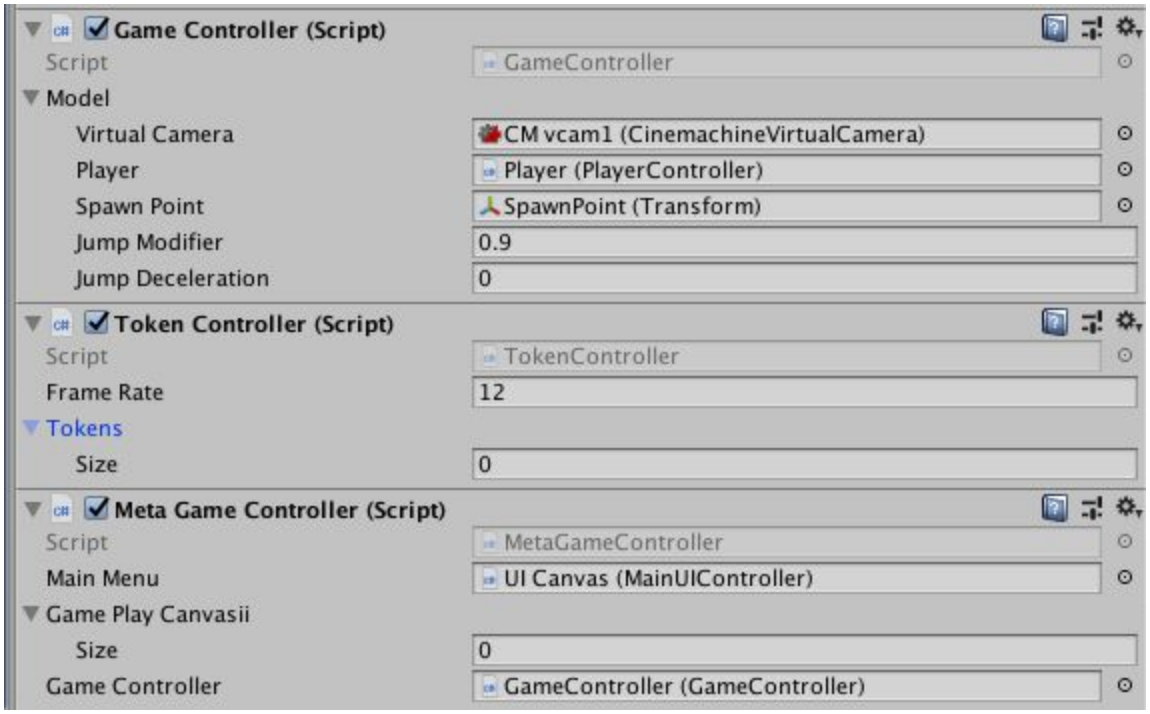
Platformer Template

Reference Documentation

This template is designed to allow creation of side scrolling platformer type games.

GameObjects in SampleScene

GameController



Model

This field contains values and references used in the game, and values which modify gameplay. You are able to access this model anywhere in your own code via:
var model = Simulation.GetModel();

Token Controller

This component controls all collectable tokens in the scene. Because there could be many hundreds of token instances, this controller animates the tokens in one big batch for performance reasons.

MetaGameController

This component controls the context switch between the menu and gameplay.

UICanvas

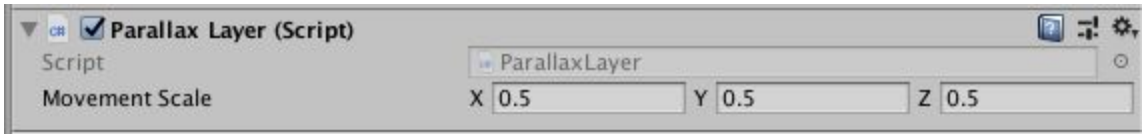
This is the main menu canvas, built using the Unity UGUI system. It contains a number of switchable tabs which are placeholders for your own GUI screens.

Player

This is the main player character in the game. This component allows customisation of audio, speeds and gravity. It can be accessed in the PlatformerModel instance.

Grid

These are the different tilemap layers which make up the scene. The ParallaxLayer component is used on background layers to create parallax scrolling effects.

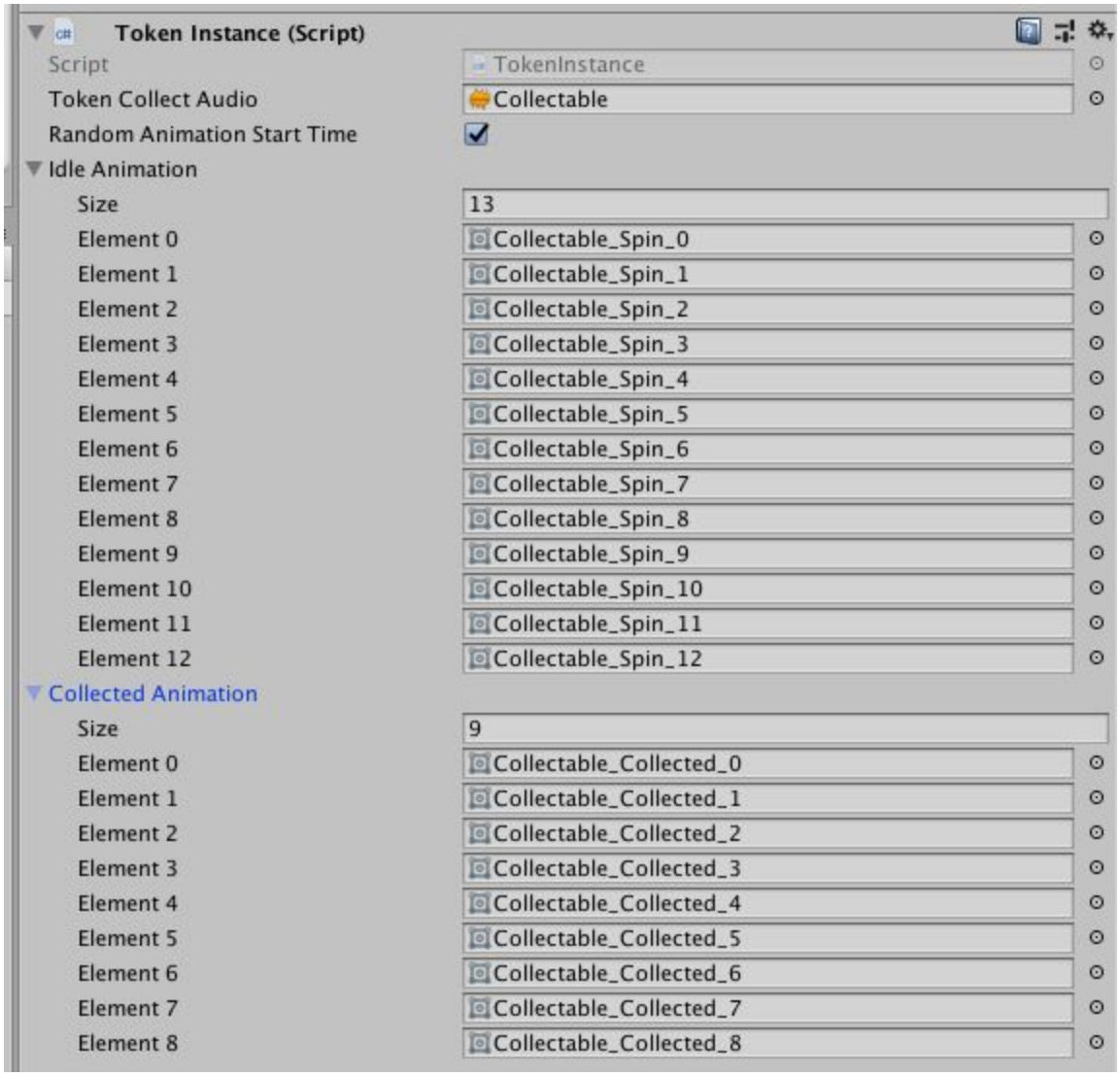


Enemies

All enemy characters in the level are placed under this game object. Enemy prefabs are added and positioned directly into the scene at edit time. If the enemy has a patrol path, it is assigned in the Enemy component.

Tokens

This is the container for all collectable tokens in the scene. Each token prefab contains a TokenInstance component, with a simple frame based set of animations which are used by the TokenController to animate tokens. This avoids having many hundreds of animator components in the scene, which would affect performance.



Zones

This is the container for game control zones. DeathZone components will cause instant death when entered by a player. A VictoryZone triggers the win condition for the scene when entered by a player.

Architecture

The code is split into four functional systems.

Mechanics:
The fundamental mechanics of the game are provided by the Mechanics namespace. This area holds all code required to create events which are used to derive gameplay. It can update the model and post events, but rarely subscribes to events. Game mechanics are generally implemented using regular Unity idioms.

Gameplay:
The Gameplay folder/namespace contains events that are triggered by the Mechanics. This is the area of code a game designer would modify to implement the rules of the game, creating gameplay. It can post events and modify the game model. Developers should try and avoid using Unity specific idioms in this code, just modify the model and post new events.

Model:
The Model folder/namespace contains classes which hold the data required for the game. Both the Mechanics and the Gameplay systems can access and modify the model. The model generally does not contain any code apart from data modification or query methods.

Visual:
The visual namespace typically provides functionality for modifying game feedback mechanisms, which are not related to mechanics or gameplay. Eg Visual Effects, Music or Animation code. Visual Systems can subscribe to events, but generally do not post events or modify the model.

Customising Behaviour

There are two ways to create behaviour when an event is fired. The Execute method of the event class can be modified, or a delegate can be assigned to the OnExecute event delegate which is fired after the Execute method has run.