**Digital Computer Fundamentals**

**Tabular Method – Opti-Circuit**

**Final Project Report**

***By***

**Hesham Medhat & Marina Zakaria**

**Tabular Method Project Report**

**CS131 Course, Faculty of Engineering, Alexandria University.**

**Computer & Systems Engineering Dpt.**

**To Professor Mustafa El-Nenaei**

**and teaching assistants:**

**• Eng/ Reham Mohamed Samir.**

**• Eng/ Rana Momtaz.**

**Index**

1. Introduction 4
2. Overview 5
3. Features 5
4. Data Structures 6
5. Functions 9
6. User Manual
7. Sample runs
8. References

**Introduction**

This project was assigned in the CS131: Digital Computer Fundamentals course in the date: Friday, April 28th, 2017.

And due : Monday,May 15th, 2017.

As the final project of the course, we were assigned this final project as teams of two.

This delivery is by :

*Hesham Medhat Mahmoud Ahmed Abou-Mousa*

&

*Marina Zakaria Boules Youssef Zakaria*

The source of this project is available on github on this link:

<https://github.com/EngHesham/Opti-Circuit/>

**Overview**

This program intends to optimize a digital logic circuit for possible solutions of lowest cost.

All the program needs as input is the minterms of the function that needs optimization and the don’t-care elements (optional).

**Features**

*In our implementation,*

The program can:

• give all possible optimal solutions.

• output all prime implicants.

• output all essential prime implicants.

• show some steps of solution.

• read input from a file.

• print output in a file

**Data Structures**

As we are learning the course Data Structures-1 in the very same semester, we are using our own implementations of Data Structures in this project.

We are using Singly Linked Lists, Doubly Linked Lists, of our own implementation.

How we used them is described thoroughly in the respective interfaces in the “Functions” section.

Here is the ILinkedList interface:

ILinkedList {

/\*\*

\* Inserts a specified element at the specified position in the

\* list.

\*/

public void add(int index, Object element);

/\*\* Inserts the specified element at the end of the list. \*/

public void add(Object element);

/\*\* Returns the element at the specified position in this list.

\*/

public Object get(int index);

/\*\*

\* Replaces the element at the specified position in this list

\* with the specified element.

\*/

public void set(int index, Object element);

/\*\* Removes all of the elements from this list. \*/

public void clear();

/\*\* Returns true if this list contains no elements. \*/

public boolean isEmpty();

/\*\* Removes the element at the specified position in this list. \*/

public void remove(int index);

/\*\* Returns the number of elements in this list. \*/

public int size();

/\*\*

\* Returns a view of the portion of this list between the

\* specified

\* fromIndex and toIndex, inclusively.

\*/

public ILinkedList sublist(int fromIndex, int toIndex);

/\*\*

\* Returns true if this list contains an element with the same

\* value as the specified element.

\*/

public boolean contains(Object o);

}

**Functions**

**A. Module: PrimeImplicants:**

IPrimeImplicants {

/\*\*

\* listing minterms in groups

\* @author Marina

\* @parameters int array of minterms

\* @return array of linked lists

\*/

SinglyLinkedList[] listing(int[] minterms);

/\*\*

\* Sorts the combinations of the implicant.

\* @author Marina

\* @param implicant input

\* @return sorted implicant

\*/

DoublyLinkedList sortImplicantCombinations(DoublyLinkedList implicant);

/\*\*

\* combine one haming distance minterms

\*

\* @parameters 2 lists return a list

\*/

SinglyLinkedList combiningTwoGroups(SinglyLinkedList group1, SinglyLinkedList group2);

/\*\*

\* performs the method combining one level till it generates

\* one group only containing the essential prime implicants

\*@author Marina

\* @parameter array of integers containing minterms

\* @return one singly linked list of prime implicants ready to

\* get combinations method

\*/

SinglyLinkedList generatePrimeImplicants(int[] minterms);

/\*\*

\* performs the method combine 2 groups depending on the

\* length of the array of lists

\* call itself recursively till the length of the list is dropped

\* down to 1 list

\* @author Marina

\* @parameter array of singly linked lists

\* @return array of singly linked lists

\*

\*/

SinglyLinkedList []combineOneLevel (SinglyLinkedList[]list);

**B. Module: EssentialPrimeImplicants:**

IEssentialPrimeImplicants {

/\*\*

\* Gets possible minterm combinations from a PI.

\* This method works recursively adding combinations to a

\* given list.

\* @param node of the list representation of PI.

\* @param sum of the combinations picked so far.

\* @param coveredMTs list where the possible combinations will be added.

\*/

public void findCombinations (final DLNode node, int sum, final DoublyLinkedList coveredMTs);

/\*\*

\* This method gets the array of MTs covered by given PIs.

\* @param primes array of PIs lists.

\* @return An array of lists of MTs covered by the PIs.

\* Indices of this array correspond to the indices of

\* the given PIs array.

\*/

public DoublyLinkedList[] coveredMinterms(DoublyLinkedList[] primes);

/\*\*

\* This method produces a list of given prime implicants.

\* @param coveredMT this is the array of lists of covered MTs by each PI.

\* Obtained from coveredMinterms method.

\* @param minterms this is the sorted array of MTs.

\* @return An array of covering PIs for each MT.

\* Indices of this array correspond to the MTs.

\*/

public DoublyLinkedList[] coveringPIs(DoublyLinkedList[] coveredMT, int[] minterms);

/\*\*

\* This method produces the solution formula.

\* @param coveringImplicants array obtained from coveringImplicants.

\* @return String of the solution formula.

\* This needs simplification to get the best possible solutions

\* for optimization.

\* At this point, we need boolean algebra solving techniques to

\* get the final optimal answers.

\*/

public String getFormula(final DoublyLinkedList[] coveringImplicants);

\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*

/\*\*

\* This method finds the essential PIs using dominance.

\* @param coveringImplicants array obtained from coveringImplicants.

\* @return The essential prime implicants in the form of a

\* string as Px Py Pz where (x,y,z..) represent the indices

\* of these implicants in the "primes" input list.

\*/

public String getEssentials(final DoublyLinkedList[] coveringImplicants);

/\*\*

\* This function uses recursion to find all possible solutions

\* of how to use the covering PIs to cover the MTs we need covered.

\* @param solutions the returner list of solutions

\* @param toCover index of coveringPIs MT that we need covered

\* @param coveringPIs obtained from coveringPIs method

\* @param thisSolution the current solution that the method

\* is trying to get in the recursive calls

\* @return the list of SLLs of solutions.

\*/

public SinglyLinkedList findSolutions(SinglyLinkedList solutions, int toCover, DoublyLinkedList[] coveringPIs, SinglyLinkedList thisSolution);

/\*\*

\* This method uses all methods in this class to obtain all the

\* possible solutions for the given primes to cover given minterms.

\* @param primes array of prime implicants (DLLs).

\* @param minterms list of minterms to cover.

\* @return a String list of solutions.

\* Each element of this list is a string solution in the form

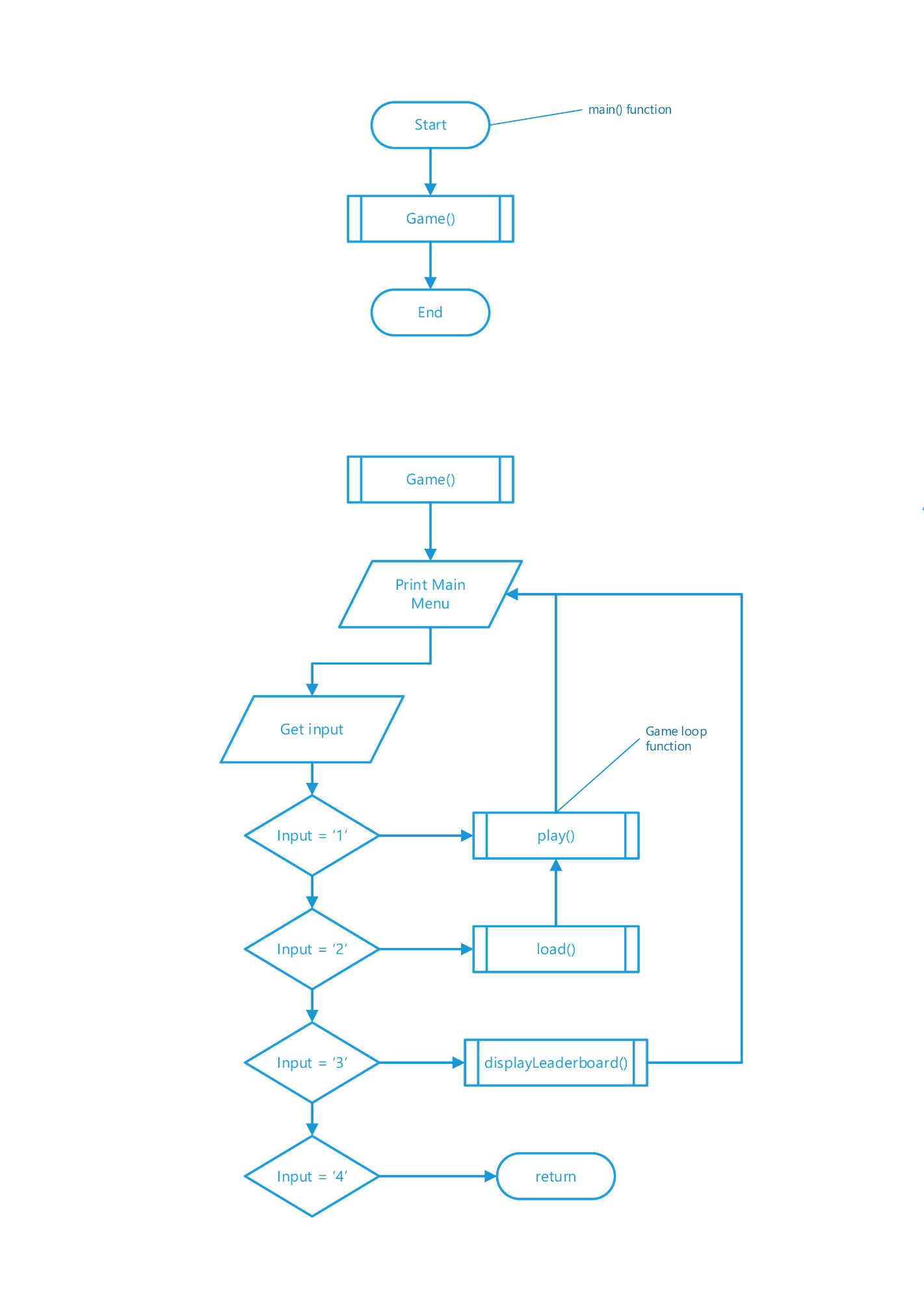
\* of PxPyPz where (x,y,z,..) are the indices of the PIs in the

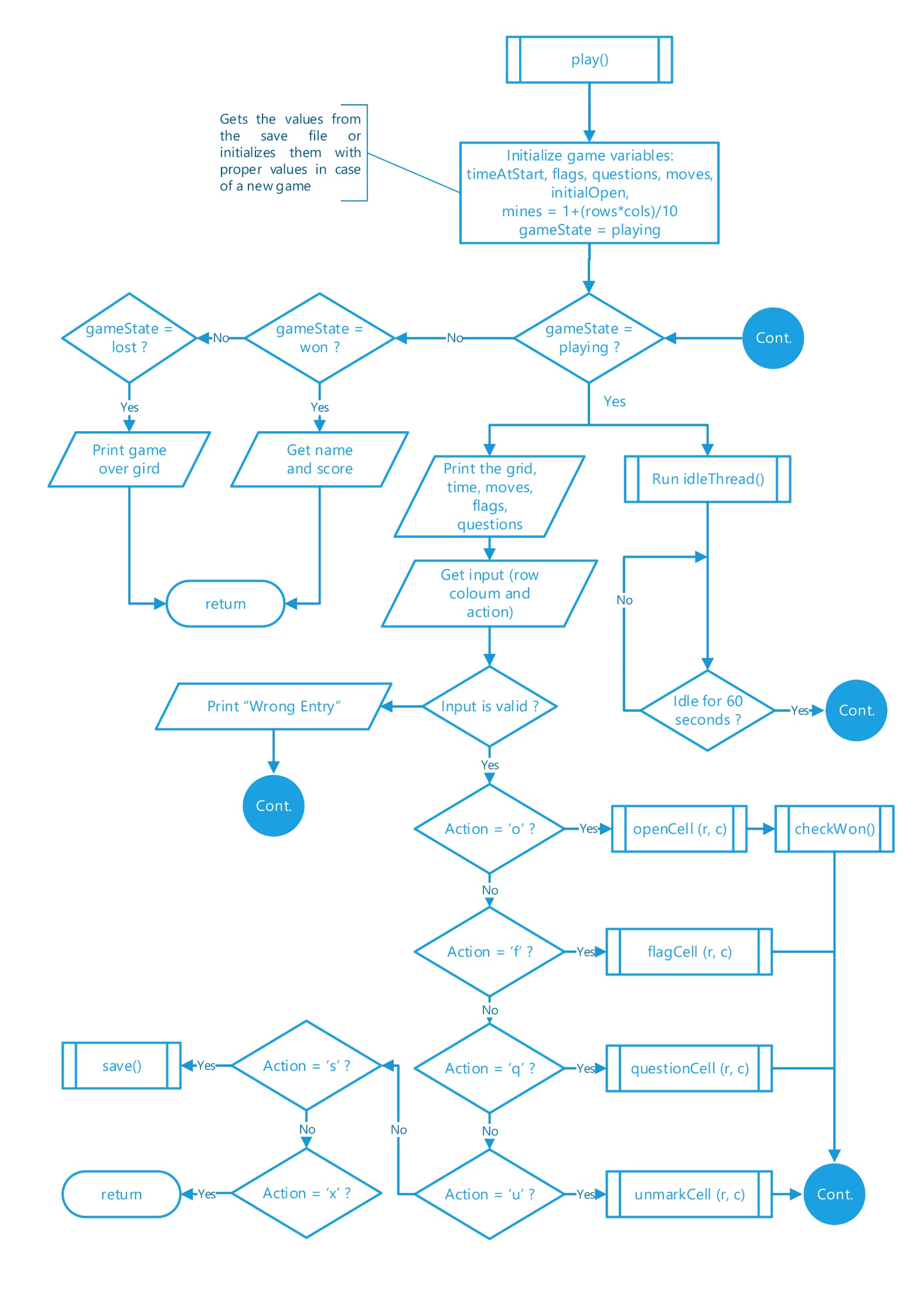
\* given primes array.

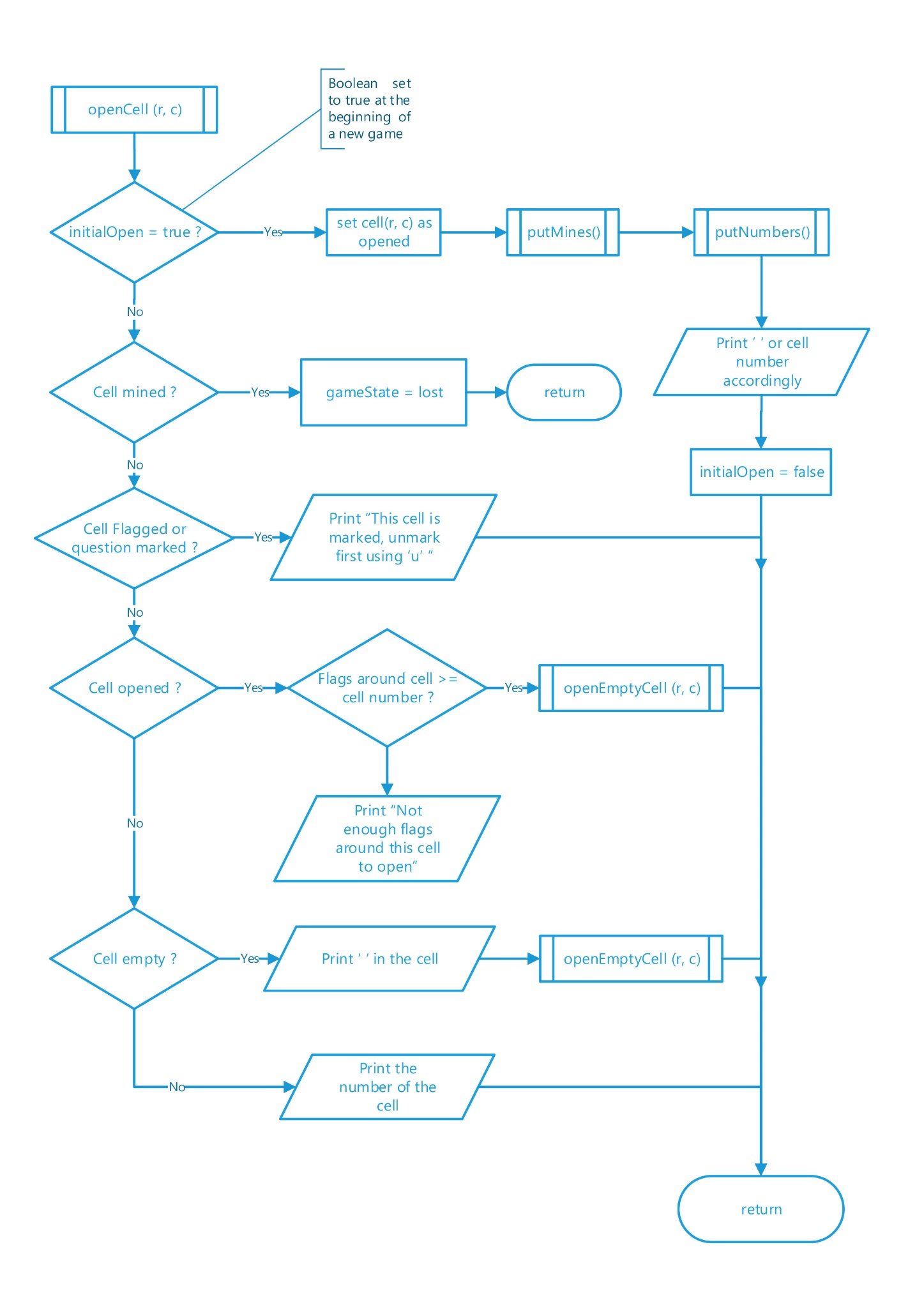
\*/

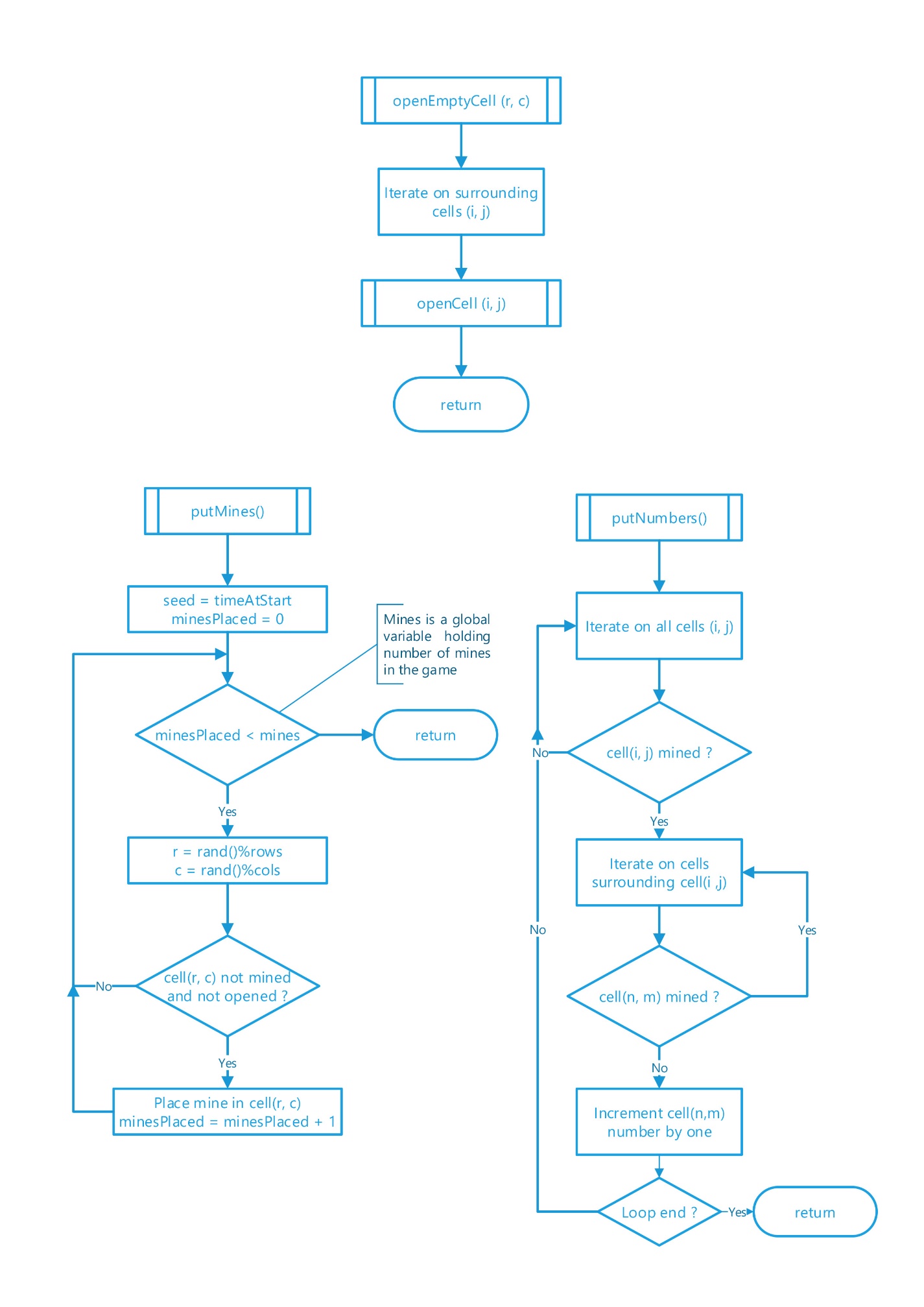
public SinglyLinkedList getSolutions(final DoublyLinkedList[] primes, final int[] minterms);

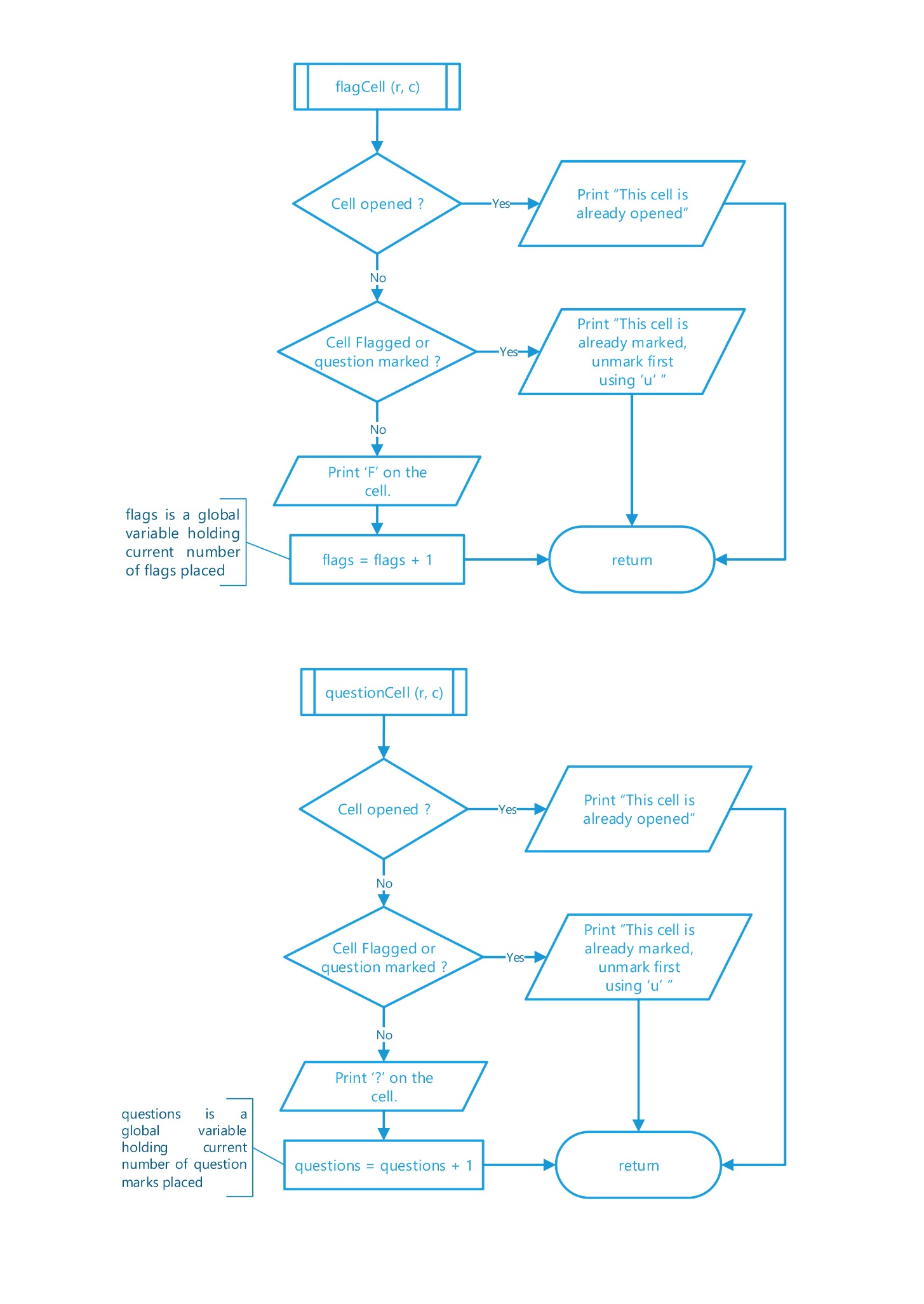
}

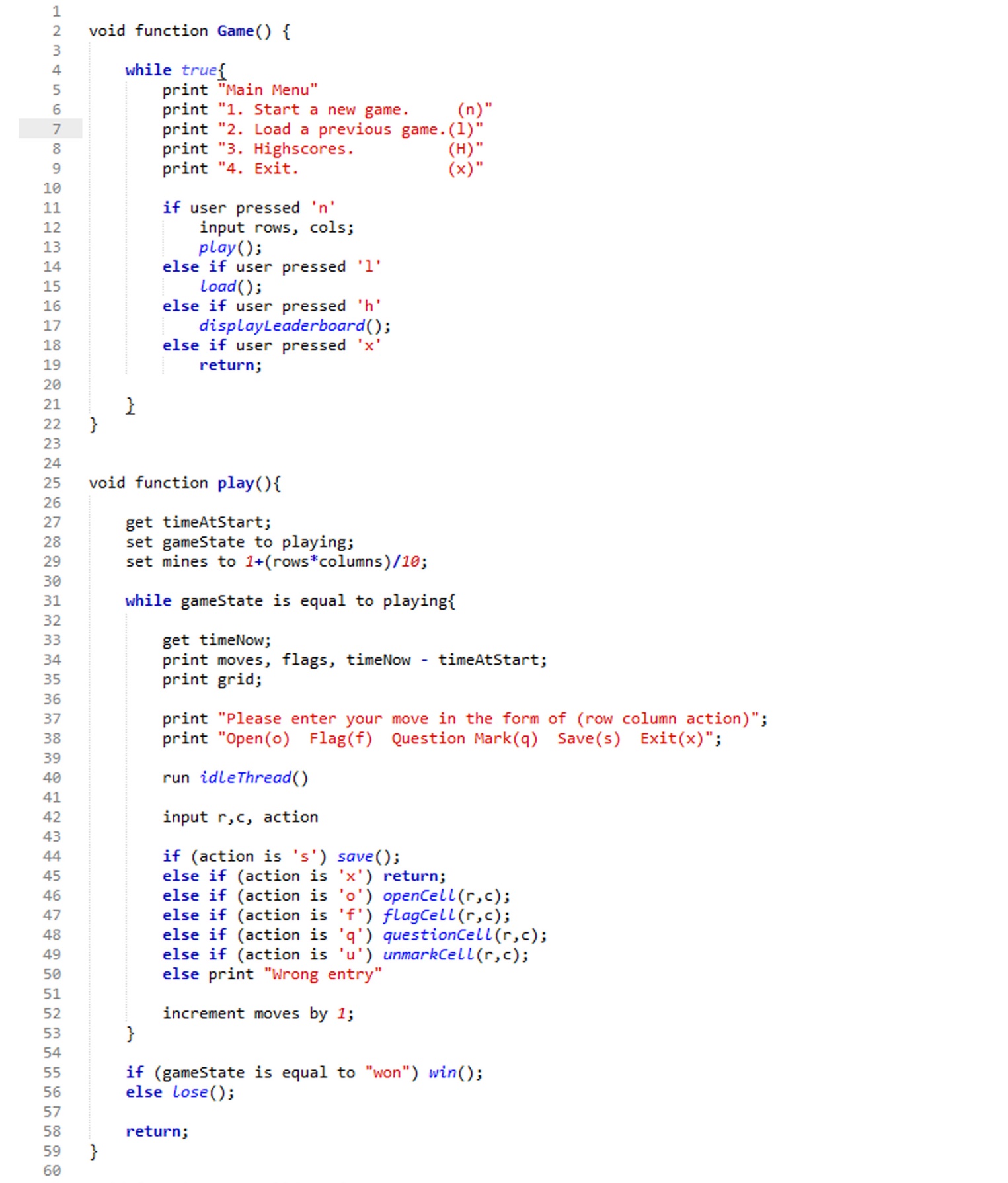
**Flowcharts**

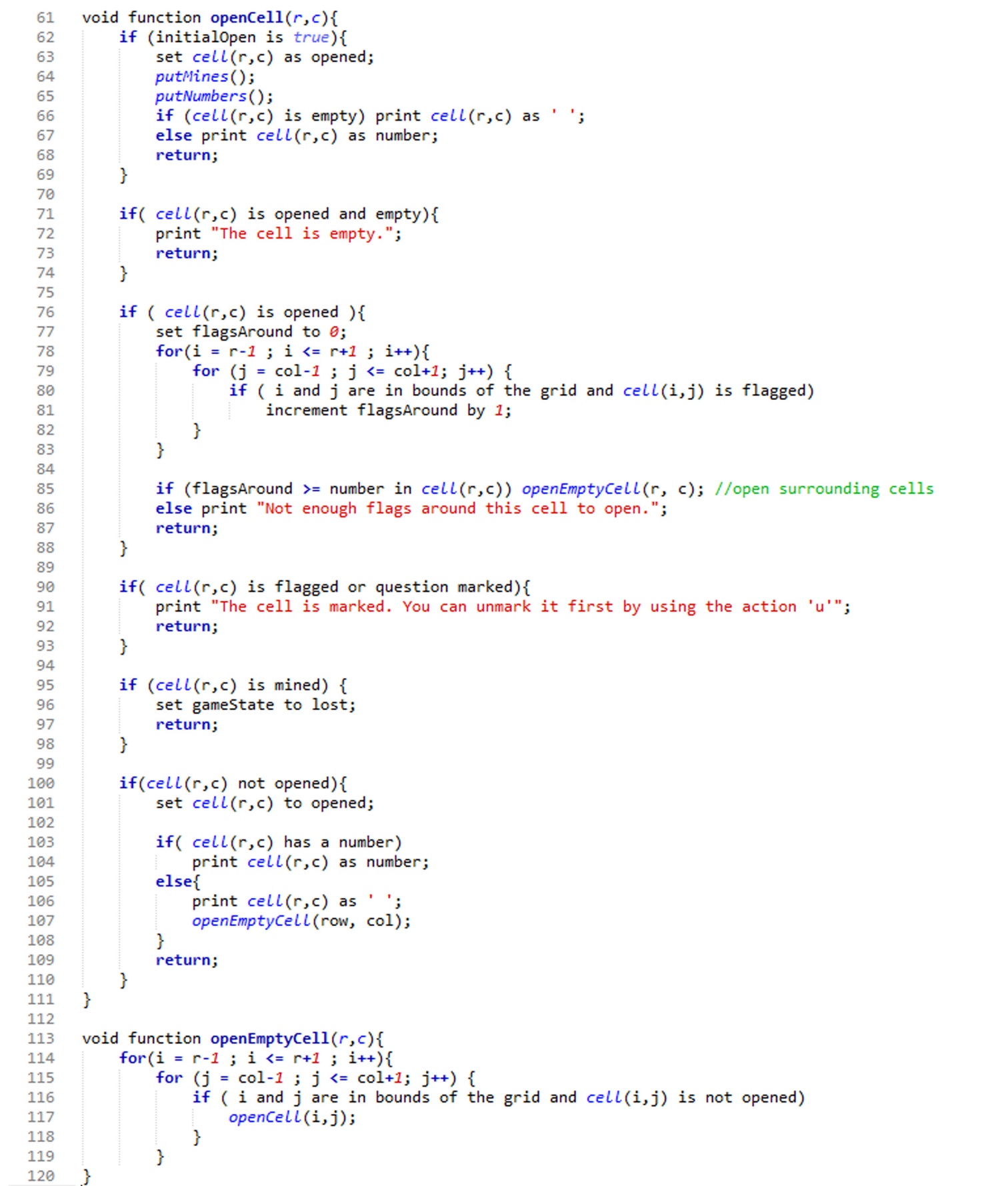
****

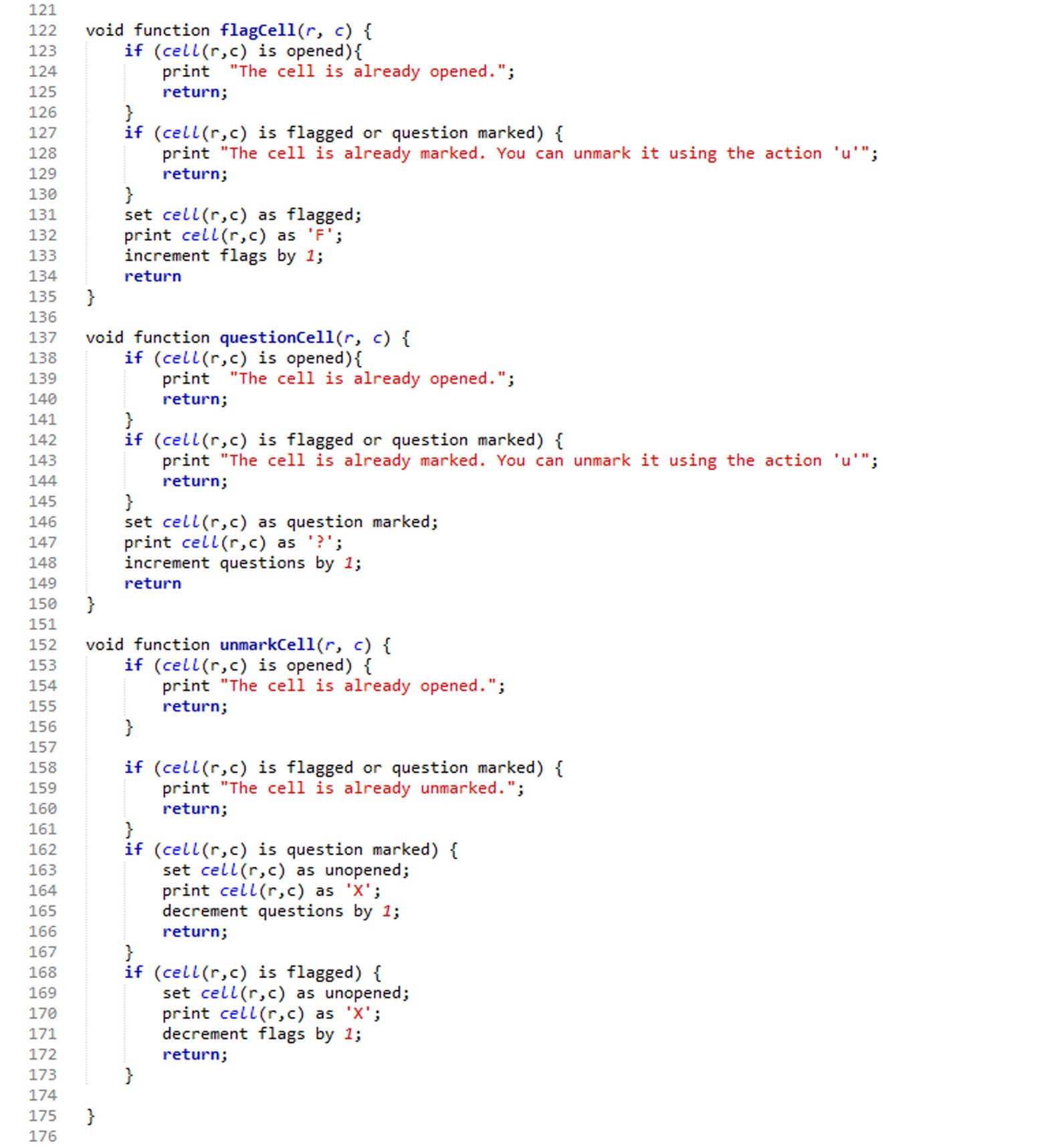
****

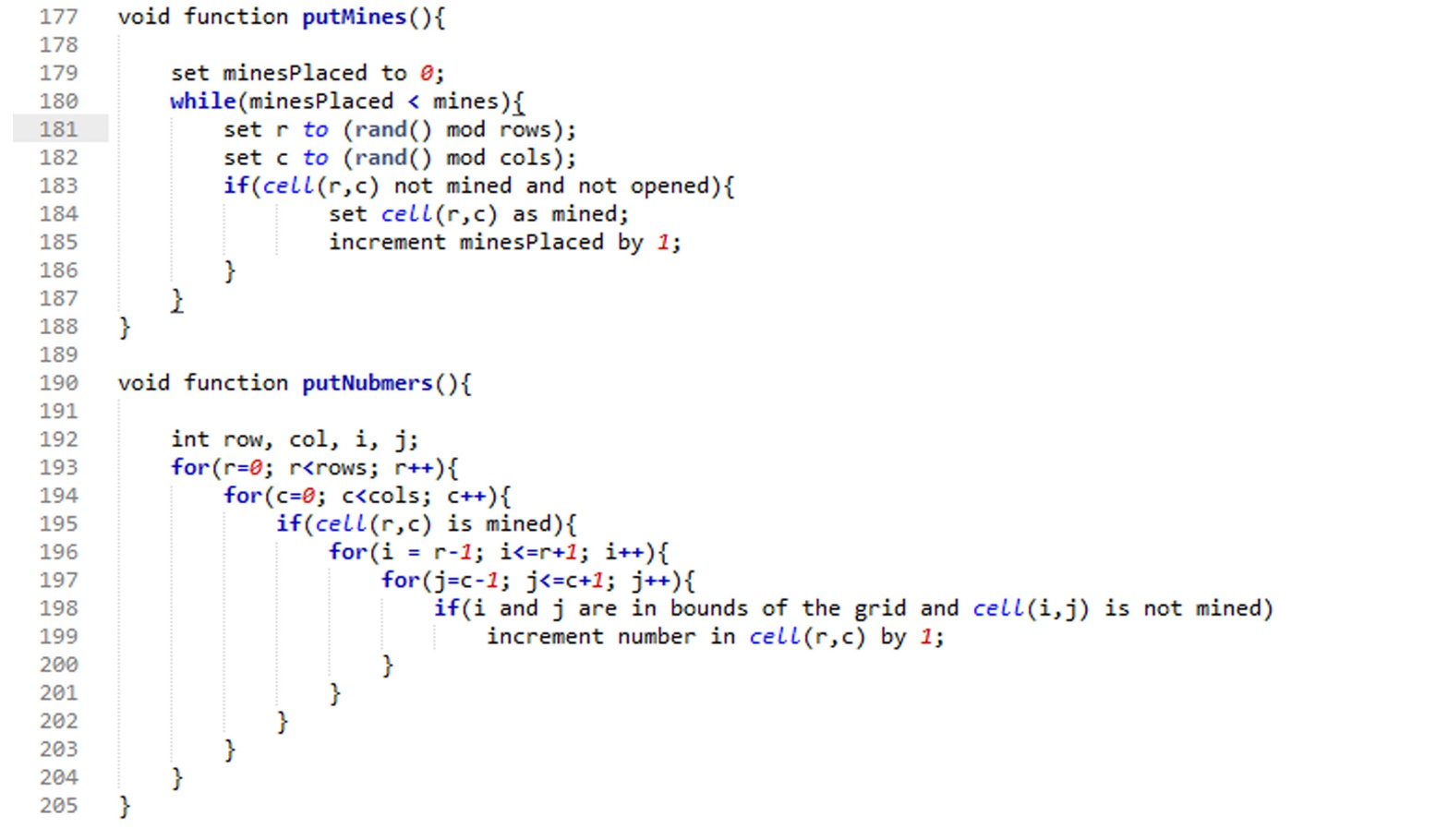
****

****

**Pseudocode**

****

****

****



**User Manual**

**Main Menu :**

In the main menu, you are asked to pick one of the following options :

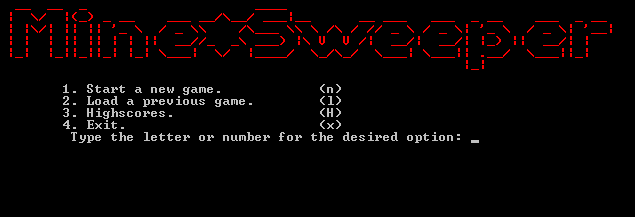
1. Start a new game. (n)

2. Load a previous game. (l)

3. Highscores. (H)

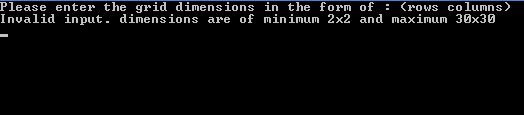
4. Exit.(x)

You may enter your desired letter (in capital or small) or enter the number of the desired option.

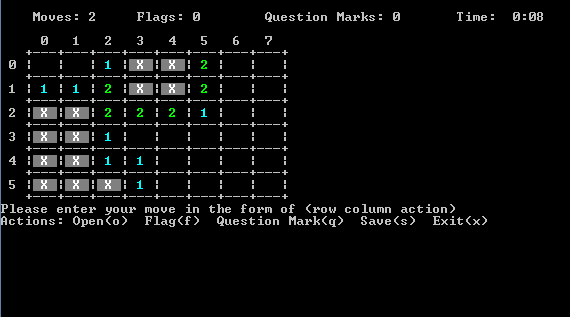


**Grid Dimensions :**

At the start of a new game, you are asked to enter the dimensions of the grid in the form of (rows columns). The game only accepts each dimension in the (2 : 30) range. If any incorrect range was used, the game shows an error message like the one in the following figure :



**Playing :**



On the top you can see the number of made moves, flags, question marks and the time in the form of minutes:seconds .

To make a move, write the number of the row followed by a space then the number of the column followed by another space then the letter of the action.

**Input :**

Actions are described as in the figure above.

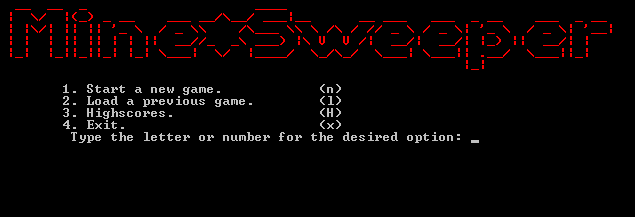
You may enter a small letter or a capital letter for the desired option or the full word excluding “exit” command which requires either ‘x’ or ‘X’ .

In saving and quitting, you are not obliged to enter a row or column, the program handles your input if it is just “s/S/Save/x/X” without a row or column in the beginning.

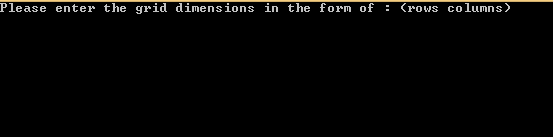
The program also handles malicious users and doesn’t crash under any input at any point in the game. It usually displays an error message and requires re-entry of your input.

**Sample Runs**

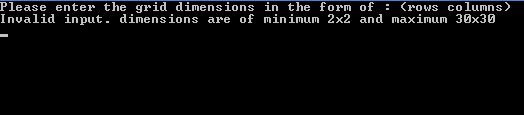
**Sample Run #1 :**



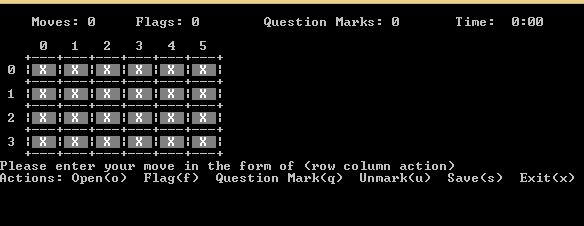
Input : N



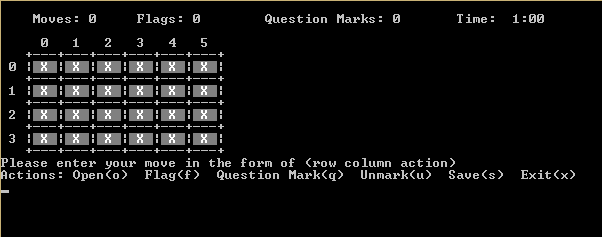
Input : 1 25



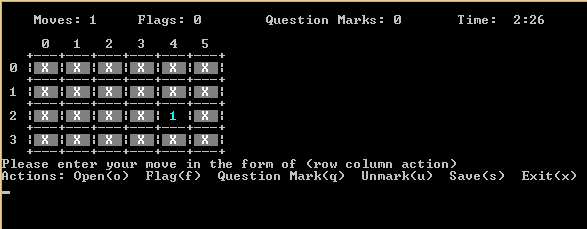
Input : 4 6



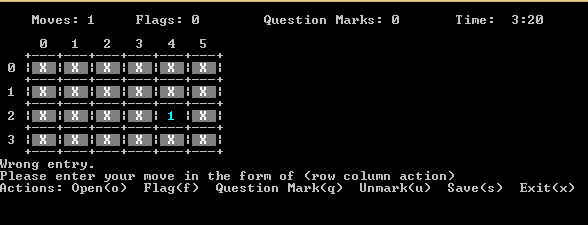
Idling for one minute..



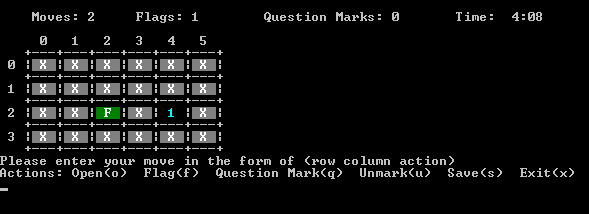
Input : 2 4 o



Input : 5 4 o

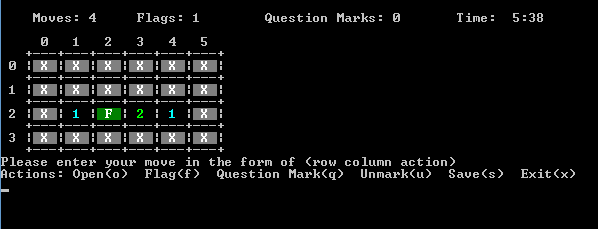


Input : 2 2 F



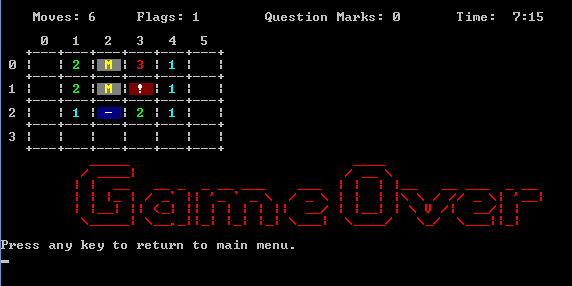
Input : 2 1 O

Input : 2 3 o

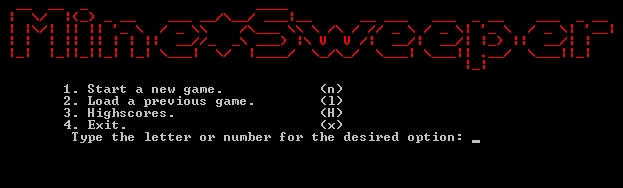


Input : 0 1 o

Input : 1 3 o



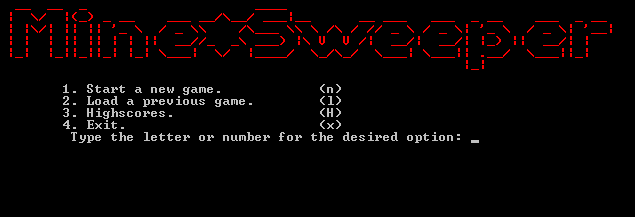
Input : 9



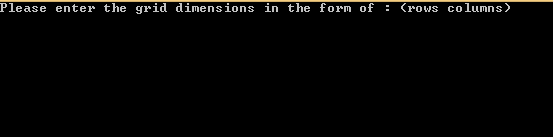
Input : x

*\*Program terminates\**

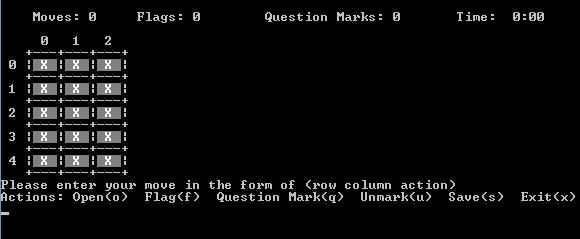
**Sample Run #2 :**



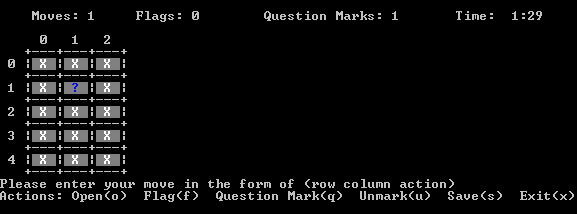
Input : 1



Input : 5 3

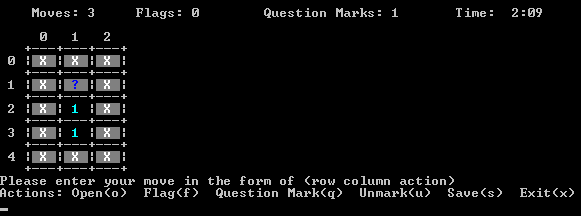


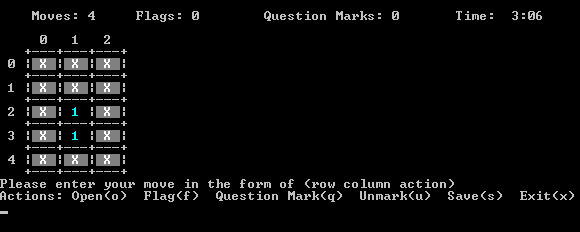
Input : 1 1 ?



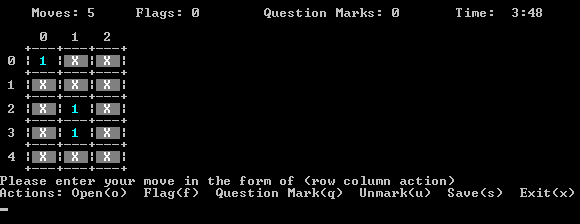
Input : 2 1 OPEN

Input : 3 1 o



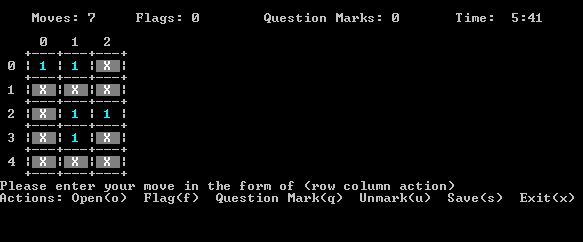
Input : 1 1 u

Input : 0 0 o

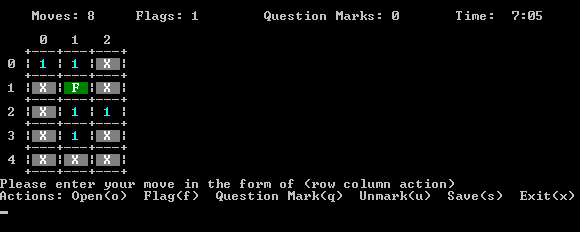


Input : 2 2 o

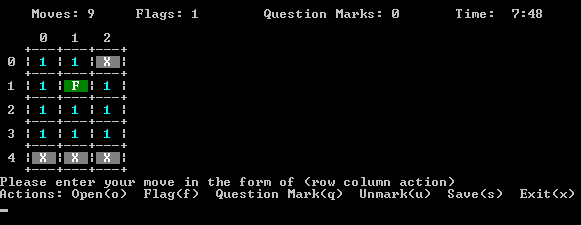
Input : 0 1 o

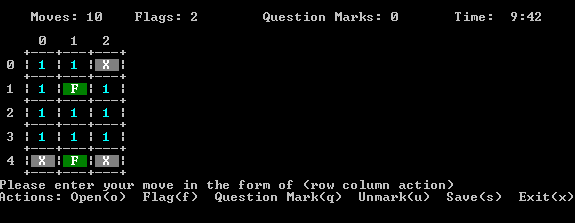


Input : 1 1 F

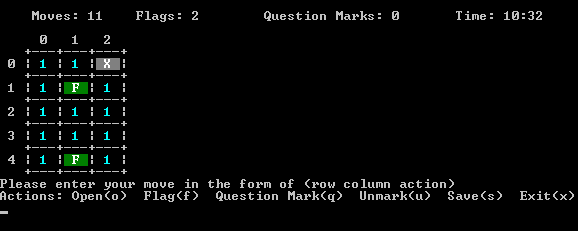


Input : 2 1 O



Input : 4 1 f

Input : 3 1 open



Input : 0 2 o



Input : Hesham Medhat Mahmoud :)



Hit : h *\*Program returns to main menu\**

Input : x

*\*Program terminates\**

**References**

Our reference is what we have been taught in lectures and through self study from various materials.

We personally have no tolerance to plagiarism.

No copied material was used in this project.

For uncertainty in syntax we looked back into the trusted websites :

[www.tutorialspoint.com](http://www.tutorialspoint.com)

[www.stackoverflow.com](http://www.stackoverflow.com)

*Thank you.*