OULU UNIVERSITY OF
APPLIED SCIENCES

Cuong Nguyen Tuan

USER INTERFACE SYSTEM IMPLEMENTATION IN GAME ENGINE

USER INTERFACE SYSTEM IMPLEMENTATION IN GAME ENGINE

Cuong Nguyen Tuan
Bachelor's Thesis
Autumn 2016
Degree Program in Information Technology
Oulu University of Applied Sciences

ABSTRACT

Oulu University of Applied Sciences
Degree programme in Information Technology

Author: Cuong Nguyen Tuan
Title of the bachelor's thesis: User Interface Components Implementation in Game Engine
Supervisor: Veikko Johannes Tapaninen
Term and year of completion: Autumn 2016 Number of pages: 52 pages

---

The thesis work was based on user interface implementation theory for RPG Maker MV, a Japanese game engine. The outcome of this work was creating a user interface system which was easy to configure and install whilst powerful enough for advanced users and developers.

In this project, many modern web technologies were used, such as HTML5, WebGL, single-page application with Angular.js 2, programming language TypeScript and Nw.js, an application runtime uses web technologies to build native app.

The project has successfully implemented the user interface framework, allows users creating their own design in-game. There was still room for further development, such as allowing users implementing new features with the provided UI tool, or allowing third-party developers integrating features into the system and tool with simple APIs.

Keywords: user interface, games, web, JavaScript, TypeScript

PREFACE

This thesis is my final work of my Bachelor study at the Oulu University of Applied Sciences. It presents the results of my study in software development and game development last 3 years.

I would like to thank my thesis supervisor, Veikko Johannes Tapaninen, for mentoring me during the work, and my English teacher, Kaija Posio, for correcting language in my thesis report.

I would like to send my love to my family, who have been always encouraging and supporting me in both work and study.

Finally, I would like to thank my girlfriend, who has been always be by my side during my study in Finland.

Oulu, September 20, 2016

Nguyen Tuan Cuong

CONTENTS

VOCABULARY

2D	Two-dimensional space
3D	Three-dimensional space
API	Application Programming Interface
Bottleneck	A kind of hardware limitation
CPU	Central Processing Unit
Cutscene	Event scene or in-game cinematic, a sequence in video game without gameplay
eval	A function in programming evaluating a string as an or some expression(s)
Fork Project	An independent development based from an existed source code
HTML	HyperText Markup Language
HTTP	HyperText Transfer Protocol
JSON	JavaScript Object Notation
MVC Pattern	Model – View – Controller Pattern
NPC	Non-playing Character
PC	Personal Computer
RPG	Role-playing Game
Sprite	2D bitmap integrated into a scene
Stargazer (GitHub)	Bookmark or favourite on GitHub

UI	User Interface
WebGL	Web Graphics Library
Window	In RPG Maker, Window is the base class for UI components
WYSIWYG	What You See Is What You Get

1 INTRODUCTION

In modern software industry, including video game industry, the user interface (UI) design has been considered as an essential element in developments. It is focuses on anticipating what users might need to do and ensuring that the interface has elements that are easy to access, understand, and use to facilitate those actions (Usability.gov 2016, cited 16.9.2016).

There are many libraries, frameworks and tools existed to help developers create user interfaces for software, web and video games, such as Qt framework (for software), Angular.js 2 (for web) and the Unity3D's UI tools (for game). They help reducing many complex tasks involve graphical renderers, allows developers implementing the user interface designs into actual products.

In game development, a game engine ships with developing tool and level editor rises in popularity since Unity3D and Unreal Engine 4 went free for educations and independent developers. They also have strong yet simple user interface design tools implemented, allow developers creating the interface with the given APIs as the game developed.

However, both Unity3D and Unreal Engine 4 are too complicated for beginners and non-programmers to develop games. Despite they give most of necessary tools for users and a rich and simple editor, developers still need a vast of knowledge in programming to develop the game, also the user interface. Consequently, many beginner-friendly game engines have been introduced, such as RPG Maker MV and GameMaker.

The main outcome of this thesis is to build a user interface system for RPG Maker MV which is easy to configure and implement. Furthermore, the system will come with a tool helping developers in user interface design implementation.

The basic idea of the implementation is creating a user interface system matched RPG Maker MV motive: "Powerful enough for a developer. Simple enough for a child." (DEGICA Co.,Ltd 2015, cited 5.8.2016). The UI framework

should be ready for production, easy to use, allows users designing an interface for games without knowing to programming whilst giving advanced users ability to easily extend by programming. On the other hand, the system has to be easy to install and update whilst maintaining the high performance.

The thesis includes the technologies used for the implementation, introduces the concepts of the user interface system and the improvements from old system to the new ones. Furthermore, the problems are presented along with solutions, giving more specific explanations to the system development.

2 WHAT IS A ROLE-PLAYING GAME?

Role-playing game, often called as RPG, is a game genre where player plays as one or some characters in a well-defined world, many of them have origins in pen-and-paper role-playing games and use much of the same terminology, settings and game mechanics (Rollings & Adams 2013, 347).

In a RPG world, player controls a character or some party members to interact with other characters, often called as non-playing characters or NPC, follows a story plot, completes some quests and grows main characters in power and abilities (Rollings & Adams 2013, 347).

FIGURE 1. Baldur's Gate, a fantasy role-playing game (Overhaul Games 2012, cited 2.9.2016)

Figure 1 describes the basic ideas in a role-playing game, where player controls different members in a party to go around the game world and fight with enemies.

One of key features in a RPG is the rich story and its settings in developing events and characters. In RPG, story often provides much of the entertainment

in the game, also in modern RPG, recorded dialogs and voiceover narration are used along with many long cutscenes to develop strong impression in the overall storyline (Adams 2009, 453 – 455).

Besides story, growing characters in power and abilities is a key feature in RPG. As the story goes, the difficult in game also increases and the characters will gain more power based on number of battles they went to and sometime based on the story development. Many RPG uses the level, stats and skills tree mechanics in design to allow player customizes and grows their characters based on their choices (Adams 2009, 453 – 455).The equipment and items are also introduced in RPG to increase characters' power, give them new abilities and/or restore their health points and can be bought from shops or dropped from enemies (Adams 2009, 453 – 455).

FIGURE 2. Path of Exile passive tree, a way to customize and power up character (Berry 2013, cited 2.9.2016)

Figure 2 shows the passive tree in the game Path of Exile, which allows player customizing the character by giving new abilities and increasing power based on the nodes he or she chooses.

RPG often has a complex menu and UI in game since it has many complexity mechanics and features. In a game engine, the UI system has to be customizable enough to support many kind of game mechanics and allows developers to make their game looks different to the others. A RPG engine will be the target for the UI development because of this.

3 TECHNOLOGIES

This chapter introduces the technologies used for the project, including the game engine, programming language and libraries.

3.1 Browser Game

A browser game is a video game that is playable over the Internet using a supported web browser (Schultheiss 2007, 344). At present, browser game is also playable on mobile devices such as smartphone and tablet with a supported browser; furthermore, it can be wrapped in a web-wrapper such as Nw.js or Electron to publish as a standalone package that allows playing without a browser and Internet.

Previously, browser game was simple text game which did not allow real-time updating and the visual was mostly based on static images. When some media technologies were introduced to browsers, such as Flash and Shockwave, browser game that time were much like a video game, supported visually update and playing; however, it required players installing plugins for their browsers to play the game. On the other hand, browser game at that time only supported 2D; 3D technology was impossibly implemented.

FIGURE 3. Dark Warriors, a text based browser game

Figure 3 shows a browser text game where the interactions between players and system are based on text only. There is no real-time update between each action and no animation for the character.

With current web technologies, especially canvas and WebGL, they allow developer making video games run on both browser and web-wrapper. Furthermore, with WebGL support in most of modern browsers. As the game is browser based, it is allowed running on multiple platforms that support web technologies.

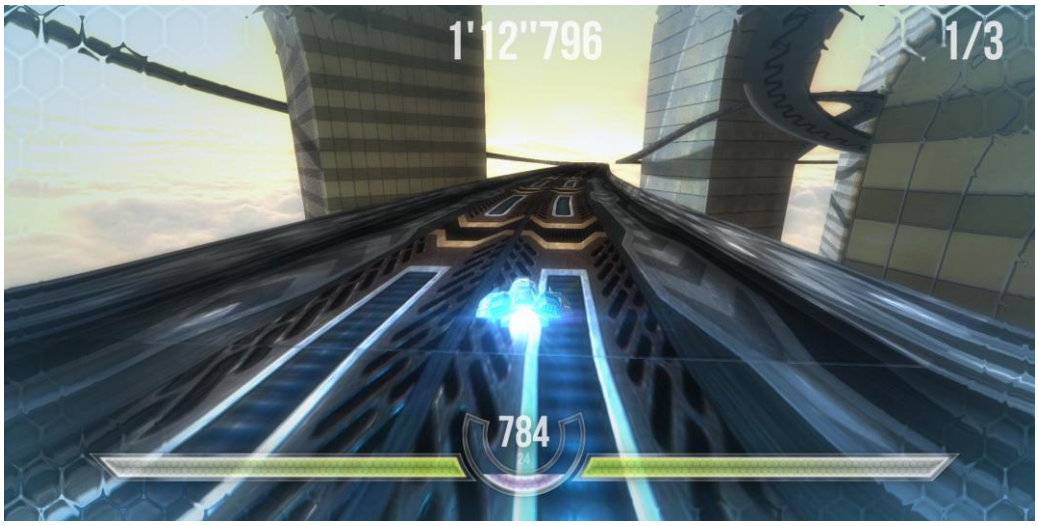

FIGURE 4. HexGL, a 3D browser game built on HTML5 and WebGL

Figure 4 introduces a well-made 3D browser game built on modern technologies. The game's graphics and mechanics are not different to traditional video games.

3.2 Game Engine

The term "game engine" had been used in mid-1990s in reference to some first-person shooter (FPG) games like the popular Doom by id Software, which it was architected with a well-defined separation between the core components, such as the 3D graphics rendering system or the audio system, and the assets, data, game worlds and rule of play. Developers began licensing the games and developing them into new products by creating new assets, world layouts, weapons, characters and game rules with minimal changes to the "engine" (Gregory, Lander & Whiting 2009, 11).

In present days, the term "game engine" has been widely used to describe a software or a game that allows developers licensing and easily creating new game products on its core engine without any major modification in the software. In development, developers do not have to develop the renderers or the systems for the game, almost necessary tools such as graphics renderer, audio system or the collision system are provided by the game engine. Furthermore,

the engine allows developers importing supported assets, data and programming the game logic at ease.

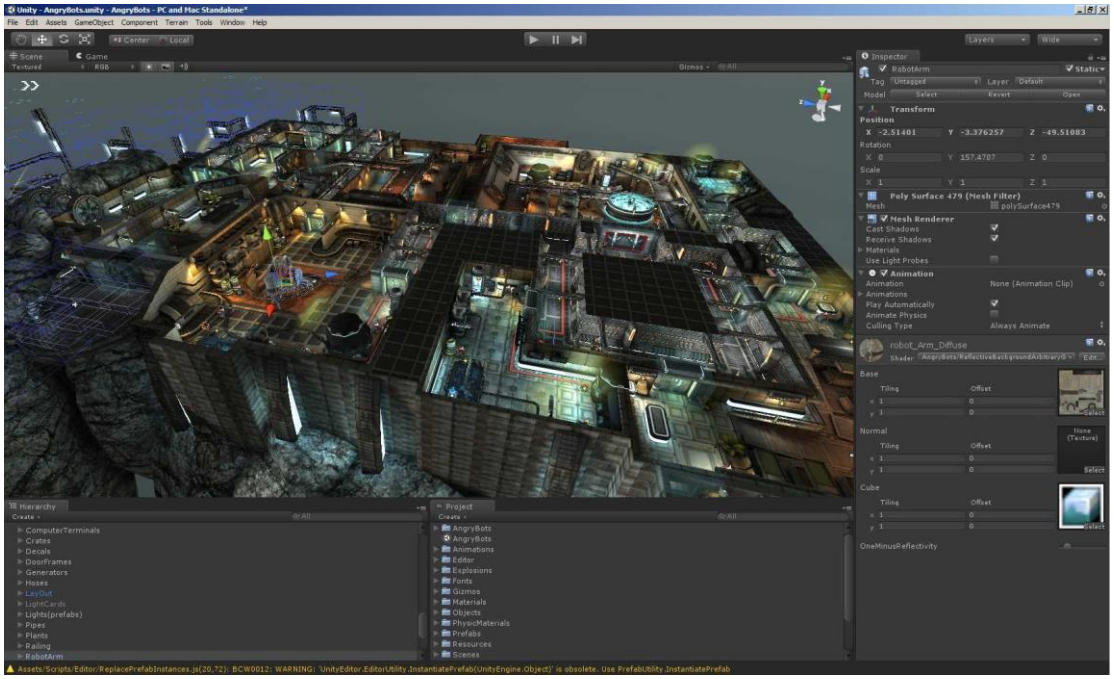

FIGURE 5. Unity3D, a popular game engine for independent developers

3.3 RPG Maker MV

RPG Maker MV is a cross-platform game engine which allows users to develop role-playing games and adventure games. The engine is “powerful enough for a developer, simple enough for a child” (DEGICA Co.,Ltd 2015, cited 5.8.2016). RPG Maker MV was developed by KADOKAWA Corporation and published internationally by Degica Co, Ltd.

FIGURE 6. RPG Maker MV editor and multiplatform distribution (DEGICA Co.,Ltd 2015, cited 5.8.2016).

The engine framework was backed by Pixi.js, a 2D renderer library for web which is introduced in chapter 2.3, and written in JavaScript language with ECMAScript 5 standard.

The selling point of RPG Maker MV is the event system, where users can easily create game without knowledge to programming. The system includes many options, from simple commands like show character message, give and take items to more advanced commands like conditional branch (works like If-Else in programming) and even script call, where users can put their JavaScript code in. The event system will handle most of game's mechanics.

RPG Maker MV is also shipped with a pack of sample data, graphics and audio. Developers can start developing their games or prototype without finding or making resources. However, users are limited to very little control over the game system, especially the game UI. Due to that reason, RPG Maker MV has been chosen as the subject of the implementation.

3.4 TypeScript

TypeScript is an open source programming language developed and maintained by Microsoft. It is introduced as a superset of JavaScript and will be compiled to plain JavaScript which will work in any browser without additional scripts (Microsoft Corporation 2012, cited 4.8.2016).

The programming language provides static typing for JavaScript, developers will have a type-safe and structure checking before testing and production. The structure and types problem will be caught in compilation. The development will be improved by shortening the error-checking time in runtime (DataArt 2014, cited 1.9.2016).

For instance, developers often get problem with JavaScript number cast, which will cast a string into a number, in worst case, it will return an invalid value. In TypeScript, the type-checking will save a lot of time for debugging problem like this. The below example code will throw an error whilst compiling because of the static type.

```
interface Actor {
    health: number;
    attack: number;
    defense: number;
}

class Monster implements Actor {
    health: number;
    attack: number;
    defense: number;

    constructor(health: number, attack: number, defense: number) {
        this._setupStats(health, attack, defense);
    }

    private _setupStats(health: number, attack: number, defense: number){
        this.health = health;
        this.attack = attack;
        this.defense = defense;
    }

    takeDamage(attack: number) {
        let damage = attack - this.defense;
        this.health = Math.max(this.health - damage, 0);
    }
}

let bigMonster = new Monster(100, 10, 5);
bigMonster.takeDamage("1"); // invalid type
```

FIGURE 7. TypeScript type-checking example code

```
test.ts(29,23): error TS2345: Argument of type 'string' is not assignable
to parameter of type 'number'.
```

FIGURE 8. TypeScript type-checking error

The sample code in figure 7 describes how to use static typing feature in Type-Script. The last two lines of code shows one valid statement and one invalid statement respectively that involved the static types. Figure 8 shows the error text when compiles the code, which tells developer to check the parameter's type in the last line of code.

Furthermore, interfaces in TypeScript always keep the data in structure, they will make sure developers pass correct data structures to system. They focuses on the shape that values have, fill the role of naming custom types and are a

powerful way of defining contracts within project as well as outside code (Microsoft Corporation 2016, cited 4.8.2016). This is especially useful since in JavaScript, developers often pass complicated data to API by using JSON.

```
interface ButtonSettings {
    text: string;
    size?: { width: number; height: number; };
    color?: string;
}

function createButton(settings: ButtonSettings) { ... }

createButton({ text: 'Submit' }); // OK
createButton({ text: 'Submit', size: { width: 70, height: 30 }}); // OK
createButton({ text: 'Submit', color: 43); // Not OK: 43 isn't a string
createButton({ text: 'Submit', size: { width: 70 }); // Not OK: size
needs a height as well
createButton({ color: 'Blue'}); // Not OK: 'text' member is required
```

FIGURE 9. Describe ButtonSettings type and type-checking example (Cavanaugh 2013, cited 4.8.2016)

As above example, TypeScript always make sure all complicated types are structured and organized by using Interfaces. In the UI system, this feature in TypeScript helps developers when defining and structuring the configuration and the data using in system (Nizet 2016, cited 2.9.2016).

3.5 Pixi.js

Pixi.js is an open source JavaScript rendering library that allows developers to "create rich, interactive graphics, cross platform applications, and games without having to dive into the WebGL API or deal with browser and device compatibility" (GoodBoyDigital 2015, cited 7.8.2016).

Pixi.js is the backed library for rendering graphics and audio in RPG Maker MV. The library fully supports WebGL and has a HTML5's canvas fallback, the performance and cross-platform ability are guaranteed.

Pixi.js is widely used for games and many media application, including advertisements and product introduction pages for many products. According to

Pixi.js GitHub repository stats in July 2016, it has more than 10000 stargazers and more than 2000 fork projects.

```
var renderer = PIXI.autoDetectRenderer(800, 600,{backgroundColor :
0x1099bb});
document.body.appendChild(renderer.view);

// create the root of the scene graph
var stage = new PIXI.Container();

// create a texture from an image path
var texture = PIXI.Texture.fromImage('_assets/basics/bunny.png');

// create a new Sprite using the texture
var bunny = new PIXI.Sprite(texture);

// center the sprite's anchor point
bunny.anchor.x = 0.5;
bunny.anchor.y = 0.5;

// move the sprite to the center of the screen
bunny.position.x = 200;
bunny.position.y = 150;

stage.addChild(bunny);
```

FIGURE 10. Pixi.js basic example, the example shows a bunny on screen (GoodBoyDigital 2015, cited 7.8.2016)

In April 2016, PixiJS v4 has been announce with a better performance for modern devices and browsers, especially for mobile devices. RPG Maker MV also had an update in August 2016 which implemented PixiJS v4 into libraries, replaced the old PixiJS v3 (Palmer 2016, cited 15.8.2016).

3.6 Nw.js

Nw.js is an open source application runtime that uses web technologies sponsored by Intel. The app runtime is created based on Chromium and node.js, which allows developers writing cross-platform apps in HTML and JavaScript (Nw.js 2015, 18.8.2016). It allows developers packaging a web application to a native application which runs on computer without a browser (Nw.js 2016, cited 14.9.2016).

Nw.js also supports most of Google Chrome extensions, allows developers implementing many existed features on Chrome Web Store.

Nw.js is used in RPG Maker MV, allows developers to publish games on PC. Furthermore, it allows developers writing game engine plugins at ease without getting access to the main editor source code.

3.7 Angular.js 2

Angular.js 2 is an open source client-side web framework. It supports building web apps with impressive performance in a short development time (Angular 2016, cited 14.9.2016).

By using the router and templates, Angular.js 2 is specialized in building single-page app, making the web works exactly like a native application when working with a web-wrapper, such as Nw.js (Angular 2016, cited 14.9.2016).

```
@RouteConfig([
    {
        path: '/scenes',
        name: 'Scenes',
        component: ScenesComponent,
        useAsDefault: true
    },
    {
        path: '/scene/:sceneId',
        name: 'SceneDetail',
        component: SceneDetailComponent
    },
    {
        path: '/scene/:sceneId/:windowId',
        name: 'WindowDetail',
        component: WindowDetailComponent
    }
])
export class AppComponent {

}
```

FIGURE 11. Routing example in Angular.js 2

The figure 11 shows the routes configuration in Angular.js 2, which is used for single-page application, allows users experiencing a web application similar to a desktop application.

In the UI implementation for RPG Maker MV, Angular.js 2 is used with Nw.js to create an editor for the game UI. This implementation provides real-time edit to UI system, allows sharing data between game player and the editor easily since they both using one instance of Nw.js.

3.8 MVC pattern

Model – View – Controller pattern, or MVC, is a software architectural pattern for implementing user interfaces which each piece of architecture is well-defined and self-contained (Freeman 2013, 67).

In a software designed in MVC pattern, it is separated into models, views and controllers which hold the data and responses, presentation and actions based on user input respectively. The view and controller classes depend on the model classes, whilst the model depend on neither the view nor the controller, allows the data structure to be built and tested independent of the visual presentation. On the other hand, MVC pattern separates the user interface logic and the business logic, with the controller and view handle the user interface logic, the model handle the business logic (Microsoft Corporation 2016, cited 15.9.2016).

The MVC pattern gives the benefit of modifying, maintaining and testing the user interface whilst the features, data and behaviours (presented by the model) built based on business logic (Kanian77 2008, cited 15.9.2016). The benefit allows developers changing the user interface without modifying the system, reduce the developing and testing time for user interface.

4 THE PREVIOUS UI IMPLEMENTATION

This chapter will introduce the concept and the actual implementation of previous UI system.

4.1 Concept

In earlier concept, the plugin was made to configure the stock UI. However, developers were not able to add their own components and edit the content of stock components. In this version, the concept introduced a way to configure UI by using Ruby's hash and high-end developers would have access to the script to make their own UI design.

In later edition, the concept introduced a way to make their own components and UI contents by using an easy-to-learn language. The idea was introducing a way to design UI without learning programming. Furthermore, the UI configuration would be separated from game's logic for easier configuring and finding features. The UI was made following the MVC pattern, so that the system would be easier to maintain and extend.

To summarise, the concept would be broken into smaller problems:

- Configuring user interface by using a high readability data format
- Allowing user creating and modifying user interface without understanding of programming
- Designing user interface system following MVC pattern, allowing third-party developers extending the system without changes to the core engine
- Making the system easy to install and learn

The main features in concept will be introduced in this subchapter, including examples and the stock UI components.

4.1.1 RPG Maker VX Ace

The earlier concept was made for RPG Maker VX Ace, previous RPG Maker version. The framework was written in Ruby.

RPG Maker VX Ace was also an RPG development engine and designed with ease of use in mind, allows a total beginner to create a complete game without knowing to programming (Perez 2014, xxiii).

Whilst Ruby was recognised by developers for its mature package management RubyGems and the web application framework Ruby on Rails, neither of them were included in RPG Maker VX Ace library.

For the extendable capability, RPG Maker VX Ace only had a part of framework opened and there was no plugin management included. All plugins were written in RPG Maker VX Ace built-in script editor and high-end developers had to configure them inside the script.

4.1.2 Stock UI components

In RPG Maker, the game is a composite of scenes, windows and sprites. For each scene, the UI components are called Windows, they take responsibility for displaying game's contents and information.

FIGURE 12. Two windows on bottom of screen (Yanfly 2016, cited 16.8.2016)

All windows in framework are derived classes from the base class Window; however, the actual code of base class is not opened for developers. They include basic properties such as position, sizes and visibility flag. Furthermore, Window class has methods to draw contents easily, allows developers to create new UI components by creating a class derived from the Window class.

It is easy to make new contents for programmers, but it is hard for most of developers to make their UI nevertheless. RPG Maker targets non-programming audiences so that mostly users do not know how to programming, and the engine does not include a graphical tool helping developers to make new contents or design their game UI. This matter leads to the concept of designing game UI without knowing to programming.

4.1.3 Configuration

The configuration was built based on Ruby's hash, a data structure that associates a value with a key (Flanagan & Matsumoto 2008, 67), and was exposed for developers to configure. The basic idea was mapping each Window with a hash that contained the properties, such as position and size, and the system would get the corresponding properties based on Window's keys.

The concept had been accepted by both developers and high-end users since announced, it allowed users finding and editing the UI's properties without knowing to programming, each key represented the meaning of property and users could find the default value as an example to configure.

In later concept edition, the configuration was made to be programmable, allowed advanced users using Ruby code to make more complex options for their UI. Furthermore, the system introduced an easy-to-learn language based on Ruby's array which could be used to edit existed UI and create new components.

```
module MainMenu
    BACKGROUND = [
        ["$horgrad[0, 0, 0, 255, 255, 255]", 255, 1, [2, 0]],
        ["$vergrad[0, 0, 0, 255, 255, 255]", 255, 1, [0, 2]],
    ]
end
```

FIGURE 13. The custom language to edit and create new contents

```
#==============================================================================
# ■ Setup Data for Window Command and Window HorzCommand
#==============================================================================

module NewScene

    # ------------------------------------------------------------------
    # Setup Commands List for :encyclopedia_command window
    # ------------------------------------------------------------------
    COMMANDS[:encyclopedia_command] = {
        :items   =>  "Items",
        :weapons =>  "Weapons",
        :armors  =>  "Armors",
    } # End COMMANDS

end

#==============================================================================
# ■ Setup Data for Window Selectable
#==============================================================================

module NewScene
    module Data

        # ------------------------------------------------------------------
        # Setup data for :encyclopedia_list window
        # ------------------------------------------------------------------
        def encyclopedia_list_data
            case get_ext(:item_kind)
            when :items
                $data_items.compact
            when :weapons
                $data_weapons.compact
            when :armors
                $data_armors.compact
            else
                []
            end
        end

    end # End Data
end
```

FIGURE 14. Programmable configuration for creating new UI

The configuration in figure 14 handles the behaviours for the user interface, tells the system getting data based on user's command.

4.1.4 Extensibility

In addition to programmable configuration, the concept introduced new APIs for developers to create their own plugins for UI system. Many parts of stock UI components were modified to be more extensible, some hidden parts of the system were also exposed to developers.

General UI components were also introduced in later concept which allowed developers to focus on drawing contents instead of creating new classes derived from the base Window class and adding them manually into each scene. This idea shortened the UI design time and gave developers ability to save and share their own UI components, lessened compatibility issues among UI plugins.

Furthermore, the system has been programmed in MVC pattern, allowed developers extending each part of the UI independently. This pattern separated the data, which was presented as Model, with the UI components, which were presented as Views, and gave the third party developers the ability to make their plugins working with UI system easily.

4.1.5 Ease of installation

The UI system was intended to be released as a plugin, since changing the code base would cause developers trouble updating their plugins. Furthermore, RPG Maker VX Ace did not include an auto-update engine, the framework would not be updated automatically so that users had to copy and paste the update themselves into their projects.

As a consequence, the UI system would be released as a plugin that required copy and paste into the project, allowed users focusing on configuring the UI instead of wasting time on setup.

4.2 Limitations

4.2.1 Ease of version updating

Each time updating a new feature for the UI system, the configuration might need to be updated. However, the configuration was a script itself so it could not be updated automatically by just updating the system. Users found it was hard to find and put new properties into the configuration, there was no way to copy and paste the new configuration structure without overwriting current settings.

There was an idea to put the new properties of configuration into new script files and merge them into the old configuration; however, it would lead to a tricky situation where the configuration files would flood other script files, causing many troubles to managing the code and plugins.

This problem also caused trouble in sharing the UI settings among users when they used different versions of UI system. The user who frequently updated the system would find that many shared settings were outdated and they would cause serious game crash without manually updating the configuration.

Consequently, the system got little update and only a few new features were introduced since released. Users found it was hard to get the new features to work with their existed configuration since there was no other way to get configuration updated automatically.

4.2.2 Ease of configure and import settings

Whilst using Ruby's hash was a fast and simple way to load the settings, it still had many drawbacks besides the version updating.

Although the keys in settings hash were self-explaining, users who was not familiar with code and programming found a hard time when configuring the UI. For number type options, users had no problem in settings; however, for options required a choice, for example background type, users sometimes got a typo and the UI system crashed.

Furthermore, users always had a hard time to configure their UI design since each time they changed settings, the game had to be reloaded to preview the changes. The scripts in RPG Maker VX Ace were loaded in initialization, including the configuration, so making the system reloading configuration in real-time was impossible and the UI preview could not be implemented.

For save and import settings, there was no easy way to backup and import some settings in the configuration. For example, if user want to save their main menu, they have to copy the hash of main menu configuration and store it somewhere and replace them into the system configuration later. In consequence, users in community had a hard time to share their UI with each other.

4.2.3 Performance

The system has been criticized the performance since released. In many cases, the system had short lags for about 2 seconds each time UI was drawn on screen, hindered playing experience since RPG games changed scene frequently. In worst case, the lag would persist until changing the scene or menu, caused dropping fps.

On programming language side, Ruby has been considered as a slow language according to the benchmark compare to other languages. The slowness had an impact on UI settings iteration and it would cause the lag when changing scene and menu. Furthermore, using hash for configuration was not the best performing idea since iterating hash cost much hardware usage, especially CPU. Therefore, the system had to trade performance for users' ease of configuring.

On system wise, RPG Maker VX Ace used GDI+, a Microsoft Windows API that enabled application to draw graphics and text on display (Microsoft Corporation 2012, cited 28.8.2016), and did not support hardware acceleration. The renderer relied on only CPU, which caused the bottleneck problem on hardware since graphics and logic were both processed by CPU. In complicated scene like battle scene, the continuous update for animation and the player status required expensive process, player would experience some lags through the play especially on old hardware. Furthermore, on start of each scene, both rendering

and configuration reader worked on the same time, the bottleneck caused a small to significant lag each time the scene changed. Consequently, reducing lag for system was nearly impossible since system was built on top of GDI+ and the engine developer company had no intend to build RPG Maker VX Ace with other multimedia library since it would cost much trouble and money.

5 NEW DEVELOPMENT

5.1 Technology Improvements

In the new development, many new technologies and tools have been presented and introduced in Chapter 2. As new technologies involve, many parts in concept has been changed and improved.

5.1.1 Using JSON for Configuration

JSON (JavaScript Object Notation) is a lightweight data-interchange format which is derived from the ECMAScript programming language but is programming language independent, JSON is also a text format that is easily parse by any programming language and machines (ECMA International 2013, cited 10.9.2016).

```
{
    "Image": {
        "Width":  800,
        "Height": 600,
        "Title":  "View from 15th Floor",
        "Thumbnail": {
            "Url":    "http://www.example.com/image/481989943",
            "Height": 125,
            "Width":  100
        },
        "Animated" : false,
        "IDs": [116, 943, 234, 38793]
    }
}
```

FIGURE 15. JSON format sample (Internet Engineering Task Force 2014, cited 10.9.2016)

With the native fast parser JSON library in JavaScript programming language which is used in RPG Maker MV and the file system in Node modules, the configuration is easily to be stored and shared in a single text file. Furthermore, the file system supports reading and writing an existed configuration file, helping the system updating the configuration, adding new options without any problem.

Saving data in JSON format also allows third party developers developing tools and add-ons in another programming languages and libraries since most of popular programming languages natively support or have library parsing JSON. Compare to Ruby's hash which was used for configuration in previous UI system, JSON is programming language independent, allows the future developments read and use existed data structure.

Saving data in an external file also supports real-time developing and editing the UI system. Developers and users can easily change the UI options, add new features and new UI components whilst testing and see the changes in-game. The statement is the base for the configure tool concept, allows developing a WYSIWYG editor for UI.

This improvement helps the system easily updating existing configuration, supports backward compatibility that are not possible in previous UI system. Data saving in JSON format is also lightweight, makes it easily to be shared between users and developers, and allows the system importing and exporting configuration without any manual edit.

5.1.2 Version updating

In previous implementation, users had some troubles in updating new version for the UI system. For updating the code, users had to manually download the code from official website and paste into the engine. Furthermore, there might be breaking changes that required users to copy and paste new options to their existing configuration.

In current development, with supports from Node.js and its package manager – npm, version updating and controlling is easier than ever. For updating the code base, users will have many options to do this.

The first option is using npm, the package manager for JavaScript that allows find, share and reuse packages of code from developers over the world (npm Inc. 2016, 10.9.2016), users can update their plugins by single command line.

```
npm install
```

FIGURE 16. Command to update packages by using npm

Alternatively, users can use Bower, a package manager for the web that always keeping track of packages and making sure they are up to date (Bower 2016, cited 10.9.2016), to install and update plugins and packages for their projects. Even though both npm and Bower are a little advanced for non-programming users because of the command line in uses, they are good ways to manage and control the version of system. Users can easily control what version they would implement to their projects and install the system with a single command.

The second option is connecting to official server by using HTTP request and automatically saves to their projects on local disk. The easiest way is writing build number in the code and embedding a HTTP request to server to get current build number and use file system to write new code on disk. This method is better for most of users since it will automatically connect to server each time test game. Users with a limited internet bandwidth can also check new version manually by running a command in system.

Besides the version control for code base, the best improvement in new development is automatically integrating new features without any breaking change in the code and in the system. In previous version, the system had a hard time to deal with configuration since the Ruby implementation in RPG Maker VX Ace would not allow developers importing gem packages and was limited in file system and data format library. In new development, with JSON and file system supports, UI system will integrate new options to configuration and patch any existing options.

```
namespace LunaEngine.Helpers {
    export function isObject(object: Object): boolean {
        return object != null && typeof object == "object"
&& !isArray(object);
    }

    export function isArray(object: Object): boolean {
        return object != null && object instanceof Array;
    }

    export function deepExtend<T>(target: T, ...sources: Object[]): T {
        for (let source of sources) {
            if (source == null) {
                continue;
            }

            for (let key in source) {
                let src = target[key];
                let copy = source[key];

                if (isArray(copy)) {
                    let clone = src && isArray(src) ? src : [];
                    target[key] = deepExtend(clone, copy);
                    continue;
                }

                if (isObject(copy)) {
                    let clone = src && isObject(src) ? src : {};
                    target[key] = deepExtend(clone, copy);
                    continue;
                }

                target[key] = copy;
            }
        }

        return target;
    }
}
```

FIGURE 17. Deep extend, a helper to integrate new configuration structure to the previous ones

FIGURE 17 shows the code of helper for integrate new configuration structure to the existing configuration. In summary, the system will do deep iteration on the new structure and copy any new couple of key-value into the configuration in user's project.

5.1.3 JavaScript and TypeScript as main programming languages

In RPG Maker MV, its libraries and framework are written in JavaScript, whilst the UI system will be implemented in TypeScript language which is a superset of JavaScript and compiled to JavaScript. TypeScript inherits all advantages of JavaScript whilst having its own strong features. This chapter will cover the improvements of JavaScript over Ruby and the advantages of using TypeScript.

According to TIOBE Index for August 2016, JavaScript is currently in top 10 popular programming languages (position 7) whilst Ruby is sitting in position 12. This popularity index proves that JavaScript is more popular, widely known by many programmers, and has more documents, tutorials and libraries than Ruby. On the other hand, the rise of Node.js and the web technologies allows the performance and libraries in JavaScript always improving and updating and the development would be much easier with the vast number of JavaScript libraries.

On performance wise, the JavaScript engine V8 developed by The Chromium Project which is used in RPG Maker MV engine, has been praised for its high-performance whilst Ruby has been being criticized for its terrible performance.

fannkuch-redux					
source	secs	KB	gz	cpu	cpu load
JavaScript V8	**74.57**	11,060	539	74.54	1% 0% 0% 100%
Ruby	2,350.70	7,048	384	2,350.01	1% 0% 0% 100%
n-body					
source	secs	KB	gz	cpu	cpu load
JavaScript V8	**33.95**	11,936	1287	33.94	1% 0% 0% 100%
Ruby	690.08	7,072	1137	689.90	1% 0% 0% 100%
regex-dna					
source	secs	KB	gz	cpu	cpu load
JavaScript V8	**3.45**	642,500	405	3.44	0% 1% 2% 100%
Ruby	24.33	145,900	442	24.32	0% 1% 0% 100%

FIGURE 18. JavaScript V8 versus Ruby Benchmarks (The Computer Language Benchmarks Game 2016, cited 5.9.2016)

The above benchmark retrieved from The Computer Language Benchmarks Game shows the superior speed of JavaScript V8 in some benchmark algorithms.

The change in programming language also comes along with the ability to load external libraries. In last version, the mini Ruby interpreter did not allow any external library whilst in new engine, the JavaScript engine allows loading all node modules and many web libraries. Many libraries are included in the development, such as Angular.js 2 and Pixi.js v4, also the File System node module is used regularly.

As for TypeScript, chapter 2.5 already mentioned about the advantages. On the other hand, TypeScript has better object-oriented programming experience than both Ruby and JavaScript, making it more scalable and easier to maintain in large project.

5.2 Developing features

This sub-chapter mentions about the problems and solutions for main features during development, includes all techniques and technologies uses in each solution.

5.2.1 Responsive and dynamic UI

In a game, developers often allow players changing the resolution to fit the screen. Furthermore, for some scaling design in UI, developers might want to have some ways to quickly setup position and size for UI components based on screen size. These needs raise up the problem of setting relative position and size based on screen size and how to setup position from the right or bottom screen instead of from the left and upper screen.

For a quick solution and an easy one for users to setup, the system will allow choosing between fixed or percentage values, where percentage ones are based on screen size. This method helps developers avoid unnecessary evaluate (programming eval) or a string parser using regular expression which are slow and dangerous when allowing users putting their expressions freely. Furthermore, the system will include options for alignments, which will allow users positioning UI components from the left, centre or right side of screen. In this solution, the configuration will include new properties to tell the system how values are calculated. Even though adding new properties would make the configuration structure more complicated, this solution keeps performance not dropping and safer for users.

```
namespace LunaEngine.DataStructure {
    export enum Align {
        Left,
        Right,
        Top,
        Bottom
    }

    export enum TransformType {
        Fixed,
        Percentage
    }

    export interface PositionValue {
        align: Align;
    }

    export interface TransformValue {
        type: TransformType;
        value: number;
    }

    export interface Window {
        width: TransformValue;
        height: TransformValue;

        horizontalPosition: PositionValue & TransformValue;
        verticalPosition: PositionValue & TransformValue;
    }
}
```

FIGURE 19. Data structure for UI component (Window) position and size

However, for more advanced UI designs, users might want many complicated calculations, also for the need of creating and modifying contents, a dynamic UI is needed. For a dynamic UI, users have access to the system APIs and complicated expressions. Basically, the system allows users putting the code into configuration with simpler calls, also there are some limits in allowing functions and expressions so that users will not accidently mess up the system.

Whilst putting some lines of code into program is easy, writing JavaScript or TypeScript code into JSON data file is nearly impossible since JSON only accepts basic data types such as string, number or array. In this case, the system needs either eval expressions or string parser to interpret the code read from JSON data.

In earlier solution, simple eval expressions were used for dynamic values and achieved a great success. However, many testers reported a huge drop in performance when using for a complicated UI where multiple dynamic values were used in a scene. After some researches, eval expressions were figured out as the reason causing poor performance and a new way to do dynamic evaluation was used instead.

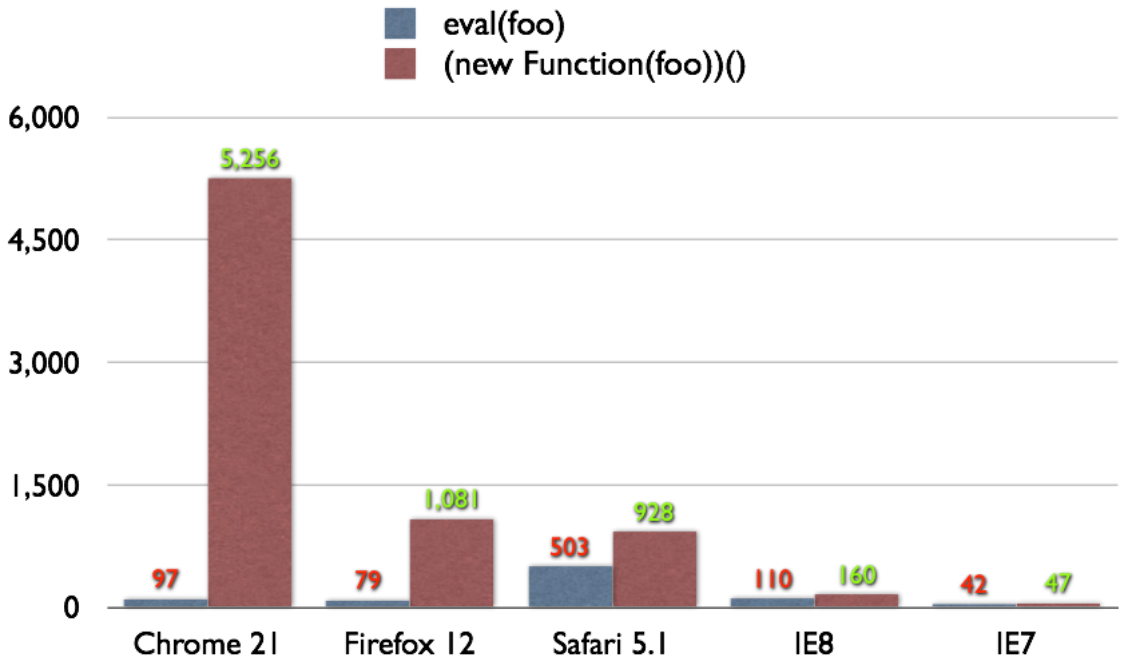

FIGURE 20. Benchmark between eval() and (new Function())() (Grisogono 2012, cited 10.9.2016)

According to Grgur Grisogono on Modus Create, using an anonymous function to evaluate values is far better than eval expression whilst achieving the same results. However, creating many anonymous functions would cause trouble for garbage collector and might be some memory leaks since many functions were created and called, without any reference, it would over-work garbage collector. In this problem, caching anonymous functions for dynamic update were introduced. Each time an anonymous function was created for dynamic value evaluation, it would be stored in an array and called when the evaluation was needed.

```
GUI.prototype._evalCondition = function(config) {
    var result = JSON.parse(JSON.stringify(config)),
        conditionals = result.conditional,
        cond,
        evaluate;

    if (!conditionals) {
        return result;
    }

    for (var i = 0; i < conditionals.length; i++) {
        cond = conditionals[i];

        if (!this._conditions[i]) {
            this._conditions[i] = new Function(
                'return ' + cond.condition + ';'
            ).bind(this);
        }

        evaluate = this._conditions[i];

        if (!evaluate()) {
            continue;
        }

        for (var key in cond.properties) {
            result[key] = cond.properties[key];
        }
    }

    return result;
};
```

FIGURE 21. How dynamic values are evaluated by using anonymous functions

In case of less advanced evaluations, a string parser to interpret the values is also introduced. For example, when users want to get the width of its container: “100% - 200” pixels, the system would interpret the “100%” as 100% of container’s width and minus by 200 pixels then return the value as a number, or when users want a formatted text “@hp Health Points”, “@hp” would be interpret into character’s health points and return the value as a string. In this solution, regular expression is used.

```
function interpretString(str: string): string {
    let reg = /\@(\w+)/g;
    let matches = str.match(reg);

    for (let match of matches) {
        switch (match) {
            case "@hp":
                // value for Health Points
                break;
            case "@mp":
                // value for Mana Points
                break;
            default:
                // no matching value
        }
    }

    return str;
}
```

FIGURE 22. Using regular expression to interpret string

In conclusion, finding a way to interpret the dynamic values is the most basic point in UI development since users always demand for more advanced functions and the needs for some complicated UI designs. For creating and modifying contents, an interpreter for contents is also needed so that the system is kept high performing and stable.

5.2.2 Configuration tool and real-time configuration

As more features has been added into the system, the configuration is more complicated and hard to manually configure, especially for high-end users. In this problem, a configuring tool is introduced, gives users ability to configure without opening JSON file whilst having full access to all features.

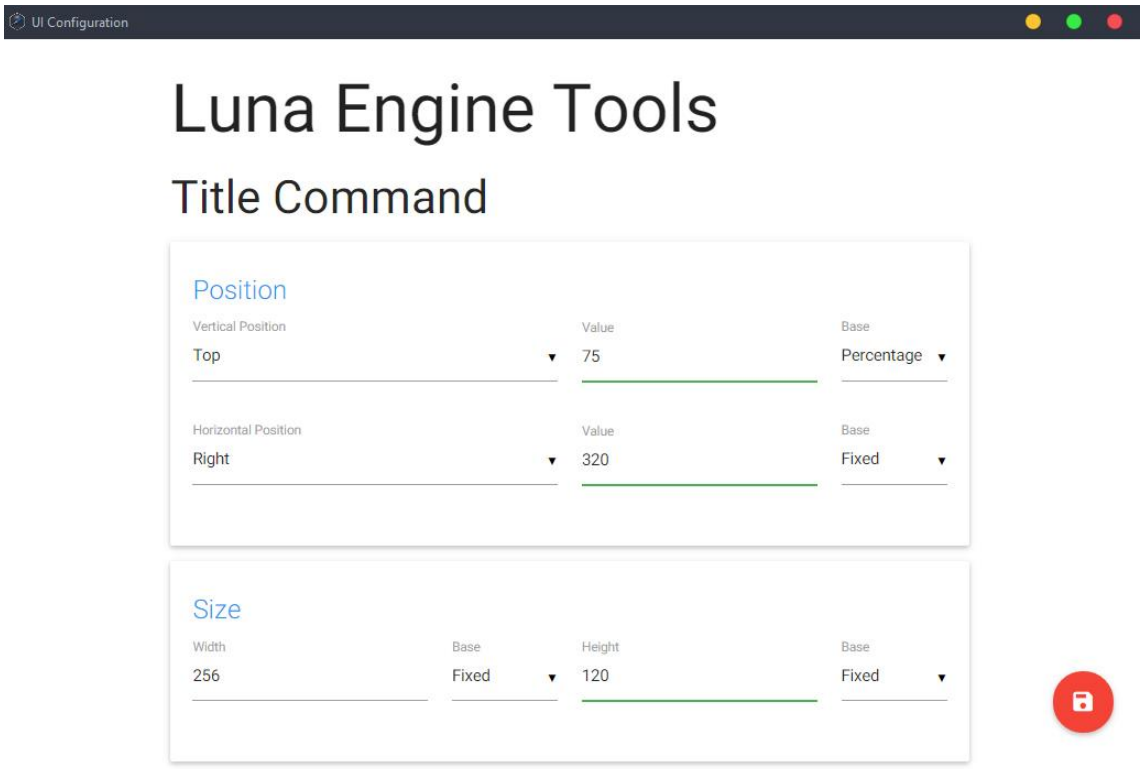

FIGURE 23. Configuration tool prototype

To make a configuration tool, it has to be accessible to JSON data format and to the game testing window data so that it can be real-time configured. Fortunately, Nw.js allows an instance of application to call another windows with different entry page and have full access to the main instance and can edit game data in real-time.

```
function OpenLunaDev() {
    let gui = require("nw.gui");
    let configWindow = gui.Window.open('./Tools/index.html', {
        width: 992,
        height: 620,
        toolbar: false
    });
    let window = gui.Window.get();

    configWindow.on("loaded", function() {
        configWindow.window.lunaWindow = window.window;
        configWindow.showDevTools();
    });
}
```

FIGURE 24. Call new window with entry point is the tool index.

In this solution, the game instance will contain the current configuration, and the tool which is called from game instance will access and change that configuration, also request an update to the engine so that users can see the changes instantly. To do this, Angular.js 2 is chosen for data binding and doing the single page app since both game and tool are web browser based.

```
<input placeholder="Position Value" id="vertical-number" type="number"
class="validate" [(ngModel)]="config.verticalPosition.value"
(ngModelChange)="requestUpdate()">
```

FIGURE 25. Binding input and configuration

Using Angular.js 2 has an advantage of type-safe since it is built on TypeScript, allows the tool read data structure from the UI system easily. Furthermore, binding data in Angular.js 2 is mostly written in the HTML templates, allows the tool being further extended as more features are added without messing the core application. The single page strategy is also used to make the tool friendlier and reduce delay time when loading a new settings list.

```
import { Injectable } from '@angular/core';

@Injectable()
export class LunaService {
    getLunaScenes() {
        let data = (<any>window).lunaWindow.LunaEngine.DataSymbol.Scenes;
        return
Promise.resolve<LunaEngine.DataSymbol.SceneRecord[]>(data);
    }

    getLunaWindows(sceneId: number) {
        let data = (<any>window).lunaWindow.LunaEngine.DataSymbol
            .getWindowsBySceneId(sceneId);
        return
Promise.resolve<LunaEngine.DataSymbol.WindowRecord[]>(data);
    }

    getLunaWindowConfig(sceneId: number, windowId: number) {
        let data = (<any>window).lunaWindow.LunaEngine.DataSymbol
            .getWindowConfigById(sceneId, windowId);
        return
Promise.resolve<LunaEngine.DataStructure.WindowAllConfig>(data);
    }
}
```

FIGURE 26. Angular.js 2 service and type-safe for configuration

On above figure, the configuration and data are requested from main game instance with the types defined in the UI system, allows the tool binds and checks data structure without any trouble.

For saving the configuration into a JSON file, File System module is used since Nw.js allows developers implementing node modules into the web application. Whilst having access to both file system and the engine structure, the tool has features to import external configuration files and gets some or all settings from that, and to export current configuration to a JSON file to backup or share with other users in community.

In conclusion, the basic idea of configuring tool implementation is the ability to bind real-time data between the tool and the game instance whilst retaining the type-safe so that the system will not be polluted with invalid values.

5.2.3 Create and modifying UI contents

In RPG Maker MV, there are already base components for creating UI, such as window, which draws icons, texts and handling inputs. However, to actually create and add a window to the scene or modify existing contents, users should learn about JavaScript, programming and the engine API to create a derived window class and draw contents on it.

The best solution for this is creating a data structure that contains window properties and let the system to read and create a window from that. Fortunately, JavaScript is a prototype-based programming language, where objects serve as prototypes, allows creating new object derived from existed object, adding and modifying properties freely. Therefore, the program creates a new blank window, iterates all data properties and set them to the window, then it will parse all values and draw contents onto the screen.

```
interface WindowDatabase {
    [index: string]: {
        // data structure for window properties
    }
}

function createWindows(config: WindowDatabase): void {
    for (let index in config) {
        let data = config[index];
        let window = new Window();
        window.uid = index; // unique id for window
        for (let property in data) {
            window.setProperty(property, data[property]); // safe
checking
        }
        SceneManager.currentScene.addChild(window); // add window to
scene
        window.refresh(); // refresh window
    }
}
```

FIGURE 27. Pseudo code to create UI components base on configuration

The prototype-based programming also allows adding new properties to UI component without messing the core engine, allows advanced users creating formulas and conditions for the dynamic UI. For example, to create a stacked layout which sorts the UI components after each other in an index order, an

user can add a new property ”index” to windows and let the dynamic UI handles the position based on ”index”. On the other hand, this programming pattern allows users creating component templates so that they can be used in future components without writing duplicated code.

For modifying existing components or third-party components, the system needs a unique name for each component to get properties for them in configuration. However, JavaScript and TypeScript does not allow getting base class name like Ruby. To solve this problem, each component needs to be named so that it can be searched by the engine to match with configuration. Furthermore, to allow the configuration tool getting the UI components list, a dictionary for UI components is introduced.

The dictionary will be an array contain both names and references to components. Each time the system initializes, it will iterate through configuration and find matching component names in the dictionary. In this way, the system will both find the components and put the properties it gets in configuration and the components will be refreshed when added into the scene. Furthermore, the configuration tool can iterate through the dictionary and show available components to be configured.

```
namespace LunaEngine.DataSymbol {
    export enum WindowType {
        Single,
        Selectable,
        Command
    }

    export interface WindowRecord {
        id: number;
        symbol: string;
        klass: any; // windowClass
        type: WindowType;
    }
}
```

FIGURE 28. Structure for the dictionary

The other problem is to manipulate components based on other components and bound data on the scene. To solve this problem, all components should be

added to an UI container which has references to both data and all included components and for the components, they should be two-way bindings so that even children components can access to their parents.

In this solution, the dynamic UI will evaluate the values based on other components or based on its parent (or container). For example, in a scene where it shows status of the whole party, it will be bound with the party data, and the UI components will iterate the party members to draw their properties. To do this, a component should have access to current scene data, also they might want to get access to its parent properties for dynamic UI.

```
{
    "sceneStatus": {
        "data": "$gameParty.members",
        "components": [
            {
                "type": "StackedDataLayout",
                "data": "@parent.data",
                "template": [
                    {
                        "type": "Text",
                        "text": "Health Points",
                        "fitSize": true
                    },
                    {
                        "type": "Text",
                        "text": "@currentData.hp",
                        "fitSize": true
                    }
                ]
            }
        ]
    }
}
```

FIGURE 29. Using bound data with dynamic UI

In case of draw UI components onto the scene, such as drawing texts and images, the Pixi.js will handle the renderer wonderfully. However, each kind of UI components will have different properties, they will have their own classes to handle what will be drawn and processed, whilst having their unique names to get references for configuration. For example, in above figure, there are component type StackedDataLayout which will iterate through a data array and draw

them respectfully, and the type Text which will draw a text onto screen. Users will have ability to create new types based on existed types, such as a button which will include an image and a text, and use them as template for future uses.

In summary, whilst having the renderer being handled by Pixi.js, the most important problem is organizing the configuration whilst having references to system data and renderer. The flow of the system will be reading configuration, iterating them and matching the component with the settings whilst the renderer processes the properties and draws them onto screen.

6 CONCLUSION

In game development, a user interface describes the game's state and how players interact with the game. Making a user interface system is important for a game engine to maintain the interaction between the systems and players, allows developers developing further in features without having much trouble in implementing user interfaces.

The thesis work introduced the theory and concept in user interface system development. The results and improvements are summarised in following table.

	Old UI System	New UI System
Install and Update abilities	• Easy to install. • Update must be done manually. • Hard to update, especially with configuration.	• Multiple ways to install. • Update can be done automatically. • New features and settings are integrated automatically.
Performance	• Without dynamic UI, the performance was good. • Performance terribly dropped in complex UI.	• The performance is good overall. • In complex UI, performance is improved in exchange for memory.
Configuration	• Configuration was written in Ruby's hash format. • Must be configured in code, there was no configuration tool.	• Configuration was written in JSON format. • Can be configured by a text editor or using the configuration tool. • Easy to configure, there are supports

	• Hard to configure, especially with dynamic UI. • Configuration was stored in code, could not be exported or imported.	from configuration tool. • Configuration is stored in an external JSON file, can be exported and imported.
Systems	• Written in Ruby, hard to maintain the structure. • Dynamic UI used eval to evaluate values that resulted in poor performance.	• Written in TypeScript, static typing and interfaces allows maintaining structure when extending system. • Dynamic UI uses anonymous functions to evaluate values that costs memory whilst giving better performance.

Despite the clearly advantages of the mentioned user interface system development, there still exists areas for continued development. The configuration structure and tool can be further extended to enhance user's experience in game development, such as allowing developing features without knowing to programming or supporting third-party plugins configuration.

The outcome of this work is a user interface system for a game engine which is easy to configure and implement any kind of design into the game and costs little performance. The project shows how a game's user interface works and how to apply modern web technologies in game development.

REFERENCES

Adams, E. 2009. Fundamentals of Game Design. 2nd edition. United States: New Riders, 453 – 455.

Angular 2009. Features & Benefits. Cited 14.9.2016, https://angular.io/features.html

Berry, J. 2013. Let's Spec Into Talent Trees: A Primer for Game Designers. Cited 2.9.2016, http://gamedevelopment.tutsplus.com/articles/lets-spec-into-talent-trees-a-primer-for-game-designers--gamedev-6691

Bower 2016. Bower – a package manager for the web. Cited 10.9.2016, https://bower.io/

Cavanaugh, R. 2013. Walkthrough: Interfaces | TypeScript. Cited 4.8.2016, https://blogs.msdn.microsoft.com/typescript/2013/01/24/walkthrough-interfaces/

DataArt 2014. What is TypeScript? Pros and Cons. Cited 1.9.2016, http://designmodo.com/typescript/

Degica Co., Ltd 2015. RPG Maker MV. Cited 5.8.2016, http://www.rpgmakerweb.com/products/programs/rpg-maker-mv

Degica Co., Ltd 2015. RPG Maker MV on Steam. Cited 5.8.2016, http://store.steampowered.com/app/363890

ECMA International 2013. Standard ECMA-404: The JSON Data Interchange Format. Cited 10.9.2016, http://www.ecma-international.org/publications/standards/Ecma-404.htm

Flanagan, D. & Matsumoto, Y. 2008. The Ruby Programming Language. United States: O'Reilly Media, 67.

Freeman, A. 2013. Pro ASP.Net MVC 5. New York: Apress, 67.

GoodBoyDigital 2015. pixi.js – Basic. Cited 7.8.2016, http://pixijs.github.io/examples/

GoodBoyDigital 2015. Pixi.js – Super fast HTML 5 2D rendering engine that uses WebGL with canvas fallback. Cited 7.8.2016, https://github.com/pixijs/pixi.js

Gregory, J., Lander, J. & Whiting, M. 2009. Game Engine Architecture. United States: CRC Press, 11.

Grisogono, G. 2012. JavaScript Performance Tips & Tricks. Cited 10.9.2016, http://moduscreate.com/javascript-performance-tips-tricks/

Internet Engineering Task Force 2014. RFC 7159 – The JavaScript Object Notation (JSON) Data Interchange Format. Cited 10.9.2016, https://tools.ietf.org/html/rfc7159

Kanian77 2008. The Benefits and Drawbacks of using The MVC Pattern. Cited 15.9.2016, https://kanian77.wordpress.com/2008/09/03/the-benefits-and-drawbacks-of-using-the-mvc-pattern/

Microsoft Corporation 2012. GDI+ (Windows). Cited 28.8.2016, https://msdn.microsoft.com/en-us/library/windows/desktop/ms533798(v=vs.85).aspx

Microsoft Corporation 2012. TypeScript – JavaScript that scales. Cited 4.8.2016, http://www.typescriptlang.org/

Microsoft Corporation 2016. Interfaces – TypeScript. Cited 4.8.2016, http://www.typescriptlang.org/docs/handbook/interfaces.html

Microsoft Corporation 2016. Model – View – Controller. Cited 15.9.2016, https://msdn.microsoft.com/en-us/library/ff649643.aspx

Nizet, J. 2016. Ninja Tips 2 – Make your JSON typed with TypeScript. Cited 2.9.2016, http://blog.ninja-squad.com/2016/03/15/ninja-tips-2-type-your-json-with-typescript/

npm Inc. 2016. npm. Cited 10.9.2016, https://www.npmjs.com/

Nw.js 2015. Nw.js – Call all Node.js modules directly from DOM and enable a new way of writing applications with all Web technologies. Cited 18.8.2016, https://github.com/nwjs/nw.js

Nw.js 2016. Getting Started with NW.js. Cited 14.9.2016, http://docs.nwjs.io/en/latest/For%20Users/Getting%20Started/

Overhaul Games 2012. Baldur's Gate: Enhanced Edition. Cited 2.9.2016, https://www.baldursgate.com/

Palmer, N. 2016. RPG Maker MV 1.3.0. Cited 15.8.2016, http://blog.rpgmakerweb.com/announcements/rpg-maker-mv-1-3-0/

Perez, D. 2014. Beginning RPG Maker VX Ace. New York: Apress, xxiii.

Rollings, A. & Adams, E. 2003. Andrew Rollings and Ernest Adams on Game Design. United States: New Riders, 347.

Schultheiss, D. 2007. Long-term motivations to play MMOGs: A longitudinal study on motivations, experience and behavior. DiGRA, 344.

The Computer Language Benchmarks Game 2016. JavaScript V8 vs Ruby. Cited 5.9.2016, http://benchmarksgame.alioth.debian.org/u64/compare.php?lang=v8&lang2=yarv

Usability.gov 2016. User Interface Design Basics. Cited 16.9.2016, https://www.usability.gov/what-and-why/user-interface-design.html

Yanfly 2012. Visual Battlers | Yanfly Channel. Cited 16.8.2016, https://yanflychannel.wordpress.com/rmvxa/battle-scripts/visual-battlers/