# Assignment #2
# 20 Questions Game

Nathan Balon
CIS 350
Data Structure & Algorithms

| What<br>_ indicates program/other is memo | Points | Due Thursday Feb 17 | Due Tuesday March 8th |
|---|---|---|---|
| **External Documentation** | **8** | | |
| Your Name | 1 | **X** | |
| Description of the problem | 2 | **X** | |
| Input Specification | 1 | **X** | |
| Output Specification | 1 | **X** | |
| Algorithm Description (english or UML) | 5 | **X** | |
| **Data Structure** | **10** | | |
| main data structure "structure" | 1 | | **X** |
| member functions / functions pre/post conditions for each | 3 | **X** | |
| Implementation and discussion. | 3 | | **X** |
| **Analysis** | **17** | | |
| How many animals are possible to identify with 20 questions? On a 1 gig machine, where the tree takes up 10 meg estimate the number of animals you can store and the number you can identify in a reasonable amount of time. | 3 | **X** | |
| Worst and/or average case time analysis for each function | 4 | | **X** |
| Worst and/or average case space analysis for each function | 4 | | **X** |
| Test Plan | 3 | **X** | |
| Sample Runs | 3 | | **X** |
| **Program Listing Style** | **15** | | |
| Your Name | 1 | **X** | |
| Description of the problem | 2 | **X** | |
| Variable Names | 3 | | **X** |
| Data Dictionary | 2 | | **X** |
| Pre/post conditions | 3 | **X** | |
| Length of functions | 2 | | **X** |
| Use of white space | 2 | | **X** |
| **Functionality** | **50** | | |
| Main | 5 | **X** | |
| Inputs questions/answers | 5 | | **X** |
| Reads the file correctly | 5 | | **X** |
| Outputs questions/answers | 5 | | **X** |
| Writes File correctly | 5 | | **X** |
| Creates Tree | 10 | | **X** |
| Walks Tree | 5 | | **X** |
| Updates Tree | 10 | | **X** |

## Description of Problem

A game is to be created which will ask the player a series of questions. The game will start by asking a question in which the user responds yes or no to the question. Based on the users answer to a question, another question will then be asked if the game has further questions to ask.

The game will ask specific questions to determine a certain type of animal. The game will terminate when the user answers a question for which it is the answer is the correct answer contained in the game or else if the game doesn't have an answer to the question, it will prompt the user for a new question to distinguish the animal from and ask for a new animal to add to the game.

For instance, the game may ask the user the question "`Does it walk itself?`" this question may either lead to another question being asked or else the game will ask if a it is a specific type of animal. An example would be the game would ask, "`Is it a dog?`" If the user answers the question correctly with a "yes" answer, then the game would terminate with the game saying it won. If the user answered the question with an answer that the game did not have answer for, then the game would prompt the user for a new question and ask a new animal to associate with the question. The question will also contain a yes or no answer to the question. So, there are two ways in which the game will terminate.

Although the game is designed to ask question about animals, it could easily be changed to ask any type questions. By having the game prompt the user for a new question, the game will continue to learn new questions and add them to its question tree.

## Input Specification

The program will use two forms of input one from a file and another from the user. The program will read from a file to create the series of questions to ask. In addition to file, all the users input to the game will be done through the terminal.

The file is used by the program to store question. When the game is first started a file is opened and read. The file will be stored in the directory from which the program is run and named `game_questions.txt`. Once the file has been opened the game will read all the lines in the file and construct a tree of question to ask. The file contains three pieces of information. The first character of a line in the file contains the answer to question. The answer will be 0 for yes, 1 for no, and 2 for root. Next, a second field is used to indicate whether the node is internal a question or a leaf node an animal. The node type is the contained in the third character position of a line. The node type will contain the values 0 for an internal node and a 1 for a leaf node. Finally, the rest of the line contains either the question or animal name that is contained in the node. Figure 1 below shows a sample of a line contained within the file.

```
0 1 Is it bigger than a breadbox
```

Figure 1 a line from the file

Once the game has been initialized, the user will then be given a series of questions. When a question is given by the game the user will enter “yes” or “no”. The game will continue asking question, to which the user will continue to answer. In the event that the game runs out of new questions to ask the user will be prompted by the message “Give me a question to distinguish it from an **animal**”. Where the **`animal`** will be the type of animal for which the game doesn't have a question to ask. After a new question is entered, the game will again prompt the user with the message, “Give me a response (yes/no) and the new animal”. The user will then answer with two words “yes/no animal”. An example of valid input would be: “yes bird”. This is all the input that the user supplies to the program.

## Output Specification

The program will use to forms of output. The game will ask a series of question to the user that will be displayed on the terminal. Also, when the game is completed the questions used by the game will be stored to a file.

When the game is started question will be displayed to the user for them to answer. For instance, the game may ask the user the question, `"Is it a mammal?"` The game will continue in this fashion on till it is complete. When the game ends a message will be displayed `"Want to play again?"` If the user answers yes the game is replayed if no is answered the game terminates.

When the user answers no to the question, `"Want to play again?"`, the game must save its state to a file. All the questions contain in the game are then written to a file, so that the next time that the game is played it will contain any new questions that were added.

## Algorithm Description

The algorithms for the program will be described using UML activity diagrams. Each major function in the program will be illustrated with an activity diagram. The functions that will be analyzed are `main`, `askQuestion`, `askQuestion`, `writeQuestionsFile`, `writeNodeFile`, `openQuestionFile`, `addQuestion`, `getNewAnimal` and `deleteTree`.

Main is the main enter point into the 20 questions game. The algorithm used by `main` is shown below in figure 2.

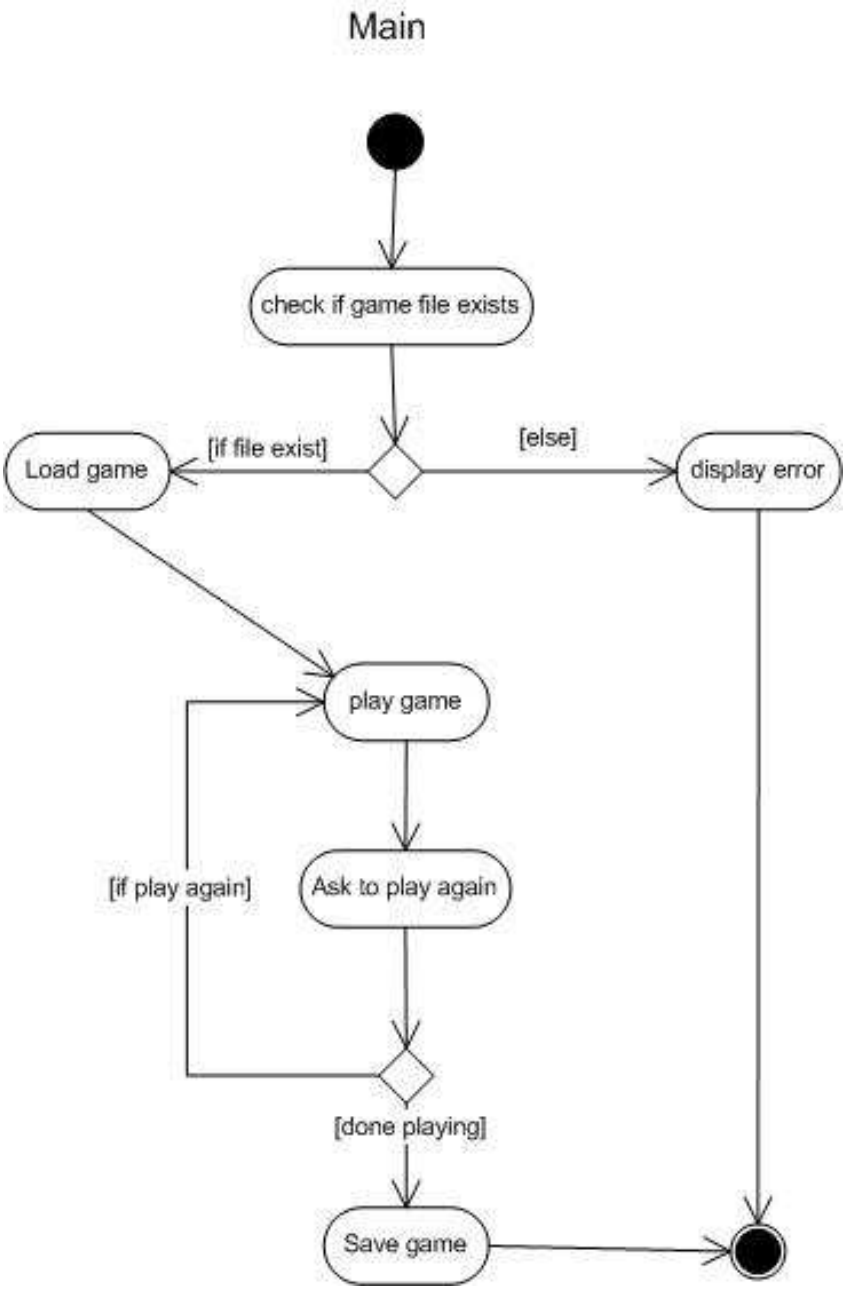


Figure 2 main

The member function play is called on a game object from within main. When play is called it starts the game. Inside the method it simply calls the private member function askQuestion, for this reason its diagram will not be shown. Inside the play method the method askQuestion is called and it is passed the root node of the tree. The member function askQuestion then recursively ask questions to the user. Figure 3 below shows the algorithm used askQuestion function.

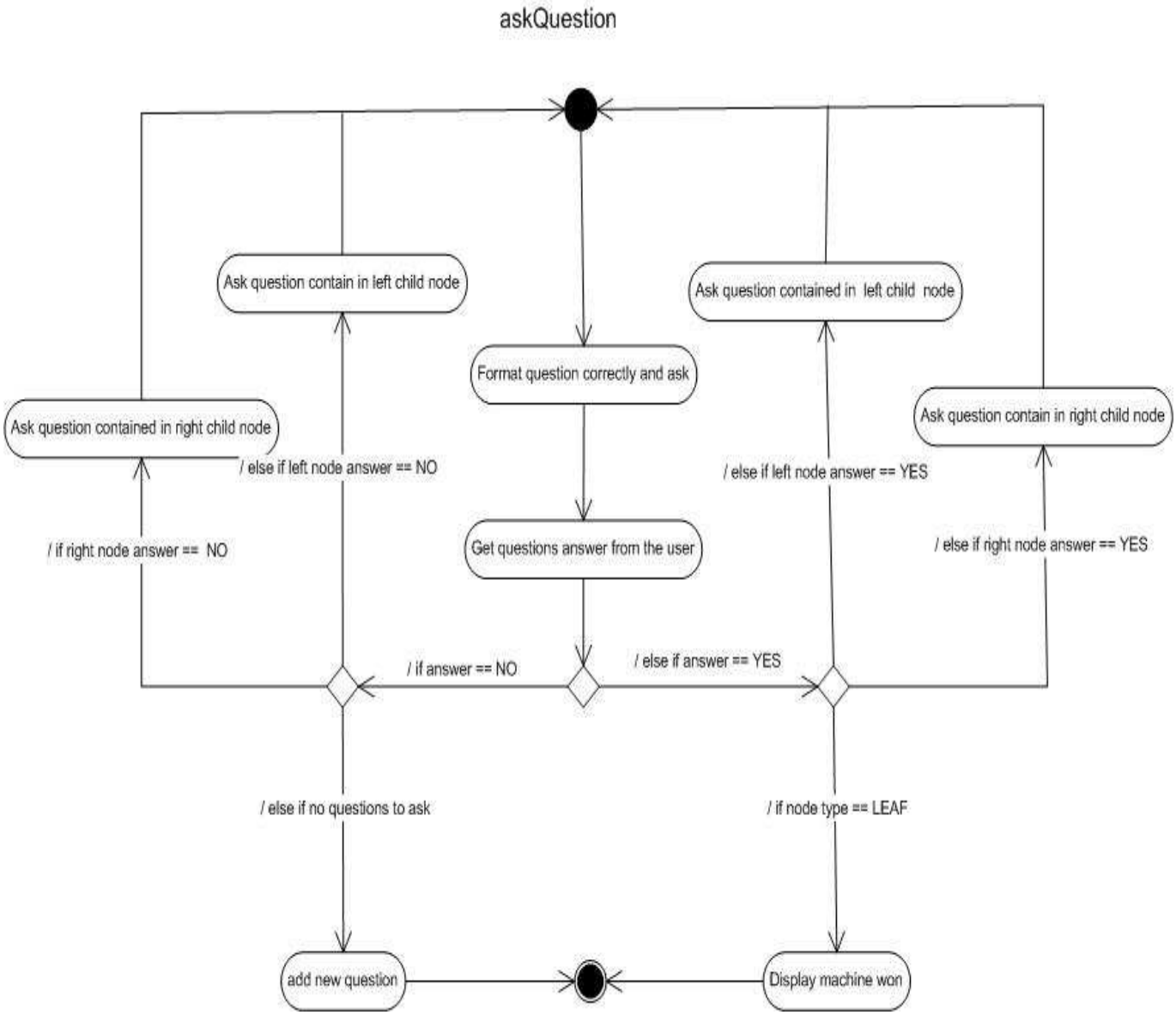


Figure 3 ask question

The member function `writeQuestionsFile` writes all the nodes contained in the tree to a file. The method `writeQuestionsFile` calls private method `writeNodeFile` which will write each node contained in the tree to a file.

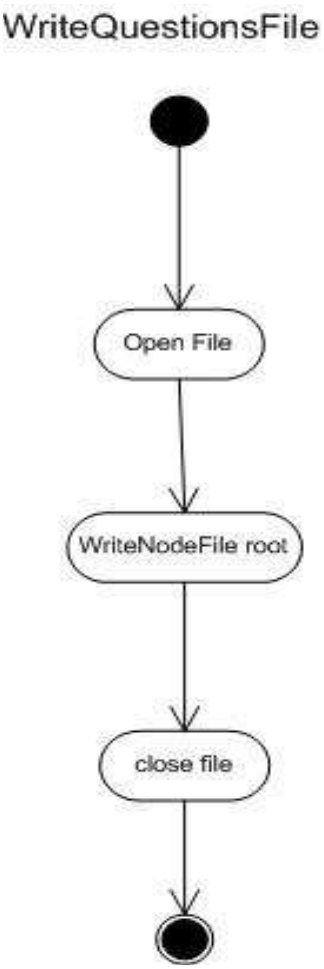


Figure 4 writeQuestionsFile

`WriteNodeFile` writes each node contained in the tree to a file. The method is called from the method `writeQuestionsFile` and it is passed an argument, the node which is the root of the tree. The method is recursively called until all nodes contained in the tree are written to a file.

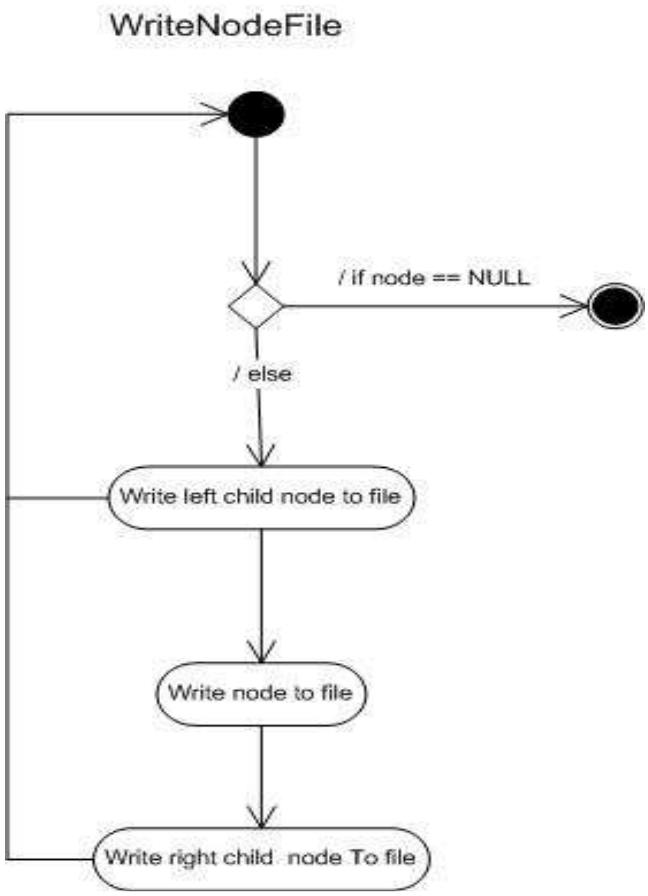


Figure 5 writeNodeFile

The destructor of a QuestionTree object will call the private deleteTree method to reclaim the resource held by a tree when it is destroyed. The method deleteTree recursively deletes the nodes contained in the tree.

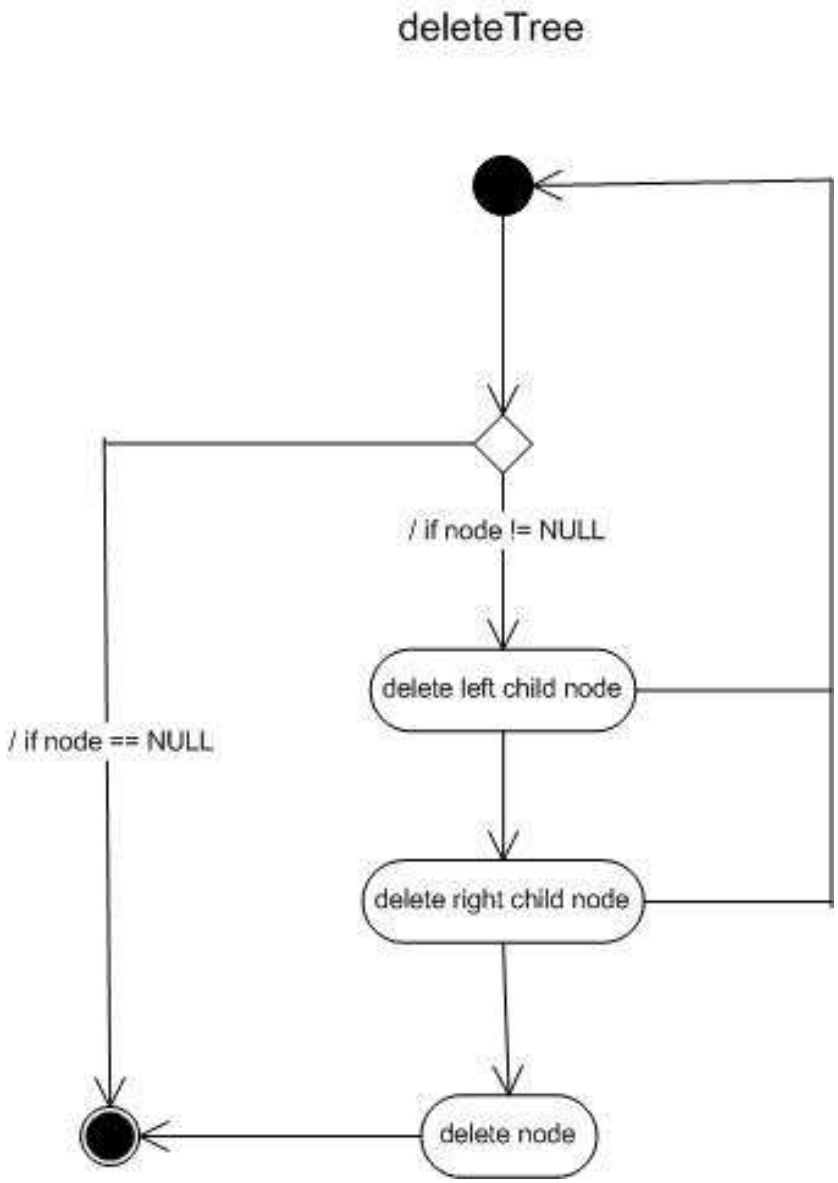


Figure 6 deleteTree

The method AddQuestion is used to add a new question to the tree when the game does not have an answer to a question.

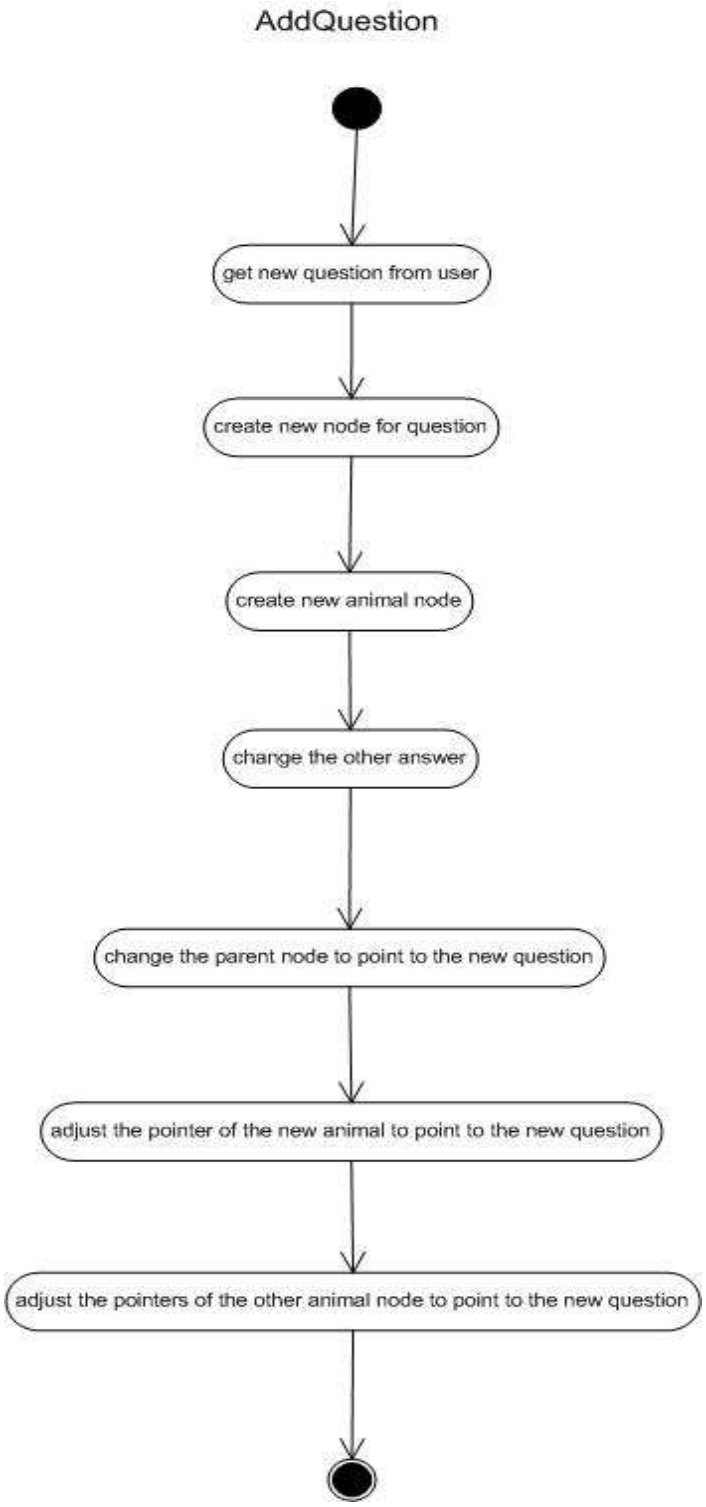


Figure 7 addQuestion

The method getNewAnimal is used add a new animal to the game. The method `getNewAnimal` is called from the method `addQuestion` when a new question is added to the game.

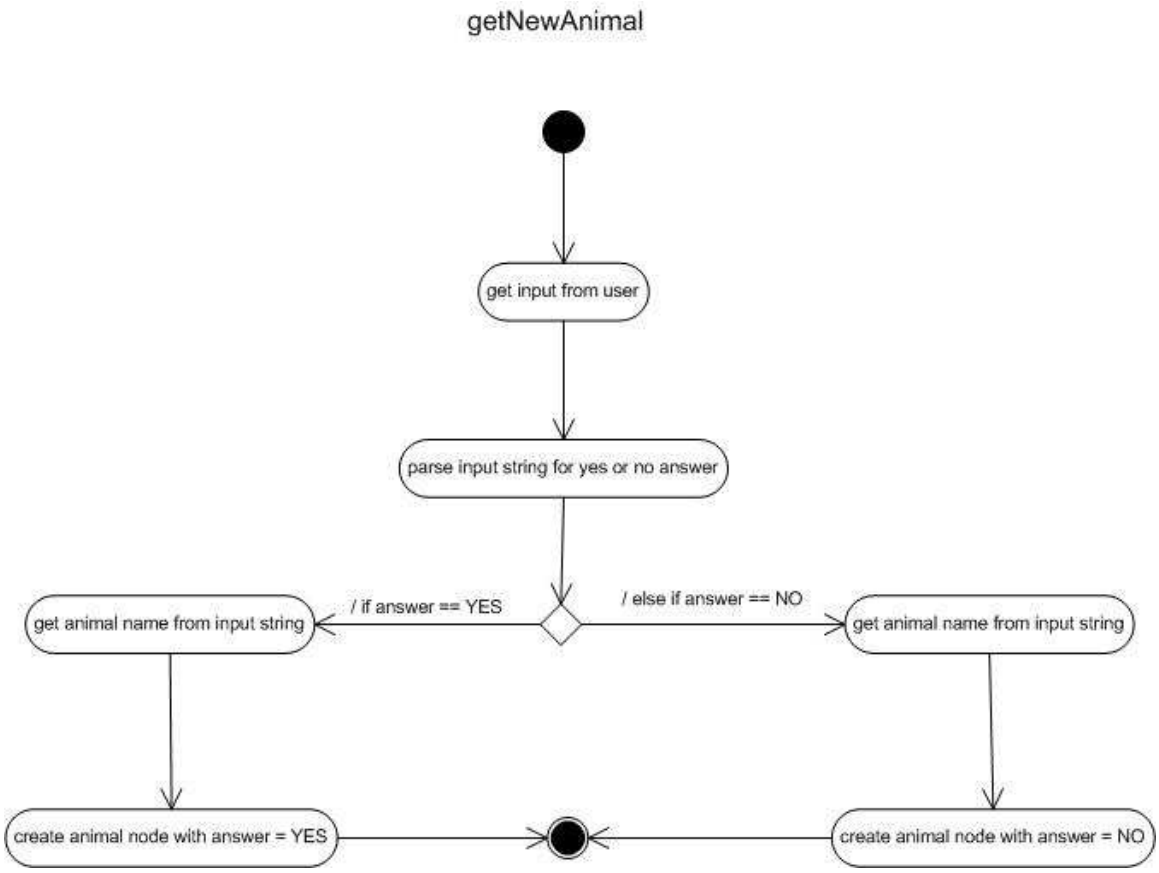


Figure 8 getNewAnimal

The method `openQuestionsFile` opens the file containing the questions to be used by the game. The method reads in line from the file calls the private method `buildNode` to create a node from the string read and then add the node to the correct location in the tree.

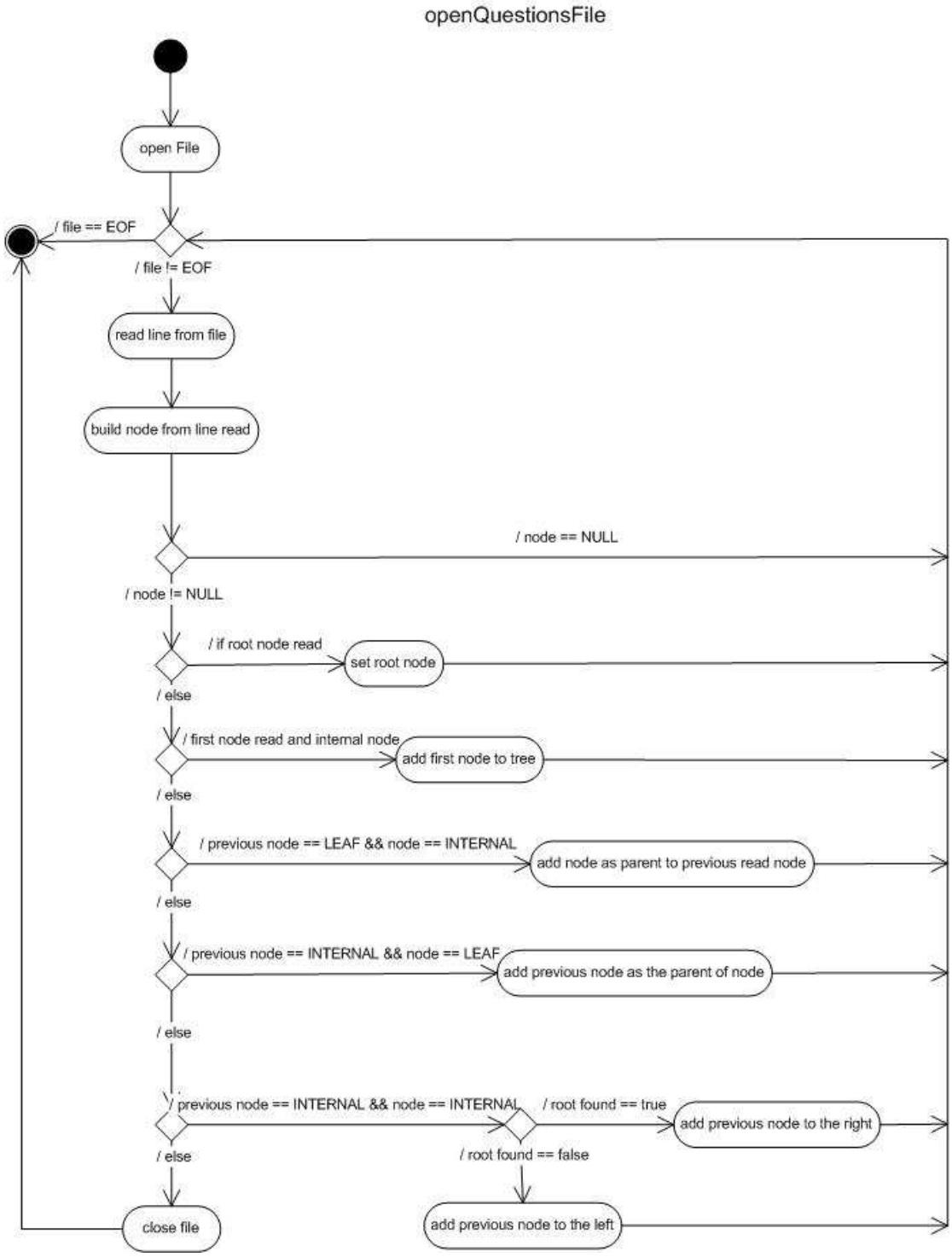


Figure 9 openQuestionsFile

The method `buildNode` is used to create a node from a string that is read from a file.

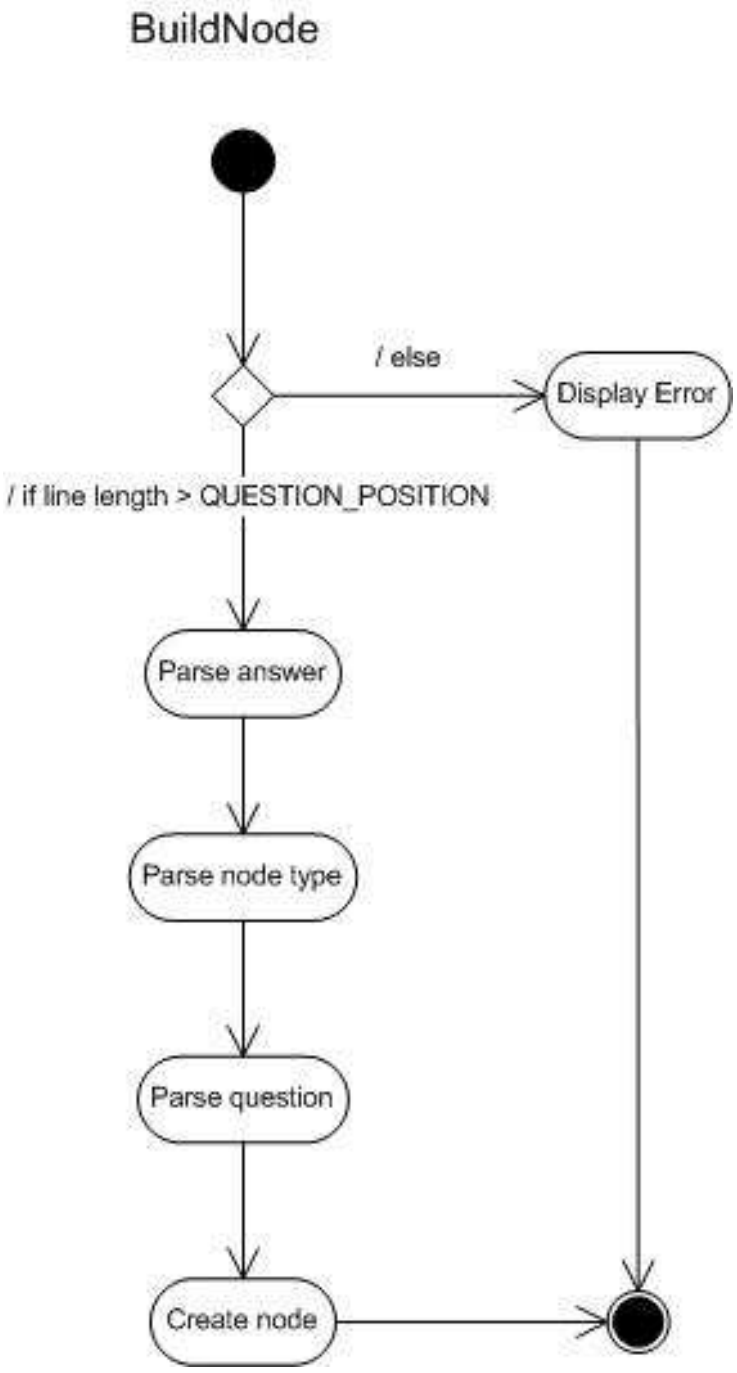


Figure 10 buildNode

## Data Structure

The class diagram below contains the member function and data used by the classes in the 20 question program.

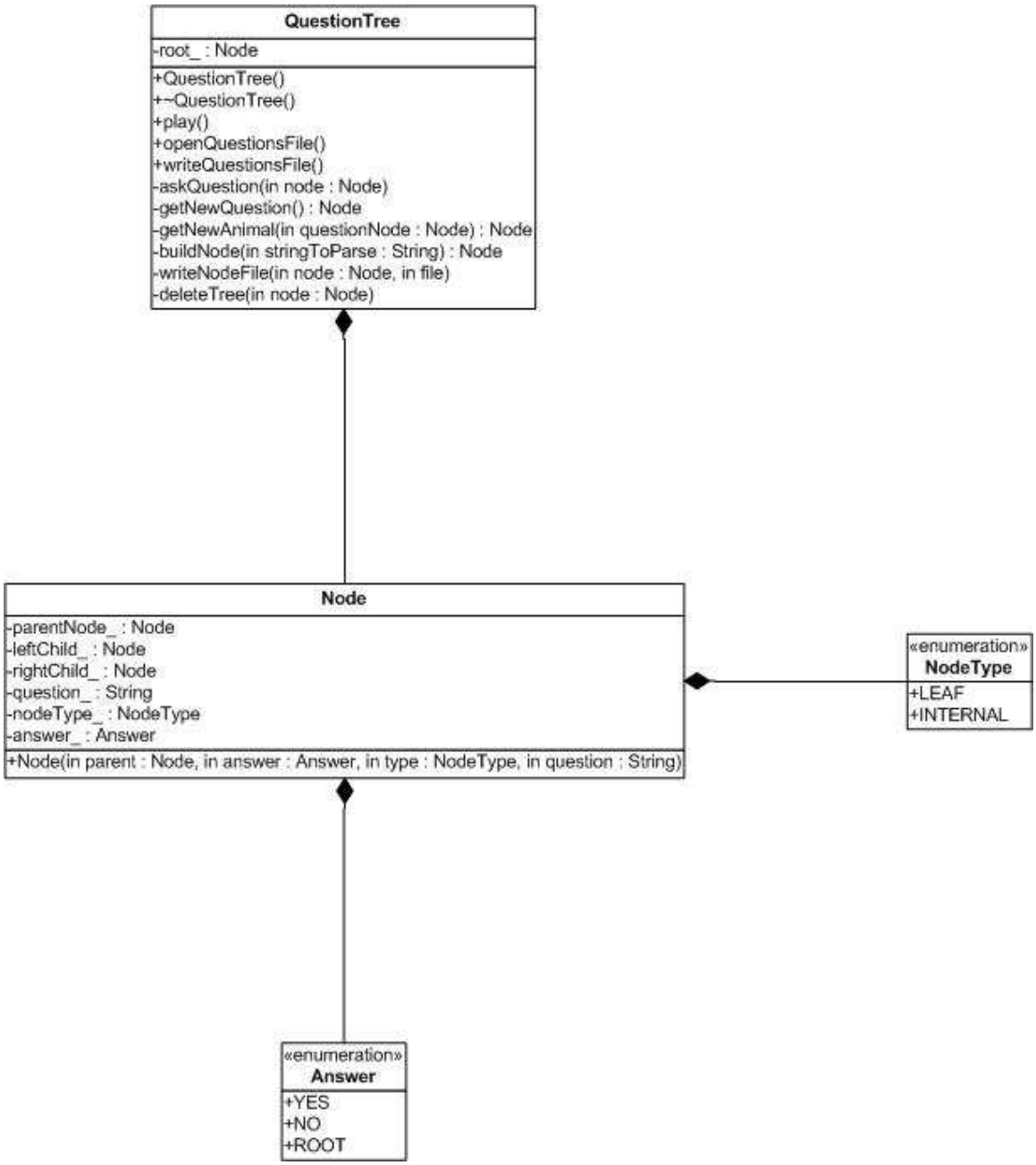


Figure 11 class digram for the 20 question game

## Member Functions: Pre and Post Conditions

| Name | Precondition | Postcondition |
|---|---|---|
| Class Node | | |
| Node | Memory on the system must be available to create a new node. | A new node object is created. |
| Class QuestionTree | | |
| QuestionTree | Memory on the system must be available to create a QuestionTree object. | A new QestionTree object is created. |
| ~QuestionTree | A question tree object exists. | The resources held by a QuestionTree object will be return to the system. |
| play | A tree has been created to use for the game and the openQuestionFile method has been called to create a tree from a file. | The end of the game has been reached and a new question may have been added to the game. |
| writeQuestionsFile | A game tree exists that will be written to a file. | The game tree is written to a file with the name given by the global constant FILE_NAME. |
| openQuestionsFile | A file must exist that is in the correct format expected to be read by the method. | A tree will be created containing nodes for each line in the file and the nodes will be inserted into the correct position in the tree. |
| askQuestion | The file containing the question has been opened and the tree has been created. | Based on the users answer another question may be asked to the user, the game may terminate with game saying it won or the user will be prompted for a new question to add. |
| writeNodeFile | A node exists to be written to a file. | If the node is not equal to null it is written to a file. The node will be written in the text format: [the answer to the question 0 for yes and 1 for no], [the type of node 0 for leaf or 1 for interior], and [the question or animal]. |
| addQuestion | The game comes to a point where it doesn't have anymore questions to ask. | A new node is created with a new question. |
| getNewAnimal | The game tree comes to leaf a node for which it doesn't have any more questions to ask. | A new animal node is created to be added to the tree. |
| deleteTree | A tree exists. | All nodes in the tree are deleted. |

| | | |
|---|---|---|
| buildNode | The string contain the correct format: which contains an integer in position 0 with the values 0, 1, or 2 and an integer in position 2 with the values of 0, 1 the rest of the string contains either a question or an animal which starts at position 4 in the string. | A new node will be created. |
| Global Functions | | |
| gameFileExists | None | Returns a bool of indicating if the game file exists in the current directory. |

Table 1 Member functions

## Implementation and Discussion

There are two classes that are used for the program. The first class is `Node`, which is used to store a question or an answer to a question. The second class is `QuestionTree`, which is used to store node objects in a tree. These two structures provide the functionality that is needed for the game of twenty questions.

`Node` is the simplest of the two data structures. Node is used to store the data that is used in the tree to play the game. Each node object contains a pointer to it left child, a pointer to its right child, a pointer to its parent, a string which contains either a question or an animal, an enumeration to tell whether the node is question or animal node and another enumeration tell the answer to the question yes or no. The enumeration wouldn't have been need but the tree has to be save to a file so there needs to be a way to determine which nodes are leafs and which are not. Also, if the game didn't need to be saved all yes nodes could be stored on left and no nodes on the right. The nodes have to be stored to a file so the answer to the question was included with each node.

One design decision that had to be made was how to implement the node, there were three possibilities create a class with mutator and accessor member functions, do not implement any member function but make the `Node` class a friend of the `QuestionTree` class or to implement it as structure. I chose to implement it as a class that is a friend to QuestionTree, so the tree object will be able to access all the private data of a `Node` object. Because the `Node` class was made a friend there are no member functions for `Node` just constructors. The only draw back of this implementation is the class can be reused in other programs.

The main data structure used in the program is questionTree. The tree is used to hold all of the questions and types of animals that are answers to the questions for the twenty questions game. The class has for public member functions the constructor, `play`, `OpenQuestionsFile` and `saveQuestionsFile`. To play the game of twenty questions the `openQuestionsFile` method is called first to load the data needed to

play the game. Next, the method `play` is called to actually play the game. Finally if the data in the tree needs to be saved the method `saveQuestionsFile` is called. There are also a number of private member functions included in the class `QuestionTree`. Some of the private methods that were includes are `buildNode`, `askQuestion`, `getNewAnimal`, and `getNewQuestion`. For instance, `buildNode` is used to create a node from a string read from the file. Also, the method `askQuestion` will recursively ask the user the questions contained in the game. These two classes `QuestionTree` and `Node` are the data structures used for the game of twenty questions.

## Analysis

How many animals are possible to identify with twenty questions?

The maximum number of animals to identify with twenty questions would happen when the tree is full with its height equal to 20. All the animals in the tree would be located at height 20. The number of possible animals to be identified would be 2^20. In the case of the twenty questions it would be able to identify 1,048,576 different animals. Although twenty questions my not seem like a lot, it is able to identify a lot of different animals.

On a I gig machine, where the tree takes up 10 megs, estimate the number of animals you can store and the number you can identify in a reasonable time?

The first thing that needs to be determined is the total amount of storage that is required to store a node in the tree. Each node will contain 3 pointers and a string. First, on a 32 bit machine each pointer will need 4 bytes of storage space, so 12 bytes will be needed to store the pointers. Second, the nodes will contain a string which is used to store either a question or an animal. One problem is that the strings will vary in length. For this problem the average string size will be used. Another problem is that the animal string will be much smaller than the question strings. Suppose that the average animal name had a length of 6 characters and the average question had a length of 24 characters. The average string length would then be 16 characters. Also, the amount of space need to store a character can vary based on the encoding that is used. For simplicity it will be assumed that the characters are encoded in ASCII so that 16 byte will be needed. The total amount of space required to store each node would be approximately 28 bytes.

If 10 Megs of storage could be used to hold the tree and each node contained approximately 24 byte, then 10,000,000 / 28 = 357,142 nodes could be stored. One thing that needs to be considered is not all the nodes in the tree are animals. All the internal nodes will be question and all the leaf nodes will be animals.

To determine how many animals can be identified, it must be determined how many traversals of the tree it would take to get to an animal. The estimated height of the tree must be found. The approximation of the height of the tree is log (357,142) ≈ 19. On the average it would take 19 traversals of the tree to reach an animal. If each traversal of the tree took 50 machine instructions, to reach an animal is the tree it would take approximately 50 * 19 = 950. Within one second, if the game did not have to wait on any user input, it would be able to access 1,000,000,000/950 = 1,052,631 animals. This is

more than is able to be stored in 10 Megs of memory. Clearly using a binary tree to store animals improves the performance, because the animals can be access logarithmical instead of linearly. There is one problem with this estimate which is if the tree isn't balanced. In the worst case all of the nodes would have to be traversed. In this case to access an animal it would take 357,142 * 50 = 17,857,100 clock cycles. This would result in only about six animals being accessed every second. The performance drastically decreases in the case of the tree being imbalanced. For, a game where human interaction is required this may be acceptable but is wasting a lot of clock cycles that could be put to better use.
Excellent

## Runtime and Space Analysis

For the analysis of the program following three variables will be used:

- m – The number of time that the program is run.
- n – The number of nodes in the tree.
- s – The length of the string used to store a question.

### Time Analysis

| Function | Worst Case | Average Case |
|---|---|---|
| Class Node | | |
| Node | O(1) | O(1) |
| Class QuestionTree | | |
| QuestionTree | O(1) | O(1) |
| ~QuestionTree | O(n) | O(n) |
| deleteTree | O(n) | O(log n) |
| getNewQuestion | O(1) | O(1) |
| getNewAnimal | O(1) | O(1) |
| buildNode | O(1) | O(1) |
| play | O(n) | O(log n) |
| askQuestion | O(n) | O(log n) |
| openQuestionsFile | O(n) | O(n) |
| writeQuestionsFile | O(n) | O(n) |
| writeNodeFile | O(n) | O(n) |
| Global Functions | | |
| gameFileExists | O(1) | O(1) |
| main | O(m * n) | O(m * log n) |

Table 2 runtime analysis

### Space Analysis

| Function | Worst Case | Average Case |
|---|---|---|
| Class Node | | |
| Node | O(s) | O(s) |
| Class QuestionTree | | |
| QuestionTree | O(s * n) | O(s * n) |

| | | |
|---|---|---|
| ~QuestionTree | O(1) | O(1) |
| deleteTree | O(1) | O(1) |
| getNewQuestion | O(s) | O(s) |
| getNewAnimal | O(s) | O(s) |
| buildNode | O(s) | O(s) |
| play | O(1) | O(1) |
| askQuestion | O(s) | O(l) |
| openQuestionsFile | O(s * n) | O(s * n) |
| writeQuestionsFile | O(s * n) | O(s * n) |
| writeNodeFile | O(1) | O(1) |
| Global Functions | | |
| gameFileExists | O(1) | O(1) |
| main | O(s * n) | O(s * n) |

Table 3 space analysis

## Test Plan

The test plan will be used to test that the program is operating correctly. For testing purposes, the program will run without a file of questions. In the event that the question file is missing from the directory the twenty question game is contained in new game will be created from the users input. A member function was added printTree which is used to test if the tree is constructed properly.

| Test # | Test | Result | Reason |
|---|---|---|---|
| 1 | Input the sample data given in the program description in the correct order without using a file. | Questions and animals previously entered should appear in the tree the next time the program is ran. | To make sure the tree is able to input correctly. |
| 2 | Check that the questions from the first run were saved to a file correctly when the program is terminated. The nodes of the tree should appear in order | The nodes of the tree should be written correctly in order. | To test that the nodes in the tree are written correctly to a file. |
| 3 | Run the program a second time to check that the nodes saved in a file are read in correctly. Travers the tree to check that nodes were recreated correctly | The nodes should appear in the same order as originally entered into the program. | To test that the tree is able to be recreated from a file. |
| 4 | Run the program enter no for “Is the animal a mammal” and then no yes for “Is it a chameleon”. | The program should display that the machine won. | To test that the program terminated correctly when the correct answer was given for a leaf node. |
| 5 | Enter in the answers to the | A new node should | To test that when the |

| | | | |
|---|---|---|---|
| | questions correctly to get to the question “Is it a cat” and answer no. Add the new question “Does it like sunflowers”, the answer yes, and the animal hamster. | be created and cat should be moved down one level in the tree. | parent has to leaf nodes and the answer is no for the animal the new question is placed correctly. |

Table 4 test plan

Very nice

## Sample Runs

```
                    The 20 Questions Game
-------------------------------------------------------------

Is the animal a mammal? yes
Is it bigger than a breadbox? yes
Is it a elephant? yes
Another win for Machine Intelligence! Want to play again
(yes/no)? yes
Is the animal a mammal? yes
Is it bigger than a breadbox? yes
Is it a elephant? no
Give me a question to distinguish it: it is hairy
Give the response (yes/ no) and the new animal: yes gorilla
Want to play again (yes/no)? yes
Is the animal a mammal? yes
Is it bigger than a breadbox? yes
it is hairy? yes
Is it a gorilla? yes
Another win for Machine Intelligence! Want to play again
(yes/no)? yes
Is the animal a mammal? yes
Is it bigger than a breadbox? no
Does it walk itself? yes
Does it like sunflower seeds? no
Is it a cat? yes
Another win for Machine Intelligence! Want to play again
(yes/no)? yes
Is the animal a mammal? no
Is it a chameleon? no
Give me a question to distinguish it: Does it fly in the
air
Give the response (yes/ no) and the new animal: yes
Invalid input, please enter again
Give the response (yes/ no) and the new animal: bird
```

```
Invalid input, please enter again
Give the response (yes/ no) and the new animal: yes bird
Want to play again (yes/no)? yes
Is the animal a mammal? no
Does it fly in the air? no
Is it a chameleon? yes
Another win for Machine Intelligence! Want to play again
(yes/no)?
```