2D-platformer

Game Engines E2013

Jakob Melnyk, jmel@itu.dk

September 27, 2013

1 Introduction

This report is the result of a project to implement a simple 2D-platformer game engine for the Game Engines, E2013 course at IT-University of Copenhagen.

The required features of the engine were:

- Side scrolling
- 2D platformer physics (walking, jumping, falling)
- Collision detection (e.g. between player and enemy)
- Sprite animations based on state (walking vs. standing)
- Architecture based on game loop and events

In addition to the required features of the engine, I have implemented animation of walking for the player and enemies.

My implementation supports a game consisting of a single level. The y-position of the camera is fixed, so while the engine supports "tall" levels, the camera will not show the player or anything outside the "standard" level height.
The engine supports 1 texture for "solid" game tiles, 1 texture for goal tile(s) and up to 6 different types of enemies (two example enemies are included).

The engine uses a variable time step, so the speed of the game can vary depending on hardware. I discuss this in more detail in section 3.1 on page 2.

The game engine is implemented in C++ for Windows using the SDL 2.0 framework[1].

I have worked with Jacob Grooss (jcgr@itu.dk) on parts of this project (mainly the design, but also parts of the implementation).

2 How To Use

Most of the changeable values for the game are placed in the *EngineSettings* header. These settings include game name, window size, controls, player/tile textures names, player speed, game gravity and the maximum possible number of enemies in the game[2].

2.1 Making levels

Currently the engine loads the "Level0.txt" file in the "levels" folder as the level for the game. First two lines describes the height and length of the level, respectively. The rest of the document describes the layout of the level.
The "levels" folder included with the example game contains an example of the formatting for the level files.

[1] http://www.libsdl.org/hg.php - SDL_image and SDL_ttf libraries were also used.
[2] These settings could be defined in a text file and loaded at runtime instead of being defined at compile time.

2.2 Adding new enemies

Adding additional enemies to the game (engine) requires inheriting from the *Enemy* class and overriding the constructor, *updateTexture* and *move* functions. In addition, the *loadEnemies()* function in *game* must have an if-clause added to include the new enemy type. *enemyTypes.h* is used to define the different values for the enemy types[3]. The engine currently supports up to 6 different enemy types.
My example game contains two enemies. A Demon that runs horizontally in the level and a Copter that flies vertically in the level.

3 Overview of the Engine

The general architecture of the engine is based on the *Game* class. It contains the game loop, a list of enemies, the player and the level. The *Game* class is also responsible for figuring out what to draw on the screen.
The *Map* class is responsible for keeping track of the map layout and loading the map. The *Enemy* and *Player* classes are responsible for keeping track of their own position and animation as well as moving their position.
The *Window*[4] class is responsible for actually creating a window and drawing the textures.

3.1 Variable time step

I have not implemented a fixed time step in my engine. This means the game speed heavily depends on the performance of the hardware the engine is run on. Very limited processing power would significantly slow down the games made on the engine and provide a very different experience compared to using the engine on a powerful machine. The problem with a fixed time step is that on a slow computer, the fixed time step could give the experience of "frame skips", where parts of the game is skipped.
Implementing a fixed time step would require keeping track of a global variable to keep track of time since last frame and then multiplying the speed of the game characters by this value for every game loop. If a game loop finishes before the time to next frame is finished, then just wait until this time has passed.

3.2 Extending the engine

In this section I discuss ways to expand the functionality of the engine.

More levels To support more than one level, the *Game* class would need to be changed. Instead of quitting the game on completing a level, a different map should be loaded, new memory allocated for the list of monsters and a new player object created.

Controls Changing the control scheme is simply changing the values in the *EngineSettings* header. Expanding the control scheme would require a change to the main game loop, so that the new controls would be registered.

[3] This could be moved to the *enemy* header file instead
[4] Source: https://github.com/Twinklebear/TwinklebearDev-Lessons/blob/master/Lesson8/src/window.cpp

More tile/enemy types The current map format only supports 10 tile types. To allow for more varied levels, the format would need to be changed to support more integer. A possible format could be "01,01,02" to indicate [solidTile], [solidTile], [goalTile] instead of just 112.

Events Implementing custom events to the engine could be done by expanding the number of *SDL_Event*s in the *HelperClass* header and then expanding the main game loop to check for these custom events.

Collectibles If the map format was expanded, the game could support collectible items (e.g. power-ups) by having the collectibles be a tile type and throwing a custom event upon player collision with the tile.

4 Conclusion

The engine successfully implements the required features of the project. There is support for side-scrolling, standard 2D-platformer physics collision detection, sprite animation and is based on a game loop and events (for win/loss conditions).

The main problems of the game engine is the lack of a fixed time step and the support of only a single level. Additionally, the game engine uses a very poor map loading algorithm and level file format making for very limited variety in levels and enemies.

4.1 Future work

As I mentioned in section 3.2, there are a few ways to improve/extend the game engine. In this section I discuss the two I would focus on if I were to improve the engine.

Fixed time step As discussed in section 3.1, there is some merit to using a fixed time step in a 2D-platformer game. Implementing a fixed time step would require a rewrite of the movement and updating of textures. If I were to implement it, I would implement it in such a way, that there would be support for choosing either a fixed or a variable time step.

More levels Changing the engine to support more than one level would make it easier for game developers to create an actual full-length game. Currently this can be done by making different game clients for each level, but this is not very desirable.

Better level format Using the current format to describe a level, there can only be 10 different values. Implementing a better map loading algorithm and a better level file format would be obvious points to improve. This would allow for more varied levels.

5 Appendix

5.1 Screenshots

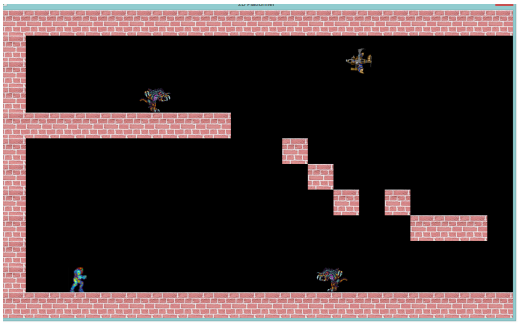

Figure 1: Screenshot of the example game: Starting position

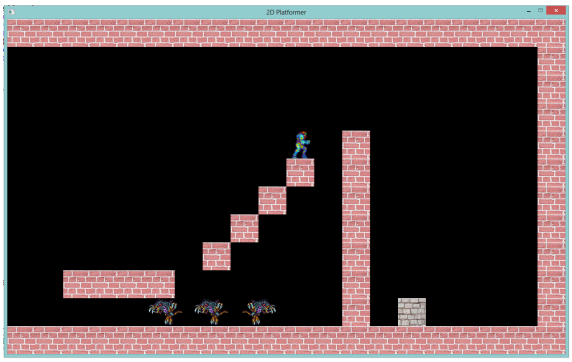

Figure 2: Screenshot of the example game: Close to the end of the level.

5.2 Video

I have included a video of the demo game in action with the report.