ITI 1121. Introduction to Computing II
Winter 2019

Assignment 2
(Last modified on February 4, 2019)

Deadline: February 25, 2019, 11:30 pm

Learning objectives

- **Implementing** a queue-based algorithm
- **Utilizing** arrays to store information
- **Experimenting** with deep copies
- **Describing** state space search algorithms

Introduction

For this assignment, we are going to look at the game *Lights Out*. This is a single-player game, played on an n by m rectangular board. Initially, the board is entirely "off". To play one step, the player selects one position on the board. When selected, the location and its (up to) four neighbours (up, down, left and right) switch value. If a location was "on", then it is now "off", and if it was "off", it is now "on". Figure 1 shows the different cases: one the left, the selected position is inside the board. In this case, that position and all four of its neighbours are switched to their respective opposite values. In the centre, the selected location is on the corner of the board, then it has only two neighbours, and the location and its two neighbours are switched to their respective opposite values. Finally, on the right, the selected location is along the border of the board, it has three neighbours, and the position and its three neighbours are switched to their respective opposite values.

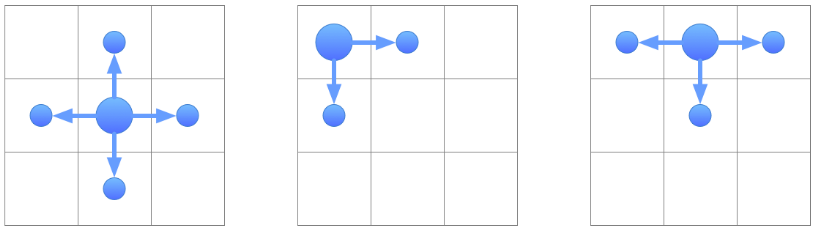

Figure 1: Selecting one position switches that location and its neighbours to the south, north, east and west. Depending on the location of the position, the total number of switched positions is five (left, the selected location is inside the board), four (right, the selected location is on the border of the board) or three (middle, the selected location is on the corner of the board).

The overall goal of the game is to go from a board that is entirely "off" to a board that is entirely "on". A secondary goal is to do this using the fewer possible number of steps. There is no limitation regarding how often a given position can be selected nor in which order positions are selected.

In Figure 2, we illustrate a successful game of *Lights Out* on a 2x2 board. This particular game solves the problem in four steps, by selecting each of the four locations on the board one after the other, clockwise starting from the top left (shown by a red star).

The minimum number of steps required to solve the game is not a simple function of the size of the board. For example, consider a 3x2 board, as shown in Figure 3: although this board is larger than the 2x2 board of Figure 2, it can be solved in only two steps, whereas the 2x2 board actually requires at least four steps to solve. Figure 3 illustrates a two-step solution.

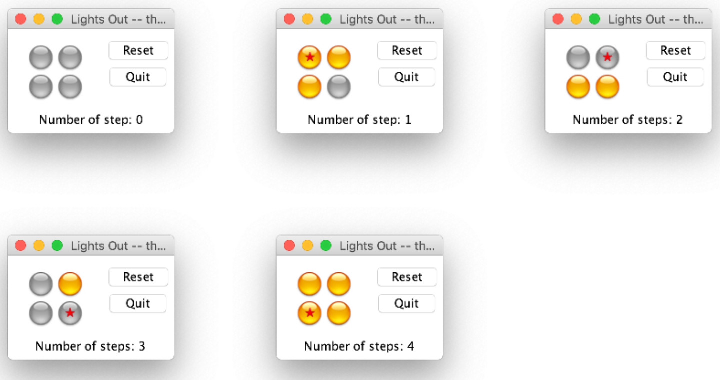

Figure 2: A successful completion of a *Lights Out* game on a 2x2 board, in four steps. The red star identifies the selected location at each step.

Another important point is the number of different solutions for a given size of board. Of course, this needs some definitions, since you can clearly extend any solution indefinitely by going back and forth between the same general board positions. We will clarify this momentarily, but for now let's simply use an intuitive definition for which a solution does not include unnecessary steps. For a board the board of size 3x2, in addition to the solution depicted by the Figure 3, we can find at least three other distinct solutions, as illustrated Figure 4. Therefore, such a 3x2 board has at least four different solutions. On a board of size 2x2, on the other hand, we can only find one such a solution, the one we have seen Figure 2.

We now need to make two simple remarks regarding this game. These remarks will help us define and eventually finding our solutions.

Remark 1 *In a solution of the game* Lights Out, *it is never useful to select the same location twice in a row, as it simply puts back the board in the same state. If a solution selects the same location twice in a row, then a shorter, equivalent solution can be built by skipping these two selections.*

Remark 1 is rather obvious, and proof is left as an exercise. It means that if you consider the following sequence on a 2x2 board: (1) Top Left (2) Bottom Right (3) Bottom Right (4) Top Right (5) Bottom Right (6) Bottom Left (which indeed reaches a full board), steps 2 and 3 cancel each other and a simpler equivalent sequence is (1) Top Left (2) Top Right (3) Bottom Right (4) Bottom Left.

The second remark is also rather simple, and the proof is also left as an exercise:

Remark 2 *In any solution of the game* Lights Out, *any pair of consecutive steps can be exchanged without changing the resulting final board.*

What Remark 2 says is that if you have a solution involving k steps, then for all $i < k$ you can simply exchange step i and step $i + 1$ without changing the configuration of the board at the end. We can illustrate this on our solution for the 2x2 board. Our initial solution is:

(1) Top Left (2) Top Right (3) Bottom Right (4) Bottom Left

We can for example exchange steps (2) and (3) and the solution will still work:

(1) Top Left (2) Bottom Right (3) Top Right (4) Bottom Left

Or we can exchange step (1) and (2):

(1) Top Right (2) Top Left (3) Bottom Right (4) Bottom Left

Since exchanging two consecutive steps does not change the solution, we can continue exchanging two more steps, for example (3) and (4):

(1) Top Right (2) Top Left (3) Bottom Left (4) Bottom Right

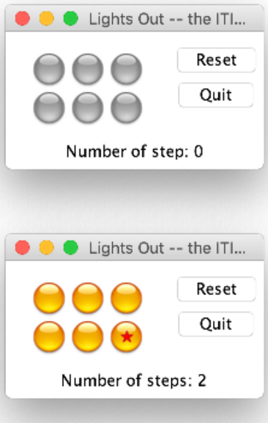

Figure 3: A successful completion of a *Lights Out* game on a 3x2 board, in two steps. The red star identifies the selected location at each step.

and again, for example (2) and (3) again:

(1) Top Right (2) Bottom Left (3) Top Left (4) Bottom Right

etc. In general, given a solution, any permutation of the steps produces and equivalent solution.

We can now combine Remarks 1 and 2 to observe that if the same position is used twice in a solution, then using Remark 2 we know that an equivalent solution can be constructed in which these two steps are consecutive, and then using Remarks 1 we can remove these two steps and obtain a shorter, equivalent solution. If we generalize that, in any solution, if a position is selected an even number of times, then an equivalent, simpler solution can be constructed by not selecting that position at all, and if a position is selected an odd number of times, then an equivalent, simpler solution can be constructed by selecting that position only once overall. Therefore, a (reduced) solution uses each board position either zero or one time. Moreover, the order in which the positions used are selected is not important, which brings us to our definition of a solution:

Definition 1 *A* Solution *for of the game* Lights Out *is a set of board positions.*

Given that a solution is simply an unordered list of board positions, in which each board position appears at most one, it will be simpler for us to represent a solution as a matrix of booleans. The matrix will be of the same size as the board, and an entry of the matrix will be *true* if that position is used in the solution, *false* otherwise.

For example, the single solution that we know for the 2x2 board is the following:

true	true
true	true

While our four solutions for the 3x2 board are the following:

true	true	false
true	true	false

true	false	false
false	false	true

false	true	true
false	true	true

false	false	true
true	false	false

Finally, the last bit of "theory" we are going to need about the game is a good way to decide if a "solution" is indeed working, that is, if it will bring a board that is entirely "off" to a board that is entirely "on". For this, we only need to remark that each time a board position or one of its neighbours is selected, that board position will switch state. Starting from an "off" state, a board position ends with in an "on" state if and only if it is switched an odd number of times. Therefore, to figure out the end state of a board position for a given solution, it is enough to check how many times that position or one of its neighbours is selected in the solution: if that number is odd, then the position will end up in the state "on", and if that number is even, it will end up in the state "off". Since we want

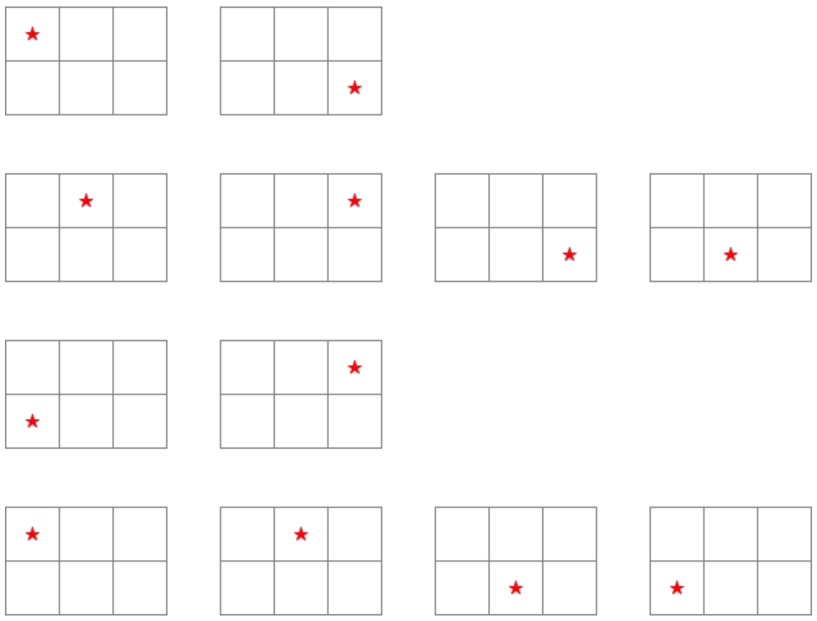

Figure 4: Each row shows a different solution of the game *Lights Out* on a 3x2 board, two in two steps and two in four steps.

the entire board to be "on" at the end, a solution will work if that property holds for all the positions, that is, for each board position, that position or one of its neighbours is selected an odd number of times.

Let's illustrate that on our 2x2 solution:

true	true
true	true

We have:

- Top Left and its neighbours: selected 3 times
- Top Right and its neighbours: selected 3 times
- Bottom Left and its neighbours: selected 3 times
- Bottom Right and its neighbours: selected 3 times

and thus all positions have an odd number of selections. This solution works.

On this 3x2 solution:

true	true	false
true	true	false

We have:

- Position(1,1) and its neighbours: selected 3 times
- Position(2,1) and its neighbours: selected 3 times
- Position(3,1) and its neighbours: selected 1 time
- Position(1,2) and its neighbours: selected 3 times
- Position(2,2) and its neighbours: selected 3 times
- Position(3,2) and its neighbours: selected 1 time

and thus all positions have an odd number of selections. This solution works.

Finally, on this 3x2 solution:

true	false	false
false	false	true

We have:

- Position(1,1) and its neighbours: selected 1 time
- Position(2,1) and its neighbours: selected 1 time
- Position(3,1) and its neighbours: selected 1 time
- Position(1,2) and its neighbours: selected 1 time
- Position(2,2) and its neighbours: selected 1 time
- Position(3,2) and its neighbours: selected 1 time

and thus all positions have an odd number of selections. This solution works.

However, the following sequence:

true	false	true
false	true	false

is not a working solution, because:

- Position(1,1) and its neighbours: selected 1 time
- Position(2,1) and its neighbours: selected 3 times
- Position(3,1) and its neighbours: selected 1 time
- Position(1,2) and its neighbours: selected 2 times
- Position(2,2) and its neighbours: selected 1 time
- Position(3,2) and its neighbours: selected 2 times

and thus both positions (1,2) and (3,2) will end up in the "off" state.

1 First Implementation [70 Marks]

We are now ready to solve our problem. We know what a solution looks like: this is a subset of all the board positions. We also know how to decide if a solution will work or not. Thus, we will start with a simple program which will try all the possible subsets of positions, and check which ones are solutions that actually work.

The following classes are going to be used for our solutions:

1.1 Solution

A **Solution** is a data structure which represents a (partial) solution to the *Lights Out* game. It portrays an array of arrays of booleans in which the current solution is stored.

We are going to build our solutions methodically. Since a solution identifies the board positions that are selected, in order to build our solution we will simply start from the position (1,1) and specify if (1,1) is selected or not in our solution. We then move to position (1,2) and we also specify if (1,2) is selected or not in our solution. We keep moving along, row by row, until the solution is completely specified.

Because we move systematically, row by row, always specifying the next position, we do not need to expressly say which board position we specify. The object will understand that we are specifying the next available position.

In order to specify the next value, we will use a method called "setNext". Two other methods are available us: the method "isReady" returns *true* if and only if the solution has been completely specified, that is, if all positions of the board have been explicitly set to either *true* or *false*. Finally, the method "isSuccessful" returns *true* if and only if the solution is completely specified and will solve the game. Of course we will also provide a method "toString" to create a String representation an object **Solution**.

For example, consider this solution for a 3x2 board:

true	true	false
true	true	false

Here is an example of code to create this solution, test for completion and success:

```java
public class Test {

    public static void main(String[] args) {
        Solution solution;
        solution = new Solution(3,2);
        solution.setNext(true);
        solution.setNext(true);
        solution.setNext(false);
        solution.setNext(true);
        solution.setNext(true);
        solution.setNext(false);
        System.out.println("Solution is ready: " + solution.isReady());
        System.out.println("The solution is:");
        System.out.println(solution);
        System.out.println("Solution is successful: " + solution.isSuccessful());
    }
}
```

The code will produce the following output:

```
> java Test
Solution is ready: true
The solution is:
[[true, true, false],
[true,true,false]]
Solution is successful: true
>
```

If you consider instead the following solution for the same board:

true	false	true
false	true	false

The code to create this solution, test for completion midway through and at the end, and test for success is :

```java
public class Test {

    public static void main(String[] args) {
        Solution solution;
        solution = new Solution(3,2);
        solution.setNext(true);
        solution.setNext(false);
        solution.setNext(true);
        System.out.println("Midway - Solution is ready: " + solution.isReady());
        solution.setNext(false);
        solution.setNext(true);
        solution.setNext(false);
        System.out.println("Solution is ready: " + solution.isReady());
        System.out.println("The solution is:");
        System.out.println(solution);
        System.out.println("Solution is successful: " + solution.isSuccessful());
    }
}
```

The code will produce the following output:

```
> java Test
Midway - Solution is ready: false
Solution is ready: true
The solution is:
```

```
[[true, false, true],
[false, true, false]]
Solution is successful: false
>
```

The specification for our class *Solution* is as follows. First, it has two constructors:

- **Solution(int width, int height)**: creates an instance of **Solution** for a board of size *width*x*height*. That solution does not have any board position value explicitly specified yet.
- **Solution(Solution other)**: creates an instance of **Solution** which is the same as the one received as a parameter. It will be for a board of the same dimension, and the same board positions are specified, with the same values. In other words, the new instance is a *deep copy* of the instance *other*. See Appendix A.

Each **Solution** has the following instance methods:

- **void setNext(boolean nextValue)**: specifies the "next" value of the solution to *nextValue*. The first call to setNext specifies the value of the board location (1,1), the second call specifies the value of the board location (2,1) etc. On a nxm board, the n^{th} call to setNext specifies the value of the board location (1,2). If *setNext* is called more times than there are positions on the board, an error message is printed out and the call is ignored.
- **boolean isReady()**: returns *true* if the solution has been entirely specified, *false* otherwise.
- **boolean isSuccessful()**: returns **true** if the solution is completely specified and is indeed working, that is, if it will bring a board of the specified dimensions from being entirely "off" to being entirely "on".
- **boolean equals(Object other)**: returns true if and only the parameter *other* is referencing an instance of a **Solution** which is the "same" as *this* instance of **Solution**. Make sure that your implementation is as reliable as possible and handle all possible cases[1].
- **String toString()**: returns a **String** representation of the solution.

Note that you need to implement the class *Solution* "from scratch", so you are not allowed to use *java.util.ArrayList* or any other non-built-in element of Java in that particular class.

1.2 ArrayListSolutionQueue

The class *ArrayListSolutionQueue* implements the interface *SolutionQueue*, which is provided. In this case, this is a special "Queue" which handle only objects that are an instance of the class *Solution* (we will see later in the semester how to wave such a restriction and have an implementation that works safely with any objects).

The interface is defined as follows:

```java
public interface SolutionQueue {
    boolean isEmpty();
    void enqueue(Solution s);
    Solution dequeue();
}
```

ArrayListSolutionQueue is an implementation of that interface which relies on an instance of *ArrayList*. For this, it uses the following instance variable:

- **ArrayList<Solution> queue**

It has the following methods:

- **ArrayListSolutionQueue**: the constructor of the class.
- **void enqueue(Solution value)**: the implementation of the method *enqueue* from the interface *SolutionQueue*.
- **Solution dequeue()**: the implementation of the method *dequeue* from the interface *SolutionQueue*.
- **boolean isEmpty()**: the implementation of the method *isEmpty* from the interface *SolutionQueue*.

[1]Because that method will be used in automated tests, you need to make sure that it works well, even if your own code doesn't use it.

1.3 LightsOut

The class *LightsOut* is the one finding all the solutions. The technique used to find these solutions is known as a *Breadth-First Search*. It uses a *Queue* in which partial solutions are stored. At each iteration, the partial solution at the front of the queue is removed from the queue. The algorithm checks if that solution is in fact complete. If it is complete, it checks if it is correct (if it works as a solution), and if so it records it, else it discards it. If the solution is not complete, then the algorithm extends that solution in every possible way. Each newly extended solution is enqueued at the back of the queue. The algorithm starts with a single, completely unspecified solution in the queue, and stops once the queue is empty.

For this assignment, the queue that you use should be an instance of the class *ArrayListSolutionQueue* which you need to provide. The set of *Solution* objects found by the algorithm are themselves stored directly in an *ArrayList* (not a *Queue*). Here is a sketch of the method *solve*, which uses a *Breadth-First Search* to find all the solutions of the game *Lights Out* for a board of size *widthxheight*.

```
Result: The list of all valid solutions for a widthxheight board.
Create a queue partialSolutions and an ArrayList instance solutions;
Initialize partialSolutions to contain a solution for a widthxheight board in which no position is specified;
while partialSolutions is not empty do
    Take the front element out of the partialSolutions queue, call this current;
    if current is fully specified and correct then
        add current to solutions;
    else
        foreach possible extension of current do
            add the extension to rear of the queue partialSolutions;
        end
    end
end
return solutions;
```

- **ArrayList<Solution> solve(int width, int height)**: The method **solve** finds all the solutions to the *Lights Out* game for an initially completely "off" board of size *widthxheight*, using a *Breadth-First Search* algorithm. It returns an **ArrayList** containing all the valid solutions to the problem.

  During the computation of the solution, the method prints out a message each time a new solution is found, along with the total time it took (in milliseconds) to find that solution.

- **main**: Create a **main** method which calls the method **solve** and then prints out the number of solutions found, as well as the details of each solution.

  The *width* and *height* used by the **main** are passed as runtime parameters to the program. If no runtime parameters are passed to the program, or if the parameters are incorrect, then a default value of 3 for both *width* and *height* is used.

A few sample runs of the program are shown in Appendix B.

2 Second Implementation [30 Marks]

Our first solution works well on very small board, but quickly becomes unusable with slightly larger boards[2].

We can easily optimize our code by remarking that once a solution is sure to fail, there is no point in continuing to extend it to the end. That solution can be dropped immediately.

Note that we build our solution systematically, working from left to right, top to bottom. So the positions that are above the "working line" have their neighbourhood already completely specified. If the conditions for a working solution are not met for these positions, then these conditions will not be met for any completed versions of that solution and that solution can be readily abandoned.

We modify our class **Solution** to add the following method:

[2]Can you figure out why the solution takes so much time as soon as the board contains more than a few positions?

- **public boolean stillPossible(boolean nextValue)**: this method returns **false** if the current solution would be impossible to finalize into a working solution, should it be extended from its current state with the value **nextValue**.

  Note that the method returns **false** if extending the current solution with **nextValue** would never yield a working solution, and **true** otherwise. Returning **true** *does not mean that the solution can necessarily be extended into a working solution*. It merely means that we do not yet know if it is still possible.

  Note as well that this method should not modify the state of the object instance of *Solution* on which it is called. It does not extend that solution with **nextValue**, it simply indicates if such an extension would annihilate its chance of leading to a working solution.

.

You need to modify the method **solve** of the class **LightsOut** to take benefit of the method **stillPossible** and provide a more efficient implementation.

A few sample runs of the updated program are shown in Appendix C.

3 (Bonus Question) Third Implementation [10 marks]

An even more efficient implementation can be easily achieved. For this, we need to remark that once a solution can only be extended one way and remain a possibly valid solution, it will never be extendable two ways after that. In other words, if a partial solution can only be extended by either *true* or *false* and remain a possible solution, but not by both *true* and *false* and remain a possible solution, then no extension of that solution which can still yield a correct solution can be extended by both *true* and *false* either. Once a solution as only one way to be extended into a possible solution, it will remain that way until it either yield a solution, or it is shown not to lead to a workable solution.

We can leverage this information in our search for solutions: in our breadth-first search, once a solution is locked into a single alternative mode (it can be possibly extended, but only one way) then we do not need to keep using our breadth-first search with it. We know that at that point that solution will only be extendable into a series of unique next solutions, with no alternatives along the way, so we can simply extend that solution "to the end" and see if we reach a working solution or not. This will save many queue operations and should make our program more efficient.

To achieve this, we modify our class **Solution** to add the following method:

- **public boolean finish()**: this method assumes that solution is currently still extendable, but only one way. It keeps extending that solution with the one correct way that it finds at each step, until the solution is complete and correct, or shown to not be work. It returns **true** if and only if the solution is extended into a complete, working solution.

  That method does change the state of the object instance of *Solution* on which it is called. If it returns **true**, then that instance is now a complete, working solution.

  As you implement this method, pay attention to special cases regarding the dimensions of the board.

.

You need to modify the method **solve** of the class **LightsOut** to use the method **finish** whenever it can and provide a more efficient implementation.

A few sample runs of the updated program are shown in Appendix D.

Academic Integrity

This part of the assignment is meant to raise awareness concerning plagiarism and academic integrity. Please read the following documents.

- `https://www.uottawa.ca/administration-and-governance/academic-regulation-14-other-important-informat`
- `https://www.uottawa.ca/vice-president-academic/academic-integrity`

Cases of plagiarism will be dealt with according to the university regulations. By submitting this assignment, you acknowledge:

1. I have read the academic regulations regarding academic fraud.

2. I understand the consequences of plagiarism.
3. With the exception of the source code provided by the instructors for this course, all the source code is mine.
4. I did not collaborate with any other person, with the exception of my partner in the case of team work.
    - If you did collaborate with others or obtained source code from the Web, then please list the names of your collaborators or the source of the information, as well as the nature of the collaboration. Put this information in the submitted README.txt file. Marks will be deducted proportional to the level of help provided (from 0 to 100%).

Rules and regulation

- Follow all the directives available on the assignment directives web page.
- Submit your assignment through the on-line submission system virtual campus.
- You must preferably do the assignment in teams of two, but you can also do the assignment individually.
- You must use the provided template classes below.
- If you do not follow the instructions, your program will make the automated tests fail and consequently your assignment will not be graded.
- We will be using an automated tool to compare all the assignments against each other (this includes both, the French and English sections). Submissions that are flagged by this tool will receive the grade of 0.
- It is your responsibility to make sure that BrightSpace has received your assignment. Late submissions will not be graded.

Files

You must hand in a **zip** file (no other file format will be accepted). The name of the top directory has to have the following form: **a2_3000000_3000001**, where 3000000 and 3000001 are the student numbers of the team members submitting the assignment (simply repeat the same number if your team has one member). The name of the folder starts with the letter "a" (lowercase), followed by the number of the assignment, here 2. The parts are separated by the underscore (not the hyphen). There are no spaces in the name of the directory. The archive a2_3000000_3000001.zip contains the files that you can use as a starting point. Your submission must contain the following files.

- README.txt
    - A text file that contains the names of the two partners for the assignments, their student ids, section, and a short description of the assignment (one or two lines).
- 01/ArrayListSolutionQueue.java
- 01/SolutionQueue.java
- 01/Solution.java
- 01/LightsOut.java
- 01/StudentInfo.java
- 02/ArrayListSolutionQueue.java
- 02/SolutionQueue.java
- 02/Solution.java
- 02/LightsOut.java
- 02/StudentInfo.java

If you are also answering to the third question, then your submission must also contain the following files:

- 03/ArrayListSolutionQueue.java
- 03/SolutionQueue.java
- 03/Solution.java
- 03/LightsOut.java
- 03/StudentInfo.java

Questions

For all your questions, please visit the Piazza Web site for this course:

- https://piazza.com/uottawa.ca/winter2019/iti1121/home

Last modified: February 4, 2019

A Shallow copy versus Deep copy

As you know, objects have variables which are either a primitive type, or a reference type. Primitive variables hold a value from one of the language primitive type, while reference variables hold a reference (the address) of another object (including arrays, which are objects in Java).

If you are copying the current state of an object, in order to obtain a duplicate object, you will create a copy of each of the variables. By doing so, the value of each instance primitive variable will be duplicated (thus, modifying one of these values in one of the copy will not modify the value on the other copy). However, with reference variables, what will be copied is the actual reference, the address of the object that this variable is pointing at. Consequently, the reference variables in both the original object and the duplicated object will point at the same address, and the reference variables will refer to the same objects. This is known as a **shallow** copy: you indeed have two objects, but they share all the objects pointed at by their instance reference variables. The Figure 5 provides an example: the object referenced by variable **b** is a shallow copy of the object referenced by variable **a**: it has its own copies of the instances variables, but the reference variables **title** and **time** are referencing the same objects.

Often, a shallow copy is not adequate: what is required is a so-called **deep** copy. A deep copy differs from a shallow copy in that objects referenced by reference variable must also be recursively duplicated, in such a way that when the initial object is (deep) copied, the copy does not share any reference with the initial object. The Figure 6 provides an example: this time, the object referenced by variable **b** is a deep copy of the object referenced by variable **a**: now, the reference variables **title** and **time** are referencing different objects. Note that, in turn, the objects referenced by the variable **time** have also been deep copied. The entire set of objects reachable from **a** have been duplicated.

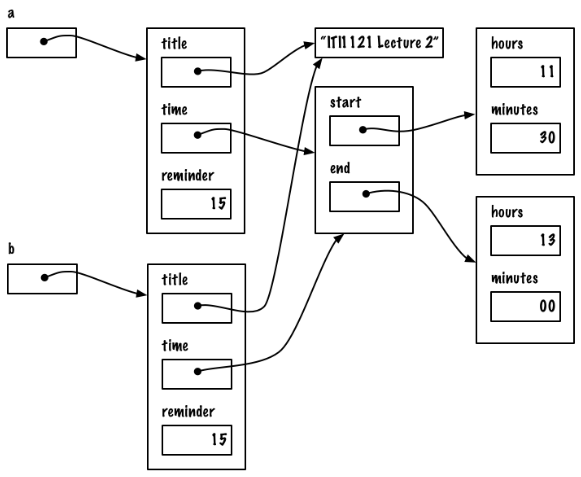

Figure 5: An example of a shallow copy of objects.

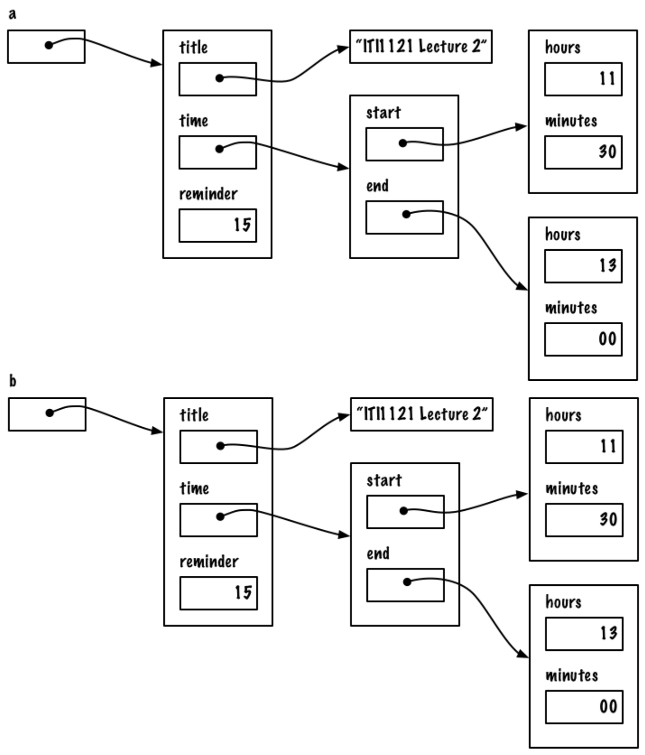

Figure 6: An example of a deep copy of objects.

You can read more about shallow versus deep copy on Wikipedia:

- Object copying

B Sample Runs, question 1

Here are some sample runs with the solution from question 1

```
>  java LightsOut 2 2
Solution found in 1 ms
****
[[true,true],
[true,true]]
In a board of 2x2: 1 solution.
> java LightsOut 3 2
Solution found in 0 ms
Solution found in 0 ms
Solution found in 0 ms
Solution found in 0 ms
****
[[true,true,false],
[true,true,false]]
****
[[true,false,false],
[false,false,true]]
****
[[false,true,true],
[false,true,true]]
****
[[false,false,true],
[true,false,false]]
In a board of 3x2: 4 solutions.
> java LightsOut 4 4
Solution found in 310 ms
Solution found in 339 ms
Solution found in 363 ms
Solution found in 392 ms
Solution found in 425 ms
Solution found in 452 ms
Solution found in 477 ms
Solution found in 492 ms
Solution found in 505 ms
Solution found in 520 ms
Solution found in 535 ms
Solution found in 545 ms
Solution found in 549 ms
Solution found in 557 ms
Solution found in 560 ms
Solution found in 561 ms
****
[[true,true,true,true],
[true,false,false,true],
[true,true,true,true],
[false,false,false,false]]
****
[[true,true,true,false],
[true,false,true,false],
[true,false,true,false],
[true,true,true,false]]
****
[[true,true,false,true],
[true,true,true,false],
[false,true,true,true],
[true,false,true,true]]
****
[[true,true,false,false],
[true,true,false,true],
[false,false,true,false],
[false,true,false,true]]
****
[[true,false,true,true],
[false,true,true,true],
[true,true,true,false],
[true,true,false,true]]
****
[[true,false,true,false],
[false,true,false,false],
[true,false,true,true],
[false,false,true,true]]
****
[[true,false,false,true],
[false,false,false,false],
[false,true,true,false],
[false,true,true,false]]
****
[[true,false,false,false],
[false,false,true,true],
[false,false,true,true],
[true,false,false,false]]
****
[[false,true,true,true],
[false,true,false,true],
[false,true,false,true],
[false,true,true,true]]
****
[[false,true,true,false],
[false,true,true,false],
[false,false,false,false],
[true,false,false,true]]
****
[[false,true,false,true],
[false,false,true,false],
[true,true,false,true],
[true,true,false,false]]
****
[[false,true,false,false],
[false,false,false,true],
[true,false,false,false],
[false,false,true,false]]
****
[[false,false,true,true],
[true,false,true,true],
[false,true,false,false],
[true,false,true,false]]
****
[[false,false,true,false],
[true,false,false,false],
[false,false,false,true],
[false,true,false,false]]
```

```
****
[[false,false,false,true],
[true,true,false,false],
[true,true,false,false],
[false,false,false,true]]
****
[[false,false,false,false],
[true,true,true,true],
[true,false,false,true],
[true,true,true,true]]
In a board of 4x4: 16 solutions.
> java LightsOut
Solution found in 1 ms
****
[[true,false,true],
[false,true,false],
[true,false,true]]
In a board of 3x3: 1 solution.
> java LightsOut -12 0
Invalid width, using default...
Invalid height, using default...
Solution found in 1 ms
****
[[true,false,true],
[false,true,false],
[true,false,true]]
In a board of 3x3: 1 solution.
> java LightsOut 5 4
Solution found in 159456 ms
****
[[false,true,true,true,false],
[false,true,false,true,false],
[false,true,false,true,false],
[false,true,true,true,false]]
In a board of 5x4: 1 solution.
```

C Sample Runs, question 2

Here are some sample runs with the solution from question 2

```
> java LightsOut 5 4
Solution found in 0 ms
****
[[false,true,true,true,false],
[false,true,false,true,false],
[false,true,false,true,false],
[false,true,true,true,false]]
In a board of 5x4: 1 solution.
> java LightsOut 5 5
Solution found in 2 ms
Solution found in 2 ms
Solution found in 2 ms
Solution found in 2 ms
****
[[true,true,false,false,false],
[true,true,false,true,true],
[false,false,true,true,true],
[false,true,true,true,false],
[false,true,true,false,true]]
****
[[true,false,true,true,false],
[false,true,true,true,false],
[true,true,true,false,false],
[true,true,false,true,true],
[false,false,false,true,true]]
****
[[false,true,true,false,true],
[false,true,true,true,false],
[false,false,true,true,true],
[true,true,false,true,true],
[true,true,false,false,false]]
****
[[false,false,false,true,true],
[true,true,false,true,true],
[true,true,true,false,false],
[false,true,true,true,false],
[true,false,true,true,false]]
In a board of 5x5: 4 solutions.
> java LightsOut 10 10
Solution found in 28 ms
****
[[true,false,true,false,false,false,false,true,false,true],
[false,true,false,false,true,true,false,false,true,false],
[true,false,true,false,true,true,false,true,false,true],
[false,false,false,true,false,false,true,false,false,false],
[false,true,true,false,true,true,false,true,true,false],
[false,true,true,false,true,true,false,true,true,false],
[false,false,false,true,false,false,true,false,false,false],
[true,false,true,false,true,true,false,true,false,true],
[false,true,false,false,true,true,false,false,true,false],
[true,false,true,false,false,false,false,true,false,true]]
In a board of 10x10: 1 solution.
> java LightsOut 13 13
Solution found in 896 ms
```

```
****
[[true,true,false,true,false,true,true,true,false,true,false,true,true],
[true,true,true,false,true,true,false,true,true,false,true,true,true],
[false,true,true,false,true,false,false,false,true,false,true,true,false],
[true,false,false,true,true,true,true,true,true,true,false,false,true],
[false,true,true,true,true,false,false,false,true,true,true,true,false],
[true,true,false,true,false,true,false,true,false,true,false,true,true],
[true,false,false,true,false,false,true,false,false,true,false,false,true],
[true,true,false,true,false,true,false,true,false,true,false,true,true],
[false,true,true,true,true,false,false,false,true,true,true,true,false],
[true,false,false,true,true,true,true,true,true,true,false,false,true],
[false,true,true,false,true,false,false,false,true,false,true,true,false],
[true,true,true,false,true,true,false,true,true,false,true,true,true],
[true,true,false,true,false,true,true,true,false,true,false,true,true]]
In a board of 13x13: 1 solution.
> java LightsOut 14 14
Solution found in 5912 ms
Solution found in 5912 ms
Solution found in 5912 ms
Solution found in 5912 ms
Solution found in 5912 ms
Solution found in 5912 ms
Solution found in 5912 ms
Solution found in 5912 ms
Solution found in 5912 ms
Solution found in 5912 ms
Solution found in 5912 ms
Solution found in 5912 ms
Solution found in 5912 ms
Solution found in 5912 ms
Solution found in 5913 ms
Solution found in 5913 ms
****
[[true,true,true,true,false,false,true,true,true,false,false,true,true,false],
[true,false,false,true,false,false,true,false,true,false,false,true,true,false],
[true,true,true,true,false,false,true,false,true,false,false,false,false,false],
[false,false,false,false,false,false,true,true,true,false,true,false,false,true],
[false,false,false,false,true,false,false,false,false,true,false,false,false,false],
[true,true,true,false,false,false,false,false,true,false,true,true,true,false],
[true,false,true,false,false,true,true,false,false,false,true,false,true,false],
[true,false,true,false,false,true,true,false,false,false,true,false,true,false],
[true,true,true,false,false,false,false,false,true,false,true,true,true,false],
[false,false,false,false,true,false,false,false,false,true,false,false,false,false],
[false,false,false,false,false,false,true,true,true,false,true,false,false,true],
[true,true,true,true,false,false,true,false,true,false,false,false,false,false],
[true,false,false,true,false,false,true,false,true,false,false,true,true,false],
[true,true,true,true,false,false,true,true,true,false,false,true,true,false]]
****
[[true,true,true,false,false,true,true,true,true,false,false,true,true,true],
[true,false,true,false,false,true,false,false,true,false,false,true,false,true],
[true,false,true,false,false,true,true,true,true,false,false,true,false,true],
[true,true,true,false,false,false,false,false,false,false,false,true,true,true],
[false,false,false,false,true,false,false,false,false,true,false,false,false,false],
[false,false,false,false,false,false,true,true,false,false,false,false,false,false],
[true,true,true,true,false,false,true,true,false,false,true,true,true,true],
[true,false,false,true,false,false,false,false,false,false,true,false,false,true],
[true,true,true,true,false,true,false,false,true,false,true,true,true,true],
[false,false,false,false,true,false,false,false,false,true,false,false,false,false],
[false,false,false,true,false,true,true,true,true,false,true,false,false,false],
[true,true,false,false,false,true,false,false,true,false,false,false,true,true],
[true,true,false,false,false,true,true,true,true,false,false,false,true,true],
[false,false,false,true,false,false,false,false,false,false,true,false,false,false]]
****
[[true,true,false,true,false,false,false,true,true,false,false,true,false,false],
[true,true,true,false,false,true,false,true,true,false,false,false,false,true],
[false,true,true,true,false,false,true,false,false,false,true,false,false,false],
[true,false,true,true,false,true,false,true,false,false,false,false,true,false],
[false,false,false,false,true,false,false,false,false,true,false,false,false,false],
[false,true,false,true,false,true,true,false,false,false,false,true,false,true],
[false,false,true,false,false,true,true,false,true,false,false,false,true,false],
[true,true,false,true,false,false,false,true,false,false,true,true,false,true],
[true,true,false,false,false,false,true,false,true,false,true,true,false,false],
[false,false,false,false,true,false,false,false,false,true,false,false,false,false],
[false,false,true,false,false,false,false,true,true,false,true,false,true,true],
[true,false,false,false,false,true,false,true,true,false,false,true,true,true],
[false,false,false,true,false,false,true,false,false,false,true,true,true,false],
[false,true,false,false,false,true,false,true,false,false,true,true,false,true]]
****
[[true,true,false,false,false,true,false,true,true,false,false,true,false,true],
[true,true,false,true,false,false,true,true,true,false,false,false,true,false],
[false,false,true,false,false,true,true,true,false,false,true,true,false,true],
[false,true,false,true,false,true,true,false,true,false,true,true,false,false],
[false,false,false,false,true,false,false,false,false,true,false,false,false,false],
[true,false,true,true,false,true,false,true,true,false,true,false,true,true],
[false,true,true,true,false,false,true,true,true,false,false,true,true,true],
[true,true,true,false,false,true,true,true,false,false,true,true,true,false],
[true,true,false,true,false,true,true,false,true,false,true,true,false,true],
[false,false,false,false,true,false,false,false,false,true,false,false,false,false],
[false,false,true,true,false,true,false,true,true,false,true,false,true,false],
[true,false,true,true,false,false,true,true,true,false,false,true,false,false],
[false,true,false,false,false,true,true,true,false,false,true,false,true,true],
[true,false,true,false,false,true,true,false,true,false,false,false,true,true]]
****
[[true,false,true,true,false,false,true,false,true,false,false,false,true,false],
[false,true,true,true,false,false,false,true,false,false,true,false,false,false],
[true,true,true,false,false,true,true,false,true,false,false,false,false,true],
[true,true,false,true,false,true,true,false,false,false,false,true,false,false],
[false,false,false,false,true,false,false,false,false,true,false,false,false,false],
[false,false,true,true,false,true,false,true,false,false,false,false,true,true],
[true,false,true,true,false,false,true,false,false,false,true,false,true,true],
[false,true,false,false,false,true,false,true,true,false,false,true,false,false],
[true,false,true,false,false,false,false,true,true,false,true,false,true,false],
[false,false,false,false,true,false,false,false,false,true,false,false,false,false],
[false,true,false,false,false,false,true,false,true,false,true,true,false,true],
[false,false,false,true,false,false,false,true,false,false,true,true,true,false],
[true,false,false,false,false,true,true,false,true,false,false,true,true,true],
[false,false,true,false,false,true,true,false,false,false,true,false,true,true]]
****
[[true,false,true,false,false,true,true,false,true,false,false,false,true,true],
[false,true,false,false,false,true,true,true,false,false,true,false,true,true],
[true,false,true,true,false,false,true,true,true,false,false,true,false,false],
[false,false,true,true,false,true,false,true,true,false,true,false,true,false],
[false,false,false,false,true,false,false,false,false,true,false,false,false,false],
[true,true,false,true,false,true,true,false,true,false,true,true,false,true],
[true,true,true,false,false,true,true,true,false,false,true,true,true,false],
[false,true,true,true,false,false,true,true,true,false,false,true,true,true],
[true,false,true,true,false,true,false,true,true,false,true,false,true,true],
[false,false,false,false,true,false,false,false,false,true,false,false,false,false],
[false,true,false,true,false,true,true,false,true,false,true,true,false,false],
[false,false,true,false,false,true,true,true,false,false,true,true,false,true],
[true,true,false,true,false,false,true,true,true,false,false,false,true,false],
[true,true,false,false,false,true,false,true,true,false,false,true,false,true]]
****
[[true,false,false,true,false,false,false,false,true,false,false,false,false,false],
[false,false,false,false,false,true,true,false,false,false,true,true,true,true],
[false,true,true,false,false,true,true,false,false,false,true,false,false,true],
[false,true,true,false,false,false,false,false,true,false,true,true,true,true],
[false,false,false,false,true,false,false,false,false,true,false,false,false,false],
[true,false,false,false,false,false,true,true,true,false,true,false,false,false],
[false,false,true,true,false,false,true,false,true,false,false,false,true,true],
[false,false,true,true,false,false,true,false,true,false,false,false,true,true],
[true,false,false,false,false,false,true,true,true,false,true,false,false,false],
[false,false,false,false,true,false,false,false,false,true,false,false,false,false],
[false,true,true,false,false,false,false,false,true,false,true,true,true,true],
[false,true,true,false,false,true,true,false,false,false,true,false,false,true],
[false,false,false,false,false,true,true,false,false,false,true,true,true,true],
[true,false,false,true,false,false,false,false,true,false,false,false,false,false]]
****
[[true,false,false,false,false,true,false,false,true,false,false,false,false,true],
[false,false,true,true,false,false,false,false,false,false,true,true,false,false],
[false,false,true,true,false,false,true,true,false,false,true,true,false,false],
[true,false,false,false,false,false,true,true,false,false,false,false,false,true],
[false,false,false,false,true,false,false,false,false,true,false,false,false,false],
[false,true,true,false,false,false,false,false,false,false,false,true,true,false],
[false,true,true,false,false,true,true,true,true,false,false,true,true,false],
[false,false,false,false,false,true,false,false,true,false,false,false,false,false],
[true,false,false,true,false,true,true,true,true,false,true,false,false,true],
[false,false,false,false,true,false,false,false,false,true,false,false,false,false],
[false,true,true,true,false,true,false,false,true,false,true,true,true,false],
[false,true,false,true,false,false,false,false,false,false,true,false,true,false],
[false,true,false,true,false,false,true,true,false,false,true,false,true,false],
[false,true,true,true,false,false,true,true,false,false,true,true,true,false]]
****
[[false,true,true,true,false,false,true,true,false,false,true,true,true,false],
[false,true,false,true,false,false,true,true,false,false,true,false,true,false],
[false,true,false,true,false,false,false,false,false,false,true,false,true,false],
[false,true,true,true,false,true,false,false,true,false,true,true,true,false],
[false,false,false,false,true,false,false,false,false,true,false,false,false,false],
[true,false,false,true,false,true,true,true,true,false,true,false,false,true],
[false,false,false,false,false,true,false,false,true,false,false,false,false,false],
[false,true,true,false,false,true,true,true,true,false,false,true,true,false],
[false,true,true,false,false,false,false,false,false,false,false,true,true,false],
[false,false,false,false,true,false,false,false,false,true,false,false,false,false],
[true,false,false,false,false,false,true,true,false,false,false,false,false,true],
[false,false,true,true,false,false,true,true,false,false,true,true,false,false],
[false,false,true,true,false,false,false,false,false,false,true,true,false,false],
[true,false,false,false,false,true,false,false,true,false,false,false,false,true]]
****
[[false,true,true,false,false,true,true,true,false,false,true,true,true,true],
[false,true,true,false,false,true,false,true,false,false,true,false,false,true],
[false,false,false,false,false,true,false,true,false,false,true,true,true,true],
[true,false,false,true,false,true,true,true,false,false,false,false,false,false],
[false,false,false,false,true,false,false,false,false,true,false,false,false,false],
[false,true,true,true,false,true,false,false,false,false,false,true,true,true],
[false,true,false,true,false,false,false,true,true,false,false,true,false,true],
[false,true,false,true,false,false,false,true,true,false,false,true,false,true],
[false,true,true,true,false,true,false,false,false,false,false,true,true,true],
[false,false,false,false,true,false,false,false,false,true,false,false,false,false],
[true,false,false,true,false,true,true,true,false,false,false,false,false,false],
[false,false,false,false,false,true,false,true,false,false,true,true,true,true],
[false,true,true,false,false,true,false,true,false,false,true,false,false,true],
[false,true,true,false,false,true,true,true,false,false,true,true,true,true]]
****
[[false,true,false,true,false,false,false,true,false,false,true,true,false,false],
[false,false,true,false,false,true,false,false,false,false,true,true,false,true],
[true,true,false,true,false,false,false,false,true,false,false,false,true,false],
[true,true,false,false,false,false,true,false,false,false,false,true,false,true],
[false,false,false,false,true,false,false,false,false,true,false,false,false,false],
[false,false,true,false,false,false,false,true,false,false,false,false,true,false],
[true,false,false,false,false,true,false,false,false,false,true,false,false,false],
[false,false,false,true,false,false,false,false,true,false,false,false,false,true],
[false,true,false,false,false,false,true,false,false,false,false,true,false,false],
[false,false,false,false,true,false,false,false,false,true,false,false,false,false],
[true,false,true,false,false,false,false,true,false,false,false,false,true,true],
[false,true,false,false,false,true,false,false,false,false,true,false,true,true],
[true,false,true,true,false,false,false,false,true,false,false,true,false,false],
[false,false,true,true,false,false,true,false,false,false,true,false,true,false]]
****
[[false,true,false,false,false,true,false,true,false,false,true,true,false,true],
[false,false,false,true,false,false,true,false,false,false,true,true,true,false],
[true,false,false,false,false,true,false,true,true,false,false,true,true,true],
[false,false,true,false,false,false,false,true,true,false,true,false,true,true],
[false,false,false,false,true,false,false,false,false,true,false,false,false,false],
[true,true,false,false,false,false,true,false,true,false,true,true,false,false],
[true,true,false,true,false,false,false,true,false,false,true,true,false,true],
[false,false,true,false,false,true,true,false,true,false,false,false,true,false],
[false,true,false,true,false,true,true,false,false,false,false,true,false,true],
[false,false,false,false,true,false,false,false,false,true,false,false,false,false],
[true,false,true,true,false,true,false,true,false,false,false,false,true,false],
[false,true,true,true,false,false,true,false,false,false,true,false,false,false],
[true,true,true,false,false,true,false,true,true,false,false,false,false,true],
[true,true,false,true,false,false,false,true,true,false,false,true,false,false]]
****
[[false,false,true,true,false,false,true,false,false,false,true,false,true,false],
[true,false,true,true,false,false,false,false,true,false,false,true,false,false],
[false,true,false,false,false,true,false,false,false,false,true,false,true,true],
[true,false,true,false,false,false,false,true,false,false,false,false,true,true],
[false,false,false,false,true,false,false,false,false,true,false,false,false,false],
[false,true,false,false,false,false,true,false,false,false,false,true,false,false],
[false,false,false,true,false,false,false,false,true,false,false,false,false,true],
[true,false,false,false,false,true,false,false,false,false,true,false,false,false],
[false,false,true,false,false,false,false,true,false,false,false,false,true,false],
[false,false,false,false,true,false,false,false,false,true,false,false,false,false],
[true,true,false,false,false,false,true,false,false,false,false,true,false,true],
[true,true,false,true,false,false,false,false,true,false,false,false,true,false],
[false,false,true,false,false,true,false,false,false,false,true,true,false,true],
[false,true,false,true,false,false,false,true,false,false,true,true,false,false]]
****
```

```
[[false,false,true,false,false,true,true,false,false,false,true,false,true,true],
[true,false,false,false,false,true,true,false,true,false,false,true,true,true],
[false,false,false,true,false,false,false,true,false,false,true,true,true,false],
[false,true,false,false,false,false,true,false,true,false,true,true,false,true],
[false,false,false,false,true,false,false,false,false,true,false,false,false,false],
[true,false,true,false,false,false,false,true,true,false,true,false,true,false],
[false,true,false,false,false,true,false,true,true,false,false,true,false,false],
[true,false,true,true,false,false,true,false,false,false,true,false,true,true],
[false,false,true,true,false,true,false,true,false,false,false,false,true,true],
[false,false,false,false,true,false,false,false,false,true,false,false,false,false],
[true,true,false,true,false,true,true,false,false,false,false,true,false,false],
[true,true,true,false,false,true,true,false,true,false,false,false,false,true],
[false,true,true,true,false,false,false,true,false,false,true,false,false,false],
[true,false,true,true,false,false,true,false,true,false,false,false,true,false]]
****
[[false,false,false,true,false,false,false,false,false,false,true,false,false,false],
[true,true,false,false,false,true,true,true,true,false,false,false,true,true],
[true,true,false,false,false,true,false,false,true,false,false,false,true,true],
[false,false,false,true,false,true,true,true,true,false,true,false,false,false],
[false,false,false,false,true,false,false,false,false,true,false,false,false,false],
[true,true,true,true,false,true,false,false,true,false,true,true,true,true],
[true,false,false,true,false,false,false,false,false,false,true,false,false,true],
[true,true,true,true,false,false,true,true,false,false,true,true,true,true],
[false,false,false,false,false,false,true,true,false,false,false,false,false,false],
[false,false,false,false,true,false,false,false,false,true,false,false,false,false],
[true,true,true,false,false,false,false,false,false,false,false,true,true,true],
[true,false,true,false,false,true,true,true,true,false,false,true,false,true],
[true,false,true,false,false,true,false,false,true,false,false,true,false,true],
[true,true,true,false,false,true,true,true,true,false,false,true,true,true]]
****
[[false,false,false,false,false,true,false,false,false,false,true,false,false,true],
[true,true,true,true,false,false,false,true,true,false,false,false,false,false],
[true,false,false,true,false,false,false,true,true,false,false,true,true,false],
[true,true,true,true,false,true,false,false,false,false,false,true,true,false],
[false,false,false,false,true,false,false,false,false,true,false,false,false,false],
[false,false,false,true,false,true,true,true,false,false,false,false,false,true],
[true,true,false,false,false,true,false,true,false,false,true,true,false,false],
[true,true,false,false,false,true,false,true,false,false,true,true,false,false],
[false,false,false,true,false,true,true,true,false,false,false,false,false,true],
[false,false,false,false,true,false,false,false,false,true,false,false,false,false],
[true,true,true,true,false,true,false,false,false,false,false,true,true,false],
[true,false,false,true,false,false,false,true,true,false,false,true,true,false],
[true,true,true,true,false,false,false,true,true,false,false,false,false,false],
[false,false,false,false,false,true,false,false,false,false,true,false,false,true]]
In a board of 14x14: 16 solutions.
> java LightsOut 15 15
Solution found in 26463 ms
****
[[true,false,true,true,false,false,true,false,true,false,false,true,true,false,true],
[false,true,true,true,false,false,false,true,false,false,false,true,true,true,false],
[true,true,true,false,false,true,true,false,true,true,false,false,true,true,true],
[true,true,false,true,false,true,true,false,true,true,false,true,false,true,true],
[false,false,false,false,true,false,false,true,false,false,true,false,false,false,false],
[false,false,true,true,false,true,true,false,true,true,false,true,true,false,false],
[true,false,true,true,false,true,true,false,true,true,false,true,true,false,true],
[false,true,false,false,true,false,false,true,false,false,true,false,false,true,false],
[true,false,true,true,false,true,true,false,true,true,false,true,true,false,true],
[false,false,true,true,false,true,true,false,true,true,false,true,true,false,false],
[false,false,false,false,true,false,false,true,false,false,true,false,false,false,false],
[true,true,false,true,false,true,true,false,true,true,false,true,false,true,true],
[true,true,true,false,false,true,true,false,true,true,false,false,true,true,true],
[false,true,true,true,false,false,false,true,false,false,false,true,true,true,false],
[true,false,true,true,false,false,true,false,true,false,false,true,true,false,true]]
In a board of 15x15: 1 solution.
>
```

D Sample Runs, question 3

Here are some sample runs with the solution from question 3

```
> java LightsOut 15 15
Solution found in 329 ms
****
[[true,false,true,true,false,false,true,false,true,false,false,true,true,false,true],
[false,true,true,true,false,false,false,true,false,false,false,true,true,true,false],
[true,true,true,false,false,true,true,false,true,true,false,false,true,true,true],
[true,true,false,true,false,true,true,false,true,true,false,true,false,true,true],
[false,false,false,false,true,false,false,true,false,false,true,false,false,false,false],
[false,false,true,true,false,true,true,false,true,true,false,true,true,false,false],
[true,false,true,true,false,true,true,false,true,true,false,true,true,false,true],
[false,true,false,false,true,false,false,true,false,false,true,false,false,true,false],
[true,false,true,true,false,true,true,false,true,true,false,true,true,false,true],
[false,false,true,true,false,true,true,false,true,true,false,true,true,false,false],
[false,false,false,false,true,false,false,true,false,false,true,false,false,false,false],
[true,true,false,true,false,true,true,false,true,true,false,true,false,true,true],
[true,true,true,false,false,true,true,false,true,true,false,false,true,true,true],
[false,true,true,true,false,false,false,true,false,false,false,true,true,true,false],
[true,false,true,true,false,false,true,false,true,false,false,true,true,false,true]]
In a board of 15x15: 1 solution.
>java LightsOut 15 25
Solution found in 451 ms
****
[[false,true,false,false,true,false,false,true,false,false,true,false,false,true,false],
[false,false,false,false,false,false,false,false,false,false,false,false,false,false,false],
[true,false,true,true,false,true,true,false,true,true,false,true,true,false,true],
[false,true,true,true,true,true,true,true,true,true,true,true,true,true,false],
[true,true,true,true,false,true,true,false,true,true,false,true,true,true,true],
[true,true,true,false,false,false,false,false,false,false,false,false,true,true,true],
[false,true,false,true,true,false,false,true,false,false,true,true,false,true,false],
[true,true,false,true,true,false,false,false,false,false,true,true,false,true,true],
[true,false,true,false,false,false,true,false,true,false,false,false,true,false,true],
[true,false,false,true,false,false,false,true,false,false,false,true,false,false,true],
[true,false,true,false,false,true,true,false,true,true,false,false,true,false,true],
[true,true,false,true,false,true,true,false,true,true,false,true,false,true,true],
[false,true,false,false,true,false,false,true,false,false,true,false,false,true,false],
[true,true,false,true,false,true,true,false,true,true,false,true,false,true,true],
[true,false,true,false,false,true,true,false,true,true,false,false,true,false,true],
[true,false,false,true,false,false,false,true,false,false,false,true,false,false,true],
[true,false,true,false,false,false,true,false,true,false,false,false,true,false,true],
[true,true,false,true,true,false,false,false,false,false,true,true,false,true,true],
[false,true,false,true,true,false,false,true,false,false,true,true,false,true,false],
[true,true,true,false,false,false,false,false,false,false,false,false,true,true,true],
[true,true,true,true,false,true,true,false,true,true,false,true,true,true,true],
[false,true,true,true,true,true,true,true,true,true,true,true,true,true,false],
[true,false,true,true,false,true,true,false,true,true,false,true,true,false,true],
[false,false,false,false,false,false,false,false,false,false,false,false,false,false,false],
[false,true,false,false,true,false,false,true,false,false,true,false,false,true,false]]
In a board of 15x25: 1 solution.

> java LightsOut 18 18
Solution found in 13624 ms
****
[[true,true,false,true,false,true,true,true,false,false,true,true,true,false,true,false,true,true],
[true,true,true,false,true,true,false,true,false,false,true,false,true,true,false,true,true,true],
[false,true,true,false,true,false,false,true,false,false,true,false,false,true,false,true,true,false],
[true,false,false,true,true,true,false,true,false,false,true,false,true,true,true,false,false,true],
[false,true,true,true,true,true,true,true,false,false,true,true,true,true,true,true,true,false],
[true,true,false,true,true,true,false,false,false,false,false,false,true,true,true,false,true,true],
[true,false,false,false,true,false,true,false,true,true,false,true,false,true,false,false,false,true],
[true,true,true,true,true,false,false,true,true,true,true,false,false,true,true,true,true,true],
[false,false,false,false,false,false,true,true,true,true,true,true,false,false,false,false,false,false],
[false,false,false,false,false,false,true,true,true,true,true,true,false,false,false,false,false,false],
[true,true,true,true,true,false,false,true,true,true,true,false,false,true,true,true,true,true],
[true,false,false,false,true,false,true,false,true,true,false,true,false,true,false,false,false,true],
[true,true,false,true,true,true,false,false,false,false,false,false,true,true,true,false,true,true],
[false,true,true,true,true,true,true,true,false,false,true,true,true,true,true,true,true,false],
[true,false,false,true,true,true,false,true,false,false,true,false,true,true,true,false,false,true],
[false,true,true,false,true,false,false,true,false,false,true,false,false,true,false,true,true,false],
[true,true,true,false,true,true,false,true,false,false,true,false,true,true,false,true,true,true],
[true,true,false,true,false,true,true,true,false,false,true,true,true,false,true,false,true,true]]
In a board of 18x18: 1 solution.

> java LightsOut 20 20
Solution found in 195372 ms
****
[[true,false,true,false,false,true,true,true,true,true,true,true,true,true,true,false,false,true,false,tru
[false,true,false,false,false,true,false,false,false,false,false,false,false,false,true,false,false,false,
[true,false,true,true,false,true,true,false,false,false,false,false,false,true,true,false,true,true,false,
[false,false,true,true,true,false,true,false,true,true,true,true,false,true,false,true,true,true,false,fal
[false,false,false,true,true,false,true,true,true,false,false,true,true,true,false,true,true,false,false,f
[true,true,true,false,false,true,false,false,false,true,true,false,false,false,true,false,false,true,true,
[true,false,true,true,true,false,true,false,true,true,true,true,false,true,false,true,true,true,false,true
[true,false,false,false,true,false,false,true,true,true,true,true,true,false,false,true,false,false,false,
[true,false,false,true,true,false,true,true,true,true,true,true,true,true,false,true,true,false,false,true
[true,false,false,true,false,true,true,true,true,true,true,true,true,true,true,false,true,false,false,true
[true,false,false,true,false,true,true,true,true,true,true,true,true,true,true,false,true,false,false,true
[true,false,false,true,true,false,true,true,true,true,true,true,true,true,false,true,true,false,false,true
[true,false,false,false,true,false,false,true,true,true,true,true,true,false,false,true,false,false,false,
[true,false,true,true,true,false,true,false,true,true,true,true,false,true,false,true,true,true,false,true
[true,true,true,false,false,true,false,false,false,true,true,false,false,false,true,false,false,true,true,
[false,false,false,true,true,false,true,true,true,false,false,true,true,true,false,true,true,false,false,f
[false,false,true,true,true,false,true,false,true,true,true,true,false,true,false,true,true,true,false,fal
[true,false,true,true,false,true,true,false,false,false,false,false,false,true,true,false,true,true,false,
[false,true,false,false,false,true,false,false,false,false,false,false,false,false,true,false,false,false,
[true,false,true,false,false,true,true,true,true,true,true,true,true,true,true,false,false,true,false,true
In a board of 20x20: 1 solution.
```