Björn Þór Jónsson bjrr@itu.dk

Game Engines

Project 1: Platformer game engine

For the first programming assignment in Game Engines I've implemented a platformer engine in JavaScript and used the Canvas element in HTML5.

The main components of the engine are four pseudo-classes / functions (JavaScript only has the notion of functions): Player, Platform, GameLoop and GameOver. Those classes and supporting variables / data are encapsulated in another function called Plafgine (plat-former-engine) for closure.

At the top of plafgine.js are a few configuration variables to define the level, which could be factored into another file:

- defaultPlayerPosition: Defines the size of the main player and where it is positioned initially.
- enemyPositions: Positions for the enemies and their sizes.
- platformDefinitions: Placement of the platforms.
- defaultEnemyAutomation: This is maybe an interesting experiment using the dynamic nature of JavaScript to have the behaviour of enemies configurable by plugging in a function that implements their movement.

There are no real physics in this platformer and it rather implements pseudo-physics by starting a jump at a fixed speed and then decreasing it on each game loop, until the jump speed reaches zero, then a fall speed is incremented until collision with a platform or the ground happens. Those are the jump, checkJump, checkFall, fallStop functions within the player object.

Different instances of the same implementation of the Player object are used for both the main player and the enemies (NPC), with their configurations differing in setting whether they are automated and then with the added behaviour-function mentioned above. Some sort of class inheritance could have been a good idea here.

The collision detection is as inefficient as can be where player collisions are checked against all platforms and all other players on each game loop. Spatial segmentation of what to check against would of course be better for any reasonably sized level.

Control of the main character is handled by registering pressed keys into a state variable

and then reading those states on each game loop (in the player implementation) and moving the character accordingly.

Camera movement is implemented by keeping the main character still and moving all other game world objects in the opposite direction when the character has reached a certain threshold on the screen. That threshold is in fact centered on the screen so the character is pretty much always in the horizontal center. There are glitches in the implementation of this camera movement that can almost always be reproduced when jumping from the highest platform; then the player doesn't fall completely to the ground, but that can be fixed by jumping again! - the cause of this should be investigated in another iteration.

Timing is used to determine when to update player sprites based on their activity; when a predefined amount of time has elapsed the portion of the sprite to render is updated.

Same goes for the lives bookkeeping, only when a set interval has elapsed can the count of lives be decreased, so all lives don't disappear instantly when multiple characters collisions are fired. So if the main character hits an enemy he loses one life and loses one again if he keeps hitting an enemy when the set interval has elapsed. Unless the main player hits the enemy from the top - jumps on top of him - then the enemy gets killed.

Then the GameLoop function / class calls itself repeatedly via setTimeout until all lives have been lost - then GameOver is called - and in each round it updates player positions, either from input or automation, checks collisions, checks if to update any pseudo-physics and then draws all players and platforms.

The game seems to run smoother in WebKit based browsers like Chrome or Safari, rather than Firefox for example.