BIT Kitchen Nightmare README

Overview

This simple SDL game demonstrates the use of SDL2 and SDL_image libraries to create a 2D game in C. The game features now have main character (MC), enemy spawning, movement, and basic collision handling. This document outlines the game's structure, initialization process, main game loop, and key functionalities such as input handling, enemy management, and rendering.

Prerequisites

SDL2 library

SDL2_image extension library

SDL2_mixer

SDL2_ttf

Update

In this version, We refactored the main game structure code so that it will not be difficult to update scale up and maintenance.

First, we split each game state into its own game loop and encapsulate it with a big game loop. As a result, it will be easy to add, delete or edit each game state and will not affect others.

File Structure

main.c: Contains the game's main loop, including initialization, input processing, game state updates, and rendering.

constant.h: Defines constants used throughout the game, such as window size, FPS settings, and game state identifiers.

AudioManager.h: Manages audio functionalities, including background music and sound effects.

constant.h

Defines essential constants for game configuration, such as screen dimensions, FPS limits, and the maximum number of enemies. It ensures consistency and easy adjustments to the game's fundamental parameters.

AudioManager.h

Manages all audio-related functionalities, loading music and sound effects, playing, pausing, and stopping audio as needed to enhance the game's immersive experience structure.

Main Function (main)

The entry point of the program. It initializes the game window and enters the main game loop, which processes input, updates game state, and renders the game until the game is exited. It also handles the timing for enemy wave spawning.

Initialization Functions

initialize_window

Purpose: Initializes SDL, creates the game window, and sets up the renderer. It also initializes SDL_image for texture loading.

Key Operations:

Initialize SDL with SDL_Init.

Create a game window with SDL_CreateWindow.

Create a renderer with SDL_CreateRenderer.

Initialize SDL_image with IMG_Init.

gameplay_setup

Purpose: Initializes game entities like the MC state, texture and initialize map texture and main_character origin .

Key Operations:

Set initial positions and properties of the MC.

Load textures for the MC, enemies, background.

Initialize the camera to follow the MC.

setup

Purpose: Initialize UI element and reset time.

Key Operations:

Load textures for UI elements.

Call game_play_setup function.

reset_game_state

Purpose: Reset game time and enemies to the beginning value.

Key Operations:

Reset delta_time, wave and last_time_frame.

Memory set the enemies.

Call gameplay_setup to reset a game process.

Update game state

update

Purpose: Updates the game state, including moving entities and handling game logic.

Key Operations:

Apply movement to the MC based on input flags.

Update enemy positions and behaviors.

Update the camera to follow the MC.

process_input(all game state)

Purpose: Processes user input to control the game, including moving the MC and navigating menus.

Key Operations:

Poll SDL events to detect keyboard presses and mouse clicks.

Set movement flags based on key states.

Handle menu interactions and game quitting.

Enemy related function

update_enemies

Purpose: Moves enemies towards the MC and handles enemy collisions.

Key Operations:

Calculate movement vectors for each enemy towards the MC.

Update enemy positions based on their movement speed and direction.

Check and resolve collisions between enemies to prevent overlapping.

initialize_enemies

Purpose: Defines the properties and textures for different enemy types.

Key Operations:

Load textures for each enemy type.

Set properties like size, speed, and health for each type.

initialize_stage1_enemies

Purpose: Sets up enemy waves for the first stage.

Key Operations:

Define the composition of each wave by specifying the number of each enemy type.

spawn_wave

Purpose: Spawns a new wave of enemies.

Key Operations:

For each enemy in the wave, find an inactive enemy slot and initialize it with the enemy type and spawn position.

Rendering

render(all game state)

Purpose: Renders the game world, entities to the screen, menu, and pause menu.

Key Operations:

Clear the previous frame.

Render the game background, MC, enemies, and UI elements based on the game state.

Present the rendered frame to the display.

render_enemies

Purpose: Display enemies on the screen.

Key Operations:

Iterate through active enemies and render them at their current positions relative to the camera.

render_health_bar

Purpose: Renders a health bar UI element.

Key Operations:

Draw a background and foreground rectangle to represent the MC's health.

Enemy Management.

render_enemy_damage

Purpose: Renders the damage that main_character does to the enemies.

Key Operations:

Load font.

Render the damage above the enemies.

Combat mechanic functions

check_collision_and_apply_damage

Purpose: To do the damage to the main character

Key Operations:

Do damage to the main character.

Check the enemy’s rect and main character rect. If the 2 rect intersect MC, the health will decrease by the enemy attack multiplied by delta time.

initialize_attacks

Purpose: To store data of each attack on the main character to the struct. (For now we only have normal attacks.)

Key Operations:

Load textures for each attack VFX.

Set properties like size, cooldown, damage.

updated_attack

Purpose: To define a place and area of the attack in the game loop.

Key Operations:

Will find the attack that the main character has and reach the time to activate and it will create an attack area in the period of time that was defined. The place of the attack_rect is

If main character face right x = MC.x + (MC.width / 2)

If main character face left x = Mc.x – [(Attack_area.width – MC.width) / 2]

render_attacks

Purpose: Display attack VFX on the screen.

Key Operations:

Render VFX at their current positions relative to the camera.

apply_attack_damage_to_enemies

Purpose: Apply damage to the enemy

Key Operations:

It will find the enemy_rect that intersects with attack_rect. If it intersects it will apply damage to the enemy by the attack damage of that attack.

Utility Functions

load_texture

Purpose: Loads an image file into an SDL_Texture.

Key Operations:

Load an image with IMG_Load.

Create a texture from the loaded surface with SDL_CreateTextureFromSurface.

update_camera

Purpose: Updates the camera position to keep the MC centered.

Key Operations:

Adjust the camera's position based on the MC's position.

Ensuring the camera stays within the bounds of the game world.

cap_framerate

Purpose: Limits the game to a fixed frame rate.

Key Operations:

Calculate the time to delay each frame to cap the game at a specified frame rate.

destroy_window

Purpose: Frees resources and quits SDL cleanly.

Key Operations:

Destroy textures, renderer, and window.

Quit SDL and SDL_image.

Audio Functions

AudioManager_Init

Purpose : Initiate audio system.

AudioManager_LoadMusic

Purpose: Load music from asset folder to be played.

AudioManager_PauseMusic

Purpose: Pause the music that is currently playing.

AudioManager_ResumeMusic

Purpose: Resume the music that is paused.

AudioManager_PlayMusic

Purpose: Play the music that is loaded from the directory.

AudioManager_StopMusic

Purpose: Stop playing the music. if it is played again the music will start from start.

AudioManager_Cleanup

Purpose: Free music object and shut down audio system.