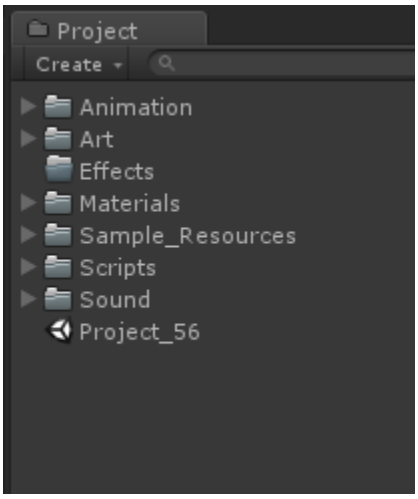
Project 56 Game Structure (Unity)

The following details entail the specification of the desired structure concerning the implementation of Project 56 on the Unity Engine.



The file structure is as what the above screenshot shows:

1) `Animation` – All animations created using sprite combinations will be listed here under their respective subfolders.
2) `Art` – All sprites, background images in either JPEG or PNG format will be listed here under their respective subfolders.
3) `Effects` – All VISUAL effects concerning the player character, mobs and environment are to be listed here under their respective subfolders.
4) `Materials` – Any custom 2D materials are to be listed here, such as Slippery material.
5) `Sample_Resources` – Any outsourced, sample assets that are to serve as PLACEHOLDERS are to be imported here neatly with their own subfolders for art, music etc.
6) `Scripts` – Any logic containing C# Code must be listed here under their own respective owners such as Player, Mobs, Level etc.
7) `Sound` – All sound files that contribute to ambience, music and sound effects are to be listed here.

Client Mandatory Requirements

The client, Knight Owl Studios, has provided a list of desired features to be seen in the game. Here are the details:

1) The game MUST be able to run on Android Devices and must be uploaded to the Play Store for distribution.
2) The game MUST be an endless runner with 1 constant level.
3) The game MUST have the following screens:
    a. Main Menu
    b. Gameplay
    c. Death
    d. Pause
    e. Shop
4) The game MUST have the following in game features:
    a. Endless running
    b. Swipe to change direction of running
    c. UI Buttons to allow Jumping, Sliding and Attacking
    d. HUD to show Score and Kills
    e. Pause Button
    f. Death
    g. Powerups
    h. Randomly generated enemies
    i. Randomly generated static obstacles
5) The game MUST have the following Google Play Services:
    a. Google Achievements
    b. Google Leaderboards
6) The game MUST have a cloud database sync that logs in with the Google Player ID

Scripting Manner

When creating or contributing to the Scripts Folder in the project, please remember to do so with the following principles/mannerism in mind:

1) Comment any significant lines of code that highlights a feature in the game that other users can understand easily.
2) Neatly separate and segregate the private variables from the public variables.
3) Avoid creating too many unnecessary FOR LOOPS since that increases the burden on the game. (Refer Big O Notation: https://en.wikipedia.org/wiki/Time_complexity)
4) Avoid dumping all collision logic onto one object such as the Player and separate the code to involved parties that are the main cause of a specific collision interaction.
5) For testing purposes, you can make certain variables public, but remember that all variables that should be secured must be declared as private before packaging for Android.
6) Please use the proper naming conventions for the variables and functions made in any script. The name needs to declare the intent of the logic involved and must follow either a CamelCase format or should have multiple words separated by an underscore. Example: superBombPowerup, user_data.

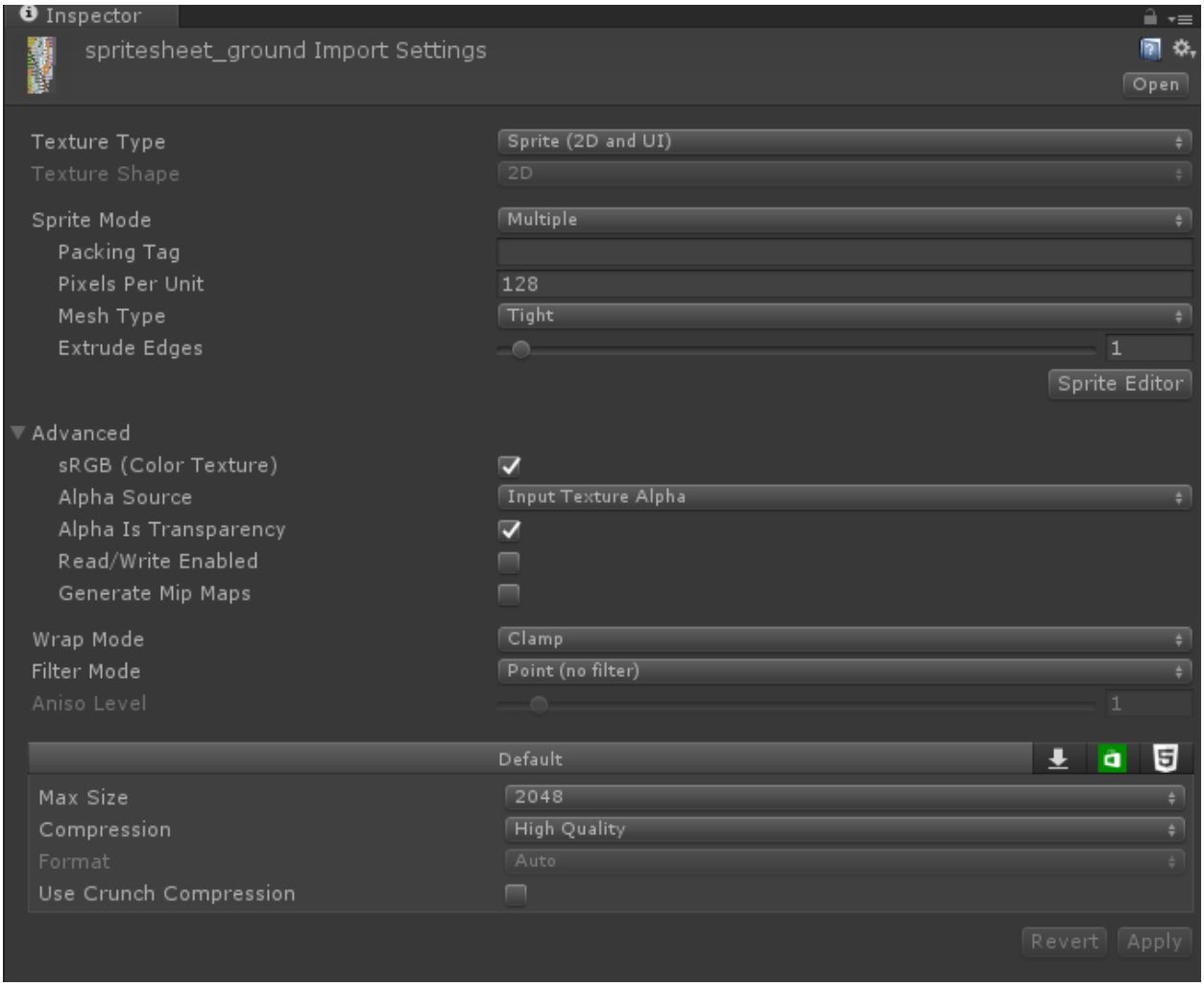
```
using System.Collections;
using System.Collections.Generic;
using UnityEngine;

public class PlayerController : MonoBehaviour {
    public bool grounded;
    public LayerMask whatIsGround;
    public float moveSpeed;
    public float jumpForce;

    private Collider2D RunnerCollider;
    private Rigidbody2D RunnerRigidBody;
    private Animator RunnerAnimator;

    // Use this for initialization
    void Start () {
        //initialize the component variables by searching for all needed components using GetComponent
        RunnerCollider = GetComponent<Collider2D>();
        RunnerRigidBody = GetComponent<Rigidbody2D> ();
        RunnerAnimator = GetComponent<Animator> ();
    }

    // Update is called once per frame
    void Update () {
        //returns true or false whether the collider is touching another collider containing the layer called 'Ground'
        grounded = Physics2D.IsTouchingLayers (RunnerCollider, whatIsGround);
        //Character will move in a direction with each frame
        RunnerRigidBody.velocity = new Vector2 (moveSpeed, RunnerRigidBody.velocity.y);

        if(Input.GetKeyDown(KeyCode.Space) && grounded == true)
        {
            RunnerRigidBody.velocity = new Vector2 (RunnerRigidBody.velocity.x, jumpForce);
        }

        RunnerAnimator.SetFloat ("Speed", RunnerRigidBody.velocity.x);
        RunnerAnimator.SetBool ("Grounded", grounded);
    }
}

```

Using Sprite sheets

It is HIGHLY IMPORTANT to apply the following principles when using a spritesheet:

Before opening the sprite editor, Make sure the

1) Sprite Mode is set to Multiple
2) Pixels Per Unit is set to 128
3) Filter Mode is set to Point (no filter)
4) Compression is set to High Quality



Unity Version

Unity version used currently is 5.6.0, Please follow suit.