Smith College

Computer Science

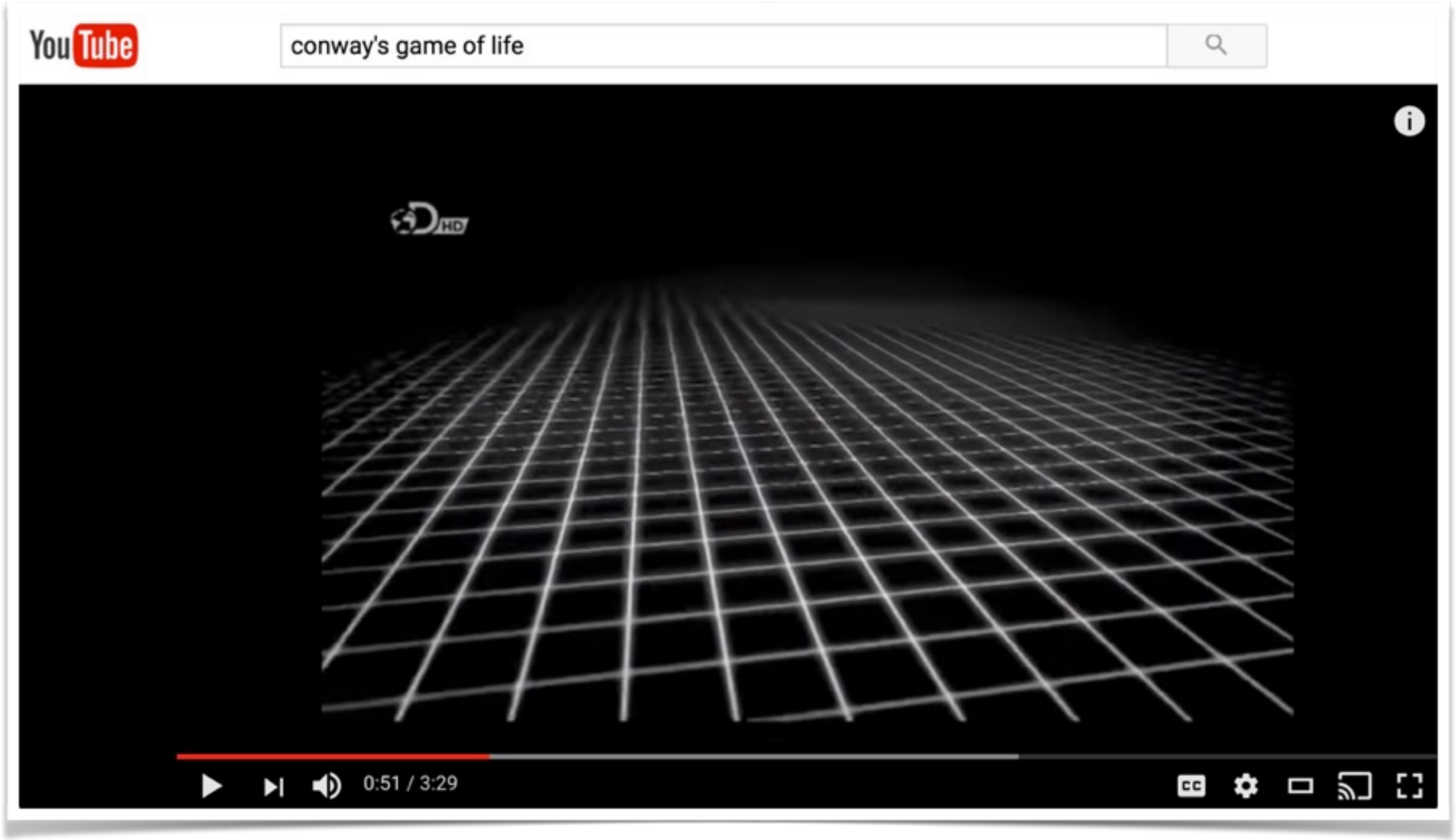
CSC352

Week #5 — Spring 2017

Dominique Thiébaut
dthiebaut@smith.edu

Programming Lab

Making the Game of Life Parallel



https://www.youtube.com/watch?v=CgOcEZinQ2I

Serial Version

- Study it
- Run it on your laptop
- Use both dish and dish2 as the array of live cells, and see how they evolve

login to your **352b** account

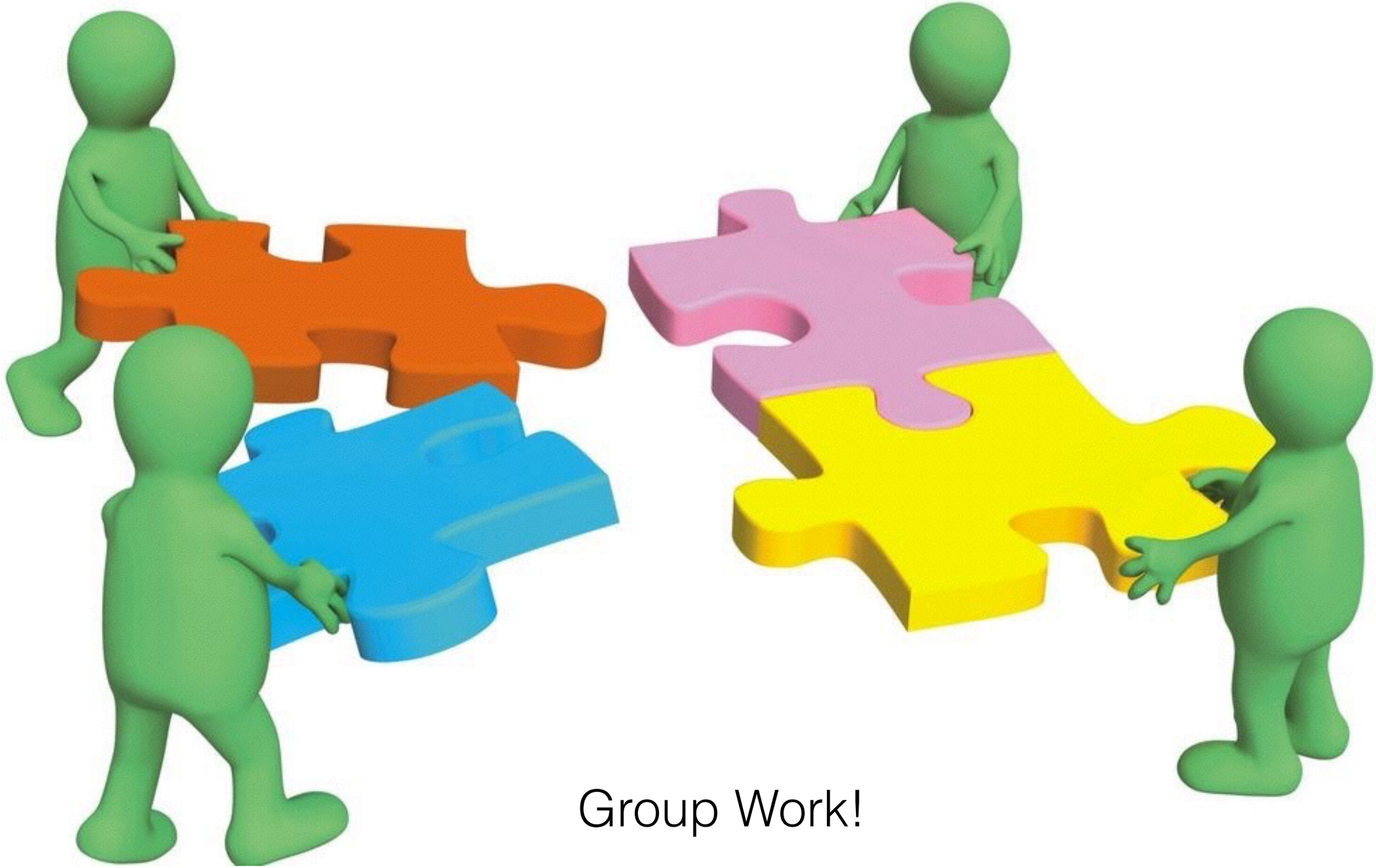
```
getCopy GameOfLife.java
javac GameOfLife.java
java GameOfLife
```

Other option:

http://cs.smith.edu/dftwiki/index.php/CSC352_Game_of_Life_Lab_2017

2-Thread Version

- As a group, discuss the different tissues associated with parallelizing the Game of Life and running it with two threads.
- List all the issues that must be addressed on the whiteboard
- How will you verify the correctness of the parallel version?
- Play-out (human play) the execution of the 2-thread program: two people or two groups play the roles of the two threads.



Group Work!

Image taken from: http://www.brocku.ca/blogs/futurestudents/files/2014/10/puzzle-work.jpg

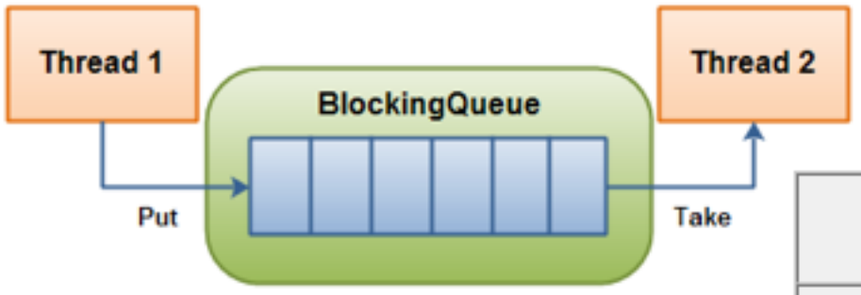
Could be Usefull…

- **What is a BlockingQueue?**
  BlockingQueue is a queue which is **thread safe** to insert or retrieve elements from it. Also, it provides a mechanism which blocks requests for inserting new elements when the queue is full or requests for removing elements when the queue is empty, with the additional option to stop waiting when a specific timeout passes. This functionality makes *BlockingQueue* a nice way of implementing the Producer-Consumer pattern, as the producing thread can insert elements until the upper limit of *BlockingQueue* while the consuming thread can retrieve elements until the lower limit is reached and of course with the support of the aforementioned blocking functionality.

https://examples.javacodegeeks.com/core-java/util/concurrent/java-blockingqueue-example/

Thread safe: Implementation is guaranteed to be free of race conditions when accessed by multiple threads simultaneously.

Using a BlockingQueue



	Throws Exception	Special Value	Blocks	Times Out
Insert	`add(o)`	`offer(o)`	`put(o)`	`offer(o, timeout, timeunit)`
Remove	`remove(o)`	`poll()`	`take()`	`poll(timeout, timeunit)`
Examine	`element()`	`peek()`		

"A Queue that additionally supports operations that ***wait*** for the queue to become non-empty when retrieving an element, and ***wait*** for space to become available in the queue when storing an element"

- BlockingQueue is an *interface* in *java.util.concurrent*
- Need to use an implementation of it:
  - **ArrayBlockingQueue**
  - DelayQueue
  - LinkedBlockingQueue
  - PriorityBlockingQueue
  - SynchronousQueue

Image & table from: http://tutorials.jenkov.com/java-util-concurrent/blockingqueue.html

Example

```java
import java.util.concurrent.ArrayBlockingQueue;
import java.util.concurrent.BlockingQueue;


public class UsingQueues {

        public static void main(String[] args) throws InterruptedException {
                BlockingQueue<Integer> toWorkerQ = new ArrayBlockingQueue<Integer>(2);
        BlockingQueue<Integer> fromWorkerQ = new ArrayBlockingQueue<Integer>(2);

        // create a worker and give it the two queues
        DemoThread t=new DemoThread( fromWorkerQ, toWorkerQ );

        // start thread
        t.start();

        // wait 1/2 second
        try {
                Thread.sleep( 500 );
            } catch (InterruptedException e) {
                e.printStackTrace();
            }

        // send work to worker
        toWorkerQ.put( 100 );

        // wait for answer back from worker
        int x = fromWorkerQ.take();

        // display the result
        System.out.println( "x = " + x );

        }

}
```

Code available here: http://cs.smith.edu/dftwiki/index.php/CSC352:_Using_BlockingQueues

Example (cont'd)

```
/**
 * DemoThread
 */
class DemoThread extends Thread {
        BlockingQueue<Integer> sendQ;
        BlockingQueue<Integer> receiveQ;

        DemoThread( BlockingQueue<Integer> sendQ,
                                BlockingQueue<Integer> receiveQ ) {
                this.sendQ = sendQ;
                this.receiveQ = receiveQ;
        }

        public void run(){
                int x=0;

                // block until there's something in the queue
                try {
                        x = receiveQ.take( );
                } catch (InterruptedException e1) {
                        e1.printStackTrace();
                }

                // do some computation
                x = x*2;

                // send results back
                try {
                        sendQ.put( x );
                } catch (InterruptedException e) {
                        e.printStackTrace();
                }
        }
}
```

Implement the 2-Thread Game of Life in Java