Assignment2: Pygame Hangman

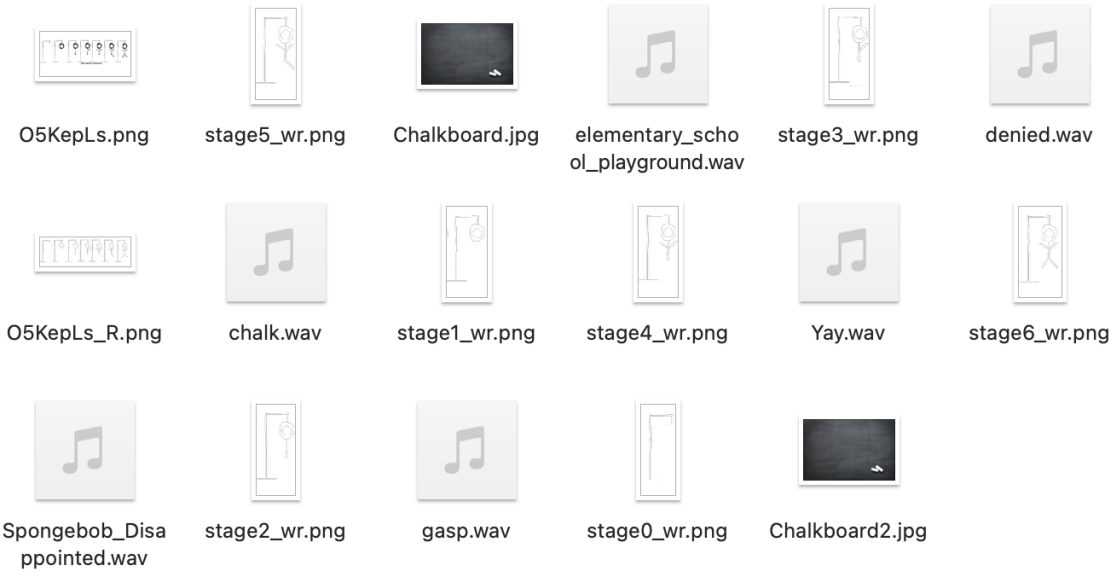
20181200 권혁채
MAS2011-01
Prof. Yongduek Seo
2022-12-26

A. Introduction

The purpose of this assignment is to graphically create the game “Hangman” using pygame, instead of stdout.

B. Programming

1. Import Packages, Create Basic Variables



These are the assets we will be using for the game. They include the background image which is Chalkboard, 6 pictures for each stage of the hangman game, and 6 sound effects.

```
import pygame
import os
import numpy as np
import math

WINDOW_WIDTH = 1350
WINDOW_HEIGHT = 900
WHITE = (255,255,255)
RED = (255, 0, 0)

current_path = os.path.dirname(__file__)
assets_path = os.path.join(current_path, 'hng_assets') #specify path to image file

background_image = pygame.image.load(os.path.join(assets_path, 'Chalkboard2.jpg')) #path to image file

words = 'ant baboon badger bat bear beaver camel cat clam cobra cougar coyote crow deer dog donkey duck ea
words = words.split()
guessed = []
repeatFlag = False
state = 0
tick = 0
```

First we must import all python packages necessary for this project. In

addition to pygame, we must import os to gain information and access to image files stored in our computer, and numpy to for random choice. Since we will be using a mouse to play this game, we will use math to detect collision between the mouse and certain buttons. Screen width and height are identical to the downscaled background image, which is 1350 x 900 pixels. All the words that will be used in the hangman game are stored in a string called words. In order to store all the words in the string as separate elements of a list, we use the split() function. The guessed variable will hold all of the letters that have already been guessed, repeatFlag indicates when the same letter has been chosen twice, state indicates which step of the hangman game we are on, and tick indicates whether or not the game has just begun or not.

2. Create Class

```
class Hangman:
    def __init__(self,name):
        self.img = pygame.image.load(os.path.join(assets_path, name + '.png')) #join connects two
        self.x = WINDOW_WIDTH / 10 + 10
        self.y = WINDOW_HEIGHT / 10 + 100
        self.width = self.img.get_width()
        self.height = self.img.get_height()

    def draw(self, screen):
        screen.blit(self.img, [self.x, self.y])

def getRandomWord(wordList): # words put in parameter
    # This function returns a random string from the passed list of strings.
    word =  np.random.choice(wordList)
    return word
```

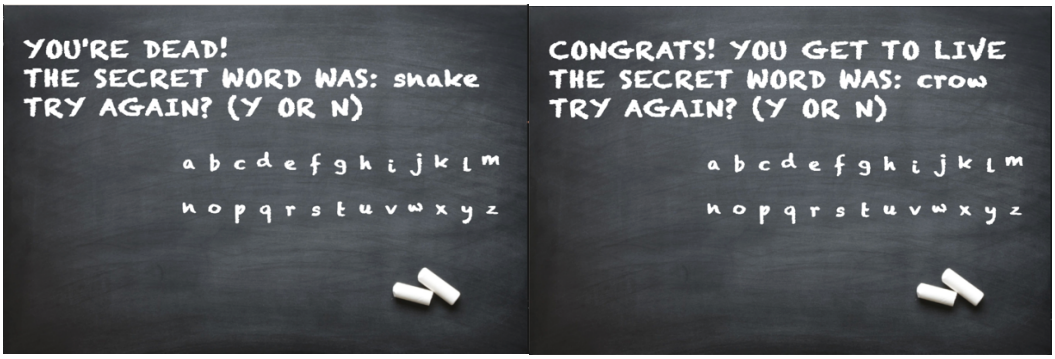
The Hangman class is a class that holds all 7 hangman state images. Therefore it loads a png file and is given a position on which it will be displayed. Since the pictures must give the illusion that one limb is added to the same picture, the locations of the pictures should be the same. The getRandomWord function uses the numpy package to randomly choose any word within the list that we split earlier.

```
def win_display():
    screen.blit(background_image, background_image.get_rect())
    text3 = word_font.render("CONGRATS! YOU GET TO LIVE" ,True, WHITE)
    screen.blit(text3, [50, 75])
    text5 = word_font.render("THE SECRET WORD WAS: " + secretWord, True, WHITE)
    screen.blit(text5, [50, 150])
    text4 = word_font.render("TRY AGAIN? (Y OR N)", True, WHITE)
    screen.blit(text4, [50, 225])

def lose_display():
    screen.blit(background_image, background_image.get_rect())
    text3 = word_font.render("YOU'RE DEAD!" ,True, WHITE)
    screen.blit(text3, [50, 75])
    text5 = word_font.render("THE SECRET WORD WAS: " + secretWord, True, WHITE)
    screen.blit(text5, [50, 150])
    text4 = word_font.render("TRY AGAIN? (Y OR N)", True, WHITE)
    screen.blit(text4, [50, 225])
```

(variable) background_image: Surface

background_image

Win_display and lose_display each contain messages that will be shown upon winning or losing the game. In order to hide the unnecessary elements, we plaster the background image on top of everything, and only show theh necessary texts.



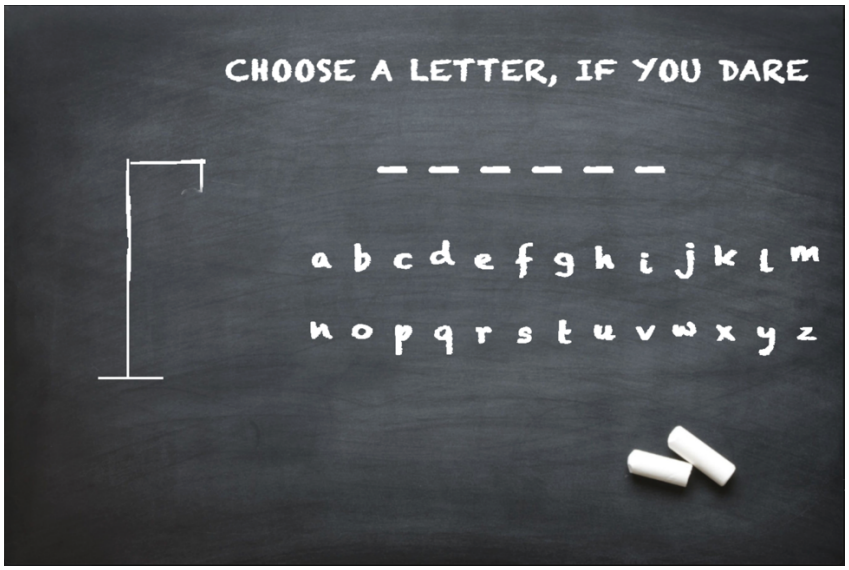
3. Initializing Necessary Variables

```
pygame.init()

pygame.display.set_caption("Hangman")
screen = pygame.display.set_mode((WINDOW_WIDTH, WINDOW_HEIGHT))
clock = pygame.time.Clock()
font = pygame.font.SysFont('chalkduster', 45, True, False) #Name of Font, Size, bold, italic
word_font = pygame.font.SysFont('chalkduster', 60, True, False)
pygame.mixer.music.load(os.path.join(assets_path, 'elementary_school_playground.wav'))
pygame.mixer.music.play(-1)

chalk_sound = pygame.mixer.Sound(os.path.join(assets_path, 'chalk.wav'))
chalk_sound.set_volume(50)
yay_sound = pygame.mixer.Sound(os.path.join(assets_path, 'Yay.wav'))
yay_sound.set_volume(0.3)
denied_sound = pygame.mixer.Sound(os.path.join(assets_path, 'denied.wav'))
gasp_sound = pygame.mixer.Sound(os.path.join(assets_path, 'gasp.wav'))
fail_sound = pygame.mixer.Sound(os.path.join(assets_path, 'Spongebob_Disappointed.wav'))

hangList = []
for i in range(7):
    hangman = Hangman('stage' + str(i) + '_wr')
    hangList.append(hangman)
```

Before we program the main part of the code, we need to load all of the assets we will use. Since this game has the background of a chalkboard, we will use the ‘chalkduster’ font. In order to make it feel like the chalkboard is in a classroom, we are going to play background sounds of an elementary school playground as the background music. In order to ensure that this sound file

repeats until the end of the game, we call play(-1). Next, we load the other 5 sound effects into their respective variables and adjust their volume. hangList contains all of the hangman stage images, their index representing which stage they are associated with.

4. Game Logic



This is an example of what the game will look like on completion. Instead of using the keyboard to type, we will display all of the alphabets on the screen and use the mouse to choose one, since way we can avoid excessive error handling.

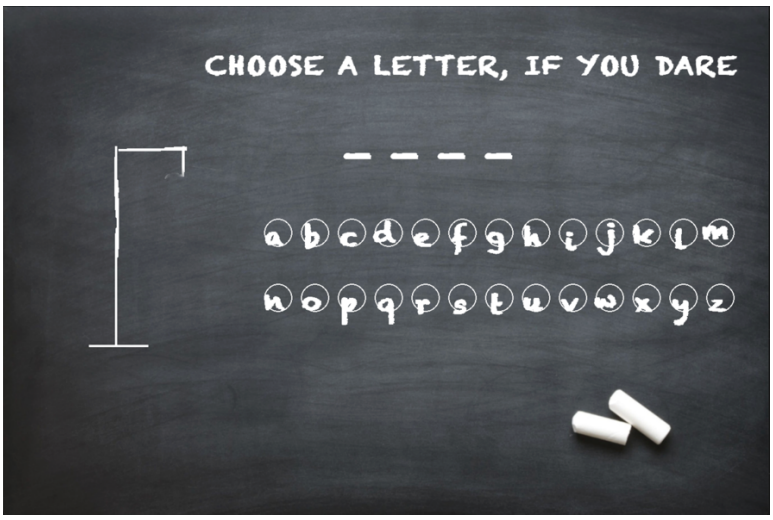
```
secretWord = getRandomWord(words)
done = False
won = True
Lost = False
#buttons
letters = []
bWIDTH = 50
bGAP = 15
startx = round(WINDOW_WIDTH - (bWIDTH + bGAP)*13) - 50 #bWIDTH + b
starty = 400
for i in range(26):
    x = startx + bGAP * 2 + ((bWIDTH + bGAP) * (i%13)) #start pos
    y = starty + ((i//13) * (bGAP + bWIDTH * 2)) #if i gets bigge
    letters.append([x,y,chr(97+i)])
```

Now let's program the actual logic of the game. First of all, we need to choose a random word from our words list. We will set three variables done, won, and Lost which each indicate whether the game is over, won, or lost. Next we must create the a~z alphabet images on the screen, and in order to do that we will create a list 'letters' which contains 3 element lists as its elements. The 3 element lists will contain the position of each alphabet and its letter. The ASCII code for lowercase a is 97, so all we have to do to get each alphabet is

add i to 97 and convert it to a character. The x position of the button is designed so that each button have a gap of at least 30 pixels, and each button will be shifted according to its index. Once the index becomes bigger than 13, it will come back to its starting position. However, if the index exceeds 13, the y position will also be incremented by a certain amount, so the 26 buttons should be aligned in 2 rows and 13 columns.

```
def update():
    for letter in letters:
        current_found = ''
        current_missed = ''
        for letter in secretWord:
            if letter in guessed:
                current_found += letter + ' '
            else:
                current_found += '_ '
        for letter in guessed:
            if letter not in secretWord:
                current_missed += letter + ' '

        text = word_font.render(current_found, True, WHITE)
        screen.blit(text, [600, 200])
        text2 = word_font.render(current_missed, True, WHITE)
        screen.blit(text2, [70,650])

        if repeatFlag == True:
            text3 = word_font.render("LETTER ALREADY CHOSEN!", True, WHITE)
            screen.blit(text3,[50,790])
        elif won == True:
            if tick != 0:
                yay_sound.play()
                win_display()
        elif won == False and Lost == True:
            fail_sound.play()
            lose_display()
        for letter in letters:
            x,y,lett= letter
        #pygame.draw.circle(screen, WHITE, (x,y), bWIDTH/2, 2)
            text = font.render(lett, True, WHITE)
            screen.blit(text, (x-text.get_width()/1.5,y - text.get_width()/1.5))
```

The update function decides what will be shown in the current game screen. Since the screen always displays the found letters and missed letters, using the guessed[] list we defined earlier, we create two strings current_found and current_missed and add all of the letters that are in guessed[] and secretWord to current_found, and the ones that are not in secretWord to current_missed. Then we render the texts in their appropriate positions. After that, if there has been a repeated input we display the words LETTER ALREADY CHOSEN! If the game has either been won or lost, we call win_display or lose_display to create the appropriate screens. Since the won and Lost variables are initialized as True and False, the first elif loop runs at the beginning of the program. In order to prevent this, we add a tick variable to ensure that this doesn't happen. Finally, the alphabet buttons we created earlier will be drawn on the screen using their respective x,y positions.



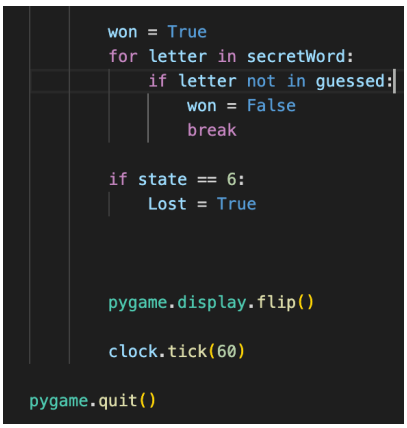
These circles represent the area in which each letter can be detected.

```
while not done:
    # 이벤트 반복 구간
    if tick == 0:
        print(secretWord)
    screen.blit(background_image, background_image.get_rect())

    text = font.render("CHOOSE A LETTER, IF YOU DARE", True, WHITE)
    screen.blit(text, [WINDOW_WIDTH/3.8, WINDOW_HEIGHT/12])
    hangList[state].draw(screen)

    update()
    tick +=1
```

This while loop contains the logic of the game during runtime. To allow the user to know what the answer is if they wish, we will print the secretWord on the terminal at the beginning of each game. After that, we will load the background image and render the words CHOOSE A LETTER, IF YOU DARE at the top of the screen. The state was initialized as 0, so in the beginning the first Hangman image will be displayed. After this, the update function will be called to print the remaining parts.

```
for event in pygame.event.get():
    if event.type == pygame.QUIT:
        done = True
    elif event.type == pygame.MOUSEBUTTONDOWN:
        repeatFlag = False
        xCoord, yCoord = pygame.mouse.get_pos()
        for letter in letters:
            x, y, lett= letter
            d = math.sqrt((x-xCoord)**2 + (y-yCoord)**2)
            if won == True or (won == False and Lost == True):
                if d < bWIDTH/2 :
                    if lett == 'y':
                        chalk_sound.play()
                        guessed = []
                        won = False
                        Lost = False
                        state = 0
                        tick = 0
                        secretWord = getRandomWord(words)
                    elif lett == 'n':
                        chalk_sound.play()
                        done = True
            else:
                if d < bWIDTH/2 :
                    if lett in guessed:
                        denied_sound.play()
                        repeatFlag = True
                    else:
                        chalk_sound.play()
                        guessed.append(lett)
                        if lett not in secretWord:
                            gasp_sound.play()
                            state += 1
```

During the game, every time the player holds down the mouse button we will get the position on which the mouse was clicked. If the position of the mouse was within the radius of any of the letters, something will happen. First of all, if the game has been finished, there will be the words TRY AGAIN? (Y OR N) on the screen. In this case, if the mouse clicks y, we will change all of the variables to their initial states again and get a new random word from the words list. If the player clicks n, done will become true and the while loop will end, ending the game. If the game hasn't finished, the letter will first be checked if it is in the guessed list and if it is, the repeatFlag will become true, and the while loop will start over, producing the denied_sound effect from the update() function. If the letter wasn't in the guessed list, the new letter will first be added to the guessed list. If the new letter is not in secretWord, then a gasp sound effect will play and the hangman state will move onto the next one. Every time a new word is selected, the chalk sound effect will play.



Finally, if all letters in secretWord are in guessed, the game will be decided to be won. If this is not the case and hangman is at the final state, the game will be decided to be lost. The framerate is set to 60 fps, and if the while loop ever ends, the program will quit.