I. Achieving Learning Goal

ATL Skills: Research (Information & Media Literacy), Self-Management (Organizational)

My research for learning goals alternated with the creation of my product due to several properties of my learning goal, which consists of learning the balance between art designs and programming during a video game development, and the methods to write and run scripts through a game developing engine. Firstly, the balance between art and programming is a subjective conclusion based on efforts spent for different areas in the product. Next, communications with a game development engine have multiple ways and uncertainties to be tested and selected when making the product. The methods and errors to research turned out to exceed the action plan in folds when I began the actual game development. The following tables present the most fundamental parts of my research. During the long research, I not only applied the skills of finding and organizing information, but also developed information and media literacy and learned a new standard of selecting platforms and information under the context of programming.

Methods and Algorithms:

Goal: Make GameObjects move or rotate		
Source	Suggested Method	Adopted & Reason
Unity Manual[1][2][3] docs.unity3d.com/ScriptReference/Vector3.html docs.unity3d.com/ScriptReference/Quaternion.html docs.unity3d.com/ScriptReference/MonoBehaviour.InvokeRepeating.html	Change the position and rotation of GameObjects with Vector3 and Quaternion following the declaration instruction, and repeatedly change towards a direction with InvokeRepeating	Adopted as meeting the needs, and the declarations of Vector3 and Quaternion have sufficient efficiency to coordinate with other parts of script
My modification: Make parameters for Vector3 and Quaternion global variables to modify as InvokingRepeating cannot run functions with parameters		

Goal: Drag and drop GameObjects		
Source	Suggested Method	Adopted & Reason
YouTube[4] www.youtube.com/watch?v=yalbvB84kCg	Track the mouse position on screen and set GameObject to the mouse position	Not adopted because the method does not apply to 3D GameObjects, and making cards 2D limits the animations
YouTube[5] www.youtube.com/watch?v=uNCCS6DjebA	Track the mouse position with Raycast and set GameObject to the position	Adopted as meeting the needs and allowing other variables to be added for reference
My modification: Move the section updating per frame from script Grabber to script SlotAndCard		

Goal: Shuffle card decks (arrays)		
Source	Suggested Method	Adopted & Reason
GeeksforGeeks[6] www.geeksforgeeks.org/shuffle-a-deck-of-cards-3/	For every position in the array, swap it with another position determined by random.	Adopted as meeting the needs, and the logic and code structure are clear, simple, and readable
My modification: none		

Errors:

Error message: Object reference not set to an instance of an object

Occurrence upon:
1. Clicking on GameObject for decorations in the background
2. Checking if the name property of an EVO is blank
3. Checking if the EVO property of a SlotContent is null

Source	Explanation	Suggested Solution	Adopted & Reason
Unity Forum[7] forum.unity.com/threads/object-reference-not-set-to-an-instance-of-an-object-c.226430/	The object does not exist.	Make sure there is an object with the name	Not adopted because it is not the case and problem I met
StackOverflow[8] stackoverflow.com/questions/64589974/nullreferenceexception-object-reference-not-set-to-an-instance-of-an-object-un	The script is trying to refer to an object that is empty.	Make sure all variables have set values and are not null	Adopted as meeting the need and critically points out the mistake I made
quest.com[9] www.quest.com/community/migration-manager-for-ad/f/forum/31022/object-reference-not-set-to-an-instance-of-an-object	The Object referred to by the "object reference" does not exist or was deleted or cleaned up.	Check if the object is null with conditional statements before using	Not adopted because I need to use the object without error, not just to avoid the error

My explanation of the error:
1. The clicked GameObjects were duplicated from the card template with the draggable tag but without properties necessary to further run the drag-and-drop.
2. Instead of being blank, the name property of EVO does not exist as the EVO does not exist.
3. Instead of being null, the EVO property of SlotContent does not exist as the SlotContent does not exist.

My solution:
1. Remove the draggable tags from the GameObjects
2. Instead of checking the name property, check if the EVO is null; as EVOs always have name properties upon declaration, the circumstances of name being blank and EVO being null are equivalent
3. Initiate the SlotContents when runtime begins

Error message:
You are trying to create a MonoBehaviour using the 'new' keyword. This is not allowed.

Occurrence upon:
1. Creating a MonoBehaviour SlotAndCard inside class SloContent in order to refer to other classes and variables declared in SlotAndCard

Source	Explanation	Suggested Solution	Adopted & Reason
StackOverflow[10] stackoverflow.com/questions/40640553/why-in-unity-im-getting-the-warning-you-are-trying-to-create-a-monobehaviour-u	MonoBehaviours are Components to attach to GameObjects and cannot be created.	Create a GameObject and use AddComponent to attach the MonoBehaviours	Not adopted because generating a GameObject just to refer to classes and variables in MonoBehaviour is unnecessary

My explanation of the error:

Unlike built-in and self-defined classes, such as GameObject and SlotContent, MonoBehaviours, or scripts, cannot be accessed alone without being attached to a GameObject running the methods defined in the MonoBehaviour.
My solution: Change the datatype of parameters in conditional statements from EVO (a self-defined class in SlotAndCard) to string, more specifically the name property of the EVO, so that accessing EVO variables declared in SlotAndCard is unnecessary

As shown above, different sources have various explanations and vagueness and do not guarantee to meet my demand. Therefore, filtering the sources is crucial for efficiency in research and development. In the case of programming, interactive forums where people ask questions for others to answer turned out to be more useful than officials or authorities, which is counterintuitive as opposing the principle of past scientific research. Unofficial discussions often include more specific cases and readable solutions, and the engine enables immediate tests of information, making coding research differ from regular research. As a result, platforms including StackOverflow, GeeksforGeeks, and Quora take up more proportion in my research than the Unity and Microsoft Manuals. However, video sources, YouTube in this case, were less helpful than the manuals. Although videos provide step-to-step guidance and screen view during the development, the information takes more time to obtain, understand, and test. Not until having watched the videos completely could I determine the ways to test and whether I could adopt the methods. Moreover, capturing certain sections of information from videos with progress bars is harder and less efficient than from scrollable web pages.

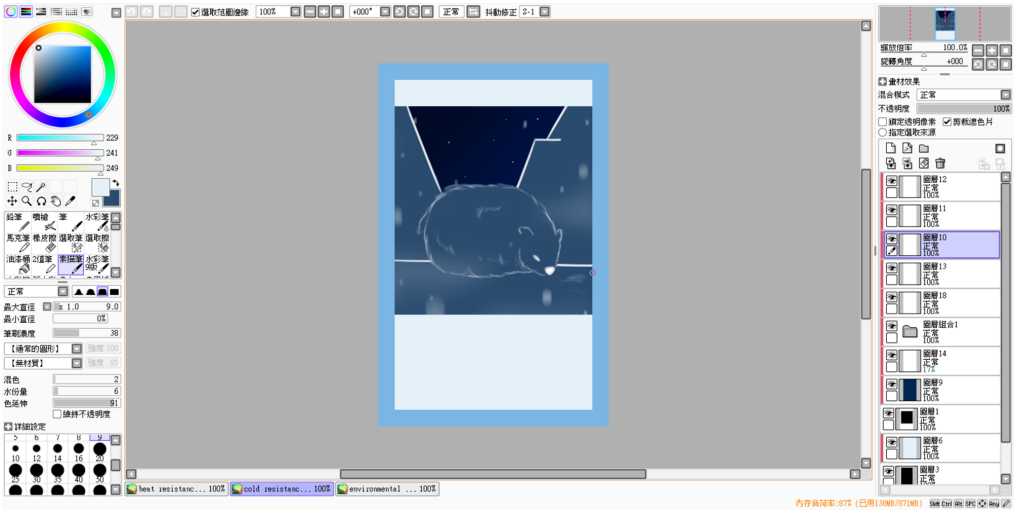
During and before the research, I reached a quick conclusion about the balance of art versus code: within standards below masterpiece, coding takes more time and effort than art designing. While scripts must be rigorously correct, visual art, without correct answers, allows more inaccuracy. Moreover, improving the image after completing the base leads to unproportionally much time spent on little effects. Therefore, I did not do extra research and spent much less time on the art work. I am not very surprised, but a little disappointed with the answer to the first half of my learning goal. I am quite satisfied with the second half, however, as the development went smoother and smoother towards the end.

II. Achieving Product Goal

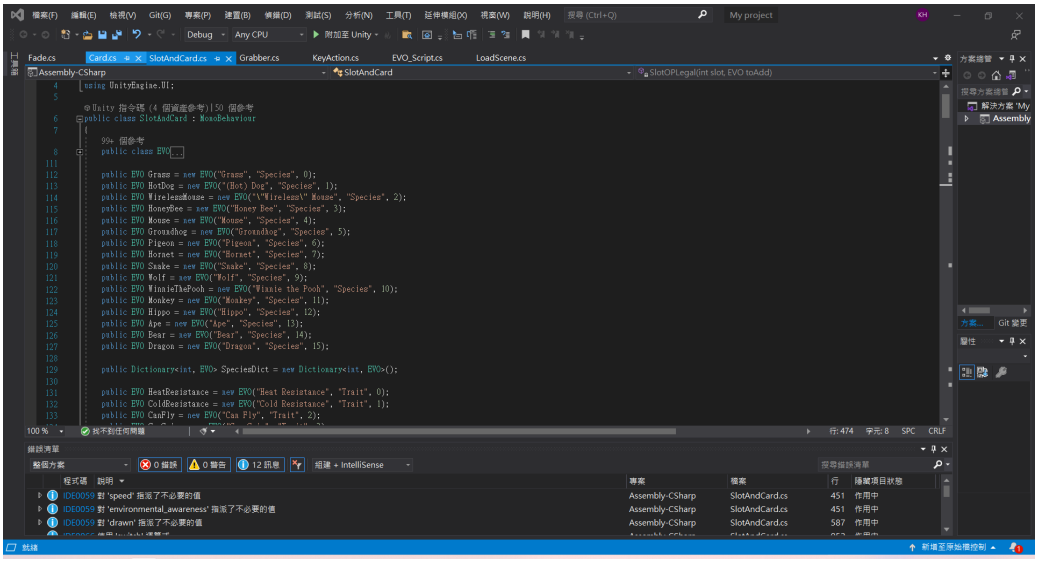
ATL Skills: Self-Management (Organizational), Thinking (Creative & Critical Thinking, Transfer)

My product goal also consists of two halves, the execution file of the video game and a document explaining the code implemented. The video game can further divide into art designs and scripts.

The art design consists of creative thinking more than other ATL skills, especially for the design of trait cards. While species and effects are easy to find references and draw, traits, as invisible or abstract concepts are hard to visualize. In the end, I determined, as species are the core of the game, I would demonstrate the traits with associated animals. This did not apply to traits *Immunity* and *Trait Extension*, however, because immunity is not a trait specific species would have, and trait extension is not a real trait. Below is the view in Paint Tool SAI, the drawing tool I used for the card designs. The card in the image is the trait *Cold Resistance* symbolized with a polar bear sleeping in snowy ice mountains.



For the game back end, while the information from forums were readable and testable, the testing from source to source was still time consuming, challenging, and frustrating, especially when finding multiple solutions from the research that were not compatible with each other. As a result, I developed the skill of establishing more effective methods of organization. I divided up the scripts, and in each script I separated the functions and variables by purposes. For example, in the script SlotAndCard, I have the self-defined class “EVO” and functions to create and initialize the “EVOs” in the top section, variables related to hand types in the next section, etc., and functions to actually run during runtime in the final section. The following image evidences my script organization. The different scripts are shown in the tabs on top, and the main screen is the first section, “EVO”, of the script SlotAndCard.

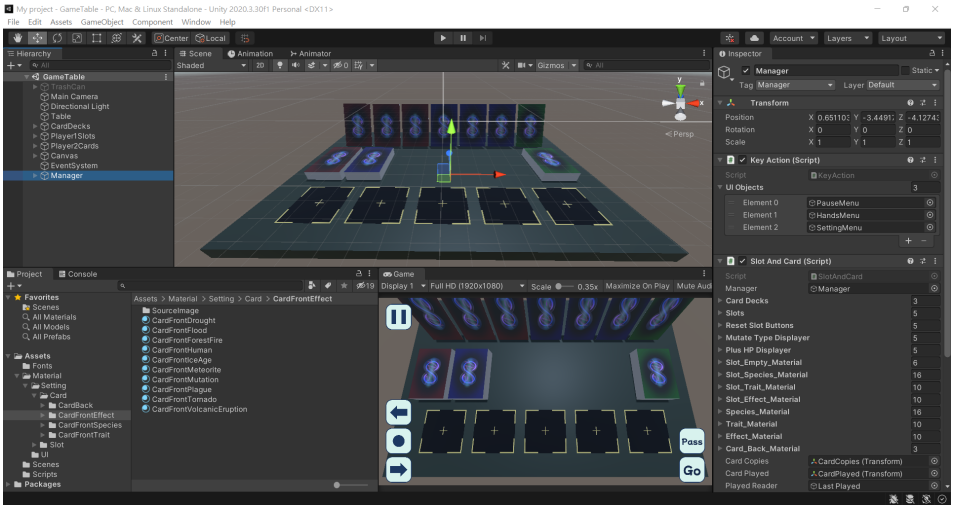


This separation is more useful than alphabetical order when testing, combing, and adding my own codes into methods found in the research. However, codes designed by myself, which takes up the majority, as well as my edits in the inspector screen often generate errors, presented in the research section above. The most serious and frequent error was “Object reference not set to an instance of an object”. The error stopped any further operations, so I could not check things after the error-generating line before I solved the problem. Although as shown in the research section, I found a feasible solution, I applied the solution in the wrong way that made my scripts run without errors but lose functionality.

Moreover, I tried to pass and use EVO type variables through functions, such as creating an EVO *a* that equals the existing EVO *Grass*, which I thought would create a clone of *Grass* with the name *a*. However, this actually makes the names *a* and *Grass* refer to the same EVO, meaning modification to *a* would also change the properties of *Grass*. I took a long time to understand this phenomenon and still could not find a clear solution.

For both the first problem, scripts without errors but also functionality, and the second, changing properties of objects I do not want to change, I used the Debug.Log function in all possible if/else paths to identify where the script begins to run not as expected. Even after identifying the location of errors, I spent much time understanding the logic of my codes that caused the malfunction, but the long thinking led to the moment I demonstrated my critical thinking and problem solving skills the most. In the two sections with errors, despite the different purpose and the different types of operations, I saw the similarity in one of their possible solutions: I need a new, complete object, an object with complete parameters, and new so distinct from all the other objects. Therefore, I came up with initialization along declaration as the solution to both problems, creating a new object so it would not be null, creating a new object so its name would not refer to objects I did not want to modify, and my previous learning allowed me to construct the codes to create and initialize self-defined objects with the efficiency and convenience of OOP. This is the critical point in my coding, as I had not thought of initialization not in the beginning of running codes before. After I solved the two major problems, I transferred the concepts and structure to other sections and began using this method more frequently in functions such as clearing slots and restarting the game.

In the end, I was able to expand my scripts quickly and build my scenes. The following picture shows the scene I built in the engine. The list on the left shows objects created and displayed (some are hidden, such as the pause menu); the inspector on the right shows components of the object, including text, renderer, and the scripts I typed and attached.



The last part of the product, a separate document explaining the logics used in the development of the game, came straightforward and less time-consuming, as the document was a transfer and translation of what I had already typed in the scripts, and I had the organized information from the research to help with the accuracy of explanations. The following picture shows a sample page from the document, listing out objects I created in the codes. For example, the first block below the title *Self-Defined Structure/Class* represents the EVO class mentioned before, which was not default and was created by myself. As shown in bullet points in the block, an EVO has a name, belongs to a category, has an index in the dictionary, has specific traits and corresponding levels, and has a mutant form (if the EVO is a species).

Initialization

Self-Defined Structure/Class:

Class 1: EVO (the exact Species/Trait/Effect)
- Name
- Category (Species/Trait/Effect)
- Index in Dictionary
- Trait Parameters
- Mutant Form (EVO)

Class 2: Slot Content
- EVO in Slot
- Mutate Type
- Additional HP

Reference Lists:

To Read:
- Species Dictionary
- Trait Dictionary
- Effect Dictionary
- Hand Types

To Modify:
- Slot Content 1~5

Template:

Card Template