iNaturalist + Unity

iNaturalist + Unity is a third-party tool created by Josh Aaron Miller to provide integration between Unity projects and the iNaturalist API.

Disclaimers

This repository is not endorsed by iNaturalist, the California Academy of Sciences, or National Geographic.

This repository is not sponsored by or affiliated with Unity Technologies or its affiliates. “Unity” is a trademark or registered trademark of Unity Technologies or its affiliates in the U.S. and elsewhere.

Demo

A pre-built demo is available for Windows and Mac.

Getting Started

1. Add this repository to your Unity project by downloading the code and moving it into your project or by adding it directly from the Unity Asset Store (coming soon!)
2. For any scripts that want to interact with iNaturalist, import this code by adding `using JoshAaronMiller.INaturalist;` at the top of the file.
3. Add an `INatManager` component to any GameObject, e.g.: `INatManager myINatManager = gameObject.AddComponent<INatManager>();` or by adding the component in the Unity editor.
4. Use the INatManager to make calls to the API following the documentation, see the examples below.

General Usage Notes

All calls to the `INatManager` require two parameters: a function to callback when the request returns successfully, and a function to callback when the request fails.

Example:

```
 INatManager iNatManager;
ObservationSearch myObservationSearch;
// ...

public void ProcessObservations(Results<Observations> myObservationResults){
// do stuff
}

public void HandleError(Error error){
// do stuff
}

iNatManager.SearchObservations(myObservationSearch, ProcessObservations, HandleError);
```

Authentication

The demo authenticates in a way that is simple to program but cumbersome for the user, i.e., by asking the user to copy/paste in their API token. For a production-ready application, you would want to instead use a web viewer to load the iNaturalist login screen within the client. See iNaturalist's recommended means for authenticating, here.

Free web viewers

For free web viewer packages, I have found these options (this is not an endorsement of any software listed below):

- **Windows**: https://github.com/tunerok/unity_browser
- **Everything else**: https://github.com/gree/unity-webview
- **Android** / **iOS**: https://assetstore.unity.com/packages/tools/utilities/game-package-manager-147711

Common Use Cases and Examples

Fetch sounds

Why: Your game/project wants to have authentic nature sounds, such as bird calls from real birds of a specific species.

Example:

```
 INatManager iNatManager = gameObject.AddComponent<INatManager>();
ObservationSearch myObservationSearch = new ObservationSearch();
myObservationSearch.SetLicense(ObservationSearch.License.Cc0); // fetch only public domain
myObservationSearch.SetBooleanParameter(SearchObject.BooleanParameter.HasSounds, true);

// set other parameters here to narrow your search as preferred
myObservationSearch.SetIconicTaxa(
    new List<ObservationSearch.IconicTaxon>() {
    ObservationSearch.IconicTaxon.Aves }); // fetch only birds

iNatManager.SearchObservations(myObservationSearch, ProcessObservations, HandleError);

public void ProcessObservations(Results<Observations> myObservationResults){
    List<Observation> myObservations = myObservationResults.results;
    foreach (Observation o in myObservations){
        List<Sound> observationSounds = o.sounds;
        foreach (Sound s in observationSounds){
            string attribution = s.attribution;
            string fileUrl = s.file_url;
            string license = s.license_code;
            string audioType = s.file_content_type;
            // do stuff with the sound file, see WebClient.DownloadFile to download
            // or load directly using UnityWebRequestMultimedia.GetAudioClip
        }
    }
}

public void HandleError(Error error){
    Debug.LogError(error.status); // print the HTTP status code
    // other error handling
}
```

Fetch pictures

Why: Your game/project wants to use authentic nature photography for flavorful immersion.

Example:

```
 INatManager iNatManager = gameObject.AddComponent<INatManager>();
ObservationSearch myObservationSearch = new ObservationSearch();
myObservationSearch.SetLicense(ObservationSearch.License.Cc0); // fetch only public domain
myObservationSearch.SetBooleanParameter(SearchObject.BooleanParameter.HasPhotos, true);

// set other parameters here to narrow your search as preferred
myObservationSearch.SetIconicTaxa(
    new List<ObservationSearch.IconicTaxon>() {
    ObservationSearch.IconicTaxon.Plantae }); // fetch only plants

iNatManager.SearchObservations(myObservationSearch, ProcessObservations, HandleError);

public void ProcessObservations(Results<Observations> myObservationResults){
    List<Observation> myObservations = myObservationResults.results;
    foreach (Observation o in myObservations){
        List<string> photoUrls = o.GetPhotoUrls(Observation.ImageSize.Large);
        // do stuff with the photo files, see WebClient.DownloadFile to download
        // or load directly using UnityWebRequestTexture.GetTexture
    }
}

public void HandleError(Error error){
    Debug.LogError(error.status); // print the HTTP status code
    // other error handling
}
```

Get information about local flora/fauna

Why: Your game/project uses a real-life place and wants to have accurate information about the animals and plants that live in that area.

Example:

```
 INatManager iNatManager = gameObject.AddComponent<INatManager>();
ObservationSearch myObservationSearch = new ObservationSearch();

public double myLatitude; // set the geolocation of the place of interest
public double myLongitude; // set the geolocation of the place of interest
public double radius = 100; // in kilometers

myObservationSearch.SetBoundingCircle(myLatitude, myLongitude, radius);
// we could have also searched by bounding box using SetBoundingBox

iNatManager.SearchObservations(myObservationSearch, ProcessObservations, HandleError);

public void ProcessObservations(Results<Observations> myObservationResults){
    List<Observation> myObservations = myObservationResults.results;
    foreach (Observation o in myObservations){
        List<Identification> identifications = o.identifications;
        foreach (Identification ident in identifications){
            if (ident.current) { // only take the latest identification per observation per user
                Taxon taxon = ident.taxon;
                string name = taxon.preferred_common_name;
                // do more stuff with this name
                // for example sum them in a dictionary to get the most popular species
            }
        }
    }
}

public void HandleError(Error error){
    Debug.LogError(error.status); // print the HTTP status code
    // other error handling
}
```

Help classify images or sounds

Why: Your game/project has a real citizen science component where users can submit labels or vote on existing labels.

Example:

```
// NOTE: This operation requires authentication, see Authentication section above

List<Observation> myObservations; // assume we have already gathered observations, see examples above
INatManager iNatManager = gameObject.AddComponent<INatManager>();

Dictionary<string, Taxon> existingGuesses = new Dictionary<string, Taxon>();
foreach (Observation o in myObservations){
    List<Identification> identifications = o.identifications;
    foreach (Identification ident in identifications){
        if (ident.current) { // only take the latest identification per observation per user
            Taxon taxon = ident.taxon;
            string name = taxon.preferred_common_name;
            existingGuesses[name] = taxon;
        }

        // present the user with the list of existingGuesses.Keys, get their guess
        // for more advanced users, you could get them to write in a new guess
        // use INatManager.SearchTaxa to perform an autocomplete search based on their input string

        string userGuessName; // assume they fill this in

        Taxon userGuessTaxon = existingGuesses[userGuessName];

        IdentificationSubmission identSub = new IdentificationSubmission();
        identSub.observation_id = o.id;
        identSub.taxon_id = userGuessTaxon.id;
        iNatManager.CreateIdentification(identSub, CreateIdentificationCallback, HandleError);
    }
}

public void CreateIdentificationCallback(Identification ident)
{
    Debug.Log("Successfully submitted identification " + ident.id);
}

public void HandleError(Error error){
    Debug.LogError(error.status); // print the HTTP status code
    // other error handling
}
```

Documentation

iNaturalist + Unity

The full documentation for this code is available here.

iNaturalist API Documentation

The official iNaturalist API documentation is available here.

Support

For questions or feedback, contact me at miller.josh [at] northeastern [dot] edu