**Documentation for the 2D Game Solution Using SDL**

**Overview**

This solution is a 2D game framework developed using **SDL (Simple DirectMedia Layer)**. It includes features for rendering particle effects, managing game objects (players, invaders, bullets), handling input devices (keyboard, mouse, controllers), and creating visually dynamic experiences with fire, plasma, explosion, and starfield effects.

**Key Features:**

* **Dynamic Particle Effects:** Explosions, fire, and plasma effects rendered in real time.
* **Game Object Management:** Includes a player, bullets, and invaders, all updated and rendered each frame.
* **Cross-Platform Controller Support:** Uses SDL GameController API for input handling.
* **Dynamic Screen Resolution Support:** Adjusts game elements when the window is resized.
* **Efficient Rendering:** Built with SDL\_Renderer for hardware-accelerated rendering.

**Code Structure**

The project is modular and organized into the following files:

**1. Main Program (Main.cpp)**

* **Purpose:** Entry point of the game.
* **Key Responsibilities:**
  + Initializes SDL subsystems and game objects.
  + Manages the game loop (event handling, updates, and rendering).
  + Handles fullscreen toggling and dynamic resolution changes.

**Code Example:**

int APIENTRY WinMain(HINSTANCE hInst, HINSTANCE hInstPrev, PSTR cmdline, int cmdshow) {

if (SDL\_Init(SDL\_INIT\_VIDEO | SDL\_INIT\_GAMECONTROLLER | SDL\_INIT\_JOYSTICK) < 0) {

std::cout << "SDL could not initialize! SDL\_Error: " << SDL\_GetError() << std::endl;

return -1;

}

SDL\_Window\* window = SDL\_CreateWindow("Starfield", SDL\_WINDOWPOS\_UNDEFINED, SDL\_WINDOWPOS\_UNDEFINED, 480, 360, SDL\_WINDOW\_SHOWN | SDL\_WINDOW\_RESIZABLE);

if (window == NULL) {

std::cout << "Window could not be created! SDL\_Error: " << SDL\_GetError() << std::endl;

return -1;

}

SDL\_Renderer\* renderer = SDL\_CreateRenderer(window, -1, SDL\_RENDERER\_ACCELERATED);

if (renderer == NULL) {

std::cout << "Renderer could not be created! SDL\_Error: " << SDL\_GetError() << std::endl;

return -1;

}

// Main game loop

while (true) {

SDL\_Event event;

while (SDL\_PollEvent(&event)) {

if (event.type == SDL\_QUIT) {

return 0;

}

}

SDL\_RenderClear(renderer);

// Add game logic and rendering here

SDL\_RenderPresent(renderer);

}

}

**2. Particle Effects**

**Explosion (Explosion.cpp and Explosion.h)**

* **Purpose:** Simulates an explosion using particles and color palettes.
* **Key Components:**
  + fn\_vInit\_Explosion: Initializes the explosion effect, including the color palette and particle setup.
  + fm\_iRun\_Explosion\_RendererStyle: Updates particle positions, applies gravity, and renders the explosion effect.

**Code Example:**

void fn\_vInit\_Explosion(SDL\_Renderer\* renderer) {

for (int i = 0; i < 256; ++i) {

colors\_explosion[i].r = i;

colors\_explosion[i].g = 255 - i;

colors\_explosion[i].b = i / 2;

}

fn\_vRandomly\_Init\_Particles();

explosion\_texture = SDL\_CreateTexture(renderer, SDL\_PIXELFORMAT\_RGBA8888, SDL\_TEXTUREACCESS\_STREAMING, 480, 360);

}

void fn\_vRandomly\_Init\_Particles() {

for (int i = 0; i < NUMBER\_OF\_PARTICLES; i++) {

particles[i].xpos = 240 + (rand() % 20 - 10);

particles[i].ypos = 180 + (rand() % 20 - 10);

particles[i].xdir = rand() % 5 - 2;

particles[i].ydir = rand() % 5 - 2;

particles[i].colorindex = 255;

particles[i].dead = 0;

}

}

**Plasma Effect (Plasma\_effect.cpp and Plasma\_effect.h)**

* **Purpose:** Renders a plasma effect using sine wave calculations and color palettes.
* **Key Components:**
  + fn\_vDemoScene\_ColorPalette\_Plasma\_init: Creates a palette for smooth plasma transitions.
  + fn\_vDemoScene\_Plasma\_RenderWithPalette: Updates and renders the plasma effect using a GPU texture.

**Code Example:**

void fn\_vDemoScene\_ColorPalette\_Plasma\_init(SDL\_Renderer\* renderer) {

for (int i = 0; i < 256; ++i) {

colors\_plasma[i].r = i;

colors\_plasma[i].g = 255 - i;

colors\_plasma[i].b = (i / 2) % 255;

}

plasma\_texture = SDL\_CreateTexture(renderer, SDL\_PIXELFORMAT\_RGBA8888, SDL\_TEXTUREACCESS\_STREAMING, 480, 360);

}

void fn\_vDemoScene\_Plasma\_RenderWithPalette(SDL\_Renderer\* renderer) {

void\* pixels;

int pitch;

SDL\_LockTexture(plasma\_texture, NULL, &pixels, &pitch);

for (int y = 0; y < 360; ++y) {

for (int x = 0; x < 480; ++x) {

int index = (x + y) % 256;

((Uint32\*)pixels)[y \* 480 + x] = SDL\_MapRGBA(renderer->format, colors\_plasma[index].r, colors\_plasma[index].g, colors\_plasma[index].b, 255);

}

}

SDL\_UnlockTexture(plasma\_texture);

SDL\_RenderCopy(renderer, plasma\_texture, NULL, NULL);

}

**3. Game Objects**

**Player and Bullets (player\_bullets\_invaders.h)**

* **Player:**
  + Properties: x, y, width, height, and velocity.
  + Methods: update ensures the player stays within screen bounds.
* **Bullets:**
  + Player bullets (mPlayerBullet): Move upwards and check for collisions with invaders.

**Code Example:**

class Player {

public:

int x, y;

int width, height;

int velocity;

void update(int screen\_width) {

if (x + velocity < 0 || x + velocity + width > screen\_width) {

velocity = 0;

}

x += velocity;

}

};

struct Bullet {

int x, y;

bool active;

void update() {

if (active) {

y -= 5;

if (y < 0) active = false;

}

}

};

**Additional Topics**

**Collision Detection**

**Code Example:**

for (int i = 0; i < bullets.size(); ++i) {

for (int j = 0; j < invaders.size(); ++j) {

if (bullets[i].x > invaders[j].x && bullets[i].x < invaders[j].x + invaders[j].width &&

bullets[i].y > invaders[j].y && bullets[i].y < invaders[j].y + invaders[j].height) {

bullets[i].active = false;

invaders[j].destroyed = true;

}

}

}