BinaryQuestions

By Kamiel de Visser | 500838438

Table of contents

Introduction

“Download the source for the game “BinaryQuestions” from the DLO, and open it in your IDE. In this game, you have to think of an object, person or animal. The program will try to guess what you are thinking of by asking questions. Every time you play the game, the AI will learn new information. Its knowledge is stored in a binary tree. If the program is closed, this tree is serialized and stored to a file. This way, the knowledge tree is saved when you close the program, and reopened when you start it again. “

In this document I will describe my thought process whilst completing the BinaryQuestions assignment.
I will do this on a step by step basis.
We were provided a start project in which we had to implement a few basic artificial intelligence concepts.

This assignment was done during the Artificial Intelligence course at the HvA.

The assignment

Question 1

Describe how the game works, by playing it several times.

The game starts off by 'asking' the user to think of a particular object, person or animal. It's goal is to guess whichever object/person/animal the user was thinking of by asking a maximum of 20 yes/no questions.

In reality the game isn't 'asking' anything, but is simply displaying the `message` of a `BTNode` object. A `BTNode` represents a Binary Tree Node.
A `BTNode` can either hold a

- String representing a question + two pointers to this BTNode's children.
- (In case of being a sub-tree)

or a

- String representing a particular object, person or animal + two empty pointers.
- (In case of being a leaf-node)

These `BTNode` objects can be used to make up a Binary Tree, which is a tree-type data structure where each node has at most two children.

The game uses this structure and these two forms a `BTNode` can take on to create a binary tree containing questions which can be entered with either a yes or a no. Answering yes or no to a question will lead you to the next subtree containing a yes/no question. This goes on and on until we run out of subtrees.
Whenever a tree has no more sub-trees (this is when we're at the so-called 'leaf-node'), instead of a question the node will contain a particular object.
This creates a situation where each leaf node's object/person/animal is accessed by answering either yes/no to it's parent question, and it's parent question is accessed by answering *their* parent's question and so on.

At the very top of this structure we'll find a question which the user is initially asked. i.e. '*Is it red?*' Their answer to this question then leads them down either the *yes* or *no* edge to another question. This may lead them to another question like *'Does it have wheels?*' which they again answer yes or no to. Eventually either 20 questions will have been asked, at which point 'the game loses' or a leaf node will have been reached, at which point the game asks
'*Are you thinking of a(n)* `<leaf node's message>`'.

The user can then again answer with either a yes (at which point ‘the game wins’) or a no (at which point ‘the game admits it’s defeat’ and asks the user to input the object/person/animal they were thinking of, along with a question that distinguishes it from the `<leaf node's message>` and the answer to that question, and expands its tree with this new user-created node.

Example output of the program:

```
Starting the "20 Binary Questions" Game!
Think of an object, person or animal.
1) Is it red?
Enter 'y' for yes and 'n' for no: y
2) Does it have wheels?
Enter 'y' for yes and 'n' for no: n
3) Is it a game character?
Enter 'y' for yes and 'n' for no: n
4) Is it an animal?
Enter 'y' for yes and 'n' for no: n
5) Are you thinking of a(n) Strawberry?
Enter 'y' for yes and 'n' for no: n
You win! What were you thinking of? Apple
Please enter a question to distinguish a(n) Strawberry from Apple: Does it grow on trees?
If you were thinking of a(n) Apple, what would the answer to that question be ('yes' or 'no')? yes
Thank you! My knowledge has been increased
Play Another Game? y

Think of an object, person or animal.

1) Is it red?
Enter 'y' for yes and 'n' for no: y
2) Does it have wheels?
Enter 'y' for yes and 'n' for no: n
3) Is it a game character?
Enter 'y' for yes and 'n' for no: n
4) Is it an animal?
Enter 'y' for yes and 'n' for no: n
5) Does it grow on trees?
Enter 'y' for yes and 'n' for no: y
6) Are you thinking of a(n) Apple?
Enter 'y' for yes and 'n' for no: y
I Win!
```

Question 2

Remove the existing knowledge tree by removing the serialization file from your file system. Then build up a new knowledge tree by playing the game for 8 questions.

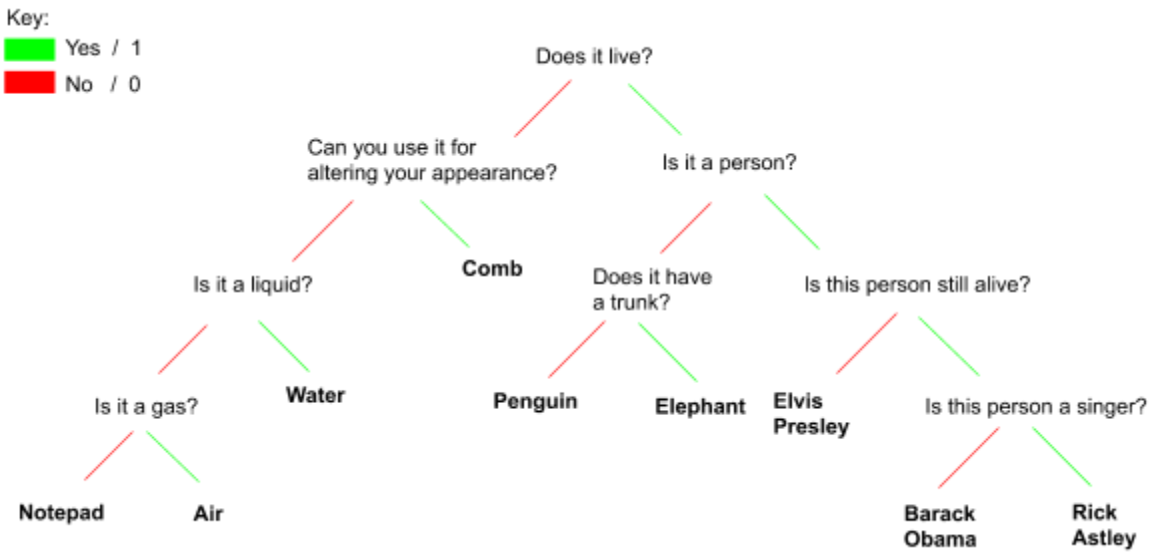
These are the logs from the session where I build up the knowledge tree. Possible items have been marked in green. Questions are marked yellow.

```
No previous knowledge found!
Initializing a new game.

Enter a question about an object, person or animal:
Does it live?
Enter a possible guess (an object, person or animal) if the response to this question is Yes: Elephant
Enter a possible guess (an object, person or animal) if the response to this question is No: Comb

Starting the "20 Binary Questions" Game!
Think of an object, person or animal.
1) Does it live?
Enter 'y' for yes and 'n' for no: y
2) Are you thinking of a(n) Elephant?
Enter 'y' for yes and 'n' for no: n
You win! What were you thinking of? Rick Astley
Please enter a question to distinguish a(n) Elephant from Rick Astley: Is it a person?
If you were thinking of a(n) Rick Astley, what would the answer to that question be ('yes' or 'no')? yes
Thank you! My knowledge has been increased
Play Another Game? y

Think of an object, person or animal.

1) Does it live?
Enter 'y' for yes and 'n' for no: no
2) Are you thinking of a(n) Comb?
Enter 'y' for yes and 'n' for no: no
You win! What were you thinking of? Notepad
Please enter a question to distinguish a(n) Comb from Notepad: Can you use it for altering your appearance?
If you were thinking of a(n) Notepad, what would the answer to that question be ('yes' or 'no')? no
Thank you! My knowledge has been increased
Play Another Game? y

Think of an object, person or animal.

1) Does it live?
Enter 'y' for yes and 'n' for no: n
2) Can you use it for altering your appearance?
Enter 'y' for yes and 'n' for no: n
3) Are you thinking of a(n) Notepad?
Enter 'y' for yes and 'n' for no: n
You win! What were you thinking of? Water
Please enter a question to distinguish a(n) Notepad from Water: Is it a liquid?
If you were thinking of a(n) Water, what would the answer to that question be ('yes' or 'no')? yes
Thank you! My knowledge has been increased
Play Another Game? y

Think of an object, person or animal.

1) Does it live?
```

```
Enter 'y' for yes and 'n' for no: y
2) Is it a person?
Enter 'y' for yes and 'n' for no: y
3) Are you thinking of a(n) Rick Astley?
Enter 'y' for yes and 'n' for no: n
You win! What were you thinking of? Elvis Presley
Please enter a question to distinguish a(n) Rick Astley from Elvis Presley: Is this person still alive?
If you were thinking of a(n) Elvis Presley, what would the answer to that question be ('yes' or 'no')? no
Thank you! My knowledge has been increased
Play Another Game? y

Think of an object, person or animal.

1) Does it live?
Enter 'y' for yes and 'n' for no: y
2) Is it a person?
Enter 'y' for yes and 'n' for no: n
3) Are you thinking of a(n) Elephant?
Enter 'y' for yes and 'n' for no: n
You win! What were you thinking of? Penguin
Please enter a question to distinguish a(n) Elephant from Penguin: Does it have a trunk?
If you were thinking of a(n) Penguin, what would the answer to that question be ('yes' or 'no')? no
Thank you! My knowledge has been increased
Play Another Game? y

Think of an object, person or animal.

1) Does it live?
Enter 'y' for yes and 'n' for no: n
2) Can you use it for altering your appearance?
Enter 'y' for yes and 'n' for no: n
3) Is it a liquid?
Enter 'y' for yes and 'n' for no: n
4) Are you thinking of a(n) Notepad?
Enter 'y' for yes and 'n' for no: n
You win! What were you thinking of? Air
Please enter a question to distinguish a(n) Notepad from Air: Is it a gas?
If you were thinking of a(n) Air, what would the answer to that question be ('yes' or 'no')? yes
Thank you! My knowledge has been increased
Play Another Game? y

Think of an object, person or animal.

1) Does it live?
Enter 'y' for yes and 'n' for no: y
2) Is it a person?
Enter 'y' for yes and 'n' for no: y
3) Is this person still alive?
Enter 'y' for yes and 'n' for no: y
4) Are you thinking of a(n) Rick Astley?
Enter 'y' for yes and 'n' for no: n
You win! What were you thinking of? Barack Obama
Please enter a question to distinguish a(n) Rick Astley from Barack Obama: Is this person a singer?
If you were thinking of a(n) Barack Obama, what would the answer to that question be ('yes' or 'no')? no
Thank you! My knowledge has been increased
Play Another Game?
```

Show the binary tree that contains the current knowledge of the program

.
The binary tree produced by above inputs can be represented as seen here:



Question 3

Write three functions that print the preorder, in order, and post order traversals of your built-up tree to the console.

Preorder traversal works by traversing the root, then traversing the left subtree, and finally traversing the right subtree.

So we basically want to grab the very root of our binary tree, which is stored in the `BTTree` created by deserializing the `.bin` file. Then we want to view that root node's message and consequently traverse all the way down the left side of that tree before returning and traversing the right side of the tree.

So we make a simple getter which returns the root node of this tree. We then pass the root node to a recursive function called TraversePreOrder, which first checks whether the passed node exists. If it does, it will print the node's message after which it recursively calls itself, now passing the left (or no / 0) node as the root node of the new subtree. If a passed node turns out to not exist, the function then returns to the previous call in the stack. Once the left subtree has been fully traversed in preorder, the function then passes the right (or yes / 1) node as the root node.

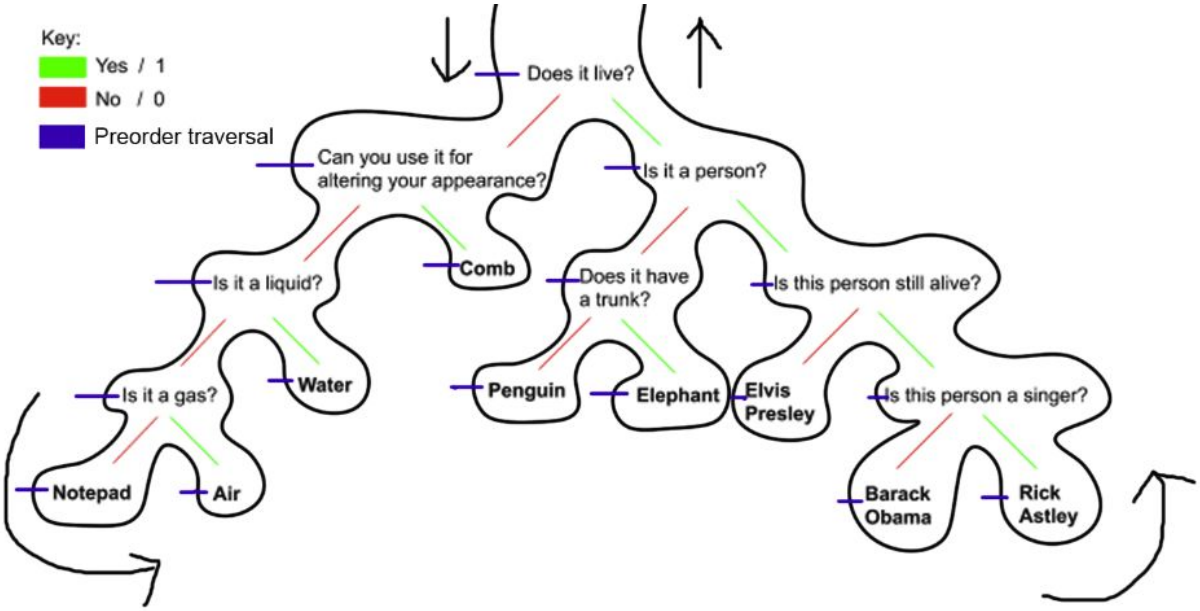
```
static void TraversePreOrder(BTNode rootNode)
{
	if (rootNode != null)
	{
		Console.WriteLine(rootNode.GetMessage());
		TraversePreOrder(rootNode.GetNoNode());
		TraversePreOrder(rootNode.GetYesNode());
	}
}
```

We'll use this function to print out the preoroder traversal by calling it with `TraversePreOrder(tree.GetRootNode())`

Running this function will print out:

```
Does it live?
Can you use it for altering your appearance?
Is it a liquid?
Is it a gas?
Notepad
Air
Water
Comb
Is it a person?
Does it have a trunk?
Penguin
Elephant
Is this person still alive?
Elvis Presley
Is this person a singer?
Barack Obama
Rick Astley
```

We can check whether this is correct by drawing the traversal route along our tree and looking at the order in which we pass items on the left side (marked by blue):



So we can conclude that our function is behaving as intended.

The inorder and postorder functions work on the same principle, only the order in which the message is printed and the left/right tree is traversed differs.

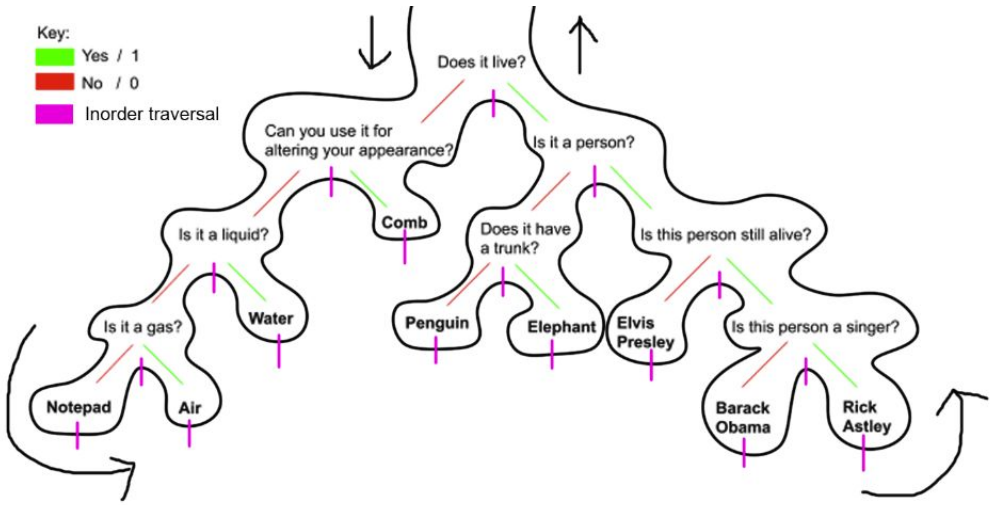
Inorder first traverses the left subtree, then prints the root message, then traverses the right subtree.

```
static void TraverseInOrder(BTNode rootNode)
{
    if (rootNode != null)
    {
        TraverseInOrder(rootNode.GetNoNode());
        Console.WriteLine(rootNode.GetMessage());
        TraverseInOrder(rootNode.GetYesNode());
    }
}
```

This prints:

```
Notepad
Is it a gas?
Air
Is it a liquid?
Water
Can you use it for altering your appearance?
Comb
Does it live?
Penguin
Does it have a trunk?
Elephant
Is it a person?
Elvis Presley
Is this person still alive?
Barack Obama
Is this person a singer?
Rick Astley
```

We can check that the above output is correct by looking at the order in which we pass items on the bottom:



And finally we have the post order traversal.

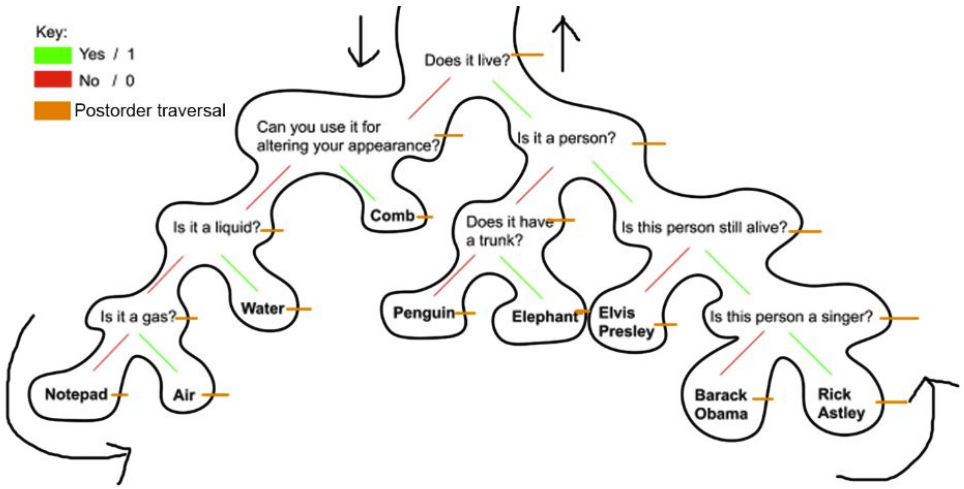
Postorder first traverses the left subtree, then traverses the right subtree after which it finally prints the root message.

```
static void TraversePostOrder(BTNode rootNode)
{
    if (rootNode != null)
    {
        TraversePostOrder(rootNode.GetNoNode());
        TraversePostOrder(rootNode.GetYesNode());
        Console.WriteLine(rootNode.GetMessage());
    }
}
```

This prints:

```
Notepad
Air
Is it a gas?
Water
Is it a liquid?
Comb
Can you use it for altering your appearance?
Penguin
Elephant
Does it have a trunk?
Elvis Presley
Barack Obama
Rick Astley
Is this person a singer?
Is this person still alive?
Is it a person?
Does it live?
```

We can check that the above output is correct by looking at the order in which we pass items on the right:



Question 4

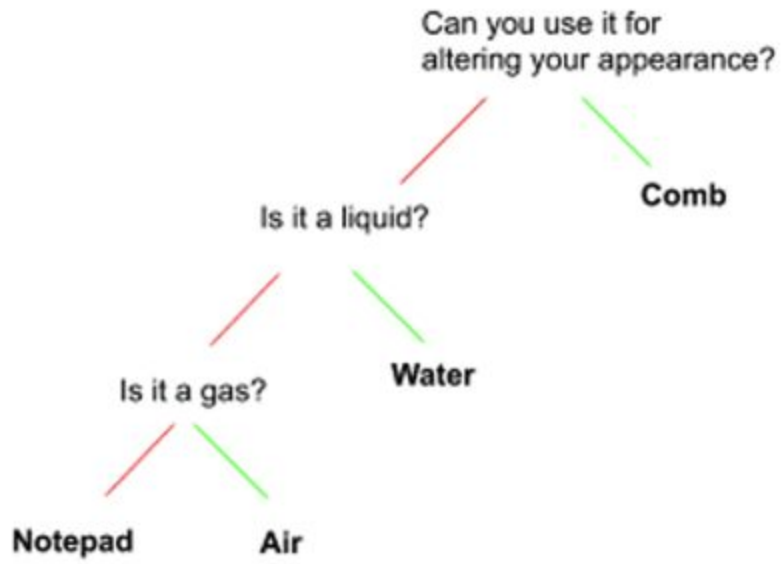
Is your tree balanced? Explain why or why not.

This tree is unbalanced.

The rules for determining whether a binary tree is height balanced are as followed: [1]

1) An empty tree is height-balanced
2) A non-empty tree is balanced if:
    a) The left subtree is balanced
    b) The right subtree is balanced
    c) The difference between heights of the left and right subtree is not more than 1

If we follow these rules, each subtree in our tree must be balanced. We can quickly see that the following subtree is not balanced, as the height difference between the rightmost leaf and the leftmost leaf is more than 1:



Since all subtrees must be balanced for our tree to be balanced, we can conclude that our tree is not balanced.

[1] Source: https://www.geeksforgeeks.org/how-to-determine-if-a-binary-tree-is-balanced/ (at 01/12/2020)

Question 5

Write an evaluation function that calculates the value of a leaf node according to the evaluation function (count the number of letters in the message).

We simply have to return the `message.Length` of a node's message to evaluate it, should the message exist. Messages should in theory always exist for a node object, but we throw an `ArgumentException` in the case that the message was missing from a node.

Note that this counts any characters as a letter, including whitespace. This doesn't really matter for this assignment though, as the values are arbitrary anyway.

```
static int Evaluate(BTNode node)
{
    string message = node.GetMessage();
    if (message != null)
    {
        return message.Length;
    }
    throw new ArgumentException("Tried to evaluate a null string");
}
```

Question 6

Write a (recursive) Minimax function that outputs the value of the tree, assuming the root node should be maximized.

In a Minimax function, it's important to know

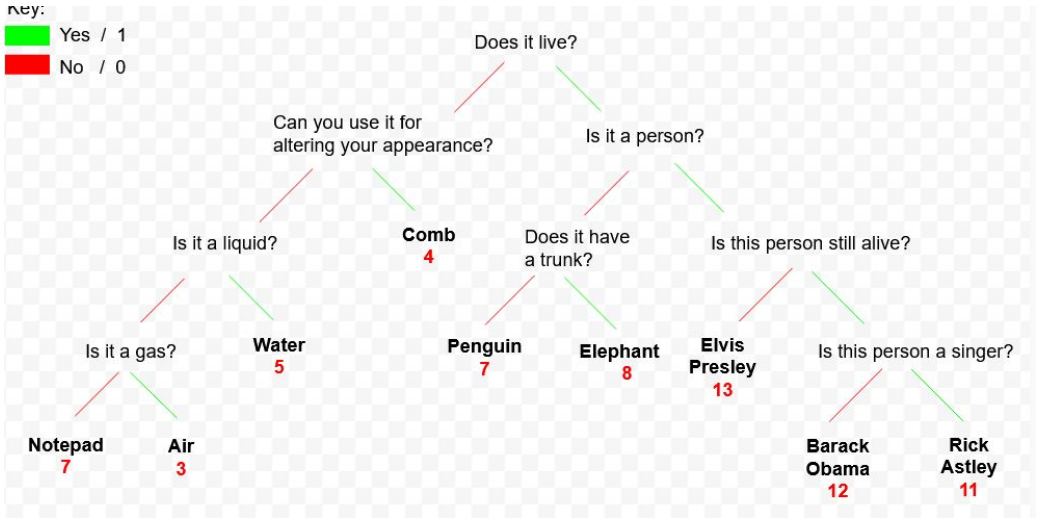
A) the node you are currently on
B) whether you are currently looking at a leaf node
C) whether you're on a *max* or *min* layer (the player that's on the max layer will optimize his own results, while the player that's on the miin layer will try to minimize the other player's results).

For each node, starting at the root node of the tree, we first determine whether the current node is a leaf node. In this project, we know we're at a leaf node whenever the node contains a question. As such, we call `if (!rootNode.IsQuestion())`.
Should we be at a leaf node, we evaluate the current node and return it's value.

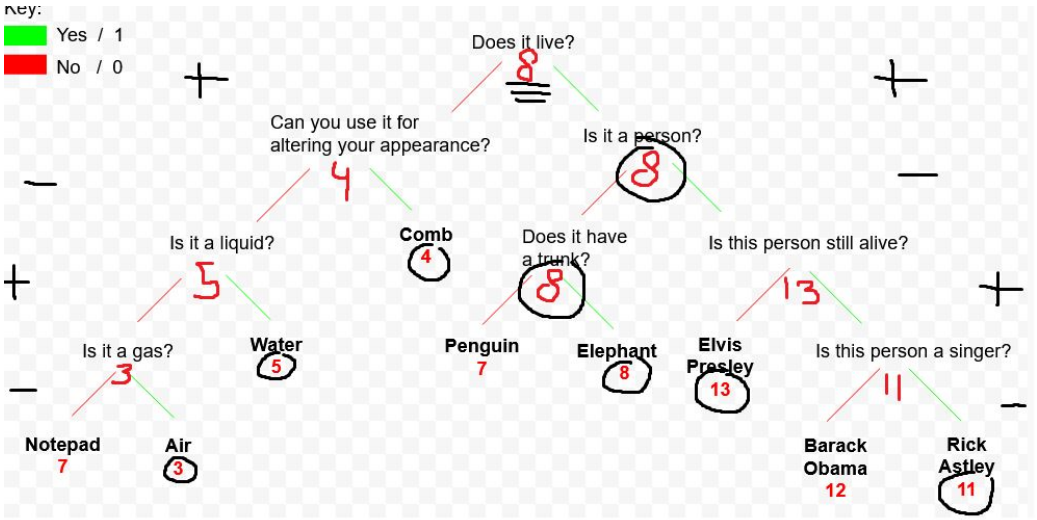
Should we not be at a leaf node yet, we need to determine whether we're on a *max* or a *min* layer. We simply pass a bool each time we call the Minimax function to keep track of this.
Based on whether we're on a *max* or *min* layer, we then recursively call the Minimax function on both children of this subtree's root node (also passing whether the next layer would be min or max) and return whichever result is bigger or smaller respectively.

```
static int MiniMax(BTNode rootNode, bool isMax)
{
    if (!rootNode.IsQuestion())
    {
        return Evaluate(rootNode);
    }

    if (isMax)
    {
        return Math.Max(MiniMax(rootNode.GetNoNode(), false), MiniMax(rootNode.GetYesNode(),
false));
    }
    else
    {
        return Math.Min(MiniMax(rootNode.GetNoNode(), true), MiniMax(rootNode.GetYesNode(),
true));
    }
}
```

This image denotes the scores that follow out of evaluating the leaf nodes.



This image denotes that the Minimax result for this tree would be 8 (or Elephant), assuming the root node is maximized (Circled in black are the winners in each layer):



We can check whether our function works as intended by checking whether we output 8 when we call it on the root node of our binary tree, considering that the root node is maximized.

```
Console.WriteLine("Output Minimax maximized: " + MiniMax(tree.GetRootNode(), true));
```

Output: `Output Minimax maximized: 8`

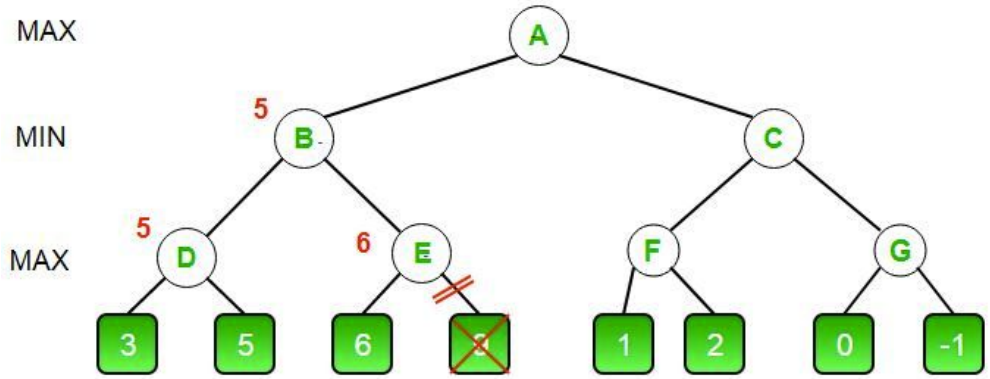
So we can (reasonably) assume our method turns out to function as intended!

Question 7

Apply alpha-beta pruning to your tree. You can choose to do this by programming an AB-pruning function, or to draw the tree out on paper and calculate it manually. Do you find alpha-beta pruning algorithm to work better than Minimax for this tree?
Explain why or why not.

The alpha beta pruning algorithm can decrease the amount of nodes you need to evaluate in the minimax algorithm. It does this by disregarding the paths that the player would rationally not play.

For example, the normal minimax algorithm would work like this:



[2]

If we want to evaluate what choice the minimizing player should make at B, we can complete the following steps:

1. If the maximizing player has to make a choice at D, we evaluate **both** 3 and 5, and can conclude that the player will choose 5.
2. If the maximizing player has to make a choice at E, we evaluate **both** 6 and 9, and can conclude that the player will choose 6.
3. If the minimizing player has to make a choice at B, we look at **both** 5 (found in step 1) and 6 (found in step 2). and can conclude the player will choose 5.

With AB pruning, we instead take these steps:

1. If the maximizing player has to make a choice at D, we evaluate **both** 3 and 5, and can conclude that the player will choose 5.
2. If the maximizing player has to make a choice at E, we evaluate 6 and find that it's already higher than five, and can conclude that the player will choose 6 (or higher).
3. If the minimizing player has to make a choice at B, we look at **both** 5 (found in step 1) and 6 (found in step 2). and can conclude the player will choose 5.

[2] Image from https://www.geeksforgeeks.org/minimax-algorithm-in-game-theory-set-4-alpha-beta-pruning/ at 04-12-2020

The crucial difference here being that once we've found that the maximizing player would *at least* be able to pick 6 in the right branch from B, we don't even have to consider the nine anymore, since we already know we would be worse off by picking the right branch at B.

To apply alpha beta pruning to our original Minimax function we need to first add two parameters so we can keep track of the *alpha* and the *beta*, which will represent the best value that the *maximizer* and *minimizer* can guarantee at a specific level in the tree. The alpha will represent the best value for the maximizer, while the beta will represent the best value for the minimizer.

In the method, we first check whether we are at a leaf node and if so, we return the evaluation value of that node.
After that, we'll look at whether we're on a layer that's maximizing or minimizing.

If we're maximizing, we first set the bestValue to the worst possible value, `int.minValue`. This is so that any higher value we'll find in a child will replace the `bestValue` with actual values from the tree.
For each child node, we then:

- Find the Minimax value for this subtree by recursively calling the MinimaxABPruning function (passing the current `child`, setting the bool to `false` to ensure the next layer will minimize, and passing the `alpha` and the `beta`)
- Assign the found `value` to the `bestValue` if it exceeds the `bestValue`
- Assign the `alpha` to the `bestValue` if it exceeds the `alpha`
- Check whether the `beta` is smaller than or equal to the `alpha`
    - Stop looking at other children if this is true

And finally return the `bestValue`.

If we're minimizing, we first set the bestValue to the worst possible value, `int.maxValue`. This is so that any lower value we'll find in a child will replace the bestValue with actual values from the tree.
For each child node, we then:

- Find the Minimax value for this subtree by recursively calling the MinimaxABPruning function (passing the current `child`, setting the bool to `true` to ensure the next layer will maximize, and passing the `alpha` and the `beta`)
- Assign the found `value` to the `bestValue` if it's lower than the `bestValue`
- Assign the `beta` to the `bestValue` if it's lower than the `bestValue`
- Check whether the `beta` is smaller than or equal to the `alpha`
    - Stop looking at other children if this is true

And finally return the best value.

```
static int MiniMaxABPruning(BTNode rootNode, bool isMax, int alpha, int beta)
{
    if (!rootNode.IsQuestion())
    {
        return Evaluate(rootNode);
    }

    BTNode[] children = new BTNode[2];
    children[0] = rootNode.GetNoNode();
    children[1] = rootNode.GetYesNode();

    if (isMax)
    {
        int bestValue = int.MinValue;
        int value;

        foreach (BTNode child in children)
        {
            value = MiniMaxABPruning(child, false, alpha, beta);
            bestValue = Math.Max(bestValue, value);
            alpha = Math.Max(alpha, bestValue);

            if (beta <= alpha)
            {
                break;
            }
        }
        return bestValue;
    }
    else
    {
        int bestValue = int.MaxValue;
        int value;

        foreach (BTNode child in children)
        {
            value = MiniMaxABPruning(child, true, alpha, beta);
            bestValue = Math.Min(bestValue, value);
            beta = Math.Min(beta, bestValue);

            if (beta <= alpha)
            {
                break;
            }
        }
        return bestValue;
    }
}
```

The above piece of code is largely based on the pseudocode found at https://www.geeksforgeeks.org/minimax-algorithm-in-game-theory-set-4-alpha-beta-pruning/ (at 04-12-2020)

We can call the function with an initial alpha of int.MinValue and an initial beta of int.MaxValue to find the result.

```
MiniMaxABPruning(tree.GetRootNode(), true, int.MinValue, int.MaxValue)
```

To see whether this process is useful for my tree, I simply count how many calls I have to make to the function to achieve the same result.

This can be seen in the following output:

```
Output Minimax maximized: 8 (with 17 calls)
Output Minimax + AB Pruning maximized: 8 (with 14 calls)
```

For my current tree we only save three calls. Still, this is a ~18% decrease in the amount of calls needed.

Especially in bigger trees this process can save quite a bit of unnecessary calculating.

What's also good to notice is that my tree is quite unbalanced. Therefore this algorithm doesn't have that much of an effect, as the tree doesn't support many situations where big branches can be left out of the evaluation. (In my case, a value is often compared to another subtree, where this algorithm's efficiency becomes more apparent when a subtree is compared to a subtree).