Nelson Engelschenschilt

Exploratory study of gathering and analyzing eye tracking data

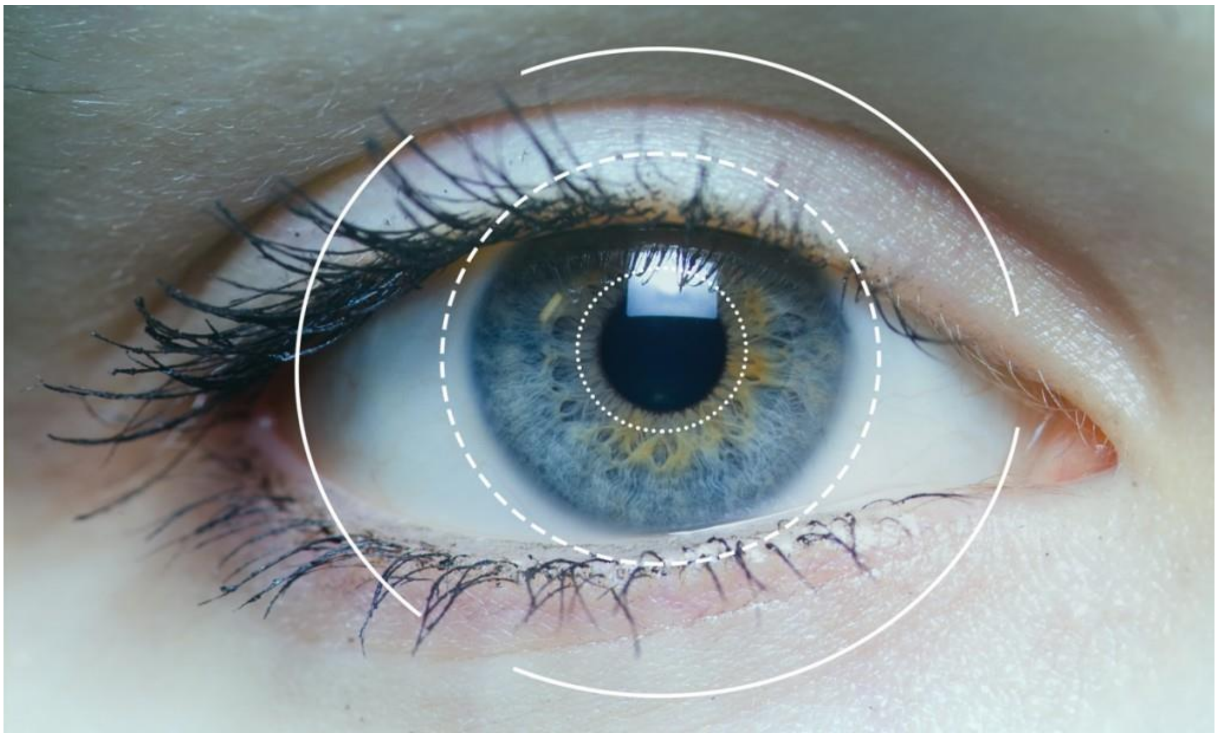


Table of Contents

Introduction

In this project I will work on gathering data reflecting the visual properties and ability of objects and items to catch the player's attention. I will explain using an example. You're working on a post-apocalyptic zombie survival game (think DayZ, Miscreated, Rust, etc.) where the gameplay mainly revolves around exploring abandoned cities, villages, shopping malls, gas stations, garages, industrial areas and other human-made facilities.

When creating such a game it is important that the player, upon entering a building, has something to catch his attention. You want to make sure the key items the player needs to collect, such as food and drinks are not too hard to find even though items like this aren't very big in size.
A chocolate bar, for example, is a very common item in games like this and even though the item itself is a very small object with a natural color, namely brown or black, that barely catches the player's attention unless it can be found on a light colored surface is always easy to find because the artists use clever tricks to make the item stand out.

The most common trick to be used for this particular object is making the wrap of the chocolate bar a certain color that is otherwise not used very often in the game and contrasts with the majority of the games environment. Comparable to the reason we use green for green screens or hot pink as the background color for sprite sheets, namely because they are colors which are very rarely used elsewhere in the environment or sprites, can easily be used as a selection mask and catch your immediate attention warning you about the possible misalignment of your sprites, we can color the wrapping paper of the chocolate bar red to contrast with the gloomy, often dark green, moss overgrown color palette of the scenery of a survival game.

Guiding or manipulating the players visual attention to purposely or unwillingly force them into making a certain decision can be an interesting and powerful concept. Let me use horror games as another example. If you wish to add a jump scare somewhere along the line of creating your game you can use an object or detail in the scenery colored in such a way that it draws the player's attention to catch him or her off guard.

There are more ways than making your objects glow or giving them an outline that can make them pop out like contrast, intensity and luminance and the ability to generate statistics about objects and their attention drawing capabilities by simply letting the players play the game like they otherwise would seems like an interesting tool to gather this valuable data.

Goal of the project

The aim of this project is to create the necessary tools to track and analyze data which can be gathered using eye tracking hardware & software. In this case I will be using the Tobii eye tracking device in combination with the Unity game engine.

Some examples of the data I aim to acquire are:

- How long has every object been onscreen?
- How long has the player focused on every object?
- How big is the object focused on relative to the screen size?
- How long is the distance between the player and every onscreen object?
- How long is the distance between the player and every focused object?
- Which objects did the player's eye gaze cross but not focus on?

Based on the data acquired a list will be generated where the first item is the object which caught the most attention. This result will be deduced from multiple factors such as how long an item had been focused on compared to how long it has been onscreen. This is important because otherwise an object can be focused on for 5 seconds while it has been onscreen for over a minute while other objects scored lower on focus time but were onscreen for a much shorter period of time.

Size and distance are also important because standing in front of an object that fills your entire screen will result in you focusing on the object for extended periods of time even though you don't have much choice in such a situation. So in order to avoid biased data these factors need to be considered when calculating likeliness of focusing an object.

Ultimately this data can be displayed in graphs to give a visual representation of the data acquired. It is impossible to avoid biased data completely so these graphs may shed light on questions that may arise when in doubt of the accuracy of the results.

You can take this even further by passing the results to a saliency map generator which can calculate the saliency of all objects in a screen capture or recording using Laurent Itti and Christof Koch's model for visual attention. However, although very interesting, I do not plan on implementing this feature in my project due to time limitations.

Assignment for test subjects

Every test subject will receive the task of finding a very specific object, for example a green Dutch-to-English dictionary. In order to avoid biased data by showing a preview of the object the player needs to find and thus causing the player to only search for objects that look the same as the preview object I will only describe the object. This will cause the test subject to be more likely to look at other objects more carefully and closely inspect them rather than quickly scanning a room and moving on. This way objects that catch attention will be looked at longer than when the player can identify the target object in the blink of an eye due to having seen it before.

The test will take place in a very modern, realistic, PBR rendered apartment consisting of 4 to 5 unique rooms and a large detailed hallway. The rooms are filled with artworks, decorations, furniture and books thus giving the player lots of objects to catch their attention.

The test subjects will receive a time limit for completing their objective in order to prevent them from walking around aimlessly, checking out the environment as long as they wish, until they happen to stumble upon their target object. Doing this also reduces the amount of variables in the test in order to achieve a more accurate result.

Not setting a time limit could, again, cause bias because the players can spend more time in a room that seems interesting to them instead of actively searching. For example, one of the rooms in the apartment has two mirrors, both with quotes written onto them. A player with no time pressure might slack off and spend time reading the quotes and thus cause these mirrors to rise in popularity and increase the 'time focused' statistic for this object.

Tobii EyeX Controller

The Tobii EyeX Controller is an eye tracking device that uses near-infrared light to track the eye movements and gaze point of a user by using cameras, illuminators and algorithms. It works as follows:

- The illuminators create a pattern of near-infrared light on the eyes which partially reflect back onto the Tobii depending on the angle of the eyes.
- The cameras take high-resolution images of the user's eyes and the patterns projected onto them.
- The image processing algorithms find specific details in the user's eyes and reflection patterns.

Based on the information gathered the eyes' position and gaze point can be calculated using sophisticated 3D eye model and gaze mapping algorithms.

Tobii EyeX Engine

The Tobii EyeX Engine is a piece of software that works similar to an operating system extension for eye tracking. It consists of the logic which allows communication between the eye tracker and the operating system and can be used to configure the tracker, combine the user's eye gaze and other input, interprets them as user interactions and mediates between multiple applications that are using eye tracking simultaneously.

Tobii EyeX Interaction, built on top of the EyeX Engine, offers basic eye gaze interactions available in the windows environment but what's really interesting is the EyeX SDK (Software Development Kit). It provides the user with access to the EyeX Engine API; a series of building blocks for making eye tracking applications consisting of code samples, demo scenes, dlls, documentation and code for integration into game engines and GUI frameworks. The SDK can be divided into three categories:

Streams provide filtered eye gaze data from the eye tracker such as:

- Gaze points - points where your eyes are looking on the screen.
- Fixations - points on the screen where your eyes linger to focus on something.
- Eye positions in millimeters relative to the center of the screen.

States provide information about the state of the EyeX system:

- Is there is a user behind the screen?
- Is the eye tracker currently able to track the user's eye gaze?
- Information about the system setup; size of display in millimeters, calibration profile, eye tracker availability, …

Behaviors are higher level interaction concepts comparable to mouse interactions such as clicking and scrolling that use regions for interaction with GUI components.

Tobii EyeX Engine API

The Tobii EyeX Engine API, abbreviation for application programming interface, is a set of subroutine definitions, protocols and tools for building eye tracking supporting applications.

Some of the major features offered by the EyeX Engine API are:

- **Eye-gaze point:** Calculates the point on the screen where the user's eyes are currently looking. Since the human eye is very active and in constant motion the raw data results of the eye-gaze point stream becomes a spread of points around the location the player is looking at rather than one single point.

- **Eye position:** The position of the user's eyes relative to the center of the screen. This value represents this distance in millimeters.

- **Fixations:** The points on the screen where your eyes linger to focus on something. These fixations can be found by looking for areas with an increased concentration of gaze points.

- **Gaze-aware region:** This defines a region on the screen that detects when the user's eye-gaze enters and leaves the region and can be used as a 2D event trigger area.

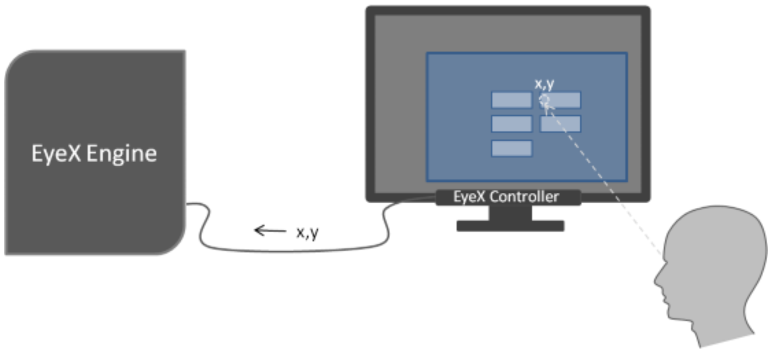
- **Activatable region:** This region works very similar to the gaze-aware region with the major difference being the ability to 'activate' the region by clicking a previously specified button. The region can also be activated by focusing the eye-gaze onto the region and triggering a fixation.

- **Pannable region:** It is possible to define regions that trigger the application to pan or scroll in the specified direction.

- **User presence:** The user's presence can be detected which, for example, allows automatic switching between manual (keyboard & mouse) and eye tracking controls.

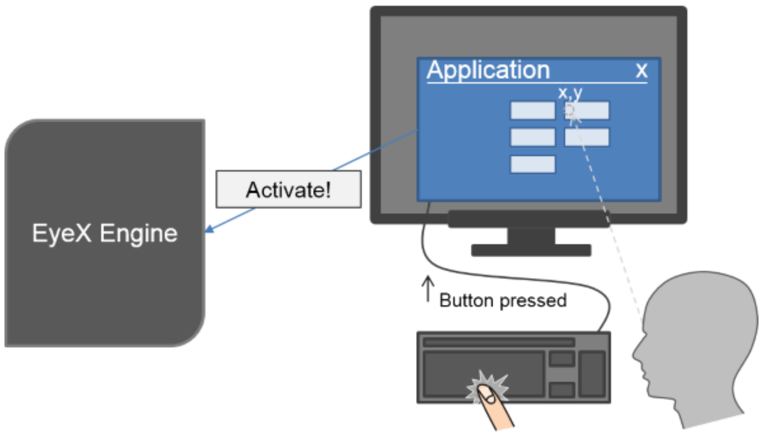
How does the EyeX Engine know what the user is looking at?[1]

1. The user looks at the screen. The EyeX Controller calculates the coordinates of the user's gaze point. The EyeX Engine receives the normalized screen coordinates and transforms them to pixel coordinates on the screen.



2. The EyeX Engine uses information from the client application to understand what kind of visual object the user is looking at and might want to interact with. Let's say the rectangles on the screen are buttons with the 'Activatable' behavior. That means that they can be clicked using eye gaze. The EyeX Engine now decides which button the user is looking at and then expects the user to indicate that he or she wants to click the button.

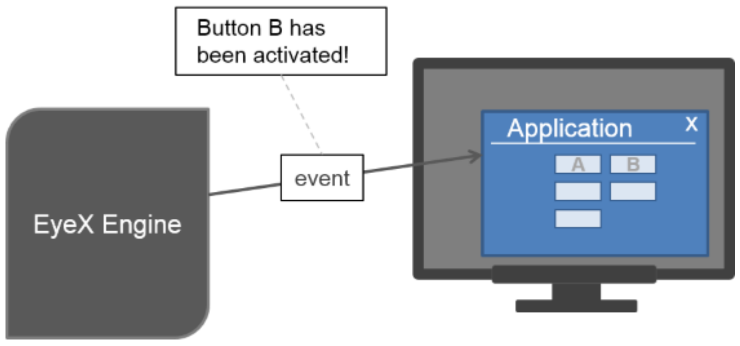
3. The client application might offer the user some different ways of actually clicking the button using eye-gaze, but it will always be some combination of looking at the thing you want to click, and then giving some secondary input to trigger the click. Let's say that in this client application the user can trigger a click on the object they are looking at by pressing the space bar on the keyboard.



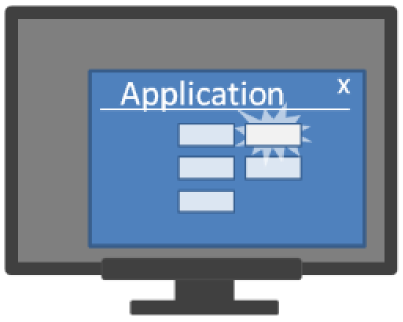
4. The user presses the space bar while looking at the button he or she wants to click. The client application informs the EyeX Engine that the user has requested to 'Activate' the activatable button currently looked at.

[1] Example taken from: http://developer.tobii.com/an-introduction-to-the-tobii-eyex-sdk/

5. The EyeX Engine generates an event that informs the client application which button the user wants to activate.



6. The client application responds to the event and for example gives the user a visual feedback that the button on the screen is clicked, and then performs the action associated with clicking that button.



Implementation of the test in the Unity game engine

The test subject will be able to control a simple character that is capable of walking around inside the apartment. He can look around in first person mode without restrictions but is incapable of jumping or performing any other action. This behavior will be handled by a simple character controller script consisting of nothing but movement logic and will be applied on a game object with collision detection handled by the character controller component.

The area the player can explore is a modern apartment, rendered using physically based rendering for increased realism and detail. It contains a living room, two large bedrooms, a kitchen/dining room, a small office room and a bathroom. Some rooms are very bright because they have windows leading to the outdoors while others are rather dark because they lack such windows or they are too small to allow a lot of light to come in. I'll have to make sure I take this into consideration when collecting data because dark colored objects would have a low saliency score in poorly lit rooms while light colored objects would score very high due to contrast.

This phenomenon is not an object specific property since it is the result of the combination between the objects base color, its ability to reflect light and thus creating highlights which often result in a high contrast difference, and the environment. Saving the amount of light exposure on every object would make for an interesting statistic; it can be used as a point of reference. For example, when working on a horror game it would be more interesting to analyze data for poorly lit objects rather than well-lit ones so you can make sure the props you want to stand out do so in the correct environments.

To log all the objects that the player's eye-gaze crossed I will use an object detector script that calculates the gaze point, performs a raycast towards the gaze point and the first object hit by the raycast will be saved to the list of items the player looked at. Following code snippet represents the Update function of the object detection script:

```
public void Update(Vector2 eyePoint)
        {
            if (!_bDetecting)
                return;

            var gazePoint = eyePoint;
            gazePoint.y = Screen.height - gazePoint.y;
            Debug.Log("Gazepoint: " + gazePoint);

            var gazePointPrep = new Vector3(gazePoint.x, gazePoint.y, -
Camera.main.transform.position.z);
            var gazePoint3D = Camera.main.ScreenToWorldPoint(gazePointPrep);

            Ray ray = Camera.main.ScreenPointToRay(gazePoint);
            RaycastHit hit;
            GameObject lastObject = _currentObject;
if (Physics.Raycast(ray, out hit, 100))
            {
                _currentObject = hit.transform.gameObject;
                var oName = hit.transform.gameObject.name;
                var matToggle =
hit.transform.gameObject.GetComponent<MaterialToggle>();

                if(matToggle)
                {
                    matToggle.Gaze();
                }
            }
            else
            {
                _currentObject = null;
            }

            if(lastObject != _currentObject)
            {
                //Create a detection
                double timestamp = PerceptorHandler.GetTime();
                var objectName = _currentObject == null ? null : _currentObject.name;
                ObjectDetection detection = new ObjectDetection(timestamp,
objectName);
                _objectDetections.Add(detection);
                Debug.Log("Object name: " + objectName);
            }
        }
```

In order to be able to add more data to the object I will make a class or struct that can save variables such as:

- The timestamp.
- The player's position at the moment of perception.
- The position of the detected object.
- The screen coverage of the detected object.
- The light exposure on the object.
- The duration for which the object has been fixated by the player.
- The duration for which the object has been onscreen.

The class will look like the following code snippet:

```
    public class ObservedObject
    {
        GameObject GameObject { get; private set; }
        DateTime Timestamp { get; private set; }
        Vector2 PlayerPosition { get; private set; }
        Vector2 ObjectPosition { get; private set; }
        float ObjectScreenCoverage { get; private set; }
        float ObjectLightExposure { get; private set; }
        float FixationDuration { get; private set; }
        float OnscreenDuration { get; private set; }
    }
```

Saving the positions is important for calculating distance between the player and the objects while saving the duration is necessary to check how long an object has been observed compared to the time the object spent onscreen.

Something more complicated to do is calculating how much of the screen an object covers. There are multiple ways to do this, the first one is less accurate but more performant and uses the bounds property of the renderer component of the object to acquire the center and extents of the mesh. These are used to calculate the bounding box of the object which can then be used to calculate the screen percentage it covers. However the scene we are using consists of a lot of concave shapes. Calculating the bounding box from, for example, a coat rack could result in a bounding box that covers much more volume than the actual object. For this reason it might be more interesting to use the second option. Although it is slower, it is much more accurate than the fast approximation method explained earlier.

For the accurate method we render the object to a texture and use a background color that can easily be distinguished from the object itself such as hot pink or pure green. Once we have the texture we can count the number of pixels that are not background pixels and divide our result by the total amount of pixels onscreen, depending on the screen resolution. This will give us the exact percentage of screen coverage by the object and completely avoid the approximation issues that occur when using the bounding box method.

Conclusion

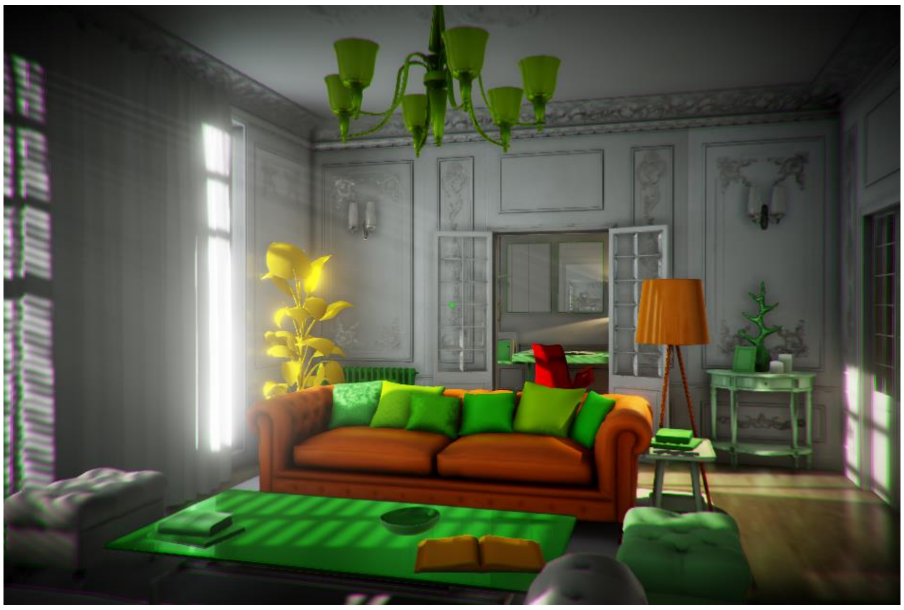
Using the Tobii EyeX SDK and the Unity game engine I will be able to extract and analyze eye tracking data in order to check if every object catches the amount of attention I want it to and if necessary alter its appearance to be more appealing to the eye. This can be used in games to make sure important key items get noticed while hidden items or secrets remain somewhat unnoticed and require more observation to figure out.

Implementation

In order to visually represent the eye tracking data, so it can be used by a level designer, I created components for Unity game objects. These components are attached to the props in the scene in order to register the required data, make the desired calculations and manage their game objects so that the user can observe the gathered date in a visual manner.

Heatmap

The heatmap component can be attached to a game object to reflect the duration it has been looked at by changing the object's color based on a color gradient. By default the color gradient goes from white to green to yellow to orange to red so the more the object has been looked at the more it turns red.



When the player looks at an object with the heatmap component, it adds the duration for which the player looks at it to the gaze duration counter.

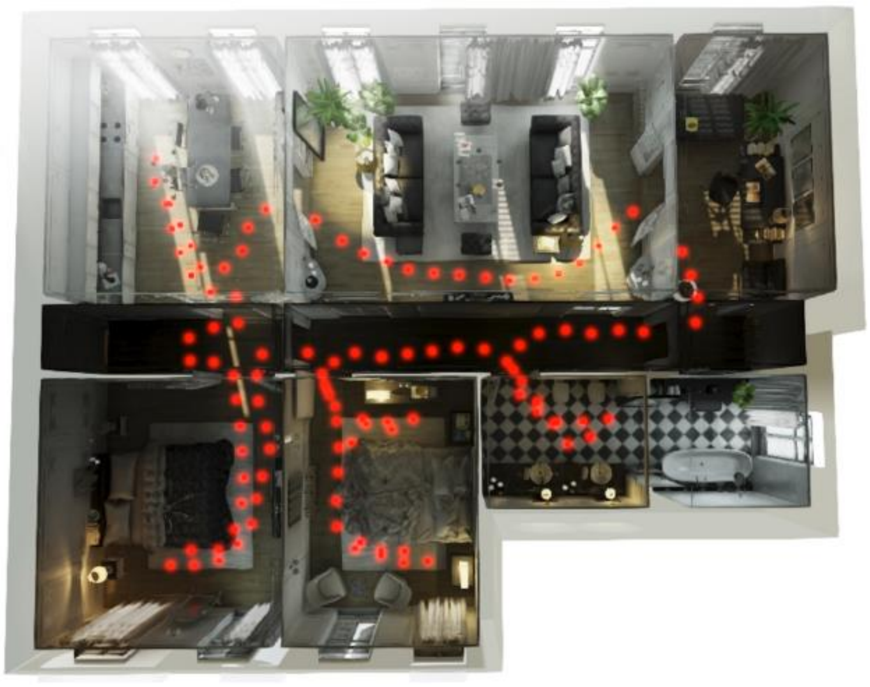
The heatmap manager stores a list of all heatmap objects so we can calculate the objects with the highest and lowest gaze duration. These duration values are then used to interpolate and results in a float which is used to get the correct color from the heatmap color gradient.

For example: An object in the scene(object A) has been looked at for 16 seconds and the object with the highest gaze duration value (object B) has been looked at for 32 seconds. Object A's color will be at 50% of the color gradient, in this case yellow, while object B will be red.

It is possible to toggle this heatmap color by pressing the 'H' key on the keyboard and the component can be attached to a group of game objects using the heatmap attacher script which adds the heatmap component to all children of an object that have a Mesh Render component.

Markers

Markers indicate the player's trajectory when exploring a level. It is comparable to the iconic strategy from the tale of Hansel and Gretel where breadcrumbs are dropped to indicate the trajectory but instead I spawn a prefab (cube by default).



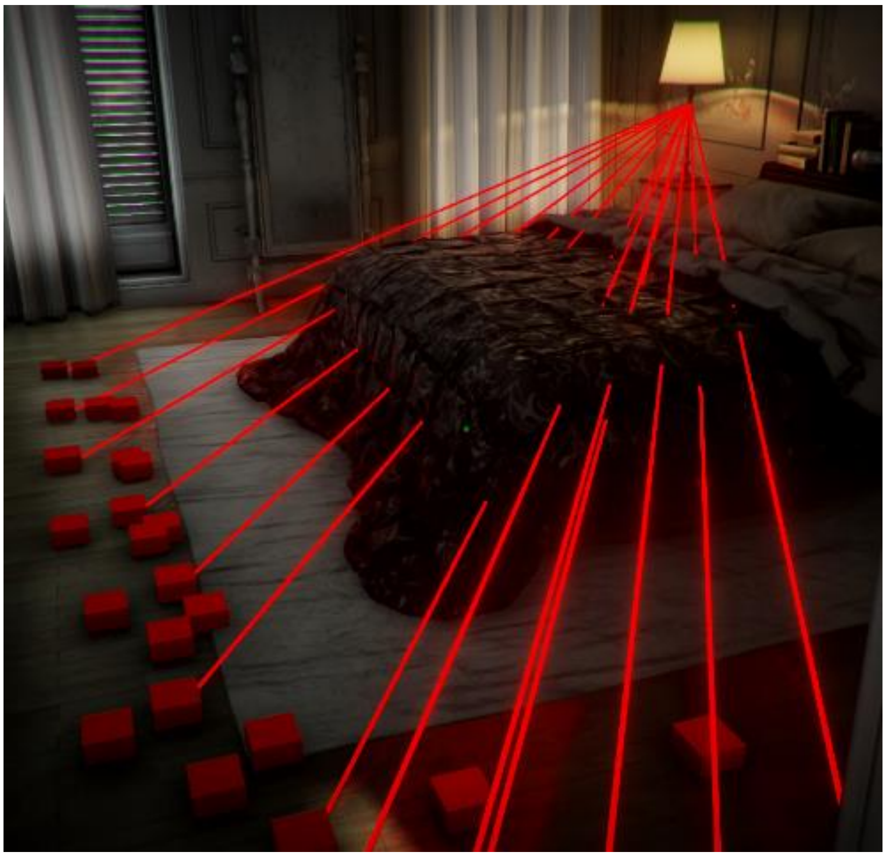
The marker data component can be attached to a game object to save a list of all markers from which the object has been looked at. This list is then used by the line manager to draw lines between the object and all the markers it has been looked at from. An attacher also exists for this component similar to the one used to attach heatmap components.

The marker manager spawns a marker every X distance from the previous marker and can toggle the visibility of all the markers by pressing the 'M' key.

Line Manager

The line manager is used to draw 2D, camera facing, lines between a game object and the markers it has been looked at from. The game object used is determined by the selection manager.

The visibility of the lines can be toggled by pressing the 'L' key.



Selection Manager

The player can click an object to see lines towards the markers it has been linked to by the marker data component. To check which object has been clicked I perform a raycast from the camera's position towards the center of the screen.

The line manager then draws the lines from that game object to its markers.

Serialization

When the player closes the game a function is called to write all the detected objects and its data to a file using serialization. Before the data is written to a file we pass it through some filters allowing us to remove objects that barely got any attention. For example when the player looks at an object on the left side of the screen and then suddenly notices an object on right side, his/her eye gaze will cross multiple other object before reaching its destination. By filtering out all objects with less than 0.15 gaze duration we get rid of this issue.

Conclusion

During the creation of this graduation work I encountered some problems; one of those problems was finding a clear goal to work towards. It took some time before my plan of action came to be but when I eventually managed to setup clear goals for myself everything went a lot smoother. Working on the project step by step rather than trying to do many things at once made sure my code didn't turn out like the mess I feared it would become if I didn't make a planning.

I learned a lot of things about the way eye tracking works and came up with a lot of creative ways to utilize it but since time is limited I could not realize all these ideas. That said I did manage to achieve what I planned to by creating visual feedback tools for a level designer to analyze his level and the way it is played by others.

In the future I can image technology like this to become very popular because of the unique way of gathering data and could even be used as a way to (partially) control your character in games or even your computer system in general.

Bibliography

- Itti Laurent, (http://ilab.usc.edu/borji/papers/IEEESMCCamReady.pdf)
- Ernst Niebur, (http://www.scholarpedia.org/article/Saliency_map)
- University of Notre Dame, (https://www.ncbi.nlm.nih.gov/pubmed/23801969)
- SaliencyToolbox, (http://www.saliencytoolbox.net/)
- Anders Tobii, (http://developer.tobii.com/tobii-gaze-sdk-4-0-2/)
- Jenny Tobii, (http://developer.tobii.com/what-is-eye-tracking/)
- Anders Tobii, (http://developer.tobii.com/eyex-engine-api-glance/)
- Jenny [Tobii], (http://developer.tobii.com/an-introduction-to-the-tobii-eyex-sdk/)
- TobiiPro, (http://www.tobiipro.com/learn-and-support/learn/eye-tracking-essentials/how-do-tobii-eye-trackers-work/
- Robert bu, (http://answers.unity3d.com/questions/691066/how-to-calculate-screen-area-coverage-of-a-3d-obje.html)