**DON'T FORGET TO CREDIT:**
**STANISLAV PETROV – PYTHON IMPLEMENTATION**
**AMRO IBRAHIM – CORE LOGIC**

1. Set up the basic project structure
   - **settings.py**: Contains game constants and configuration
   - **README.md**: Project documentation

   **Core libraries**
   - **Pygame** For graphics and input handling
   - **numpy** For efficient array operations
   - **numba** For performance optimization
   - **math** For mathematical calculations
   - **struct** Binary data handling
   - **Vector2** 2D vectors
   - **random** Random number generation
   - **sys** System operations

2. Implemented WAD file reading
   - **wad_reader.py:**
     Class: **WADReader**

     Key methods:
     **read_header:**
     Reads the 12-byte WAD header
     Returns: wad_type, lump_count, init_offset

     **read_directory:**
     Reads the WAD file directory
     Returns: list of lump information

     **read_vertex:**
     Reads vertex coordinates (x, y)
     Returns: Vector2 position

     **read_linedef:**
     Reads wall definitions
     Returns: Linedef object

     **read_sector**:
     Reads sector information
     Returns: Sector object

   - **wad_data.py:**
     Class: **WADData**

     Key methods:
     **load_map:**
     Loads a specific map from the WAD file

     **load_textures:**
     Loads texture data from the WAD file

     **load_sprites:**
     Loads sprite data from the WAD file

3. Created data structures for game elements
   - **data_types.py**

   Class: **TextureMap**

   Stores texture information

   'name', 'flags', 'width', 'height', 'patch_count', 'patch_maps'

   Class: **Sector**

   Represents a map sector

   'floor_height', 'ceil_height', 'floor_texture', 'ceil_texture', 'light_level'

   Class: **Linedef**

   Represents a wall line

   'start_vertex_id', 'end_vertex_id', 'flags', 'line_type', 'sector_tag'

   Class **Node**

   BSP tree node

   'x_partition', 'y_partition', 'dx_partition', 'dy_partition', 'bbox'

4. Implemented Binary Space Partitioning for efficient rendering
   - **bsp.py**

   Class **BSP**:

   Key Methods:

   **render_bsp_node:**

   Traverses the BSP tree for rendering

   Parameters: node_id - current node to process

   **is_on_back_side:**

   Determines if player is behind a partition

   Parameters: node - BSP node to check

   **check_bbox:**

   Checks if bounding box is in view

   Parameters: bbox - bounding box to check

   **render_sub_sector:**

   Renders a subsector

   Parameters: sub_sector_id - ID of subsector to render

5. Added player movement and controls
   - **player.py**
     Class: **Player**

     Key Methods:
     **control:**
     Handles player input and movement
     **Uses pygame.key.get_pressed()** for input

     **get_height:**
     Manages player height and floor interaction
     Updates player position relative to floor

     **update:**
     Updates player state
     Calls **control()** and **get_height()**

6. Implemented the rendering system
   - **view_renderer.py**
     class **ViewRenderer**:

     Decorators:
     **@staticmethod**
     - A static method is a method that belongs to the class rather than an instance of the class
     - Doesn't require self parameter
     - Can't access or modify class/instance state
     - Can be called without creating an instance
     - Used for utility functions that don't need class data

     **@njit**
     - Numba's Just-In-Time (JIT) compiler decorator
     - Converts Python code to machine code for faster execution
     - Significantly improves performance of numerical computations
     - Works best with loops and array operations
     - Has some limitations on supported Python features

Key Methods:

**Static njit draw_column:**

Draws a vertical line

Parameters: framebuffer, x, y1, y2, color

**draw_flat:**

Draws floor/ceiling

Parameters: texture ID, lighting, coordinates

**draw_wall_col:**

Draws a wall column

Parameters: framebuffer, texture, coordinates, lighting

- **map_renderer.py**
  **Class: MapRenderer:**

  Key Methods:

  **draw_linedefs:**

  Draws wall lines

  **draw_player_pos:**

  Draws player position and FOV

  **draw_vertexes:**

  Draws map vertices

- **seg_handler.py**
  Class: **SegHandler**

  Key Methods:

  **draw_solid_wall_range:**

  Renders solid walls between x1 and x2

  Handles texture mapping and lighting

  **draw_portal_wall_range:**

  Renders portal walls (windows, doors)

  Handles different ceiling/floor heights

**scale_from_global_angle:**

Calculates wall scaling based on distance and angle

Used for perspective correction

**clip_portal_walls:**

Clips portal walls to prevent rendering errors

Handles overlapping walls

**classify_segment:**

Classifies segments for rendering

Determines if wall is solid or portal

- **asset_data.py**

  Class: **AssetData**:

  Key Methods:

  **load_textures:**

  Loads wall and floor textures

  Handles texture mapping

  **load_sprites:**

  Loads game sprites

  Manages sprite animations

  **load_palette:**

  Loads color palette

  Handles color mapping

7. Added map switching and game state management
    - **main.py**
      Class: **DoomEngine:**

      Key Methds:
        **change_map:**
            Switches to a different map
            Parameters: map_name - name of map to load

        **run:**
            Main game loop
            Handles events, updates, and rendering

        **check_events:**
            Handles user input and events
            Includes map switching and exit

        **update:**
            Updates game state
            Calls update methods for all components

    - **GameLauncher.java**
      Class: GameLauncher extends Jframe

      Key Methods:
        **launchGame:**
            Launches the Python game
            Handles input validation
            Manages process creation

        **GameLauncher:**
            Creates GUI window
            Sets up input fields
            Initializes buttons

        **validateInputs:**
            Validates user inputs
            Shows error messages if invalid

- **launch_game.py**

  Additional methods:

  **get_player_speed:**

  Gets player speed from user input

  Validates input values

  **get_rotation_sensitivity:**

  Gets rotation sensitivity from user input

  Validates input values

- **if __name__ == '__main__':**

  Main entry point

  Handles command-line arguments

  Launches game with specified parameters