Critter Collector

Frontend Documentation

Last Updated: 11/27/2021

Overview

Critter Collector is version 2 of an educational location-based mobile game being developed at the University of Florida. The game is similar in concept and design to the mobile game Pokémon Go™, but uses real animals based on the region the player is playing in the world. The goal of Critter Collector is to educate users about different animal species in their area as well as providing them enjoyment. The original prototype [Jei's Jungle] was built using the Unity game engine. Critter Collector is a rebuilt version of the prototype using Unreal Engine.

Current State

As of writing this documentation, Critter Collector has a semi-completed rebuild of the prototype. The current features include:

- Mobile Build to Android Devices
- Realtime GeoLocation Updates
- MapQuest Integration (displaying the user location on a map)
- Menu Screens: Home, Settings, Game Overlay, Encyclopedia
- Music & Sound Effects
- Local Frontend Saved Game-State
- Game Layout/Player Controller
- Backend connection to retrieve animal data
- Spawning Animals to screen

General Info

Critter Collector is built using Unreal Engine 4 version 4.25.4. This choice was made during the research phase of development in the interest of integrating older plugins to streamline parts of the game. This is no longer relevant as the plugins were not needed, or not applicable later in the development cycle. The project was generated using the following settings:

- Template: Third Person
- Structure: Blueprint
- Quality: Maximum
- Raytracing: Disabled
- Target Hardware: Mobile/Tablet
- Starter Content: Yes

Below is the current project flowchart to demonstrate how the game works at a high level. It should be noted that this chart does not include the Animal Screen described in the Menu Structure section of this document, as it is not currently implemented.

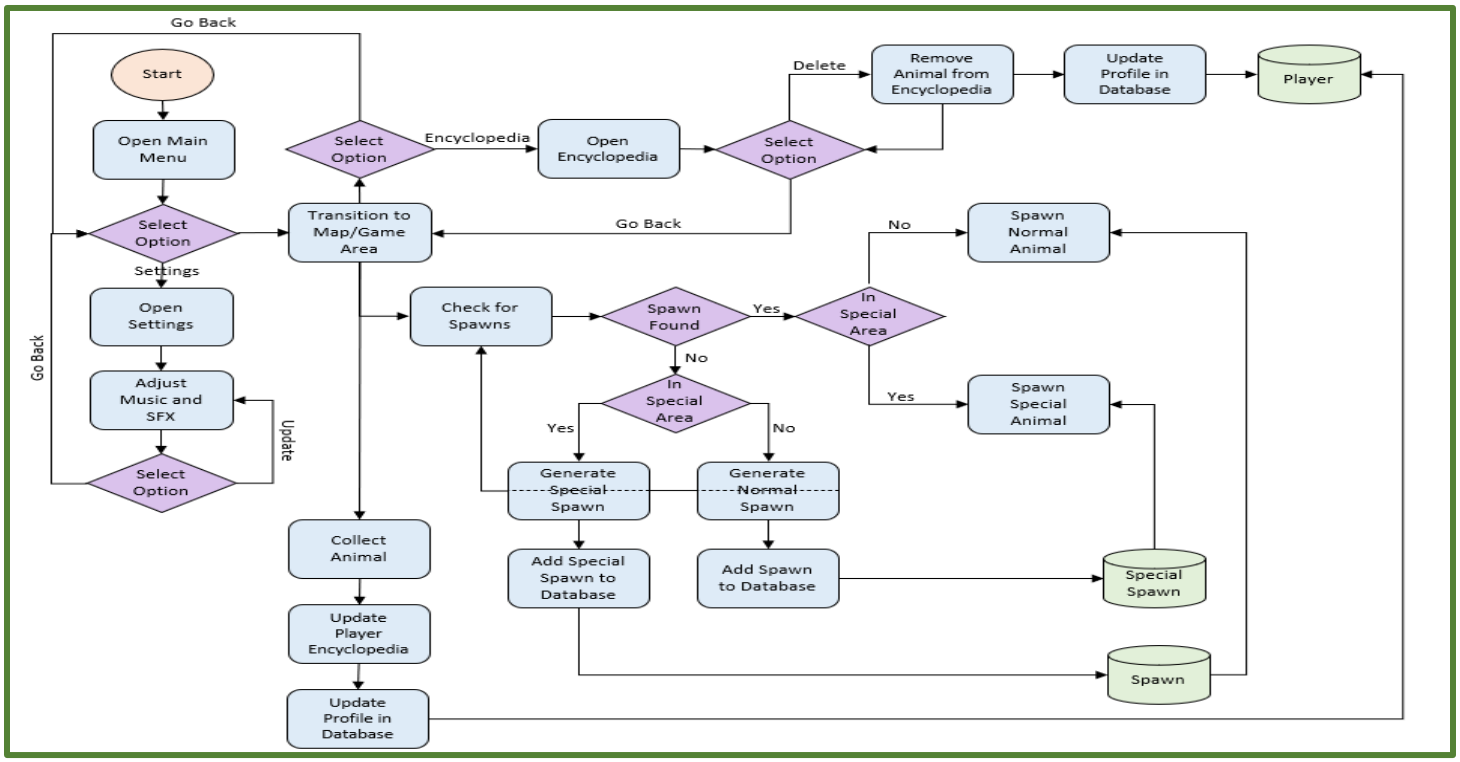


Diagram 1: Project Flowchart as of 12/2021

Project Setup

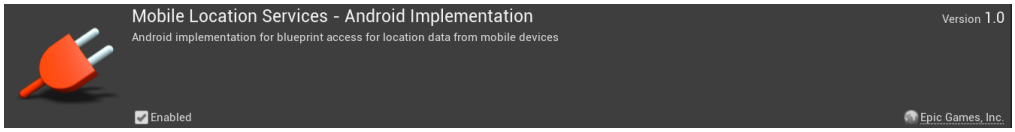
To set up this project locally you will need to do the following steps. Note this is only the setup process for Windows, with the intention to build the game on an Android device. For other machines/devices please review the complete Unreal Engine Documentation:

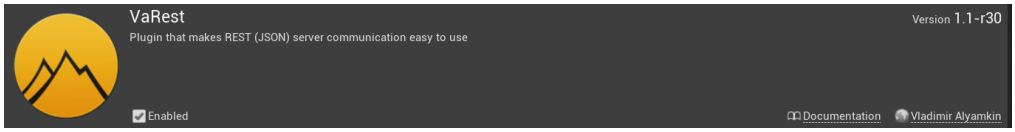
1. Download Epic Installer to get the Epic Launcher
2. Download Frontend Repository from Github & unzip it to a directory.
3. Download Android Studio following the installation instructions for UE4
4. Download Visual Studio (or another supported editor for UE4 to edit C++ files)
5. Open the Epic Launcher & go to the Library tab
6. Click the "+" icon next to *Engine Versions* & select 4.25.4
7. Click install & wait for installation to finish
8. Launch Unreal Engine 4.25.4 & click on the *More* button
9. Click the *Browse…* button & navigate to where you unzipped the project from step 2
10. Choose on the *.uproject* file (currently named CritterCollector.uproject)
11. Click the *Open Project* button in the launcher & wait for the engine to load the game.
12. Check the project settings for the SDK requirements in Edit>Project Settings>Platforms>Android SDK.
    a. The SDK, NDK, & JDK should all be set. If they are not set refer to the installation instructions in step 3.
    b. The JDK does not have to come from Android Studio, you can use the one from your Java installation on your computer if you have it. Normally located in Windows at *C:/Program Files/Java/jdk[version info]*
13. Check the project settings for the *Location Services – Android* section in Edit>Project Settings>Plugins>Location Services – Android. Make sure all 3 check boxes are enabled.
    a. If you do not see this in your Project Settings, please review the Plugins Section of this document to ensure you have all of the added plugins enabled.
14. Plug your Android device into your computer (your device must have developer mode turned on first)
15. From the main screen in Unreal Engine click the *Build* button & wait for it finish.
16. From the main screen in Unreal Engine click the down arrow next to the *Launch* button & select your device. Your device must be on & unlocked. The build to your device will take several minutes to complete but should open automatically.
    a. If you don't see your device in the dropdown list, click the *Device Manager…* option to find it. If it still doesn't appear check the UE4 Android Setup.
17. The first time it launches it will ask for location permissions, click allow & the close the app and relaunch the app on your device. You should see a map of your location.

Plugins

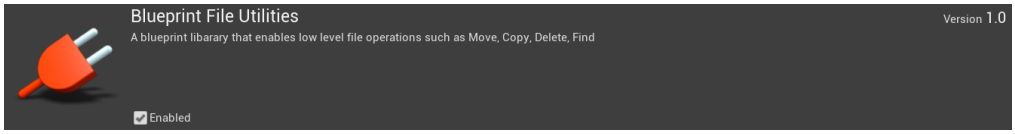
The following plugins are required for Critter Collector. Verify before launching to a device that the following are *Enabled* in your Plugins menu.

Required:

- Blueprints: Mobile Location Services – Android Implementation
  - Used for accessing location data on android mobile devices
  - 

- Network: VaRest
  - Used for making REST (JSON) calls to backend
  - 

Suggested:

- Blueprints: Blueprint File Utilities
  - Used for low level file operations during development
  - 

Upon downloading the project and opening for the first time, these requirements should be detected by the engine and automatically added to your plugins. If you manually add them to the project you will need to restart Unreal Engine to use them.

Note: In the future should IOS implementation be desired, the appropriate plugin requirements can be found in Edit>Plugins>Blueprints. You will also need to set the IOS requirements in the Edit>Project Settings

Menu Structure

The game requires at least 5 screens for base functionality. Below are wireframes showing each screen and the navigation components for them. At this time the Animal Screen is not implemented. Each of these is implemented as an Interface Widget Blueprint

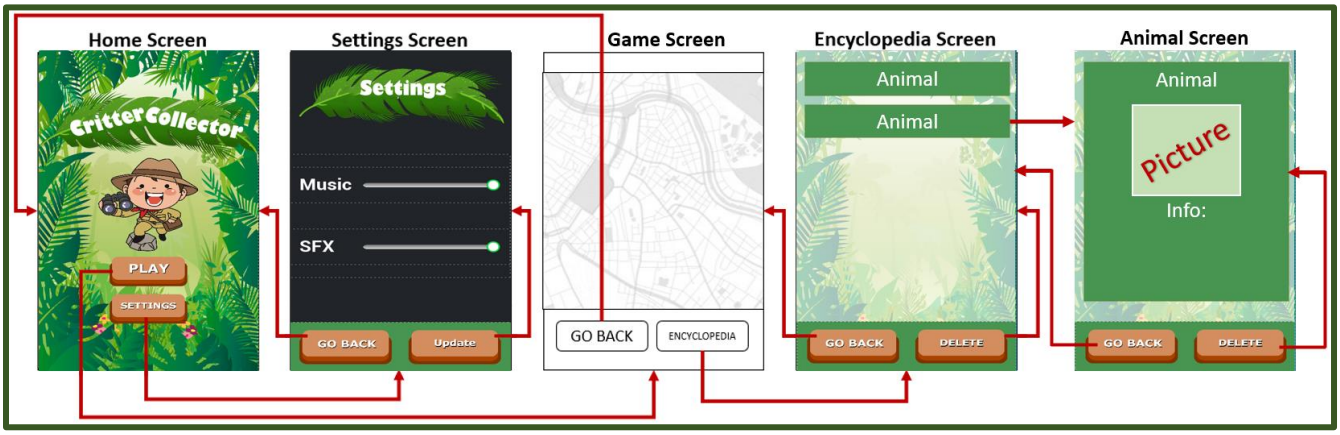


Figure 1: Game wireframes and navigation layout

Home Screen:

- The MainMenu Widget is generated on game launch. The Play button generates the GameOverlay Widget for the game screen, and the Settings button generates the Settings Widget.
- Note the Play button is the only component in the game that uses *Remove All Widgets*. This is only used as a safeguard against interfaced generated memory leaks.

Settings Screen:

- When the Settings Widget is constructed the game checks if a SaveGameSettings blueprint class and save game slot called GameSettings are generated. If they already exist this sets the values for the Music and SFX sliders.
- This Widget is used to update the Music and SFX classes. When the Update button is hit the value (between 0.0 to 1.0) is saved as a float to the SaveGameSettings blueprint class. After the save state is reloaded and the new sound values are passed to the NewSoundMix Class which operates as the master audio mixer to control the Music and SFX. Clicking Update will overwrite the previous GameSettings save game slot.

Game Screen:

- The Game Screen uses the GameOverlay Widget, which is a static overlay that adds the Go Back button and Encyclopedia button to the bottom of the player's screen. The back button will return to the Home Screen and the Encyclopedia generates the Encyclopedia Screen.

Encyclopedia Screen:

- This Widget is where the player will see a list of all the animals they have found in the game. The back button returns to the Game Screen, and the delete button will allow player to remove animals from their list on the screen. Clicking on an animal in the list will open the Animal Screen for the selected animal.
- Note currently the encyclopedia does not generate the list of animals to the screen. Therefore, the delete button has no functionality and there is no way to access the Animal Screen.

Animal Screen:

- This will be a dynamically created Widget that uses the selected animal's ID and data from the backend to generate the information for the page. The back button will return to the Encyclopedia Screen and the delete button will remove the animal from the user's list of animals then return to the Encyclopedia Screen.
- Note at this time the Animal Screen has not been implemented.

Sounds

Currently the sound structure for the game consists of the NewSoundMix Class and the Music/SFX Sound Classes. The NewSoundMix is a Sound Mix Class meaning it can be used as the master audio mixer. Both the Music and SFX Sound Classes are set as array elements in the NewSoundMix Class. The sound volumes settings are saved and updated in the Settings Screen.

Any additional future Sound Classes should be added as new elements to the array in NewSoundMix to avoid audio errors such as double effects. When adding new sound effects or music to the game, they must be the *.wav* file type. Once added to the files right click on them and select *Create Cue* to generate a usable version for the game. Open the Cue and added it to a Sound Class (Music or SFX are the only options currently).

It should be noted that the *easy-lemon_Cue* has looping enabled so the game music will persist until the app is closed.

Save Game

The frontend of the game has a local GameSettings save game slot that is used to maintain the sound volumes between play sessions. This is not generated until the first time the player changes the volume in the game. Once created it will automatically load when the game launches each time and anytime the Settings Screen is opened or updated.

Once generated the SaveGameSettings Class is constructed which holds all of the values and information for the game. Currently there is only an array of floats called SoundsVolumes in the class. In the future this file could be updated to include the player's collected animals' data to reduce load time from the backend.

MapQuest API

Located in the *APICaller* blueprint and the *MapBillboard* blueprint, the game map comes from a call to the MapQuest API. The custom function *Start Map API Caller* is called when the game launches. This function will build the URL that is required for the API call to MapQuest for static images, and when ready will call the custom function *Download Map Billboard*. This function takes the URL as a string and passes it to the *Download Image* node. When complete the image is passed on for binding.

The *Start Map API Caller* is only called once, however the *Init Location Services* node discussed in the GeoLocation section of this document, creates an internal loop that will trigger in the function at a given frequency. To prevent race conditions during the API call process, there is a delay buffer added with a built-in check if the download is finished.

The MapQuest API requires the following components in the URL to work:

1. API Domain: http://open.mapquestapi.com/staticmap/v5/map?
2. API Key: key=2tQI9RZrN2U4FWcEstN19mMN1QGIvnxV&
3. API Coords: center=[LATITUDE],[LONGITUDE]&
4. API Format: zoom=18&format=png&size=@2x

- Example URL: http://open.mapquestapi.com/staticmap/v5/map?key=2tQI9RZrN2U4FWcEstN19mMN1QGIvnxV&center=29.6471,-82.3472&zoom=19&format=png&size=@2x

Note that the use of MapQuest is not required. The MapBox API for static images is a viable alternative. The node sequence in the APICaller blueprint would just need to be updated to the correct URL formats

Image Binding

Once the static image is returned from a Map API, the image is set as the texture parameter for a dynamic material as seen in Figure 2 below. This dynamic material is created on construction of an object when the image is set in the *MapBillboard* class as seen in Figure 3 below.

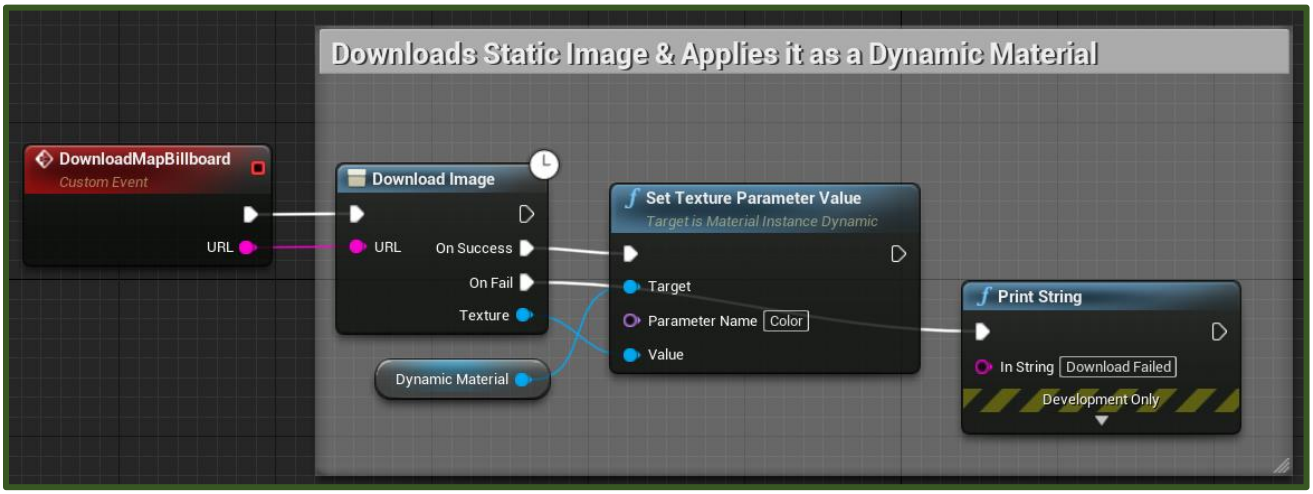


Figure 2: Following Download the image texture is applied as a dynamic material

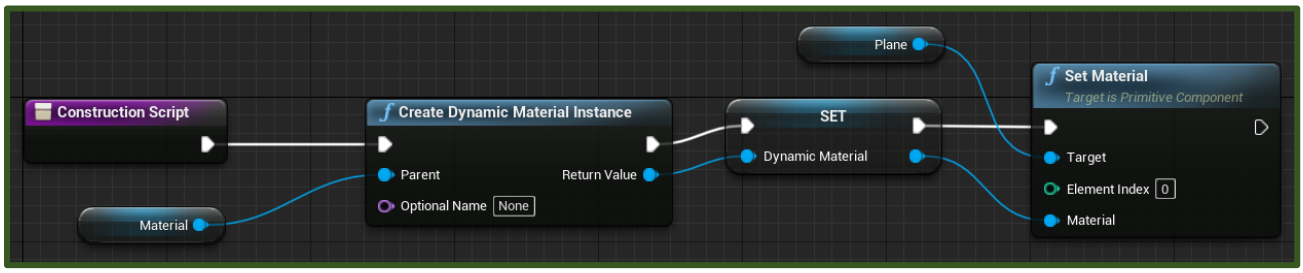


Figure 3: The Dynamic mMaterial is created as an instance of a material and applied to a plane

This will allow the map to appear in the game and has collision as an object so the player's character can stand on it. Since the API gets an updated image every few seconds this will automatically update the same object with the new texture in real time.

Note that the *Material* node in Figure 3 has a default material set as *DownloadedMaterial* which is custom material file with a material called *Color*. That same name is used when the map texture is applied as a parameter in Figure 2.

GeoLocation Service

The GeoLocation Service is split into two pieces in the Level blueprint (currently called MainMenu) and the *APICaller* blueprint. The section in the Level blueprint will only run once but sets up how often the rest is repeatedly updated.

In the Level blueprint (Figure 4), the *Init Location Services* node is called. This node controls GeoLocation accuracy and update frequency in milliseconds. This must be called first for location services followed by the *Start Location Services* node. If the service is initialized and starts correctly then an instance of the *APICaller* class is created and the *Start Map API Caller* function is called.

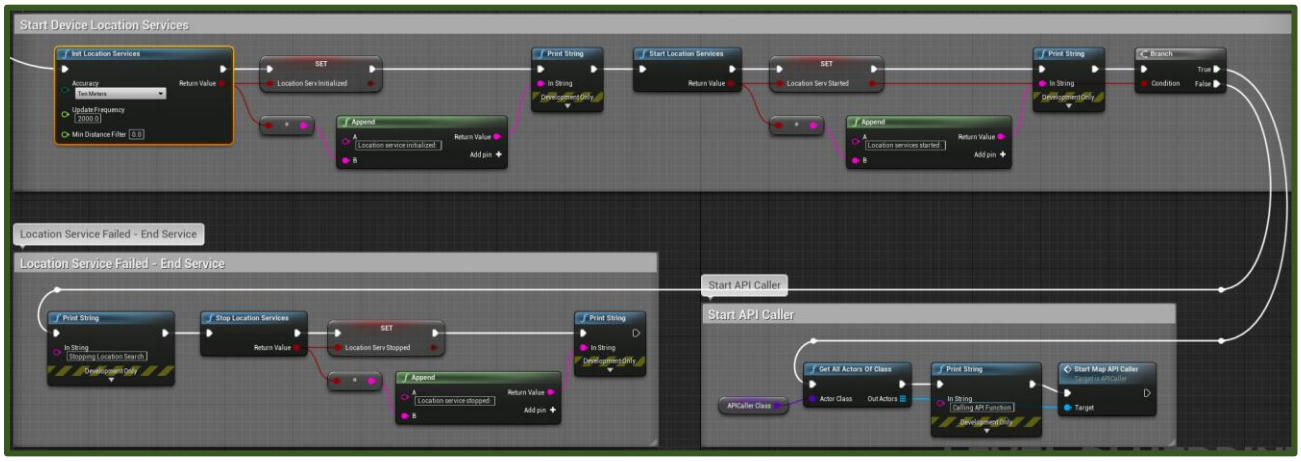


Figure 4: Level Blueprint of Location Service Initialization & Start

In the *APICaller* blueprint (Figure 5) the *Get Location Services Impl* node creates a location object and passes it to the *Bind Event to On Location Changed* node. This node will update any time the location changes, which is set by the *Init Location Services* node from the Level blueprint. Currently it updates based on time, but it can be set to update based on distance. Every time an update is triggered an *OnLocationChanged* custom event, which will split the location object into the 6 components that Android devices return. Only the Latitude and Longitude are modified and used for the Map API URL formatter.

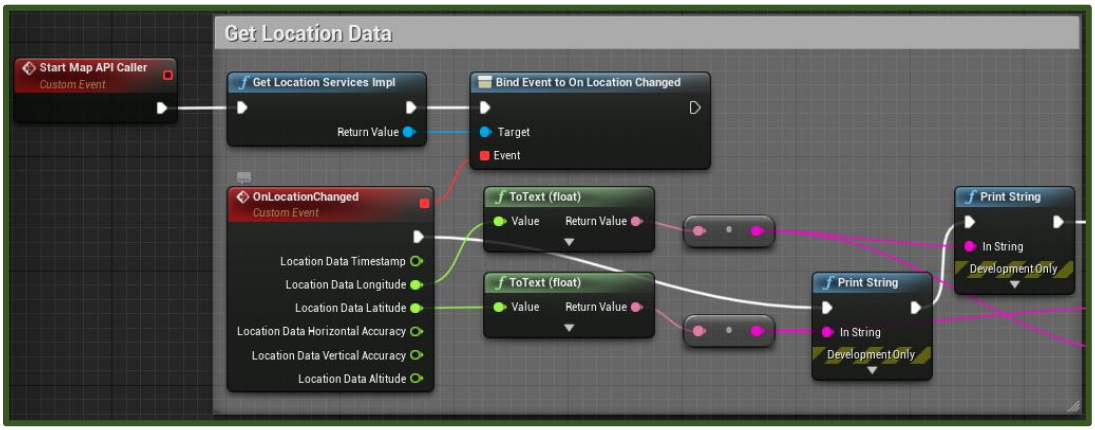


Figure 5: APICaller Blueprint Location Service Event Process

Misc

Additional items that should be noted for the project.

- When setting up the project do not place the files more than about 5-7 directories deep on your computer. There is character limit to how long the path name to build files can be, and if the path names are too long the build/launch process will fail and corrupt other files.
- When you build to your mobile device if you get an error along the lines of "…Attempt to construct staged filesystem reference from absolute path…" then do the following while your device is still connected to the computer:
    - Open CMD
    - Type:
        - adb shell
        - cd sdcard
        - ls (check if UE4Game is listed, if yes continue)
        - rm -r UE4Game
    - Delete the current game app you have on your device.
    - Then Rebuild the game in UE4 & then launch the project to your device again.
- There is currently not a .gitIgnore file in the repo. You cannot push all the build files to the repo so you need to push the changes before you build/launch to your device. If it is successful then delete all the generated build files and continue from there. A .gitIgnore file is needed for the repo but it has to be added first to a clean repo to avoid version locked files in the project.

Recommended Next Steps

Based on the current features the following would be logical next steps for development (not in any order):

- Add Animal Data to the local frontend saved game info
- Add Animals to Encyclopedia Screen
- Add dynamic Animal Screens
- Setup clean repo with .gitIgnore file pre-set
- Change game map to an auto tiling system (either time/distance/collision based)
- Change Map API to call backend rather than direct API
- Add Player Authentication Screen
- Add Creature Models