Programmer’s Documentation

Project Title: Walk3D
Author: *Kaba Kevin Zsolt*
Neptune code: *FF64XM*
Date: *2025 november 12*

1. Overview

This program implements a **console-based 3D raycasting engine** inspired by early first-person games such as *Wolfenstein 3D*. The display is rendered using ASCII characters to simulate walls, ceilings, and floors in a pseudo-3D perspective.

The engine supports:

- Real-time player movement and rotation.
- CSV-based map loading (maps with `#` walls, `.` floor, and `X` player start).
- A main menu system with options such as *Resume*, *Load Map*, *Save Map*, and *Quit*.
- A secondary menu to list and load available `.csv` map files.
- Saving the current map state back to a CSV file.
- Basic cross-platform terminal handling for both Windows and Linux.

The program structure separates **data**, **rendering**, **menus**, and **game logic** into distinct functional areas within one source file.

2. Methods and Algorithm Explanation

2.1. Raycasting Algorithm

The **core rendering algorithm** simulates a 3D environment using the raycasting method:

1. Each column of the screen represents a ray cast from the player's position at a specific angle within the player's field of view.
2. The ray moves forward step by step until it hits a wall (`#`) or exceeds the render distance.
3. The distance to the wall determines the height of the vertical slice rendered for that column.
4. The program assigns ASCII shading characters based on the wall’s distance — closer walls appear darker, farther walls lighter.
5. The lower part of the screen is filled with a gradient floor pattern to simulate depth.

This produces the illusion of a 3D scene in a text-based console environment.

2.2. Map System

Maps are stored in `.csv` files where:

- `#` represents a wall.
- `.` represents open space.

- `X` marks the player's starting position.

The map loader parses these files, storing their content into a dynamically allocated 2D array.

Example map file content:

```
#,#,#,#
#,.,.,#
#,X,.,#
#,#,#,#
```

2.3. Menu System

Menus are dynamically created lists of selectable items.

- Each menu (`main_menu`, `maps_menu`) is represented by a `menu_t` structure containing items, capacities, and selection indices.
- Each menu item includes a label and a function pointer (`menu_action_t`) that defines its behavior.
- Menus are navigated using the keyboard (W/S or UP/DOWN), and actions are executed with ENTER.

The system allows dynamic insertion/removal of menu items based on program state (e.g., adding "Resume" only if a map is loaded).

---

3. Data Structures

3.1. `map`

Stores the current map grid and player starting position.

```
typedef struct map {
    int width;
    int height;
    char *m;                // Flattened 2D array storing map characters
    int player_start_x;
    int player_start_y;
} map;
```

3.2. `screen`

Manages the ASCII display buffer.

```
typedef struct screen {
    int width;
    int height;
    char *display;          // Buffer containing characters to render
} screen;
```

3.3. `player_model`

Contains player position, viewing angle, and movement parameters.

```
typedef struct player_model {
    double x, y;            // Player coordinates
    double a;               // Player viewing angle
    double fov;             // Field of view
    double speed;           // Movement speed
    double turn_speed;      // Rotation speed
    int render_distance;    // Maximum visible distance
} player_model;
```

3.4. `menu_t` and `menu_item`

Dynamic menu system with function pointers for item actions.

```
typedef void (*menu_action_t)(void);

typedef struct menu_item {
    char *text;             // Item label
    menu_action_t action;   // Function to execute
} menu_item;

typedef struct menu_t {
    menu_item *items;
    int count;              // Number of items
    int capacity;           // Allocated size
    int selected;           // Index of currently selected item
} menu_t;
```

---

4. Major Modules and Functions

4.1. Initialization

- `get_screen_size()` Detects console size and allocates the display buffer.
- `setup_player_global()` Initializes player position and default parameters.
- `menu_init(menu_t *m)` Allocates and initializes a new menu.

4.2. Rendering and Game Loop

- `calc_column(int x)` Core of the raycasting renderer. Casts a ray for a given screen column and fills the display buffer with appropriate shading based on wall distance.

- `render_screen()` Clears the terminal and prints the full ASCII display buffer to the console efficiently.
- `game_loop()` Handles input, updates player position and angle, toggles menus, and continuously renders the game frame-by-frame.

4.3. Menu Actions

Function	Description
`action_resume()`	Exits the menu and resumes the game.
`action_quit()`	Sets a flag to terminate the main loop.
`action_load_map()`	Opens the map selection menu.
`action_load_selected_map()`	Loads the currently highlighted map file.
`action_save_map()`	Saves the current map with player position marked as `X`.

4.4. File and Map Handling

- `map load_map(char file_path[])` Loads a CSV map file, determines width and height, and fills the `map` structure.
- `print_map(const map *m)` Prints the map in 2D mode for debugging.
- `action_save_map()` Saves the current map back to a CSV file.

4.5. Utility Functions

- `current_timestamp_ms()` Returns the current system time in milliseconds (used for frame timing).
- `update_screen_size()` Reallocates screen buffer when terminal dimensions change.
- `get_shade(double distance_to_wall)` Returns a shading character based on the distance from the player.

---

5. Program Flow Summary

1. **Initialization**:
   - Console dimensions and player parameters are set up.
   - Menus are initialized.
2. **Main Menu**:
   - Player can select *Load Map* or *Quit*.
3. **Map Selection**:
   - The program scans the current directory for `.csv` files.

    - User can load one, returning to the game view.

4. **Game Rendering**:

    - The player moves with `W`, `A`, `S`, `D`.
    - Raycasting generates a 3D illusion in the terminal.

5. **In-Game Menu**:

    - Pressing ESC toggles the main menu.
    - The user can save, resume, or quit.

---

6. Memory Management and Cleanup

All dynamic allocations (menus, map data, and display buffer) are freed at the end of the program.

```
menu_free(&main_menu);
menu_free(&maps_menu);
free(output_screen.display);
free(game_map.m);
free(current_map_file);
```

---

7. Platform Compatibility

- On **Windows**, the program uses `GetAsyncKeyState()` and `GetSystemTimeAsFileTime()` for input and timing.
- On **Unix-like systems**, `termios`, `ioctl`, and `gettimeofday()` are used instead.
- The program abstracts away platform-specific differences via conditional compilation (`#ifdef _WIN32`).