CS 540-1: Introduction to Artificial Intelligence
Homework Assignment # 6

Assigned: 12/1
Due: 12/12 before 9:55 a.m.

Hand in your homework:

This homework includes a written portion and a programming portion. Please type the written portion and hand in the file in pdf format and name it as **WrittenPart.pdf**. The first page of the pdf must include a header with: **your name, Wisc username, class section, HW# , date and, if handed in late, how many days late it is**. The programming portion will be required to write in Java, and all files needed to run your program, including any support files but input/output/debug files, should be submitted. Finally, please create a folder named as <**Wisc username**>_**HW#**, put **all your codes** as well as **WrittenPart.pdf** into the folder and compress it as <**Wisc username**>_**HW#.zip**. This final zip file should then be uploaded to the appropriate place on the course Moodle website.

Late Policy:

All assignments are due at the beginning of class on the due date. One (1) day late, defined as a 24-hour period from the deadline (weekday or weekend), will result in 10% of the total points for the assignment deducted. So, for example, if a 100-point assignment is due on a Wednesday 9:55 a.m., and it is handed in between Wednesday 9:55 a.m. and Thursday 9:55 a.m., 10 points will be deducted. Two (2) days late, 25% off; three (3) days late, 50% off. No homework can be turned in more than three (3) days late. Written questions and program submission have the same deadline. A total of two (2) free late days may be used throughout the semester without penalty.

Assignment grading questions must be raised with the instructor within one week after the assignment is returned.

Collaboration Policy:

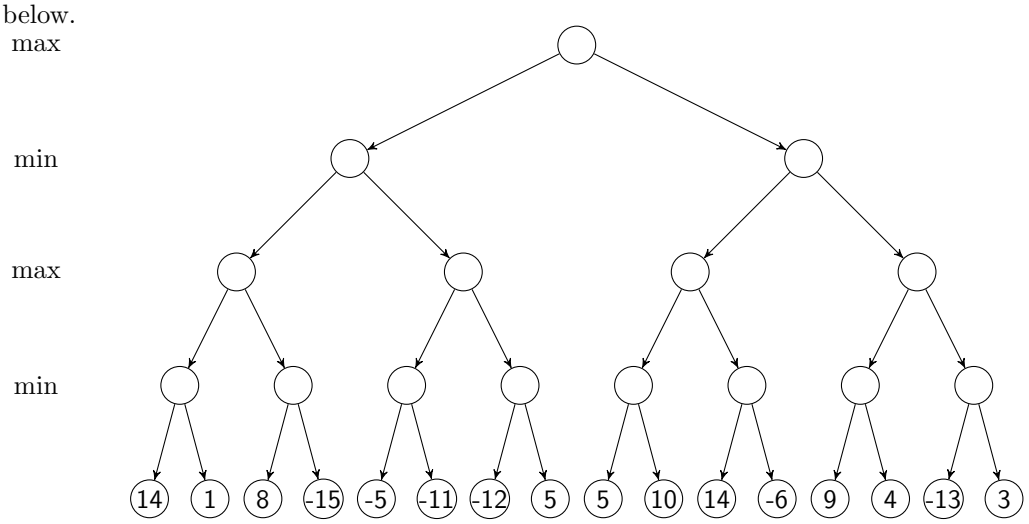
You are to complete this assignment individually. However, you are encouraged to You are to complete this assignment individually. However, you are encouraged to discuss the general algorithms and ideas with classmates, TAs, and instructor in order to help you answer the questions. You are also welcome to give each other examples that are not on the assignment in order to demonstrate how to solve problems. But we require you to:

- not explicitly tell each other the answers
- not to copy answers or code fragments from anyone or anywhere
- not to allow your answers to be copied
- not to get any code on the Web

In those cases where you work with one or more other people on the general discussion of the assignment and surrounding topics, we suggest that you specifically record on the assignment the names of the people you were in discussion with.

Question 1: Game Playing[25]

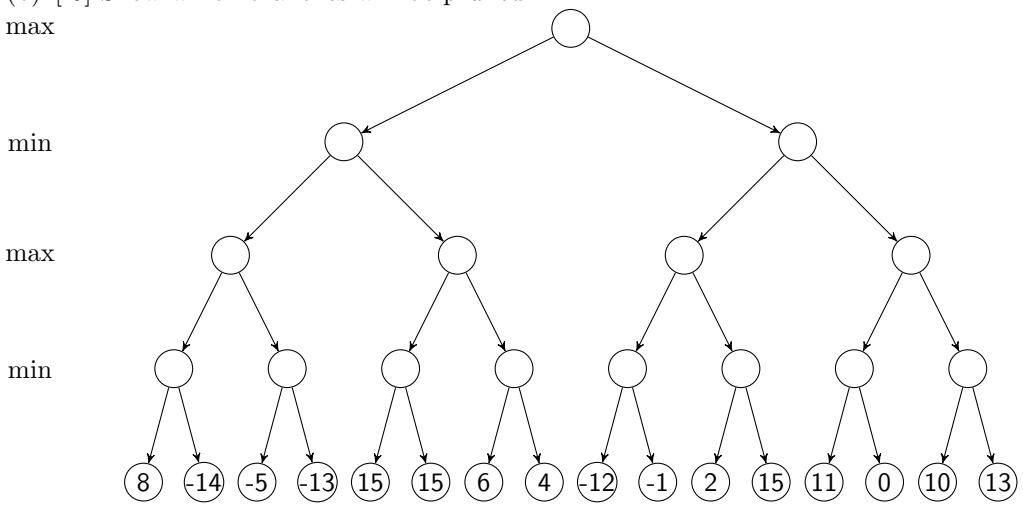
a) [10] Use the Minimax algorithm to compute the minimax value at each node for the game tree below.



b) [15] Use Alpha-Beta Pruning to compute the minimax value at each node for the game tree below, assuming children are visited left to right. You are asked to:

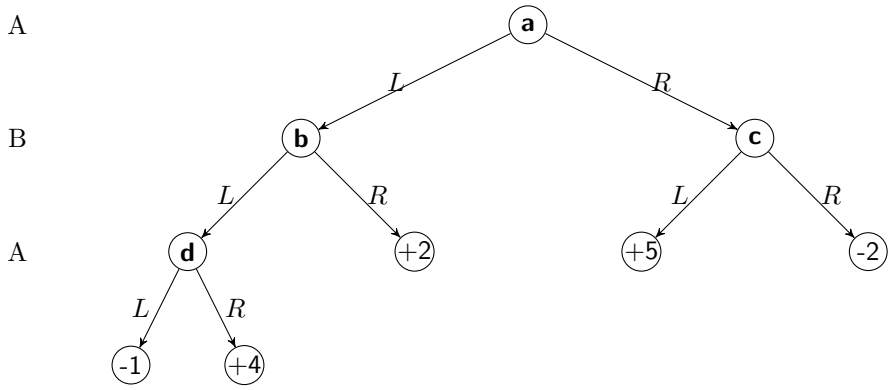
(a) [10] Show the alpha and beta values at each node.

(b) [5] Show which branches will be pruned.



Quesiton 2: Game Theory[25]

a) [10] Consider the following zero-sum game. You are asked to:
   (a) [5] Write down the pure strategies for player A and B (by denoting the choice for internal states a to d).
   (b) [5] Write down the matrix normal form of this game.



b) [15] The table below shows a matrix normal form of a non-zero game. The two numbers in each entry represent the gains for player A and B respectively. You are asked to:
   (a) [10] Apply iterative elimination of strictly dominated strategies to this matrix normal form.
   (b) [5] Show what strategies will player A and B choose in the end and explain the reason.

		B			
		I	II	III	IV
A	I	3,5	1,3	3,2	8,3
	II	6,5	1,4	4,8	1,3
	III	7,9	9,5	2,6	3,2
	IV	3,9	6,2	3,6	5,4

Question 3: Factor-Multiples Game[50]

In this question, your task is to implement Minimax algorithm to solve a game problem, the Factor-Multiples game. The algorithm has two functions: the Max-value function and the Min-value function. You will have to implement both of these functions for this game.

Game description

Factor-Multiples is a two player game. The game starts with a list of n integers ranging from $1 \ldots n$. Players take turns to cross out one number from the list. There are some restrictions on which numbers can be crossed out during a given move. The restrictions are as follows:

- During the first move the first player chooses an even number which is **less than** $\frac{n}{2}$ to cross out. For example if $n = 9$ then the legal numbers for the first move are 2 and 4. If $n = 8$ then the only legal number for the first move is 2. This number is saved as the $lastMove$.
- During subsequent moves each players take turns. The number that a player can cross out should be a multiple or factor of the $lastMove$ (1 is a factor of all other numbers). Also this number may not be one of those that have already been crossed out. After crossing out, the number is saved as the new $lastMove$.

When a player cannot cross out a number during his move, he looses the game.

An example of the game play is given below.

Sample Input

```
n=7
```

Sample Output

```
Player 1: 2
Player 2: 4
Player 1: 1
Player 2: 7
Winner: Player 2
```

Methods to implement

In this programming question, you are only required to implement four methods of the class PlayerImpl, which are showed as follows:

```
public class PlayerImpl implements Player {
       public ArrayList<Integer> generateSuccessors(int lastMove, int [] crossedList);
       public int max_value(GameState s);
       public int min_value(GameState s);
       public int move(int lastMove, int [] crossedList);

}
```

A description of the methods is given below:

- `public ArrayList<Integer> generateSuccessors(int lastMove, int [] crossedList)`: `lastMove` is the number that was crossed out during the previous move. If it is the first move, then this value would be equal to -1. `crossedList` is an integer array that identifies which numbers have been

crossed out from the list. If `crossedList[i]`$\neq$`0` then the number i has already been crossed out. Thus the length of the `crossedList` will be $(n+1)$, with `crossedList[0]` **never used**. You can also access n from the member variable `n` of the class `PlayerImpl`. Given these two inputs, this method returns an `ArrayList` of integers which are next valid numbers that can be crossed out.

- `public int max_value(GameState s)`: This method takes a `GameState` as its input. The pseudo code for this method can be found in Professor Zhu's slide. It returns the best game-theoretical value with respect to the max player. It should also update the object `s`. If you check `GameState.java`, you will find that the class has four variables. Before invoking this method, you should assign values to two of them, namely `crossedList` and `lastMove`. The method should update the other two variables of this object. `crossedList` is the same variable as described before. The constructor of `GameState` uses a clone of the `crossedList`. So, you do not have to worry about it when you manipulate the `crossedList` within a `GameState` object. The variable `leaf` identifies if the given state is a leaf or not. It is your task to update this variable when necessary. `lastMove` is the number which if crossed out during the previous move will lead to this `GameState`. If no number is yet to be crossed out, then this value is -1. `bestMove` is the number that when crossed out will yield the best value for the player. It should be noted that multiple numbers might yield the best value. In that case, **choose the maximum of those numbers**. It is also possible that after reaching a certain game state, the player has no choice but to loose. In that case, all next possible moves will have a game-theoretical value of -1. Even in that case, **choose the maximum of those numbers as the** `bestMove`. Moreover, if `s` turns out to be a leaf, then this method should return -1.
- `public int min_value(GameState s)`: The pseudocode for this method can also be found in Professor Zhu's slide. And just like the previous method, it returns the best game-theoretical value with respect to the min player. It should also update the object `s` as before. Moreover, if `s` turns out to be a leaf, then this method should return 1.
- `public int move(int lastMove, int [] crossedList)`: This is the upper level method which is used to determine the actual next move of a player. Given the two inputs, it is the task of this method to return the number which if crossed out gives the highest game-theoretical value. This method should use the `max_value` method to achieve this. Whenever a player decides on the next best move, he considers himself as the max player. That is why you should call `max_value` method within this method. The other player then automatically become the min player and his method gets called from within the `max_value` method. If no numbers can be crossed out then the `move` method should return -1.
  This method is called whenever a player has to make a move. The game tree generated by this method is not saved in memory. During each call of the method, the game tree is generated again. But each game tree is rooted at a different game state during different method invocation.

The I/O parts have already been written for you. So, you are **NOT** responsible for any console input or output. You are only responsible for implementing these four methods.

How to test

We will test your program on multiple test cases where we will vary the input n, and the format of testing commands will be like this:

```
java HW6 <n> <modeFlag>
```

where `n` is the value of n in the Factor-Multiples game, and `modeFlag` is an integer ranging from 1 to 3, controlling which types of players will play the game. Your program should give the moves of the players and the winner as output: Also we will not test for $n > 30$.

When `modeFlag`=1, both players are computers and you will see a game played between them. When `modeFlag`=2, the first player is human and the second player is the computer. When `modeFlag`=3, the

first player is the computer and the second player is human. We will only test for `modeFlag`=1. For the other two modes you can play against your own code and see that if you can win!!!

So an example command for testing the code could be:

```
java HW6 7 1
```

As part of our testing process, we will unzip the file you return to us, remove any class files, call `javac *.java` to compile your code, and call the main method of HW6 with certain parameters. Make sure that your code runs on one of the computers in the department because we will conduct our test on such computers.

Deliverables

1. Hand in (see the cover page for details) your modified version of the code skeleton we provided you with your implementation of the Minimax algorithm. Also include any additional java class files needed to run the program.
2. Optionally, in the written part of your homework, add any comments about the program that you would like us to know.

Partial Grading Scheme

1. Correct implementation of the `generateSuccessors` method [20]
2. Correct implementation of the `max_value` method [10]
3. Correct implementation of the `min_value` method [10]
4. Correct implementation of the `move` method [10]