Hangman

11 March, 2023

Team 13 Members:

- Aijun Fan
- Dharshani Nuwanmali Rathnayake Rathnayake Mudhiyanselage
- Ke Jia
- Lakshitha Nisal Dharmarathna Kariyawasam Pathirage
- Qixiao Zhang

Code Repository:

- https://github.com/jiakeke/hangman.git

Project Management:

- https://trello.com/b/G8SOrkfG/project-of-team-13

Project 5. Hangman

Requirements:

The game:

One player thinks a word and the other needs to guess it. Every time is a wrongly guessd letter, the other player will start to draw the hanged man picture. The player who is trying to guess, wins if the hanged man picture is not finalized.

The program should have the following options to choose from: friuts, animals, car brands, countries. Eeach category should contain at least 15 words to randomly choose from. The player should be abel to select the category and than game starts.

The program should allow user to play over and over again until user chooses to quit using it. When the game has stoped the program should display number of plays and numer of wins.

35 points

!!!! Bonus points:5 for creating any kind of UI for the program.

HangmanImage, this class is in charge of drawing hangman's body and the gallows.

- This class requires PIL(Pillow) installed.

In [1]:
```
from PIL import Image, ImageDraw, ImageOps
from io import BytesIO
```

```
class HangmanImage:
    """Draws the hangman image one body part at a time following incorrect guesses."""
    def __init__(self):
        self.image = Image.new('RGB', (300,300), (255,255,255))
        self.image = ImageOps.expand(self.image, border=4, fill="black")
        self.draw = ImageDraw.Draw(self.image)
        self.draw_structure()
        self.body_parts = [self.draw_head,
                           self.draw_torso,
                           self.draw_right_arm,
                           self.draw_left_arm,
                           self.draw_right_leg,
                           self.draw_left_leg]
        self.idx = 0

    def draw_structure(self):
        self.draw.line((140,280,280,280), fill="black", width=3)
        self.draw.line((225,280,225,40), fill="black", width=3)
        self.draw.line((225,40,110,40), fill="black", width=3)
        self.draw.line((110,40,110,60), fill="black", width=3)

    def draw_next(self):
        self.body_parts[self.idx]()
        self.idx+=1

    def draw_head(self):
        self.draw.ellipse((90,60,130,100), outline='black', width=2)

    def draw_torso(self):
        self.draw.line((110,100,110,160), fill="black", width=3)

    def draw_right_arm(self):
        self.draw.line((110,125,140,120), fill="black", width=3)

    def draw_left_arm(self):
        self.draw.line((110,125,80,120), fill="black", width=3)

    def draw_right_leg(self):
        self.draw.line((110,160,135,195), fill="black", width=3)

    def draw_left_leg(self):
        self.draw.line((110,160,85,195), fill="black", width=3)
        self.draw.text((98,70),"x  x", fill="black")

    def get_encoded_image(self):
        b = BytesIO()
        self.image.save(b, format='png')
        return b.getvalue()
```

Here is the vocabulary for guessing.

In [2]:
```
words = [
    ('Animals',
        ('bat bear beaver cat cougar crab deer dog donkey duck eagle fish frog '
         'goat leech lion lizard monkey moose mouse otter owl panda python '
         'rabbit rat shark sheep skunk squid tiger turkey turtle weasel whale '
         'wolf wombat zebra')),
    ('Car brands',
        ('audi bently buick chevrolet chrysler dodge ferrari ford iveco '
         'jaguar lamborghini lincoln maybach mini nissan peugeot porsche '
         'skoda toyota volkswagen')),
    ('Colors',
        ('red orange yellow green blue indigo violet white black brown')),
    ('Countries',
```

```
        ('america canada china denmark finland france germany iceland iran '
         'italy japan netherland norway pakistan poland portugal spain '
         'srilanka sweden syria')),
    ('Fruits',
        ('apple banana blackberry blueberry cantalope cherry grape grapefruit '
         'grapes kiwi lemon lime litchi mango orange peach pear pineapple '
         'strawberry tomato watermelon')),
    ('Shapes',
        ('square triangle rectangle circle ellipse rhombus trapazoid chevron '
         'pentagon hexagon septagon octogon')),
    ]

words_dict = dict([(k, v.split()) for k, v in words])
```

This class containing the Hangman game play logic.

- Choose randow word by given category.
- Verify the letter entered by the user is correct or not.
- Calculate the number of attempts remaining for the user.
- Finally determine whether the user wins or not

In [3]:

```
import random
from collections import defaultdict


class HangmanGame:

    def __init__(self, word_category=''):
        if not word_category:
            word_category = words[0][0]
        self.word_category = word_category
        self.status = 0 # 0 = in progress, 1 = win, -1 = lose
        self.get_target_word()
        self.remaining_guesses = 6
        self.build_letter_dict()
        self.word = list("_"*len(self.target_word))

    def get_target_word(self):
        # get a random word by category for guessing
        self.target_word = random.choice(words_dict[self.word_category])

    def build_letter_dict(self):
        self.letters = defaultdict(list)
        for idx, letter in enumerate(self.target_word):
            self.letters[letter.lower()].append(idx)

    def guess(self, guess):
        # Guess whether the input letter is correct or not
        correct_guess = False
        guess = guess.description.lower()
        if len(guess) == 1:
            if guess in self.letters:
                for idx in self.letters[guess]:
                    self.word[idx] = guess
                del self.letters[guess]
                correct_guess = True
            else:
                self.remaining_guesses -= 1
                if self.remaining_guesses == 0:
                    self.status = -1
            if not self.letters:
                self.status = 1
        return correct_guess
```

This class is the primary class of this project, contains all of the UI elements for interacting with the game.

- Control the progress of the entire game.
- Generate the UI(user interface) of the entire game
- Call the HangmanImage class to draw the the hangman.
- Call the HangmanGame class to guess the letter is correct or not.

In [4]:
```
import string
from ipywidgets import widgets
from IPython.display import clear_output


class App:

    def __init__(self, game, image):
        self.game = game
        self.games = 0
        self.wins = 0
        self.image = image
        self.word_category = self.game.word_category
        self.default_margin = "20px 0px 0px 20px"
        self.get_word_category_setter()
        self.get_letter_buttons()
        self.get_app()

    def guess(self, guess):
        """
        Sends the guessed letter to the HangmanGame class and receives a response
        indicating whether or not the guess was correct.
        """
        correct_guess = self.game.guess(guess)
        if not correct_guess:
            self.image.draw_next()
        if self.game.status != 0:
            self.disable_all_letters()
            self.games += 1
            if self.game.status == 1:
                self.wins += 1

    def get_hangman_image(self):
        """Builds a widget to display the hangman image."""
        self.hangman_image = widgets.Image(value=self.image.get_encoded_image())

    def get_letter_buttons(self):
        """Builds buttons enabling the player to guess each letter of the alphabet"""
        self.letter_buttons = [widgets.Button(description=item,
                                              layout=widgets.Layout(height='25px',
                                                                    width='33px'),
                                              button_style="warning",
                                              tooltip=f"Click to guess {item}.")\
                               for item in string.ascii_uppercase]
        for button in self.letter_buttons:
            button.on_click(self.letter_button_handler)
            button.on_click(self.guess)
            button.on_click(self.get_app)
        self.letter_button_box = widgets.HBox(self.letter_buttons)

    def letter_button_handler(self, button):
        """Disables the letter buttons after they have been clicked on."""
        button.disabled = True
        button.tooltip = ""
```

```
    def enable_all_letter_buttons(self):
        """Reset all letter buttons when a new game is started."""
        for button in self.letter_buttons:
            button.disabled = False
            button.on_click(self.guess)
            button.on_click(self.get_app)

    def disable_all_letters(self):
        """Disables all of the letter buttons when the game is over."""
        for button in self.letter_buttons:
            button.disabled = True
            button.tooltip = ""

    def get_word_category_setter(self):
        """Builds a dropdown widget enabling player to set the category of the
           target word.
        """
        categories = list(words_dict.keys())
        self.word_category_setter = widgets.Dropdown(
                                        description="Category",
                                        options=categories,
                                        value=self.word_category,
                                        layout=widgets.Layout(
                                                height="auto",
                                                width='180px'))
        self.word_category_setter.observe(self.set_word_category, names=['value'])

    def set_word_category(self, _):
        """Sets the category of word and resets the game."""
        self.word_category = self.word_category_setter.value
        self.reset()

    def get_player_info(self):
        """Gets player info/stats to be displayed in the app."""
        player_name = widgets.HTML(f"<h3>Player Result</h3>",
                                   layout=widgets.Layout(margin="0px 0px 0px 100px"))
        games = widgets.HTML(f"<h4>Games: {self.games}</h4>",
                                   layout=widgets.Layout(margin="0px 0px 0px 100px"))
        wins = widgets.HTML(f"<h4>Wins: {self.wins}</h4>",
                                   layout=widgets.Layout(margin="0px 0px 0px 100px"))
        return widgets.VBox([player_name, games, wins])

    def get_header(self):
        """
        Builds the header section for the Hangman app.

        Contains:
            Secret Word:
                A blank word representing the word to be guessed.
                Letters are revealed following correct guesses.
            Remaining Guesses:
                The number of guesses remaining before the player loses the game.
            Player Result:
                The current player's wins out of games.

        The appearence of the header section is dependent on the status of
        the current game.
        """
        info = {-1:['red', self.game.target_word],
                0:['black',' '.join(self.game.word)],
                1:['green', self.game.target_word]}
        secret_word = widgets.HTML(
                        f"<h1><font color={info[self.game.status][0]}>Secret Word:\
                          {info[self.game.status][1]}</h1>",
                        layout=widgets.Layout(width="500px"))
        remaining_guesses = widgets.HTML(
```

```
                    f"<h2><font color='black'>Remaining Guesses:\
                      {str(self.game.remaining_guesses)}</h2>",
                    layout=widgets.Layout(height='auto'))
        player_info = self.get_player_info()
        self.header = widgets.HBox(
                    [widgets.VBox([secret_word, remaining_guesses]),player_info])

    def get_message(self):
        """A message that is displayed after the game is finished (win/lose)."""
        text = "You Win!" if self.game.status == 1 else "You Lose"
        return widgets.HTML(f"<h1><font color='blue'> {text} </h1>",
                            layout=widgets.Layout(margin=self.default_margin))

    def get_left_sidebar(self):
        """
        Builds the left sidebar section of the app.

        Contains:
            Win/Lose Message (Only after game has ended)
            Play Again Button (Only after game has ended)
        """
        if self.game.status == 0:
            self.left_sidebar = widgets.VBox([self.word_category_setter])
        else:
            message = self.get_message()
            play_again = self.get_play_again()
            self.left_sidebar = widgets.VBox([self.word_category_setter,
                                              message,
                                              play_again])

    def get_footer(self):
        """
        Builds the footer section of the app.

        Contains:
            Letter Buttons
        """
        footer_text = widgets.HTML(
            "<h3>Click on a letter below to guess the word.</h3>",
            layout=widgets.Layout(margin="0px 0px 0px 70px"))
        self.footer = widgets.VBox([footer_text, self.letter_button_box])
        self.footer.layout.margin = "-60px 0px 0px 0px"

    def get_play_again(self):
        """
        Creates a button allowing the player to play again.

        Button only becomes active after the game has ended.
        Clicking the button resets the app.
        """
        play_again = widgets.Button(description="Play Again",
                                    button_style="success")
        play_again.on_click(self.reset)
        play_again.layout.margin = self.default_margin
        return play_again

    def get_app(self, *args):
        """Builds and displays the app using the widgets.AppLayout template."""
        self.get_header()
        self.get_left_sidebar()
        self.get_hangman_image()
        self.get_footer()
        self.app = widgets.AppLayout(header=self.header,
                                     left_sidebar=self.left_sidebar,
                                     center=self.hangman_image,
                                     footer=self.footer)
```

```
        if self.game.status == 0 and self.game.remaining_guesses < 6:
            self.word_category_setter.disabled = True
        else:
            self.word_category_setter.disabled = False
        clear_output(wait=True)
        display(self.app)

    def reset(self, *args):
        """Resets the app to start a new game."""
        if hasattr(self, "guessed_words"):
            del self.guessed_words
        self.image.__init__()
        self.game.__init__(self.word_category)
        self.get_letter_buttons()
        self.get_app()
```

Here is the main program process

In [5]:
```
app = App(HangmanGame(), HangmanImage())
```

Secret Word: mouse

Remaining Guesses: 4

Player

Result

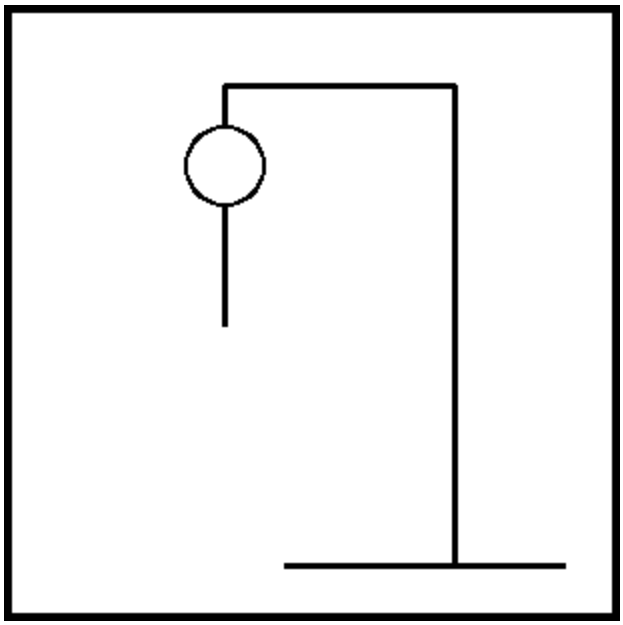
Games: 4

Wins: 4

Category Animals

You Win!

Play Again



Click on a letter below to guess the word.

A B C D E F G H I J K L M N O P Q R S T U V W X Y Z

In [ ]: