MATH TUG WAR
FINAL SEMESTER EXAM PROJECT REPORT

(Prepared to Fulfill the Requirements for Passing the Final Exam Semester 1)

By:

Ababil khoerun Imam (24031554001)

Alena Destria Pramesti (24031554030)

Citra Fanysa (25031554015)

Lecturer:

Hasanuddin Al-Habib, S.Si., M.Si

STUDY PROGRAM DATA SCIENCE
FACULTY OF MATHEMATICS AND NATURAL SCIENCES
STATE UNIVERSITY OF SURABAYA
2025

TABLE OF CONTENTS

CHAPTER 1……………………………………………………………………………………. **2**
INTRODUCTION………………………………………………………………………….... **2**
1.1 BACKGROUND……………………………………………………………………...2
1.2 PROBLEM STATEMENT……………………………………………………………2
1.3 OBJECTIVES…………………………………………………………………………3
CHAPTER 2……………………………………………………………………………………. **4**
ANALYSIS AND DESIGN UI/UX…………………………………………………………4
2.1 APPLICATION NECESSITY ANALYSIS……………………………………..4
2.2 FLOWCHART…………………………………………………………………………5
CHAPTER 3……………………………………………………………………………………..**7**
IMPLEMENTATION…………………………………………………………………………. **7**
3.1 CODE EXPLANATION……………………………………………………………..8
3.2 APPLICATION SCREENSHOT…………………………………………………….23
CHAPTER 4……………………………………………………………………………………...**27**
ATTACHMENT………………………………………………………………………………..**27**
4.1 EMBED FILE…………………………………………………………………………27
REFERENCES…………………………………………………………………………………...**28**

CHAPTER 1

INTRODUCTION

1.1 BACKGROUND

Mathematics education often faces challenges in attracting students' interest, especially in concepts that are considered abstract such as fractions, decimals, and operations involving numbers. Conventional learning approaches are sometimes less able to visualize the dynamics of calculations and relationships between numbers, leading to difficulty in understanding and low learning motivation.

As technology has evolved, gamification (the application of game elements in non-game content) has proven to be effective in increasing engagement and retention of subject matter. Educational games offer an interactive, challenging, and instant feedback environment, which is essential in the teaching-learning process.

The project "Math Tug War" was developed as a solution to overcome these challenges. This game adopts a visual competitive mechanism such as tug-of-war, where speed and accuracy in answering math problems (fractions and decimals) will determine the movement of the "math" on the "tug" towards victory. Using the Pygame framework, the project aims to create a fun learning experience while honing your numeracy skills quickly and accurately.

1.2 PROBLEM STATEMENT

The main objectives of the development of the Math Tug War are:

1. **Improve Understanding:** Provides an interactive platform to train and reinforce the understanding of mathematical operation concepts, specifically fractions and decimals.
2. **Increases Counting Speed:** Practicing quick numeracy skills under time pressure, which is a crucial skill in math.
3. **Provides Instant Feedback:** Provides visual and audio notifications as soon as players enter answers, allowing for self-correction and more effective learning.
4. **Creating Healthy Competition:** Encouraging competition through *a transparent and accessible* leaderboard system.

1.3 OBJECTIVES

The scope of the project includes the following aspects:

1. **Development Platform**: The game is developed using the Python programming language with the Pygame library.

2. **Focus of Matter:** Mathematical operations are limited to addition, subtraction, multiplication, and division for simple fractional and decimal numbers.

3. **Game Mode:** The game is *a single-player time attack* with the aim of recording the highest possible score, measured by the distance of movement of the marbles and the number of correct answers.

4. **Core Features:** Includes *state* management (Main Menu, Name Input, Game, Leaderboard, Game Over), automatic question generator, *game timer*, and local score storage.

CHAPTER 2

LITERATURE REVIEW AND THEORETICAL FOUNDATIONS

2.1 APPLICATION NECESSITY ANALYSIS

Educational games, or *edutainment*, are a genre designed to educate as well as entertain. The main principles applied in this game are:

2.1.1 Games a Media of Learning

- **Intrinsic Motivation:** The elements of score, challenge, and progress visualization (moving marbles) serve as intrinsic motivational drivers for players to keep learning.
- **Active Learning:** The player does not passively receive information, but rather actively solve problems and input inputs, which strengthens the cognitive neural pathways.
- **Structured Repetition:** This game provides automatic repetition of practice problems, which is essential for math mastery, but is presented in a dynamic and not boring format

2.1.2 Indonesian Student's Fractions and Decimals

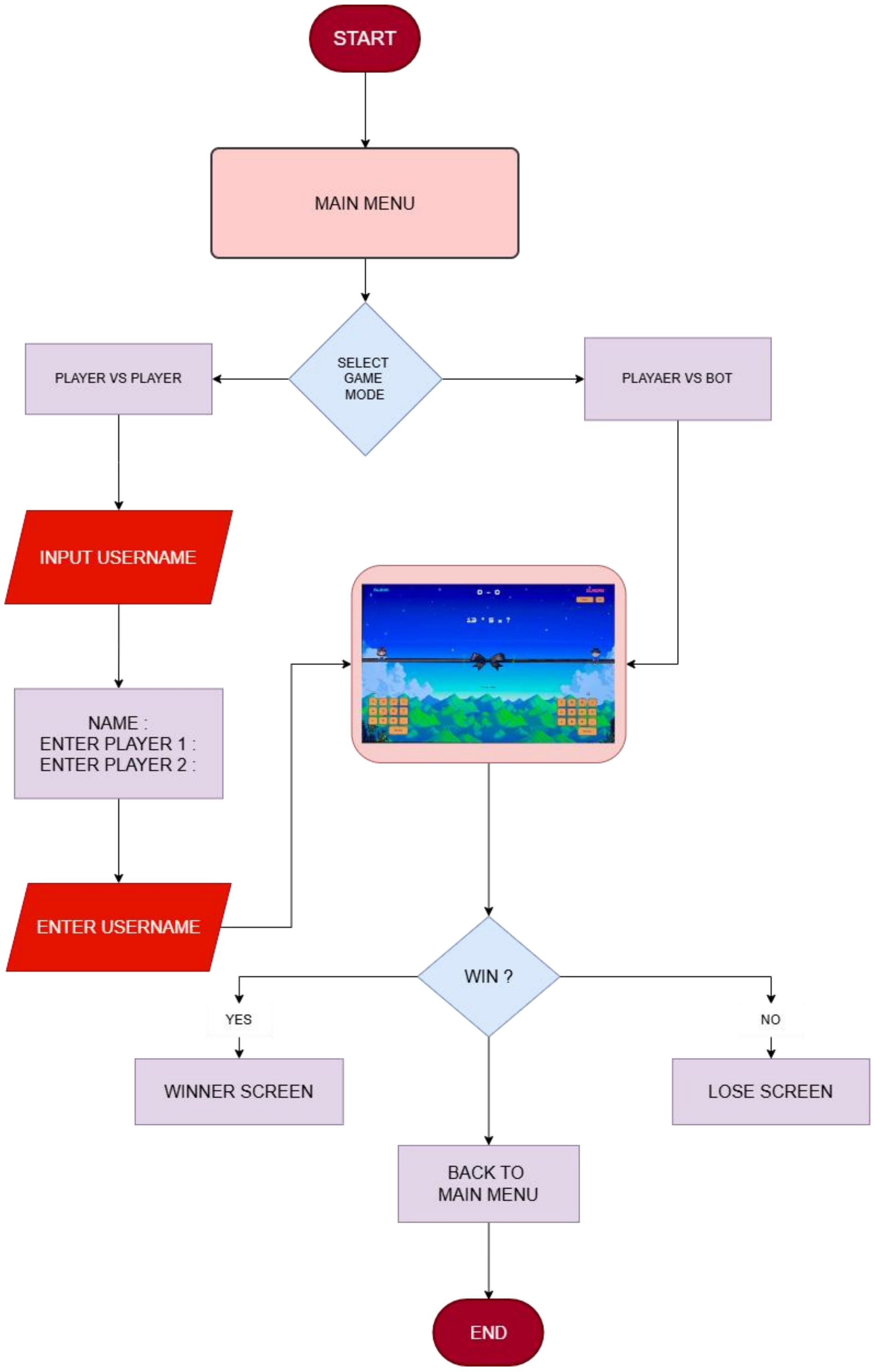
The main focus of the game is on Fractional and Decimal numbers, which is an important foundation before moving on to algebra and calculus.

- **Fractions:** Represented in `a/b format`. The game must be able to generate problems that involve basic fraction operations and receive answer input in fractional or decimal form. The `Fraction` class of Python is used as a basis for calculation to ensure the mathematical accuracy of the questions and answer keys.
- **Decimal:** Represented by the use of the dot character (`.`) as a decimal separator.
- **Accuracy vs. Speed:** The challenge is designed to strike a balance between the speed of the input and the accuracy of the answer to get the maximum points.

2.1.3 Pygame Development Framework

- **Accessibility:** Easy to implement and relatively lightweight, perfect for logic-focused educational games.
- **Sprite functionality:** Makes it easy to manage visual elements such as marbles, strings, buttons, and text.
- **State Management:** A code structure that allows for seamless transitions between screens/game conditions (Menu, Game Play, Leaderboard) using the *Finite State Machine* (FSM) system.

2.2 FLOWCHART



2.3 USER INTERFACE DESIGN SKETCH

2.1 Main Menu View

Element UI:

- Game title
- knob:
    - *Player vs Player*
    - *Player vs BOT*
    - *Leaderboard*

2.2 Name Input Display

Element:

- Textbox to fill in the player's name
- "Continue" or "Start" buttonombol

2.3 Tampilan Gameplay

Element UI:

- Math problems (big and clear)
- Textbox answers
- Timer
- Draw rope (rope bar)
- Left and right characters
- True/false answer sound effects
- Soft color background so as not to interfere with focus
- Error indicators
- The question is in the upper center so that it is immediately visible
- Answer input is just below the question
- The rope is at the bottom as an indicator of victory
- the left and right areas show the player and the BOT

2.4 Game Over View

Element UI:

- Text "Winner: <name>"
- Knob:
    - *Replay*
    - *Back to Menu*

2.5 Leaderboard View

Element:

- Player score order
- Average BOT time
- Back button

1. ***STATE_MAIN_MENU:*** The game's main entry point. From here, players can choose to start, view the leaderboard, or exit.
2. ***STATE_NAME_INPUT*****:** Mandatory status before playing. Once the name is entered, the status switches to Game Play.
3. ***STATE_GAME_PLAY:*** The core state in which the problem is generated, the running time, and the movement of the marbles occur based on the player's input.
4. ***STATE_GAME_OVER:*** Status is reached when the final condition of the game is met (the marble reaches the limit or the time runs out). Players can return to the main menu or view the leaderboard.
5. ***STATE_LEADERBOARD*****:** Passive status to display the best score, which can be accessed from the Main Menu or after the Game Over.

STATE	DESCRIPTION	TRANSITION
STATE_MAIN_MENU (0)	The game's initial view, containing the Start, Leaderboard, and Exit buttons.	STATE_GAME_PLAY
STATE_NAME_INPUT (1)	A screen for players to enter their name/initials before playing.	STATE_GAME_PLAY
STATE_GAME_PLAY (2)	The main game logic, featuring questions, timers, and player input.	STATE_GAME_OVER
STATE_LEADERBOARD (3)	Displays a list of the top 10 saved scores.	Back to STATE_MAIN_MENU
STATE_GAME_OVER (4)	The end creen, displaying the score, and the option to return to the menu.	Back to STATE_MAIN_MENU

3.1 User Interface (UI) Design

The interface is designed to be minimalist yet informative, with a bright and user-friendly aesthetic.

- **Main Menu:** Using large, high-contrast buttons (for example, black text on a light gray background).
- **Game Screen:**
- **Question Area:** Math problems are clearly displayed in the upper center.
- **Input Area:** An interactive text box for entering answers.
- **Rope Marble Visualization:** A horizontal line (rope) with a marble object in the center. The marbles will shift to the right (win) each time the answer is correct, and may shift slightly to the left or stop if the answer is wrong/time up, illustrating the Tug-of-War mechanic.
- **Time Indicator:** A bar or number that visually shows the remaining time per question
- **Score/Distance:** Text that displays the current score or the position of the marbles.

3.2 Core Game Mechanics (Rope Marbles)

The game's mechanics are based on a fast-paced question-and-answer cycle:

- **Question Generation:** Each question is randomly generated, ensuring a fraction/decimal combination that results in a reasonable answer.
- **Timer:** Each question has a time limit (e.g., 10-15 seconds). The remaining time is the determining factor in the bonus score.
- **Player Input:** The player enters an answer. The game must process input as fractions (example: 1/2) or decimals (example: 0.5).
- **Feedback and Movement:**
- **Correct Answer:** The marbles move forward (to the right) for a certain distance (e.g., 50 units). Points are earned based on distance and time remaining.
- **Wrong Answer/Time Out:** The marble stops or can move backwards slightly, reducing the total time of the game.
- **Game Over Condition:** The game ends when the marbles reach the victory limit on one side of the screen, or when the total game time (if any) runs out.

3.3 Data Retention Scheme (Leaderboard)

The highest score data is stored locally in JSON format. The data schema used is an array of objects, with each object containing:

FIELD	TIPE DATA	DESKRIPSI
name	string	Player's name/initial.
Score	integer	The final score achieved (accumulated points from the correct answer).
Time stamp	string	The time and date of the score.

CHAPTER 3

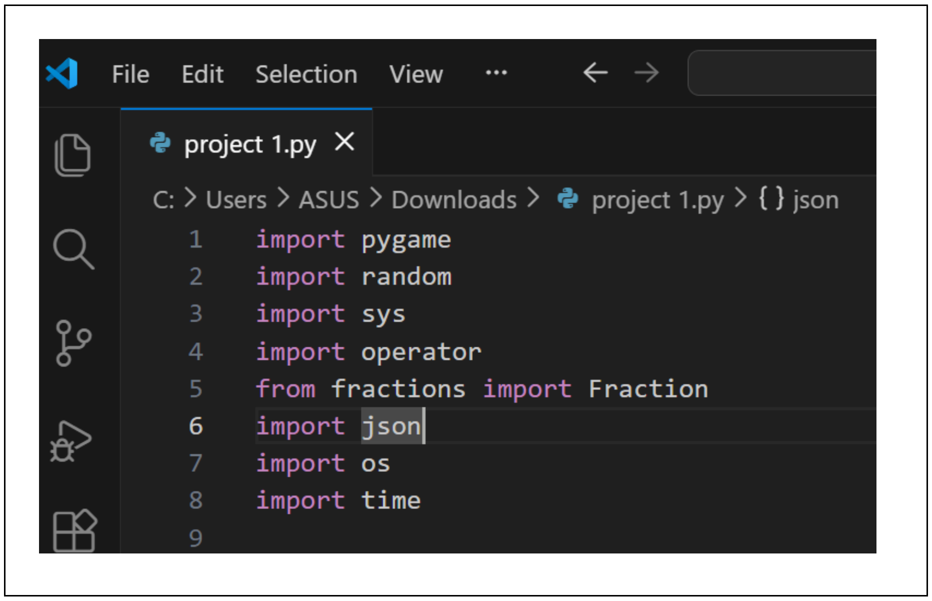
IMPLEMENTATION

3.1 CODE EXPLANATION

Game Architecture: *State Machine*

The game is organized based on a *state machine* model to set up Logical gameplay of the game:

A. Impor Library

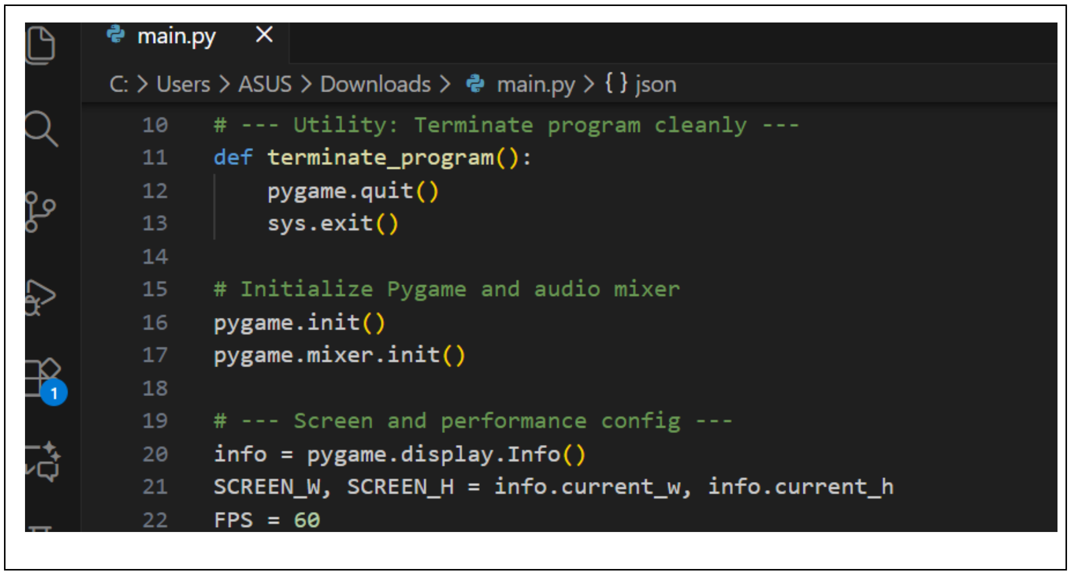


```python
import pygame
import random
import sys
import operator
from fractions import Fraction
import json
import os
import time

```

Description: Imports all necessary Python modules and libraries (pygame, random, sys, operator, fractions, json, os, time).

- `pygame`: The main framework for graphics, input, and sound.
- `sys`: Used for `the sys.exit()` function to stop the program cleanly.
- `operators`: Provides functions such as `operator.add`, `operator.sub`, which are used to perform mathematical operations dynamically in `the Game` class.
- `fractions. Fraction`: **The most crucial.** It is used to ensure mathematical calculations (especially divisions and fractions/decimals) have absolute precision and avoid *floating-point errors*.
- `json` and `os`: For file management (reading and writing scores to `score.json`).
- `time`: Used to record a *timestamp* when a new score is reached.

B. Program Termination & Initialization


main.py

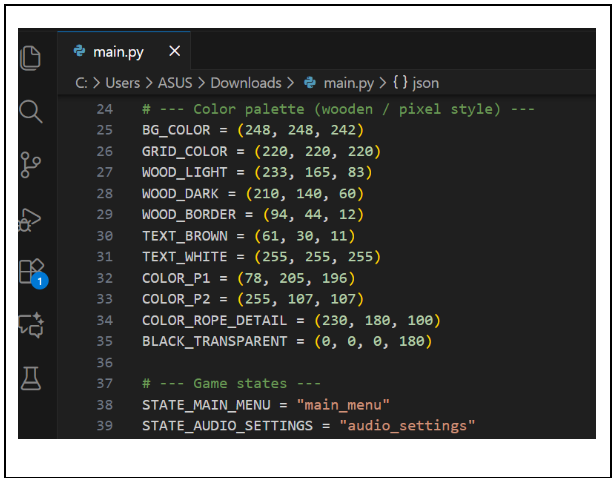
C: > Users > ASUS > Downloads > main.py > { } json

```python
# --- Utility: Terminate program cleanly ---
def terminate_program():
    pygame.quit()
    sys.exit()

# Initialize Pygame and audio mixer
pygame.init()
pygame.mixer.init()

# --- Screen and performance config ---
info = pygame.display.Info()
SCREEN_W, SCREEN_H = info.current_w, info.current_h
FPS = 60
```

Description: Definition of the function terminate_program(). Initialize Pygame and sound mixers. This part prepares the Pygame and ensures the closure is neat.

- `terminate_program()`: A mandatory function that calls `pygame.quit()` (release Pygame resources) and `sys.exit()` (terminate the script). This ensures the program comes out without leaving the process running.
- `pygame.init()`: Initializes all Pygame modules.
- `pygame.mixer.init()`: Initializes the audio module to play sound *effects*

C. Konfigurasi Global & Konstanta


main.py

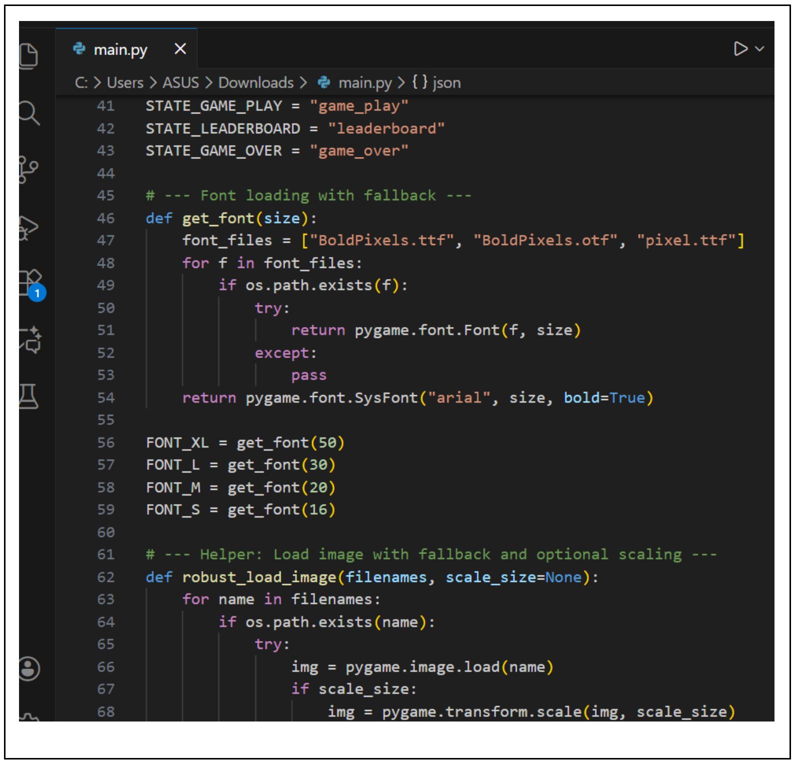
C: > Users > ASUS > Downloads > main.py > { } json

```python
# --- Color palette (wooden / pixel style) ---
BG_COLOR = (248, 248, 242)
GRID_COLOR = (220, 220, 220)
WOOD_LIGHT = (233, 165, 83)
WOOD_DARK = (210, 140, 60)
WOOD_BORDER = (94, 44, 12)
TEXT_BROWN = (61, 30, 11)
TEXT_WHITE = (255, 255, 255)
COLOR_P1 = (78, 205, 196)
COLOR_P2 = (255, 107, 107)
COLOR_ROPE_DETAIL = (230, 180, 100)
BLACK_TRANSPARENT = (0, 0, 0, 180)

# --- Game states ---
STATE_MAIN_MENU = "main_menu"
STATE_AUDIO_SETTINGS = "audio_settings"
```

Description: Specifies screen resolution, FPS, color definition (RGB), and constants for game state (GAME STATES).

- **Screen Resolution:** Gets the current monitor screen resolution (`SCREEN_W`, `SCREEN_H`) so that games can run in adaptive *fullscreen* mode.
- **FPS (Frames Per Second):** Determines the speed at which *the game loop* runs (for example, 60 times per second).
- **Color (RGB):** Defines a memorable color constant (e.g., `WHITE`, `BLACK`, `RED_LIGHT`) in an RGB tuple format.
- **GAME STATES:** An integer constant (`STATE_MAIN_MENU = 0`, etc.) that functions as a state machine (*Finite State Machine*). The `current_state` variable (in the Main Loop) uses this constant to determine which screens to draw and which logic to run.

D. Score Data Management (JSON)

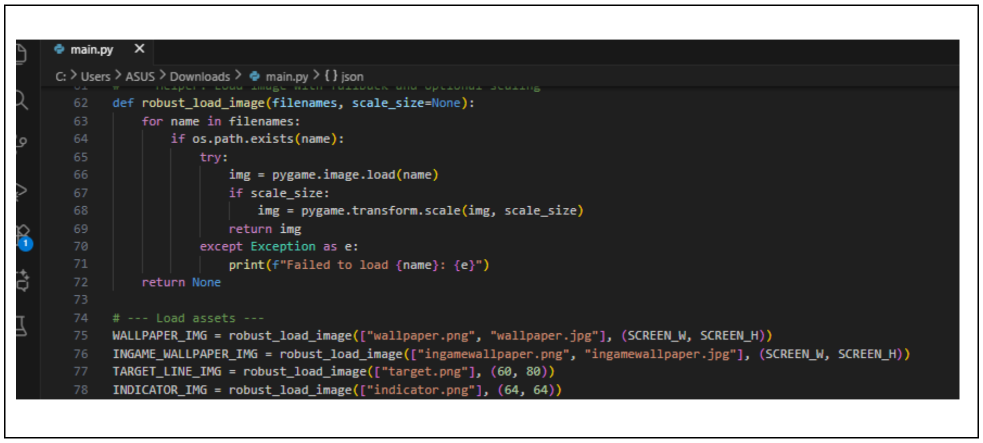


```python
STATE_GAME_PLAY = "game_play"
STATE_LEADERBOARD = "leaderboard"
STATE_GAME_OVER = "game_over"

# --- Font loading with fallback ---
def get_font(size):
    font_files = ["BoldPixels.ttf", "BoldPixels.otf", "pixel.ttf"]
    for f in font_files:
        if os.path.exists(f):
            try:
                return pygame.font.Font(f, size)
            except:
                pass
    return pygame.font.SysFont("arial", size, bold=True)

FONT_XL = get_font(50)
FONT_L = get_font(30)
FONT_M = get_font(20)
FONT_S = get_font(16)

# --- Helper: Load image with fallback and optional scaling ---
def robust_load_image(filenames, scale_size=None):
    for name in filenames:
        if os.path.exists(name):
            try:
                img = pygame.image.load(name)
                if scale_size:
                    img = pygame.transform.scale(img, scale_size)
```

Description: Function to load and store leaderboard data from/to JSON file (score.json)

- `load_leaderboard()`: Trying to open `score.json` file. If the file is missing or corrupted (`json error. JSONDecodeError`), it returns an empty list. This is important so that the program does not *crash* on the first installation.
- `save_leaderboard()`: Converts score data to JSON format, sorts, and saves only the top 10 scores into `score.json file`.

E. Player Data Classes/Structures



main.py ×

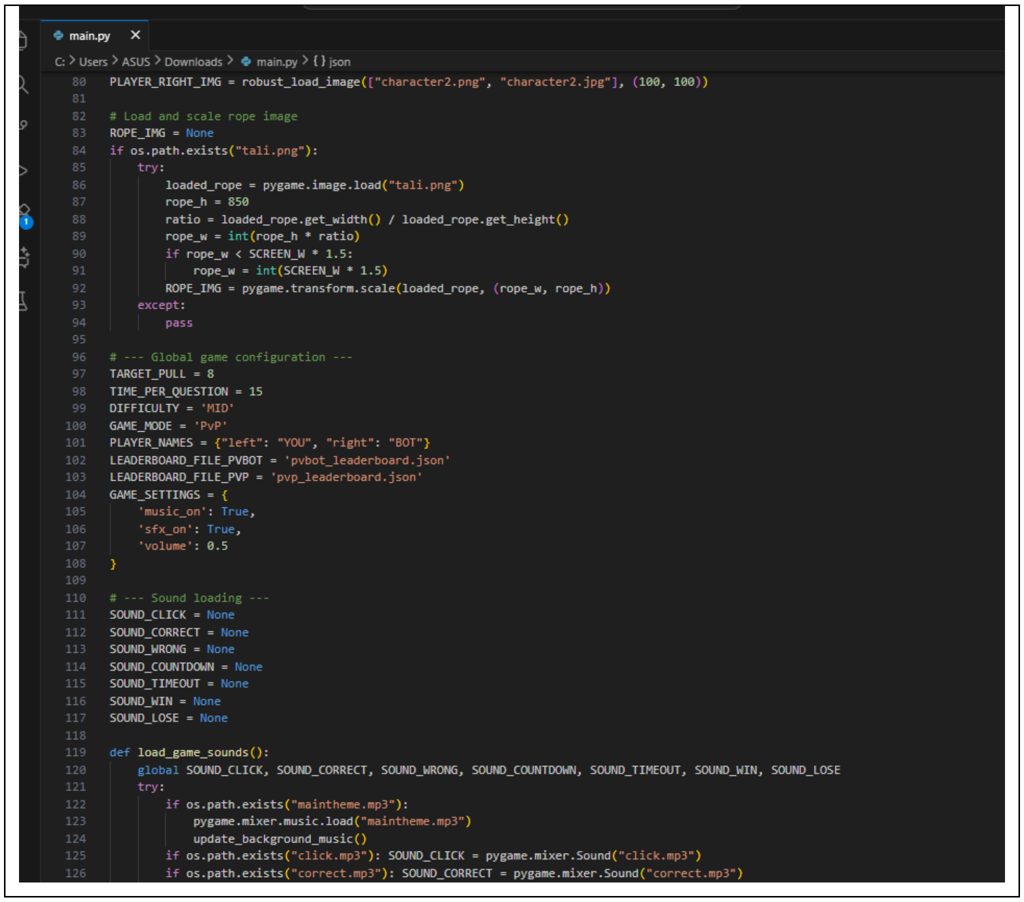
C: > Users > ASUS > Downloads > main.py > {} json

```python
def robust_load_image(filenames, scale_size=None):
    for name in filenames:
        if os.path.exists(name):
            try:
                img = pygame.image.load(name)
                if scale_size:
                    img = pygame.transform.scale(img, scale_size)
                return img
            except Exception as e:
                print(f"Failed to load {name}: {e}")
    return None

# --- Load assets ---
WALLPAPER_IMG = robust_load_image(["wallpaper.png", "wallpaper.jpg"], (SCREEN_W, SCREEN_H))
INGAME_WALLPAPER_IMG = robust_load_image(["ingamewallpaper.png", "ingamewallpaper.jpg"], (SCREEN_W, SCREEN_H))
TARGET_LINE_IMG = robust_load_image(["target.png"], (60, 80))
INDICATOR_IMG = robust_load_image(["indicator.png"], (64, 64))
```

Description: Definition of a Player class to store player data (name, score, playing time).

A simple data container class. Objects from this class are used to store the name, score, and time when the player completes the game, before the data is serialized to JSON.

F. Input Box Class



main.py ×

C: > Users > ASUS > Downloads > main.py > {} json

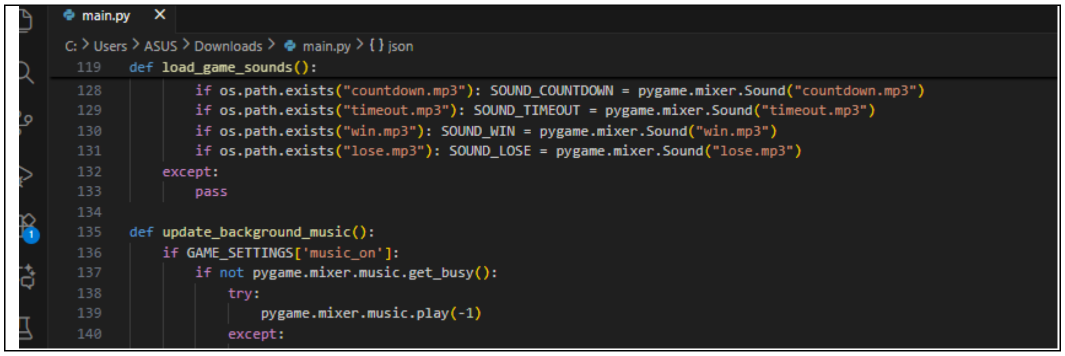
```python
PLAYER_RIGHT_IMG = robust_load_image(["character2.png", "character2.jpg"], (100, 100))

# Load and scale rope image
ROPE_IMG = None
if os.path.exists("tali.png"):
    try:
        loaded_rope = pygame.image.load("tali.png")
        rope_h = 850
        ratio = loaded_rope.get_width() / loaded_rope.get_height()
        rope_w = int(rope_h * ratio)
        if rope_w < SCREEN_W * 1.5:
            rope_w = int(SCREEN_W * 1.5)
        ROPE_IMG = pygame.transform.scale(loaded_rope, (rope_w, rope_h))
    except:
        pass

# --- Global game configuration ---
TARGET_PULL = 8
TIME_PER_QUESTION = 15
DIFFICULTY = 'MID'
GAME_MODE = 'PvP'
PLAYER_NAMES = {"left": "YOU", "right": "BOT"}
LEADERBOARD_FILE_PVBOT = 'pvbot_leaderboard.json'
LEADERBOARD_FILE_PVP = 'pvp_leaderboard.json'
GAME_SETTINGS = {
    'music_on': True,
    'sfx_on': True,
    'volume': 0.5
}

# --- Sound loading ---
SOUND_CLICK = None
SOUND_CORRECT = None
SOUND_WRONG = None
SOUND_COUNTDOWN = None
SOUND_TIMEOUT = None
SOUND_WIN = None
SOUND_LOSE = None

def load_game_sounds():
    global SOUND_CLICK, SOUND_CORRECT, SOUND_WRONG, SOUND_COUNTDOWN, SOUND_TIMEOUT, SOUND_WIN, SOUND_LOSE
    try:
        if os.path.exists("maintheme.mp3"):
            pygame.mixer.music.load("maintheme.mp3")
            update_background_music()
        if os.path.exists("click.mp3"): SOUND_CLICK = pygame.mixer.Sound("click.mp3")
        if os.path.exists("correct.mp3"): SOUND_CORRECT = pygame.mixer.Sound("correct.mp3")
```

Description: A class for creating a text input box that can be drawn and receive keyboard input, used to enter the player's name.

- `__init__`: Create a Pygame box (`pygame. Rect`) and set up the font.
- `handle_event`: Specifically listening to `pygames. MOUSEBUTTONDOWN` (to enable/disable the box) and `pygame. KEYDOWN` (to accept alphanumeric or Backspace characters). It also handles transitions to `STATE_GAME_PLAY` when **the ENTER** key is pressed.

`draw`: Draw the box and text that is inserted into the screen.

G. Class Score



main.py ✕

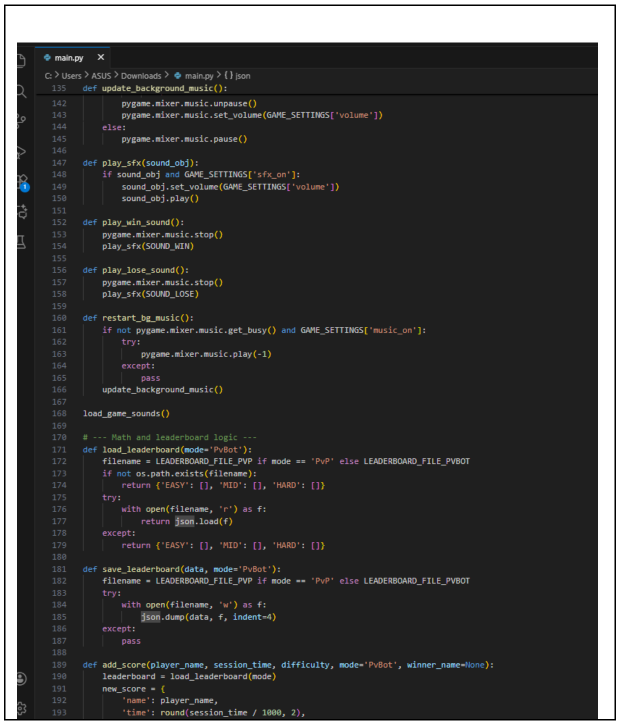
C: > Users > ASUS > Downloads > main.py > { } json

```python
def load_game_sounds():
            if os.path.exists("countdown.mp3"): SOUND_COUNTDOWN = pygame.mixer.Sound("countdown.mp3")
            if os.path.exists("timeout.mp3"): SOUND_TIMEOUT = pygame.mixer.Sound("timeout.mp3")
            if os.path.exists("win.mp3"): SOUND_WIN = pygame.mixer.Sound("win.mp3")
            if os.path.exists("lose.mp3"): SOUND_LOSE = pygame.mixer.Sound("lose.mp3")
        except:
            pass

def update_background_music():
    if GAME_SETTINGS['music_on']:
        if not pygame.mixer.music.get_busy():
            try:
                pygame.mixer.music.play(-1)
            except:
```

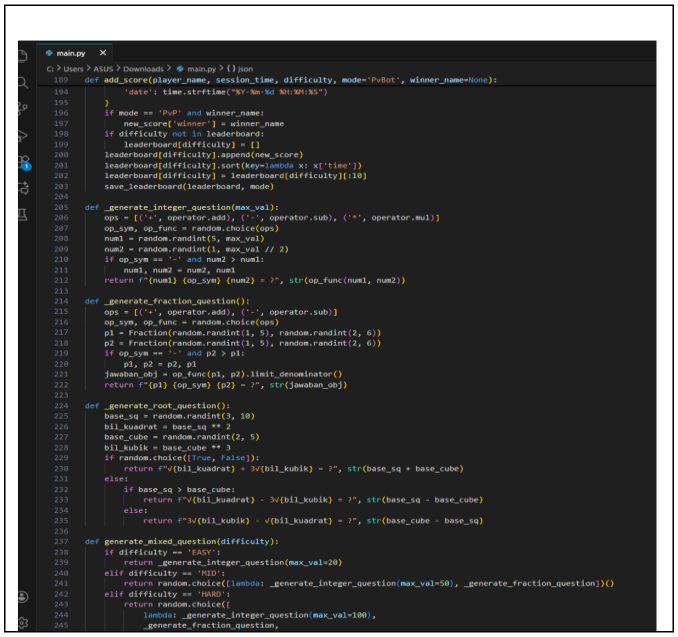
Description: A small utility class used within the Game class. Despite its name Score, in the context of math problems, it is used to store:

1. The mathematical value of the operant (e.g., Fraction (1, 2)).
2. Input string entered by the player ("1/2").
3. Correctness status of the answer (is_correct).

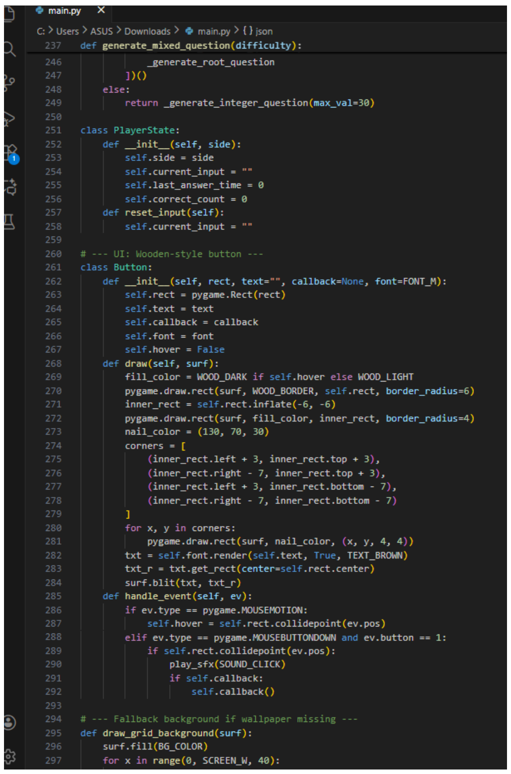
H. Class Game



```python
def update_background_music():
            pygame.mixer.music.unpause()
            pygame.mixer.music.set_volume(GAME_SETTINGS['volume'])
        else:
            pygame.mixer.music.pause()

def play_sfx(sound_obj):
    if sound_obj and GAME_SETTINGS['sfx_on']:
        sound_obj.set_volume(GAME_SETTINGS['volume'])
        sound_obj.play()

def play_win_sound():
    pygame.mixer.music.stop()
    play_sfx(SOUND_WIN)

def play_lose_sound():
    pygame.mixer.music.stop()
    play_sfx(SOUND_LOSE)

def restart_bg_music():
    if not pygame.mixer.music.get_busy() and GAME_SETTINGS['music_on']:
        try:
            pygame.mixer.music.play(-1)
        except:
            pass
    update_background_music()

load_game_sounds()

# --- Math and leaderboard logic ---
def load_leaderboard(mode='PvBot'):
    filename = LEADERBOARD_FILE_PVP if mode == 'PvP' else LEADERBOARD_FILE_PVBOT
    if not os.path.exists(filename):
        return {'EASY': [], 'MID': [], 'HARD': []}
    try:
        with open(filename, 'r') as f:
            return json.load(f)
    except:
        return {'EASY': [], 'MID': [], 'HARD': []}

def save_leaderboard(data, mode='PvBot'):
    filename = LEADERBOARD_FILE_PVP if mode == 'PvP' else LEADERBOARD_FILE_PVBOT
    try:
        with open(filename, 'w') as f:
            json.dump(data, f, indent=4)
    except:
        pass

def add_score(player_name, session_time, difficulty, mode='PvBot', winner_name=None):
    leaderboard = load_leaderboard(mode)
    new_score = {
        'name': player_name,
        'time': round(session_time / 1000, 2),
```



```python
def add_score(player_name, session_time, difficulty, mode='PvBot', winner_name=None):
        'date': time.strftime("%Y-%m-%d %H:%M:%S")
    }
    if mode == 'PvP' and winner_name:
        new_score['winner'] = winner_name
    if difficulty not in leaderboard:
        leaderboard[difficulty] = []
    leaderboard[difficulty].append(new_score)
    leaderboard[difficulty].sort(key=lambda x: x['time'])
    leaderboard[difficulty] = leaderboard[difficulty][:10]
    save_leaderboard(leaderboard, mode)

def _generate_integer_question(max_val):
    ops = [('+', operator.add), ('-', operator.sub), ('*', operator.mul)]
    op_sym, op_func = random.choice(ops)
    num1 = random.randint(5, max_val)
    num2 = random.randint(1, max_val // 2)
    if op_sym == '-' and num2 > num1:
        num1, num2 = num2, num1
    return f"{num1} {op_sym} {num2} = ?", str(op_func(num1, num2))

def _generate_fraction_question():
    ops = [('+', operator.add), ('-', operator.sub)]
    op_sym, op_func = random.choice(ops)
    p1 = Fraction(random.randint(1, 5), random.randint(2, 6))
    p2 = Fraction(random.randint(1, 5), random.randint(2, 6))
    if op_sym == '-' and p2 > p1:
        p1, p2 = p2, p1
    jawaban_obj = op_func(p1, p2).limit_denominator()
    return f"{p1} {op_sym} {p2} = ?", str(jawaban_obj)

def _generate_root_question():
    base_sq = random.randint(3, 10)
    bil_kuadrat = base_sq ** 2
    base_cube = random.randint(2, 5)
    bil_kubik = base_cube ** 3
    if random.choice([True, False]):
        return f"√{bil_kuadrat} + 3√{bil_kubik} = ?", str(base_sq + base_cube)
    else:
        if base_sq > base_cube:
            return f"√{bil_kuadrat} - 3√{bil_kubik} = ?", str(base_sq - base_cube)
        else:
            return f"3√{bil_kubik} - √{bil_kuadrat} = ?", str(base_cube - base_sq)

def generate_mixed_question(difficulty):
    if difficulty == 'EASY':
        return _generate_integer_question(max_val=20)
    elif difficulty == 'MID':
        return random.choice([lambda: _generate_integer_question(max_val=50), _generate_fraction_question])()
    elif difficulty == 'HARD':
        return random.choice([
            lambda: _generate_integer_question(max_val=100),
            _generate_fraction_question,
```



main.py

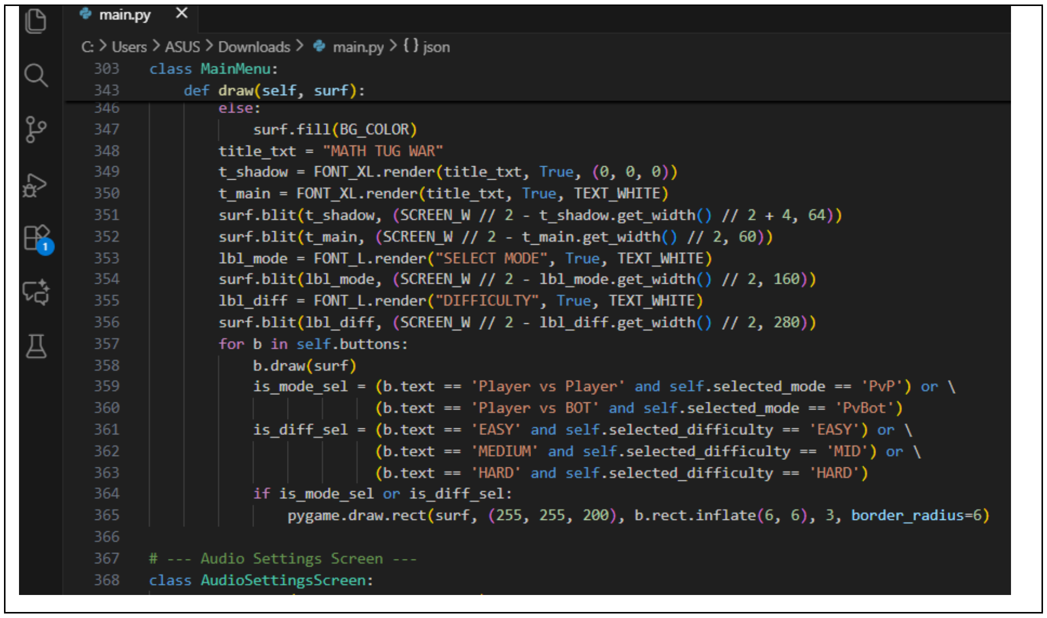
C: > Users > ASUS > Downloads > main.py > { } json

```python
def generate_mixed_question(difficulty):
                _generate_root_question
            ])()
        else:
            return _generate_integer_question(max_val=30)

class PlayerState:
    def __init__(self, side):
        self.side = side
        self.current_input = ""
        self.last_answer_time = 0
        self.correct_count = 0
    def reset_input(self):
        self.current_input = ""

# --- UI: Wooden-style button ---
class Button:
    def __init__(self, rect, text="", callback=None, font=FONT_M):
        self.rect = pygame.Rect(rect)
        self.text = text
        self.callback = callback
        self.font = font
        self.hover = False
    def draw(self, surf):
        fill_color = WOOD_DARK if self.hover else WOOD_LIGHT
        pygame.draw.rect(surf, WOOD_BORDER, self.rect, border_radius=6)
        inner_rect = self.rect.inflate(-6, -6)
        pygame.draw.rect(surf, fill_color, inner_rect, border_radius=4)
        nail_color = (130, 70, 30)
        corners = [
            (inner_rect.left + 3, inner_rect.top + 3),
            (inner_rect.right - 7, inner_rect.top + 3),
            (inner_rect.left + 3, inner_rect.bottom - 7),
            (inner_rect.right - 7, inner_rect.bottom - 7)
        ]
        for x, y in corners:
            pygame.draw.rect(surf, nail_color, (x, y, 4, 4))
        txt = self.font.render(self.text, True, TEXT_BROWN)
        txt_r = txt.get_rect(center=self.rect.center)
        surf.blit(txt, txt_r)
    def handle_event(self, ev):
        if ev.type == pygame.MOUSEMOTION:
            self.hover = self.rect.collidepoint(ev.pos)
        elif ev.type == pygame.MOUSEBUTTONDOWN and ev.button == 1:
            if self.rect.collidepoint(ev.pos):
                play_sfx(SOUND_CLICK)
                if self.callback:
                    self.callback()

# --- Fallback background if wallpaper missing ---
def draw_grid_background(surf):
    surf.fill(BG_COLOR)
    for x in range(0, SCREEN_W, 40):
```

Description: The main class that handles game logic: question creation, number input, answer check, score, timer, and game display.

- `__init__`: Sets the initial variables (score, marble_x marble position), loads sound effects, and initializes the timer.
- *`get_random_operand`*: Generate random numbers that can be integers, decimals, or fractions, ensuring the variation of the questions.
- *`generate_new_question`*: Selects a random operator, generates two passes, and calculates the correct answer *(`self.answer`). The logic* here ensures that there is no division with zero.
- *`convert_input_to_fraction`*: Handle the player's string input (e.g., `"1.5"` or `"3/2")` and convert it to *a* *`Fraction`* object so that it can be accurately compared to `the self.answer`.
- *`check_answer`*: Answer check logic. If it does, it awards points and shifts *`self.marble_x`* to the right. If it is wrong, it gives a penalty and shifts to the left.
- *`update`*: The part called each frame *(`dt`)*. This reduces the total playing time *(`self.time_left`)* and time per question *(`self.current_question_time`)*. If time runs out, it triggers *`end_game()`*.
- *`draw`*: Draw all the visual elements: background, marble rope, marble position, math problem, input box, timer bar, and score/time leftover.

I. Class Game Over Screen

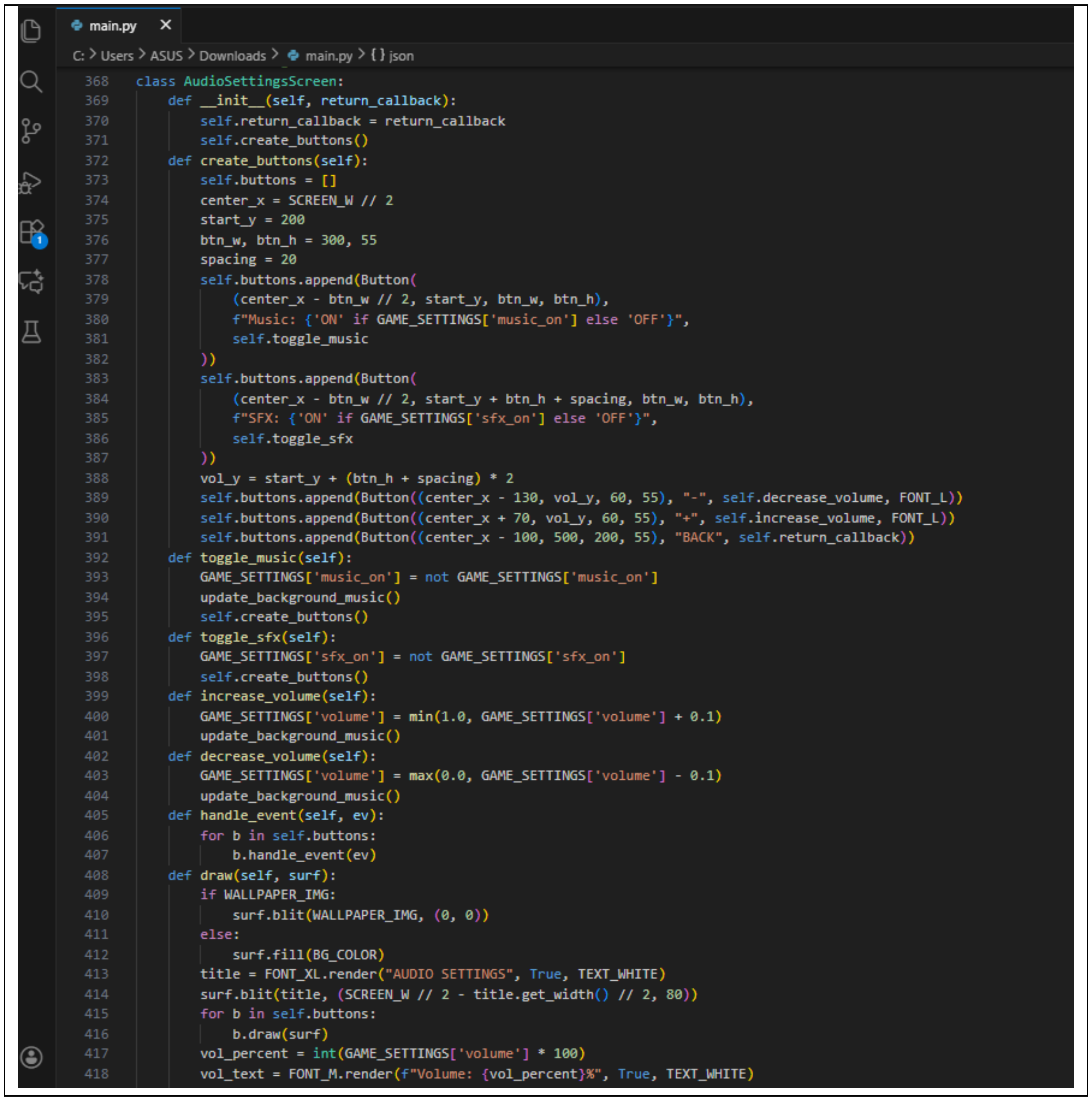


```python
class MainMenu:
    def draw(self, surf):
            else:
                surf.fill(BG_COLOR)
            title_txt = "MATH TUG WAR"
            t_shadow = FONT_XL.render(title_txt, True, (0, 0, 0))
            t_main = FONT_XL.render(title_txt, True, TEXT_WHITE)
            surf.blit(t_shadow, (SCREEN_W // 2 - t_shadow.get_width() // 2 + 4, 64))
            surf.blit(t_main, (SCREEN_W // 2 - t_main.get_width() // 2, 60))
            lbl_mode = FONT_L.render("SELECT MODE", True, TEXT_WHITE)
            surf.blit(lbl_mode, (SCREEN_W // 2 - lbl_mode.get_width() // 2, 160))
            lbl_diff = FONT_L.render("DIFFICULTY", True, TEXT_WHITE)
            surf.blit(lbl_diff, (SCREEN_W // 2 - lbl_diff.get_width() // 2, 280))
            for b in self.buttons:
                b.draw(surf)
                is_mode_sel = (b.text == 'Player vs Player' and self.selected_mode == 'PvP') or \
                              (b.text == 'Player vs BOT' and self.selected_mode == 'PvBot')
                is_diff_sel = (b.text == 'EASY' and self.selected_difficulty == 'EASY') or \
                              (b.text == 'MEDIUM' and self.selected_difficulty == 'MID') or \
                              (b.text == 'HARD' and self.selected_difficulty == 'HARD')
                if is_mode_sel or is_diff_sel:
                    pygame.draw.rect(surf, (255, 255, 200), b.rect.inflate(6, 6), 3, border_radius=6)

# --- Audio Settings Screen ---
class AudioSettingsScreen:
```

Description: A class to display the Game Over screen, including the final score and a button to return to the menu.

- Displays the player's final score.
- Provides a button to return to the Main Menu (`handle_event` handles button clicks).

J. Class Leaderboard Screen

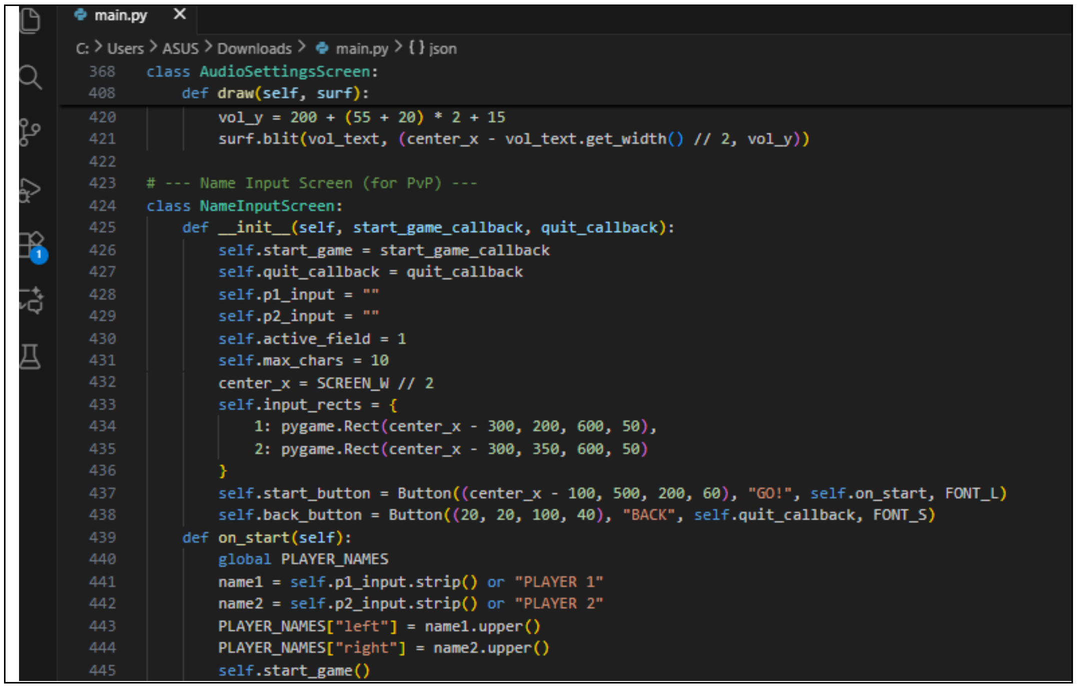


```python
class AudioSettingsScreen:
    def __init__(self, return_callback):
        self.return_callback = return_callback
        self.create_buttons()
    def create_buttons(self):
        self.buttons = []
        center_x = SCREEN_W // 2
        start_y = 200
        btn_w, btn_h = 300, 55
        spacing = 20
        self.buttons.append(Button(
            (center_x - btn_w // 2, start_y, btn_w, btn_h),
            f"Music: {'ON' if GAME_SETTINGS['music_on'] else 'OFF'}",
            self.toggle_music
        ))
        self.buttons.append(Button(
            (center_x - btn_w // 2, start_y + btn_h + spacing, btn_w, btn_h),
            f"SFX: {'ON' if GAME_SETTINGS['sfx_on'] else 'OFF'}",
            self.toggle_sfx
        ))
        vol_y = start_y + (btn_h + spacing) * 2
        self.buttons.append(Button((center_x - 130, vol_y, 60, 55), "-", self.decrease_volume, FONT_L))
        self.buttons.append(Button((center_x + 70, vol_y, 60, 55), "+", self.increase_volume, FONT_L))
        self.buttons.append(Button((center_x - 100, 500, 200, 55), "BACK", self.return_callback))
    def toggle_music(self):
        GAME_SETTINGS['music_on'] = not GAME_SETTINGS['music_on']
        update_background_music()
        self.create_buttons()
    def toggle_sfx(self):
        GAME_SETTINGS['sfx_on'] = not GAME_SETTINGS['sfx_on']
        self.create_buttons()
    def increase_volume(self):
        GAME_SETTINGS['volume'] = min(1.0, GAME_SETTINGS['volume'] + 0.1)
        update_background_music()
    def decrease_volume(self):
        GAME_SETTINGS['volume'] = max(0.0, GAME_SETTINGS['volume'] - 0.1)
        update_background_music()
    def handle_event(self, ev):
        for b in self.buttons:
            b.handle_event(ev)
    def draw(self, surf):
        if WALLPAPER_IMG:
            surf.blit(WALLPAPER_IMG, (0, 0))
        else:
            surf.fill(BG_COLOR)
        title = FONT_XL.render("AUDIO SETTINGS", True, TEXT_WHITE)
        surf.blit(title, (SCREEN_W // 2 - title.get_width() // 2, 80))
        for b in self.buttons:
            b.draw(surf)
        vol_percent = int(GAME_SETTINGS['volume'] * 100)
        vol_text = FONT_M.render(f"Volume: {vol_percent}%", True, TEXT_WHITE)
```

Description: A class to display a list of the highest scores (leaderboard) loaded from JSON data.

- `update_data`: Reloads data from JSON files before they are displayed, ensuring the latest score is available.
- `draw`: Draw the title, column header (Ranking, Name, Score), and data of the top 10 players from the loaded file.

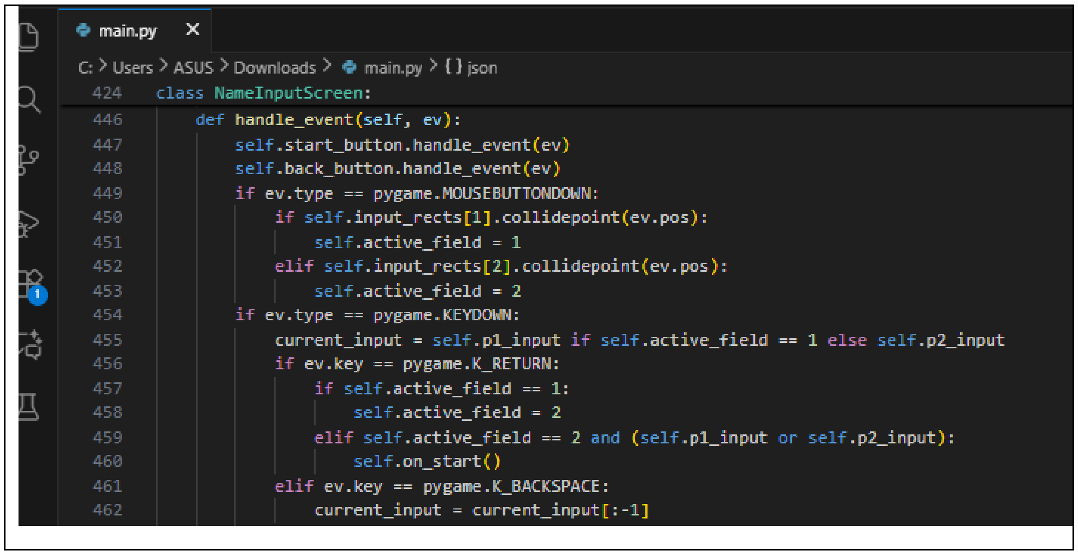
K. Class Main Menu



```python
class AudioSettingsScreen:
    def draw(self, surf):
        vol_y = 200 + (55 + 20) * 2 + 15
        surf.blit(vol_text, (center_x - vol_text.get_width() // 2, vol_y))

# --- Name Input Screen (for PvP) ---
class NameInputScreen:
    def __init__(self, start_game_callback, quit_callback):
        self.start_game = start_game_callback
        self.quit_callback = quit_callback
        self.p1_input = ""
        self.p2_input = ""
        self.active_field = 1
        self.max_chars = 10
        center_x = SCREEN_W // 2
        self.input_rects = {
            1: pygame.Rect(center_x - 300, 200, 600, 50),
            2: pygame.Rect(center_x - 300, 350, 600, 50)
        }
        self.start_button = Button((center_x - 100, 500, 200, 60), "GO!", self.on_start, FONT_L)
        self.back_button = Button((20, 20, 100, 40), "BACK", self.quit_callback, FONT_S)
    def on_start(self):
        global PLAYER_NAMES
        name1 = self.p1_input.strip() or "PLAYER 1"
        name2 = self.p2_input.strip() or "PLAYER 2"
        PLAYER_NAMES["left"] = name1.upper()
        PLAYER_NAMES["right"] = name2.upper()
        self.start_game()
```

Description: Classes for creating and drawing the main menu screen (Start, Leaderboard, Exit buttons). handle_event: Listen for mouse clicks on the Start, Leaderboard, or Exit buttons, and change the global current_state as you choose.

L. Initial Setup & Object Initialization

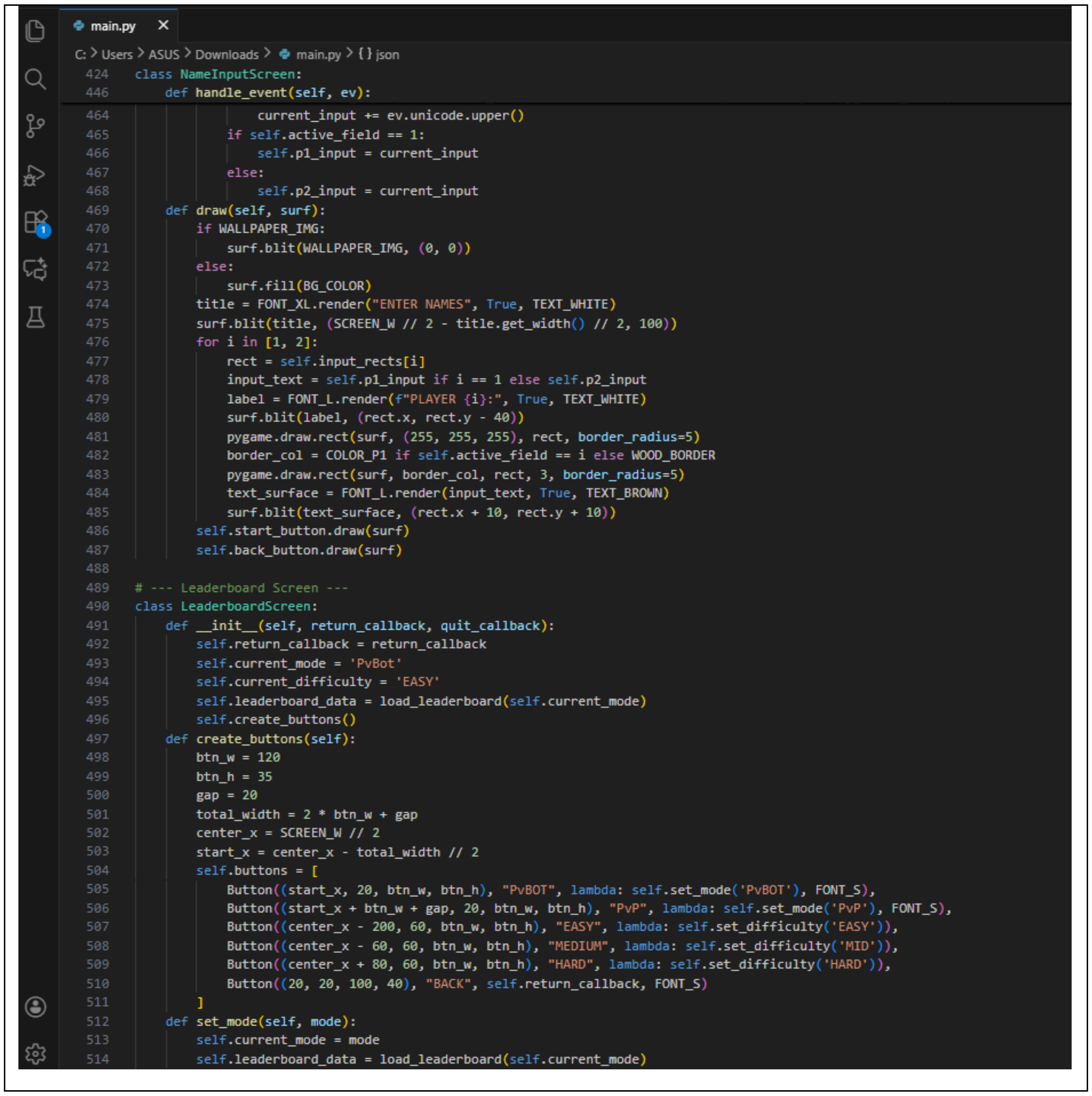


```python
class NameInputScreen:
    def handle_event(self, ev):
        self.start_button.handle_event(ev)
        self.back_button.handle_event(ev)
        if ev.type == pygame.MOUSEBUTTONDOWN:
            if self.input_rects[1].collidepoint(ev.pos):
                self.active_field = 1
            elif self.input_rects[2].collidepoint(ev.pos):
                self.active_field = 2
        if ev.type == pygame.KEYDOWN:
            current_input = self.p1_input if self.active_field == 1 else self.p2_input
            if ev.key == pygame.K_RETURN:
                if self.active_field == 1:
                    self.active_field = 2
                elif self.active_field == 2 and (self.p1_input or self.p2_input):
                    self.on_start()
            elif ev.key == pygame.K_BACKSPACE:
                current_input = current_input[:-1]
```

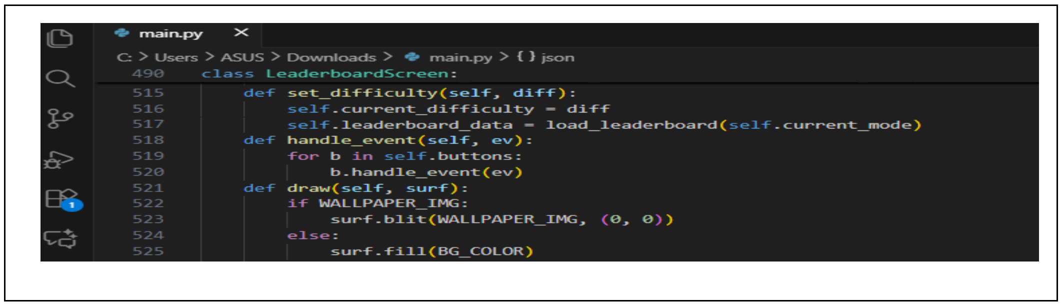
Description: Once all classes and constants are defined, this section is run once before *the game loop*.

- Create a Pygame window (`screen`) and a `clock` object.
- Load the initial `leaderboard_data` data.
- Initializes all the main objects *(`main_menu`, `name_input_screen`, `leaderboard_screen`)*.
- Specifies the initial state of the program *(`current_state = STATE_MAIN_MENU`)*.

M. Main Functions run_game()

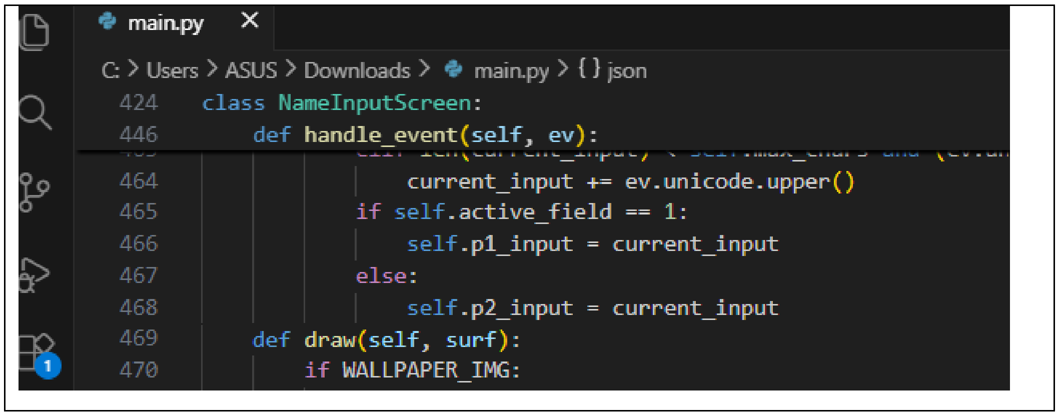


```python
class NameInputScreen:
    def handle_event(self, ev):
                    current_input += ev.unicode.upper()
                if self.active_field == 1:
                    self.p1_input = current_input
                else:
                    self.p2_input = current_input
    def draw(self, surf):
        if WALLPAPER_IMG:
            surf.blit(WALLPAPER_IMG, (0, 0))
        else:
            surf.fill(BG_COLOR)
        title = FONT_XL.render("ENTER NAMES", True, TEXT_WHITE)
        surf.blit(title, (SCREEN_W // 2 - title.get_width() // 2, 100))
        for i in [1, 2]:
            rect = self.input_rects[i]
            input_text = self.p1_input if i == 1 else self.p2_input
            label = FONT_L.render(f"PLAYER {i}:", True, TEXT_WHITE)
            surf.blit(label, (rect.x, rect.y - 40))
            pygame.draw.rect(surf, (255, 255, 255), rect, border_radius=5)
            border_col = COLOR_P1 if self.active_field == i else WOOD_BORDER
            pygame.draw.rect(surf, border_col, rect, 3, border_radius=5)
            text_surface = FONT_L.render(input_text, True, TEXT_BROWN)
            surf.blit(text_surface, (rect.x + 10, rect.y + 10))
        self.start_button.draw(surf)
        self.back_button.draw(surf)

# --- Leaderboard Screen ---
class LeaderboardScreen:
    def __init__(self, return_callback, quit_callback):
        self.return_callback = return_callback
        self.current_mode = 'PvBot'
        self.current_difficulty = 'EASY'
        self.leaderboard_data = load_leaderboard(self.current_mode)
        self.create_buttons()
    def create_buttons(self):
        btn_w = 120
        btn_h = 35
        gap = 20
        total_width = 2 * btn_w + gap
        center_x = SCREEN_W // 2
        start_x = center_x - total_width // 2
        self.buttons = [
            Button((start_x, 20, btn_w, btn_h), "PvBOT", lambda: self.set_mode('PvBOT'), FONT_S),
            Button((start_x + btn_w + gap, 20, btn_w, btn_h), "PvP", lambda: self.set_mode('PvP'), FONT_S),
            Button((center_x - 200, 60, btn_w, btn_h), "EASY", lambda: self.set_difficulty('EASY')),
            Button((center_x - 60, 60, btn_w, btn_h), "MEDIUM", lambda: self.set_difficulty('MID')),
            Button((center_x + 80, 60, btn_w, btn_h), "HARD", lambda: self.set_difficulty('HARD')),
            Button((20, 20, 100, 40), "BACK", self.return_callback, FONT_S)
        ]
    def set_mode(self, mode):
        self.current_mode = mode
        self.leaderboard_data = load_leaderboard(self.current_mode)
```



```
C: > Users > ASUS > Downloads > main.py > { } json
class LeaderboardScreen:
    def set_difficulty(self, diff):
        self.current_difficulty = diff
        self.leaderboard_data = load_leaderboard(self.current_mode)
    def handle_event(self, ev):
        for b in self.buttons:
            b.handle_event(ev)
    def draw(self, surf):
        if WALLPAPER_IMG:
            surf.blit(WALLPAPER_IMG, (0, 0))
        else:
            surf.fill(BG_COLOR)
```

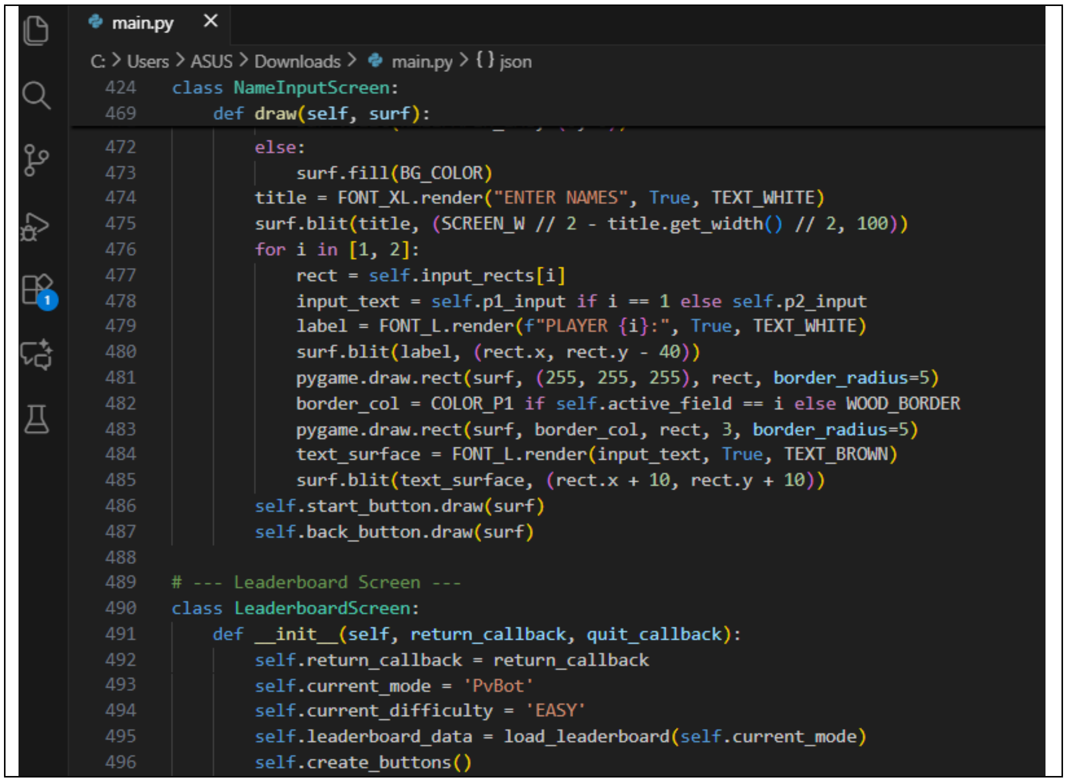
M1. Initialization of Loop & Delta Time



```
C: > Users > ASUS > Downloads > main.py > { } json
class NameInputScreen:
    def handle_event(self, ev):
                current_input += ev.unicode.upper()
            if self.active_field == 1:
                self.p1_input = current_input
            else:
                self.p2_input = current_input
    def draw(self, surf):
        if WALLPAPER_IMG:
```

Description: dt = clock.tick(FPS) / 1000.0: Calculates the time (in seconds) that elapsed between the current frame and the previous frame (delta time). This dt value is essential for making the movement and timer run consistently, regardless of CPU speed.

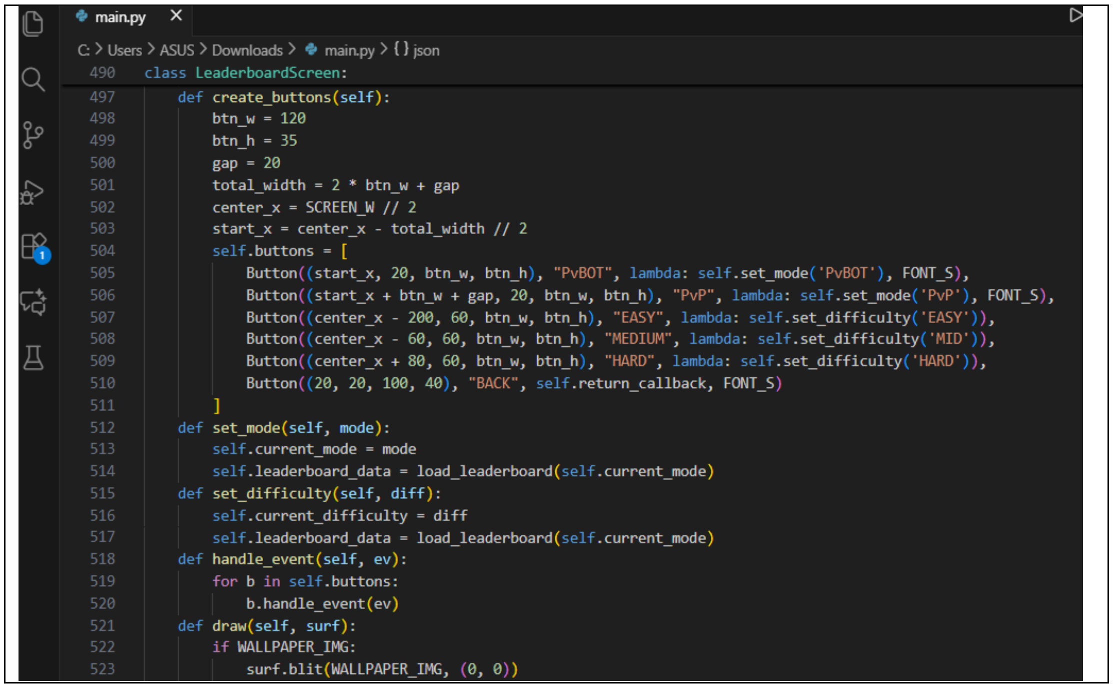
M2. Event Handling (Input)



```python
class NameInputScreen:
    def draw(self, surf):
        else:
            surf.fill(BG_COLOR)
        title = FONT_XL.render("ENTER NAMES", True, TEXT_WHITE)
        surf.blit(title, (SCREEN_W // 2 - title.get_width() // 2, 100))
        for i in [1, 2]:
            rect = self.input_rects[i]
            input_text = self.p1_input if i == 1 else self.p2_input
            label = FONT_L.render(f"PLAYER {i}:", True, TEXT_WHITE)
            surf.blit(label, (rect.x, rect.y - 40))
            pygame.draw.rect(surf, (255, 255, 255), rect, border_radius=5)
            border_col = COLOR_P1 if self.active_field == i else WOOD_BORDER
            pygame.draw.rect(surf, border_col, rect, 3, border_radius=5)
            text_surface = FONT_L.render(input_text, True, TEXT_BROWN)
            surf.blit(text_surface, (rect.x + 10, rect.y + 10))
        self.start_button.draw(surf)
        self.back_button.draw(surf)

# --- Leaderboard Screen ---
class LeaderboardScreen:
    def __init__(self, return_callback, quit_callback):
        self.return_callback = return_callback
        self.current_mode = 'PvBot'
        self.current_difficulty = 'EASY'
        self.leaderboard_data = load_leaderboard(self.current_mode)
        self.create_buttons()
```

- Loop for ev in pygame.event.get(): Continuously retrieves all user inputs (button presses, mouse clicks) and system events (e.g., closing windows).
- If structure current_state == X:: Calls only the handle_event function of the object that is relevant to *the current state*. This ensures that, for example, keyboard input is only processed by the Game when STATE_GAME_PLAY is active.

M3. Logic Update & Images



main.py

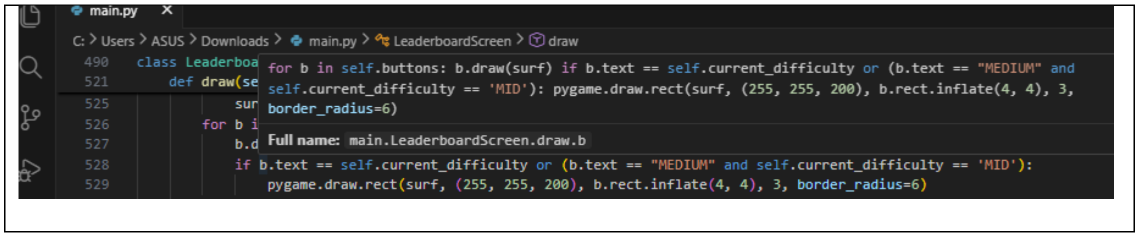
C: > Users > ASUS > Downloads > main.py > { } json

```
class LeaderboardScreen:
    def create_buttons(self):
        btn_w = 120
        btn_h = 35
        gap = 20
        total_width = 2 * btn_w + gap
        center_x = SCREEN_W // 2
        start_x = center_x - total_width // 2
        self.buttons = [
            Button((start_x, 20, btn_w, btn_h), "PvBOT", lambda: self.set_mode('PvBOT'), FONT_S),
            Button((start_x + btn_w + gap, 20, btn_w, btn_h), "PvP", lambda: self.set_mode('PvP'), FONT_S),
            Button((center_x - 200, 60, btn_w, btn_h), "EASY", lambda: self.set_difficulty('EASY')),
            Button((center_x - 60, 60, btn_w, btn_h), "MEDIUM", lambda: self.set_difficulty('MID')),
            Button((center_x + 80, 60, btn_w, btn_h), "HARD", lambda: self.set_difficulty('HARD')),
            Button((20, 20, 100, 40), "BACK", self.return_callback, FONT_S)
        ]
    def set_mode(self, mode):
        self.current_mode = mode
        self.leaderboard_data = load_leaderboard(self.current_mode)
    def set_difficulty(self, diff):
        self.current_difficulty = diff
        self.leaderboard_data = load_leaderboard(self.current_mode)
    def handle_event(self, ev):
        for b in self.buttons:
            b.handle_event(ev)
    def draw(self, surf):
        if WALLPAPER_IMG:
            surf.blit(WALLPAPER_IMG, (0, 0))
```

Description: The structure if current_state == X:: Similar to M2, it only calls *the update* and *draw* logic of the relevant object.

- object.update(dt): Updates all internal logic (timer, marble position, etc.).
- object.draw(screen): Redraws all the visual elements on the screen (background, UI object, question).

M4. Screen Updates & Terminal



main.py

C: > Users > ASUS > Downloads > main.py > LeaderboardScreen > draw

```
class Leaderboa
    def draw(se
            sur
        for b i
            b.d
            if b.text == self.current_difficulty or (b.text == "MEDIUM" and self.current_difficulty == 'MID'):
                pygame.draw.rect(surf, (255, 255, 200), b.rect.inflate(4, 4), 3, border_radius=6)
```

```
for b in self.buttons: b.draw(surf) if b.text == self.current_difficulty or (b.text == "MEDIUM" and
self.current_difficulty == 'MID'): pygame.draw.rect(surf, (255, 255, 200), b.rect.inflate(4, 4), 3,
border_radius=6)

Full name: main.LeaderboardScreen.draw.b
```

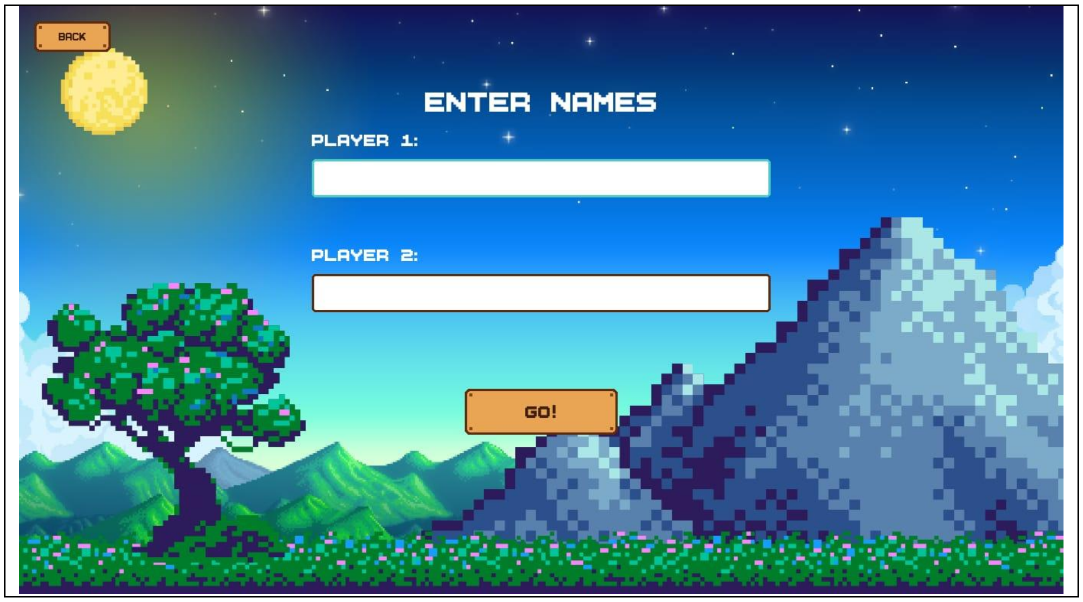
Description: pygame.display.flip(): Required to be called at the end of each loop. It displays a hidden image buffer (which was just drawn by the draw () function to the main screen, so the user can see the changes.

- **if __name__=="__main__": run_game(): Ensures that the run_game() function is only executed when the script is executed directly, and not when it is imported as a module.**

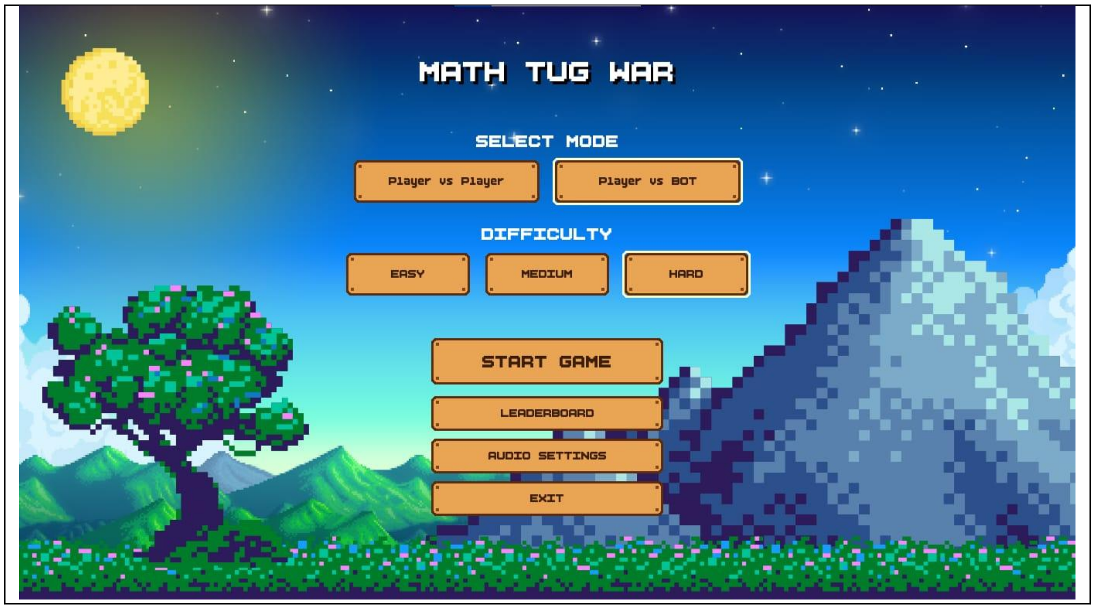
3.2 APPLICATION SCREENSHOTS

Some aspects of *the Math Tug War* game require visualization to help the reader understand the gameplay more clearly. Therefore, the following are screenshots of each display in *the Math Tug War* game to support visual understanding and interpretation.

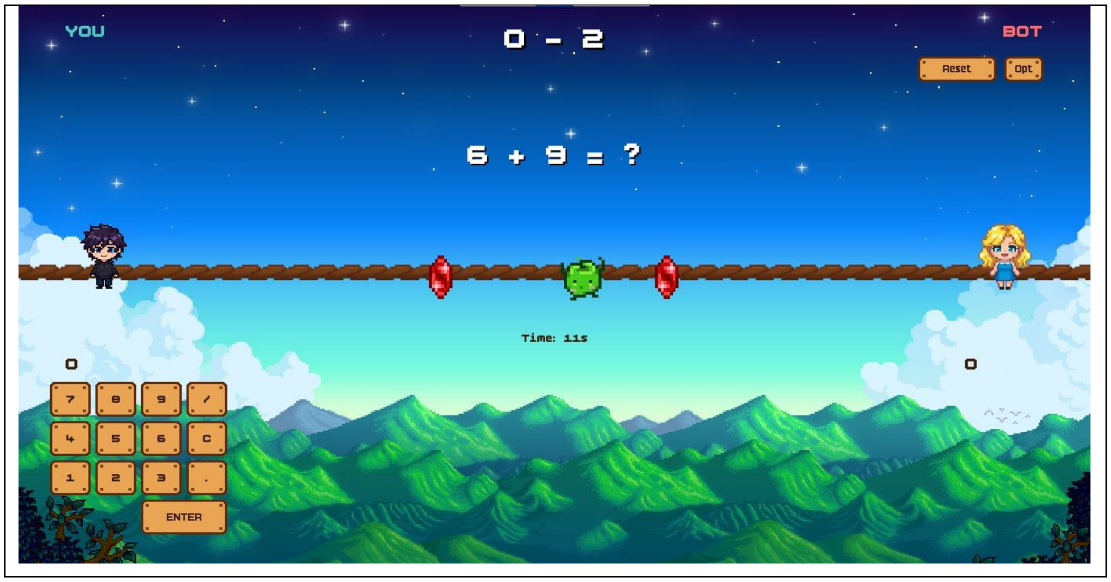
3.1.1 Main Screen



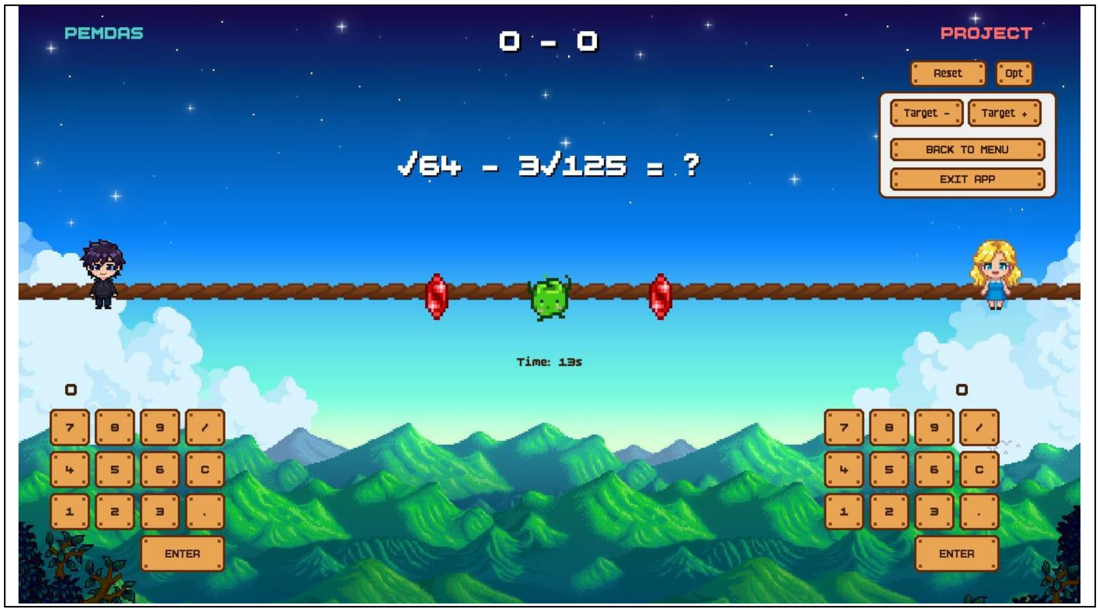
3.1.2 Game Levels



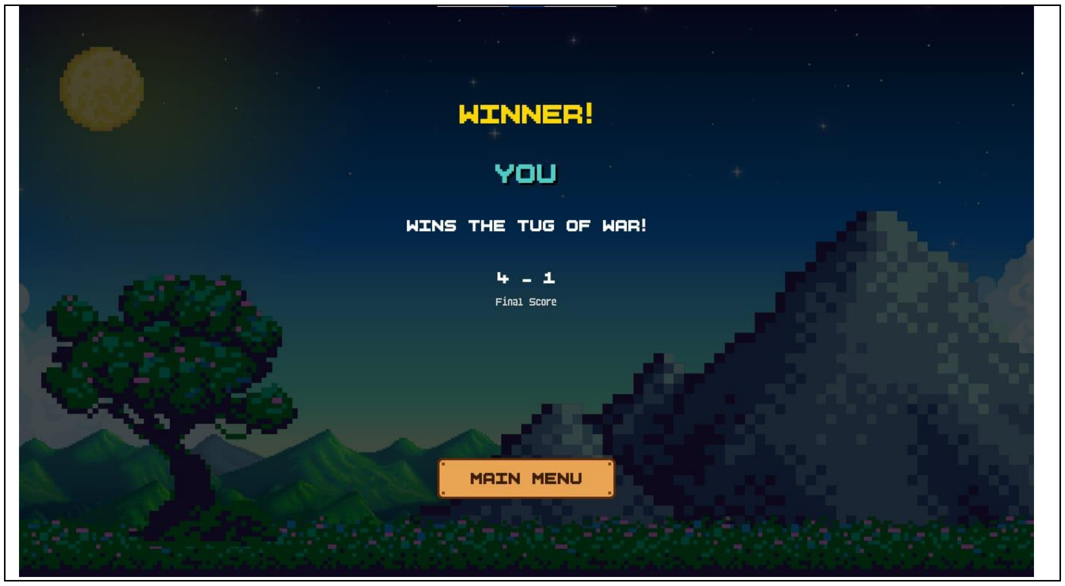
3.1.3 Easy game



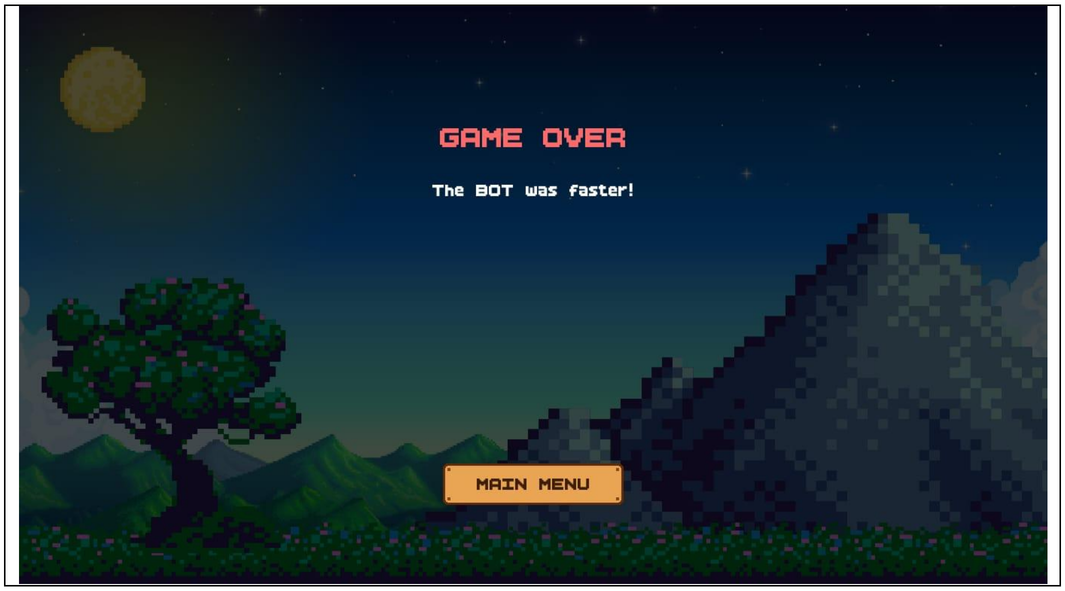
3.1.4 Hard Game



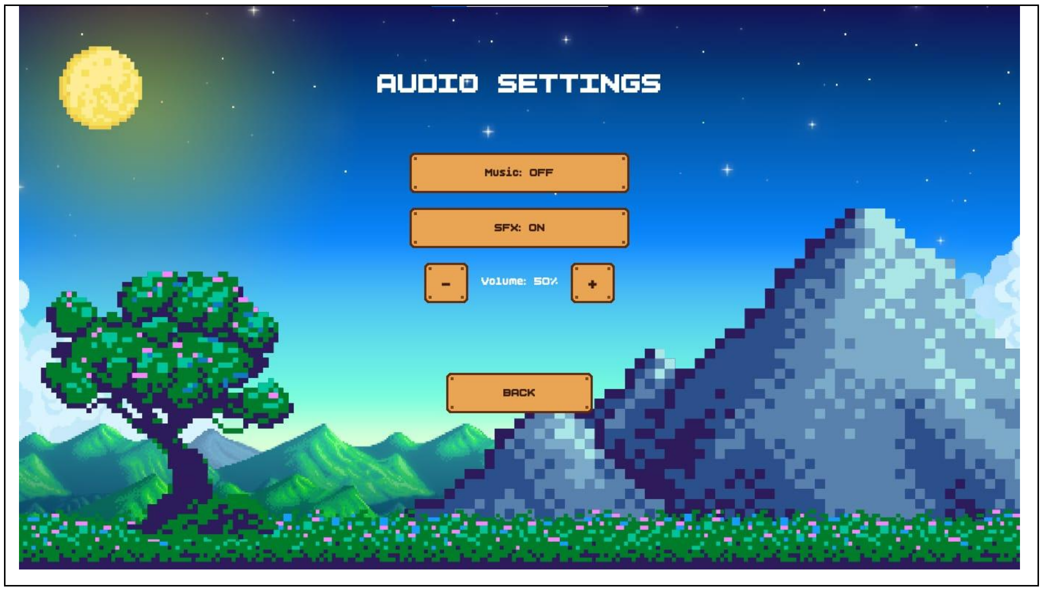
3.1.5 Win Screen



3.1.6 Lose Screen



3.1.7 Audio settings

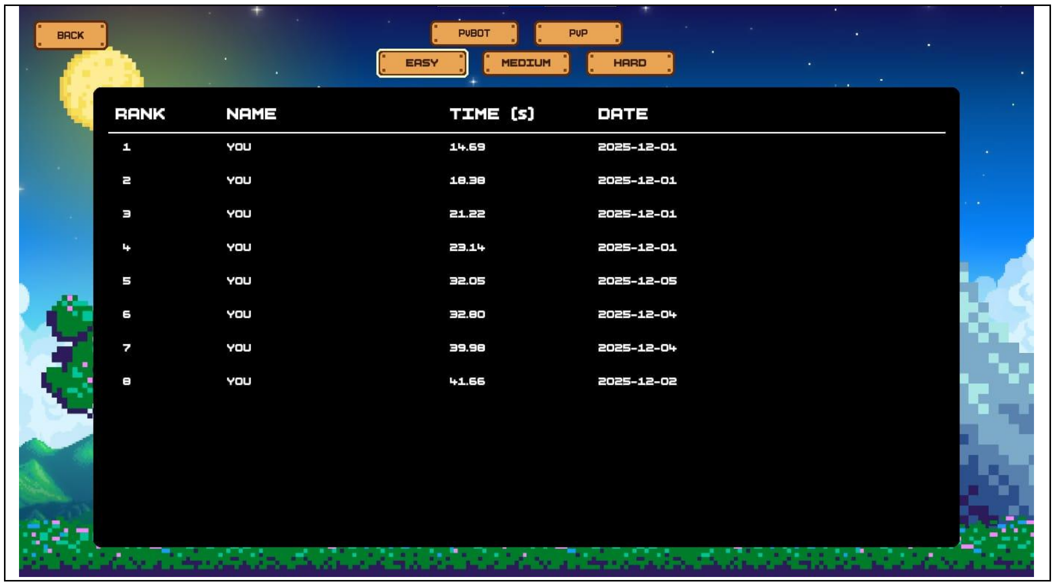


Music: OFF

SFX: ON

Volume: 50%

3.1.8 List of players



RANK	NAME	TIME (s)	DATE
1	YOU	14.69	2025-12-01
2	YOU	18.38	2025-12-01
3	YOU	21.22	2025-12-01
4	YOU	23.14	2025-12-01
5	YOU	32.05	2025-12-05
6	YOU	32.80	2025-12-04
7	YOU	39.98	2025-12-04
8	YOU	41.66	2025-12-02

CHAPTER 4
ATTACHMENT

4.1 Embed File

Embed below is the complete folder of assets and the full code of *Pull Mine.* *https://drive.google.com/drive/folders/1kbVfVhMCe5OGFtbqgIh4-O0QVzE-Gzqn*

*RE*FERENCES

Pygame Development Team. (2025). *Pygame Documentation* (Versi terbaru). Diakses dari https://www.pygame.org/docs/.

(Relevansi: Dokumentasi resmi library Pygame yang digunakan sebagai kerangka kerja (framework) utama untuk grafis, input, dan loop permainan.)

Python Software Foundation. (2025). *Python Documentation* (Versi 3.x). Diakses dari https://docs.python.org/.

(Relevansi: Dokumentasi bahasa pemrograman Python yang digunakan sebagai fondasi keseluruhan kode program.)

Python Software Foundation. (2025). fractions — Rational numbers. In *Python Documentation* (Versi 3.x). Diakses dari https://docs.python.org/3/library/fractions.html.

(Relevansi: Penggunaan modul fractions untuk memastikan perhitungan matematika (pecahan dan desimal) dilakukan dengan presisi absolut, menghilangkan eror floating point standar.)

Gamma, E., Helm, R., Johnson, R., & Vlissides, J. (1995). *Design Patterns: Elements of Reusable Object-Oriented Software*. Addison-Wesley.

*(Relevansi: Prinsip Pemrograman Berorientasi Objek (OOP) yang diterapkan melalui penggunaan kelas (Game, InputBox, MainMenu, dll.) dan pola **Finite State Machine (FSM)** untuk mengelola status game.)*

McConnell, S. (2025). *Code Complete: A Practical Handbook of Software Construction* (3rd ed. - Edisi terbaru). Microsoft Press.

(Relevansi: Panduan praktik terbaik dalam penulisan kode yang bersih, mudah dibaca, dan modular, diterapkan pada struktur file main