Developer Documentation

Hoppa Ellie
Group 257
Software Engineering Project DAT255

Hannes Häggander
Dženan Baždarević
Denise Jing
Dong Yingdong
Max Modig
Su Yu-Hsuan

Overview

The game is built using the Unity3D engine version 5.01. It's a flexible game engine due to its libraries and in-editor functions to make game developing accessible to a non-game developer but exceedingly effective for those who code, too. We decided to use Unity3D because of these valuable aspects. It's easy to learn and can be used to make complex games.
The game is a 2D side scroller meaning that no 3D elements are used for gameplay, this is mostly because of the resources needed for making a 3D game compared to its 2D counterpart.

Installing the application

To install the game, find the "Builds" folder and you will find all our builds. There is also a "Final build" folder where the final build is located. To install this build on your android device. Transfer the APK file to your phone and install the game like you would any other application from the Google Play marketplace.

To make your own build, go to File and Build Settings then press build for Android. You might need to locate the Android SDK if you have not done so already.

Getting started

Clone the project from our github: https://github.com/HannesHaggander/G258
APK (installation) files are located in the *Builds* folder.

To modify the game logic, locate the code files in the "Scripts" folder within the project. All the C# files are documented and they all include a introduction as well as a description for every method in the file. For example how they are connected to other files and how to modify them.

Single use files

All player related files are named *PlayerX* where X is the functionality. Like *PlayerMovement*, *PlayerInputs*, etc, to differentiate from other areas these are only to be put on the player and nothing else. Then there are files like *CameraFollow* that can be placed on anything but is primarily for the player.

Adding levels

To add new levels look for the *LevelSelect* scene and add a game object to the main node array of game objects and it will be included when that node is selected. You still need to configure the sub-nodes (levels) to load the right level etc.
Step by step:

1. Create prefab
2. Add a sprite, box collider, and a *SubMenuLoadLevel* script to the prefab
3. Find / Create your level node in the *LevelSelect* scene and add your prefab to the level array

4. Configure what level that should be loaded when your prefab is pressed

Inputs

If there is need for another input, implement it in the PlayerInput script and then refer to that script. This makes it easier to change later on if need be. Instead of checking every file for inputs you can easily change it from one script and apply it to all inputs later by using one input-script.

Player

Currently the player prefab is highly configurable, speeds and different game checks can be changed for every level. When making a level you can tailor make your player movements to your levels advantage and make it more fun. Keep in mind that the player might be surprised if the player behaves differently on every different level and might end up confused if inputs behave differently to what they're used to. A good idéa is to have one configuration for one segment of the game, e.i. ice levels might have lower traction to the ground in comparison to the jungle level. This keeps the player interested to advance and find out what the next set of conditions are.

Save and load

The save and load function uses a binary formatter to keep file data saved. To use this file use the *Save* function from the *SaveAndLoad* script and pass on score you want to save. the *Save* function only accepts a score and a progression int for parameters, this can be changed if need be. You only have to change the class created in the *SaveAndLoad* to save more variables.

Goal

To end the level and advance to a new one we use a trigger collider that checks if there is a player in its trigger, if there is one the next level is loaded. The next level is decided by a public string located on the goal script. The easiest way to implement a goal right now is to use the goal-prefab and then configure what the next level is.

Dependencies

Unity 3d 5.01 or later versions (http://unity3d.com/get-unity)
Android device (can also run through the Unity editor)
Android SDK