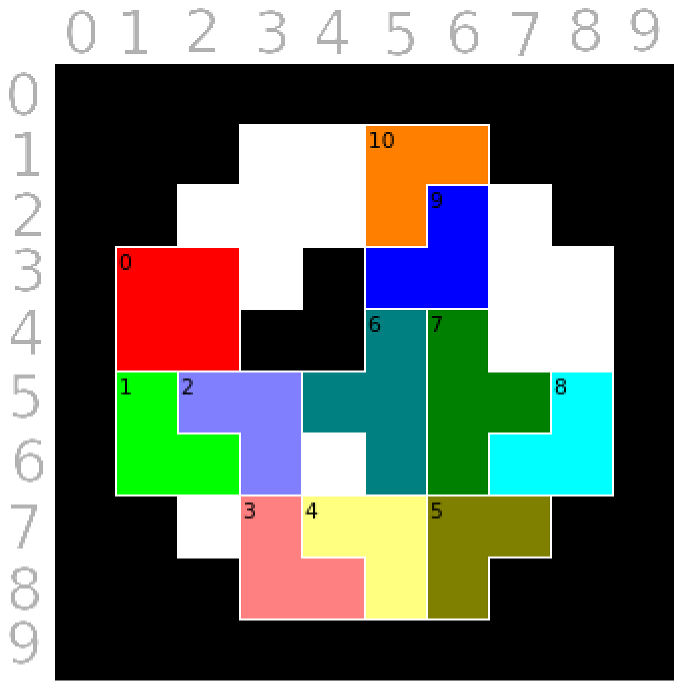
Breadth-first search

1. **Download this Java starter kit**. Unzip it. Examine it. Build it.

2. **Consider this puzzle**, which I adapted from one of the Professor Layton games:



   The goal is to move the red piece (0) to a position adjacent to the top. That is, move it to where the orange piece (10) is currently located. Black pieces cannot be moved. The pieces cannot overlap. The pieces cannot be rotated. Think about how you would implement this game.

3. **Implement some method to detect invalid states.** A state is invalid if any two pieces overlap, or if any piece overlaps with a black square. (Note that there is an FAQ below that may provide some helpful tips.)

4. **Implement breadth first search.** If needed, use your favorite search engine to find some pseudocode for breadth first search. Then, implement it. Please be careful to keep your general-purpose code separated from your problem-specific code. A good implementation of breadth first search should not need to know anything about the problem it operates on. This will make it easier for you to use this algorithm with other problems.

   If you want to get a jump on the next assignment, you can implement uniform cost search instead. Uniform cost search is a generalization of breadth first search, so it can do breadth first search.

5. **Use your implementation of breadth first search to solve this puzzle.** Print the solution (from beginning to end) in a format that expresses the (x,y) positions of all 11 pieces relative to their starting positions. Here is an example of correct output (for an incorrect solution):

```
(0,0) (0,0) (0,0) (0,0) (0,0) (0,0) (0,0) (0,0) (0,0) (0,0) (0,0)
(1,0) (0,0) (0,0) (0,0) (0,0) (0,0) (0,0) (0,0) (0,0) (0,0) (0,0)
(1,0) (0,0) (0,0) (0,0) (0,0) (0,0) (0,0) (0,0) (0,0) (0,1) (0,0)
(1,0) (0,0) (0,0) (0,0) (0,0) (0,0) (0,0) (0,0) (0,-1) (0,1) (0,0)
(1,0) (0,0) (0,0) (0,0) (0,0) (0,0) (0,-1) (0,0) (0,-1) (0,1) (0,0)
(1,0) (0,0) (0,0) (0,0) (0,0) (0,0) (0,0) (0,0) (0,-1) (0,1) (0,0)
(1,0) (0,0) (0,0) (0,0) (0,0) (0,0) (0,-1) (0,0) (0,-1) (0,1) (0,0)
(1,0) (0,0) (0,0) (0,0) (0,0) (0,0) (0,0) (0,0) (0,-1) (0,1) (0,0)
(1,0) (0,0) (0,0) (0,0) (0,0) (0,0) (0,1) (0,0) (0,-1) (0,1) (0,0)
(1,0) (0,0) (0,0) (-1,0) (0,0) (0,0) (0,1) (0,0) (0,-1) (0,1) (0,0)
(1,0) (0,0) (0,0) (-1,0) (0,-1) (0,0) (0,1) (0,0) (0,-1) (0,1) (0,0)
(1,0) (0,0) (0,0) (-1,0) (0,-1) (0,0) (0,1) (0,0) (0,-1) (0,1) (0,-1)
(1,0) (0,0) (0,0) (-1,0) (0,-1) (0,0) (0,1) (0,0) (0,-1) (-1,1) (0,-1)
```

Save the output to a file named "results.txt".

6. zip up your source code. Example:

```
zip -r -9 submission1.zip src
```

(You can also use tarballs if you prefer. RAR is not supported because no open source implementation is available.) Make sure your archive contains a script named build.bash (or build.bat, if you use Windows) that builds (but does not execute) your code. Make sure any filenames in your code use relative paths. (I do not have a folder named "C:\\Users\\Your Name\\Desktop\\School\\Artificial Intelligence\\Project 1" on my server, and even if I did, it would not contain the files that yours does.) Your archive should contain your source code **and the results.txt file** (so the grader does not have to run your code), but it should not contain any binaries. (My server will reject your submission if it includes .class files.) Click on the "Project submission" link on the main class page to submit your archive file. If you have any difficulties with the submission server, please e-mail your submission directly to the grader.

FAQ:

1. **Q: How should I encode state in this game?**
A: This game has 11 movable pieces. I recommend that you encode the state of this game with an array of 22 bytes (expressing the horizontal and vertical position of each piece). In Java, you can allocate an array of 22 bytes like this:

```
byte[] state = new byte[22];
```

I also recommend that you consider the array of 22 zeros to represent the initial state of this game. (In java, arrays are initialized with zeros when you allocate them, but since some other languages do not do this, it is a good habit to explicitly initialize values.)

(If you enjoy programming challenges, it is possible to encode the state of this game into just 6 bytes.)

If you think you need to encode concepts in your state like the color of each piece, the number of squares in each piece, or the relative locations of each square in the piece, you are confused about what state is. This game has millions of possible states. Your project will autonomously explore the space of these states to find a path to the goal. In doing so, your computer's memory will be filled with hundreds of thousands of state objects. If each state object includes a bunch of redundant information that never changes while the game is played, then you are being wasteful with memory, and you might run out of it.

2. **Q: How does one detect invalid states?**
   A: One good way is to "draw" the board. If you "draw" the whole thing, and never "draw" in the same place twice, it is a valid state. If you "draw" somewhere twice, you found an invalid state. (Except don't actually draw using a graphics object. That would be ridiculously inefficient. Just use a 10-by-10 array of booleans, or something like that.)

3. **Q: How big is the final solution?**
   A: The correct answer requires 114 moves. So, your output should report a path of 115 states through which a player must pass in order to optimally reach the goal.

4. **Q: I found pseudocode for breadth first search with trees. I cannot find anything for puzzle games. What should I do?**
   A: Puzzle games are actually a lot like trees. The starting state is the root, and the actions you can perform are the branches.

5. **Q: My pseudocode says I need a Queue or Set, but there is no Queue or Set class in Java. Those are just interfaces. What should I do?**
   A: I like to use TreeSet or TreeMap. In addition to working like a set, TreeSet keeps its contents sorted, so you can use it as a priority queue. (In fact, TreeSet can work as a double-ended priority queue. Even the PriorityQueue class does not provide double-ended functionality. Another little-know property of sorted trees is that it is possible to reference their contents by index in log(n) time. ...but I digress.)

6. **Q: How does one compare state objects in Java?**
   A: Never use Comparable. Use Comparator instead. (Comparable lets the object determine how comparisons are performed. That is an awkward design because objects should not need to know how or why they are being stored. Comparator lets you sort any way you like, without giving control to the object.) Here is a Java example that uses a TreeSet with a Comparator to keep track of a bunch of GameState objects:

```
import java.util.Comparator;
import java.util.TreeSet;

class GameState
{
	GameState prev;
	byte[] state;

	GameState(GameState _prev)
	{
		prev = _prev;
		state = new byte[22];
	}
}

class StateComparator implements Comparator<GameState>
{
	public int compare(GameState a, GameState b)
	{
		for(int i = 0; i < 22; i++)
		{
			if(a.state[i] < b.state[i])
				return -1;
			else if(a.state[i] > b.state[i])
				return 1;
		}
		return 0;
	}
}

class Main
{
	public static void main(String args[])
	{
		StateComparator comp = new StateComparator();
		TreeSet<GameState> set = new TreeSet<GameState>(comp);
		GameState a = new GameState(null);
		a.state[21] = 7;
		GameState b = new GameState(null);
		b.state[14] = 3;
		GameState c = new GameState(null);
		c.state[21] = 7;
		if(!set.contains(a))
			System.out.println("Passed 1");
		else
			System.out.println("oops 1");
		set.add(a);
		if(set.contains(a))
			System.out.println("Passed 2");
		else
			System.out.println("oops 2");
		if(!set.contains(b))
			System.out.println("Passed 3");
		else
			System.out.println("oops 3");
		if(set.contains(c))
			System.out.println("Passed 4");
		else
			System.out.println("oops 4");
	}
}
```

7. **Q: Wouldn't HashSet be a little bit faster than TreeSet? Can I use HashSet instead of TreeSet?**
   Yes. I do not care what data structures you use, as long they works (and do not bypass your learning experience, of course).

8. **Q: How do I know if my solution is correct?**
   A: Here is a little program that graphically visualizes this game. (Click the mouse to watch it move the pieces in illegal ways.) With very little work, you could hook this code up to your game to visualize your solution. Then, watch it solve the puzzle. If it only performs valid moves, and it solves it in the right number of moves, your solution is correct. This code also contains some other parts that may be useful in this assignment, so I think it would be worthwhile for you to play with it. You are welcome to draw from this code in formulating your solution.

```
import javax.swing.JFrame;
import java.awt.event.ActionListener;
import java.awt.event.ActionEvent;
import javax.swing.Timer;
import javax.swing.JPanel;
import java.awt.Graphics;
import java.awt.Image;
import javax.imageio.ImageIO;
import java.io.IOException;
import java.awt.Graphics;
import java.io.File;
import java.awt.event.MouseListener;
import java.awt.event.MouseEvent;
import java.util.Random;
import java.awt.Color;

class View extends JPanel implements MouseListener {
        Viz viz;
        Random rand;
        byte[] state;
        Graphics graphics;
        int size;

        View(Viz v) throws IOException
        {
                viz = v;
                rand = new Random(0);
                state = new byte[22];
                size = 48;
        }

        public void mousePressed(MouseEvent e)
        {
                state[rand.nextInt(22)] += (rand.nextInt(2) == 0 ? -1 : 1);

                for(int i = 0; i < 11; i++)
                System.out.print("(" + state[2 * i] + "," +
                        state[2 * i + 1] + ") ");
                System.out.println();
                viz.repaint();
        }

        public void mouseReleased(MouseEvent e) {    }
        public void mouseEntered(MouseEvent e) {    }
        public void mouseExited(MouseEvent e) {    }
        public void mouseClicked(MouseEvent e) {    }

        // Draw a block
        public void b(int x, int y)
        {
                graphics.fillRect(size * x, size * y, size, size);
        }
```

```
		// Draw a 3-block piece
		public void shape(int id, int red, int green, int blue,
			int x1, int y1, int x2, int y2, int x3, int y3)
		{
			graphics.setColor(new Color(red, green, blue));
			b(state[2 * id] + x1, state[2 * id + 1] + y1);
			b(state[2 * id] + x2, state[2 * id + 1] + y2);
			b(state[2 * id] + x3, state[2 * id + 1] + y3);
		}

		// Draw a 4-block piece
		public void shape(int id, int red, int green, int blue,
			int x1, int y1, int x2, int y2,
			int x3, int y3, int x4, int y4)
		{
			shape(id, red, green, blue, x1, y1, x2, y2, x3, y3);
			b(state[2 * id] + x4, state[2 * id + 1] + y4);
		}

		public void paintComponent(Graphics g)
		{
			// Draw the black squares
			graphics = g;
			g.setColor(new Color(0, 0, 0));
			for(int i = 0; i < 10; i++) { b(i, 0); b(i, 9); }
			for(int i = 1; i < 9; i++) { b(0, i); b(9, i); }
			b(1, 1); b(1, 2); b(2, 1);
			b(7, 1); b(8, 1); b(8, 2);
			b(1, 7); b(1, 8); b(2, 8);
			b(8, 7); b(7, 8); b(8, 8);
			b(3, 4); b(4, 4); b(4, 3);

			// Draw the pieces
			shape(0, 255, 0, 0, 1, 3, 2, 3, 1, 4, 2, 4);
			shape(1, 0, 255, 0, 1, 5, 1, 6, 2, 6);
			shape(2, 128, 128, 255, 2, 5, 3, 5, 3, 6);
			shape(3, 255, 128, 128, 3, 7, 3, 8, 4, 8);
			shape(4, 255, 255, 128, 4, 7, 5, 7, 5, 8);
			shape(5, 128, 128, 0, 6, 7, 7, 7, 6, 8);
			shape(6, 0, 128, 128, 5, 4, 5, 5, 5, 6, 4, 5);
			shape(7, 0, 128, 0, 6, 4, 6, 5, 6, 6, 7, 5);
			shape(8, 0, 255, 255, 8, 5, 8, 6, 7, 6);
			shape(9, 0, 0, 255, 6, 2, 6, 3, 5, 3);
			shape(10, 255, 128, 0, 5, 1, 6, 1, 5, 2);
		}
	}

	public class Viz extends JFrame
	{
		public Viz() throws Exception
		{
			View view = new View(this);
			view.addMouseListener(view);
			this.setTitle("Puzzle");
			this.setSize(482, 505);
			this.getContentPane().add(view);
			this.setDefaultCloseOperation(JFrame.EXIT_ON_CLOSE);
			this.setVisible(true);
		}

		public static void main(String[] args) throws Exception
		{
			new Viz();
```

```
        }
}
```

9. **Q: What common mistakes do you see with this assignment?**
   A: Common mistakes include...
   - Adding values that never change to the encoding of state. There is no reason to make a million copies of values that remain constant.
   - Using an array or linked list to keep track of visited states. Those data structures are too inefficient for that purpose.
   - Assuming the set works without testing it. If you did not test it, you do not know if it works.
   - Using shallow copy where a deep copy is needed. In Java, the "=" operator only copies a reference. It does not deep-copy the object. If values seem to be mysteriously changing, this is probably your problem.

10. **Q: How would one debug this project?**
    A: Simplify the problem. For example, change the goal to something trivial, like "move piece 10 one step to the left". That would make this puzzle really easy to solve. You could do it in just one action. Use the visualization code provided above to see exactly what your algorithm is doing. If your algorithm solves that easy challenge correctly, try a two-step challenge, like "move piece 9 one step up". (This is a two-step challenge because one cannot move piece 9 up until after one first moves piece 10 to the left.) After you debug that one, I'll bet the whole thing will probably start working. But if not, try a three-step challenge.

11. **Q: Could you post some debug spew for us to compare against?**
    A: Okay. Here it is. Here are instructions for using it:
    - Whenever you push a state into your queue, print "Push: " + stateToString(state).
    - Whenever you pop a state from your queue, print "\nPop: " + stateToString(state).
    - Run your program and pipe it to a file. (You can break after a few seconds of execution.)
    - Diff your file with this one.
    - If necessary, adjust your code so that yours pushes the candidate neighbors in the same order as this one. (0-left, 0-right, 0-up, 0-down, 1-left, 1-right, 1-up, 1-down, 2-left, ...)
    - Find the first difference. (If the difference involves the order in which neighbors are pushed, go to the previous step.)
    - Determine why your code gives different results.

    For convenience, here is a little code to convert a state to a string:

```
static String stateToString(byte[] b)
{
        StringBuilder sb = new StringBuilder();
        sb.append(Byte.toString(b[0]));
        for(int i = 1; i < b.length; i++) {
                sb.append(",");
                sb.append(Byte.toString(b[i]));
        }
        return sb.toString();
}
```