Project Report - First assignment - GAME ENGINES

Andrea Distler

28 November, 2012

Contents

1 Introduction

This is the project report for the first assignment of the Game Engines course.

A short description of the source code and the used libraries can be found in section 2. The files are available on GitHub. The libraries are also made available on GitHub for easier installation.

Section 3 describes the outcome of the project.

Section4 gives an overview of the used mechanics and points out the key features of the project.

Section 5 gives a detailed description of the important classes and their relations to each other. It also points out challenging parts of the code.

I have invested an unnecessarily big amount of time into this small assignment, and I want to elaborate on my workflow in section 6.

2 The Files

All files can be found on GitHub.
`https://github.com/JungleJinn/GE---SimplePlatformerEngine`.

2.1 Used Libraries

The project was written in C++, using additional libraries:

- Included in the repository
  - SDL image
- External
  - openGL. Installed manually. Files in zip
  - glm. Installed manually. Files in zip
  - SDL 1.2.15. `http://www.libsdl.org/download-1.2.php`
    Used environment variable named SDK_SDL

3 Outcome

The project's result is a little game in which the player can control a bomb. The player cannot directly control the bomb's position. However, it is possible to change the bomb's mass and therefore indirectly possible to steer the bomb. The goal of the game is to survive as long as possible, which is made hard by other bombs which present themselves as obstacles. Depending on the player's mass, the enemy bomb will affect the player's further movement. If the player moves backwards, the game is over. For a screenshot of the game see figure 1.

There is a short explanation of the controls and rules of the game in the beginning. Once a player loses, there is also a gameover screen, telling the player that the game is now over.

The assignment was to create a simple physics engine which can be used for a platformer. The game features:

- 2D physics
- openGL rendering
- an intro screen and an outro screen
- a dragon with a hat
- sprites and texture atlases
- collision detection and response
- bounciness
- very intuitive controls

4 Overview

This section gives an overview over used mechanics.

4.1 Game Loop

Listing 1: The game loop

```
lastUpdateTime = clock();
Init();

quit = false;
while (!quit)
{
  Play();
}
```

First the game is being set up, then the game loop starts. The game loop is quite basic. While the game has not been quit, it is playing.

4.2 Messaging

The Input is realized by using SDL events. The first thing done in the Play method is to fetch all pending input events. These events are then sorted by what is important, what is to be used in the game, and what not. Then the newly written message system starts to create and forward messages.

Listing 2: Messaging

```
SDL_Event sdlEvent = SDL_Event();
while (SDL_PollEvent(&sdlEvent))
{
  // user closes window
  if (sdlEvent.type == SDL_QUIT)
  {
    quit = true;
    break;
  }
  else if (sdlEvent.type == SDL_KEYDOWN)
  {
    messenger->SendMessage(InputMessage(InputMessage::KEY_DOWN
        , sdlEvent.key.keysym.sym));
  }
  else if (sdlEvent.type == SDL_KEYUP)
  {
    messenger->SendMessage(InputMessage(InputMessage::KEY_UP,
        sdlEvent.key.keysym.sym));
  }
```

```
18 }
```

The messenger can send messages of different types to receivers of type IMessageReceiver. All objects of this type automatically register and unregister at the messenger at creation and deletion. The receivers decide which kinds of messages they want to process, but each receiver gets all sent messages.

4.3 Physics

The physics are calculated in two dimensions. The physics system is responsible for movement and for collisions as well as for collision response. Bodies are implemented as simple rigid bodies. Forces can be applied. There are two basic collision shapes: Circles and Rectangles. The collision detection happens in two states, first the broad phase, which only checks the radius of the objects, and the narrow phase, in which the collision normal is calculated. The normal is then used to determine the resulting forces acting on both colliders. Bodies have a coefficient of restitution which adds to the collision response. The combination method is to use the average coefficient of restitution of both bodies.

For integrating forces, the integration method receives a value named deltaTime. The delta time is calculated like in the following listing:

Listing 3: deltaTime

```
1 int now = clock();
2 deltaTime = (float)((now - lastUpdateTime) / CLOCKS_PER_SEC) /
      timeScale;
3 lastUpdateTime = now;
```

The whole game execution can be slowed or fastened by setting different timeScale values. At a value of 1.0 it is executed in real-time, and other values stretch or squish the perceived time.

4.4 Sprites

Sprites are used for the player and for the enemy bombs. They are stored in a spritesheet and their texture coordinates are adapted to fit certain sprite frames.

4.5 Camera

The camera is moving with the player all the time. A problem encountered with a potentially endless level was whether the floor and ceiling should always be recreated. The player can move as far as he wants to, therefore an inifinite amount of rectangular walls would be needed. The solution which avoids creating endless amounts of floor tiles was to always move the floor and ceiling with the player (not unlike the camera). The texture coordinates change to create the visual feedback of the player's velocity. The illusion that the player moves in relation to the floor is created, whereas the player and the floor have a relative velocity of zero. The background also follows the player.

4.6 Dragon

In the first phase of the game the player can choose the launch angle by rotating a dragon's head.

5 Important Classes

In figure 2 a selection of classes is depicted.

Game. The game class manages the game states and

6 Workflow

Listings

List of Figures

Figure 1: A screenshot of the game.

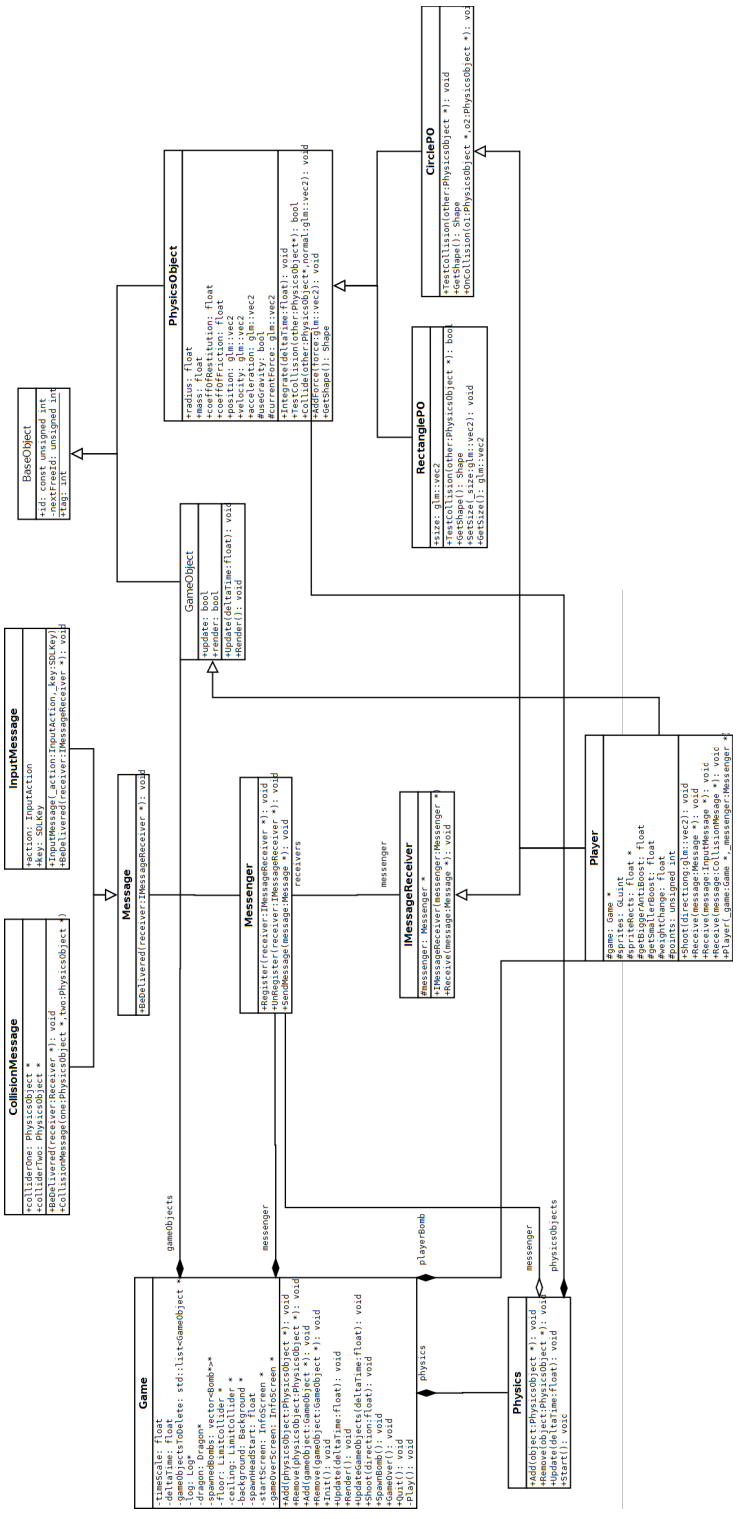

Figure 2: A sketch of the most important classes.