Identifying How Any Interface Elements Are Created in The Gamemaker Engine.

Jibran Wafi Prawiko[1], Normalisa[2], Pradana Atmadiputra[3]

[1, 2, 3]Computer Science, International University Liaison Indonesia, BSD City, Indonesia, 15310
e-mail: [1]jibran.prawiko@stud.iuli.ac.id, [2]normalisa@iuli.ac.id, [3]pradana.atmadiputra@iuli.ac.id

Abstract. A GUI, whether well or badly designed, is one of the most important requirements for any video game. The role of a GUI is to present the player with their in-game character's current status, whether it's the number of ammunitions of a weapon, the current active quest, or the player's current location within the game's map. Creating GUI elements should be one of the first guides that are taught after a GameMaker tutorial provides a guide on creating a camera object. The creation process of a GUI element can be divided into two parts; its positioning and functionality. In theory, simplifying the development process, in this case the algorithm, for creating GUI elements is possible, mainly by categorizing every GUI element based on what they are presenting to the player. Developers could then create the same algorithm for every element that falls in one category, thus simplifying the development process.

Keywords: GUI elements, game development, functionality, positioning, developmental process, simplification, categorization.

1. INTRODUCTION

Every video game released from the earliest years has an element to indicate various statuses during gameplay, whether it is the status of the score, the status of health, the status of the current level, etc.

This type of element is generally known as a GUI (stood for Graphical User Interface). A GUI is a layer of technology with which a user engages that allows them to visually interact through things like icons, menus, and other graphics **(Walkme, 2022)**. Human interactions take place using pointing devices such as a mouse, keyboard, stylus, or touch. GUI examples include computer monitors, smartphones, tablets, gaming systems, or other consoles **(Walkme, 2022)**.

The same can be said with its implementations in a video game. GUI elements in a video game take a digital form. A GUI in a video game consists of many static elements (often as objects) that display specific values to indicate the current status of the player. It is an important factor of any video game when looking at the lack of titles which do not possess any GUI elements. Thus, creating a GUI should be one of the core parts that should be included in every game development tutorial for a game engine.

Gamemaker Studio 2 has been one of the most popular choices as a lightweight game engine for independent game developments. And with every Gamemaker Studio 2 tutorial having a different developmental process, it will be difficult to conduct small experiments in the project based on the amount of knowledge gained at that point, with the reasons of being afraid that those experiments will affect their progress, and prevents beginner developers from continuing to follow said tutorials. one of those experiments may regard the creation of an additional GUI element.

This research aims to create an algorithm for creating any GUI elements in the Gamemaker engine, regardless of whatever game is created in the engine and also in whatever progress the development is currently in. Developers should be able to apply this algorithm whether their game is near completion, or just started with their character movement. Furthermore, the algorithm should not break or affect other game mechanics to the point of being unable to progress through the developmental process when following a certain tutorial.

The scope of this research aims to cover the essential parts and overall process of creating a GUI element. This includes the positioning of the GUI, the different layers for each GUI element, what events and variables should be included when creating a single GUI element, and finally what types of GUI element can be added into the game using the algorithm. The last part also aims to detect if there are any limitations of this research, in the case of which GUI element cannot be placed using that algorithm.

2. LITERATURE REVIEW

Gamemaker Studio 2 is a tool designed to empower game developers and their team to make new and innovative games as well as prototype ideas in the fastest and most intuitive way possible across multiple target platforms **(YoYo Games, N.d)**. The engine provides a built-in programming language known as GML (stands for GameMaker Language). It is a programming language that is based around C, meaning developers who are familiar with the following language could find familiarity when working with GML.

The architecture of the Gamemaker engine uses an event-driven programming paradigm. An event-driven paradigm is a paradigm where entities (objects, services, and so on) communicate indirectly by sending messages to one another through an intermediary. The messages are typically stored in a queue before being handled by the consumers **(Sayfan, 2022)**.

In game development, events are discreet moments in the game loop where things are made to happen based on what you have programmed for them **(YoYo Games, N.d)**.

One of the events that can be found in the GameMaker engine is the draw event. Draw events are a type of event that govern what is presented on the screen when running the game **(YoYo Games, N.d)**. The example would be when drawing an object as a background for another object, such as drawing a rectangular frame behind a score element.

There is also the camera event, which has more universal implementations in the general game development scope. In the general gaming term, A camera is a player's vantage point in a game, her eye into the world **(Kelly, N.d)**. It is a crucial mechanic that is utilized for both 2D and 3D game development.

The Gamemaker uses two types of tools when creating the camera object, the viewports of a room and the camera itself. Viewports are, basically, little windows into the game world that enable the player to see parts of a room, either scaled or 1:1 depending on the game itself, and as such they are essential when your game room is larger than the display size. The cameras are what define exactly what will be shown in each viewport **(YoYo Games, N.d)**.

Every 2D video game is built upon how many pixels it has. A pixel is the smallest unit of a digital image or graphic that can be displayed and represented on a digital display device (Rouse, 2020). The number of pixels in a video game is an important factor when developing GUI elements.

Since objects tend to have sprites attached to them, the position of an object with a sprite can be referred to in the sprite's origin point. The origin of point can be found within the sprites in GameMaker, it acts mainly as a pivot point to when an object is rotated into a different angle **(MashArcade, 2020)**. Although, the second role of the origin point is as the value that is declared when referring to the position of an object. Manually declaring an object's position from a script came from the origin point of the object's sprite.

However, the positioning of an invisible object that contains no sprite must be set manually. The position could either be the same as other objects, or it could be set when the developer inserts the object into the room.

Finally, the location of a GUI element is also defined by its layer. Layering in a 2D plane, whether related to game development or otherwise, determines the order of objects in the game room. The principles of layers in a 2D game, especially in the platformer's genre, are similar to the layers found in art-development software. In a sense, each instance of a created object should have their own specific layers, whether the object has the same or a different purpose than another object.

3. METHODOLOGY

Before identifying the methodology, it is important to understand what type of GUI elements are implemented in different video games, specifically in various 2D games since Gamemaker is an engine specializing in 2D game development. This is done by analysing the interface from 3 of the most popular 2D video games from each generation, starting from the first year of the second generation of video games in 1976 to the last year of the eighth generation of video games in 2020.

For a summarization of the analysis, the GUI element types of a video game can be classified into 3 categories based on what kind of content or gameplay status is shown within an element:

- A GUI element in the form of an image of another object.
- A GUI element that shows a string or integer.
- A GUI element that shows a meter.

The GUI elements taken for this research will be imported from existing titles of 2D games. In which the titles selected depends on which sprite set is available to download on the internet. Each of the imported GUI elements shall use the mentioned calculation formula to calculate their positioning, and an algorithm for creating their functionalities based on their category. All the code written in this research will not be taken from outside references (except for the camera object), though rather created by the author itself.

Positioning

Since GUI elements are created as game objects, the position of a GUI element in any game should follow the camera object. The camera object is an invisible object that should be placed in the middle of the room, it has a fixed position that moves consistently as the player moves. GUI elements are objects that must act as an extension to the screen, meaning the positioning of those objects must be set in the camera object's X and Y axis. However, doing so will result in the engine outputting the elements at the middle of the screen.

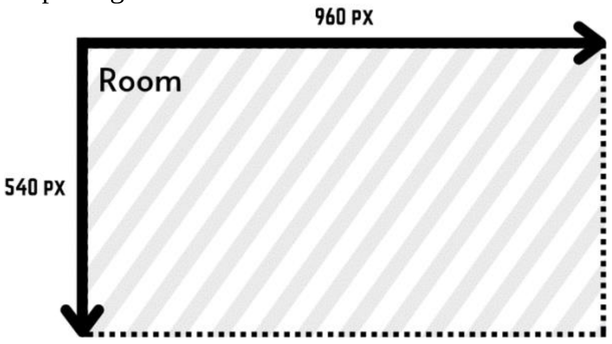


Figure 3.1.

This problem can be fixed by determining the current resolution of the room. The *room_width* and *room_height* are functions that will return the value of the room's width and height when called. Both the first pixel of the room's width and height begins at the left-top corner of the room. The camera's position in the middle of the room is set by *camera_get_view_width(view_camera[0]) * 0.5* and *camera_get_view_height(view_camera[0]) * 0.5* in the camera object. *View_camera[0]* refers to the area of the previously-created viewport.

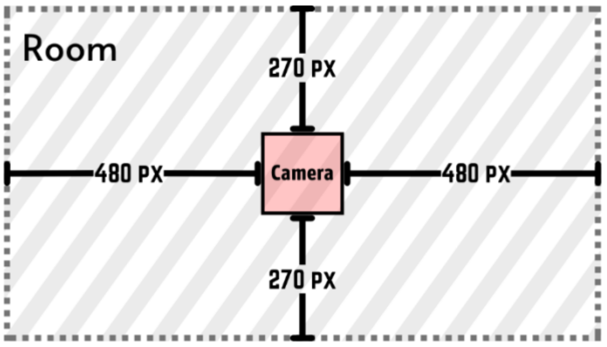


Figure 3.2. GameMaker's camera object position

Determining the current pixel of the room can be useful to measure the position of a GUI element. If the initial position of the element is set to the position of the camera object, then the image above represents the positions of the elements when set to the same x and y value as the camera object in the middle of the room. Determining the distance between the camera object and the room's width and height is required to create the simplified formula for positioning GUI elements later.

A GUI object's position should not extend the limits of the screen set for the game. Unless the object is scaled, this value and positioning limit can still be calculated.

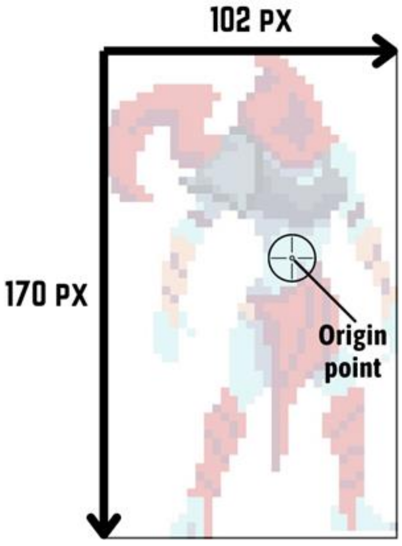


Figure 3.3. Sprite origin point

The first step is to calculate the horizontal and vertical distances of an object sprite from its origin point.

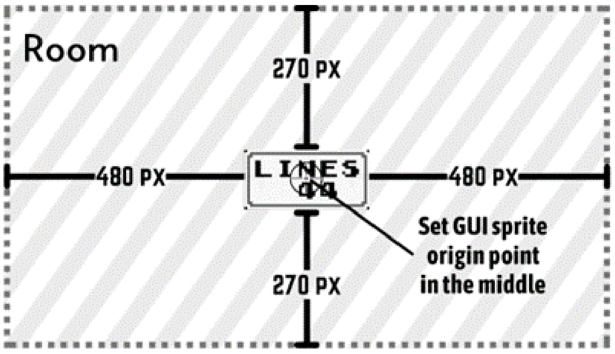
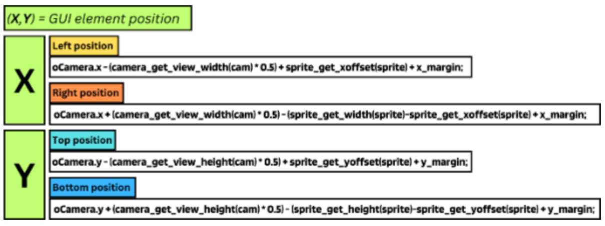


Figure 3.4. GUI object placement based on the camera

The second step is to calculate the distance between the origin point of the object and size of the room.



```
oCamera.x - (camera_get_view_width(cam) * 0.5) + sprite_get_xoffset(sprite) + x_margin;
oCamera.x + (camera_get_view_width(cam) * 0.5) - (sprite_get_width(sprite)-sprite_get_xoffset(sprite) + x_margin;
oCamera.y - (camera_get_view_height(cam) * 0.5) + sprite_get_yoffset(sprite) + y_margin;
oCamera.y + (camera_get_view_height(cam) * 0.5) - (sprite_get_height(sprite)-sprite_get_yoffset(sprite) + y_margin;
```

Figure 3.5. Simplified formula for GUI positions

It is now possible to determine the positioning of the object without extending its value. Above are the final calculations, where if it is implemented inside the object of a GUI element, then the object should be located at one of the four corners of the screen.

The resolution of the room for this project will be 1280px * 720px, the GUI elements imported from the internet will all have different functionalities and resolution for testing purposes.

Functionality.

The methodology of GUI functionalities is identified by analysing what are the functions and variables that must be included to create a GUI element based on its type and role.

Type	Functions
A GUI element in the form of an image of another object	Basic mathematical functions to change the value of the integer.
	Function to change the string into a different string.
A GUI element that shows a string or integer	The ability to change the sprite during certain conditions.
A GUI element that shows a meter	Having multiple meters based on what status the meter represents.
	Functions to either decrease or increase the meter.

Table 3.1. Requirements to develop GUI functionality

The Imported GUI elements will take the form of a sprite set available on the internet. This function on the table above shows that the sprite set should feature multiple sprites that handle the same function, such as multiple meter-type GUI sprites with different colours.

4. RESULTS

Sprite sets

All sprite set assets are taken from a royalty-free source found in itch.io, they are all created by the user Wenrexa, who specializes in creating sprite sheets for GUI elements.

1. **GUI sprite set 1 "GreenBlack"**

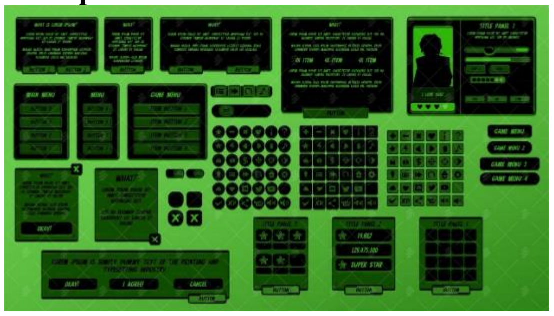


Figure 4.1. "GreenBlack" by Wenrexa

2. **GUI sprite set 2 "Hologram Interface"**

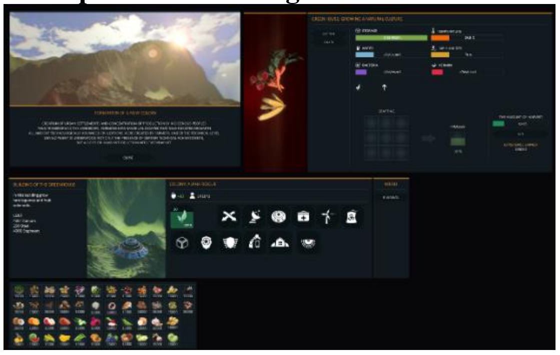
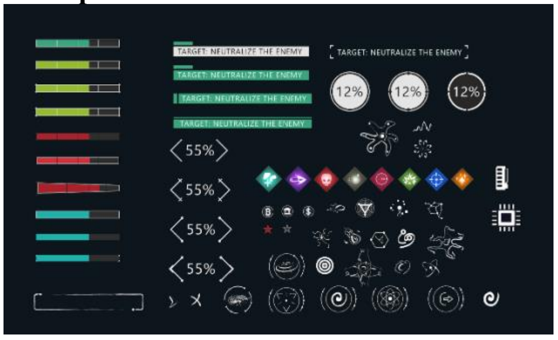


Figure 4.2. "Hologram Interface" by Wenrexa

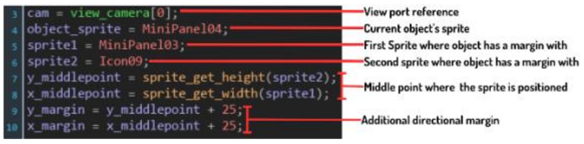
3. **GUI sprite set 3 "Sci-Fi UI"**


TARGET: NEUTRALIZE THE ENEMY
TARGET: NEUTRALIZE THE ENEMY
TARGET: NEUTRALIZE THE ENEMY
TARGET: NEUTRALIZE THE ENEMY
TARGET: NEUTRALIZE THE ENEMY
12%
12%
12%
55%
55%
55%
55%
Figure 4.3. "Sci-Fi UI" sprite set by Wenrexa

Positioning

The events used to create the positioning of the GUI elements are the same across all the GUI sprite sets. The following objects from the sprite set only feature two events:

- **The Create event** where it contains the variables needed for the third chapter's formula.



```
cam = view_camera[0];
object_sprite = MiniPanel04;
sprite1 = MiniPanel03;
sprite2 = Icon09;
y_middlepoint = sprite_get_height(sprite2);
x_middlepoint = sprite_get_width(sprite1);
y_margin = y_middlepoint + 25;
x_margin = x_middlepoint + 25;
```

View port reference
Current object's sprite
First Sprite where object has a margin with
Second sprite where object has a margin with
Middle point where the sprite is positioned
Additional directional margin

Figure 4.4. variable description inside create event.

- **The End Step event** where it contains the four main positioning formulas in the last chapter.

Here are the results:

1. **GUI sprite set 1 "GreenBlack" – Positioning.**

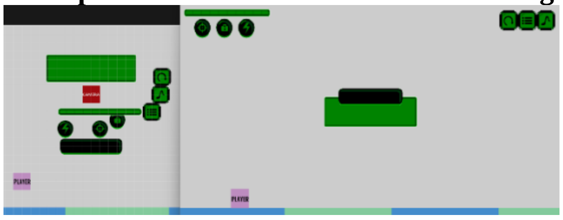


Figure 4.5. The positionings of GUI elements from the "GreenBlack" sprite sheet.

2. **GUI sprite set 2 "Hologram Interface" – Positioning.**

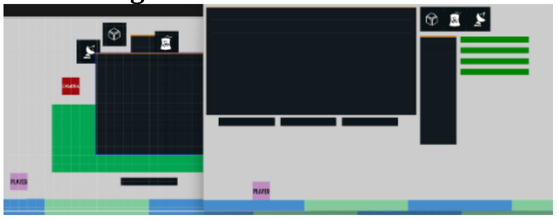


Figure 4.6. The positionings of GUI elements from the "Hologram Interface" sprite sheet.

3. **GUI sprite set 3 "Sci-Fi UI" – Positioning.**

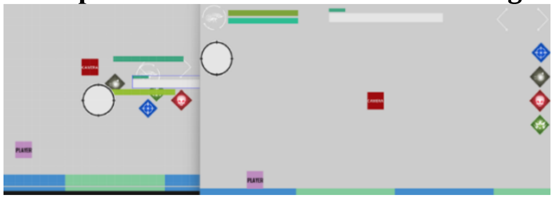


Figure 4.7. The positionings of GUI elements from the "Sci-Fi UI" sprite sheet.

Since the formula already simplified the algorithm, developers only need to edit the variables on an object's create event that can determine the margins. Furthermore, this also means that developers do not need to place objects in the correct order and/or position when editing the game in the room editor.

Functionality

The events used to create the functionality of the GUI elements are the same across all the sprites in the GUI sprite sheet presented in the previous part. The solutions for each of the requirements mentioned in chapter 3 will be summarized by using a table.

Type	Functions	Universal solution
A GUI element in the form of an image of another object	Basic mathematical functions to change the value of the integer.	Create new invisible object, add **create event** for empty text variable, add **step event** to modify the values of the text, add **draw event** to create a new instance of the string and position the invisible object to the textholder object using the **draw_text_transformed()** method.
	Function to change the string into a different string.	
A GUI element that shows a string or integer.	The ability to change the sprite during certain conditions.	Inside the object, add a **step event**. Whithin the step event, create positions where **if (condition) {sprite_index = other_sprite}**; sprite_index is used for changing the main sprite into a different sprite.
A GUI element that shows a meter	Functions to either decrease or increase the meter.	Inside the object, add the **draw GUI event**. There are only two methods required for the solution: The **draw_healthbar()** method for drawing the bar itself and the bar's value, and the **clamp()** method for setting the minimum and maximum value of the bar.
	Having multiple meters based on what status the meter represents.	Have multiple **draw_healthbar()** methods on the same **draw GUI event**. Set each method with different positions and values of the meter.

Table 4.1. Solutions for functionality requirements.

The results of implementing the same solutions within each GUI element category are as follows:

1. **GUI sprite set 1 "GreenBlack" – Functionality.**

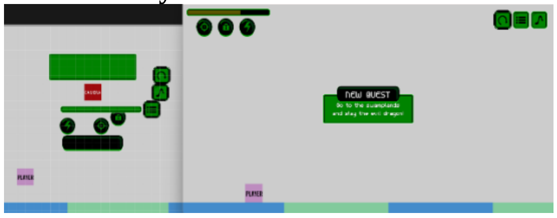


Figure 4.8. The functionalities GUI elements from the "GreenBlack" sprite sheet.

2. **GUI sprite set 2 "Hologram Interface" – Functionality.**

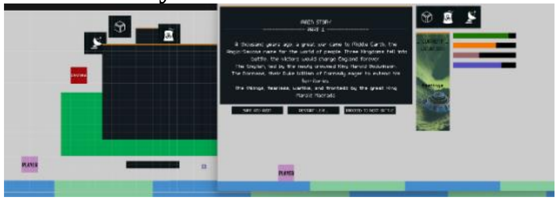


Figure 4.9. The functionalities GUI elements from the "Hologram Interface" sprite sheet.

3. **GUI sprite set 3 "Sci-Fi UI" – Functionality.**

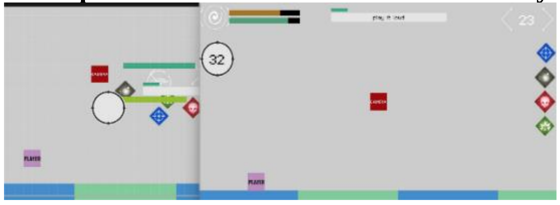


Figure 4.10. The functionalities GUI elements from the "Sci-Fi UI" sprite sheet.

From the results above, it is proven that there is a possibility to insert functionalities of different GUI elements in the same category by using the same events and methods.

5. CONCLUSION

The results taken have successfully fulfilled the aim and purpose of this research; to create an algorithm that could simplify the process of creating GUI elements in GameMaker, from the object's positionings and its functionality. The following results have also concluded that GameMaker developers only needed to edit the variables inside the create event for changing the positions of objects, and showed how the same formula can be used and still output the same results across different objects. Meanwhile, developers can edit certain conditions with the same output in the step event and draw events to change the functionalities of objects.

REFERENCES

Kelly, T. (N.d, N.d N.d). *Camera*. Retrieved from www.whatgamesare.com: https://www.whatgamesare.com/camera.html#:~:text=Camera%20A%20camera%20is%20a%20player%E2%80%99s%20vantage%20point,scrolling%2C%20movable%2C%20floating%2C%20tracking%2C%20pushable%20and%20first%20person.

MashArcade. (2020, 5 12). *GameMaker Studio 2.3: Using Origins in Sequences!* Retrieved from www.youtube.com: https://www.youtube.com/watch?v=HPJCFfmI8v4

Rouse, M. (2020, 8 31). *Pixel*. Retrieved from www.techopedia.com: https://www.techopedia.com/definition/24012/pixel

Sayfan, G. (2022, 11 8). *Introduction to event-based programming*. Retrieved from aiven.io: https://aiven.io/blog/introduction-to-event-based-programming

Walkme. (2022, 12 19). *Graphical User Interface (GUI)*. Retrieved from www.walkme.com: https://www.walkme.com/glossary/gui/

YoYo Games. (N.d, N.d N.d). *Cameras And Viewports*. Retrieved from manual.yoyogames.com: https://manual.yoyogames.com/GameMaker_Language/GML_Reference/Cameras_And_Display/Cameras_And_Viewports/Cameras_And_View_Ports.htm

YoYo Games. (N.d, N.d N.d). *Introduction To GameMaker*. Retrieved from manual.yoyogames.com: https://manual.yoyogames.com/Introduction/Introduction_To_GameMaker_Studio_2.htm

YoYo Games. (N.d, N.d N.d). *Object Events*. Retrieved from manual.yoyogames.com: https://manual.yoyogames.com/The_Asset_Editors/Object_Properties/Object_Events.htm

YoYo Games. (N.d, N.d N.d). *The Draw Events*. Retrieved from manual.yoyogames.com: https://manual.yoyogames.com/The_Asset_Editors/Object_Properties/Draw_Events.htm