# Code Documentation: MathAdventure

UMEDCTA Repository

December 3, 2025

## Contents

# 1 MathAdventure/README.md

```
# The Hermeneutic Hero: Legends of Logic

## Overview
This is a retro-style RPG adventure game designed to teach mathematical strategies through gameplay. You play as a Teacher/Hero who must defeat "Number Monsters" using cognitive strategies like "Rearranging to Make Bases" (RMB) and "Sliding".

## Requirements
- Python 3.x
- Pygame (`pip install pygame`)
- Pandas (`pip install pandas`)

## How to Play
1. Navigate to the `MathAdventure` directory.
2. Run the game:
    ```bash
    python3 math_adventure.py
    ```
3. **Controls:**
    - **Arrow Keys:** Move around the world.
    - **Enter:** Start Game / Confirm.
    - **1, 2, 3:** Select Spells in Battle.
    - **Keyboard:** Type answers to math problems.

## The Spells (Strategies)
- **Count:** Basic counting. Good for small numbers.
- **RMB (Rearranging to Make Bases):** Use this when adding numbers. You must find the "Gap" to the next base (e.g., 10, 20, 30).
- **Sliding:** Use this for subtraction. Adjust both numbers to make the subtrahend a friendly base number.

## Credits
Based on the UMEDCTA project.

```

## 2 MathAdventure/math_adventure.py

```python
import pygame
import sys
import os
import random
import importlib.util
import contextlib
import io

# --- CONFIGURATION ---
SCREEN_WIDTH = 800
SCREEN_HEIGHT = 600
TILE_SIZE = 40
FPS = 60

# COLORS
WHITE = (255, 255, 255)
BLACK = (0, 0, 0)
GREEN = (50, 200, 50) # Hero
RED = (200, 50, 50)   # Enemy
BLUE = (50, 50, 200)  # UI
GRAY = (100, 100, 100)
GOLD = (255, 215, 0)
PURPLE = (128, 0, 128)

# FONTS
pygame.font.init()
FONT_MAIN = pygame.font.SysFont('Arial', 24)
FONT_TITLE = pygame.font.SysFont('Arial', 48, bold=True)
FONT_SMALL = pygame.font.SysFont('Courier New', 18)

# Context manager to suppress stdout
@contextlib.contextmanager
def suppress_stdout():
    s = io.StringIO()
    old_stdout = sys.stdout
    sys.stdout = s
    try:
        yield
    finally:
        sys.stdout = old_stdout

# --- IMPORT STRATEGIES ---
# We need to dynamically import from the sibling directory
current_dir = os.path.dirname(os.path.abspath(__file__))
project_root = os.path.dirname(current_dir)
strategies_path = os.path.join(project_root, 'Calculator', 'Python_Tests')
sys.path.append(strategies_path)

try:
    with suppress_stdout():
        import SAR_ADD_RMB
        import SAR_SUB_Sliding
        import counting_on_back
    MODULES_LOADED = True
except ImportError as e:
    print(f"Warning: Math modules not found: {e}")
    MODULES_LOADED = False

# --- GAME CLASSES ---

class GameState:
    OVERWORLD = 1
```

```
    BATTLE = 2
    MENU = 3
    GAME_OVER = 4

class Player:
    def __init__(self, x, y):
        self.grid_x = x
        self.grid_y = y
        self.hp = 100
        self.max_hp = 100
        self.xp = 0
        self.level = 1
        self.spells = ["Count"] # Starting spell
        if MODULES_LOADED:
            self.spells.append("RMB") # Unlock for testing
            self.spells.append("Sliding") # Unlock for testing

    def move(self, dx, dy, world):
        new_x = self.grid_x + dx
        new_y = self.grid_y + dy
        if world.is_walkable(new_x, new_y):
            self.grid_x = new_x
            self.grid_y = new_y
            return True
        return False

    def draw(self, screen, offset_x, offset_y):
        rect = pygame.Rect(
            self.grid_x * TILE_SIZE + offset_x,
            self.grid_y * TILE_SIZE + offset_y,
            TILE_SIZE, TILE_SIZE
        )
        pygame.draw.rect(screen, GREEN, rect)
        # Draw eyes to make it look like a character
        pygame.draw.circle(screen, BLACK, (rect.x + 12, rect.y + 12), 4)
        pygame.draw.circle(screen, BLACK, (rect.x + 28, rect.y + 12), 4)

class Enemy:
    def __init__(self, x, y, difficulty):
        self.grid_x = x
        self.grid_y = y
        self.difficulty = difficulty
        self.value = random.randint(10, 99) # The number representing the monster
        self.defeated = False

    def draw(self, screen, offset_x, offset_y):
        if self.defeated: return
        rect = pygame.Rect(
            self.grid_x * TILE_SIZE + offset_x,
            self.grid_y * TILE_SIZE + offset_y,
            TILE_SIZE, TILE_SIZE
        )
        pygame.draw.rect(screen, RED, rect)
        text = FONT_SMALL.render(str(self.value), True, WHITE)
        screen.blit(text, (rect.x + 5, rect.y + 10))

class World:
    def __init__(self, width, height):
        self.width = width
        self.height = height
        self.tiles = {} # (x,y): type
        self.enemies = []
        self.generate()

    def generate(self):
```

```
        # Simple room generation
        for x in range(self.width):
            for y in range(self.height):
                if x == 0 or x == self.width - 1 or y == 0 or y == self.height - 1:
                    self.tiles[(x,y)] = 'wall'
                else:
                    self.tiles[(x,y)] = 'floor'
                    # Random obstacles
                    if random.random() < 0.1:
                        self.tiles[(x,y)] = 'rock'

        # Spawn enemies
        for _ in range(5):
            ex, ey = random.randint(2, self.width-2), random.randint(2, self.height-2)
            if self.tiles[(ex,ey)] == 'floor':
                self.enemies.append(Enemy(ex, ey, 1))

    def is_walkable(self, x, y):
        return self.tiles.get((x,y)) == 'floor'

    def get_enemy_at(self, x, y):
        for e in self.enemies:
            if e.grid_x == x and e.grid_y == y and not e.defeated:
                return e
        return None

    def draw(self, screen):
        # Simple camera centering could go here, but we'll just draw fixed for now
        offset_x = 50
        offset_y = 50

        for (x,y), tile_type in self.tiles.items():
            rect = pygame.Rect(x*TILE_SIZE + offset_x, y*TILE_SIZE + offset_y, TILE_SIZE, TILE_SIZE)
            if tile_type == 'wall':
                pygame.draw.rect(screen, GRAY, rect)
            elif tile_type == 'rock':
                pygame.draw.rect(screen, (80,80,80), rect)
            else:
                pygame.draw.rect(screen, (200,200,200), rect, 1) # Grid lines for floor

        for e in self.enemies:
            e.draw(screen, offset_x, offset_y)

        return offset_x, offset_y

class BattleSystem:
    def __init__(self, player, enemy):
        self.player = player
        self.enemy = enemy
        self.turn = "PLAYER"
        self.log = ["A Wild Number appeared!", f"It is a {enemy.value}!"]
        self.current_problem = None
        self.state = "SELECT_SPELL" # SELECT_SPELL, SOLVING, ENEMY_TURN, WIN, LOSE
        self.input_buffer = ""

    def update(self, events):
        if self.state == "SELECT_SPELL":
            pass # Waiting for key press mapped in main loop

        elif self.state == "SOLVING":
            for event in events:
                if event.type == pygame.KEYDOWN:
                    if event.key == pygame.K_RETURN:
                        self.check_answer()
                    elif event.key == pygame.K_BACKSPACE:
```

```
                    self.input_buffer = self.input_buffer[:-1]
                else:
                    if event.unicode.isnumeric() or event.unicode in ['-', '.']:
                        self.input_buffer += event.unicode

    def cast_spell(self, spell_name):
        self.current_spell = spell_name
        self.input_buffer = ""

        if spell_name == "RMB":
            # Generate RMB problem
            # We want to add something to the enemy value to make a base
            self.addend = random.randint(1, 9)
            self.target_base = ((self.enemy.value // 10) + 1) * 10
            self.gap = self.target_base - self.enemy.value

            self.log.append(f"Casting RMB on {self.enemy.value} + {self.addend}...")
            self.log.append(f"How many does {self.enemy.value} need to reach {self.target_base}?")
            self.current_answer = str(self.gap)
            self.state = "SOLVING"

        elif spell_name == "Sliding":
            # Generate Sliding problem
            # Enemy value is Minuend. We need a subtrahend.
            if self.enemy.value < 10: self.enemy.value += 20
            subtrahend = random.randint(5, self.enemy.value - 5)

            target_sub = ((subtrahend // 10) + 1) * 10
            k = target_sub - subtrahend

            self.log.append(f"Casting Sliding on {self.enemy.value} - {subtrahend}...")
            self.log.append(f"To make {subtrahend} a base ({target_sub}), what is K?")
            self.current_answer = str(k)
            self.state = "SOLVING"

        elif spell_name == "Count":
            # Simple counting check
            self.log.append(f"What comes after {self.enemy.value}?")
            self.current_answer = str(self.enemy.value + 1)
            self.state = "SOLVING"

    def check_answer(self):
        if self.input_buffer == self.current_answer:
            self.log.append("Correct! The spell hits!")
            self.enemy.defeated = True
            self.state = "WIN"
        else:
            self.log.append(f"Fizzle! Expected {self.current_answer}.")
            self.state = "ENEMY_TURN"
            self.input_buffer = ""

    def enemy_attack(self):
        dmg = random.randint(5, 15)
        self.player.hp -= dmg
        self.log.append(f"The Number attacks! -{dmg} HP")
        if self.player.hp <= 0:
            self.state = "LOSE"
        else:
            self.state = "SELECT_SPELL"

    def draw(self, screen):
        # Draw Battle UI
        pygame.draw.rect(screen, BLACK, (0, 0, SCREEN_WIDTH, SCREEN_HEIGHT))

        # Enemy
```

```
        pygame.draw.rect(screen, RED, (SCREEN_WIDTH//2 - 50, 100, 100, 100))
        val_text = FONT_TITLE.render(str(self.enemy.value), True, WHITE)
        screen.blit(val_text, (SCREEN_WIDTH//2 - 20, 130))

        # Player Stats
        stats = FONT_MAIN.render(f"HP: {self.player.hp}/{self.player.max_hp}", True, GREEN)
        screen.blit(stats, (50, 400))

        # Log
        y = 450
        for line in self.log[-4:]:
            text = FONT_SMALL.render(line, True, WHITE)
            screen.blit(text, (50, y))
            y += 25

        # Input/Menu
        if self.state == "SELECT_SPELL":
            menu_y = 400
            screen.blit(FONT_MAIN.render("Select Spell:", True, GOLD), (400, menu_y))
            for i, spell in enumerate(self.player.spells):
                txt = FONT_MAIN.render(f"{i+1}. {spell}", True, WHITE)
                screen.blit(txt, (420, menu_y + 30 + i*30))

        elif self.state == "SOLVING":
            # Draw Input Box
            pygame.draw.rect(screen, BLUE, (300, 300, 200, 50), 2)
            ans_txt = FONT_MAIN.render(self.input_buffer, True, WHITE)
            screen.blit(ans_txt, (310, 310))

        elif self.state == "WIN":
            screen.blit(FONT_TITLE.render("VICTORY!", True, GOLD), (300, 250))
            screen.blit(FONT_SMALL.render("Press SPACE to continue", True, WHITE), (320, 300))

        elif self.state == "LOSE":
            screen.blit(FONT_TITLE.render("DEFEAT...", True, RED), (300, 250))

# --- MAIN GAME LOOP ---

def main():
    pygame.init()
    screen = pygame.display.set_mode((SCREEN_WIDTH, SCREEN_HEIGHT))
    pygame.display.set_caption("The Hermeneutic Hero: Legends of Logic")
    clock = pygame.time.Clock()

    # Init Game Objects
    world = World(20, 15)
    player = Player(1, 1)
    game_state = GameState.MENU
    battle = None

    running = True
    while running:
        events = pygame.event.get()
        for event in events:
            if event.type == pygame.QUIT:
                running = False

            # --- MENU INPUT ---
            if game_state == GameState.MENU:
                if event.type == pygame.KEYDOWN:
                    if event.key == pygame.K_RETURN:
                        game_state = GameState.OVERWORLD
                    elif event.key == pygame.K_q:
                        running = False

```

```
            # --- OVERWORLD INPUT ---
            elif game_state == GameState.OVERWORLD:
                if event.type == pygame.KEYDOWN:
                    moved = False
                    if event.key == pygame.K_LEFT: moved = player.move(-1, 0, world)
                    if event.key == pygame.K_RIGHT: moved = player.move(1, 0, world)
                    if event.key == pygame.K_UP: moved = player.move(0, -1, world)
                    if event.key == pygame.K_DOWN: moved = player.move(0, 1, world)

                    if moved:
                        # Check for encounter
                        enemy = world.get_enemy_at(player.grid_x, player.grid_y)
                        if enemy:
                            game_state = GameState.BATTLE
                            battle = BattleSystem(player, enemy)

            # --- BATTLE INPUT ---
            elif game_state == GameState.BATTLE:
                if battle.state == "SELECT_SPELL":
                    if event.type == pygame.KEYDOWN:
                        if event.key == pygame.K_1 and len(player.spells) >= 1:
                            battle.cast_spell(player.spells[0])
                        elif event.key == pygame.K_2 and len(player.spells) >= 2:
                            battle.cast_spell(player.spells[1])
                        elif event.key == pygame.K_3 and len(player.spells) >= 3:
                            battle.cast_spell(player.spells[2])

                elif battle.state == "WIN":
                    if event.type == pygame.KEYDOWN and event.key == pygame.K_SPACE:
                        game_state = GameState.OVERWORLD
                        battle = None

                elif battle.state == "LOSE":
                    if event.type == pygame.KEYDOWN and event.key == pygame.K_SPACE:
                        # Reset game
                        player.hp = player.max_hp
                        player.grid_x, player.grid_y = 1, 1
                        game_state = GameState.OVERWORLD
                        battle = None

                elif battle.state == "ENEMY_TURN":
                     # Auto advance for now
                     pygame.time.wait(1000)
                     battle.enemy_attack()

        # --- UPDATE ---
        if game_state == GameState.BATTLE:
            battle.update(events)

        # --- DRAW ---
        screen.fill(BLACK)

        if game_state == GameState.MENU:
            screen.fill((20, 0, 20))
            title = FONT_TITLE.render("THE HERMENEUTIC HERO", True, GOLD)
            subtitle = FONT_MAIN.render("Legends of Logic", True, WHITE)
            start_txt = FONT_MAIN.render("Press ENTER to Start", True, GREEN)
            quit_txt = FONT_SMALL.render("Press Q to Quit", True, GRAY)

            screen.blit(title, (SCREEN_WIDTH//2 - title.get_width()//2, 150))
            screen.blit(subtitle, (SCREEN_WIDTH//2 - subtitle.get_width()//2, 210))
            screen.blit(start_txt, (SCREEN_WIDTH//2 - start_txt.get_width()//2, 400))
            screen.blit(quit_txt, (SCREEN_WIDTH//2 - quit_txt.get_width()//2, 500))

        elif game_state == GameState.OVERWORLD:
```

```
            off_x, off_y = world.draw(screen)
            player.draw(screen, off_x, off_y)

            # UI
            pygame.draw.rect(screen, BLUE, (0, 500, SCREEN_WIDTH, 100))
            screen.blit(FONT_MAIN.render(f"Hero: {player.name if hasattr(player, 'name') else
            ↪  'Teacher'}", True, WHITE), (20, 520))
            screen.blit(FONT_SMALL.render("Use Arrow Keys to Move. Find the Numbers!", True, WHITE),
            ↪  (20, 550))

            # Draw Level/XP
            lvl_txt = FONT_MAIN.render(f"LVL: {player.level}", True, GOLD)
            screen.blit(lvl_txt, (SCREEN_WIDTH - 100, 520))

        elif game_state == GameState.BATTLE:
            battle.draw(screen)

        pygame.display.flip()
        clock.tick(FPS)

    pygame.quit()
    sys.exit()

if __name__ == "__main__":
    main()

```

# 3 MathAdventure/requirements.txt

```
pygame
pandas

```