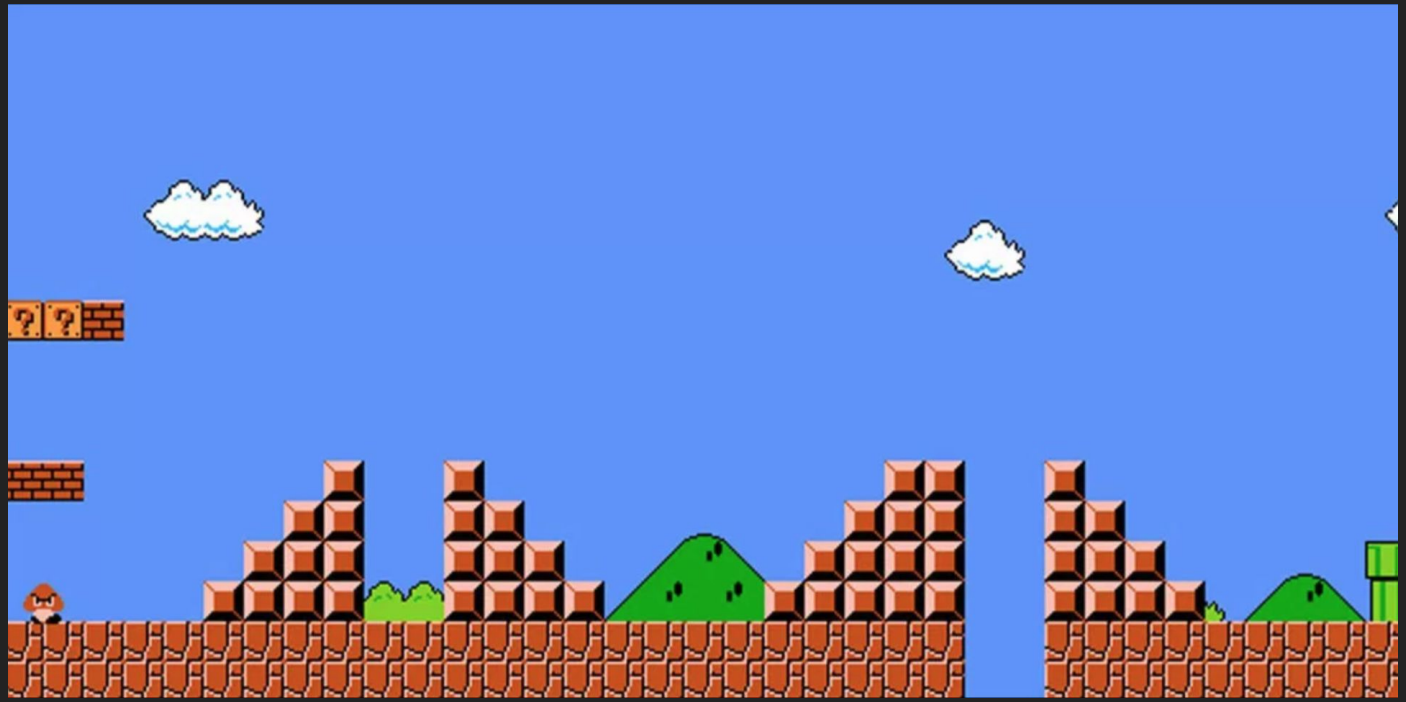
Platformer Levels

(Tilesets and Tilemaps)

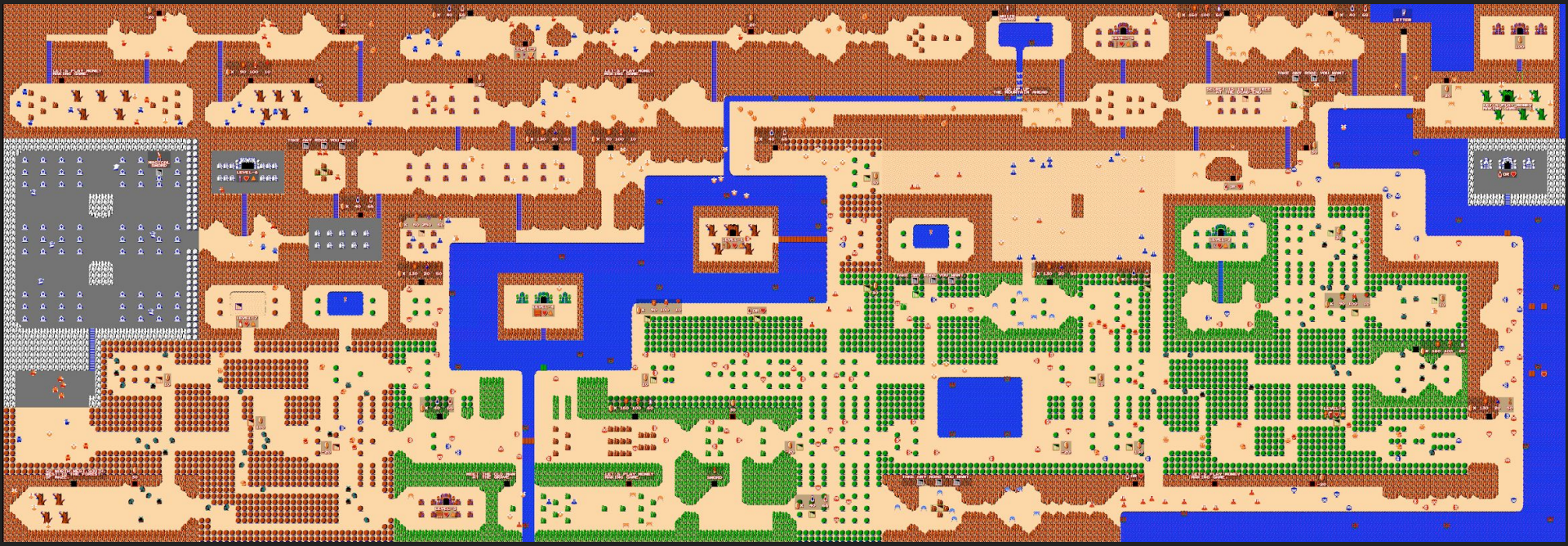


Super Mario Bros. (NES)

The Legend of Zelda (NES)

A Link to the Past (SNES)

Spelunky




WELCOME TO WARP ZONE!
4
3
2

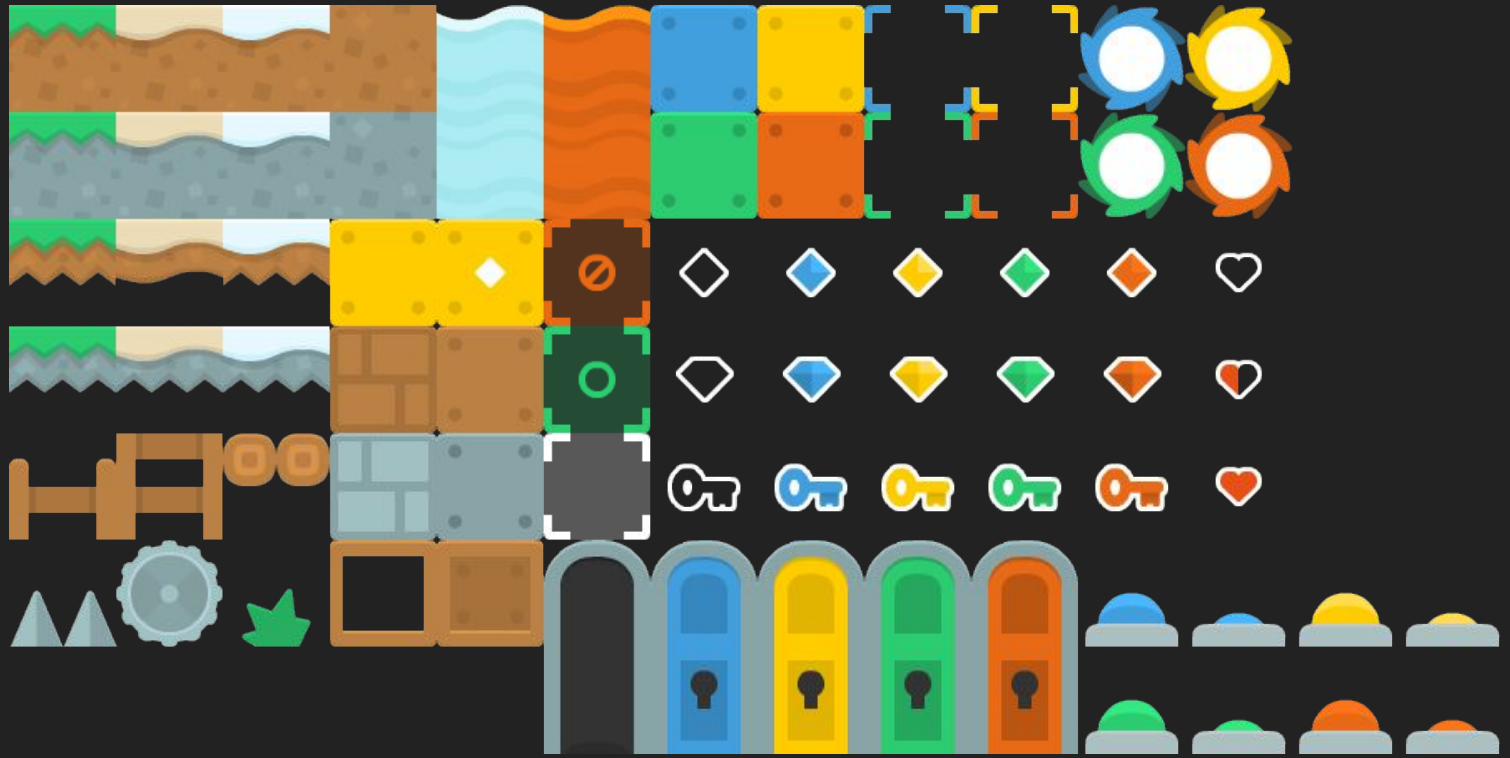
Tileset

(It’s just a texture/image)

Tileset

Downloaded from:
http://www.supermariobrosx.org/forums/viewtopic.php?f=49&t=4572

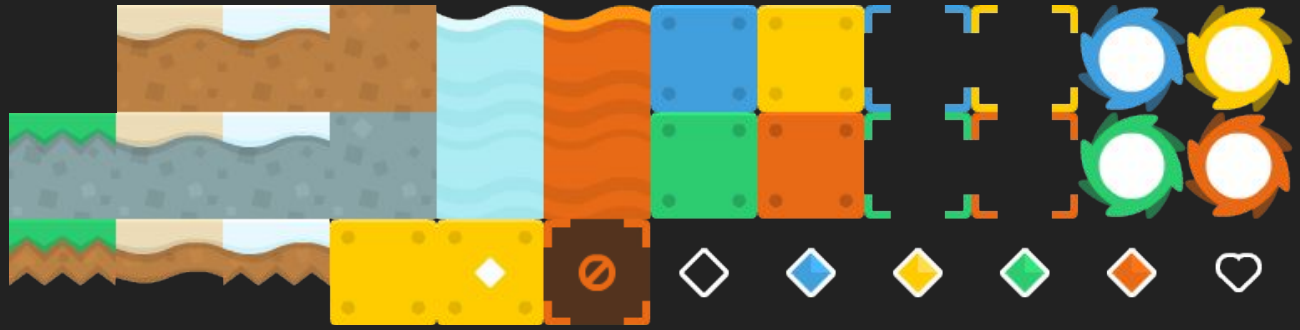
Tileset



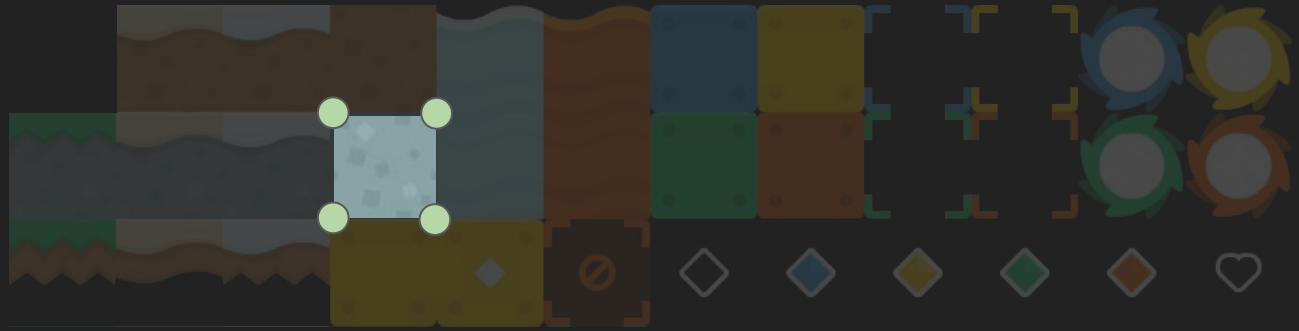
Drawing a single tile is like drawing a single sprite.

We need the UV coordinates of the individual sprite.

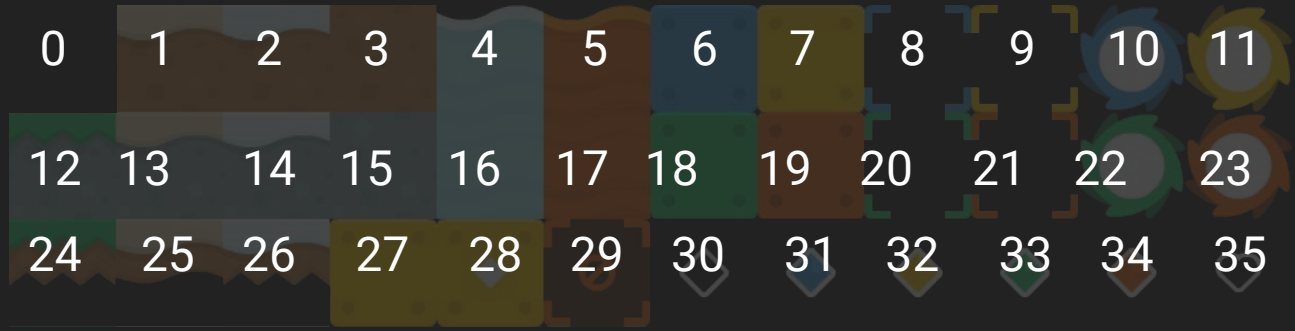
Tileset



Tileset

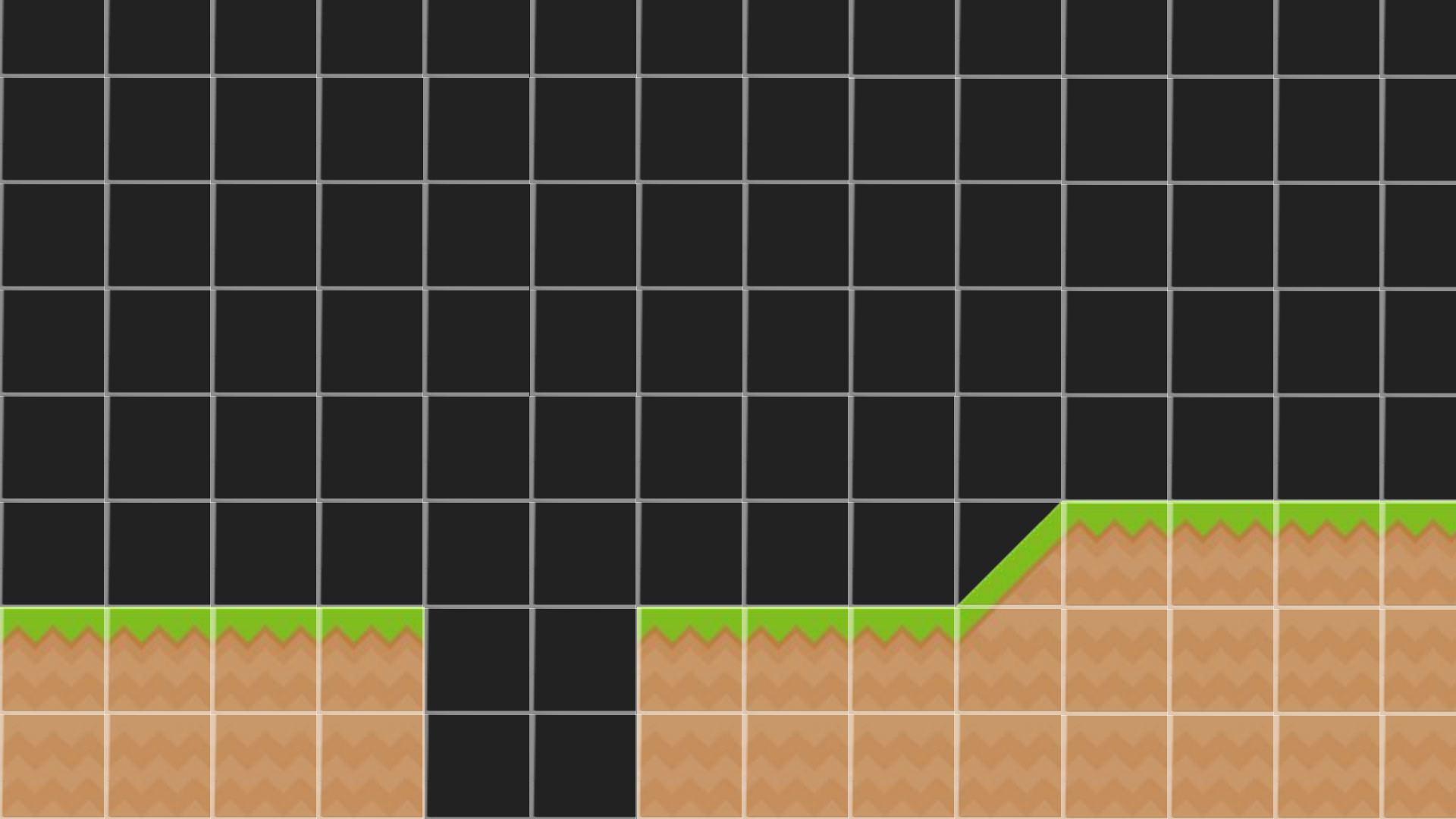


Tileset


0 1 2 3 4 5 6 7 8 9 10 11
12 13 14 15 16 17 18 19 20 21 22 23
24 25 26 27 28 29 30 31 32 33 34 35

Tilemap

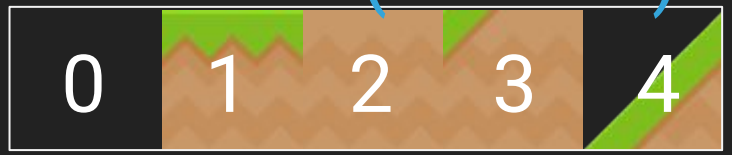
(It’s just an array)



0	0	0	0	0	0	0	0	0	0	0	0	0	0
0	0	0	0	0	0	0	0	0	4	1	1	1	1
1	1	1	1	0	0	1	1	1	3	2	2	2	2
2	2	2	2	0	0	2	2	2	2	2	2	2	2

Tilemap (array)

Tileset (texture)



```
unsigned int levelData[] =
{
    0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0,
    0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0,
    0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0,
    0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0,
    0, 0, 0, 0, 0, 0, 0, 0, 0, 4, 1, 1, 1, 1,
    1, 1, 1, 1, 0, 0, 1, 1, 1, 3, 2, 2, 2, 2,
    2, 2, 2, 2, 0, 0, 2, 2, 2, 2, 2, 2, 2, 2
};
```

Building and Rendering

We build a list of vertices and texture coordinates then draw the level as one object.

Drawing Tilemaps is Similar to

Drawing Monospaced Fonts

For each character in a string

- Draw 2 Triangles
- Use UV coordinates for character

Top of Map.h

```
#pragma once
#define GL_SILENCE_DEPRECATION
#ifdef _WINDOWS
#include <GL/glew.h>
#endif

#define GL_GLEXT_PROTOTYPES 1
#include <vector>
#include <math.h>
#include <SDL.h>
#include <SDL_opengl.h>
#include <SDL_image.h>
#include "glm/mat4x4.hpp"
#include "glm/gtc/matrix_transform.hpp"
#include "ShaderProgram.h"
```

The rest of Map.h

```
class Map {
    int width;
    int height;
    unsigned int *levelData;

    GLuint textureID;
    float tile_size;
    int tile_count_x;
    int tile_count_y;

    std::vector<float> vertices;
    std::vector<float> texCoords;

    float left_bound, right_bound, top_bound, bottom_bound;

public:
    Map(int width, int height, unsigned int *levelData, GLuint textureID, float tile_size, int
tile_count_x, int tile_count_y);
    void Build();
    void Render(ShaderProgram *program);
};
```

Map.cpp

```
#include "Map.h"

Map::Map(int width, int height, unsigned int *levelData, GLuint textureID, float tile_size, int
tile_count_x, int tile_count_y)
{
    this->width = width;
    this->height = height;
    this->levelData = levelData;
    this->textureID = textureID;
    this->tile_size = tile_size;
    this->tile_count_x = tile_count_x;
    this->tile_count_y = tile_count_y;

    this->Build();
}
```

Map.cpp (Build)

```
void Map::Build()
{
    for(int y = 0; y < this->height; y++) {
        for(int x = 0; x < this->width; x++) {

            int tile = levelData[y * width + x];
            if (tile == 0) continue;

            float u = (float)(tile % tile_count_x) / (float)tile_count_x;
            float v = (float)(tile / tile_count_x) / (float)tile_count_y;

            float tileWidth = 1.0f/(float)tile_count_x;
            float tileHeight = 1.0f/(float)tile_count_y;

            float xoffset = -(tile_size / 2); // From center of tile
            float yoffset = (tile_size / 2); // From center of tile

            vertices.insert(vertices.end(), {
                    xoffset + (tile_size * x),              yoffset + -tile_size * y,
                    xoffset + (tile_size * x),              yoffset + (-tile_size * y) - tile_size,
                    xoffset + (tile_size * x) + tile_size,  yoffset + (-tile_size * y) - tile_size,

                    xoffset + (tile_size * x),              yoffset + -tile_size * y,
                    xoffset + (tile_size * x) + tile_size,  yoffset + (-tile_size * y) - tile_size,
                    xoffset + (tile_size * x) + tile_size,  yoffset + -tile_size * y
                });
```

Map.cpp (Build Continued)

```
            texCoords.insert(texCoords.end(), {
                    u, v,
                    u, v+(tileHeight),
                    u+tileWidth, v+(tileHeight),

                    u, v,
                    u+tileWidth, v+(tileHeight),
                    u+tileWidth, v
                });
        }
    }

    left_bound = 0 - (tile_size / 2);
    right_bound = (tile_size * width) - (tile_size / 2);

    top_bound = 0 + (tile_size / 2);
    bottom_bound = -(tile_size * height) + (tile_size / 2);
}
```

Map.cpp (Render)

```
void Map::Render(ShaderProgram *program)
{
    glm::mat4 modelMatrix = glm::mat4(1.0f);
    program->SetModelMatrix(modelMatrix);

    glUseProgram(program->programID);

    glVertexAttribPointer(program->positionAttribute, 2, GL_FLOAT, false, 0, vertices.data());
    glEnableVertexAttribArray(program->positionAttribute);

    glVertexAttribPointer(program->texCoordAttribute, 2, GL_FLOAT, false, 0, texCoords.data());
    glEnableVertexAttribArray(program->texCoordAttribute);

    glBindTexture(GL_TEXTURE_2D, textureID);
    glDrawArrays(GL_TRIANGLES, 0, (int)vertices.size() / 2);

    glDisableVertexAttribArray(program->positionAttribute);
    glDisableVertexAttribArray(program->texCoordAttribute);
}
```

Updates to main.cpp

```
// Add to GameState (remove platforms)
Map *map;

// Add to top of file
#define LEVEL1_WIDTH 14
#define LEVEL1_HEIGHT 5

unsigned int level1_data[] =
    {
        0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0,
        0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0,
        0, 0, 0, 0, 0, 0, 0, 0, 0, 1, 1, 1, 1, 1,
        1, 1, 1, 1, 0, 0, 1, 1, 1, 2, 2, 2, 2, 2,
        2, 2, 2, 2, 0, 0, 2, 2, 2, 2, 2, 2, 2, 2
    };
```

Updates to main.cpp

```
// Inside of Initialize
GLuint mapTextureID = LoadTexture("tileset.png");
state.map = new Map(LEVEL1_WIDTH, LEVEL1_HEIGHT, level1_data, mapTextureID, 1.0f, 4, 1);

// Inside of Update
We need to pass state.map to Update methods.


// Inside of Render
state.map->Render(&program);
```

Let’s Code!

Add tileset.png
Add Map.h and Map.cpp
Update main.cpp

We need to
check for collisions
with our tilemap.

Convert our entity (Player/AI) position to the grid coordinates of the tilemap and check if a tile is there.

Be sure not to try to read something larger than the array size of your map (or negative).

You may want 0 and other numbers to be “not solid”.

Map.cpp (IsSolid)

```
bool Map::IsSolid(glm::vec3 position, float *penetration_x, float *penetration_y)
{
    *penetration_x = 0;
    *penetration_y = 0;

    if (position.x < left_bound || position.x > right_bound) return false;
    if (position.y > top_bound || position.y < bottom_bound) return false;

    int tile_x = floor((position.x + (tile_size / 2)) / tile_size);
    int tile_y = -(ceil(position.y - (tile_size / 2))) / tile_size; // Our array counts up as Y goes down.

    if (tile_x < 0 || tile_x >= width) return false;
    if (tile_y < 0 || tile_y >= height) return false;

    int tile = levelData[tile_y * width + tile_x];
    if (tile == 0) return false;

    float tile_center_x = (tile_x * tile_size);
    float tile_center_y = -(tile_y * tile_size);

    *penetration_x = (tile_size / 2) - fabs(position.x - tile_center_x);
    *penetration_y = (tile_size / 2) - fabs(position.y - tile_center_y);

    return true;
}
```

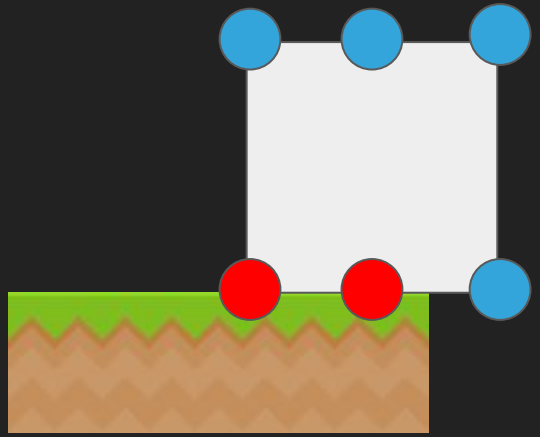
Entity.h

```
// At the top
#include "Map.h"

// In the middle
void CheckCollisionsX(Map *map);
void CheckCollisionsY(Map *map);
```

Entity.cpp

```
void Entity::Update(float deltaTime, Entity *objects, int objectCount, Map *map)
{
    collidedTop = false;
    collidedBottom = false;
    collidedLeft = false;
    collidedRight = false;

    velocity += acceleration * deltaTime;

    position.y += velocity.y * deltaTime;        // Move on Y
    CheckCollisionsY(map);
    CheckCollisionsY(objects, objectCount);    // Fix if needed

    position.x += velocity.x * deltaTime;        // Move on X
    CheckCollisionsX(map);
    CheckCollisionsX(objects, objectCount);    // Fix if needed
}
```

CheckCollisionsY



Checks for partially on ledge.

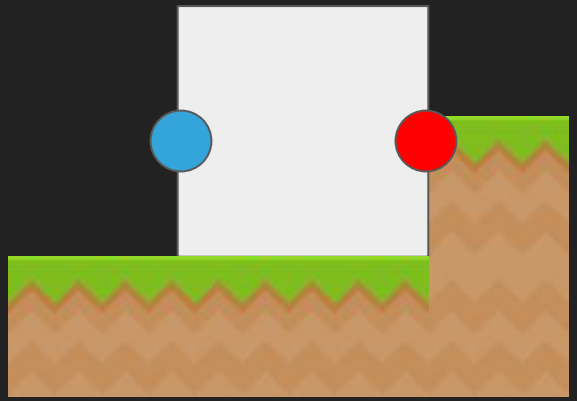
Entity.cpp (CheckCollisionsY)

```
void Entity::CheckCollisionsY(Map *map)
{
    // Probes for tiles
    glm::vec3 top = glm::vec3(position.x, position.y + (height / 2), position.z);
    glm::vec3 top_left = glm::vec3(position.x - (width / 2), position.y + (height / 2), position.z);
    glm::vec3 top_right = glm::vec3(position.x + (width / 2), position.y + (height / 2), position.z);

    glm::vec3 bottom = glm::vec3(position.x, position.y - (height / 2), position.z);
    glm::vec3 bottom_left = glm::vec3(position.x - (width / 2), position.y - (height / 2), position.z);
    glm::vec3 bottom_right = glm::vec3(position.x + (width / 2), position.y - (height / 2), position.z);

    float penetration_x = 0;
    float penetration_y = 0;

    if (map->IsSolid(top, &penetration_x, &penetration_y) && velocity.y > 0) {
        position.y -= penetration_y;
        velocity.y = 0;
        collidedTop = true;
    }
    else if (map->IsSolid(top_left, &penetration_x, &penetration_y) && velocity.y > 0) {
        position.y -= penetration_y;
        velocity.y = 0;
        collidedTop = true;
    }
    else if (map->IsSolid(top_right, &penetration_x, &penetration_y) && velocity.y > 0) {
        position.y -= penetration_y;
        velocity.y = 0;
        collidedTop = true;
    }
```

Entity.cpp (CheckCollisionsY Continued)

```
    if (map->IsSolid(bottom, &penetration_x, &penetration_y) && velocity.y < 0) {
        position.y += penetration_y;
        velocity.y = 0;
        collidedBottom = true;
    }
    else if (map->IsSolid(bottom_left, &penetration_x, &penetration_y) && velocity.y < 0) {
        position.y += penetration_y;
        velocity.y = 0;
        collidedBottom = true;
    }
    else if (map->IsSolid(bottom_right, &penetration_x, &penetration_y) && velocity.y < 0) {
        position.y += penetration_y;
        velocity.y = 0;
        collidedBottom = true;
    }
}
```

CheckCollisionsX



Checks for side collisions.

Entity.cpp (CheckCollisionsX)

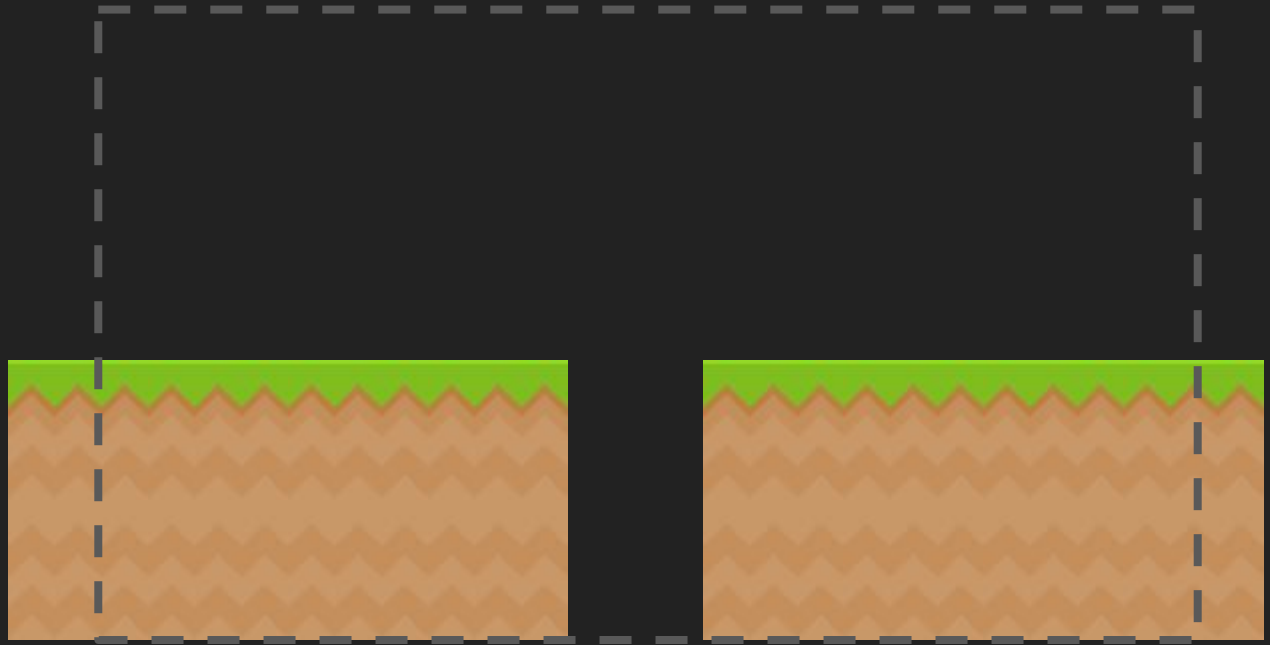
```
void Entity::CheckCollisionsX(Map *map)
{
    // Probes for tiles
    glm::vec3 left = glm::vec3(position.x - (width / 2), position.y, position.z);
    glm::vec3 right = glm::vec3(position.x + (width / 2), position.y, position.z);

    float penetration_x = 0;
    float penetration_y = 0;

    if (map->IsSolid(left, &penetration_x, &penetration_y) && velocity.x < 0) {
        position.x += penetration_x;
        velocity.x = 0;
        collidedLeft = true;
    }

    if (map->IsSolid(right, &penetration_x, &penetration_y) && velocity.x > 0) {
        position.x -= penetration_x;
        velocity.x = 0;
        collidedRight = true;
    }
}
```

Let’s Code!

Collisions with the TileMap

We need to scroll
as the player moves.

We are translating everything so the player is at the center of the screen.



We are moving translating things into view.

The View Matrix

```
viewMatrix = glm::mat4(1.0f);

program.SetViewMatrix(viewMatrix);
```

We can translate the view matrix at the end of update and set it at the start of render.

```
// End of Update
viewMatrix = glm::mat4(1.0f);
viewMatrix = glm::translate(viewMatrix,
          glm::vec3(-state.player->position.x, 0, 0));


// Start of Render
program.SetViewMatrix(viewMatrix);
```

Let’s Code!

Sidescrolling!

Let’s Get Organized!

We are going to need to
load textures and display text
in different places,
let’s build a utility class.

Util.h

```
#pragma once
#define GL_SILENCE_DEPRECATION

#ifdef _WINDOWS
#include <GL/glew.h>
#endif

#define GL_GLEXT_PROTOTYPES 1
#include <vector>
#include <SDL.h>
#include <SDL_opengl.h>
#include "glm/mat4x4.hpp"
#include "glm/gtc/matrix_transform.hpp"
#include "ShaderProgram.h"

class Util {
public:
    static GLuint LoadTexture(const char* filePath);
    static void DrawText(ShaderProgram *program, GLuint fontTextureID, std::string
text, float size, float spacing, glm::vec3 position);
};
```

Util.cpp

```
#include "Util.h"

#define STB_IMAGE_IMPLEMENTATION
#include <SDL_image.h>
#include "stb_image.h"

GLuint Util::LoadTexture(const char* filePath) {
    int w, h, n;
    unsigned char* image = stbi_load(filePath, &w, &h, &n, STBI_rgb_alpha);

    if (image == NULL) {
        std::cout << "Unable to load image. Make sure the path is correct\n";
        assert(false);
    }

    GLuint textureID;
    glGenTextures(1, &textureID);
    glBindTexture(GL_TEXTURE_2D, textureID);
    glTexImage2D(GL_TEXTURE_2D, 0, GL_RGBA, w, h, 0, GL_RGBA, GL_UNSIGNED_BYTE, image);

    glTexParameteri(GL_TEXTURE_2D, GL_TEXTURE_MIN_FILTER, GL_NEAREST);
    glTexParameteri(GL_TEXTURE_2D, GL_TEXTURE_MAG_FILTER, GL_NEAREST);

    stbi_image_free(image);
    return textureID;
}
```

Util.cpp (DrawText)

```
void Util::DrawText(ShaderProgram *program, GLuint fontTexture, std::string text, float size,
float spacing, glm::vec3 position) {
    float width = 1.0f / 16.0f;
    float height = 1.0 / 16.0f;

    std::vector<float> vertices;
    std::vector<float> texCoords;

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

        int index = (int)text[i];

        float u = (float)(index % 16) / 16.0f;
        float v = (float)(index / 16) / 16.0f;

        float offset = (size + spacing) * i;

        texCoords.insert(texCoords.end(), {
            u, v,
            u, v + height,
            u + width, v,
            u + width, v + height,
            u + width, v,
            u, v + height,
        });
```

Util.cpp (DrawText - Continued)

```
            vertices.insert(vertices.end(), {
                offset + (-0.5f * size), 0.5f * size,
                offset + (-0.5f * size), -0.5f * size,
                offset + (0.5f * size), 0.5f * size,
                offset + (0.5f * size), -0.5f * size,
                offset + (0.5f * size), 0.5f * size,
                offset + (-0.5f * size), -0.5f * size,
            });
        }

        glm::mat4 modelMatrix = glm::mat4(1.0f);
        modelMatrix = glm::translate(modelMatrix, position);
        program->SetModelMatrix(modelMatrix);

        glUseProgram(program->programID);

        glVertexAttribPointer(program->positionAttribute, 2, GL_FLOAT, false, 0, vertices.data());
        glEnableVertexAttribArray(program->positionAttribute);

        glVertexAttribPointer(program->texCoordAttribute, 2, GL_FLOAT, false, 0, texCoords.data());
        glEnableVertexAttribArray(program->texCoordAttribute);

        glBindTexture(GL_TEXTURE_2D, fontTexture);
        glDrawArrays(GL_TRIANGLES, 0, (int)(text.size() * 6));

        glDisableVertexAttribArray(program->positionAttribute);
        glDisableVertexAttribArray(program->texCoordAttribute);
    }
```

Let’s add Util.h to main.cpp
We need to remove the following from main.cpp, Entity.h and Map.h

```
#define STB_IMAGE_IMPLEMENTATION
#include <SDL_image.h>
#include "stb_image.h"
```

Now we can update main.cpp to use our new Util class.

```
// Add variable on top
GLuint fontTextureID;

// Inside Initialize
fontTextureID = Util::LoadTexture("font.png");
GLuint mapTextureID = Util::LoadTexture("tileset.png");
state.player.textureID = Util::LoadTexture("ctg.png");

// Inside Render
Util::DrawText(&program, fontTextureID, "Hello!", 1.0f, -0.5f,
                                                 glm::vec3(0, 0, 0));
```

Let’s Code!

Let’s update the map, player start position and view matrix.

Add a wall.
Start into the level.
Stop scrolling if too far to the left.

Updated Map

```
#define LEVEL1_WIDTH 14
#define LEVEL1_HEIGHT 8

unsigned int level1_data[] =
{
    3, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0,
    3, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0,
    3, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0,
    3, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0,
    3, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0,
    3, 0, 0, 0, 0, 0, 0, 0, 0, 1, 1, 1, 1, 1,
    3, 1, 1, 1, 1, 1, 1, 1, 1, 2, 2, 2, 2, 2,
    3, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2
};
```

New Player Start Position

```
// Inside Initialize
state.player->position = glm::vec3(5, 0, 0);


// Inside Update
viewMatrix = glm::translate(viewMatrix,
             glm::vec3(-state.player.position.x, 3.75, 0));
```

Look at how the view behaves when moving to the left.

How can we fix it?

Updated View Behavior

```
// Inside Update
viewMatrix = glm::mat4(1.0f);
if (state.player->position.x > 5) {
    viewMatrix = glm::translate(viewMatrix,
         glm::vec3(-state.player->position.x, 3.75, 0));
} else {
    viewMatrix = glm::translate(viewMatrix, glm::vec3(-5, 3.75, 0));
}
```

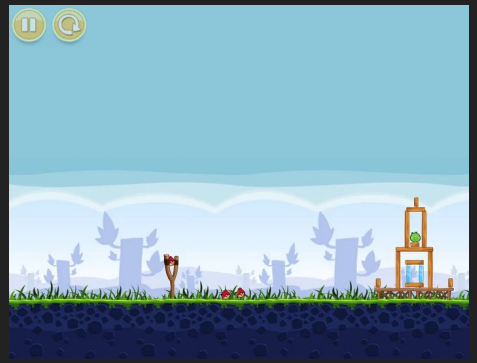
Review:
Game Mode

Main Menu

Chapter Select

Level Select

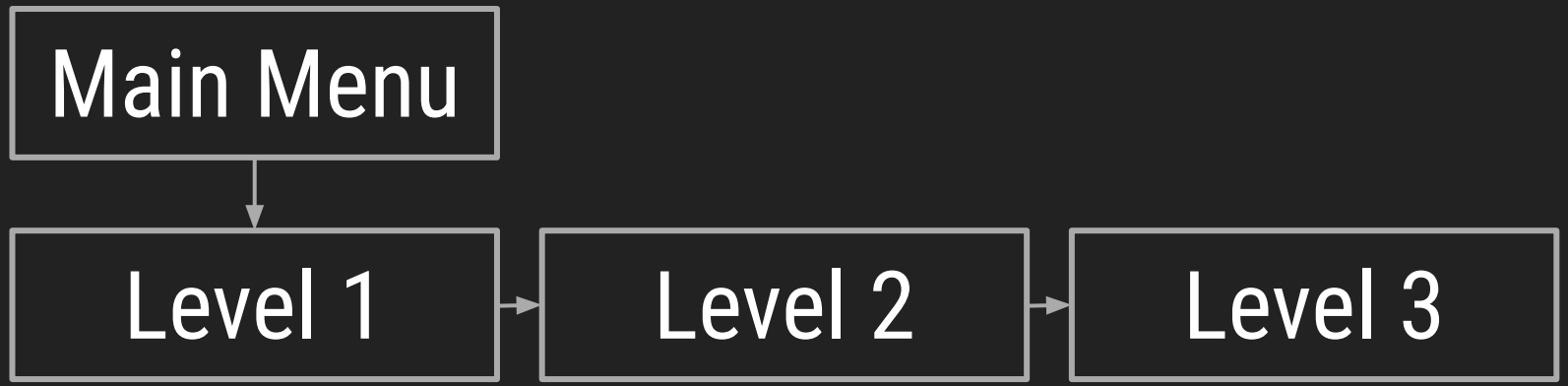
Cut Scene



Game Level

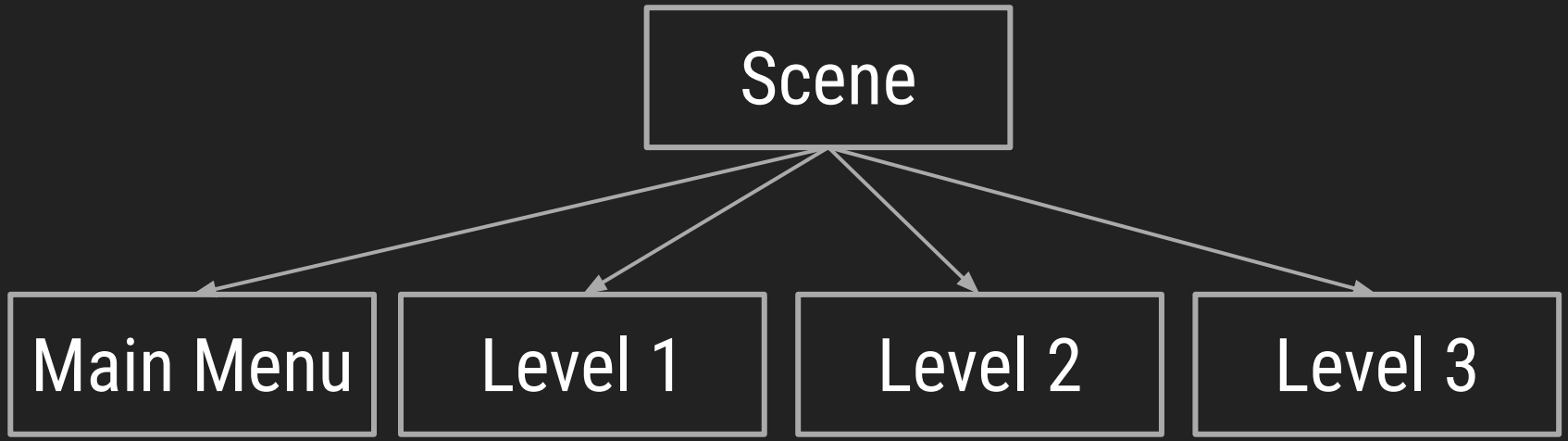
Win Screen

Platformer Project



You can just display text for “You Lose” and “You Win”.

We'll make a "Scene" Class and Derive, MainMenu, Level1, Level2, Level3.



This will allow us to:

Have a list of Scenes.

Set a pointer to Current Scene.

Simply call functions such as:

```
currentScene->Initialize();
currentScene->Update();
currentScene->Render();
```

Scene.h

```
#pragma once
#define GL_SILENCE_DEPRECATION

#ifdef _WINDOWS
#include <GL/glew.h>
#endif
#define GL_GLEXT_PROTOTYPES 1
#include <SDL.h>
#include <SDL_opengl.h>
#include "glm/mat4x4.hpp"
#include "glm/gtc/matrix_transform.hpp"
#include "ShaderProgram.h"

#include "Util.h"
#include "Entity.h"
#include "Map.h"
```

Scene.h (continued)

```
struct GameState {
    Map *map;
    Entity *player;
    Entity *enemies;
};

class Scene {
public:
    GameState state;
    virtual void Initialize() = 0;
    virtual void Update(float deltaTime) = 0;
    virtual void Render(ShaderProgram *program) = 0;
};
```

Scene.cpp

```
#include "Scene.h"
```

(All functions are virtual, that’s it!)

Let’s make a Scene!

Scene.h
Scene.cpp

Let’s make a Level and move code out of main.cpp

Level1.h
Level1.cpp

Level1.h

```
#include "Scene.h"

class Level1 : public Scene {

public:
    void Initialize() override;
    void Update(float deltaTime) override;
    void Render(ShaderProgram *program) override;
};
```

Level1.cpp

```
#include "Level1.h"

#define LEVEL1_WIDTH 14
#define LEVEL1_HEIGHT 8

unsigned int level1_data[] =
{
    3, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0,
    3, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0,
    3, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0,
    3, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0,
    3, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0,
    3, 0, 0, 0, 0, 0, 0, 0, 0, 1, 1, 1, 1, 1,
    3, 1, 1, 1, 1, 1, 1, 1, 1, 2, 2, 2, 2, 2,
    3, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2
};
```

Level1.cpp

```
void Level1::Initialize() {
    GLuint mapTextureID = Util::LoadTexture("tileset.png");
    state.map = new Map(LEVEL1_WIDTH, LEVEL1_HEIGHT, level1_data, mapTextureID, 1.0f, 4, 1);

    // Move over all of the player and enemy code from initialization.

}

void Level1::Update(float deltaTime) {
    state.player->Update(deltaTime, state.player, state.enemies, ENEMY_COUNT, state.map);
}

void Level1::Render(ShaderProgram *program) {
    state.map->Render(program);
    state.player->Render(program);
}
```

There are a lot of little updates to main.cpp

```
// Add headers
#include "Scene.h"
#include "Level1.h"

// Add some variables and SwitchToScene function
Scene *currentScene;
Level1 *level1;

void SwitchToScene(Scene *scene) {
    currentScene = scene;
    currentScene->Initialize();
}

// Clear stuff out of initialize and add this to the bottom:
level1 = new Level1();
SwitchToScene(level1);
```

There are a lot of little updates to main.cpp

Update ProcessInput, add currentScene-> wherever it says state.

Update Update to call update from the currentScene:
currentScene->Update(FIXED_TIMESTEP);

Add currentScene-> wherever it says state.

Update Render
currentScene->Render(&program);

Checkpoint:
Let’s make sure we can
compile and we have
the same thing we had before.

Let’s talk about having multiple scenes and switching scenes.

Let’s make another Level!

Level2.h
Level2.cpp

We can copy code from Level1 and make some updates.

Change some of the tilemap!

We need a way to switch scenes.

Update main.cpp

```
// Add include for Level2
#include "Level1.h"
#include "Level2.h"

// Remove Level1 *level1 and add an array of scenes
Scene *currentScene;
Scene *sceneList[2];

// Initialize the levels and start at the first one
sceneList[0] = new Level1();
sceneList[1] = new Level2();
SwitchToScene(sceneList[0]);
```

Update Scene.h

```
// Add include for nextScene
struct GameState {
	Map *map;
	Entity *player;
	Entity *enemies;
	int nextScene;
};
```

Update Level1.cpp and Level2.cpp

```
// In Initialize set nextScene to -1, don’t forget to change in both!

state.nextScene = -1;
```

Update main.cpp

```
// Inside of our main loop, we can check if nextScene has changed.

if (currentScene->state.nextScene >= 0) SwitchToScene(sceneList[currentScene->state.nextScene]);
```

Let’s Code!