Game Engines E2013

2D Platformer Engine

Jacob Grooss (jcgr@itu.dk)

Overview

For this assignment I have created a game engine for a 2D platformer game. I have spoken and shared ideas with Jakob Melnyk and helped him with some problems over the course of the assignment.

My engine has been written in C++ with the use of the SDL framework. The engine is intended to support simple platform games, 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.

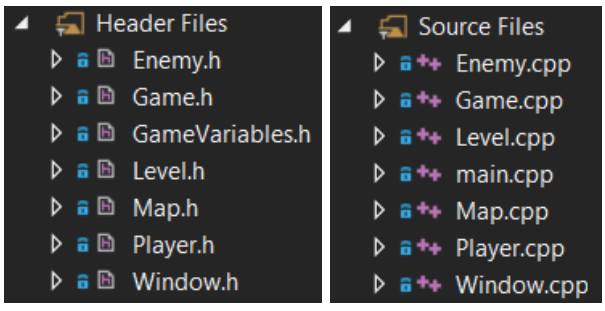
The Engine

I will explain the overall structure of the engine, the design and then cover some of the things that could have been improved upon, along with how they could have been improved.

The structure

This section will be used for explaining the overall structure of the engine.

The engine consists of seven header (.h) files and seven c++ (.cpp) files as seen in the pictures to the right. Most of these files are related, with the exceptions of GameVariables.h and main.cpp.



The GameVariables.h file acts as a config file as it contains a lot of static values that are used by the engine. Therefore it does not need a .cpp file. The main.cpp file is simply used for testing the engine and has no need for a header file.

As stated above, the main.cpp is used for testing the engine. It creates a new Game object which is then initialized. Initializing the Game object creates a Window object of a set height and width, which is used to draw the game upon.

The game is then told to start playing, which creates a new Level object and tells it to load a level with a name format of “Level0.txt”, where the 0 can be replaced with the level the player has gotten to. Every time a level is completed, the level number will be increased and a new level will be loaded. When there are no more levels, the game is over and will show the end screen.

The Level class is the one that actually contains game logic. It creates a new player, it loads the map from the given file and creates the enemies for the level. When the initial load has finished, the main loop will start running until either the level is completed or the player closes the game.

The first thing the main loop does, is check for events. For example input from the person playing the game. Based on the input, the window will be closed, the player will be moved or nothing will happen.

After checking for input, the loop updates the map (if anything needs updating), the player and the enemies. This updating involves switching to the right textures based on the state of objects, moving the entities that needs moving and checking for collision.

After updating the objects, the entire game is drawn (background, the map, the player and the enemies). This entire loop keeps running until the player quits the game or finished the level.

When the player reaches the end of the level, another screen will be drawn. It tells the player that he has completed the level. After proceeding, the Game class will attempt to find the next level, or tell the player that he has won the game.

Design

I will, in this section, go over the design of the classes. I will go through the classes in the order they are used by the engine. I will skip the main.cpp file as it is only used to start the engine for testing purposes.

Game

The Game class was not designed to do much. Its main purpose is to initialize a window for the engine to use, make sure to load the correct level and tell the player the game is over when there are no more levels.

Window

The Window class represents the screen that the engine draws everything on. The source code for it has been found at https://github.com/Twinklebear/TwinklebearDev-Lessons/tree/master/Lesson8 and has only been modified slightly (to make it easier to change the size of the window).

The Window class was designed to take of everything with regards to drawing. It creates a screen, a renderer that is used to draw things to the screen and it takes care of drawing the textures it is given. This allows for an easy way to visualize the game, as the engine should not have to do anything but initialize the window and use its draw function.

Level

The Level class was designed to take care of most of the game logic. It accepts input from the one playing, it updates both the map (if anything needs updating) along with the entities (the player and

the enemies), moves whatever needs moving (along with doing collision detection) and draws the game screen.

Map

The Map class represents the map of the level. It loads a file by first reading the amount of rows and columns the map should have, and then it reads the rows which are represented by numbers from 0 to 9. The only other thing the map class can do, is to figure out where the player is supposed to spawn in the level.

Player

The Player class represents the player. It contains the player's position and velocity and the textures used to draw the player. As moving is done by the Level class, the Player class has very little logic in it, mainly used for updating the player's speed and killing and respawning the player.

Enemy

The Enemy class represents the enemies in the game. It has the position, velocity and textures of the enemies, but, like the Player class, does very little. It can update the speed and texture of the enemy, but it lets Level handle the movement of the enemies.

Improvements

Currently, the engine is pretty closed off, which is something I will address in this section. The only things that can be easily changed are the levels and the textures of everything. The levels are defined in separate .txt files and the paths to the different textures are stored in the GameVariables.h file. There are plenty of things that could be done to make it more user/friendly, and I will go over a few of them here.

The first thing to improve would be to make the classes easier to work with. As it is, the Level class takes care of a lot of important things, such as moving entities and drawing everything. Instead of doing that, it would make more sense to give each entity its own function for moving, and just have the Level class call that function when necessary. The same goes for drawing.

Another thing to improve would be how enemies work. Currently, the Enemy class contains two hard-coded enemies. Because of this, the Enemy class would have to be edited to allow for more enemies. Instead, having an abstract Enemy class that could be inherited from to create new enemies would be a better idea.

The level/loading could also do with an update. As it is, it is very simple and does not support more than ten different things as it uses the numbers from 0 to 9 to determine what each part of the map file represents.

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.