Tile-based Game Plugin for Unity Engine

Salhazan Nasution[1], Arbi Haza Nasution[2] and Arif Lukman Hakim[1]
[1]*Department of Informatics Engineering, Universitas Riau, Pekanbaru, Indonesia*
[2]*Department of Informatics Engineering, Universitas Islam Riau, Pekanbaru, Indonesia*

Keywords: Unity, Plugin, Tile-Based Game, Level Editor.

Abstract: Nowadays video games have become mainstream in the modern lives of people in the world. Along with that, the game development process has been dramatically improved by the emergence of free to use game engines. One of the most used game engines worldwide is Unity. Unity provides strong features to support its developers, one of them is the ability to use plugins. Meanwhile, tile-based games are also very popular. Without support from any kind of plugins, development of tile-based games will increase development time, especially in level editing process. Readjusting each tile to a perfect position while it is being added to Unity scene view is very time-consuming. This plugin aims to solve this problem by extending the Unity editor script. This plugin provides support for creating and deleting tiles, enables the developers to cut the time needed to create levels in tile-based games.

1 INTRODUCTION

The popularity of video games has increased since the first emergence of video games around the 1950s. Video games have always been a component of pop culture and being the center of the multi-billion dollar entertainment industry. Nowadays because of the internet, the availability of game development software has been increasing, along with the ease of game ready device distributions and game distributions itself, encouraging advancement in the game development industry (Bergonse, 2017). Game development software, usually called game engine is a tool designed to reduce budget, complexity, and time needed to market in a game development process. This software creates an abstraction layer above the main functionality of creating a video game. This abstraction layer is packed with the tools that are designed to function as the functional components that can be modified or added with an addition of third party component (Halpern, 2018).

Game engines have been popular than before. The most popular game engine with the number of developers is Unity, with more than 5,5 million developers listed in 2016, then followed by Unreal Engine, that has reached more than 4 million developers in 2017 (Salomão et al., 2019).

Unity is a game engine that gives huge benefits compared to the other game engine that is available on the market nowadays. Unity gives drag and drop capability on its visual workflow and supports scripting in C# language, which is very popular. Unity has long been supporting 3D and 2D graphics; also Unity provides a set of tools for these two types of graphics that is always improving and always been very easy to use on each update. Unity also made especially to support developers using plugins from third party software. Unity also provides its asset store that has so many plugins for Unity itself, made from developers and made for developers (Halpern, 2018).

The plugin itself consisted of many things, among them are assets like 3D models, 2D sprites, textures, materials, sound effects, music, scripting, particle effects, and many more. All of them are available in Unity asset store, from the free to paid ones.

Beside of that, tile-based system for making games have been used widely, thus becoming a de facto for a standard approach to making games for most of the game design technology. This system is not only for 2D games, 3D games could also use the same system. Tile-based systems are popular because this system solves a different amount of problems that are so complex to be solved with other ways (Spuy, 2010).

Unity engine does not provide support to make games with tile-based system, related plugins with good functionality had a paywall, and the free ones lack good functionality, so the developer has diffi-

Nasution, S., Nasution, A. and Hakim, A.
Tile-based Game Plugin for Unity Engine.
DOI: 10.5220/0009103700550063
In *Proceedings of the Second International Conference on Science, Engineering and Technology (ICoSET 2019)*, pages 55-63
ISBN: 978-989-758-463-3

culty in terms of accessibility to good tile-based system plugin. This study was done in order to make a better tile-based system than before more accessible to more Unity developers especially in terms of education.

2 LITERATURE REVIEW

The foundation of this research is the history of previous researches related to the plugin for game engines.

The first research is entitled "Uni-CAVE: A Unity3D Plugin for Non-head Mounted VR Display Systems". Unity3D has been popular and is free to use by many game developers especially for designing and constructing a virtual environment. However, Unity3D itself has not yet have an immersive projection base Virtual Reality (VR). The objective of this research is to give a free solution to adapt Unity3D with any VR immersive projection system. Uni-CAVE provides support for multiple VR display configurations, multiple stereo techniques, 3D head, and wand tracking system, and display synchronization. Uni-CAVE is a plugin that is meant to ease the development of immersive 3D VR display systems on Unity3D. This plugin has been tested on a six-sided CAVE environment, using active quad-buffered stereo, on twenty curved tiled display screen system that uses side-by-side stereo, also two projector power wall using quad buffered active stereo and dual-pipe active stereo. These configurations can be saved as Unity prefabs, allowing to be reused multiple times by developers (Tredinnick et al., 2017).

The second research is entitled "A Game Engine Plug-in for Efficient Development of Investigation Mechanics in Serious Games". Educative Serious Games (SG) has been widely used in education, training, and domain like sports and medical treatment. Regardless of the increasing interest on SGs, the distributions of SGs is constrained by the difficulty of combining the effectiveness of education with entertainment, a high resource needed to the development process, and involves different fields of study. This research proposes a development framework to support investigation in SG. This framework provides a description template which allows developers to define what items that had to be investigated in a structural way possible. The descriptor is processed on runtime by the game engine plugin which is responsible to manage the virtual environment and the corresponding gameplay. The objective of this research is to ease the development of Open World Serious Games also for knowledge domain experts that haven't had any programming experience yet. This plugin provides an XML template to easily manage global variables, choice of the game plot and the corresponding solution, data distributions in different POIs (Point of Interests), management of suspected parameters, graphic visualization of POI information with the creation of custom graphic elements, translation of user interface into different natural languages, management of graphic skins, management of the music, management of the solution of the game (one solution for each plot), generation of the help of the game, and the parsing of the XML file containing parameters of the game (Carmosino et al., 2017).

Another research is entitled "HPGE: An Haptic Plugin for Game Engines, Serious Games". The motivation behind using haptic feedback in a game is the educational benefits from the sensory and motor experience, also active exploration from the player. Besides, many VR applications have promising force feedback devices. The objective of this research is to create a plugin that allows haptic device integration within Unity3D, because Unity3D does not support haptic device integration. The features included in this plugin are drag-and-drop support, Graphical User Interface (GUI), haptic texture rendering, custom haptic force-feedback, and device logging. This plugin has been used in two serious games that have been developed to teach geometry to children (Balzarotti and Baud-bovy, 2017).

Another interesting research is "An Unreal Engine 4 Plugin to Develop CVE Applications Leveraging Participant's Full Body Tracking Data. Along with the availability of powerful open-source game engines, labs that have research in a VR field have been migrated to use game engines, which is a new development environment that is far more productive, customizable, extensible, easy-to-use interface, rich of features, and almost no budget needed, with the addition of multi-platform development. On the other side, CVE (Collaborative Virtual Environment) itself, allows two or more people to work in a virtual working environment to do work that is difficult to be done by one person. The unique feature that stands out from CVE is the ability to bring full-body movement from a person in the real world to the VR. The objective of this research is to implement a plugin that could manage full-body tracking in a multiplayer CVE network environment in Unreal Engine 4. This plugin features mainly the network communication plugin and the animation plugin. The network communication plugin is used to manage the interaction between participants in a network, while the animation plugin manages the skeletal animation of participant's 3D model in the environment (Luongo and Leoncini, 2018).

The last reviewed research is entitled "UnrealHaptics: A Plugin System for High Fidelity Haptic Rendering in the Unreal Engine". VR devices have been more affordable, such as Oculus Rift and HTC Vive. Modern game engines like Unity and Unreal also simplifies the VR development process dramatically. However, these VR devices are only limited to two types of human sense, vision and hearing. This limits a large number of people that cannot benefit from VR contents like the blinds. To overcome this problem, a haptic device is created, which is to stimulate sense of touch. Because Unreal Engine does not support haptic device, hence this study is done to create a plugin for Unreal Engine that allows Unreal Engine to support haptic devices. This plugin consists of three separate plugins that support one another. Those plugins are named HAPTICO, COLLETTE, and FORCECOMP. HAPTICO is a plugin that enables developers to use haptic devices directly from Unreal Engine 4 automatically. COLLETTE is the collision detection plugin that does the collision detection integrated with the haptic device. FORCECOMP is the plugin used to calculate force computation needed for the physical simulation of forces (O. Rudel et al., 2018).

2.1 Tile based Game

Tile has always been popular in the context of 2D game development, as the reusable terrain element, that allows bigger terrain construction, which is because of the size and management issues cannot be handled with normal bitmaps. This technique is very popular in the past, but this technique is back in demand in the context of a handheld device. Beside of that, there is a growing trend towards the revival of the 2D games with the emphasis on more populated terrain, more intelligent actions, stronger effects, and character design using 3D modeling instrument for 2D games. Tile is a tiny bitmap, usually 16x16 or 32x32 pixels, used as terrain constructing element. This technique is still used until now, and this technique offers many benefits for game developers. Lesser game storage size and faster terrain processing can be achieved because only little information is needed to be described. Another benefit from this system is if the developer wants to construct bigger terrain, developer only had to add new tiles without designing all bitmap from the beginning. Games that are using tiles as the basis graphic component within are called tile-based game (Karouzaki et al., 2007).

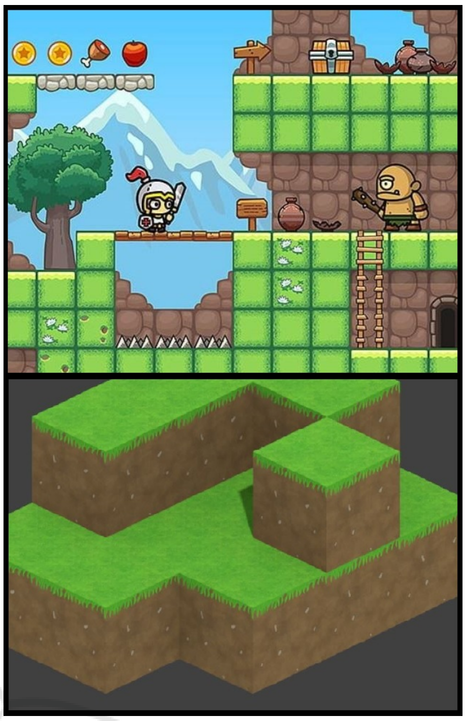


Figure 1: Tile-based games 2D (top), and 3D (bottom).

2.2 Game Engine

A game engine represents all basis of a game, providing functionality to do optimized and efficient graphics rendering, accessing file system, player inputs through input devices such as keyboard and mouse, audio player, network connectivity also saving and loading game status (Freiknecht et al., 2016).

Game engine is a framework for game development that has some core parts, the audio engine, rendering engine, and physics engine. The audio engine plays a key role in the game. If the player's character is fighting an enemy, and the mechanism uses a sword, the sword will produce a sound effect while the sword comes in contact with the enemy's sword. That part of the work is done by the audio engine. Audio engine is also used to play background music to enhance the atmosphere of the game. Rendering engine helps to choose what is actually displayed to the player. The output is a visual treat to the player while playing the game. Rendering helps to make the graphical content of the game feels alive. Physics engine helps to simulate physics in the game (Nandy and Chanda, 2016).

2.3 Unity3D

Unity (also known as Unity3D) is a game engine and an Integrated Development Environment (IDE) to make interactive media, usually video games. The Chief Executive Officer (CEO) of Unity, David Helgason describes that "Unity is a tool set that used to make games, and Unity is a technology that executes graphics, audio, physics, interactions, and networking". Unity is known by the capability of fast prototyping and big amount of publishing target platform (Karouzaki et al., 2007).

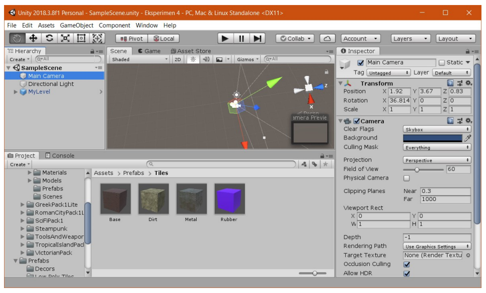


Figure 2: Unity Engine Editor Interface.

2.4 Plugin

Plugin is a binary extension unit that used for application which the architecture allows to introduce functionality to end users after application installation. Plugin is a software entity that has a close relation with components. Component-based development usually does not consider that components cannot be added in the application after installation process. Components usually used to facilitate application construction itself (Cervantes and Villalobos, 2006). Plugin is a part of the code that modifies runtime behavior. Language can be considered as a framework that provides a set of extension parts which plugins could implement new functionalities to the language. According to programming language extension aspects, there are three categories of an extension part : (1) algorithm related. (2) creation of new rules and rule execution cycle management and (3) language syntax related (Cuadrado and Molina, 2006).

2.5 Microsoft Visual Studio 2017

Microsoft Visual Studio 2017 is an integrated development environment (IDE) from Microsoft. Visual Studio is used to develop computer programs such as websites, web apps, web services, and mobile apps. The latest version of Visual Studio 2017 has full features to develop apps for Android, iOS, Windows, web and cloud (S. Durano, 2018).

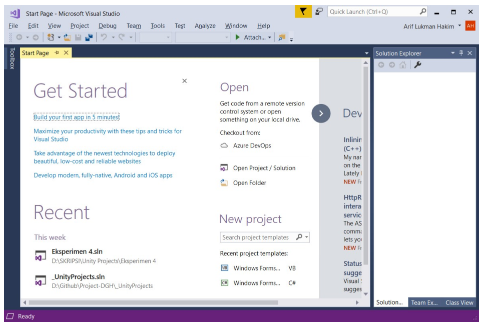

Start Page - Microsoft Visual Studio
File Edit View Project Debug Team Tools Test Analyze Window Help
Get Started
Open
Recent
New project
Eksperimen 4.sln
_UnityProjects.sln
Solution Explorer
Ready

Figure 3: Microsoft Visual Studio IDE Interface.

3 METHODOLOGY

In this section, we detail the research methodology in this research as depicted in Figure 4.

3.1 Literature Study

This step is done to get knowledge on the theory basis that are needed to do this research. Literatures that have been studied in this research is related to tile-based game, game engine, Unity3D and plugin.

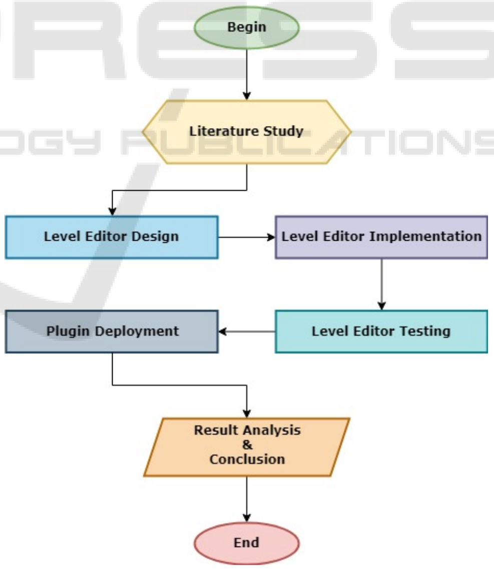


Figure 4: Research methodology.

3.2 Level Editor Design

After the preparations of tools and materials needed, the next step is to design the level editor. The level editor will be implemented using a grid system. The grid will be used to position the tiles. The level editor itself is described as in Figure 5.

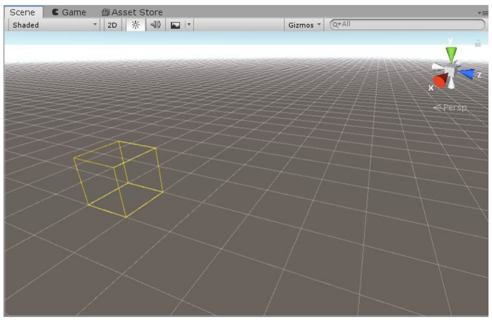


Figure 5: Grids as shown in Unity Editor.

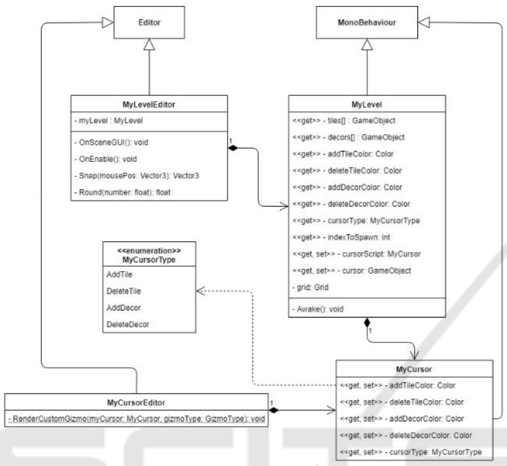

Editor
MonoBehaviour
MyLevelEditor
- myLevel : MyLevel
- OnSceneGUI(): void
- OnEnable(): void
- Snap(mousePos: Vector3): Vector3
- Round(number: float): float
MyLevel
<<get>> - tiles[] : GameObject
<<get>> - decors[] : GameObject
<<get>> - addTileColor: Color
<<get>> - deleteTileColor: Color
<<get>> - addDecorColor: Color
<<get>> - deleteDecorColor: Color
<<get>> - cursorType: MyCursorType
<<get>> - indexToSpawn: int
<<get, set>> - cursorScript: MyCursor
<<get, set>> - cursor: GameObject
- grid: Grid
- Awake(): void
<<enumeration>> MyCursorType
AddTile
DeleteTile
AddDecor
DeleteDecor
MyCursor
<<get, set>> - addTileColor: Color
<<get, set>> - deleteTileColor: Color
<<get, set>> - addDecorColor: Color
<<get, set>> - deleteDecorColor: Color
<<get, set>> - cursorType: MyCursorType
MyCursorEditor
- RenderCustomGizmo(myCursor: MyCursor, gizmoType: GizmoType): void
1
1
1

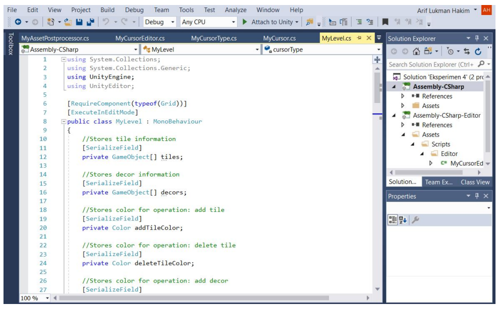
Figure 6: Level editor class diagram.

As shown in Figure 6, the main component of this system is MyLevelEditor script. MyLevelEditor script extends the Unity default Editor script to improve Unity's base editor as shown in the code in Figure 7. In this script there is OnSceneGUI() method that used to position the 3D cursor on the grid, and manages what will happen when mouse is clicked, moved, dragged, and when mouse is leaving scene view. Also, there is a Snap method that used to snap the 3D cursor to the squares in the grid. The Round() method used to round values needed to snap the 3D cursor to get the right position.

Afterward there is a script named MyLevel which is used to contain all properties that can be modified and configured by the developers using this plugin. These properties will be used in the MyLevelEditor script. In this script there is only one method, which named Awake(). This Awake() method is executed in edit mode to allow editing of properties out of game runtime.

MyLevel script needs another script named MyCursor. Game object that had MyCursor script will be the 3D cursor in the scene view. MyCursor also contains configurations for the color of the 3D cursor.

And then there is a script named MyCursorEditor that used to change the 3D cursor color as in the configurations saved in MyCursor script. The color of the 3D cursor will also change depends on the type of operation in the MyCursorType enumeration.



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

[RequireComponent(typeof(Grid))]
[ExecuteInEditMode]
public class MyLevel : MonoBehaviour
{
    //Stores tile information
    [SerializeField]
    private GameObject[] tiles;

    //Stores decor information
    [SerializeField]
    private GameObject[] decors;

    //Stores color for operation: add tile
    [SerializeField]
    private Color addTileColor;

    //Stores color for operation: delete tile
    [SerializeField]
    private Color deleteTileColor;

    //Stores color for operation: add decor
    [SerializeField]
```

Figure 7: MyLevelEditor extends Unity's editor script.

3.3 Level Editor Implementation

Level editor that has been designed before will be implemented in Unity. The level editor will be implemented by extending the script named Editor in Unity, and then adding GUI element in the Unity scene view, so the developers can add, move also delete objects that are needed in the level.

3.4 Level Editor Testing

The implemented level editor will be tested, has the functionalities run as expected. If there are deficiencies and bugs, a correction will be made so the functionalities could run as expected.

3.5 Plugin Deployment

After the testing process has done, the plugin will be deployed to game developers. Game developers could try all of the features implemented in the plugin using their own self-made asset. This plugin will be deployed with a user guide and a questionnaire. The guide will be used to inform developers how exactly to use this plugin when a questionnaire is used to get feedbacks from the developers about the performance of this plugin.

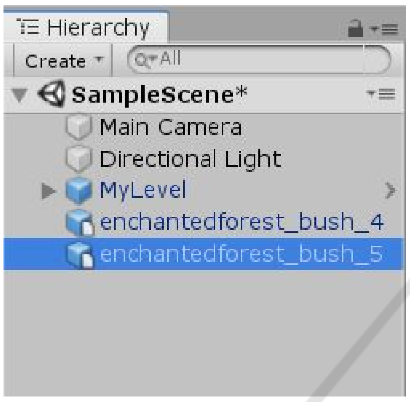
3.6 Result Analysis and Discussion

The result of this research is the questionnaires that are collected from the game developers that were using this plugin. From all the scores obtained from the questionnaire, analysis and conclusion can be made, is this plugin effective or not to speed up the process of developing a tile-based game.

4 RESULT AND DISCUSSION

4.1 Graphical User Interface

This level editor implemented using a Unity game object. This game object will be placed in a hierarchy and will appear in the scene view as shown in Figure 8. To create levels, developers have to select the level editor game object in the hierarchy by clicking it. Afterward the level editor can be used freely in the scene view.



SampleScene*
Main Camera
Directional Light
MyLevel
enchantedforest_bush_4
enchantedforest_bush_5

Figure 8: Level editor object named MyLevel in the hierarchy.

This game object is called MyLevel. MyLevel has a child object named Cursor. This Cursor object is used to be a 3D cursor that will appear in the scene view to help level editing process. Besides the 3D cursor, the grid will appear to help game developers to see the positions that tiles could be added onto. The position of the tile will be placed exactly above the square of the grid as shown in Figure 9.

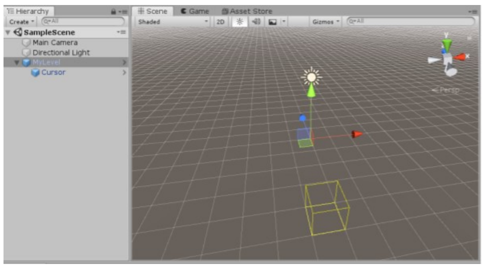
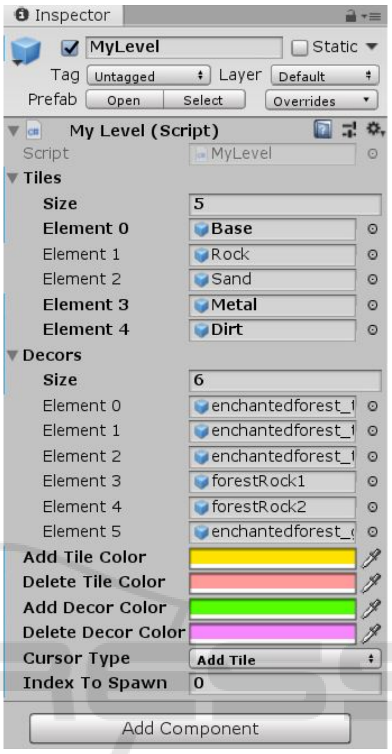


Figure 9: Level editor display in the scene view when MyLevel object has been selected in the hierarchy.

The 3D cursor can be moved by moving mouse pointer in the scene view. This 3D cursor will be used to choose where the added or deleted tile will be in the scene view.

Later, when MyLevel object is selected in the hierarchy, a component of the game object will appear in the inspector, which is the representation of MyLevel script. In this component, there will be properties to be configured freely by the game developers as shown in Figure 10 and explained in Table 1.

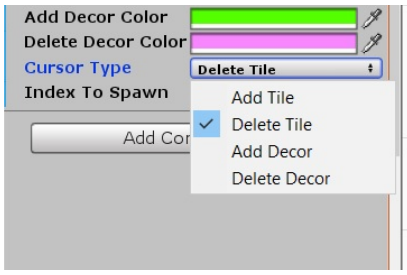


MyLevel
Tag Untagged Layer Default
Prefab
My Level (Script)
Script MyLevel

Tiles	
Size	5
Element 0	Base
Element 1	Rock
Element 2	Sand
Element 3	Metal
Element 4	Dirt
Decors	
Size	6
Element 0	enchantedforest_
Element 1	enchantedforest_
Element 2	enchantedforest_
Element 3	forestRock1
Element 4	forestRock2
Element 5	enchantedforest_
Add Tile Color	🟨
Delete Tile Color	🟥
Add Decor Color	🟩
Delete Decor Color	🟪
Cursor Type	Add Tile
Index To Spawn	0

Figure 10: MyLevel game object on the inspector.

All of those properties above could be modified as desired by the developers. To add or delete an object, tile or decoration, game developers may choose operation type in the CursorType property as shown in Figure 11.



Add Decor Color	🟩
Delete Decor Color	🟪
Cursor Type	Delete Tile
Index To Spawn	

Add Tile
✓ Delete Tile
Add Decor
Delete Decor

Figure 11: CursorType property display after it has been clicked on the inspector.

The color of the 3D cursor can change based on the color configuration in the properties of MyLevel game object as shown in Figure 12.

After choosing an operation type developers can add or delete an object, tile or decoration, by clicking the 3D cursor on the desired position in the scene

Table 1: Properties from MyLevel script component.

No.	Property Name	Functionality
1.	Tiles	Contains the amount and the types of tile objects that can be added to the scene view.
2.	Decors	Contains the amount and the types of decoration objects that can be added to the scene view. These decoration objects will be added above the exact position of the tiles.
3.	Add Tile Color	Contains the color of the 3D cursor for the tile addition operation.
4.	Delete Tile Color	Contains the color of the 3D cursor for the tile deletion operation.
5.	Add Decor Color	Contains the color of the 3D cursor for the decoration addition operation.
6.	Delete Decor Color	Contains the color of the 3D cursor for the decoration deletion operation.
7.	Cursor Type	Contains all types of operation that can be done by the game developer in the level editor. The color of the 3D cursor will change correspondingly to the four color properties above.
8.	Index Spawn	A field to choose the index from a tile or decoration that will be added to the scene view.

view.

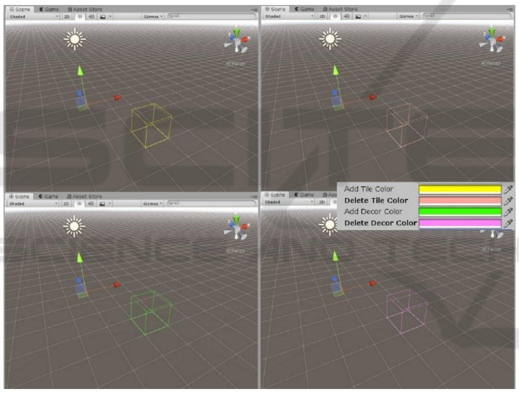


Figure 12: 3D cursor color based on operation type, A) Tile addition, B) Tile deletion, C) Decoration addition, and D) Decoration deletion.

Without the use of this plugin, game developers will consume more time creating levels in tile-based games, because the coordinate of the tiles need to be configured for each new tile added to the level. The coordinate must be perfect in order to make the tiles not colliding with each other and leaving a small spaces between tiles. Hence, the tiles will be placed in a perfect way to create the aesthetic of tile-based game world. The implementation without the use of this plugin is shown in Figure 13.

With the use of the plugin, developers will have their time reduced to create levels, because this plugin provides a point and click feature on tile operations as shown in Figure 14.

The same process also applies to deleting tiles. Without the use of the plugin, tile deletion is done

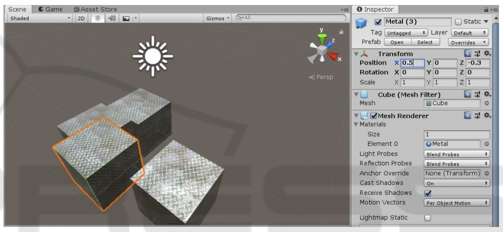

Metal (3)
Tag Untagged Layer Default
Transform
Position X 0.5 Y 0 Z -0.3
Rotation X 0 Y 0 Z 0
Scale X 1 Y 1 Z 1
Cube (Mesh Filter)
Mesh Cube
Mesh Renderer
Materials
Size 1
Element 0 Metal
Light Probes Blend Probes
Reflection Probes Blend Probes
Anchor Override None (Transform)
Cast Shadows On
Receive Shadows ✓
Motion Vectors Per Object Motion
Lightmap Static

Figure 13: Coordinate configuration for each new tile added to the scene view without plugin use.

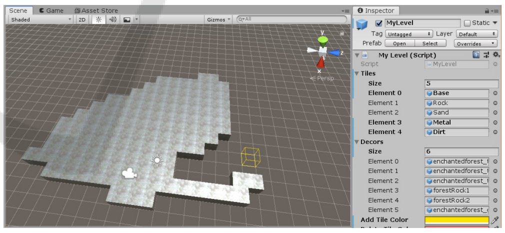

MyLevel
Tag Untagged Layer Default
My Level (Script)
Script MyLevel
Tiles
Size 5
Element 0 Base
Element 1 Rock
Element 2 Sand
Element 3 Metal
Element 4 Dirt
Decors
Size 6
Element 0 enchantedforest_
Element 1 enchantedforest_
Element 2 enchantedforest_
Element 3 forestRock1
Element 4 forestRock2
Element 5 enchantedforest_
Add Tile Color

Figure 14: Point and click feature for adding new tiles to the scene view using the plugin.

by deleting each tile manually from the hierarchy as shown in Figure 15.

Meanwhile using the plugin, developers may choose the tile deletion operation type, then clicking and dragging on the tiles which they want to delete as shown in Figure 16.

This plugin only supports 3D tile-based game development. But game developers may use it to build different type and genres of games, because the level editor is independent with the main game mechanics and game graphics scripting.

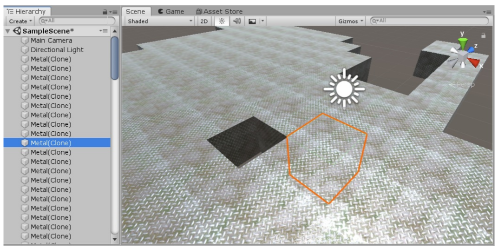


Figure 15: Deleting tile in the scene view without plugin usage.

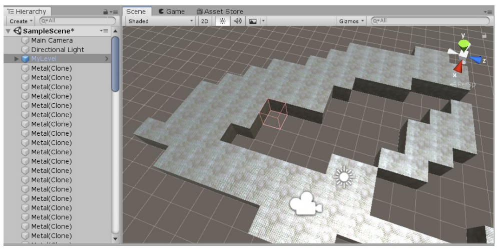


Figure 16: Deleting tiles in the scene view using the plugin.

4.2 Plugin Convenience Test Results

Convenience test has done by five developers, and these results have been collected from the questionnaires filled by each developer. A questionnaire with a Likert scale is answered based on defined multiple choice questions, where the answers represent developers' rating and opinion of some criteria like satisfaction, agreement, and so forth (Lubiano et al., 2017). Following (Sagi et al., 2015), the convenience rate of this plugin from the answers given by respondents is calculated by the formula below:

$$ConvenienceRate(\%) = \frac{AverageScore}{IdealScore} \times 100\% \quad (1)$$

The average score is calculated by the formula below:

$$AverageScore = \frac{TotalScore}{AmountOfQuestions} \quad (2)$$

Table 4 shows the plugin convenience test results according to Likert scale calculation.

The plugin convenience test has the results as shown in Table 4. In the first question, it can be seen that there is one person chose the "Neutral" choice with the score of three points, three people chose the "Agree" choice with the score of four points and one person chose the "Very Agree" choice with the score of five points. Thereafter the total choice of the respondents will be multiplied by the score of their choices, resulting in a total score for the first question equals to 20 points. This total score will be used

Table 2: Likert scale.

No.	Category	Score
1.	Very Agree (VA)	5
2.	Agree (A)	4
3.	Neutral (N)	3
4.	Disagree (D)	2
5.	Very Disagree (SD)	1

Table 3: Convenient rate.

No.	Category	Score
1.	Very Inconvenient	0%-19.99%
2.	Inconvenient	20%-39.99%
3.	Slightly Convenient	40%-59.99%
4.	Convenient	60%-79.99%
5.	Very Convenient	80%-100%

to calculate the plugin convenience rate for the first question by dividing total score with the number of respondents equals to five and then multiplied by 100%, resulting in 80% of convenience rate for the first question. According to the Likert scale that has been designed as shown in Table 4, then the category of 80% rate of convenience is "Very Convenient". The calculations will be the same for the next question until the last question. Afterward the total score and the convenience rate of each question will be averaged, resulting in the final plugin convenience rate of 77,7%. According to the Likert scale, this results in the category of "Convenient".

5 CONCLUSION

This plugin is made to ease the development of tile-based game on Unity game engine. This plugin has the ability to point and click tile level editing directly in the scene view, also supporting custom tiles made by developers to be used along with this plugin. With this plugin, game developers who want to develop tile-based game don't have to deal with the time-consuming tile configuration for each new tile added to the scene and also manual tile deletion on the scene view. Hence, this plugin creates the opportunity for game developers to create levels larger in scale than before, because of how time-saving it is. This plugin has been tested by five game developers with experience in game development. Results as shown in results and discussion, the convenience of this plugin is rated "Convenience" with an average percentage of 77.7%.

Table 4: Plugin convenience test result.

No	Statement	Total Answer					Total Score	%
		VD	D	N	A	VA		
1.	3D cursor controls are very convenient.			1	3	3	20	80
2.	Operation type selection for tile and decoration objects is very convenient.			2	2	1	19	76
3.	Adding tiles and decorations to the scene view is very convenient.			1	1	3	22	88
4.	Deleting tiles and decorations from the scene view is very convenient.			1	1	3	22	88
5.	Choosing colors for every 3D cursor type is very convenient.			1	2	2	21	84
6.	Adding new types of tile and decoration to the inspector is very convenient.			2	2	1	19	76
7.	Adding new types of tile and decoration to the scene view is very convenient.		3	1	1		13	52
Total							136	
Average							19,4	77,7

ACKNOWLEDGEMENTS

The author would like to thank the developers involved in the plugin convenience test for their support and honest feedback throughout the execution of this research.

REFERENCES

Balzarotti, N. and Baud-bovy, G. (2017). HPGE: An Haptic Plugin for Game Engines. *Games and Learning Alliance*, 10653:330–339.

Bergonse, R. (2017). Fifty Years on, What Exactly is a Videogame? An Essentialistic Definitional Approach. *The Computer Games Journal*, 6(4):239–255.

Carmosino, I., Bellotti, F., Berta, R., De Gloria, A., and Secco, N. (2017). A game engine plug-in for efficient development of investigation mechanics in serious games. *Entertainment Computing*, 19:1–11.

Cervantes, H. and Villalobos, S. C. (2006). Using a Lightweight Workflow Engine in a Plugin-Based Product Line Architecture. *Component Based Software Engineering*, 4068:198–205.

Cuadrado, J. S. and Molina, J. G. (2006). A Plugin-Based Language to Experiment with Model Transformation Jesús. *Model Driven Engineering Languages and Systems*, 4199:336–350.

Freiknecht, J., Geiger, C., Drochtert, D., Effelsberg, W., and Dörner, R. (2016). Game Engines Jonas. *Serious Games*, pages 127–161.

Halpern, J. (2018). *Developing 2D games with Unity : independent game programming with C#.*

Karouzaki, E., Savidis, A., Katzourakis, A., and Stephanidis, C. (2007). Tile Dreamer: Game Tiles Made Easy. *Universal Access in Human Computer Interaction. Coping with Diversity*, 4554:382–391.

Lubiano, M. A., Salas, A., De, S., Sáa, R. D., Montenegro, M., and Gil, M. Á. (2017). Soft Methods for Data Science. 456:329–337.

Luongo, C. and Leoncini, P. (2018). An UE4 Plugin to Develop CVE Applications Leveraging Participant's Full Body Tracking Data. *Augmented Reality, Virtual Reality, and Computer Graphics*, pages 610–622.

Nandy, A. and Chanda, D. (2016). *Beginning Platino Game Engine*.

O. Rudel, M., Johannes, G., Weller, R., and Zachmann, G. (2018). UnrealHaptics: A Plugin-System for High Fidelity Haptic Rendering in the Unreal Engine. *IEEE Computer Graphics and Applications*, 38(2):28–30.

S. Durano, V. M. (2018). *Understanding Game Application Development*.

Sagi, F. N., Udiana, I. M., and Ramang, R. (2015). Kajian Faktor-Faktor Penyebab Ketidakefektifan Kinerja Terminal Bus Haumeni Kota Soe Kabupaten Timor Tengah Selatan. *Teknik Sipil*, IV(2):183–194.

Salomão, A., Andaló, F., and Luiz Horn Vieira, M. (2019). How Popular Game Engine Is Helping Improving Academic Research: The DesignLab Case. *Advances in Human Factors in Wearable Technologies and Game Design*, 795:416–424.

Spuy, R. V. D. (2010). *Game Design with Flash*.

Tredinnick, R., Boettcher, B., Smith, S., Solovy, S., and Ponto, K. (2017). Uni-CAVE: A Unity3D plugin for non-head mounted VR display systems. *IEEE Virtual Reality*, pages 393–394.