CM2303 - One Semester Individual Project

40 Credits

Initial Plan

A sprite-based video game engine and editor for creating component-based games

Jack Edwards

Supervised by
Yukun Lai

Moderated by
Paul Rosin

January 30, 2017

Contents

List of Figures

1. Project Description

Modern technology has made the creation of video games a much more accessible task to individuals without the funds for expensive computer hardware and software, causing a huge increase in the number of independent games on the market. A contributing factor to this is the wide availability of high-powered game engines and editors such as Unity and Unreal Engine, which abstract a lot of the low-level details of a game to allow the user to focus on high-level development. Unity in particular features an entity-component system, allowing users to add functionality to a game by creating custom, modular components. This leads to a very efficient workflow by obscuring the complex details that allow the engine to function (e.g. rendering, memory management) and allowing the user to focus on implementing functionality specific to their game. Although there are many engines to choose from these days, the majority of the most popular engines focus on 3D game development and either feature no support for developing 2D games or very limited support, usually considering it an afterthought. Regardless of this, 2D video games are still popular and very much in demand.

This project aims to allow the creation of 2D games, utilising an entity-component system to allow users to easily add functionality to their games by creating modular components, without the need to consider every aspect of the underlying architecture of the game engine. There are three main features that the project aims to deliver - a game engine, a game editor and an example game. The game engine will be the underlying engine that users will use as a base for their games and will include low-level features that allow the engine to function and several high-level features that users are able to take advantage of. The game editor will be a Windows desktop application that will integrate with the game engine and allow users to create games using a graphical interface as opposed to creating them entirely by writing code. The example game will be a clone of the popular game 'Brick Breaker' and will showcase the capabilities of the game engine.

2. Aims & Objectives

This project can be divided into several aims and objectives that must be achieved to produce a high quality, finished product.

2.1 Software Development Methodology

To ensure that the development process proceeds as smoothly as possible and the product meets a high standard, I must follow a software development methodology. For this project, I will use a subset of the Agile[1] methodology called Scrum[2]. Using the Scrum methodology, work is completed in sprints. A list of all the desired features of the product is created at the beginning of the development process, and when each sprint begins a subset of those features will be added to the sprint to be completed before the end of the sprint. Using the Scrum methodology allows for a flexible development process by utilising minimal planning, because if any changes need to be made it will not cause too much disruption to the development process. Another advantage of using Scrum is that the product is constantly kept in a shippable state, because testing is performed at the end of each sprint to ensure that the software works correctly at all times.

In this project, I will be working in 12 1-week long sprints dated from 30/01/17 - 23/04/17. Each sprint will begin with the implementation of any features in the current sprint, followed by fixing any outstanding bugs and ending with testing the features implemented in the sprint. If bugs are found in features implemented in any given sprint, the task of fixing those bugs will be added to the next sprint - due to this, I should be conservative when deciding which features to implement in each sprint to ensure that there will always be enough time to complete all of the tasks in a sprint. If a sprint is completed early (i.e. there are no remaining tasks), development can continue by working on tasks from the next sprint.

2.2 Software Architecture

Due to the large scope of the project, the architecture of the software must be considered carefully. In this project I will use the Onion architecture[3] for the development of the engine and the editor. The Onion architecture describes a layered architecture that helps to keep code clean and reusable by only allowing layers to reference inner layers in the project. The 3 layers that I will use in my projects are Core, Data and UI where the core is at the centre, the data layer is outside and can reference the core, and the UI layer is the outer-most layer and can reference both the core and data layers. This architecture allows the core of the project to be separated from the data and the presentation of the software. This type of architecture also makes for better maintainability and testability.

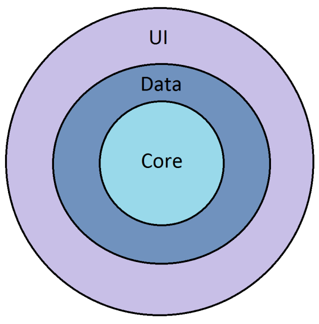


Figure 2.1: Onion architecture diagram

2.3 Game Engine

The game engine will be the underlying engine that users will use as a base for their games and will include a number of low-level features that allow the engine to function correctly as well as a selection of high-level features that the user can take advantage of. For example, the engine will feature built-in support for detecting collisions between entities and reacting to them within any component. The main features of the engine are as follows:

- **Entity-component system** - A system that allows functionality to be added to a game by creating generic entities, creating custom components and attaching components to entities.
- **Scenes** - Entities, components and resources will belong to a scene, and a game can contain multiple scenes. Scenes can be used to separate levels, areas of a game etc.
- **Rendering** - The engine will feature sprite rendering and text rendering. Image files can be used as textures for sprites, and fonts can be used when rendering text.
- **Game Loading** - The engine will be able to launch a game from a .crunch file - an XML file that describes the scenes, entities, components and resources in a game. This is the file that will be output by the game editor.
- **Resource saving/loading** - The engine will be able to load and save resources. For example, an image can be loaded to be used as a texture, or a text file could be saved to track the players score.
- **UI Controls** - The engine will contain several predefined controls to allow the user to create their own user interfaces. These controls will include labels, text boxes and check boxes.
- **Audio** - The engine will support audio clips that can be be used for sound effects and background music. The user will be able to control the audio volume and decide whether an audio clip should loop.
- **Animation** - The engine will allow animated sprites to be used within games. Users will be able to use looping animations that play continuously and animations that will begin when they are triggered.

The engine will also include several ready-made components for users to take advantage of that will handle various commonly included functionality such as collision detection/reaction, transformable game cameras and user interface controls (buttons, text boxes etc.).

2.4 Game Editor

The game editor will be a Windows desktop application created using WPF (Windows Presentation Framework) that will integrate with the game engine and allow users to create games using a graphical interface by adding and transforming entities, attaching components to entities, adjusting the values a components properties and handling game resources. The main areas of the editor are as follows:

- **Entity hierarchy** - A tree view that allows users to see the hierarchy of entities within their game and add new entities.

- **Entity properties view** - A pane that allows users to edit values related to a single entity, such as its name and its enabled status. Each component attached to the specified entity is also displayed along with the properties of each component, also allowing the user to edit these properties using an appropriate control (a text box for string values, e.g. a slider for integer values).
- **Resources view** - A list of all of the resources included in the current game (images, sounds etc.). This area allows the user to add resources that already exist on their computer, remove resources that already exist in the project and create new resources for the game to use.
- **Scene viewer** - A Monogame instance that allows the user to transform entities in a graphically interactive manner. For example, if the user adds an entity with a Sprite component the sprite will be displayed at the specified coordinates in the scene viewer. The user is then able to select an entity and translate, rotate and scale it by manipulating on-screen handles using the mouse cursor.
- **Game view** - Allows the user to test the game they are creating by running it directly within the editor.

2.5 Example Game

After the completion of the engine and editor I will produce an example game - a clone of the popular 'Brick Breaker' game, to showcase all of the capabilities of the engine. This game will contain the following features:

- Sprite rendering
- Text rendering
- User input detection/reaction
- Multiple scenes and navigating between scenes
- Resource loading/saving
- Collision detection/reaction
- Sprite animation
- Sound effects
- Background music
- An adjustable game camera
- Various user interface controls

2.6 Software

Throughout the development process I will use a range of different software and libraries to allow me to work efficiently and effectively.

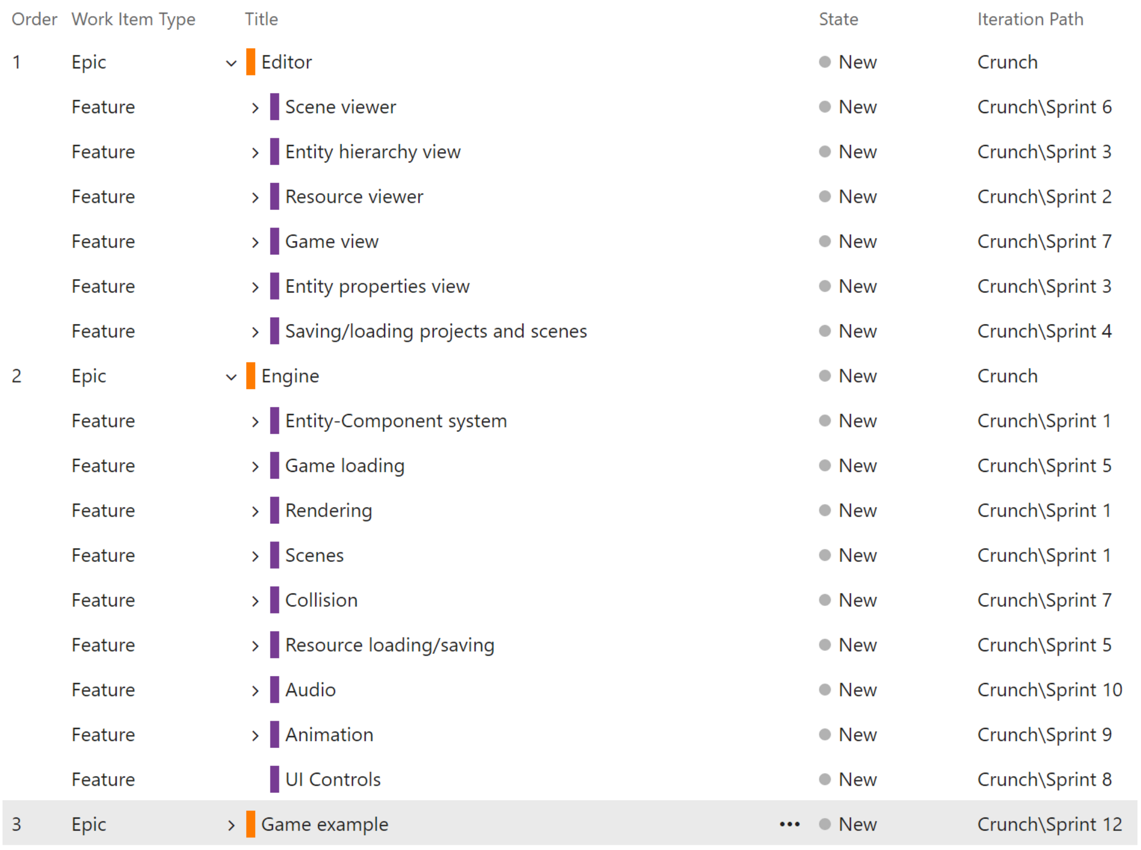
- **Visual Studio 2015** - I will be writing all of my code within Visual Studio[4] due to its superb support of my language of choice, C#. As well as allowing me to write code, it features built-in support for checking code into source control, building & deploying projects, visually creating WPF interfaces and integration with Visual Studio Team Services[5].

- **Visual Studio Team Services** - This is the software I will be using to handle source control because of its superior integration with Visual Studio. As well as managing source control, it has fantastic features for managing projects - specifically projects that use the Scrum architecture. I will use the software to manage sprints, create user stories and assign tasks to certain sprints. It also supports the ability to mark tasks as 'Complete' when the associated code is checked into source control, allowing me to always have an up-to-date overview of what is left to complete in each sprint.

- **Monogame** - I will be using Monogame[6], an open-source implementation of Microsoft's deprecated XNA framework, to create the game engine. Due to the scope of the project I decided that using Monogame would be the best choice because it will allow me to focus on the higher-level details of the engine. For example, if I were to create the engine from the ground up using DirectX I would not be able to fit all of the desired features of the project into my time plan.

3. Work Plan

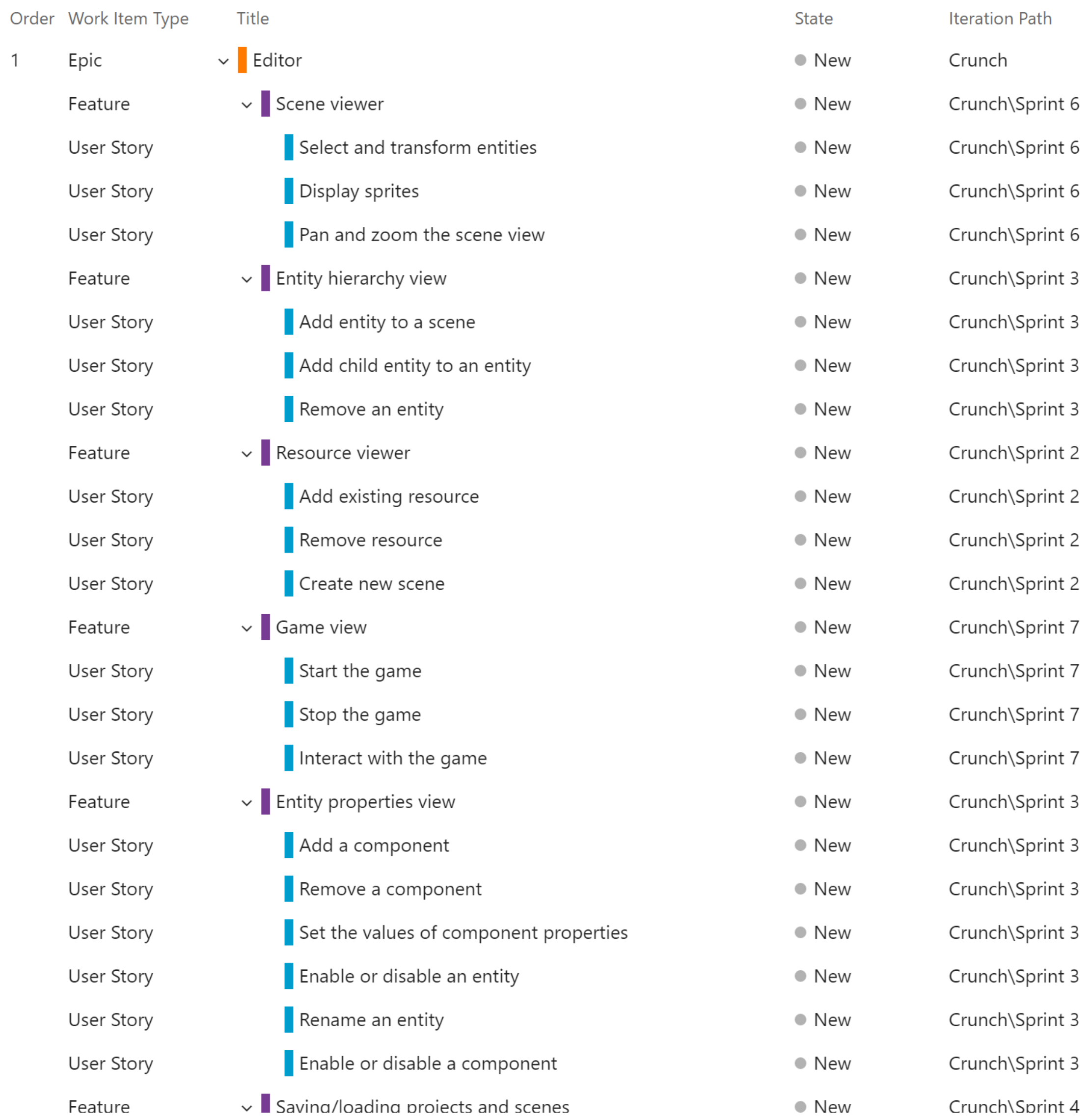
Due to the fact I will be working in sprints and I have decided to use Visual Studio Team Services to manage those sprints, I decided that the most reliable way to create a work plan would be using the software. Using VSST, I have outlined all of the features that need to implemented in the engine, the editor and the example game and assigned each feature to a specific sprint. I have made sure to leave spare time in each sprint to account for any bugs that need to be fixed, different features that need to be added or if the project is otherwise delayed for some reason. Each sprint in figure 2 lasts one week, from Monday to Sunday, beginning on 30/01/17.



Order	Work Item Type	Title	State	Iteration Path
1	Epic	Editor	New	Crunch
	Feature	Scene viewer	New	Crunch\Sprint 6
	Feature	Entity hierarchy view	New	Crunch\Sprint 3
	Feature	Resource viewer	New	Crunch\Sprint 2
	Feature	Game view	New	Crunch\Sprint 7
	Feature	Entity properties view	New	Crunch\Sprint 3
	Feature	Saving/loading projects and scenes	New	Crunch\Sprint 4
2	Epic	Engine	New	Crunch
	Feature	Entity-Component system	New	Crunch\Sprint 1
	Feature	Game loading	New	Crunch\Sprint 5
	Feature	Rendering	New	Crunch\Sprint 1
	Feature	Scenes	New	Crunch\Sprint 1
	Feature	Collision	New	Crunch\Sprint 7
	Feature	Resource loading/saving	New	Crunch\Sprint 5
	Feature	Audio	New	Crunch\Sprint 10
	Feature	Animation	New	Crunch\Sprint 9
	Feature	UI Controls	New	Crunch\Sprint 8
3	Epic	Game example	New	Crunch\Sprint 12

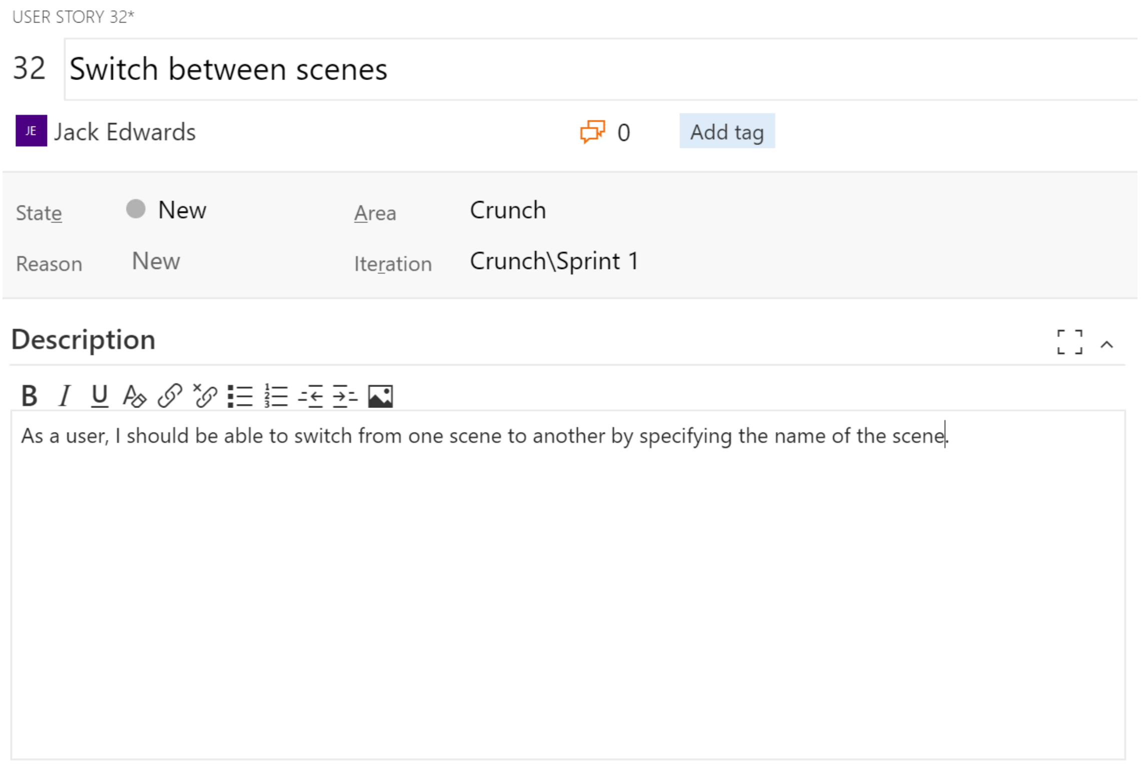
Figure 3.1: Time plan

To help to determine how long a feature may take to implement, as well as helping with the actual development of a feature, I have assigned user stories to each feature, as shown in figure 3. These user stories describe exactly what the software should be able to do from a users perspective. For example, the engine has a 'Scenes' feature that contains the user story 'Switch between scenes'. The description of this user story describes how a user should be able to switch between different scenes, as shown in figure 4. At the beginning of each sprint I will add more specific tasks to each user story (e.g. 'Create Scene class') to help with the development process.



Order	Work Item Type	Title	State	Iteration Path
1	Epic	Editor	New	Crunch
	Feature	Scene viewer	New	Crunch\Sprint 6
	User Story	Select and transform entities	New	Crunch\Sprint 6
	User Story	Display sprites	New	Crunch\Sprint 6
	User Story	Pan and zoom the scene view	New	Crunch\Sprint 6
	Feature	Entity hierarchy view	New	Crunch\Sprint 3
	User Story	Add entity to a scene	New	Crunch\Sprint 3
	User Story	Add child entity to an entity	New	Crunch\Sprint 3
	User Story	Remove an entity	New	Crunch\Sprint 3
	Feature	Resource viewer	New	Crunch\Sprint 2
	User Story	Add existing resource	New	Crunch\Sprint 2
	User Story	Remove resource	New	Crunch\Sprint 2
	User Story	Create new scene	New	Crunch\Sprint 2
	Feature	Game view	New	Crunch\Sprint 7
	User Story	Start the game	New	Crunch\Sprint 7
	User Story	Stop the game	New	Crunch\Sprint 7
	User Story	Interact with the game	New	Crunch\Sprint 7
	Feature	Entity properties view	New	Crunch\Sprint 3
	User Story	Add a component	New	Crunch\Sprint 3
	User Story	Remove a component	New	Crunch\Sprint 3
	User Story	Set the values of component properties	New	Crunch\Sprint 3
	User Story	Enable or disable an entity	New	Crunch\Sprint 3
	User Story	Rename an entity	New	Crunch\Sprint 3
	User Story	Enable or disable a component	New	Crunch\Sprint 3
	Feature	Saving/loading projects and scenes	New	Crunch\Sprint 4

Figure 3.2: User stories assigned to each feature


USER STORY 32*

32 Switch between scenes

Jack Edwards

State	New	Area	Crunch
Reason	New	Iteration	Crunch\Sprint 1

Description

As a user, I should be able to switch from one scene to another by specifying the name of the scene.

Figure 3.3: The description of the 'Switch between scenes' user story

Alongside development, I will be meeting with my project supervisor to review my progress every two weeks on a Friday, beginning on 03/02/17.

Bibliography

[1] Agilemethodology.org *The Agile Movement*. http://agilemethodology.org/.

[2] Scrummethodology.com *An Empirical Framework For Learning (Not a Methodology)*. http://scrummethodology.com/.

[3] Chetan Vihite's Blog *Understanding Onion Architecture - Chetan Vihite's Blog*. http://blog.thedigitalgroup.com/chetanv/2015/07/06/understanding-onion-architecture/.

[4] Visual Studio *Visual Studio | Developer Tools and Services | Microsoft IDE*. https://www.visualstudio.com/.

[5] Visual Studio *Agile, Git, CI | Visual Studio Team Services*. https://www.visualstudio.com/team-services/.

[6] Monogame.net *MonoGame | Write Once, Play Everywhere*. http://www.monogame.net/.