A Python Oracle Engine

“Consulting the Oracle“ by John William Waterhouse - Christie's, LotFinder: entry 5263447 (sale 7823, lot 19), Public Domain, https://commons.wikimedia.org/w/index.php?curid=7715929

Whenever I sit down to plan or write a story, I have a stack of solo roleplaying rules and a handful of polyhedral dice. The roleplaying rules contain various oracles, lists of words, from which word pairs can be randomly selected. When I want to develop some more information about events, or the way a character acts, I roll on these oracle tables and interpret the word-pairs, for meaning based on the context of the story. I have been stunned by how successful it is.

Sometimes, though, I want to quickly consult an oracle without dragging out the books and dice. So, I quickly coded up this Oracle Engine in Python. I deliberately designed it to run on a vanilla Python install without any extra libraries other than those that come as standard in any Python install. This means anyone with a python installation can run the .py script and be pretty confident it will work.

Downloading and Setting It Up

Download the zipped up script and associated files from here: ***TheMultiOracle.zip***

Unzip the archive into a folder of your choosing. You will find it contains a Python script file, *TheMultiOraclev1c.py*, and a resources subdirectory with the support and data files *Oracle_Word_Listsv0.csv* and *OracleLastUsed.txt.*

That’s the setup done.

What the Files Do

The Python file *TheMultiOraclev1c.py*, creates the GUI and runs the oracles. As you'd expect the command to get it running is:

```
python3 TheMultiOraclev1c.py
```

or

```
python TheMultiOraclev1c.py
```

The two support and data files do the following;

- *Oracle_Word_Listsv0.csv*: This is the list of oracles the script will look up. You can add extra lists into this for custom oracles.
- *OracleLastUsed.txt*: This just holds the last used settings. If it gets deleted accidentally, it will be recreated and repopulated with the default.

Adding Your Own Oracles

You will not need to touch the coding if you want to add extra oracles.

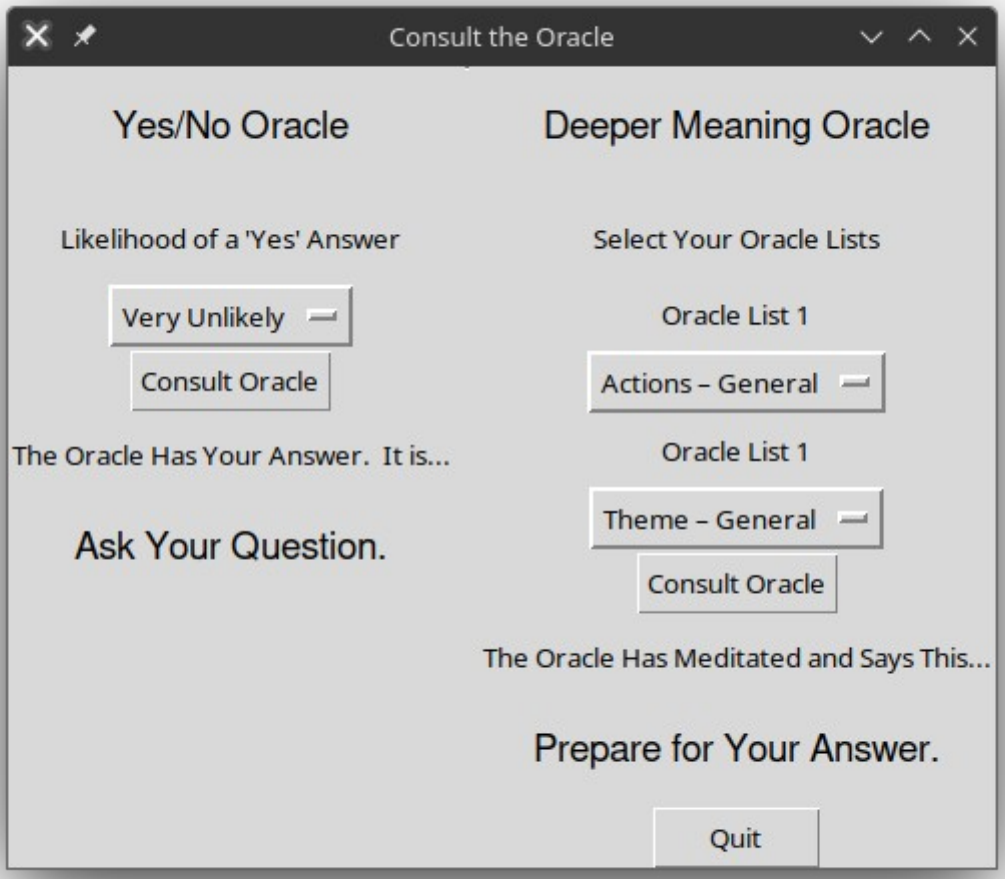
Open the *Oracle_Word_Listsv0.csv* in your favourite spreadsheet editor and add your new word lists as a new column. Give it a title, and include the number of elements in your list at the top in row 3.

Oracle Word Lists				
Actions – General	Theme – General	Goblin Alone - Action	Goblin Alone – Descriptor	Goblin Alone – Them
100	106	148	105	1
Pursue	Affection	Achieve	Ancient	Affection
Accentuate	Age	Agree	Angry	Age
Puncture	Agreement	Appease	Artificial	Agreement
Fascilitate	Alibi	Apprehend	Athletic	Alibi
Ease	Balance	Approach	Awful	Balance
Elevate	Beauty	Argue	Barely	Beauty
Focus	Behaviour	Befriend	Bitter	Behaviour
Educate	Belonging	Believe	Broad	Belonging
Pretend	Benefit	Build	Bulky	Benefit
Extol	Challenge	Burrow	Busy	Challenge
Appreciate	Clarity	Carry	Cautious	Clarity
Solicit	Coldness	Chill	Charming	Coldness
Defend	Comfort	Clamber	Clean	Comfort
Accuse	Community	Climb	Clumsy	Community
Cruise	Companion	Combine	Cluttered	Companion
Increase	Concerns	Command	Comical	Concerns
Erode	Construction	Complete	Considerate	Construction
Decrease	Containment	Confuse	Creative	Containment
Manage	Curiousity	Consume	Crooked	Curiousity
Mess up	Custom	Cook	Cute	Custom
Complicate	Dance	Coordinate	Daintily	Dance
Simplify	Danger	Crawl	Delicious	Danger

If you delete the two default oracles, *Actions – General* and *Theme – General*, included in the oracle list, you will need to change the content of the *OracleLastUsed.txt* file to carry the names of two of the oracles in your custom list, and also change the lines referring to *Actions – General* and *Theme – General* in the code to whatever your oracle names are.

The Interface

The oracle interface includes two separate oracles; A *Yes/No Oracle* and the *Deeper Meaning Oracle*.



The Yes/No Oracle

If you frame your question to require a “Yes” or “No” answer, then you can use the *Yes/No Oracle*. Select the likelihood that your answer will be “Yes” using the drop down, then press the *Consult the Oracle* button.

The oracle will display its answer which will be one of; “*Absolutely Yes, and* ...”, all the way down to “*Absolutely No, and ...*” Here is the list of possible answers and how you interpret them.

Yes, and ...	An exceptionally good “yes”. This is a “yes” with an added benefit.
Yes	The expected result is true.
Yes, but ...	The expected result is true, but there is a minor problem.
No, but ...	The result is the opposite to what is expected, but there is some positive aspect to it.
No	The result is the opposite to what is expected
No, and ...	The result is that things haven't turned out as expected, and even worse, there is an added sting in the tail.

Example*:* I have written a story where the main character is careening down a street in a shopping trolley and there is a sharp bend in the road ahead. On the inside corner there is a steep earth bank rising above the road, and on the other side there is a guard rail, and beyond that a big drop into a swamp. While I could dictate what happens, I like a add a bit of serendipity or disaster into my stories so it doesn’t end up looking

contrived. I consult the Yes/No Oracle, with the question in my mind “Does the character make it around the curve safely?” I assign the likelihood of success as *Unlikely*. I am being kind. I could have selected *Very Unlikely* or *Nearly Impossible*. Pressing the button yields the answer “*Yes, but...*” I interpret that as the character somehow manages to lean the shopping trolley in such a way that they successfully manage to get around the corner but now they’re facing an on-coming car.

The *Yes/No Oracle* is the same as for the “*A Goblin Alone*” solo rules, I developed during the *Goblin Errands* gamejam on Itch.io. It is available on a pay what you like basis, which most people interpret as “Free.” You can find it here: **A Goblin Alone**

The Deeper Meaning Oracle

The *Deeper Meaning Oracle* can be used to introduce more information about a scene, or a character’s behaviour. Consulting this oracle will yield two words which are interpreted based on the context. Interpret the words based on what you know of your world, your character, and the story. It may be that you can't get any meaning from it, in which case, press the button again.

The oracle pairs are selected using the two drop downs.

***Example**:* While planning the story “Love is in the Air”, I wanted to know why the research station the crew of the *Addington* were investigating, had been vacated. This was a perfect question for an oracle. In this case I used the Action and Theme oracles from the *Star-forged* science fiction solo roleplaying game rules.

The results were; *Action: Falter* and *Theme: Poverty*. In the context it was easy to interpret this as the station had been abandoned because the researching funding had dried up. This meant that the station had also been stripped of valuable or easily portable equipment by the research organisation.

The oracle lists included with this script include the four oracle lists from my “*A Goblin Alone*” solo rules, mentioned above.

The Code

For people who like that sort of thing, here is the python code.
You can download it from here: ***TheMultiOracle.zip***

```
#!/usr/bin/env python3
# -*- coding: utf-8 -*-
"""
Created on Saturday 28th June 2025
Creative Commons 4.0: BY-NC-SA

@author: hamy

TheMultiOraclev1c.py

Based on the General GUI using Tkinter, this script is designed to load
a csv full of various Oracle word lists.  The user can select the particular
pair of lists they want to "consult." Pressing the go-button will role
random selection from the word-lists and display the selected words.

When the program starts, it will read the last used word lists from a
seperate file so the user doesn't have to reselect each time.


"""
import os
import csv
import random
import tkinter as tk
```

```
from tkinter import ttk

BaseDir = os.getcwd()  # Just something to put in the dialog to start with
ResourceDir = os.path.join(BaseDir,"Resources")

# On loading find and look at the file which contains the last used
# word-list titles, and uses them as the starting point.
# If the file doesn't exist, it will create it and prepopulate it with
# The titles of the first two oracle lists in the oracle list csv.

class ConsultTheOracle(tk.Tk):
    def __init__(self):
        super().__init__()

        #Set up the basic variables and data.  Load the defaults.

        self.LastUsedNm = os.path.join(ResourceDir,"OracleLastUsed.txt")
        # Check file exists, and if not, create it.
        if os.path.isfile(self.LastUsedNm):
            #Load the initial selection settings
            LastUsedFile = open(self.LastUsedNm)
            LastUsedContentList=LastUsedFile.readlines()
            self.YNOracleSelProb = LastUsedContentList[0].strip("\n") #These
will be replaced by user selected list names
            self.SelOracle1 = LastUsedContentList[1].strip("\n")
            self.SelOracle2 = LastUsedContentList[2].strip("\n")
            LastUsedFile.close()

        else:
            self.YNOracleSelProb = "Either Way" #These will be replaced by user
selected list names
            self.SelOracle1 = "Actions – General"
            self.SelOracle2 = "Theme – General"
            LastUsedFile = open(self.LastUsedNm,'w')
            LastUsedFile.write("Either Way\nActions – General\nTheme – General")
            LastUsedFile.close()

        #Identify all the Oracle Table Names
        #Open the oracle list and find the first two oracle-list names.
        # Filename is Oracle_Word_Listsv0.csv
        TargOracleListNm = os.path.join(ResourceDir,"Oracle_Word_Listsv0.csv")
        OracleFile = open(TargOracleListNm)
        OracleFileContent=csv.reader(OracleFile)
        OracleFileContentList = list(OracleFileContent)
        OracleFile.close()

        #The second row is the list of Oracle List titles
        ListTitles = OracleFileContentList[1]

        #The third row is the number of members in each list
        ListCounts = OracleFileContentList[2]

        #We want to create a dictionary with the Oracle List Name as the key,
and the
        #rest of the column being loaded in as a list associated with that key.

        self.OracleDict = {}

        AvailOracles = len(ListTitles)
```

```
        #print(AvailOracles)

        #Gather the oracle lists.
        for OracleID in range (0,AvailOracles,1):

            ListBlock = []
            for ListIndex in range(int(ListCounts[OracleID])):
                ListBlock.append(OracleFileContentList[ListIndex+3][OracleID])

            self.OracleDict.update({ListTitles[OracleID]:ListBlock})


        YNOracleLikelihoodList = ["Almost Guaranteed","Very Likely",
                                  "Likely","Probably","Either Way",
                                  "Possibly","Unlikely","Very Unlikely",
                                  "Almost Impossible"]


        #Build the GUI using Tkinter

        self.title("Consult the Oracle")

        # Building The Interface
        # left frame
        self.left_frame = tk.Frame(self)
        self.left_frame.pack(side=tk.LEFT, fill=tk.BOTH, expand=True)
        tk.Label(self.left_frame, text="Yes/No Oracle",font=("Helvetica",
14)).pack(pady=20)
        tk.Label(self.left_frame, text="Likelihood of a 'Yes'
Answer").pack(pady=10)

        #Set up the dropdown
        self.YNopt = tk.StringVar(self)
        self.YNopt.set(value=self.YNOracleSelProb)
        tk.OptionMenu(self.left_frame, self.YNopt,
*YNOracleLikelihoodList).pack()


        self.YNOracle_button = ttk.Button(self.left_frame, text='Consult
Oracle', command = self.YesNoOracleGo).pack()

        tk.Label(self.left_frame, text="The Oracle Has Your Answer.  It
is...").pack(pady=10)

        self.YNOutLabel_text = tk.StringVar()
        self.YNOutLabel_text.set("Ask Your Question.")
        self.OutputLabel = tk.Label(self.left_frame,
textvariable=self.YNOutLabel_text,font=("Helvetica", 14)).pack(pady=15)

        # create a vertical separator
        self.separator = ttk.Separator(self, orient=tk.HORIZONTAL)
        self.separator.pack(side=tk.LEFT, fill=tk.Y, padx=5)

        # right frame
        self.right_frame = tk.Frame(self)
        self.right_frame.pack(side=tk.RIGHT, fill=tk.BOTH, expand=True)
        tk.Label(self.right_frame, text="Deeper Meaning
Oracle",font=("Helvetica",14)).pack(pady=20)
```

```
        tk.Label(self.right_frame, text="Select Your Oracle
Lists").pack(pady=10)

        tk.Label(self.right_frame, text="Oracle List 1").pack(pady=5)

        self.Oracle1set = tk.StringVar(self)
        self.Oracle1set.set(value=self.SelOracle1)
        self.Oracle2set = tk.StringVar(self)
        self.Oracle2set.set(value=self.SelOracle2)
        tk.OptionMenu(self.right_frame,self.Oracle1set,*ListTitles).pack()

        tk.Label(self.right_frame, text="Oracle List 1").pack(pady=5)
        tk.OptionMenu(self.right_frame,self.Oracle2set,*ListTitles).pack()

        self.YNOracle_button = ttk.Button(self.right_frame, text='Consult
Oracle', command = self.DeeperOracle).pack()

        tk.Label(self.right_frame, text="The Oracle Has Meditated and Says
This...").pack(pady=10)

        self.OracleOutLabel_text = tk.StringVar()
        self.OracleOutLabel_text.set("Prepare for Your Answer.")
        self.OracleOutputLabel = tk.Label(self.right_frame,
textvariable=self.OracleOutLabel_text,font=("Helvetica", 14)).pack(pady=15)

        self.exit_button = ttk.Button(self.right_frame, text='Quit', command =
self.CloseDown).pack()


    def CloseDown(self):
        #Save the last used selections
        LastUsedOutputTxt = self.YNopt.get() + "\n" + self.Oracle1set.get() + "\
n" + self.Oracle2set.get()

        LastUsedFile = open(self.LastUsedNm,'w')
        LastUsedFile.write(LastUsedOutputTxt)
        LastUsedFile.close()
        self.destroy()


        #Into the Left Frame add the labels, dropdown, and "go" button
        # For the Yes/No Oracle

    def YesNoOracleGo(self):
        AnswerOut = ["Absolutely No, and ...","No","No, but...","Yes,
but...","Yes","Absolutely Yes, and ..."]
        ProbTable = {"Almost Guaranteed":[0,1,5,10,80,100],
                     "Very Likely":[3,10,15,20,83,100],
                     "Likely":[5,20,25,30,85,100],
                     "Probably":[8,35,40,45,88,100],
                     "Either Way":[10,45,50,55,90,100],
                     "Possibly":[12,55,60,65,92,100],
                     "Unlikely":[15,70,75,80,95,100],
                     "Very Unlikely":[17,80,85,90,97,100],
                     "Almost Impossible":[20,90,95,99,100,100]}

        YNSelProb = self.YNopt.get()

        #Pull out the selected probability list.
```

```
        YNSelOracleWts = ProbTable[YNSelProb]

        #Select the oracle answer using provided weightings
        YNOracleAnswer = random.choices(AnswerOut,cum_weights =
YNSelOracleWts,k=1)
        self.YNOutLabel_text.set(YNOracleAnswer)

    def DeeperOracle(self):
        self.SelOracle1 = self.Oracle1set.get()
        self.SelOracle2 = self.Oracle2set.get()

        self.OracleWord1 = random.choice(self.OracleDict[self.SelOracle1])
        self.OracleWord2 = random.choice(self.OracleDict[self.SelOracle2])

        # print("Oracle ",self.SelOracle1," - yields ",self.OracleWord1)
        # print("Oracle ",self.SelOracle2," - yields ",self.OracleWord2)

        OracleDelivery1 = "Oracle: "+self.SelOracle1+" - Yields:
"+self.OracleWord1 + "\n" + "Oracle: "+self.SelOracle2+" - Yields:
"+self.OracleWord2
        # print(OracleDelivery1)
        self.OracleOutLabel_text.set(OracleDelivery1)



if __name__ == "__main__":
  app = ConsultTheOracle()
  app.mainloop()
```

It is my intention to convert this into a Godot application too. The code is offered as is, where is. If this is useful to you, great! Enjoy!

www.techmonkeybusiness.com