Tutorial 9 Level Maps

Before starting this tutorial, you must complete the last one and have it working. Failure to do so will result in not being able to complete these steps. It is your choice but is recommended to crate a new branch each tutorial.

For this tutorial we will be implementing a level map. This level map will allow us to control player collisions with the map. Whilst similar to a traditional tiling implementation, our approach will differ slightly. Normally each tile would represent a different sprite and draw that particular sprite in the corresponding location. For simplicity sake we will be using our map only for collisions and using a pre-prepared full screen background image.

Create a LevelMap class and complete the header with the following:

```
#ifndef _LEVELMAP_H
#define _LEVELMAP_H

#include "constants.h"

class LevelMap
{

public:
	LevelMap(int map[MAP_HEIGHT][MAP_WIDTH]);
	~LevelMap();

	int GetTileAt(unsigned int h, unsigned int w);

private:
	int** m_map;

};

#endif
```

The implementation is a little tricky. You will also need to define MAP_HEIGHT and MAP_WIDTH in your constants.h. width should be the value 16, and height the value 13. These values represent how many tiles across and down there are on the map.

Notice how m_map is a pointer to a pointer. Don't worry too much about this. This is simply how you dynamically create arrays in code, you may remember this being mentioned in semester 1. This means we can change the map width and height and the code will still be able to handle it.

Below I have supplied a lot of code snippets to help you create this file. It is important that the level map is put together correctly otherwise you will have a multitude of issues that can stop the compiling of the program.

1. In the constructor we need to copy over data passed

```
/*
 * when making a map, remember:
 * 0 = empty space,  1 = blocked/occupied space
 */

LevelMap::LevelMap(int map[MAP_HEIGHT][MAP_WIDTH])
{
	//Allocate memory for the level map
	m_map = new int* [MAP_HEIGHT];
	for(unsigned int i = 0; i < MAP_HEIGHT; i++)
	{
		m_map[i] = new int[MAP_WIDTH];
	}

	//populate the array
	for(unsigned int i=0; i<MAP_HEIGHT; i++)
	{
		for(unsigned int j = 0; j< MAP_WIDTH; j++)
		{
			m_map[i][j] = map[i][j];
		}
	}
}
```

You should notice the use of "unsigned" keyword. Unsigned integers are integers that can only hold non-negative whole numbers.

2. In the destructor we need to tidy up the used memory:

```
//delete all elements of the array
for(unsigned int i =0; i<MAP_HEIGHT;i++)
{
	delete[] m_map[i];
}
delete[]m_map;
```

3. Then populate the method to retrieve this data:

```
if(h < MAP_HEIGHT && w < MAP_WIDTH)
{
	return m_map[h][w];
}

return 0;
```

Ensure you understand each of these methods before continuing. We will be using this class in the next section. Build your application (F6) and you should have no errors.

In your GameScreenLevel1.h file, we are going to add a new method and variable to be able to use this LevelMap class. This will allow us to use the LevelMap data for Level 1 to control our characters movement. Add the following:

1. A new private method:
   ```
   void SetLevelMap();
   ```
2. A new private variable:
   ```
   LevelMap* m_level_map;
   ```
   Don't forget your includes.
3. You then will need the set the variable above to nullptr in the GameScreenLevel1 constructor.

Now you should implement the new SetLevelMap() method in your GameScreenLevel1.cpp file. Ensure you understand this method as you implement it. You should then call SetLevelMap() in your set up code for GameScreenLevel1, this will be the same place where you initialise your character. Ensure this is copied correctly as mistyping this map leads to common errors:

```
int map[MAP_HEIGHT][MAP_WIDTH] = {  { 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 },
                                    { 1,1,1,1,1,1,0,0,0,0,1,1,1,1,1,1 },
                                    { 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 },
                                    { 0,0,0,0,1,1,1,1,1,1,1,1,0,0,0,0 },
                                    { 1,1,0,0,0,0,0,0,0,0,0,0,0,0,1,1 },
                                    { 0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0 },
                                    { 0,0,0,0,0,0,0,1,1,0,0,0,0,0,0,0 },
                                    { 1,1,1,1,1,1,0,0,0,0,1,1,1,1,1,1 },
                                    { 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 },
                                    { 1,1,1,1,1,1,1,1,1,1,1,1,1,1,1,1 } };

//clear any old maps
if(m_level_map != nullptr)
{
	delete m_level_map;
}

//set the new one
m_level_map = new LevelMap(map);
```

When you look at the 0 and 1 values in the collision map above, you will see that there are a couple of 1s 5 rows up from the bottom, in the centre. These are for the POW block that we will add in a

future tutorial. For now, you can either leave them as 1s meaning that your character will collided with that area of space, or set them to 0s and allow your character to pass through that area.

Next, we need to edit our character class, so it knows about this new LevelMap. Alter the constructor of your Character class to also accept a LevelMap pointer as a parameter.

```
public:
	Character(SDL_Renderer* renderer, string imagePath, Vector2D start_position, LevelMap* map);
```

You will also need to adjust the Character classes you have made to take in the map too, adjust the parameters in the header and the cpp files for them.

```
CharacterMario::CharacterMario(SDL_Renderer* renderer, string imagePath, Vector2D start_position, LevelMap* map) : Character(renderer, imagePath, start_position, map)
```

1. Now create a new private variable to store this LevelMap pointer inside the Character header:

   ```
   LevelMap* m_current_level_map;
   ```

2. Then, set the pointer passed into the constructor to this new m_current_level_map pointer, this should be done in the constructor in Character.cpp.

   ```
   m_current_level_map = map;
   ```

3. You will also have to pass the Map through to this new Character constructor from your GameScreenLevel1 set up code. It is probably showing as an error at the moment.

   ```
   SetLevelMap();

   //set up player character
   mario = new CharacterMario(m_renderer, "Images/Mario.png", Vector2D(64, 330), m_level_map);
   ```

With this in place, we should now be able to get our character to collide with the floors but for this you need to have implemented gravity as instructed in a previous tutorial.

For those that haven't set up gravity and jumping, return to additional work from tutorial 7. You should have something like this:

```
void Character::AddGravity(float deltaTime)
{
	if (m_position.y + 64 <= SCREEN_HEIGHT)
	{
		m_position.y += GRAVITY * deltaTime;
	}
	else
	{
		m_can_jump = true;
	}
}

void Character::Jump()
{
	if(!m_jumping)
	{
		m_jump_force = INITIAL_JUMP_FORCE;
		m_jumping = true;
		m_can_jump = false;
	}
}
```

In the Character cpp file, locate the Update function and adapt it as below. This states if gravity should be applied or not:

1. We first find the tile that our character is on. TILE_WIDTH and TILE_HEIGHT is the number of pixels of the tile. Both should be set to 32 in the Constants.h file.

```
//collision position variables
int centralX_position = (int)(m_position.x + (m_texture->GetWidth() * 0.5) )/
TILE_WIDTH;
int foot_position = (int)(m_position.y + m_texture->GetHeight()) / TILE_HEIGHT;
```

2. Then we check if this is an empty tile. If it is then we add gravity, otherwise we have collided and we allow the character to jump again. Ensure you only have one call to AddGravity, remove the existing call as it will give unexpected behaviour.

```
//deal with gravity
if(m_current_level_map->GetTileAt(foot_position,centralX_position) == 0)
{
	AddGravity(deltaTime);
}
else
{
	//collided with ground so we can jump again
	m_can_jump = true;
}
```

With this in place, run your application. You should have a character that falls down the screen and hits the floor!

Additional Work

As stated earlier there are collisions happening where the POW block will go. To be able to set this area to be a non-collision space after the POW block has been destroyed, we need a function in the LevelMap that will allow us to change any value on the map to something else.

This is the prototype you will require for your LevelMap header file. It should have public access so that other classes, such as the Character or GameScreenLevel1, can access it.

```
void ChangeTileAt(unsigned int row, unsigned int column, unsigned int new_value);
```

It is up to you to write the code required to implement this in the LevelMap.cpp file. We be using this in the next tutorial.