Game Engines

MGAE-E2013

2D Platformer Engine

IT-University of Copenhagen

Jacob Claudius Grooss, jcgr@itu.dk

May 22, 2013

1 Introducton

This report has been written as part of a project on the Game Engines E2013 (MGAE-E2013) course on the Games course at the IT-University of Copenhagen.

For this assignment I have created a game engine for a 2D platformer game, written in C++ with the use of the SDL framework[1]. I have worked, and shared ideas, with Jakob Melnyk (jmel).

2 Features

The engine is intended to support simple 2D platform games built in tiles, where jumping and avoiding enemies while trying to reach a goal area are the key points. The engine contains the following features:

- A player that has the ability to walk and jump.
- Enemies that fly or walk. In the case of the walking enemy, it is also affected by gravity.
- Collision detection, both between entities (player/enemies) and the walls, and between the player and the enemies.
- A side-scrolling camera that follows the player.
- Animations for entities.

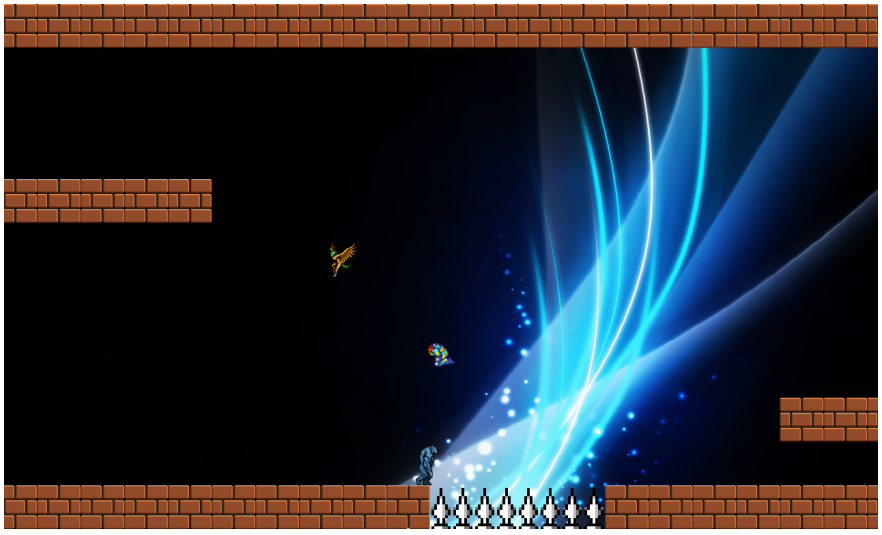

Figure 1: A picture of the game in progress.

3 Overall Structure

In this section, I will go over the structure of the engine, which consists of the following classes: A map, a player, enemies, a level, a window to display everything to and a class for shared functions.

[1] http://www.libsdl.org/

Map

The **map** class is a representation of the world the player is traversing. It contains information about the layout of the world, such as where solid blocks and spikes are found, where the goal is, where the player spawns and where enemies spawn. The information about the world is loaded from a .txt file (see section 6.1 for an example and explanation of the data format).

Player

The person playing the game needs to be able to interact with it, which is the purpose of the **player** class. It represents the player character, which can move around the map and collide with other objects. Collision results in different events depending on what is hit (stopping if a wall is hit, dying if something hostile is collided with, etc.).

Enemies

Obstacles in a 2D platformer is important and some kind of enemies are usually used for this purpose. In the engine, enemies are represented by the **enemy** class, which is a very basic enemy. It contains very basic behaviour, but it can be extended by other classes to create specific enemies with their own behaviour. This makes it very easy to implement new enemies.

Level

The **level** class represents an entire level in the engine. It contains a map, a list of the enemies in the map and the player character.
The level handles input from the keyboard tells the player to move depending on the input. The level also updates and moves the enemies according to their behaviour. It also takes care of telling the window class where to draw everything.

Window

The **window** class is the one that handles any form for drawing to the screen. It initializes SDL, creates a window for it to draw on and helps with loading and drawing of images in an easy way. It is used behind the scenes.

HelperClass

As there are some things that are used by multiple classes (for example the amount of collision points for entities), having such information gathered in one place makes it easier to keep track of. That is the purpose of the **helperclass** class. It contains values and functions used by multiple other classes, so they do not have to be defined a lot of different places.

These classes are the main ones that make up the engine. Used together they can produce a simple 2D platformer game, in which the player's objective is to get from one point to another.

4 Improvements

Currently, the engine works to a certain degree. It supports simple 2D platformer games, where the player has to dodge enemies and deadly obstacles in order to reach a goal. But that is pretty much all it supports. There is no fighting, interacting with objects, acquiring new abilities or anything along those lines. Implementing such things would be one of the main focuses if improvements were to be made.

When it comes to implementing new things, enemies work fairly well. One can extend the **Enemy** class to create new kinds of enemies with different behaviours, but loading them into a level from a map file is not easy. The reason is that the map uses a very basic data format, which is not easy to extend. Therefore, changing the data format of the map files would be something to look into.

Introducing an actual camera class would be a welcome change as well. In the current implementation, there is no actual camera. Things are just drawn to the screen in relation to where the player character currently is. A separate camera would be easier for the user of the engine to work with, as the user would have more control over how it works.

Another thing that could be improved upon is the performance on fast vs. slow computers. As it is, there is nothing that takes framerate into account, and the engine will therefore run slower on older/slower computers, which worsens the experience for people with slow computers.

These are just a few of the possible improvements. Each of them would take some work to get done, but each would make the engine a lot better.

5 Conclusion

The engine I have written allows for users to create a very simple 2D platformer game. The engine is fairly simple to use, but there is very little customization, The games that can be created with it revolves around the player navigating his/her was from the start to the goal, while avoiding enemies and lethal obstacles.

6 Appendix

6.1 Map data file and layout

The map uses the following data format: The first line determines the amount of rows in the map, the second determines the amount of columns. The lines after represents the layout of the map, with the numbers meaning the following things:

- **0**: An empty space.
- **1**: A solid block.
- **2 & 3**: 2 and 3 both represent the goal, but there is a difference: 2 determines the place where the goal texture is drawn and is used for checking for collision with the goal, where 3 is only used for checking for collision with the goal.
  This two-part design is used because drawing the goal texture multiple places looks strange, but collision might be needed for multiple tiles. 3 was introduced to handle that.
- **4**: Spikes.
- **5**: The spawn point of one kind of enemy.
- **6**: The spawn point of another kind of enemy.
- **9**: The spawn point the player.

```
1  12
2  40
3  1111111111111111111111111111111111111111
4  1000000000000000000000000000000000000001
5  1000006000000000000000000000000000000001
6  1000000000000000000000000000000000000001
7  1111111110000000000000000000000000000001
8  1000000000000000000000000000000000000001
9  1000000000005000000000000000000000000001
10 1000000000000000000000000000000000000001
11 1000000000000000000000000000000000000001
12 1000000000000000000001111000000000300001
13 1090000000000000000000000000000000200001
14 1111111111111144441111111111111111111111
```

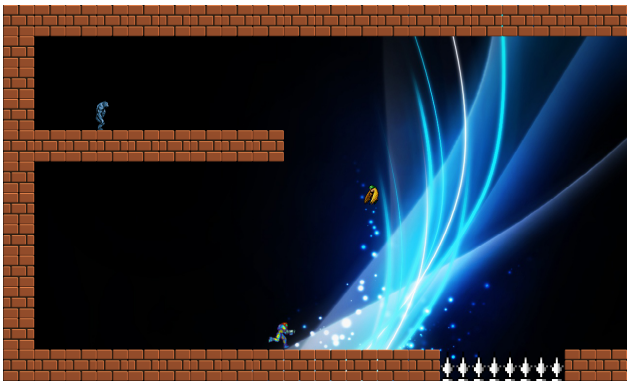

Figure 2: A picture of the game in progress.

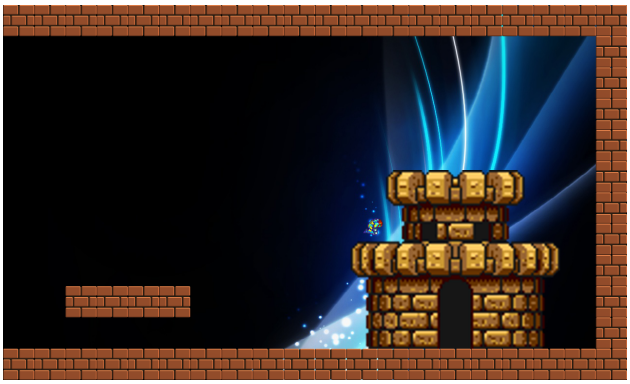

Figure 3: The player about to get to the goal.

Figure 4: The screen shown after completing a level.

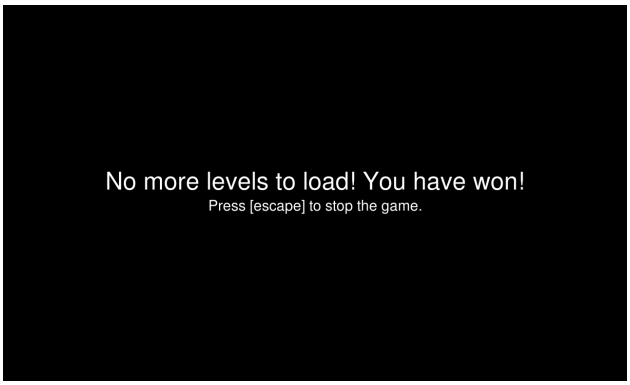

Figure 5: The screen shown when all the levels have been completed.