CMSC 425 final project report

SAFARI SNAP

Short description:

This game is a photography-adventure game in which the player, which happens to be a ball with a camera, is tasked to photograph animals in the wild. The player needs to take 5 pictures and then the pictures will be evaluated, with points being awarded based on various parameters of each picture such as orientation of the photographed subject, placement of the subject, and what it was doing when photographed.

Team members: Nicholas Hung

MATERIALS

Unity was used to create the game. I used Blender to create the terrain, tree, elephant, bird and giraffe models, as well as the giraffe and bird textures. I also created the animations of the animals, as well as their rigging in Blender. For the animals' animation, I did review images and watched a couple videos of certain actions like sitting but they were definitely challenging to replicate. Therefore, my animations are not entirely accurate. I also created a 'filmstrip' image using Paint.net to serve as a background for my pictures-taken counter in the top right of the game. The camera crosshair (the white square corners when the camera is activated) was also created with Paint.net.

IMPORTANT GAMEPLAY CONTROLS

- F1 lets you toggle between 3rd and 1st person.
- To activate/deactivate the camera, press the right mouse button (must be in 1st person mode).
- To take a picture, press the left mouse button after the camera has been activated.
- Spacebar lets you jump when in 3rd person mode.
- use the Spacebar to go through the pictures during the picture evaluation phase
- Q and E let you rotate your view in 1st person mode.
- W,A,S,D are used for moving along the z and x axis.
- X will freeze/unfreeze the current position of the player.
- Use the mouse clickwheel to zoom in and out when the camera is activated.

TECHNICAL GAMEPLAY INFORMATION

FIRST PERSON VIEW

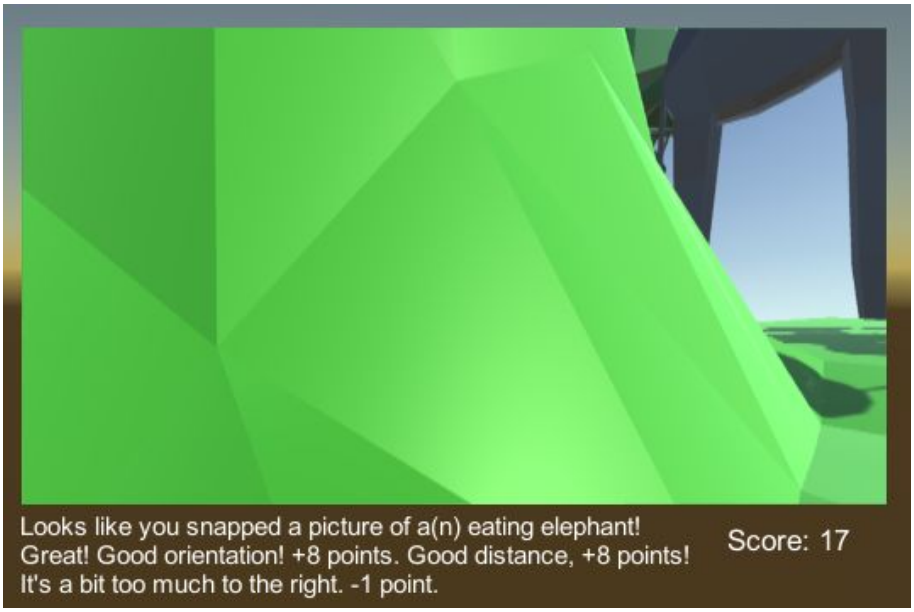
I wanted to have the player be able to use their mouse to look around while in 1st person mode. To do this, when the player is in 1st person mode, the current x and y coordinates based on the mouse cursor's location are added to the current rotation, clamped and the resulting rotation is set to the current one. You can also zoom in and out as you would with a real camera using the mouse click wheel, which changes the camera's field of view value.

GETTING THE TARGET

To get a reference of the target object the player is photographing, I used a single raycast to where the mouse cursor is when clicked.

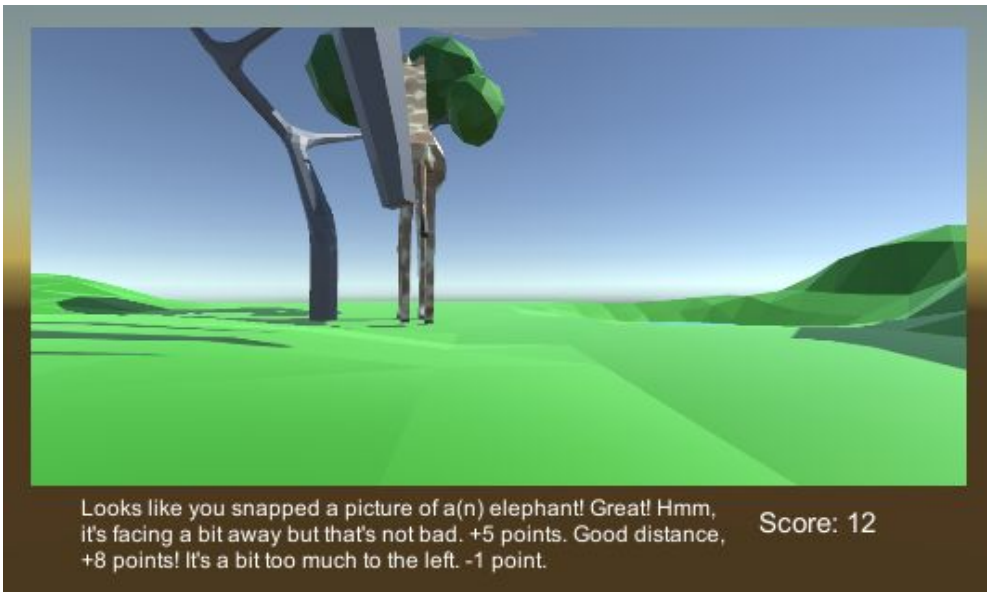
Because I only select one animal via the raycast, it's possible that other animals also show up in the image as well, and the target hit by the raycast may not even appear to be the subject of the picture as there may have been animals closer to the camera that were not hit. This could be an issue that further developments to my game could address.

For example, if possible it might be good to see if multiple objects could be hit by the same raycast, and then compare them. Another possible improvement might be to have additional raycasts in different directions.This could also help detect if an image is consisting of mostly terrain, as seen below.



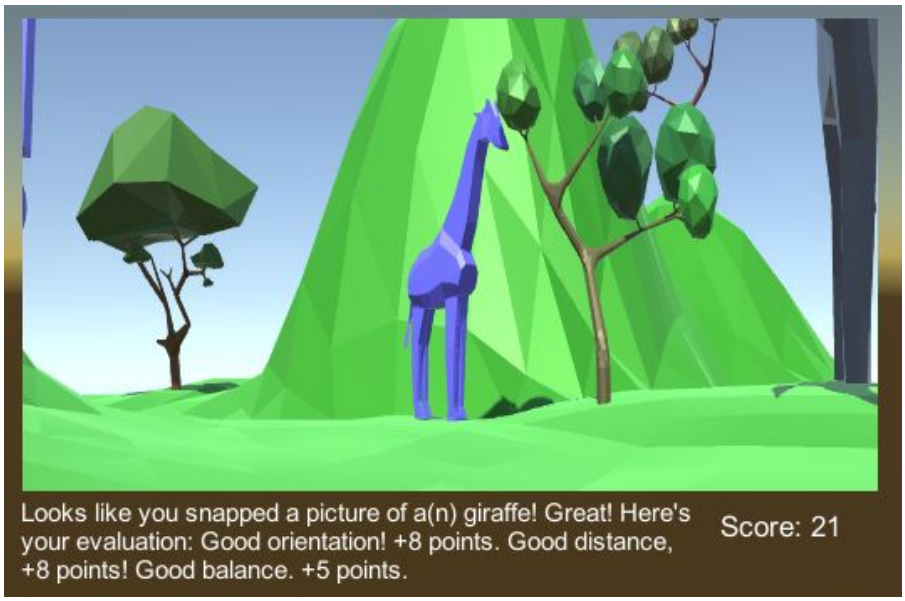
This image really should not have gotten as many points as it did.

Another problem that can occur is when a target hit by the raycast doesn't really show in the image, as seen below. In cases like these, if possible it might be good to see if multiple objects could be hit by the same raycast, and then compare them. In the scenario below, perhaps the giraffe should have been accounted for rather than the elephant.



The elephant was still detected because its trunk was in the path of the raycast, even though I wanted the giraffe.

EVALUATING PICTURES TAKEN

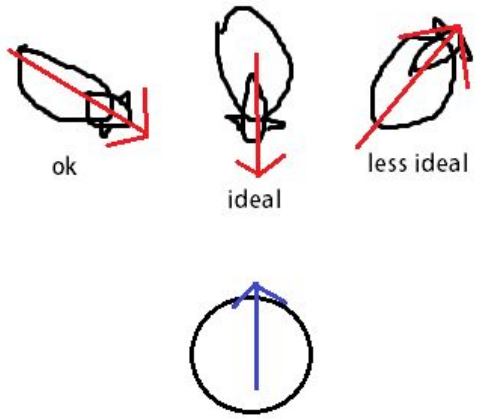


A good image.

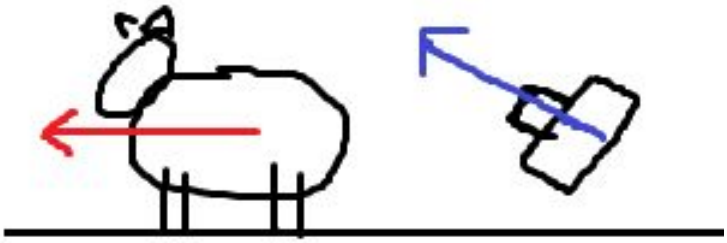
I decided on several parameters to look at when evaluating pictures: target orientation, target distance, target centeredness and the target's activity. These parameter values are collected when a picture is taken (see the CameraScript.cs script). Point distributions for the criteria can be seen in GameManager.cs.

ORIENTATION

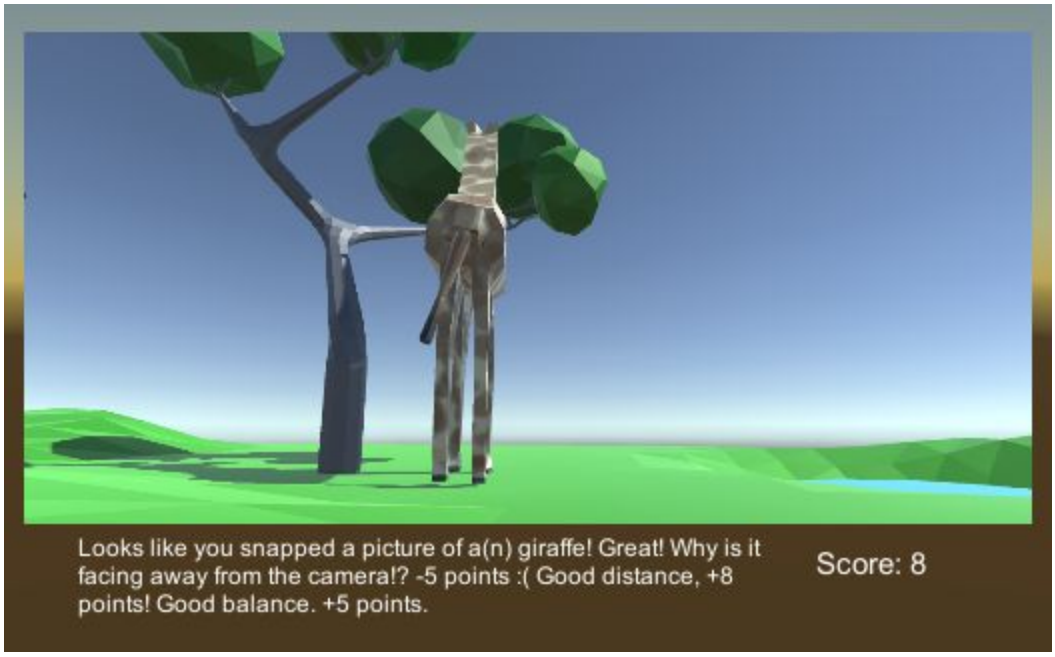
For orientation, I decided that the best orientation would be if the target is facing the camera. The more away from the camera the target is facing, the fewer the points to be awarded to the player. To do this I took the dot product between the camera's forward and the target's forward vectors. If the dot product was positive, then the target was facing away from the camera. (See below image)



Orientation based on forward vectors



One thing to note is that a potentially problematic situation with using the forward vectors can be seen in the above image. If the camera is at a steep enough angle vertically, the dot product may be small enough to no longer count as facing away. My game doesn't account for this because it seems at the current angles I allow for the camera to rotate there isn't a significant impact on the results.



Taking a picture of the rear end is not ideal for my game.

I think I was successful overall in being able to determine orientation this way. There were times when I thought a picture was still satisfactory when the animal was facing slightly away from the camera, so I adjusted my idea to accept small positive dot product values as well.

DISTANCE

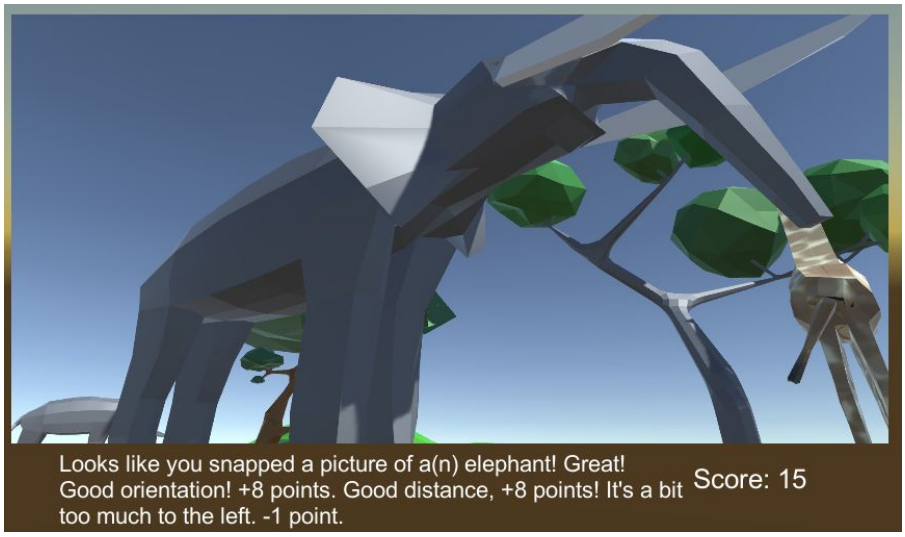
The player in both images is at the same distance away from the bird, but the above photo's camera is zoomed in, changing the field of view amount, which needs to be considered.

For distance, I took the magnitude of the vector resulting from the difference of the target's and the camera's positions. Then I took into account the camera's current field of view value by multiplying the distance by the current field of view value divided by the default field of view value. I multiply the distance by a fraction of the default field of view value because as the field of view value gets smaller when the camera zooms in, the perceived distance between the camera and the target also gets smaller.

CENTEREDNESS

For centeredness, I wanted to look at the target's x and y values relative to the center of the viewport. Unity conveniently has a function called WorldToViewportPoint that can be used to convert an object's position to viewport coordinates which range from 0 to 1.

However, there are cases where this does not provide satisfactory results. For example, in the image below, the player is pretty close to the elephant and looks well-centered (to me, at least), but it was recorded as being a bit far to the left.



An example of what I think is a fairly ‘centered’ image but my program does not.

I think an improved technique might be to include a factor that can reduce the impact of distance from center of the viewport depending on the player’s distance away from the target in my calculations (similar to what I did for taking into account the camera’s zoomed-in amount for distance). This way the closer you are to the target, the more ‘centered’ it will be.

Another way might be to make a bounding box around the target, take a coordinate from each end of the box, and check their converted viewport x coordinates, which range from 0 to 1. If the two points form a line that occupies a majority of the distance from 0 to 1, then we can perhaps say that the target fills most of the x-axis at least and so it’s mostly centered along the x-axis. The same test can be performed for the y-axis.

ACTIVITY OF TARGET

Tracking the target's current activity was fairly straightforward. By having a controller script for each animal, having an enum for the states, and a getter to give me a particular animal's current state, I was able to get the needed activity information through just method calls.

CUSTOM INTERFACE AND CLASSES

I created an Animal interface that contains an animal behavior enum that defines some basic, shared animal behaviors like idle, eating, sitting, etc, as well as a getState method that returns an animal’s current behavior. All my specific animal controllers (i.e. GiraffeController) implement the Animal interface.

Additionally, I made a class called Snapshot, which stores all the data of each picture taken. These Snapshots objects are stored in a list in an instance of GameManager, which

persists through the whole game and doesn't get destroyed when a new scene is loaded. This class also determines, based on the values of the data, whether the subject of an image is centered, too much to the left or right, or too far or too close and assigns the corresponding enum value as instance variables that can be accessed.

I also made a class called ScoreCard, which stores the strings of the comments for the evaluation step, like "Good distance, +8 points!". I do this to make the code less messy and to be able to generate the complete evaluation in a separate function (see GameManager.cs).

MODEL CREATION WITH BLENDER

To learn the basics of modeling animals, trees, rigging and animation, I mainly relied on the tutorials created by Grant Abbitt and Sebastian Lague.

Grant Abbitt: https://www.youtube.com/channel/UCZFUrFoqvqlN8seaAeEwjlw
Sebastian Lague: https://www.youtube.com/channel/UCmtyQOKKmrMVaKuRXz02jbQ

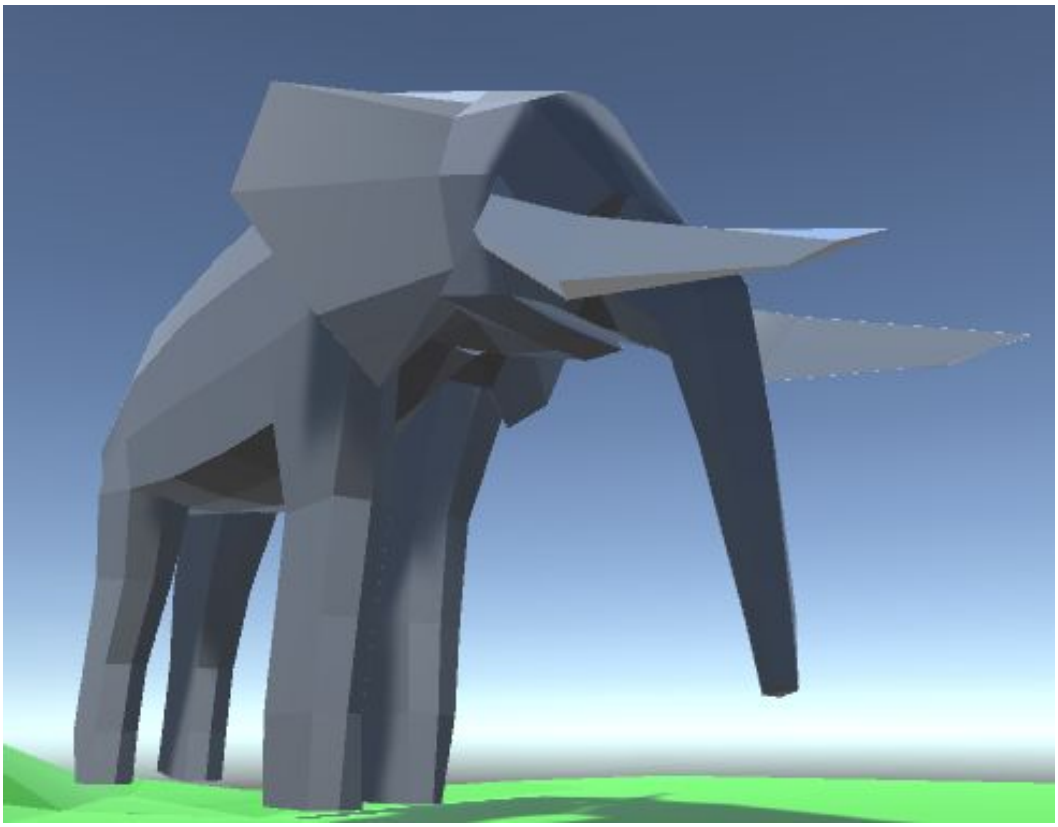
For creating the terrain, this tutorial by Shahan Akhter was very helpful:
https://www.youtube.com/watch?v=FACzCHqzzTg

I made 3 animal models: elephant, giraffe, and bird. I have idle and eating as the common animations between the three animals, sitting and getting up for the elephant and giraffe, and angry as a special trigger animation for the elephant.

Unfortunately I could not get walking animations to work with my generic rigs when applying root motion so I decided to scrap my humanoid-like character as the player and not go through with creating walk animations for my animal models. The problem was that every time my model would complete their walk and move forward, they would go back to where they started. I believe this was because their mesh collider did not move with their armature, but I was unable to figure out a way to move the collider with the armature.

ELEPHANT

- I used this image as a reference (https://www.google.com/search?biw=1920&bih=969&tbm=isch&sa=1&ei=3o8UXNfbKOaygge_sYCQCA&q=elephant+side+view++public+domain&oq=elephant+side+view++public+domain&gs_l=img.3...3772.5030..5184...0.0..0.68.439.10.....1....1..gws-wiz-img.3etQagFsOks#imgrc=YE_6G7wdBPcjvM:) to create my elephant mesh.
- If you get close to the elephant, it should trigger an 'angry' animation.
- The elephant does not have a texture but was colored within Blender by assigning colors to the faces of the mesh.

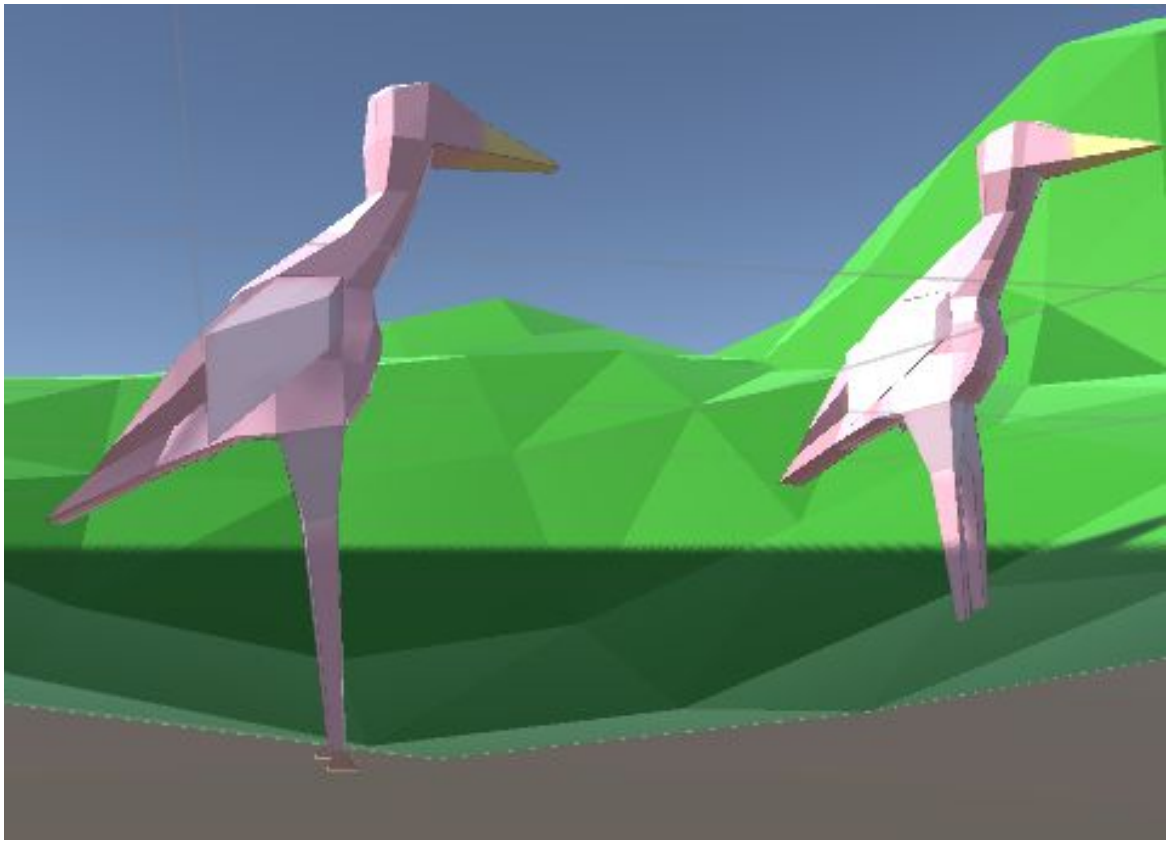


GIRAFFE

- I used this image as a reference (https://www.google.com/search?biw=1920&bih=920&tbm=isch&sa=1&ei=WET_W7-jNO_I_QbNtp3gAw&q=giraffe+side+view+public+domain&oq=giraffe+side+view+public+domain&gs_l=img.3...40883.41758..42141...0.0..0.46.269.7.....0....1..gws-wiz-img.Ncwt8vzAmsY#imgrc=dBe14w4zOg30yM) to create my giraffe mesh.
- I also created a texture for the giraffe in Blender.

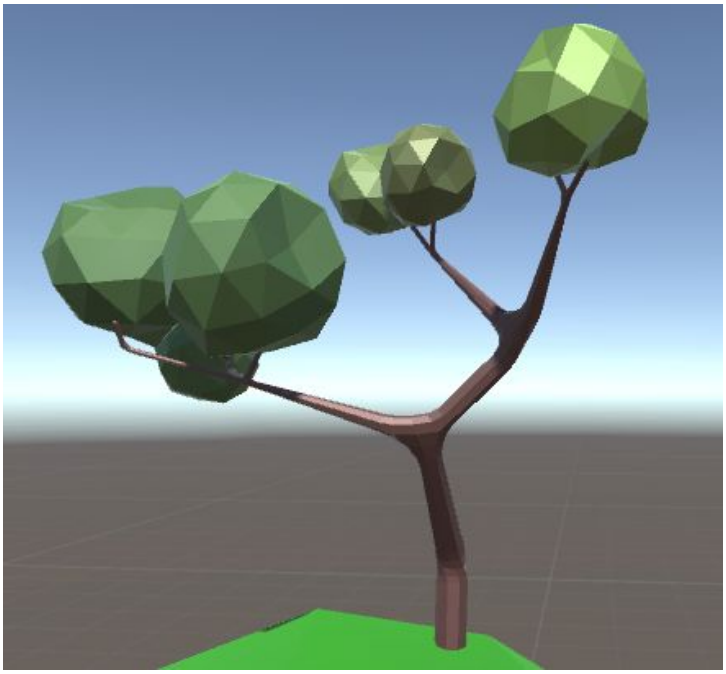
BIRD

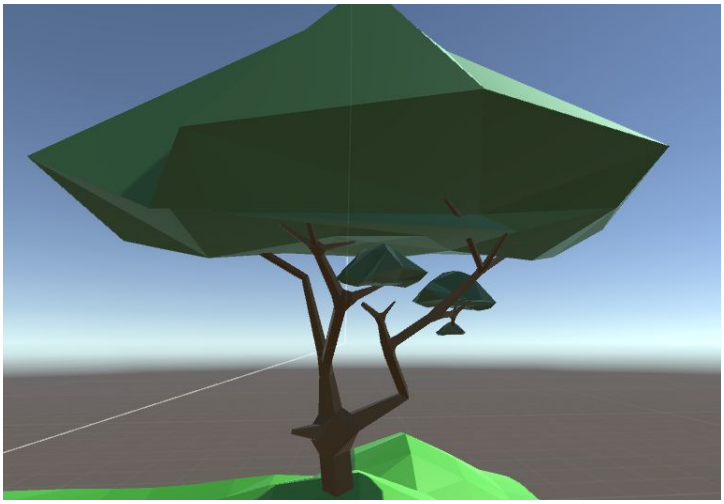
- I used this image as a reference (https://www.google.com/search?q=stork+side+view&source=lnms&tbm=isch&sa=X&ved=0ahUKEwiViJjBvvjeAhWRneAKHcL_CxkQ_AUIDigB&biw=1920&bih=920#imgrc=V-EusU6KJwoR0M) to make my bird mesh.
- I created a texture for this bird in Blender.



TREES

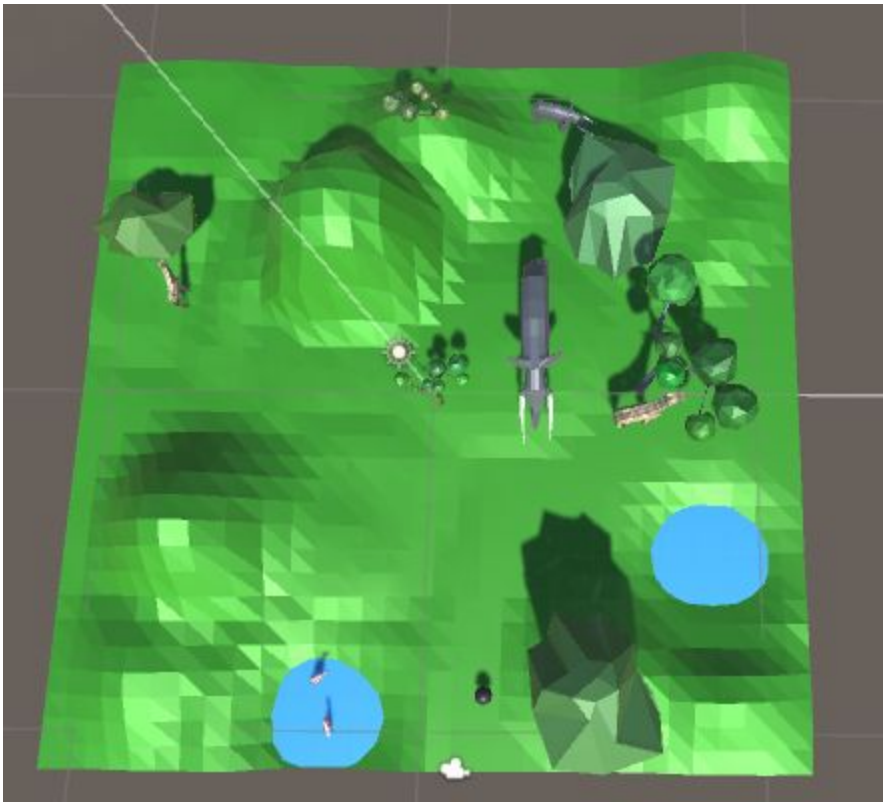
- There are 2 kinds of trees in my game.





TERRAIN

- Constructed from a flat plane
- The ‘water’ sections are just blue-colored planes from Unity used to cover up some depressions in my imported terrain model



SOURCES OF DIFFICULTY / THINGS I LEARNED

Working with imported models and their animations was difficult for reasons stated before. Additionally, models imported from Blender are confusing because there's a plane and armature in the hierarchy and it took some time to figure out where to attach colliders so they wouldn't fall through the terrain. Getting the models' forward vector was also a bit annoying because the way Blender's coordinate system is and how I made my models according to it affects where the forward will point in Unity. I realized this after I took some pictures of my elephant model with its head facing directly towards the camera and having my evaluation complain that it was facing away.

However, learning how to create, rig and animate models in Blender was fun and became more enjoyable the more I practiced doing it. Making the animations I felt is surprisingly simple in Blender, with the help of inverse kinematics.

I also feel that after creating this game I have a good understanding of how I can apply geometry to help solve practical problems, as well as a much better appreciation for the mathematics used in designing games and for engines like Unity that make it very easy to do things like convert world coordinates to viewport coordinates without having to do the matrix multiplication myself.

KNOWN ISSUES

Pressing the ‘X’ key sometimes doesn’t immediately freeze the player and may require several tries. The camera movement sometimes might not be smooth when using the ‘Q’ or ‘E’ key to rotate.

SOURCES I USED FOR HELP

For assistance with getting a camera scope or UI working, I found SpeedTutor’s tutorial helpful: https://www.youtube.com/watch?v=BUL1qDo5NYk

I referred to these links when exploring how to move the camera with my mouse to look around in first person mode and for zooming in. For the mouse look-around, I specifically utilized the code snippet supplied by AndyP-123 with very little modification (I just adjusted some clamp values and added a clamp for the y rotation) found in the first link.

https://answers.unity.com/questions/29741/mouse-look-script.html
https://answers.unity.com/questions/245412/fps-mousescript.html
https://stackoverflow.com/questions/8465323/unity-fps-rotation-camera
https://answers.unity.com/questions/25118/how-do-you-zoom-with-a-sniper.html
https://answers.unity.com/questions/218347/how-do-i-make-the-camera-zoom-in-and-out-with-the.html

The below helped with capturing a snapshot.
https://answers.unity.com/questions/649079/how-to-get-a-screenshot-in-game.html
https://answers.unity.com/questions/850451/capturescreenshot-without-ui.html
https://answers.unity.com/questions/649079/how-to-get-a-screenshot-in-game.html
https://answers.unity.com/questions/1173902/how-to-make-a-ui-image-appeardisappear.html
https://stackoverflow.com/questions/46595055/readpixels-was-called-to-read-pixels-from-system-frame-buffer-while-not-inside

These helped a bit with collecting some of the important criteria data for evaluating pictures.
https://answers.unity.com/questions/720447/if-game-object-is-in-cameras-field-of-view.html
https://answers.unity.com/questions/57978/calculate-the-size-of-a-gameobject-on-screen.html