// File: PlayerObject.cs

using Silk.NET.Maths;

namespace TheAdventure.Models;

public class PlayerObject : RenderableGameObject

{

private const int \_speed = 128;

public int Lives { get; private set; } = 3;

public void LoseLife()

{

Lives--;

if (Lives > 0)

{

Console.WriteLine($"You have {Lives} lives left.");

}

if (Lives <= 0)

{

GameOver();

}

}

public enum PlayerStateDirection

{

None = 0, Down, Up, Left, Right,

}

public enum PlayerState

{

None = 0, Idle, Move, Attack, GameOver

}

public (PlayerState State, PlayerStateDirection Direction) State { get; private set; }

public PlayerObject(SpriteSheet spriteSheet, int x, int y) : base(spriteSheet, (x, y))

{

SetState(PlayerState.Idle, PlayerStateDirection.Down);

}

public void SetState(PlayerState state) => SetState(state, State.Direction);

public void SetState(PlayerState state, PlayerStateDirection direction)

{

if (State.State == PlayerState.GameOver || (State.State == state && State.Direction == direction))

return;

if (state == PlayerState.None && direction == PlayerStateDirection.None)

SpriteSheet.ActivateAnimation(null);

else if (state == PlayerState.GameOver)

SpriteSheet.ActivateAnimation(Enum.GetName(state));

else

SpriteSheet.ActivateAnimation($"{Enum.GetName(state)}{Enum.GetName(direction)}");

State = (state, direction);

}

public void GameOver() => SetState(PlayerState.GameOver, PlayerStateDirection.None);

public void Attack()

{

if (State.State != PlayerState.GameOver)

SetState(PlayerState.Attack, State.Direction);

}

public void UpdatePosition(double up, double down, double left, double right, int width, int height, double time)

{

if (State.State == PlayerState.GameOver) return;

var pixelsToMove = \_speed \* (time / 1000.0);

var x = Position.X + (int)(right \* pixelsToMove) - (int)(left \* pixelsToMove);

var y = Position.Y + (int)(down \* pixelsToMove) - (int)(up \* pixelsToMove);

var newState = (x == Position.X && y == Position.Y) ?

(State.State == PlayerState.Attack && SpriteSheet.AnimationFinished ? PlayerState.Idle : PlayerState.Idle)

: PlayerState.Move;

var newDirection = State.Direction;

if (y < Position.Y) newDirection = PlayerStateDirection.Up;

else if (y > Position.Y) newDirection = PlayerStateDirection.Down;

else if (x < Position.X) newDirection = PlayerStateDirection.Left;

else if (x > Position.X) newDirection = PlayerStateDirection.Right;

if (newState != State.State || newDirection != State.Direction)

SetState(newState, newDirection);

Position = (x, y);

}

}

// File: Engine.cs

using System.Reflection;

using System.Text.Json;

using Silk.NET.Maths;

using TheAdventure.Models;

using TheAdventure.Models.Data;

using TheAdventure.Scripting;

namespace TheAdventure;

public class Engine

{

private readonly GameRenderer \_renderer;

private readonly Input \_input;

private readonly ScriptEngine \_scriptEngine = new();

private readonly Dictionary<int, GameObject> \_gameObjects = new();

private readonly Dictionary<string, TileSet> \_loadedTileSets = new();

private readonly Dictionary<int, Tile> \_tileIdMap = new();

private Level \_currentLevel = new();

private PlayerObject? \_player;

private int \_heartTextureId;

private DateTimeOffset \_lastUpdate = DateTimeOffset.Now;

private bool \_isGameOver = false;

public Engine(GameRenderer renderer, Input input)

{

\_renderer = renderer;

\_input = input;

\_input.OnMouseClick += (\_, coords) => AddBomb(coords.x, coords.y);

}

public void SetupWorld()

{

\_player = new(SpriteSheet.Load(\_renderer, "Player.json", "Assets"), 100, 100);

var heartPath = Path.Combine("Assets", "heart.png");

if (!File.Exists(heartPath))

{

throw new FileNotFoundException("Heart image not found!", heartPath);

}

\_heartTextureId = \_renderer.LoadTexture(heartPath, out var heartTexture);

if (\_heartTextureId == 0)

{

throw new Exception("Failed to load heart texture.");

}

var levelContent = File.ReadAllText(Path.Combine("Assets", "terrain.tmj"));

var level = JsonSerializer.Deserialize<Level>(levelContent);

if (level == null) throw new Exception("Failed to load level");

foreach (var tileSetRef in level.TileSets)

{

var tileSetContent = File.ReadAllText(Path.Combine("Assets", tileSetRef.Source));

var tileSet = JsonSerializer.Deserialize<TileSet>(tileSetContent);

if (tileSet == null) throw new Exception("Failed to load tile set");

foreach (var tile in tileSet.Tiles)

{

tile.TextureId = \_renderer.LoadTexture(Path.Combine("Assets", tile.Image), out \_);

\_tileIdMap.Add(tile.Id!.Value, tile);

}

\_loadedTileSets.Add(tileSet.Name, tileSet);

}

if (level.Width == null || level.Height == null) throw new Exception("Invalid level dimensions");

if (level.TileWidth == null || level.TileHeight == null) throw new Exception("Invalid tile dimensions");

\_renderer.SetWorldBounds(new Rectangle<int>(0, 0, level.Width.Value \* level.TileWidth.Value,

level.Height.Value \* level.TileHeight.Value));

\_currentLevel = level;

\_scriptEngine.LoadAll(Path.Combine("Assets", "Scripts"));

}

public void ProcessFrame()

{

var currentTime = DateTimeOffset.Now;

var msSinceLastFrame = (currentTime - \_lastUpdate).TotalMilliseconds;

if (\_isGameOver) return;

\_lastUpdate = currentTime;

if (\_player == null) return;

double up = \_input.IsUpPressed() ? 1.0 : 0.0;

double down = \_input.IsDownPressed() ? 1.0 : 0.0;

double left = \_input.IsLeftPressed() ? 1.0 : 0.0;

double right = \_input.IsRightPressed() ? 1.0 : 0.0;

bool isAttacking = \_input.IsKeyAPressed() && (up + down + left + right <= 1);

bool addBomb = \_input.IsKeyBPressed();

\_player.UpdatePosition(up, down, left, right, 48, 48, msSinceLastFrame);

if (isAttacking) \_player.Attack();

if (\_player.Lives <= 0)

{

if (!\_isGameOver)

{

Console.WriteLine("Game Over! Toate viețile s-au terminat.");

\_isGameOver = true;

}

return;

}

\_scriptEngine.ExecuteAll(this);

if (addBomb)

{

AddBomb(\_player.Position.X, \_player.Position.Y, false);

}

}

public void RenderFrame()

{

\_renderer.SetDrawColor(0, 0, 0, 255);

\_renderer.ClearScreen();

var playerPosition = \_player!.Position;

// 1. RENDER WORLD

\_renderer.CameraLookAt(playerPosition.X, playerPosition.Y);

RenderTerrain();

RenderAllObjects();

// 2. RENDER UI

\_renderer.CameraLookAt(0, 0); // UI fix pe ecran

for (int i = 0; i < \_player.Lives; i++)

{

var heartRect = new Rectangle<int>(10 + i \* 24, 10, 32, 32); // era 16x16

\_renderer.RenderTexture(\_heartTextureId, new Rectangle<int>(0, 0, 512, 512), heartRect); // era 16x16 sursă

}

// 3. REVENIM la camera world, pentru consistență

\_renderer.CameraLookAt(playerPosition.X, playerPosition.Y);

// 4. FINALIZĂM

\_renderer.PresentFrame();

}

public void RenderAllObjects()

{

var toRemove = new List<int>();

foreach (var gameObject in GetRenderables())

{

gameObject.Render(\_renderer);

if (gameObject is TemporaryGameObject { IsExpired: true } tempGameObject)

{

toRemove.Add(tempGameObject.Id);

}

}

foreach (var id in toRemove)

{

\_gameObjects.Remove(id, out var gameObject);

if (\_player == null) continue;

var tempGameObject = (TemporaryGameObject)gameObject!;

var deltaX = Math.Abs(\_player.Position.X - tempGameObject.Position.X);

var deltaY = Math.Abs(\_player.Position.Y - tempGameObject.Position.Y);

if (deltaX < 32 && deltaY < 32)

{

\_player.LoseLife();

}

}

\_player?.Render(\_renderer);

}

public void RenderTerrain()

{

foreach (var currentLayer in \_currentLevel.Layers)

{

for (int i = 0; i < \_currentLevel.Width; ++i)

{

for (int j = 0; j < \_currentLevel.Height; ++j)

{

int? dataIndex = j \* currentLayer.Width + i;

if (dataIndex == null) continue;

var currentTileId = currentLayer.Data[dataIndex.Value] - 1;

if (currentTileId == null) continue;

var currentTile = \_tileIdMap[currentTileId.Value];

var tileWidth = currentTile.ImageWidth ?? 0;

var tileHeight = currentTile.ImageHeight ?? 0;

var sourceRect = new Rectangle<int>(0, 0, tileWidth, tileHeight);

var destRect = new Rectangle<int>(i \* tileWidth, j \* tileHeight, tileWidth, tileHeight);

\_renderer.RenderTexture(currentTile.TextureId, sourceRect, destRect);

}

}

}

}

public IEnumerable<RenderableGameObject> GetRenderables()

{

foreach (var gameObject in \_gameObjects.Values)

{

if (gameObject is RenderableGameObject renderableGameObject)

{

yield return renderableGameObject;

}

}

}

public (int X, int Y) GetPlayerPosition() => \_player!.Position;

public void AddBomb(int X, int Y, bool translateCoordinates = true)

{

var worldCoords = translateCoordinates ? \_renderer.ToWorldCoordinates(X, Y) : new Vector2D<int>(X, Y);

SpriteSheet spriteSheet = SpriteSheet.Load(\_renderer, "BombExploding.json", "Assets");

spriteSheet.ActivateAnimation("Explode");

TemporaryGameObject bomb = new(spriteSheet, 2.1, (worldCoords.X, worldCoords.Y));

\_gameObjects.Add(bomb.Id, bomb);

}

}

using Silk.NET.Maths;

using Silk.NET.SDL;

using SixLabors.ImageSharp;

using SixLabors.ImageSharp.PixelFormats;

using TheAdventure.Models;

using Point = Silk.NET.SDL.Point;

namespace TheAdventure;

public unsafe class GameRenderer

{

private Sdl \_sdl;

private Renderer\* \_renderer;

private GameWindow \_window;

private Camera \_camera;

private Dictionary<int, IntPtr> \_texturePointers = new();

private Dictionary<int, TextureData> \_textureData = new();

private int \_textureId;

public GameRenderer(Sdl sdl, GameWindow window)

{

\_sdl = sdl;

\_renderer = (Renderer\*)window.CreateRenderer();

\_sdl.SetRenderDrawBlendMode(\_renderer, BlendMode.Blend);

\_window = window;

var windowSize = window.Size;

\_camera = new Camera(windowSize.Width, windowSize.Height);

}

public void SetWorldBounds(Rectangle<int> bounds)

{

\_camera.SetWorldBounds(bounds);

}

public void CameraLookAt(int x, int y)

{

\_camera.LookAt(x, y);

}

public int LoadTexture(string fileName, out TextureData textureInfo)

{

using (var fStream = new FileStream(fileName, FileMode.Open))

{

var image = Image.Load<Rgba32>(fStream);

textureInfo = new TextureData()

{

Width = image.Width,

Height = image.Height

};

var imageRAWData = new byte[textureInfo.Width \* textureInfo.Height \* 4];

image.CopyPixelDataTo(imageRAWData.AsSpan());

fixed (byte\* data = imageRAWData)

{

var imageSurface = \_sdl.CreateRGBSurfaceWithFormatFrom(data, textureInfo.Width,

textureInfo.Height, 8, textureInfo.Width \* 4, (uint)PixelFormatEnum.Rgba32);

if (imageSurface == null)

throw new Exception("Failed to create surface from image data.");

var imageTexture = \_sdl.CreateTextureFromSurface(\_renderer, imageSurface);

\_sdl.FreeSurface(imageSurface);

if (imageTexture == null)

throw new Exception("Failed to create texture from surface.");

int currentId = \_textureId++;

\_textureData[currentId] = textureInfo;

\_texturePointers[currentId] = (IntPtr)imageTexture;

return currentId;

}

}

}

public void RenderTexture(int textureId, Rectangle<int> src, Rectangle<int> dst,

RendererFlip flip = RendererFlip.None, double angle = 0.0, Point center = default)

{

if (\_texturePointers.TryGetValue(textureId, out var imageTexture))

{

var translatedDst = \_camera.ToScreenCoordinates(dst);

\_sdl.RenderCopyEx(\_renderer, (Texture\*)imageTexture, in src,

in translatedDst,

angle,

in center, flip);

}

}

public Vector2D<int> ToWorldCoordinates(int x, int y)

{

return \_camera.ToWorldCoordinates(new Vector2D<int>(x, y));

}

public void SetDrawColor(byte r, byte g, byte b, byte a)

{

\_sdl.SetRenderDrawColor(\_renderer, r, g, b, a);

}

public void ClearScreen()

{

\_sdl.RenderClear(\_renderer);

}

public void PresentFrame()

{

\_sdl.RenderPresent(\_renderer);

}

}

V 2

// File: Engine.cs

using System.Reflection;

using System.Text.Json;

using Silk.NET.Maths;

using TheAdventure.Models;

using TheAdventure.Models.Data;

using TheAdventure.Scripting;

namespace TheAdventure;

public class Engine

{

private readonly GameRenderer \_renderer;

private readonly Input \_input;

private readonly ScriptEngine \_scriptEngine = new();

private readonly Dictionary<int, GameObject> \_gameObjects = new();

private readonly Dictionary<string, TileSet> \_loadedTileSets = new();

private readonly Dictionary<int, Tile> \_tileIdMap = new();

private Level \_currentLevel = new();

private PlayerObject? \_player;

private int \_heartTextureId;

private int \_gameOverTextureId;

private DateTimeOffset \_lastUpdate = DateTimeOffset.Now;

private bool \_isGameOver = false;

public Engine(GameRenderer renderer, Input input)

{

\_renderer = renderer;

\_input = input;

\_input.OnMouseClick += (\_, coords) => AddBomb(coords.x, coords.y);

}

public void SetupWorld()

{

\_player = new(SpriteSheet.Load(\_renderer, "Player.json", "Assets"), 100, 100);

var heartPath = Path.Combine("Assets", "heart.png");

if (!File.Exists(heartPath))

{

throw new FileNotFoundException("Heart image not found!", heartPath);

}

\_heartTextureId = \_renderer.LoadTexture(heartPath, out var heartTexture);

if (\_heartTextureId == 0)

{

throw new Exception("Failed to load heart texture.");

}

var gameOverPath = Path.Combine("Assets", "game\_over.png");

if (!File.Exists(gameOverPath))

{

throw new FileNotFoundException("Game Over image not found!", gameOverPath);

}

\_gameOverTextureId = \_renderer.LoadTexture(gameOverPath, out \_);

var levelContent = File.ReadAllText(Path.Combine("Assets", "terrain.tmj"));

var level = JsonSerializer.Deserialize<Level>(levelContent);

if (level == null) throw new Exception("Failed to load level");

foreach (var tileSetRef in level.TileSets)

{

var tileSetContent = File.ReadAllText(Path.Combine("Assets", tileSetRef.Source));

var tileSet = JsonSerializer.Deserialize<TileSet>(tileSetContent);

if (tileSet == null) throw new Exception("Failed to load tile set");

foreach (var tile in tileSet.Tiles)

{

tile.TextureId = \_renderer.LoadTexture(Path.Combine("Assets", tile.Image), out \_);

\_tileIdMap.Add(tile.Id!.Value, tile);

}

\_loadedTileSets.Add(tileSet.Name, tileSet);

}

if (level.Width == null || level.Height == null) throw new Exception("Invalid level dimensions");

if (level.TileWidth == null || level.TileHeight == null) throw new Exception("Invalid tile dimensions");

\_renderer.SetWorldBounds(new Rectangle<int>(0, 0, level.Width.Value \* level.TileWidth.Value,

level.Height.Value \* level.TileHeight.Value));

\_currentLevel = level;

\_scriptEngine.LoadAll(Path.Combine("Assets", "Scripts"));

}

public void ProcessFrame()

{

var currentTime = DateTimeOffset.Now;

var msSinceLastFrame = (currentTime - \_lastUpdate).TotalMilliseconds;

if (\_isGameOver) return;

\_lastUpdate = currentTime;

if (\_player == null) return;

double up = \_input.IsUpPressed() ? 1.0 : 0.0;

double down = \_input.IsDownPressed() ? 1.0 : 0.0;

double left = \_input.IsLeftPressed() ? 1.0 : 0.0;

double right = \_input.IsRightPressed() ? 1.0 : 0.0;

bool isAttacking = \_input.IsKeyAPressed() && (up + down + left + right <= 1);

bool addBomb = \_input.IsKeyBPressed();

\_player.UpdatePosition(up, down, left, right, 48, 48, msSinceLastFrame);

if (isAttacking) \_player.Attack();

if (\_player.Lives <= 0)

{

if (!\_isGameOver)

{

Console.WriteLine("Game Over! Toate viețile s-au terminat.");

\_isGameOver = true;

}

return;

}

\_scriptEngine.ExecuteAll(this);

if (addBomb)

{

AddBomb(\_player.Position.X, \_player.Position.Y, false);

}

}

public void RenderFrame()

{

\_renderer.SetDrawColor(0, 0, 0, 255);

\_renderer.ClearScreen();

var playerPosition = \_player!.Position;

\_renderer.CameraLookAt(playerPosition.X, playerPosition.Y);

RenderTerrain();

RenderAllObjects();

\_renderer.CameraLookAt(0, 0);

for (int i = 0; i < \_player.Lives; i++)

{

var heartRect = new Rectangle<int>(10 + i \* 24, 10, 32, 32);

\_renderer.RenderTexture(\_heartTextureId, new Rectangle<int>(0, 0, 512, 512), heartRect);

}

if (\_isGameOver)

{

RenderGameOver();

}

\_renderer.CameraLookAt(playerPosition.X, playerPosition.Y);

\_renderer.PresentFrame();

}

private void RenderGameOver()

{

var (width, height) = \_renderer.GetWindowSize();

// alegem dimensiune mai mică ca să încapă garantat

int renderWidth = 256;

int renderHeight = 256;

var dest = new Rectangle<int>(

(width - renderWidth) / 2,

(height - renderHeight) / 2,

renderWidth,

renderHeight

);

\_renderer.RenderTexture(\_gameOverTextureId, new Rectangle<int>(0, 0, 2084, 2084), dest);

}

public void RenderAllObjects()

{

var toRemove = new List<int>();

foreach (var gameObject in GetRenderables())

{

gameObject.Render(\_renderer);

if (gameObject is TemporaryGameObject { IsExpired: true } tempGameObject)

{

toRemove.Add(tempGameObject.Id);

}

}

foreach (var id in toRemove)

{

\_gameObjects.Remove(id, out var gameObject);

if (\_player == null) continue;

var tempGameObject = (TemporaryGameObject)gameObject!;

var deltaX = Math.Abs(\_player.Position.X - tempGameObject.Position.X);

var deltaY = Math.Abs(\_player.Position.Y - tempGameObject.Position.Y);

if (deltaX < 32 && deltaY < 32)

{

\_player.LoseLife();

}

}

\_player?.Render(\_renderer);

}

public void RenderTerrain()

{

foreach (var currentLayer in \_currentLevel.Layers)

{

for (int i = 0; i < \_currentLevel.Width; ++i)

{

for (int j = 0; j < \_currentLevel.Height; ++j)

{

int? dataIndex = j \* currentLayer.Width + i;

if (dataIndex == null) continue;

var currentTileId = currentLayer.Data[dataIndex.Value] - 1;

if (currentTileId == null) continue;

var currentTile = \_tileIdMap[currentTileId.Value];

var tileWidth = currentTile.ImageWidth ?? 0;

var tileHeight = currentTile.ImageHeight ?? 0;

var sourceRect = new Rectangle<int>(0, 0, tileWidth, tileHeight);

var destRect = new Rectangle<int>(i \* tileWidth, j \* tileHeight, tileWidth, tileHeight);

\_renderer.RenderTexture(currentTile.TextureId, sourceRect, destRect);

}

}

}

}

public IEnumerable<RenderableGameObject> GetRenderables()

{

foreach (var gameObject in \_gameObjects.Values)

{

if (gameObject is RenderableGameObject renderableGameObject)

{

yield return renderableGameObject;

}

}

}

public (int X, int Y) GetPlayerPosition() => \_player!.Position;

public void AddBomb(int X, int Y, bool translateCoordinates = true)

{

var worldCoords = translateCoordinates ? \_renderer.ToWorldCoordinates(X, Y) : new Vector2D<int>(X, Y);

SpriteSheet spriteSheet = SpriteSheet.Load(\_renderer, "BombExploding.json", "Assets");

spriteSheet.ActivateAnimation("Explode");

TemporaryGameObject bomb = new(spriteSheet, 2.1, (worldCoords.X, worldCoords.Y));

\_gameObjects.Add(bomb.Id, bomb);

}

}

// File: PlayerObject.cs

using Silk.NET.Maths;

namespace TheAdventure.Models;

public class PlayerObject : RenderableGameObject

{

private const int \_speed = 128;

public int Lives { get; private set; } = 3;

public void LoseLife()

{

Lives--;

if (Lives > 0)

{

Console.WriteLine($"You have {Lives} lives left.");

}

if (Lives <= 0)

{

GameOver();

}

}

public enum PlayerStateDirection

{

None = 0, Down, Up, Left, Right,

}

public enum PlayerState

{

None = 0, Idle, Move, Attack, GameOver

}

public (PlayerState State, PlayerStateDirection Direction) State { get; private set; }

public PlayerObject(SpriteSheet spriteSheet, int x, int y) : base(spriteSheet, (x, y))

{

SetState(PlayerState.Idle, PlayerStateDirection.Down);

}

public void SetState(PlayerState state) => SetState(state, State.Direction);

public void SetState(PlayerState state, PlayerStateDirection direction)

{

if (State.State == PlayerState.GameOver || (State.State == state && State.Direction == direction))

return;

if (state == PlayerState.None && direction == PlayerStateDirection.None)

SpriteSheet.ActivateAnimation(null);

else if (state == PlayerState.GameOver)

SpriteSheet.ActivateAnimation(Enum.GetName(state));

else

SpriteSheet.ActivateAnimation($"{Enum.GetName(state)}{Enum.GetName(direction)}");

State = (state, direction);

}

public void GameOver() => SetState(PlayerState.GameOver, PlayerStateDirection.None);

public void Attack()

{

if (State.State != PlayerState.GameOver)

SetState(PlayerState.Attack, State.Direction);

}

public void UpdatePosition(double up, double down, double left, double right, int width, int height, double time)

{

if (State.State == PlayerState.GameOver) return;

var pixelsToMove = \_speed \* (time / 1000.0);

var x = Position.X + (int)(right \* pixelsToMove) - (int)(left \* pixelsToMove);

var y = Position.Y + (int)(down \* pixelsToMove) - (int)(up \* pixelsToMove);

var newState = (x == Position.X && y == Position.Y) ?

(State.State == PlayerState.Attack && SpriteSheet.AnimationFinished ? PlayerState.Idle : PlayerState.Idle)

: PlayerState.Move;

var newDirection = State.Direction;

if (y < Position.Y) newDirection = PlayerStateDirection.Up;

else if (y > Position.Y) newDirection = PlayerStateDirection.Down;

else if (x < Position.X) newDirection = PlayerStateDirection.Left;

else if (x > Position.X) newDirection = PlayerStateDirection.Right;

if (newState != State.State || newDirection != State.Direction)

SetState(newState, newDirection);

Position = (x, y);

}

}

using Silk.NET.Maths;

using Silk.NET.SDL;

using SixLabors.ImageSharp;

using SixLabors.ImageSharp.PixelFormats;

using TheAdventure.Models;

using Point = Silk.NET.SDL.Point;

namespace TheAdventure;

public unsafe class GameRenderer

{

private Sdl \_sdl;

private Renderer\* \_renderer;

private GameWindow \_window;

private Camera \_camera;

private Dictionary<int, IntPtr> \_texturePointers = new();

private Dictionary<int, TextureData> \_textureData = new();

private int \_textureId;

public GameRenderer(Sdl sdl, GameWindow window)

{

\_sdl = sdl;

\_renderer = (Renderer\*)window.CreateRenderer();

\_sdl.SetRenderDrawBlendMode(\_renderer, BlendMode.Blend);

\_window = window;

var windowSize = window.Size;

\_camera = new Camera(windowSize.Width, windowSize.Height);

}

public void SetWorldBounds(Rectangle<int> bounds)

{

\_camera.SetWorldBounds(bounds);

}

public void CameraLookAt(int x, int y)

{

\_camera.LookAt(x, y);

}

public int LoadTexture(string fileName, out TextureData textureInfo)

{

using (var fStream = new FileStream(fileName, FileMode.Open))

{

var image = Image.Load<Rgba32>(fStream);

textureInfo = new TextureData()

{

Width = image.Width,

Height = image.Height

};

var imageRAWData = new byte[textureInfo.Width \* textureInfo.Height \* 4];

image.CopyPixelDataTo(imageRAWData.AsSpan());

fixed (byte\* data = imageRAWData)

{

var imageSurface = \_sdl.CreateRGBSurfaceWithFormatFrom(data, textureInfo.Width,

textureInfo.Height, 8, textureInfo.Width \* 4, (uint)PixelFormatEnum.Rgba32);

if (imageSurface == null)

throw new Exception("Failed to create surface from image data.");

var imageTexture = \_sdl.CreateTextureFromSurface(\_renderer, imageSurface);

\_sdl.FreeSurface(imageSurface);

if (imageTexture == null)

throw new Exception("Failed to create texture from surface.");

int currentId = \_textureId++;

\_textureData[currentId] = textureInfo;

\_texturePointers[currentId] = (IntPtr)imageTexture;

return currentId;

}

}

}

public void RenderTexture(int textureId, Rectangle<int> src, Rectangle<int> dst,

RendererFlip flip = RendererFlip.None, double angle = 0.0, Point center = default)

{

if (\_texturePointers.TryGetValue(textureId, out var imageTexture))

{

var translatedDst = \_camera.ToScreenCoordinates(dst);

\_sdl.RenderCopyEx(\_renderer, (Texture\*)imageTexture, in src,

in translatedDst,

angle,

in center, flip);

}

}

public Vector2D<int> ToWorldCoordinates(int x, int y)

{

return \_camera.ToWorldCoordinates(new Vector2D<int>(x, y));

}

public void SetDrawColor(byte r, byte g, byte b, byte a)

{

\_sdl.SetRenderDrawColor(\_renderer, r, g, b, a);

}

public void ClearScreen()

{

\_sdl.RenderClear(\_renderer);

}

public void PresentFrame()

{

\_sdl.RenderPresent(\_renderer);

}

public (int Width, int Height) GetWindowSize()

{

return \_window.Size;

}

}