COMSC-205PY FALL 2020 Assignment 6

Due December 10th at 9:00am Eastern Time

Submission checklist

Your Assignment 6 submission should be a zip file which contains the following directory structure:

- **Assignment6**
  - **BinaryTree.py**
  - **TreeNode.py**
  - **GameTreeReader.py**
  - **GuessingGame.py**
  - **UnrestrictedGuessingGame.py**

At the top of each file that you create or modify, clearly state:

- Your name and the name of the assignment
- Attribution for any sources used, including people (other than instructors) who you asked for help
- Description of the file

Read all the way through this assignment twice before you start working on it. The second time through, take notes, plan out how you will work on this.

Background

In the classic Twenty Questions game, one person thinks of something--anything--and other people try to guess the thing by asking yes or no questions to narrow down what sort of thing it is. If they guess it in fewer than 20 questions, they win. For this assignment, you will write a program for a person to play 20 Questions with the computer. In this version of the game, the human thinks of something and the computer will try to guess it. (We will ignore the 20 questions or fewer criteria for our game.)

You will implement two versions of the 20 Questions guessing game. The first version has a **pre-specified set** of things. The second version allows an **unrestricted** set of things; it begins with the pre-specified set but allows the user to add objects to the set.

The Assignment

Guessing Game

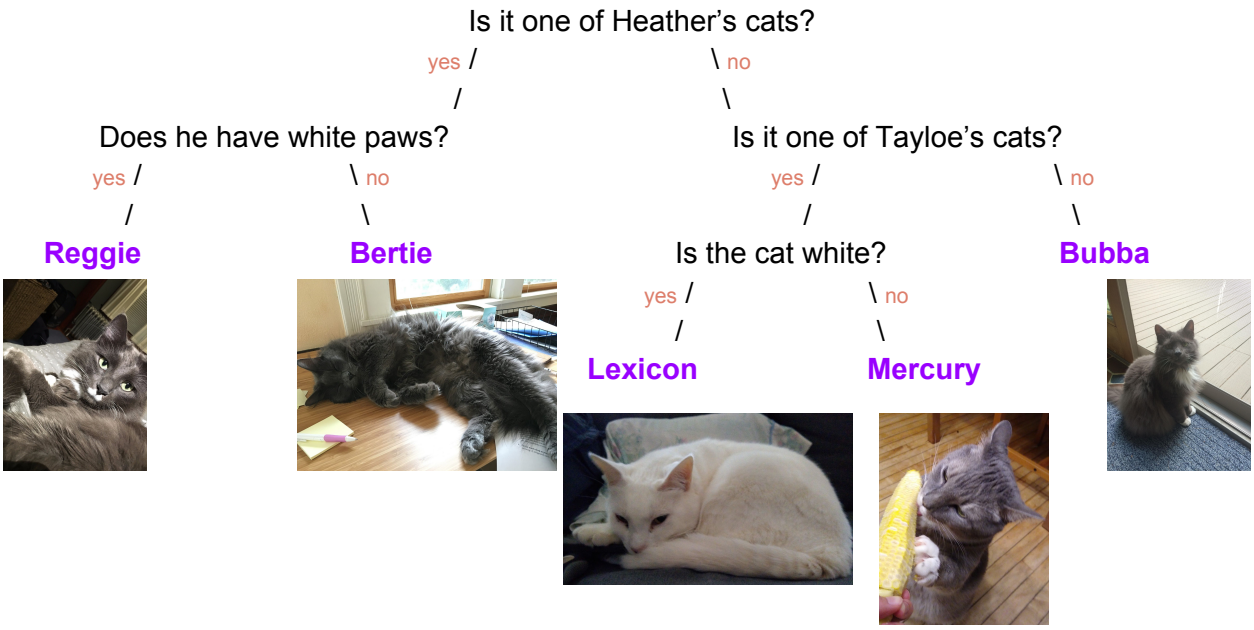
In this variation of the game, the choice of things is **pre-specified** and is shown to the user at the start.

- You must have at least 16 pre-specified things
- These pre-specified things should be shown at the start of the game

Your application will need to read in a text file at runtime and store it in an appropriate data structure, which is a binary tree (not a binary search tree). Starter code is provided to read in the

text file, along with our implementation of BinaryTree. You do not need to change any of the starter code, though if you have a good reason to change it you can. You should read over it to get a sense of how it works. The starter code contents:

- **BinaryTree.txt** - a data structure
- **BinaryTreeNode.txt** - a data structure
- **GameTreeReader.java** - text file reading code
- **tree.txt** - a sample file with the tree shown below:



Notice the following about this structure:

- It is a binary tree (but not a binary search tree)
- The things (cats) are all leaf nodes
- The questions are all internal (non-leaf) nodes
- The **left child branches** all correspond to **yes** answers while the **right child branches** all correspond to **no** answers.

We will use a text file to store structured data about the questions and the things in a way that allows the structured data to be read into a Python program. The example tree.txt file shows the structure needed:

```
Is it one of Heather's cats?
    Does he have white paws?
      Reggie
      Bertie
    Is it one of Tayloe's cats?
      Is the cat white?
           Lexicon
           Mercury
      Bubba
```

Each tree level is separated by a tab. For example, the root node has 0 tabs; the root's children have one tab; the root's grandchildren have two tabs, and so on. You may use the text.txt file to get started on this assignment, or use its structure and replace the questions and things with ones from your own theme. Eventually, you must replace it with **your own txt file** that has at least **16 things** and questions to distinguish them. But to get started, it is strongly recommended that you use this smaller tree to test. **Do not use spaces; use tabs!**

You will need to write **GuessingGame.py**. It should be a class, and an attribute of the class will be a BinaryTree. We have provided code to create a BinaryTree object from a tab-formatted txt file. See GameTreeReader.py for how to call the getTree method in GameTreeReader. In GuessingGame.py, write code to navigate through the tree, taking user input to go left or right as a user answers yes or no. At the end of the game, the program should show the final answer, and ask if the guess was correct or not. It should then ask if the user wants to play again, going back to the start if yes and quitting if no.

Now you are ready to write your own set of questions and things (the fun part of the assignment!) and encode them in a txt file. Pay close attention to the structure and the nesting of the tags and the attributes in the example txt file. The GameTreeReader class will not work if the structure diverges from the structure of the example file.

Unrestricted Guessing Game

In this part, implement an "unrestricted" 20 questions game. You will need to write a class **UnrestricedGuessingGame.py**. It should *inherit* from GuessingGame.py. You will need to expand your application to let the user pick a thing that is not on the pre-specified list.

- Start asking questions in the same way.
- If the computer gets to a point where there is only one thing remaining, and guesses incorrectly (because the thing the user is thinking of is unknown), **learn the thing.**
- This will require asking the user:
    - What <thing> were you thinking of?
    - Please give me a yes/no question that would have determined your thing.
    - Is the answer to your question yes or no?
- Modify your data structure based on the answer so that the new thing is in the list for a new round that is started within the session

When implementing UnrestrictedGuessingGame, the goal is for the class to inherit as much as possible from its parent class. If designed efficiently, there does not need to be very much code in UnrestrictedGuessingGame.py. This will likely mean that you will revise or redesign methods in GuessingGame.py as you go.

How to Approach the Assignment

Be sure to break your process into smaller steps. We strongly recommend that you approach the steps **in the order outlined below** and that you aim to complete Steps 1 and 2 no later than one week before the project due date.

1. Create your own set of things and write your own tree file
    1. Also create a smaller set of 2-3 things and use this as you write your code
2. Design the user interaction portion of the game
    1. think of information that needs to be presented to the user
    2. think of information that needs to be gotten from the user
    3. sketch out the design on paper
3. Design and implement the backend
    1. think of the data (state) that needs to be stored -> this should help you decide what attributes to have in the classes
    2. think of the behavior that must be supported by the various classes -> this should help you decide what methods are required
    3. implement the methods, testing constantly!
4. DEBUG :)

Rubric

Each section will be given a letter grade. The average of those letters will generate your final score. Note: Moodle does not allow letters, so they will be translated into a percentage there.

	F	D	C	B	A
File Reading	File reading is missing	The tree is hard-coded, rather than being read from a file	The txt file is missing	A txt file is mostly read in by the program, with a bug or two, or there are fewer than 16 things in the file	A txt file is included, read in by the program, and made into a tree
Guessing Game	Not present	Game does not take user input, or does not navigate tree	Game does not accept some inputs, or skips many questions, or only navigates some of the tree	Game skips a question, or does not replay, or does not ask if answer was correct	Game asks yes/no questions, navigating through the tree, asking for user input until reaching the answer. At the answer, game asks if the answer was correct, and asks to play again (and responds appropriately)
Unrestricted Features	No change is made/not present	New question and answer are not added, or game cannot be replayed	New answer is added, but replaces old one; new question is added, but replaces old one; other major bad bug	Gameplay is changed from before, or tree is added to in slightly incorrect location, or similarly small bug	Game plays as before. When responding “no” to a final answer, a new answer and question can be added, and the game restarts. Replaying the game, the new answer and question are included in the correct location.

Comments, naming, style	No comments present, no name in file, method names incorrect	Name in file, some methods named correctly, no comments	Name in file, description of file in comment, methods named correctly (typos ok), more than one error that prevents running	Name in file, description of file and most methods, methods named correctly, some variables named meaningfully, significant over- or under-use of comments or one error that prevents running	Name in file, description of file and all methods, methods named correctly, all variables named meaningfully, other comments as needed, no errors that prevent running