GameMaker Language and Its Applications

Author: Logan Brandt

Abstract:

GameMaker Language is a programming language that was designed specifically to create 2D games. GameMaker is also committed to trying to be beginner friendly for newer game developers. This has impacted design choices in the language, what features it has included, and how these features are implemented.

The language that I have chosen to work on for this project is GML, also known as GameMaker Language. GML was developed by Yoyo games, and the first version of it was released under the name Armino. Since then, the language has grown a lot and been updated multiple times. Currently, the environment for the language is called Gamemaker Studio 2, and uses a Graphical User Interface to help the users. In this project, I explored different features of GML, and how it has implemented these features to help make the task of creating a game easier for developers. I also explored what GML does not do, as it is not to be used for developing every type of game. Overall, I learned that GameMaker Studio and its language GML is specifically designed to be beginner friendly with many of the features it chose, yet still have a layer of depth for more experienced programmers to choose this language.

The first important thing to consider about GML is what it was designed for: making games. This design goal impacts everything about the language, such as what variables to use, what methods to include, etc. Because of this, there are methods and variables present in this language that are not in others. For example, there is the "lives" variable, which can be used to store the lives of the player. This variable is innately global, as in it can be used anywhere in the program immediately, but it does not have to be specified by the programmer. GML is one of the only languages I know of that automatically creates global variables for the programmer to use. This is an example of them creating their architecture with their intended audience in mind.

The next thing to consider about GML is it's layout. For GML, there is a Graphical User Interface with many different tabs and windows for different features of the language. This is vastly different from other languages, as they are not nearly as reliant on GUI's to navigate the different areas of the code. In GML, there are a list of different pieces to code called resources. As of Version 2.2.4.374, there are 15 different resources (GML). Each resource is for a different aspect of making the game, and each can do different things. By default, it does not show what inside of resources, but clicking on the resource shows all the resources of that type that are created. You can also create folders to store these resources in a way without as much clutter (Hitti).

Some of these resources are simple. One example is the Sprite resource. This allows the user to store sprites, which are 2D drawing that could depict anything shown on screen, from the character to the environment to weapons used by the character. This tool has an option to let you create your own sprites using a rudimentary tile system. It also allows for multiple frames of the sprite, or different positions the sprite can have. This can allow for sprites to change their position to look animated or to change how they look based on the action they perform. There are many different actions that can be taken with just the sprite resource, and it is not even the most complex. The three most complex, and arguably the most important, are script, object and room (GML).

The first resource to go over is the script resource. This operates similarly to writing functions in other languages such as Java. You use command line arguments to take in input, and use that input to either return something, or make changes in the code. One big difference GameMaker has in this aspect, however, is that the arguments can not be named. They are referred to as argument0, argument1, etc. There is also a maximum number of arguments a script could have, which is sixteen (GML). This number of arguments for any kind of function or script is rare, so this limit is not a drawback to its main purpose. Other than the fact that you can't name variables, however, scripts really are not all that different from functions in other languages. They work as one familiar in programming would expect, and don't have any other unique features. Scripts do exist separately from objects, and all objects would be able to call a script. This is beneficial for having a function that would work for two different objects, as in a language such as Java, objects would usually be in two different classes and need inheritance to use the same function. GameMaker makes this process simpler and easier to use for the programmer.

The object resource stores an object in the system that the creator can influence and manipulate. This resource holds many powerful tools, and is the main power behind creating game logic and making things happen. When you first create an object, you can assign it a sprite, which will be what the object looks like when it is first created. Inside of the object class, there are a plethora of events that can occur. These events are things that occur upon specific timings or when they are called. For example, there is a create event that will execute whenever the object is first created. This functions like a constructor in other languages such as Java, and allows for declaration of local variables that can be used elsewhere in the programmer. A more unique event that the object resource has is the Step event. This event is important for allowing for changes to happen in the game in real time. The way it works is that it occurs for every frame of the game. Frames operate by showing one image of a game, then another, and another, while adjusting what is happening on the screen between each frame based on player input and code for the environment. For example, if a game operates in 60 frames per second, then the step event would be called 60 times, and it's code would run 60 times. This is where user input would impact an object to move, for example, or for an AI to use existing code to move as well. While different objects can have different code in their step events, step will occur the same number of times for both of them, meaning that the objects will "step" at the same time. These are just two different features of the object resource, but there are many more that exist that can have a major impact on the object they are coded in.

The object resource has similarities to other OOP. Since the basis for the code was in C++, GML was able to use the code to make objects in their own way. Each object has its own set of variables, and these variables can be made independent of the object. Many of the things that can occur with objects in other languages can happen here, including things such as inheritance. Another important thing to note is that not all objects have to have a sprite, as there can be invisible objects that work in the background while the game occurs in the foreground. Invisible objects are paramount for adding logic into the game in ways that objects the players sees could not accomplish. A common feature for invisible objects is for them to keep track of the score to see if a player has won or run out of lives. They can also set up alarms. Alarms are events that are triggered when an alarm is set somewhere else, and then goes off after a certain amount of time. This is a way for different objects to directly interact with each other, as an alarm in one object can activate the code in a different object. GameMaker has 12 alarms that can be programmed in, but alarms are not unique code. An alarm going off can trigger multiple objects at once if they have code for it, which can make it pretty useful. An example to use this is if a player creates an explosion that destroys everything on the screen, having an alarm could allow for everything to get destroyed all at once. This solution seems far easier than something that could occur with objects in a language such as Vizard. For Vizard, you would have to loop through and delete all the objects manually if the event were triggered. Objects can interact with other objects and do some crazy things, but only under one condition: they have to be in a room.

The room resource is where all of the gameplay occurs. All games in GameMaker must have at least one room. In order for an object to be able to execute, it must be added to a room. Objects not in the room will not execute, as they will not be created, and thus won't be able to run. To add an object to a room, a creator can drag it from the right side where it is shown and drop it into a tile. It does not matter where invisible objects go, and they will not impair the movement of other objects on screen. This can also be done with sprites, but sprites will not move once placed into the room. Objects, however, can operate within the room, and perform actions as their code permits, such as print text to screen, move around as a sprite, or change variables that affect the overall game. Rooms also use the depth keyword to determine which objects are on top. The lowest depth is on top, and can include negative numbers (GML). This is useful for determining which objects are in the background and what objects are in the foreground. Room also has its own special editor that it uses, and contains a few unique elements.

At the top of the room editor is an option between instances and background. Based on which you choose, the middle section of the room editor changes accordingly. The instances shows the instances of all objects in the room on creation, but it can't determine which objects will be created once the game launches. This changes on a room by room basis, so some rooms can have different instances of objects in them at any time. You also have a checkbox, which can be checked on or off. This can be useful for debugging, as you can turn off some objects in order to test others in the room. The other option for the top section is background. This is used to change the background of the room, including the color of the room and if the background will be a sprite or not. Sprites used as a background can also be animated, and can move in the background as well. By changing the horizontal and vertical speed, you can have the background sprite move across the screen, with positive moving the sprite right and down when positive and left and up when negative. This can allow for dynamic backgrounds that can change during gameplay, and can be interesting to see. Finally, in the bottom right are the room properties. These contain things such as room creation code and the width and height of the room. Making a sprite the background of a room does not automatically change the overall width of a room, so it must be adjusted manually if you want a sprite to take up the entire background (GML). Rooms are powerful tools for game creation in GameMaker, and are essential for adding in the visuals to make the game playable.

On top of having different resources that perform different functions for the language, GameMaker Studio also comes with specific keywords specifically for the language. These keywords exist in order to make programming for a gaming language easier, and are adapted for the GameMaker Language. The keywords that are unique to GameMaker are other, all, and noone. For noone, this keyword is used to determine if there are any instances of an object nearby. This can be used in comparison, such as determining if there are any enemies near the player. As for all, this keyword will get all instances of a certain type of object, and set them accordingly. This is good to use when you want to change something about objects, such as if a button is pressed that changes the color of enemies from green to red. Finally, there is the other keyword. This keyword has two different meanings, depending on context. Those are when it is used in a collision event, and when it is used by the with function. For collision, other is used to indicate the object that the current object that has the collision code is colliding with. Here, the keyword is used to store data on the object, and can be called with the keyword. This is similar to a function using self in java. The second meaning for other comes from its use in the with() function. This function is between two different objects, and here other would specify the object who does not have the with() in their code (GML). This can be used outside of collision code to specify how two objects might interact, such as pressing a button to change the direction of a projectile. These built-in keywords exist in the language to assist in game design, and are they are able to do so in an efficient manner.

The use of these keywords can make a lot of things far easier for the programmer. One example from my own experience is writing the collision function. Last year for Computer Graphics, there was a project to code up Space Invaders. The software that we used was Vizard, a type of 3D software that had a basis in python. Some of this work involved determining if a bullet fired from the ship had connected with an enemy unit. This code took place in two nested for loops and had to go through every single bullet and every alien on screen. At this point, we had not learned about collision masks, which is the set of points within a sprite that would signal that something had collided with it. Because of this, the code took some time to write, so the coordinates had to be specified for each of the four corners of both the alien and the bullet to see if they collided. GameMaker automatically makes a collision masks for sprites created for the user, which was something Vizard did not do. This is also beginner friendly, as newer programmers do not have to worry about initializing it to turn collisions on. There is also a built in event for handling collisions, and the calculations are done for the programmer instead of having to calculate them. Both objects colliding are also already specified, as one of them has the code for the collision object, and the "other" is the object that is triggering the collision code. GameMaker was much easier to write collision code for, and showed that this language was optimized over Vizard to handle collisions between two different objects.

Moving on from this example, it's time to look at something unique about GameMaker I've discovered while working with the language. What I've learned is that it also comes with some built in global variables. The variables for score, health, and lives are built in and already declared whenever a new game is created through GameMaker Studio 2. Normally, to make a global variable, you would have to specify global before the variable name, but in the case of these three, any method calling these variables will change the global variables. These variables also differ from how other global variables differ in the language. For other variables to be declared global, they have to use global.variableName. In the case of these three, however, that is not necessary, and any of the objects can use and change them. It is an interesting feature to have in a language, as it is quite unusual to have already declared global variables in a language. Still, this does save the developer from having to write global. in front of every keyword, and these variables are quite common in game development, so it makes sense to include them (GML).

The way in which GameMaker is able to interpret code is also different from other languages. GameMaker is supposed to end each line with a semicolon, but if the user misses a semicolon, an error is not thrown, and instead the program interprets the code as if there were a semicolon there. This is a nice feature for someone with no experience, as it means they do not have to remember to use a semicolon at the end of each line in order to have their code running properly. The downside to this is that a programmer would not be able to write multiple different statements on a single line, although standard programming practice usually goes against doing this. This does not work, however, for using brackets, as only the line after a line that needs brackets will be read if there are none included, much like other languages. Another helpful feature is that by right-clicking on a function, you are able to pull up a manual page for it. This is especially helpful for newer programmers who are unsure of how a program operates. An additional feature that can be accessed by right clicking is a code snippet. These can be used to create sample code for a variety of statements, such as a single line comment, a for loop, or an if statement. This is useful to newer programmers who are unsure of how to write them, and more experienced programmers who don't want to write out everything from scratch. All of these features exist to make programming easier for programmers, and make the language much more friendly for newer programmers.

GameMaker also has some different methods from other languages for when it comes to performing certain actions. There is no print function in GameMaker, as since everything is displayed in a 2D plane, the x and y coordinates for any text must also be known. As with other languages, there are a few different print functions in GameMaker. One of them is draw_text(x, y, String). This method prints a string that starts at the specified x and y coordinates to screen. There are a few variations on this method, such as including color, but other than that nothing too major. There is also a draw event in objects, that will draw something on the screen when they are drawn. This event can be used to recolor the character if need be, or print some of their dialogue to the screen, and is used when it is called by the method it is in. While it does contain a way to print to the screen, GameMaker having a 2D window to print to and not a command line prompt changes how it is able to print text to the screen.

After discussing all of the things that GameMaker can do, and can do better than other languages, it's time to bring up what it can not do. Probably the most important distinction for GameMaker is that it can only do 2D games. GameMaker is specifically designed with 2D in mind, and as a result does not have any functionality porting over to 3D. The developers have consciously made this choice, saying "Because we're completely focused on 2D games, anyone aspiring to make something in three-dimensions is essentially forced to leave" (Grubb). While they have made advancements into making their software robust to work in 2D, it does not translate to 3D. This is a limitation on the software that has a real impact on who can and can not use the software.

GameMaker was designed to be beginner friendly and accessible. When asked about this, general manager of YoYogames James Cox said “Anecdotally, we do have quite a large segment of the community that comes into GameMaker: Studio with nothing but a guided interest in making a game, this is why we do our best to provide support – one of the biggest hurdles of game development is finding comprehensible help – and assist novice devs on a path to making their first game” (Grubb). GameMaker wants to serve as a starting point for developers, and help support them in the process of creating their first game. While they want to function as a haven for newer developers, GameMaker also wants to be useful for more experienced developers, as “it has made it clear that it capable and robust enough to support a seasoned studio making its third or fourth release on Steam” (Grubb). This means trying to create a software that appeases both newer programmers and more experienced programmers with their software. These two crowds often want different things implemented into the language for them, and can impact what the studio behind GameMaker Language decides to add in to the language.

GameMaker Language is a living language, as it is still being worked on and updated all the time for it’s growing community of developers. The language is capable of doing incredible things, as developers have produced some amazing games from GameMaker. This is a language designed to guide someone into making their first game, and then give them the tools to make more games from there.

Works Cited

(Will be abbreviated to GML during in-text citations)

"GameMaker Studio 2." Edited by YoyoGames, *GameMaker Studio 2*, 2019, docs2.yoyogames.com/.

Grubb, Jeff. "GameMaker Studio Creators Look Back at 17 Years of Development." *VentureBeat*, VentureBeat, 6 Sept. 2017, venturebeat.com/2017/09/03/gamemaker-studio-creators-look-back-at-17-years-of-development/.

Hitti, Allejandro. "25 Of My Favorite Tips, Tricks, and Upcoming Features in GameMaker Studio 2." *Amazon*, Greenhaven Press/Gale, 2008, developer.amazon.com/blogs/appstore/post/b071dd08-f9ae-4d2a-892e-4bff23fc032b/25-of-my-favorite-tips-tricks-and-upcoming-features-in-gamemaker-studio-2?cmp=US_2017-00_Inf_InfBlogs&ch=Inf&chlast=Inf&pub=AlH&publast=AlH&type=org&typelast=org.

YoyoGames.com. *My First Game - Intro To GameMaker - Space Rocks*. Performance by FriendlyCosmonaut, *YouTube*, YouTube, 23 Oct. 2018, www.youtube.com/watch?v=raGK_j1NVdE.