|  |
| --- |
| Game Engine 2D |
| Summer project 2016 |
|  |
| The contents of this document is exclusively about the game engine project, developed in the summer of 2016, by Thomas Boel Micheelsen, spare time Java Developer. You will find documentation for various implementations and features of the game engine, and proper guides will be provided for the use of this product. In depth explanation of the functions and capabilities of the product, as well as the limitations, are also apart. |
|  |

Game Engine 2D

Summer project 2016

# Java Developed

• • •

As may have been hinted in the document résumé, this game engine is developed in Java. This means that the use of game engine requires you to be working with java for your game as well.

Whilst this certainly limits your opportunities for game language, it does not mean that you will find this as bad solution.

The game engine requires for you to have intermediate knowledge in the java language, and have an understanding of computer software programming.

Although much seems to be required, a basic understanding of the fundamentals of programming can get you a long way with the use of the game engine.

# List of contents

Sprite System

* Reads images
* Sprites are assigned to objects
* Manipulates with images

Render System

* Renders in layers
* Renders in chunks for optimization

Game-types

* Bird-perspective
* Side-perspective

Game mechanics

* Key binding
* Save / Load functions
* Player support
* Entity & Entity AI support
* Terrain support
* Item support

# Sprite System

The sprite system is capable of loading images and manipulating them in way, which allows for multiple textures in one single image. The system is used customized, so that one can choose the size of one’s spritesheet as well as changing the resolution of each individual texture. It is important to note, that the gametype plays a big role in the way the sprite system works, because the manipulation is not done with the same methods and techniques.

## Birdview Sprite System

This system preferably uses block spritesheets and item spritesheets, and for players it will remain the same as for the sideview sprite system, because all we really need to do here, is to load a series of images for the animation, with a specific player/entity resolution, also customizable. It will also be possible to scale the textures loaded, as a 256x256 texture would normally take up a huge part of the screen. In this sprite system, layers are assigned to an objects sprite. This means that the lower the priority is set, the earlier it will be rendered, so an item rendered at layer 5 will render “behind” an item at layer 6.

## Sideview Sprite System

With the sideview sprite system it’s a little bit different. As far as terrains goes, one will be able to assign a big texture, such as a 1000x500 texture, which will then work as a global value for all terrain textures. Here multiple terrain textures can be added and generate a world where collision and terrain is made up of bare textures. As mentioned earlier the player sprites are the same, and have close to no changes from the birdview and sideview. Layers will function as foreground, map and background. Background being “skybox” textures, map being where the player and the terrain is rendered and foreground being various aesthetics and whatever the user decides, all layers will be customizable and there will be support for custom layers which can work in between existing layers.

# Render System

As mentioned the render system works with layers, this is true for both gametypes. The way rendering works is simple. We’re dealing with active rendering, which means that x amount of times each second a “render” method is called, or rather a draw method. This method redraws the screen using object positions and loaded chunks to deal with how much to actually render. If we were to render the whole map at once, 30 times every second, then one can imagine that it would cause major FPS drops because it would be too hard for the program to handle. Hence we use chunks, these allow us to specify what to render and what not to. This means that it will never cause any problems for the player of the game, as there should not be too much to render, of course unless the game deals with 512x512 per each 16x16 pixel and thousands are rendered at each time, then it can get a bit tricky. Just make sure, as the user, not to go too crazy with the texture resolutions and chunks sizes.

# Gametypes

Gametypes has been mentioned a lot of times at this point, but what are they? As one may have picked up, there are two gametypes; birdview and sideview. Each gametype functions as its own game of sorts. With sideview we have gravity, specific predefined layers, more room for texture creativity and a lot more, when we with birdview, are limited a lot more when it comes to terrain.

When creating a game class, a method will have to be implemented, and it is here the user defines the gametype for their game. Some methods are only for one type of gametype whilst others function with both. Think of it as two different game engines for two different types of games.

Examples of games with sideview:

* Terraria
* Super Mario

Examples of games with birdview:

* The legend of Zelda (1986)
* The binding of Isaac

# Game mechanics

## Key binding

Key binding plays a big role in a game, if the game is too hard to play because of bad key binding, it can really quickly become irritating and frustrating for some players and they might end up stop playing or complaining. As a user it is important to make key binding customizable for the players, but also to have a set standard value for each keybinded action which is easy to use and makes sense. The game engine supports custom key binding and makes sure it is easy for the user and the players to interact with.

## Save / Load

What is a game without saving and loading? Imagine playing for 5 hours, then having to go, turning off the pc, and when you come back, all is lost. The game engine supports both local data storage as well as MySQL Database support, for those who wants to build a game where changing local filed should not be possible. Encrypted data is also supported, but is limited to my own encryption skills and can most likely be broken by people with understanding on the subject with a bit of time. But the encryption is clearly meant for the average player so that cheating cannot find place.

## Player Support

Most games these days have a player of sorts, that is of course unless one is making a solitaire game, but to my understanding and experience, these sorts of games are nothing but GUI and game logic, no need for advanced rendering or something as far away as AI. Thus the game engine has a built in player controller which the user may want to add to their own player. Each gametype has its own individual and unique player controller as the worlds function a lot differently. One has gravity, the other doesn’t, one has height, the other doesn’t, etc.

Players will also be able to contain an inventory, the inventory can either be visual or not, either way items will be stored in the inventory should it be implemented, otherwise items are not possible to exist on a player.

## Entity & AI Support

For the more advanced games, should the user want to create entities, the game engine also offers support for simple entity AI. This means entities following other entities, looking at them, moving around in random patterns. Should one want to add their own AI, there will be a class which can help the user with some base structure, but do not expect too much.

## Item Support

As the last game misc. support, we have items. The user can add attributes to their items, make them equippable and add damage attributes. Items may also affect the world or the player when “used” in any sort of context. Items will go into a player inventory should it exist, otherwise item support does not work, as there would be no way to store the items to the player.