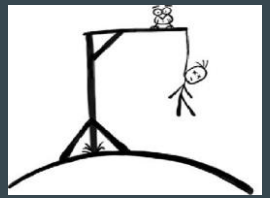
Hangman Project

...

by Swam Htet Lin, Daniel, Yernar and Arman



- INTRODUCTION ABOUT HANGMAN PROJECT
- THE BACKGROUND FUNCTION OF CODE
- HOW GAME GRAPHIC WORK
- HOW WE IMPORT WORD AND BACKGROUND MUSIC
- CONCLUSION

language learning

pronunciation

What skill does Hangman teach?

concentration

spelling

Vocabulary

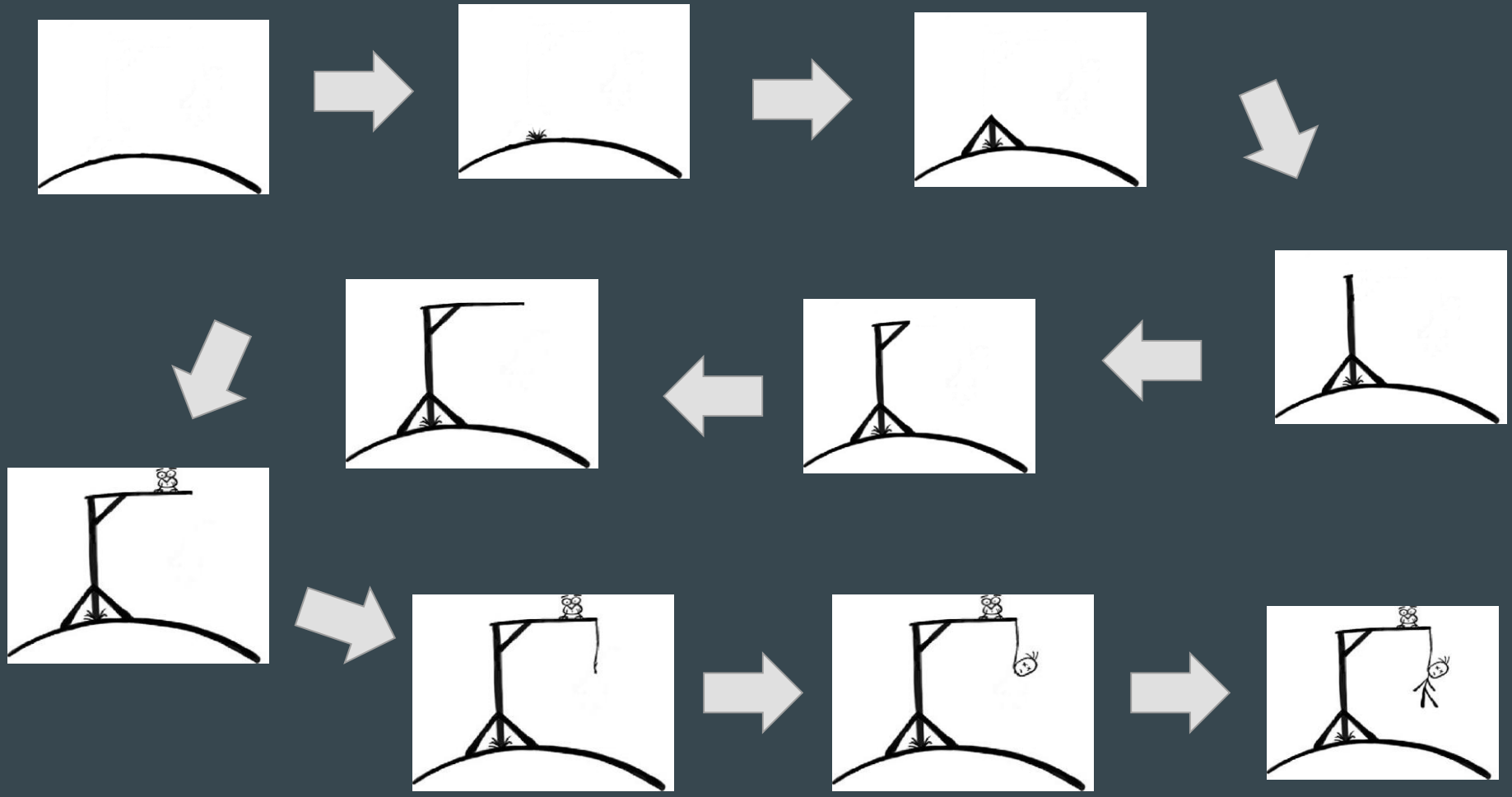
THE BACKGROUND FUNCTION OF CODE

```python
LIVES = 9

class Hangman:

    def __init__(self, word, alphabet):
        self.dictionary = {}
        self.word = word
        self.error_count = 0
        self.init_dictionary(alphabet)
        self.remaining_letters_count = len(self.dictionary.keys())
        self.previous_plays = []

    def init_dictionary(self, alphabet):
        if len(self.word) > 20:
            raise Exception("Input length must be less or equal to 20 characters.")
        for i in range(0, len(self.word)):
            if self.word[i] == ' ':
                continue
            elif self.word[i] not in alphabet:
                raise Exception(self.word[i], " not in alphabet.")
            elif self.word[i] not in self.dictionary.keys():
                self.dictionary[self.word[i]] = [i]
            else:
                self.dictionary[self.word[i]].append(i)
```



```
    def play(self, letter):
        if self.has_won() or self.has_lost():
            return "Oops! The game is over!"
        elif letter in self.previous_plays:
            return "You have already selected " + letter + "."
        elif letter in self.dictionary.keys():
            self.previous_plays.append(letter)
            self.remaining_letters_count -= 1
            return self.dictionary.get(letter)
        else:
            self.previous_plays.append(letter)
            self.error_count += 1
            return "Oops! " + letter + " is not part of the solution."

    # Losing condition.
    def has_lost(self):
        return self.error_count == LIVES

    # Winning condition.
    def has_won(self):
        return self.remaining_letters_count == 0

if __name__ == "__main__":
    alphabet = ["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"]
    hangman = Hangman("LAGARTIJO", alphabet)
```

```
# English alphabet.
alphabet = ["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"]

# Game loop...
while True:
    # Initialize Hangman Model with word (new) & alphabet. Must validate all characters are within alphabet.
    while True:
        word = None
        try:
            word = choice(word_list)  # Use random.choice to pick a random word from the list
            word = word.upper()
            hangman = Hangman(word, alphabet)
            break
        except:
            if word is None:
                print("No word was generated.")
            else:
                print(word + " is not valid.")
```

HOW GAME GRAPHIC WORKS

You can't beat it!

_ _ _ _ _ _ _ _ _

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				

```python
import pygame as pg
import os

DEF_GAME_TITLE = "You can't beat it!"
BACKGROUND_COLOR = (255, 255, 255) # WHITE
FONT_COLOR = (0, 0, 0)
BUTTON_PRESSED = (220, 220, 220)
BUTTON_ACTIVE = (240, 230, 140)
IMAGES_COUNT = 9
IMAGES_FORMAT = ".png"
KEYS = 26
PADDING = 5

class HangmanView:

    def __init__(self, width, height):
        self.width = width
        self.height = height
        pg.init()
        self.font = pg.font.SysFont('Arial', 25)
        self.init_screen()
        self.load_images()
        self.init_buttons()
        self.draw_reload_button()

    # Creates a display with a given width and height.
    def init_screen(self):
        self.screen = pg.display.set_mode([self.width, self.height])
        self.screen.fill(BACKGROUND_COLOR)
        pg.display.set_caption(DEF_GAME_TITLE)
        pg.display.update()
        self.init_chars_y = 0.2 * self.height
```

```
# def reset_display(self):

# Loads the images for the Hangman.
def load_images(self):
    cdir = os.path.dirname(os.path.realpath(__file__))
    cdir = os.path.join(cdir, "images")
    self.image_list = []
    for i in range(0, IMAGES_COUNT + 1):
        path = str(i) + IMAGES_FORMAT
        path = os.path.join(cdir, path)
        img = pg.image.load(path)
        self.image_list.append(pg.transform.scale(img, (int(self.width / 3), int(self.height / 2))))
    self.reload_image = pg.transform.scale(pg.image.load(os.path.join(cdir, "reload.png")), (int(self.width / 15), int(self.width / 15)))

# Draws an image corresponding to the score on the left pane.
def update_image(self, count):
    self.screen.blit(self.image_list[count], (int(self.width * 0.02), int(self.height * 0.1)))
    pg.display.update()

# Initializes default dimensions for the buttons related to the screen size.
def init_buttons(self):
    self.init_buttons_x = int(self.width * 0.38)
    self.init_buttons_y = int(self.height * 0.5)
    self.button_width = int(self.width * 0.6 * 0.1)
    self.button_height = self.button_width
    self.reload_button_x = int(self.width * 0.9)
    self.reload_button_y = int(self.height * 0.9)
```

```python
# Draws a particular button given its value (consecuently its position), color and value.
def draw_button(self, index, color, value):
    x = index % 10
    y = int(index / 10)
    pg.draw.rect(self.screen, color, (self.init_buttons_x + x * self.button_width,
                                      self.init_buttons_y + y * self.button_height,
                                      self.button_width - PADDING,
                                      self.button_height - PADDING))
    self.screen.blit(self.font.render(value, True, FONT_COLOR),
                     (self.init_buttons_x + x * self.button_width + 2 * self.button_width / 5,
                      self.init_buttons_y + y * self.button_height + self.button_height / 5))
    pg.display.update()

def draw_reload_button(self):
    self.screen.blit(self.reload_image, (self.reload_button_x, self.reload_button_y))

def draw_character(self, position, character):
    x = position % 10
    y = int(position / 10)
    if character == '_':
        character = '____'
        self.screen.blit(self.font.render(character, True, FONT_COLOR),
                         (self.init_buttons_x + x * self.button_width + 2 * self.button_width / 5,
                          self.init_chars_y + y * self.button_height + self.button_height / 5))
    else:
        self.screen.blit(self.font.render(character, True, FONT_COLOR),
                         (self.init_buttons_x + x * self.button_width + 2 * self.button_width / 5 + int(self.button_width * 0.25),
                          self.init_chars_y + y * self.button_height + self.button_height / 5))
    pg.display.update()

def clear_word(self):
    pg.draw.rect(self.screen, BACKGROUND_COLOR, (int(0.4 * self.width), 0, int(0.6 * self.width), 0.5 * self.height))
    pg.display.update()

def draw_feedback(self, feedback):
    if len(feedback) > 50:
        raise Exception("Feedback length must be less or equal to 20 characters.")
    else:
        pg.draw.rect(self.screen, BACKGROUND_COLOR,
                     (PADDING * 5, self.height - 10 * PADDING, self.width * 0.9 - 5 * PADDING , 10 * PADDING))
        self.screen.blit(self.font.render(feedback, True, FONT_COLOR),
                         (PADDING * 5, self.height - 10 * PADDING))
        pg.display.update()
```

```
    # Mouse-Event Scanner.
    def button_pressed_scanner(self):
        for event in pg.event.get():
            if event.type == pg.QUIT: # Exit was pressed.
                quit()
            if event.type == pg.MOUSEBUTTONDOWN:
                (x, y) = pg.mouse.get_pos()
                if x >= self.reload_button_x and y >= self.reload_button_y:
                    return -1
                for i in range(0, KEYS):
                    column = i % 10
                    row = int(i / 10)
                    if x >= self.init_buttons_x + column * self.button_width and x <= self.init_buttons_x + column * self.button_width + self.button_width - PADDING:
                        if y >= self.init_buttons_y + row * self.button_height and y <= self.init_buttons_y + row * self.button_height + self.button_height - PADDING:
                            return row * 10 + column
            return None

if __name__ == "__main__":
    my_hangman_view = HangmanView(1000, 600)
    my_hangman_view.load_images()
    my_hangman_view.init_buttons()
    my_hangman_view.update_image(9)
    while 1:
        my_hangman_view.button_pressed_scanner()
```

HOW WE IMPORT WORD AND BACKGROUND MUSIC

OxfordRequest.py > ...

```python
import requests
from random import randint as r
import json  # Make sure to import the json module

API_ID = 'your_api_id'
API_KEY = 'your_api_key'
LANG = 'en'
URL = 'https://od-api.oxforddictionaries.com:443/api/v2/entries' + LANG + '/registers=Rare;'

class OxfordRequest:
    def __init__(self):
        response = requests.get(URL, headers = {'app_id': API_ID, 'app_key': API_KEY})
        if response.status_code != 200:
            print(f"API request failed with status code {response.status_code}")
            print(f"Error message: {response.text}")
            self.dictionary = None
            self.total_words = 0
        else:
            try:
                self.dictionary = response.json()
                self.total_words = len(self.dictionary.get("results", []))
            except json.JSONDecodeError:
                print("An error occurred while decoding the JSON response:")
                print(response.text)
                self.dictionary = None
                self.total_words = 0

    def get_random_word(self):
        if not self.dictionary or not self.total_words:
            print("Cannot get a random word because the dictionary is empty or not initialized.")
            return None
        try:
            random = r(0, self.total_words - 1)
            return self.dictionary["results"][random].get("word")
        except Exception as e:
            print("An error occurred while getting a random word:")
            print(str(e))
            return None

```


word_list = [
'wares',
'soup',
'mount',
'extend',

```python
import pygame
from pygame import mixer

pygame.init()
pygame.mixer.init()

pygame.mixer.music.load("background.mp3")
pygame.mixer.music.set_volume(0.5)  # Adjust the volume as needed
pygame.mixer.music.play(-1)  # -1 plays the music on an infinite loop
pygame.quit()
```

ကျေးဇူးတင်ပါသည်

Paxмem

Thanks for attention

Paxмam

We hope you will have a great summer holiday