## Component Design and Integration Plan

1. **Determine the required programming systems and components.**

*Analyse and evaluate the functionality required by the project to determine the components and systems that need to be created.*

* Summarise game concept: clearly declare what considerations need to make for components
* Components: nodes, battle system, health display,
* Purpose of components, what do they do?
* Provide brief summary of purpose

Feral Kingdom will be a Pokémon-esque turn based combat game with an overworld to move between battles, the player will be able to complete a battle either by killing the opposing monster in the battle or by running away from the battle, the latter will not be marked as a win state, and thus the player would have to come back and complete the fight to be able to beat the game. Considerations need to be made with the monster components as to how the enemies and player take damage and deal damage to allow the system to be reused for each monster type to make each monster’s play style be unique from the others. There needs to be a system to control the battles and to switch between the player and enemy turns, a component that allows the player to heal their monster, a system to select between the different monsters at the start of the game.

The components needed to fit with the concept and function within the game include, a system that manages the monsters health and damage, a system to allow the turn based combat to function, a component that allows the player to enter into the battle from an overworld and a component that tracks the amount of battles completed to allow the player to complete the game after a certain number of battles won. A system to save the players monster’s health value throughout the game is required for full functionality.

The monster health and damage system is required to give the player challenge and to allow the whole battle to take place, this will assign health values to the monsters, a minimum to maximum damage range for the monster to apply a random value between the min and max to deal that to the opposing monster.

The battle management system will control the swapping between the player’s and enemy’s turn and restrict the actions of each to only occur during their respective turns and to control the win and lose conditions of the battle, ending the battle when they are met. This forms a structure for the player to play within.

Overworld battle nodes, or the component that allows the player to access the battle will need to load the specific battle that the player wants to access and provide a visual indicator of a location the player can access.

The system to store the monster’s data will need to be present throughout the game and store the monster’s current health and allow access to other scripts to load the data from this component.

A Main menu component will need to have functionality to be able to select the monster that the player wishes to use throughout the game and clear any saved data when a new game is started.

A player movement system for the overworld will allow the player to move between different battles and locations to either battle, heal or end the game, this system can be simple, only moving the player horizontally.

A camera component that follows the player while they are in the overworld would need do only that, follow the player.

A component for healing the players monster would need to accessible to lower the overall difficulty of the game by still leave plenty of challenge for the player. It would only need to do heal the players monster.

A way to beat the game, a Finish Line component, it would need to have certain conditions that need to be met before the player can beat the game. It would need to check if these conditions have been met before activating.

1. **Document your process for analysing, evaluating, sourcing, and selecting components.**
   1. *Analyse the functionality required by each component.*
   2. *Evaluate possible designs for each component.*
   3. *Source various references or third-party resources for each component.*
   4. *Compare the possible design solutions for each component and select the most appropriate solution for the project.*

* Describe what the component does and how it functions
* Ways of implementation, Original, new, potentially
* Resources to help design components, tutorials, guides, articles, how they help create the component
* Select the implementation method, describe why its suitable, compare new implementation against old.

Battle Manager System: Restricts the actions of the player and enemy to their respective ‘turn’ or ‘phase’ to stop the player from attacking during the enemy’s attack, executes the code to end the battle depending on which conditions are met, either making the player run from the battle, complete the battle or lose the battle. It will mainly function through the player pressing either the attack or run button, dealing damage to the enemy, checking if they’re dead from that attack, completing the battle and loading back to the overworld if the enemy is dead, or dealing damage back to the player and flipping to the players turn again if the enemy isn’t dead yet. I implemented it through a script attached to an empty game object in my first attempt at making this game, the script was long and convoluted, making it ineffective for its purpose. In the current version of the game, the script is attached to the game scene in the same way as prior, but this time the script is much shorter, only containing a timer for the turns, definitions for the player attack and win and loss conditions, making the script much less bloated and contains only the necessary coding. The original implementation was directly taken from [this Brackeys tutorial](https://www.youtube.com/watch?v=_1pz_ohupPs&t=1240s) with added functionality to serve for my specific purposes, the new system is loosely inspired by the same tutorial, but was mainly constructed through consultation with TAFE Lecturer, Joshua Ferguson, who gave instruction on how exactly to code the component to function for my purposes.

Monster Component: This component will act as the ‘Unit’, It will hold the monsters max health value, its damage range and the model to be used when battling, this is to be designed dynamically so that a variety of monsters could be created with the same component but with their own uniqueness. The first iteration of implementation that this component went through had it hold value for the monsters max health, exp and armour, its base value for health, armour and damage, and its current values for all of them, this component became broken in the first iteration with constant overwrites from a separate component and was thus scrapped. The second iteration of implementation is present in the current version of the game, holding only the monster’s max and current health, the monster’s max and min values of damage along with the necessary code for if the monster is dead or not, implemented onto prefabs of the monster’s models. The first iteration was again, thoroughly based on the previously mentioned Brackeys tutorial and improved upon to fit my vision. The second iteration was constructed with help from Joshua again, but was still loosely inspired by the tutorial.

Battle Node Component: The component acts as the interactable that loads the player into a battle scene, containing functionality to mark the battle as completed when the player beats the monster in the battle and with the completion of the battle nodes, the final node will open, allowing the player to beat the game. It functions by using unity’s built in functions for loading game scenes when the player walks inside of the attached trigger and presses ‘e’ on their keyboard to load into the selected battle and at the moment that the player loads back into the overworld scene, the node checks if the player beat the battle or not, disabling the node from being activated if the player has beaten it. The first iteration of implementation had the nodes only existing as visual game objects and the rest of the functionality was contained within a different component, the scene loader script was used to load a scene. In the second iteration, the functionality was mainly held in the one component, with the script attached to a game object with a trigger and a ‘battleIndex’ value that is set in the unity inspector that tells it what battle to load, also contains a field for the completion status of the relevant battle. The first version was based upon the previous Brackeys tutorial with several adaptions made to function with the other convoluted components, while the second version was made entirely with the help of Joshua.

Game Manager System: This system will remain present throughout the game and hold the functionality to set which monster the player is using and hold the data said monster so it can be loaded within the battle scenes and the functionality to hold the completion status of the various battles as well as the mechanics that are required as a constant throughout the game. In the first iteration of implementation, this system contained **all** the monster’s stats as individual variables, as well as holding functionality for loading and saving these stats to text files, creating a real mess of code that is constantly conflicting with each other. The second iteration of implementation only held a reference to the component that contained the monster’s stats, saving on code length and complexity, as well as containing the functionality for holding the battle completion stats and functionality for loading from the battle scene into the overworld, loading into a pause menu and functionality for clearing the data saved within the component to start the game anew. The first iteration was mainly original with inspiration from [this Immergo Media playlist of tutorials](https://www.youtube.com/playlist?list=PLy7lD4g7kKGDHdKhlWtZQ_8nGfw6z6yPH), while the second iteration was, again created with the help from Joshua to suite what was required of it.

Camera: This component would move the Unity camera it a way that follows the player at a comfortable viewing angle and rotation while not taking away from the gameplay, it should function by using the position of the player game scene to set the camera’s position to the player’s location but with a offset to allow for third person viewing of the player. The implementation of this could be done by simply making the camera child of the player game object in the unity editor or through code to update the position of the camera in real time separate from the player. I used code from the tutorial videos by Sykoo in [this Tutorial playlist](https://www.youtube.com/playlist?list=PLMKGE-XmGjMTzNpssH84xJcQ35CXJeLLl) that I made, choosing this as a more refined implementation method.

Main Menu: The main menu will work as a UI with buttons that will execute different functions within the main menu component, namely, a new game button, a quit button and monster selection buttons. The new game button will clear all saved data if there is any and show the monster selection screen, pressing any button on the selection screen will set the monster relating to that button to the monster that the player will use throughout the game. It will also be used when pausing the game. The first implementation iteration had the main menu existing in the map scene and disappearing when you pressed button, the second iteration was its own scene, making the main menu its own scene means that there is much less to worry about with restricting player movement and having the screen pop up when you load back in from battle.

Player Movement: This component will simply translate the player object depending on the button the player presses, w for forwards, s for backwards and so on. The first iteration of this include checks for triggers for different battles and locations as well as saving the last position of the object for when the player loaded back in, the second iteration became much simpler in that it only does the movement and nothing else.

Heal Node Component: This component will be located in the overworld to allow the player to heal their monsters outside of battles and will only activate when on top of it and pressing ‘e’, it works by applying negative damage to the players monster, resulting in a positive increase in the monsters health. The first iteration of this was a button that you accessed by visiting a separate place and once pressed it would heal the players monster to full health, the second iteration is simply an area you stand in and press a button on your keyboard to heal the monster in lots of 50.

Final Node Component: this will act as the finishing line for the player, once the player has beaten the required number of battles and stands here and presses ‘e’ the game will revert to the main menu. This component will need to check for the completed battles when it is activated. This could be implemented by having a game object that only spawns in when you’ve beaten all three battles, a object that is constantly present but only activates when all battles are one or as a sequence that activates at the end of the final battle. I chose to have it as a constant object that gives a visible goal for the player.

1. **Evaluate the technical impact of each component on the overall design of the project.**

*Consider how the components will impact on the overall design of the project. Address how the components will be integrated into the project with consideration for the software being used in the development pipeline.*

* Outline when, how, why, what they communicate with, when they make the connections (references made, find components etc.) and reason they make this connection

Monster Component: this component doesn’t initiate any of the communications with the other components but is accessed and referenced by the Game Manager and Battle Manager.

Game Manager System: this system is connected to nearly every other component, it personally connects with the Monster component when the player selects a monster at the new game screen, referencing the monster stats of the prefab of the monster that was selected, accesses it to setup the monster for future use and for use in referencing the game object and model that the component is attached to. It finds all the Battle Nodes in the scene when the player goes to leave a battle to access the functionality of the node and run the code for the nodes completion if the player had won the battle. This system once more makes reference of the Battle Node component when the player activates the final node, accessing the component to check if all the battle nodes are completed.

Battle Node component: this component accesses the Game Manager component at the first frame of the scene, checking if the ‘battleIndex’ value of the battle node is present in its table of battle nodes, adding it if it isn’t or loading the completed status of the node from the table if it is already present. It accesses the Game Manager again when the player enters its trigger to check if the players game object has the Player tag on it and once more to load into a battle scene to set the current battle variable in the game manager to the battle that is being loaded into.

Battle Manager System: The Game Manager system is accessed at the very first frame to retrieve the players Monster stats saved in it as well as the model for the monster, the stats are used to set the players health UI to the correct values at this time as well, It also accesses the monster component of the instantiated enemy monster as to ensure the enemy’s health UI is displaying the correct values as well and that the enemy is at max health for the start of the battle. The enemy’s and player’s monster’s component is then accessed again when the player attacks, applying a random value between the min and max damage of the player to the enemy and checking if the enemy is dead, if it is dead the Game Manager is then accessed again to check if the current battle exists in its battle node list, if it is it sets the value of the current battle to be completed and loads the over world, if the enemy isn’t dead the enemy’s health UI updated with the accessed component’s values. The player’s and enemy’s monster components are accessed again during the enemy’s turn to apply the random number between the enemy’s min and max damage to the players health and to check if the player is dead, accessing the Game Manager system to load the overworld if the player is dead, or updating the players health UI with the player’s monster component. If the player chooses to press the run button when it is their turn, the Game Manager is accessed to load the overworld.

Main Menu Component: this component will access the Game Manager at the first frame to reference the ‘lastSceneIndex’ variable for use for checking if the game is paused or just started. The Game Manager will be accessed once the player presses the new game button to clear any saved data, then when one of the selection buttons is pressed for the monsters the Game Manager is accessed to set the current monster as the selected one.

Player Movement: Player movement does not reference or access any other components but is essential to move into a game object to activate it.

Heal Node: The healing component checks the players game object for the Player tag and if ‘e’ is pressed, it accesses the Game Manager to access the monster component that it is attached to, healing the monster by 50 points of health.

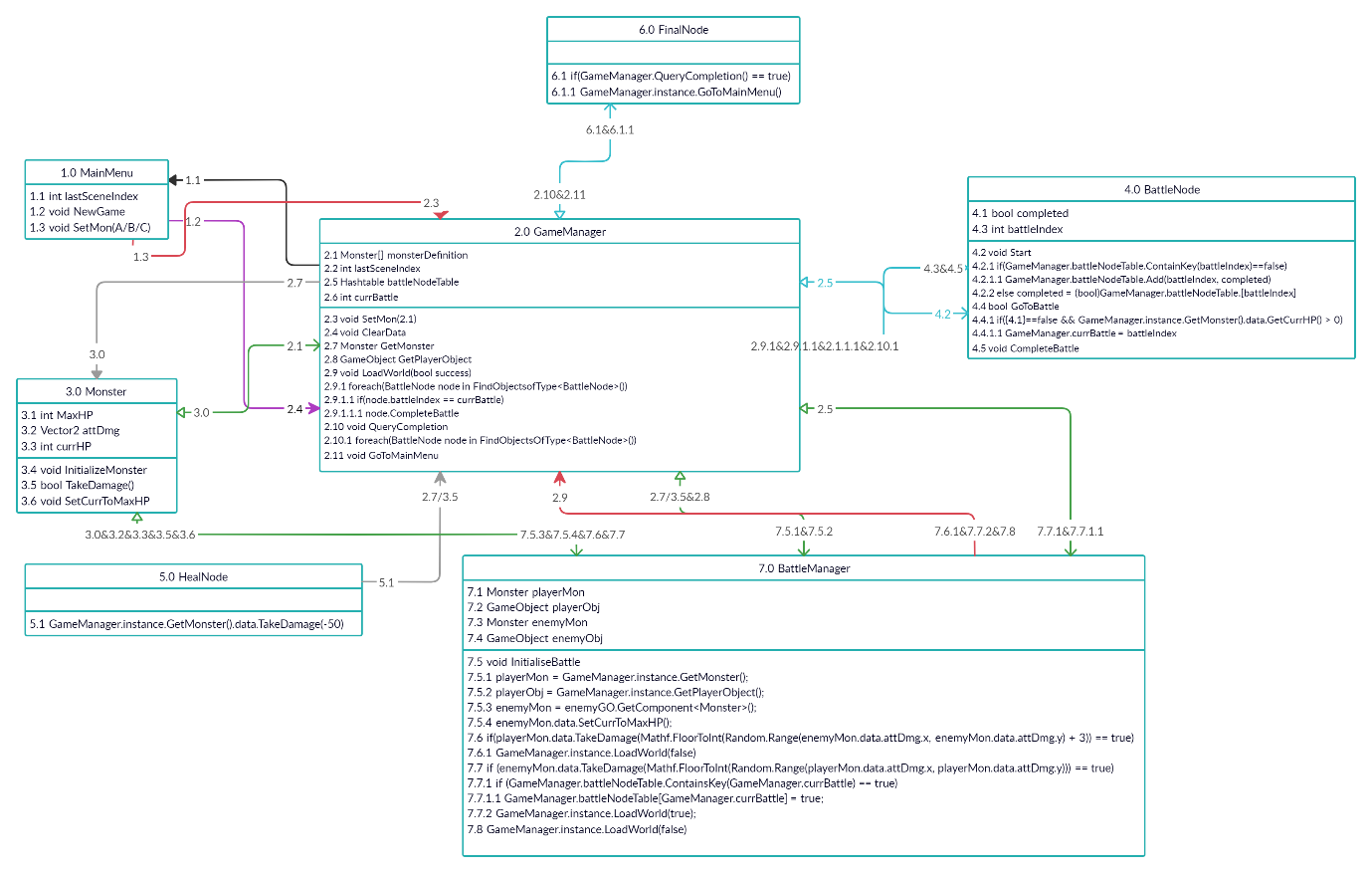
Final Node: The ending component checks the players game object for the Player tag and if ‘e’ is pressed, it accesses the Game Manager to check if all of the battles have been beaten, taking the player to the main menu through the Game Managers functionality if they are.

Camera: The camera only references the transform of the player game object to be able to move according, it does not access or reference any other component.

1. **Create a domain model for the overall project detailing how each component/system responds to one another and their flow of data.**

*Insert a picture of your domain model for your project. Ensure you specifically outline the data that each component sends and receives.*

* *How everything is connected*



1. **Address the cost of implementing each of the components.**

*Using the three-point estimation formula, determine the estimated time that each component will take to implement into the project to achieve the correct functionality required by the overall project. Explain the elements of development taken into consideration that were used to determine your optimistic, pessimistic, and most likely estimations for each component’s calculation.*

* Apply to production cycle as whole, not individual components
* Provide overview of pessimistic, optimistic, realistic scenarios, how long components construction will take according to each scenario
* How long, times by the avg wage of a software dev, Hours just sitting at comp,
* List time for each components and implementation time

The Pessimistic scenario takes into consideration the worst possible extent of my procrastination and laziness, my tendency to get distracted easily and my inexperience in the field of coding, assuming that I spend at least a quarter of the time it takes for each component watching tutorials and testing code and correcting code to fix bugs taking up another quarter of the time.

In an Optimistic scenario, a quarter of the time is taken up by the testing and debugging, with the rest being spent on implementation and code writing. This scenario assumes that all the code that needs to be written, won’t require much in the way of tutorials and can be done mostly from memory or prior knowledge.

The most Likely scenario would have a moderate amount of time for tasks taken up by tutorials, but the testing and debugging wouldn’t take up as much time because of the information provided by the tutorials, while implementation will still take a reasonable time to complete because of the beginner level of coding knowledge.



1. **Address any possible licensing issues in regard to third party components or scripts being used.**

*Identify and explain any possible licensing issues that may arise in relation to third party components or scripts being used, or code used from any external resources (such as tutorials, public repositories, asset stores, etc.).*

* List third party assets used,
* Address third party assets and their licensing
* Explain the ‘Fair Use’ of tutorials

The licensing for the font that I have used in my game, allows the use of said font within any publication, only requesting contact with the author of the font if it is used, as they would like to receive a copy of, and/or help further the publication with custom made fonts or clip art. This font is a so called “postcard-ware”, an uncommon form of novelty shareware, requiring the user of the font to send a postcard to the author for the user to be allowed to use the font, as it is a novelty shareware form, its generally regarded that the postcard is optional.

The font used in the UI is called RunEnglish and is from this link: <https://www.dafont.com/runenglish.font>

“These fonts cannot be sold for a profit. (Although, if you really enjoy using these fonts and need to find some financial way of showing your appreciation.

Any and all donations will be gladly accepted.) CD-ROM and Shareware

distributors are requested to notify me before distributing these fonts.

These fonts may be used in any publication. Please contact me if these fonts do appear in a publication. I'd like to receive a copy of it, and I may be able to provide further assistance by creating custom fonts or clip art for you.

These fonts are Postcard-ware. If you like these fonts, please send me a

postcard or letter (using an attractive or interesting stamp). If you supply me with your email address, I'll try to write back. You can get my current postal address by visiting my WWW home page, listed below. If you discover any errors, inaccuracies, or problems with these fonts, or this document, I encourage you to contact me. My current postal address is listed on my WWW home page, or you can email me.

©copyright - 1996 - Daniel Steven Smith

email: fontmaster@geocities.com

WWW: <http://www.geocities.com/TimesSquare/4948/index.html>”

The code produced by following [this Immergo Media playlist of tutorials](https://www.youtube.com/playlist?list=PLy7lD4g7kKGDHdKhlWtZQ_8nGfw6z6yPH) and [this Brackeys tutorial](https://www.youtube.com/watch?v=_1pz_ohupPs&t=1240s) is considered and falls under ‘fair use’. The ‘fair use’ of a product is determined by 4 factors, The purpose and character of the use, including whether such use is of commercial nature or is for non-profit educational purposes, The nature of the copyrighted work, The amount and substantiality of the portion used in relation to the copyrighted work as a whole and The effect of the use upon the potential market for, or value of, the copyrighted work. The code produced is being used for a non-profit purpose as the game is being submitted as evidence for a TAFE assessment and is being used in a “transformative” way that differs from what the code was originally for, as it is code it could be considered intellectual property but as the use is of a educational nature it is still fair use, the amount of code that is still 100% a direct copy from these tutorials is 0 and all of the code has been altered to fit with what I needed them for and as the production of this game is for a non-profit and educational purpose there will be no negatives or damage to profit for the owners of the tutorials. Because all of this, this game does not infringe upon the copyright of the tutorial makers.