Interaction Multimodale et Affective Rapport

A. Freeman, F. Hategekimana

June 2023

1 Research

1.1 Introduction

The multimodal and affective interaction course prepares students for the creation of man-machine interfaces capable of using several modalities and adapting to the user and the context of use. The aim of this course is to bring emotional and sensory modes of interaction to the table.

This report presents all the work carried out on a prototype interactive system in the form of a game. This report respects the advanced modes of interaction seen in the course. This project also aims to answer a research question of our choice. Here we have two of them, because we felt they were both important and closely linked. This allows us to test the differences between two modalities that seem to function in a similar way. The results are interesting and open the door to other developments. For this project, we had 3 main procedures to put in place:

- Eye tracker
- Sound
- Arousal

1.2 Literature Review

1.2.1 Nicholas Graham. Use of Eye Movements for Video Game Control

For this project, we searched for articles related to our modality using relevant keywords such as "eye tracker", "sound", "modality" and "human-computer interaction". From the pages identified, we selected the article most directly related to the use of eye trackers in video games.

This is where we came across Nicholas Graham's article entitled "Use of Eye Movements for Video Game Control" as the main guide for our research into the use of eye tracking in video game control. This is a 2006 reference that detailed the use of eye-tracking technology in games such as Quake, Neverwinter Nights and Lunar Command. The aim of this study was to determine the effectiveness of eye trackers for video games.

Figure 7: In *Neverwinter Nights*, players communicated with their avatar through eye-based pointing.

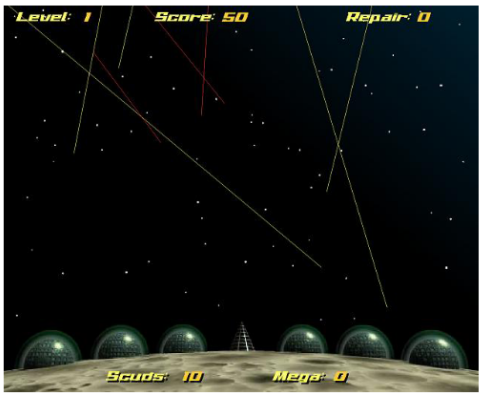


Figure 8: Players fired at missiles descending from the top of the screen by looking at them and pressing a button.

In this article, a comparison of player scores and/or time according to interaction modality was carried out. In all of the games studied, a significant decrease in score was observed when eye tracking was used. In addition, participants reported increased immersion when using eye-tracking in one of the games.

The results of this study were achieved by collecting objective and subjective data from the players. The objective data made it possible to establish the effective efficiency of the games played by using the time taken to complete the tasks and the score obtained by the players. The subjective criteria collected were used to determine the players' experience of these games.

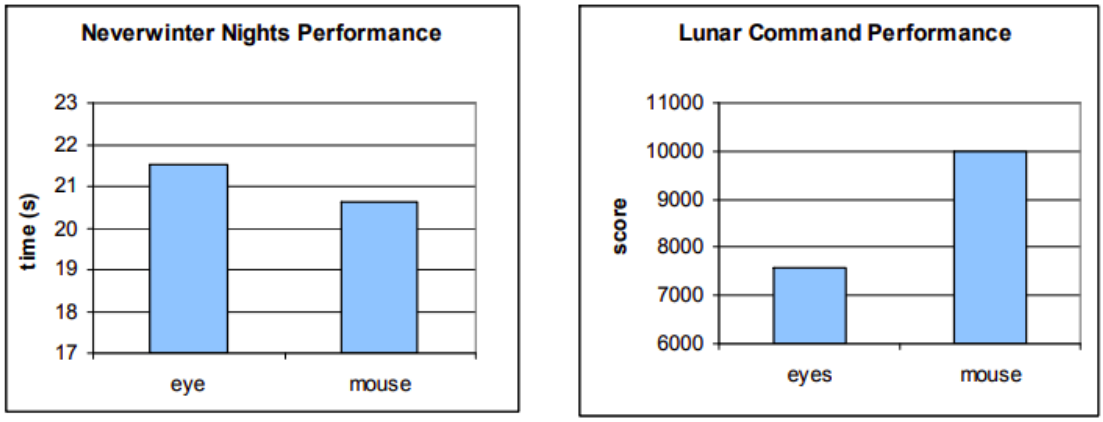


Figure 1: Objective Measures. Time of execution for the task and score for different game. In each case, the mouse perform better.

Question	Quake2		Neverwinter Nights		Lunar Command	
	Eyes (%)	Mouse (%)	Eyes (%)	Mouse (%)	Eyes (%)	Mouse (%)
Which did you enjoy playing with more?	42	58	83	17	42	58
Which was easier to learn?	33	67	67	33	33	67
Which was easier to use?	8	92	50	50	8	92
With which did you feel more immersed in the gaming world?	83	17	83	17	92	8
For which did the controls feel more natural?	25	75	67	33	42	58
Which would you prefer to use in the future?	33	67	67	33	42	58

Table 1: Analysis of subjective measures. Subjects were asked to indicate which modality they preferred along 6 different criteria.

Figure 2: Subjective measures: different questions used in

The results of the study indicate that the eye tracker can increase the amount of immersion experienced when playing a video game. Additionally, it is show that users found the eye tracker more enjoyable to use when playing a game that required the user to direct an avatar around a screen by pointing. This study also suggests that point-and-click games such as Whack-a-Mole or Fruit Ninja may be suitable for interaction with eye tracking. We set ourselves the goal of testing these assertions in our project.

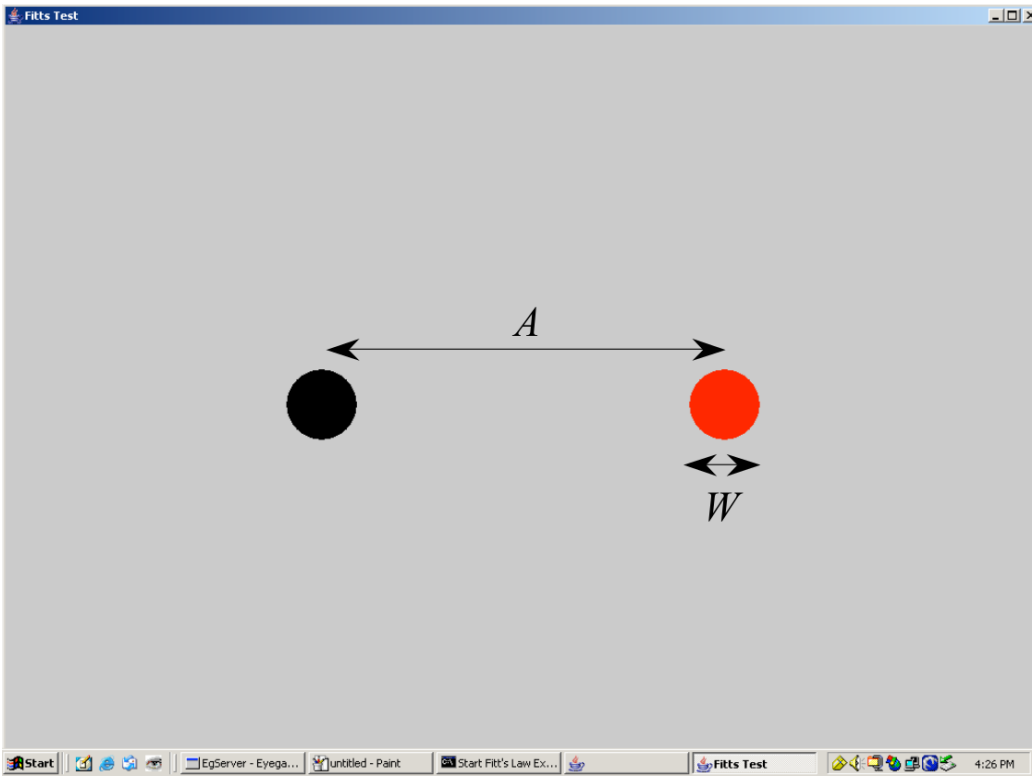
1.2.2 A Fitt's Law Comparison of Eye Tracking and Manual Input in the selection of Visual Targets

We were asked to apply Fitt's Law to our game. To do this, we were given a second article on Fitt's law and its use with the eye tracker. Fitt's law is explained later in our report.

The article is entitled: "A Fitt's Law Comparison of Eye Tracking and Manual Input in the selection of Visual Targets". In this article, a group of researchers tried to compare the performance of Eye tracking based input devices against manual input devices. They tested 4 input devices:

- Mouse
- Stylus
- Eye tracker with manual input
- Eye tracker with dwell-time input

The goal was to test the efficiency of clicking consequently between two points with each of the four devices.



As expected by the team, eye tracking devices are following Fitt's law for big target. But when those targets are smaller, the lack of accuracy of those devices doesn't help gathering relevant values. As you can see in the figure 3, the index of difficulty (depending of the size and the distance of the object), limit the values for the eye tracking devices.

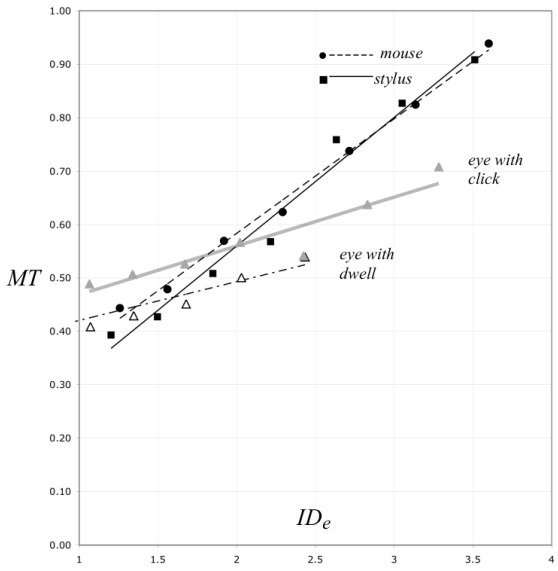


Figure 3. Regression lines per input device comparing Movement Time (s) with Effective Index of Difficultv.

Input Technique	Mouse	Stylus	Eye with Manual Click	Eye with Dwell Time Click
Fast	*3.7*	*3.1*	*3.9*	*4.3*
Accurate	*4.2*	*4.1*	*3.4*	*2.7*
Easy to Learn	*4.5*	*4.3*	*4.1*	*3.8*

Table 3. Mean score per questionnaire item for each input technique.

In conclusion, Eye tracking devices are faster but less accurate than manual input devices. Each Eye tracking device perform systematically better in the speed area compared to traditional manual input devices. But with the gain of speed comes the lose of accuracy. The error rate is greater for the Eye tracking based devices.

The research group mad the constatation that, Eye tracking devices are well fitted for the selection of objects for context than a coordinate based selection articulation.

1.3 Research Questions

Thanks to the articles we were able to find two research questions. We tried many attempt to merge those questions but they are two different subjects with only the eye tracker modality as an intersection.

- Question I: Are **instant point and click** based games better adapted to eye tracking modalities than mouse ?
- Question II: Does **Fitts's law** work with an eye tracking modality ?

The first one is a comparative study about the efficiency of one modality compared to the other. It try to find which one is the most suited to be applied in a well defined context (in this case, the context is an instant point and click game).

The second question try to analyse a specific property of the eye tracking modality. We want to know if it follow the rule that modalities like mouse or graphic tablet follow. It was also suggested to test it with a screen device like a table. Unfortunately, the design of our game don't really fit the requirements for a tablet usage.

Context of the game

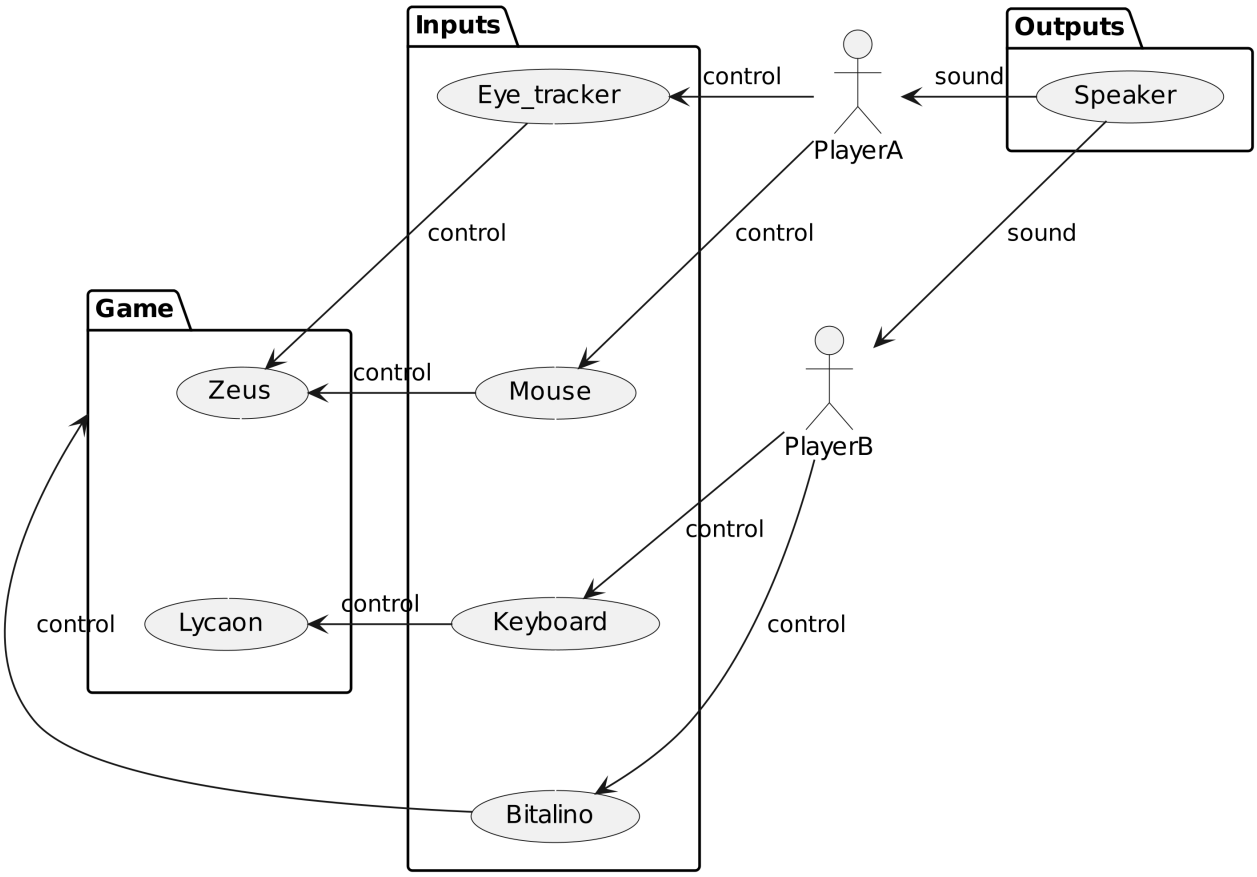
We set out to create a game where players take on the role of characters from Greek mythology. We were inspired by the myth of Lycaon.

According to legend, Lycaon was a king of Arcadia renowned for his cruelty and wickedness. One day, Zeus, the king of the gods, decided to descend to Earth to check whether the rumours of Lycaon's reprehensible deeds were true. He disguised himself as a man and went to Lycaon's palace to test his nature. Lycaon, suspecting that this stranger might be a god, decided to put him to the test. He organised a banquet and served Zeus human flesh, hoping to deceive him. However, Zeus, who knew the true content of the dish, was outraged and revealed his true identity.

To punish Lycaon and put an end to his cruelty, Zeus decided to turn him into a wolf. Lycaon underwent a terrible metamorphosis, becoming a ferocious, savage wolf. This transformation is the origin of the term "lycanthropy", which refers to the transformation of a human being into a werewolf. As well as the punishment inflicted on Lycaon, this story also illustrates the power and wrath of Zeus towards those who commit impious acts or defy the gods. The myth of Lycaon and Zeus is often quoted to underline the importance of hospitality and to warn of the consequences of contempt for the gods.

In our game, the scenario is a little different. Here, Lycaon has just deceived Zeus and is now trying to escape. Zeus won't transform Lycaon but just wants to kill him. The only way for Lycaon to survive is to jump into Tartarus, a place where Zeus has no power. In the game, the Tartarus is a hole at the end of the map.

To answer the research questions, we need to set up a test environment. This environment must be testable and reproducible. The test environment takes this form:



In this settings each player control one character. The player that use Zeus will use the mouse and the eye tracker modalities. The player controlling Lycaon will use the keyboard and the bitalino. We found that the bitalino can be used either by the player incarnating Zeus than the player incarnating Lycaon since both can be exited during the game. At the beginning we thought that we will only use the speaker for the user playing Lycaon, but both of them can hear the music, because this modality is used for the game immersion.

1.4 User tests

To respond to each research question, we made user tests. For the two question, we use the same protocol since everything is evaluated in a session.

Initial protocol

- Welcome the players and explain the goal of the project
- Give them a first evaluation questionnaire:
  - Current user of PC, mouse, eye tracker, video games ?
- Preparation:
  - Explain the game
- Have them play the game with measurements:
  - Eye tracker first then Mouse
  - Collect data for objective evaluation
- Post-experimental questionnaire
- After lab
  - save the questionnaire
  - save csv files

Questionnaires

- **Initial questionnaire**: To know about the tested population
- **User experience questionnaire**: To collect the subjective impression of the users.

The initial questionnaire give us an overview of the players. But we took only the person who have played Zeus, because only their feedback is relevant for the two research questions:

ID	point and click	Video game freq	Mouse freq	Eye tracker freq
0	yes	daily	daily	never
1	yes	yearly	daily	once
2	yes	daily	daily	never
3	yes	weekly	daily	never
4	no	monthly	daily	once
5	no	weekly	daily	never

We removed uninformative data such as the sex of the player since all of the participant where man. The majority of the player as an experience with point and click games. The group have an heterogeneous representation of frequency (how many time they play video game). Everyone use a computer daily (removed for space economy) and use a mouse in the same frequency, so they are acquainted with this modality. But most of them have never used an eye tracker an for two of the that was once in the past. So we can reasonably assert that their performances will be impacted.

1.4.1 Question 1

Are instant point and click based games better adapted to eye tracking modalities than mouse ?
Background

After reading the article on the use of the Eye tracker compared to the mouse on 3 different games and the fact that this study was carried out in 2006. We wanted to see if there was any change today. In fact, the technologies around eye tracking devices has evolved. Nowadays we have better qualities eye trackers and even VR headset. We made the hypothesis that those progress can lead to a better accuracy in the domain of point and click based games.

Our game has two players, each with their own game-play. The first is a player who has to survive by platforming and collecting items along the way. The second is an attacker who tries to kill his target.

We just need to be able to determine which of these two modalities is more accurate. To do this, we need to look for an evaluation metric.

Definition
The aim of our study is to see whether the use of eye trackers is better for point and click games.

Metrics
We have three metrics for evaluating an interface:

- Effectiveness
- Efficience
- Satisfaction

In our case, we are more interested in the efficiency (performance) and the satisfaction (immersion) of the two modality. That's why we will collect two kinds of information.

Accuracy

In the first article, the team used time to accomplish a task or the score of a session as a measure for the performance. But things are different in the case of our game.

It's a competitive zero-sum game. There will always be a winner and a loser. As a result, we won't be able to reproduce exactly the same evaluation context as in the study. The study was based mainly on hitting a stationary target. However, in our game, the target (Lycaon) tries to avoid the attacks of the aimer (Zeus), which means that he is constantly on the move and that assaults have a high chance of never hitting. We still have a way of measuring the difference by taking the margin of error (the distance between the point of impact and the position of the target). This hypothesis is based on the fact that a more precise object will always aim closer.

Maniabillity

We want to get feedback from users on how they felt while playing (Zeus). The aim is to find out whether the eye tracker modality is more pleasant and intuitive from the user's point of view.

Protocol

The method is quite simple. For each attack that the player playing as Zeus makes, we collect the position of the impact site and the position of Lycaon (the fleeing target). We also record the time elapsed since the start of the game.

strike pos	Lycaon pos	Lycaon speed	time
...	...	...	...

Where:

- **strike pos**: Coordinates (x,y,z?) of the lighting strike when the player controlling Zeus click.
- **Lycaon pos**: Coordinates (x,y,z?) of Lycaon when the impact occur.
- **Lycaon speed**: Speed of Lycaon when the lighting strike occur.
- **time**: Time of the impact since the beginning of the game.

Precision

From there, we simply measure the error by calculating the norm of the distance between the points of contact. In this way, we can average the error between each game. This average is calculated in pixels on the game screen.

This data is collected automatically by the game and stored in a csv file.

Note that we collect the coordinates of the point of impact and Lycaon directly without calculating the distance to avoid impacting the game's performance.

Handiness

We use a binary questionnaire to determine which modality is the most preferred according to these criteria:

- **Most enjoyed**: The one that players enjoyed the most in terms of gaming experience. We're not talking about performance here.Review
- **Easiest**: This measures the cognitive impact of using these modalities.
- **Natural**: This measures the relevance of the modality in the context of a point and click game.
- **Best at**: This measures the subjective level of performance perceived by the user.
- **Harder**: This is just the opposite of Easiest.

It should be noted that these measures have been taken from the SUS (System Usability Scale) document. We have removed certain questions that we felt were redundant or not related to the topic and we also adapt the remaining ones to fit the goal of our research.

Actions taken

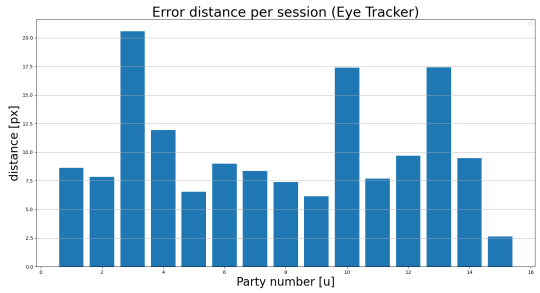
We first welcomed the players and then began a calibration phase with the material. We then had our guinea pigs play. The player embodying Zeus first started a few games with the eye tracker and then finished with the mouse.

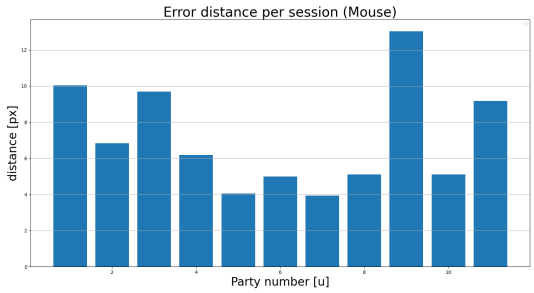
We collected around 5 games for the eye tracker and 3 games for the mouse.

Analysis of results

Accuracy

The first thing we note is that accuracy rates could vary greatly depending on the player. We begin with an analysis based solely on the games. We began by measuring the accuracy error rate (the distance between the point of impact and the position of the fleeing object) for each attack. We made one for the mouse modality

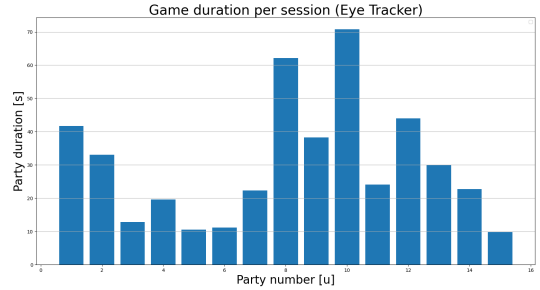


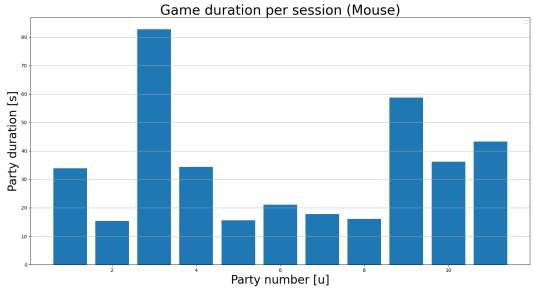


and another for the Eye tracker modality.

As shown in these graphs, we can see a great disparity between each part. This does not allow us to establish much about the general effectiveness of one modality over another in terms of parts.

We also made an observation about the duration of the games using the time elapsed in each game. This could be a good indicator of the effectiveness of one modality over another, as a more precise modality will allow the game to be finished more quickly. Given that none of our players playing Lycaon managed to reach the finish line before being killed. We are certain that the last attack of each game is a decisive one, eliminating the player playing Lycaon. Here's another comparison between each game.





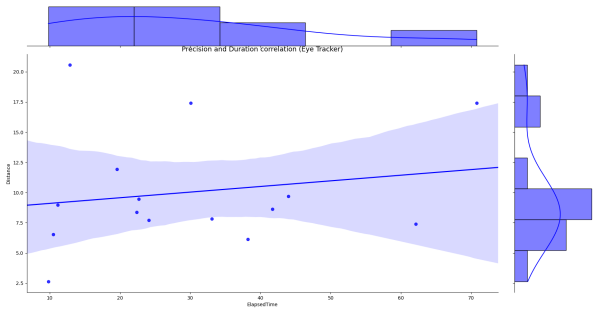
The results are quite similar to the previous graphs and will require further study of the use of this type of programme.

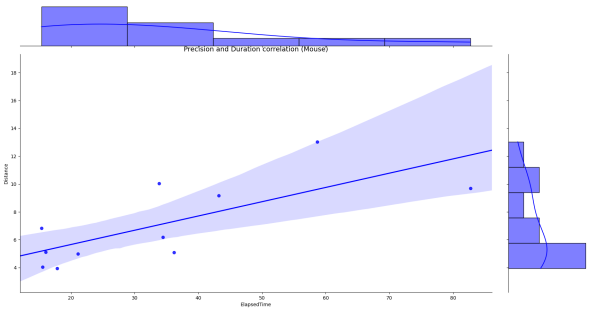
We can see that these graphs are fairly correlated.

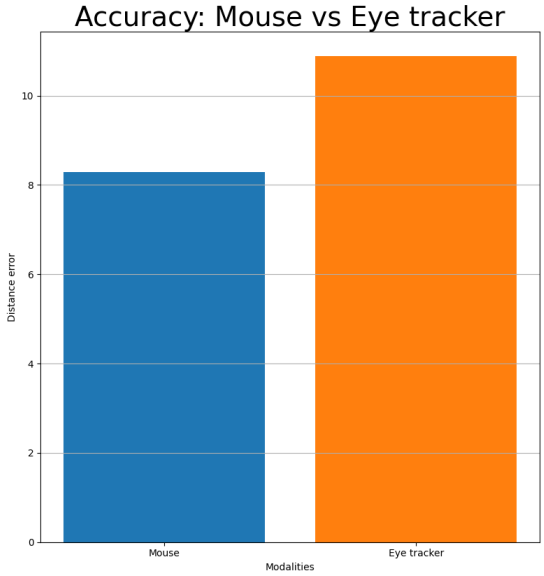
This shows us that there would be a way of explaining the same things with the measurement of time or the precision error. In our case, we concentrate more on the precision error, which has the advantage of giving us more information about what is happening in a game.

We can also see that the margin of error for games played with the mouse is smaller than the margin of error for games played with the Eye tracker.

The overall result shows that the mouse is more accurate than the eye tracker.







User feedback questionnaire
We have also gathered feedback from players who have used Zeus.

ID	Most enjoyed	Easiest	Natural	Best at	Harder
0	Eye tracker	Mouse	Eye tracker	Mouse	Eye tracker
1	Eye tracker	Eye tracking	Eye tracker	Eye tracking	Mouse
2	Mouse	Mouse	Eye tracker	Mouse	Eye tracker
3	Eye tracker	Eye tracker	Eye tracker	Eye tracker	Mouse
4	Eye tracker	Mouse	Eye tracker	Mouse	Eye tracker
5	Eye tracker	Mouse	Eye tracker	Mouse	Eye tracker

The results are quite interesting. Almost all the players liked the eye tracker better than the mouse. They were also unanimous in saying that the eye tracker was the most natural way to play. However, the mouse remained the easiest modality for the majority of users and remains the modality for which users feel they have achieved the best results (which seems to have been confirmed in the objective measurements). The results confirm the study carried out in the first article concerning the immersion potential offered by the eye tracker compared with the mouse. What's more, the eye tracker is seen as a natural tool because it adapts to our eye movements. So the learning curve is relatively short.

Interpretation

These results can be interpreted in different ways. First of all, let's talk about the factors that influence the mouse's superiority.

Our population is made up partly of gamers and partly of non-gamers, but they all use a computer and the mouse on a daily basis. As a result, there is a strong influence in terms of mastery of this tool. In terms of gaming, the mouse was the second modality, which meant that she was able to benefit from the learning effect, which can significantly improve her results.

Conversely, the factors that favour the mouse's performance are also factors that don't favour the Eye tracker's performance. For example, all the people who took part in the user test had either never used the Eye tracker or had used it once in the past. Calibration was also a problem. Indeed, despite the preparation of a good quality experimental environment. Users could have calibration problems with the eye tracker, which required the player to be well positioned (and almost motionless) when playing. There were also problems with a mismatch between the object being pointed at and the position of the ray trace (a kind of target for pointing at what the pointer is aiming at). Some users wore glasses, which can also influence the calibration and therefore the accuracy.

A factor that may work in favour of the eye tracker comes from the second study. In fact, it is explained that where the mouse is effective in pointing to coordinates in space. The eye tracker is more effective for selecting objects in space. In our game, the user had to focus on either Lycaon or the eye tracker. So the eye tracker can be really effective if the calibration is optimal.

Conclusion

The results obtained do not allow us to state with certainty that the mouse is a better modality than the eye tracker. One way of verifying this would be to carry out the tests starting with the mouse to check the learning phenomenon. The tests could also be carried out with a larger population of people with good vision. Or use more modern technologies such as VR headsets, which provide more reliable calibration.

1.4.2 Question 2

Does Fitts's law work with an eye tracking modality ?

Background

At the request of the faculty, we have studied Fitt's Law to see if the Eye tracker modality does indeed comply with this law. With this request, we also received an article confirming that the Eye tracker complies with this law.

Definition

Fitt's law is a law of interfaces that states the following:

The larger the object and the closer it is to the pointing device, the shorter the access time. This law is expressed using a formula. The larger and closer an object is to the cursor, the shorter the access time.

$$T = a + b * log_2\left(1 + \left(\frac{D}{W}\right)\right)$$

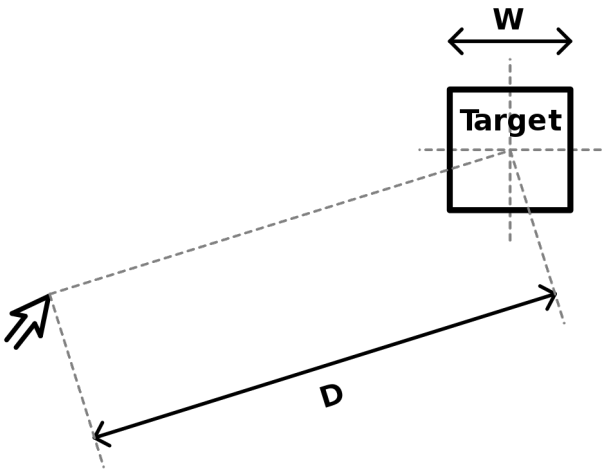


Figure 3: This is a visual representation of Fitt's law. We have in the left, a mouse that represent the pointing device and we have in the right, a target object that have a size represented by it's width (W). There is also a distance between this two entities represented by D

where,

- T is the time to move the cursor to the object
- a, b are parameters fitted to the observed data

- D is the distance between the cursor and the object
- W is the size of the object being pointed at

TODO: add the computation of the ID index of difficulty
Metrics

As explained in the previous research question, we have a moving target. However, in the article, the experiment was carried out with two stationary targets. To be able to measure Fitt's law, we add a new mechanism: nectar. In Greek mythology, nectar is a divine drink that enabled the gods of Olympus to maintain their immortality, eternal youth and power. We're adding this mechanism to the gameplay of Zeus. After a certain number of lightning bolts (3 in total), Zeus becomes tired and must consume nectar to regain his strength.

The nectar is an object that appears on the screen of the player who is playing Zeus. When it appears, the player can do nothing other than select the nectar. In this way, we are able to simulate an experience similar to the one in the article, as the nectar is immobile from Zeus's point of view.

The nectar appears in the game as a 3D object, as does the ray trace, which can be controlled using the mouse or eye tracker. A Unity function is used to project any 3D object in the game onto the 2D plane of the screen. This gives us objects with measurable metrics for evaluating Fitt's law.

Protocol

When the nectar appears on the screen, a stopwatch is started and does not stop until the player embodying Zeus clicks on the nectar. We collect the respective positions of the nectar and the cursor, the size of the nectar and the capture time. Otherwise, the protocol remains the same as the previous research question.

pointer pos	nectar pos	task time (T)	width (W)	game time
...	...	...	...	...

Where:

- **pointer pos**: Coordinates (x,y) of the ray trace of the pointer (mouse or eye tracker) when the nectar appear.
- **nectar pos**: Coordinates (x,y) of the nectar when it appear.
- **task time**: The time taken by the user controlling Zeus to click the nectar after it appear.
- **game time**: Time since the beginning of a session.

Actions taken

The information is collected in a separate csv file ("fitt.csv").

Analysis

Unfortunately, the results are not conclusive for the two modalities. You can see the graph for the eye tracker and for the mouse.

As shown in these graphs. The relationship between task difficulty (Index of difficulty) and task completion time (effective time) seems doubtful. These results seem surprising given that when we tested the game, we found much more regular results. We don't make any more graphs given the results obtained.

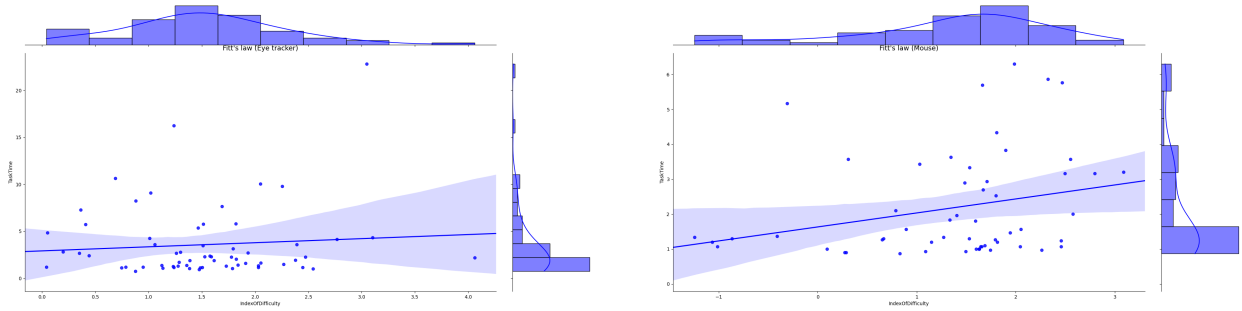


Figure 4: Those are the graph representing the points given for Fitt's law. The left one represent the measure mad with the eye tracker and the right one represent measures made by the mouse. We have the index of difficulty for the x axis and the time to perform the task

Interpretation

There are a few points to consider in explaining these results. The first element to consider is the ability of the player embodying Zeus to change context. We're talking about context here because it goes well beyond changing the target. The aim of the player embodying Zeus is to kill his opponent, which becomes his first priority. This leads the player to concentrate so much on the moving target that they don't notice the nectar appearing on the screen.

The first point is the main reason why the results are wrong, and it's difficult to remove incorrect points because the time taken to notice the nectar can vary between 0.5 and 1 second.

In addition to the quality, the quantity of the data is not in our favour. If we have collected enough click data in the first search question. We see this number divided by three for this question (as a reminder, the nectar appeared after three attacks). We can see from the graphs that we don't have many points overall and therefore too few results when we take each player individually.

Conclusion

The results collected are few in number and contain a variation factor that is too large and unpredictable to give a conclusive result. Given that the Eye tracker is supposed to follow Fitt's law, we could stop at this graph and say that the results do seem to show a straight line, but it would be better to base ourselves on a collection with better quantitative and qualitative data.

2 Zeusfury : Multimodal Mayhem

We developed our game in the Unity Game Engine as recommended by the course. You can find our source code, under MIT license at our git repository. We used multiple free assets from the Unity Assets Store and from the free assets given as part of the Junior Programmer Pathway. Our background in Computer Science made completing these pathways straightforward and in the fraction of the projected time, we still learnt quite a few interesting things, especially Unity specific notions, patterns and architectures.

2.1 Education

Unity has it's own learning center called Unity Learn. We used this extensively to familiarize ourselves with the editor. The development of the Game, game logic, prefab designs and implementations was the work of A. Freeman whilst the literature review and establishment of the testing protocol and the statistical anylsis was the work of H. Hategekimana.

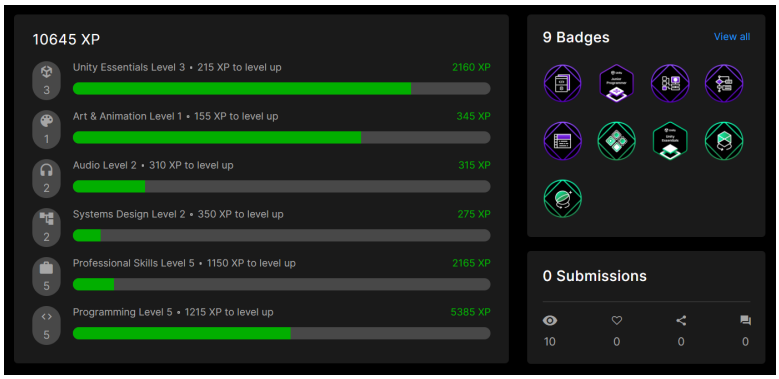


Figure 5: A. Freeman's Unity Learn Progress

We both completed Unity Essentials pathways to be familiar with the editor, A. Freeman completed the Junior Programmer Pathway to be able to be fully functional in Unity scripting. Multiple optional challenges were implemented in the courses for the sake of practice and free assets from those tasks made their way into our game.

2.2 Assets

- Low Poly Simple Nature Pack this was the main asset pack that contained the trees, rocks, ground areas, grass flowers, shrubs, mushrooms etc... It was extensively used when designed the different strip types.
- Low Poly Cliff Pack the cliff pack was great to design the cliff strip with some extra challenge due to it containing canyons Lycaon had to jump across. This forced players to use the spacebar and dash accordingly to get through the canyons.
- Lightning Bolt this asset was extremely useful for designing the lightning strikes coming from Zeus. It was just a matter of programatically configuring the start and end positions of the lightning strike, the degree of chaos (squigliness of the line), thickness and setting it to be triggered upon enter. We were lucky such an asset existed as it could easily be quite expensive !
- Low Poly Cloud Pack this was a simple cloud pack that we used to design the large cumulus from the which Zeus casts his lightning strikes. It's simple though a user did comment they thought it represented a rock.

There were multiple other assets that we tried using, notably one that contained a script with a fully fledged fps controller, but it resulted too complex to adapt to our context. Some assets may still be present in the project though not used, there is still some cleanup in the assets folder to be done.

2.3 Strip Prefabs

We created a base prefab of a strip of land from the which we created a couple of variants. It initially was a simple plain and the two variants were pine forest and jungle forest. There is also a cliff prefab from the which we created no variants.

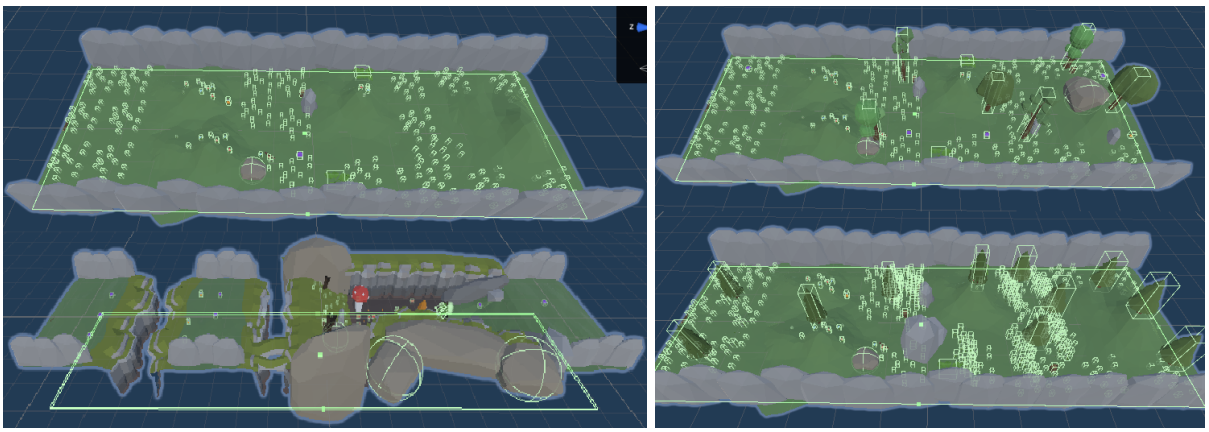


Figure 6: Cliff and Green Strip Prefabs (left), Green Strip Prefab Variants (right), note that the bounding boxes match the extremities of the strips, the visualisation is distorted due to the rendering.

The nice thing about prefab variants is that they inherit modifications from the parent, e.g. if we change the parent the children are modified as well, but not the other way around. Having multiple strips with varying amounts of obstacles that Lycaon can hide behind resulted in better game balancing.

2.4 Strip Spawning Algorithm

To spawn the strips one after the other, we used bounding boxes of colliders to figure out the positions at the which to spawn new strips as the game progressed. All this happens in the `spawnStrip()` function from `GameManager.cs`.

```
public List<GameObject> stripPrefabs;
public Queue<GameObject> strips = new Queue<GameObject>();
...
```

```
1 void spawnStrip() {
2     int k = Random.Range(0, stripPrefabs.Count);
3     GameObject newStrip = Instantiate(stripPrefabs[k]);
4     Bounds precStripBounds = strips.ToArray()[strips.Count-1].GetComponent<BoxCollider>().bounds;
5     Bounds newStripBounds = newStrip.GetComponent<BoxCollider>().bounds;
6     newStrip.transform.position = strips.ToArray()[strips.Count-1].transform.position +
7     Vector3.forward * (precStripBounds.size.z+newStripBounds.size.z) / 2;
8     strips.Enqueue(newStrip);
9 }
```

`stripPrefabs` is a list of the available strip prefabs, upon the spawning of a new strip we choose randomly which prefab to spawn (line 2) and we instantiate it (lin 3). The challenge is then to position it correctly, e.g. it's "start" has to stick with the "end" of the latest spawned strip. We use a queue called `strips` for the sake of managing the logic as it's perfectly suited for the task : when we spawn a strip, we enquue it, when we remove a strip when out of frame, we dequeu and call destroy on it.

Figure 7: Bounding Box of Strip Prefab managed by Box Collider, notice the alignment with the z-axis.

The logic of lines 6-7 is that the position at the which we want to spawn the new prefab will be a vector sum of the position of the next strip in the queue, e.g. the strip at the before last position in the array. We sum that position with a multiple of `Vector3.forward` which is a shorthand for Vector(0, 0, 1), which is co-linear with the strips, multiplied by the half width of the before last bound width summed with that of the strip being added.

2.5 Split Screen Setup

The split screen setup of the game was quite technical to put into place. We initially tried to do it with vannila Unity cameras, but very quickly we found out the preferred solution was to use the Cinemachine package. Indeed, when it comes to implementing things like cameras following players, shaking when moving etc, the implementations are always similar, Cinemachine offers interfaces to quickly activate these behaviours without having to implement them ourselves. With a traditional Unity camera, following Lycaon would require moving the camera to a new position with a certain offset at every `LateUpdate()` call whilst with cinemachine it's as simple as creating a Cinemachine virtual camera and assigning a `Follow Player` game object and a `Look At`.

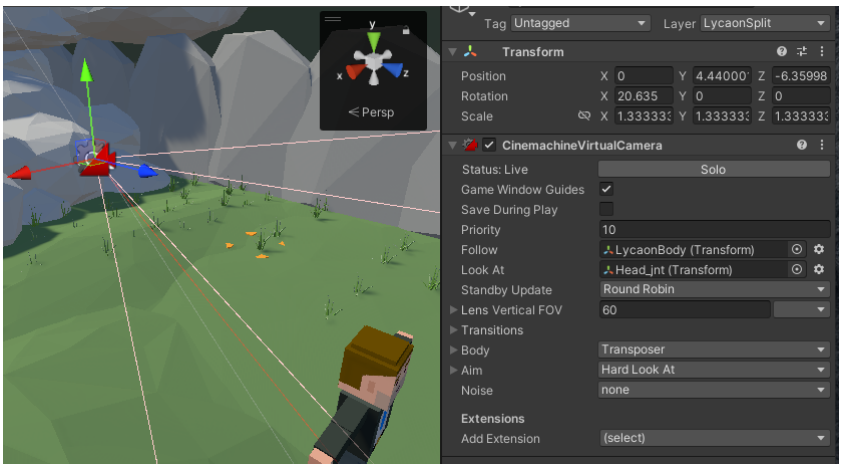


Figure 8: Virtual Camera Confign note the `Follow` set to LycaonBody's transform, e.g. the camera is always following Lycaon's Body whilst the `Look At` is set to Lycaon's head, in order to be able to see the map over Lycaon's head. This simple configuration allows us to rotate Lycaon's Body from the script attached to LycaonBody, `LycaonController.cs` and the camera follows the rotation (controllable via `A`, `D` keys)

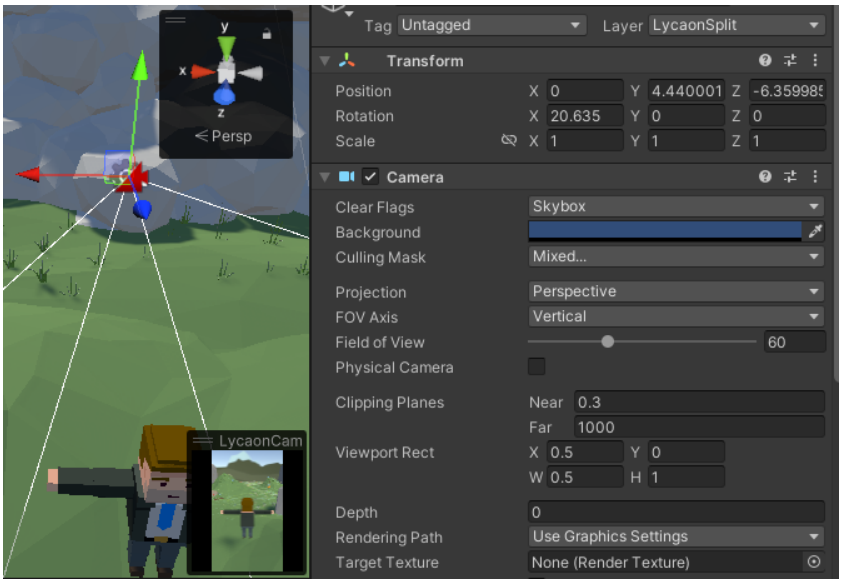


Figure 9: Camera viewport config, $W = 0.5$ means the camera will render only on half of the screen width whilst $X = 0.5$ means it'll generate starting from the bottom middle, $H = 1$ means it'll take up the full screen height, an analogous configuration for the left split view is done in Zeus's Cinemachine Brain.

Since we were using a split screen config, we had to use multiple Cinemachine Brains. Note that the layers

`ZeusSplit` and `LycaonSplit` are necessary to segregate which Cinemachine Brain manages which Virtual Camera. A cinemachine brain allows the management of multiple Cinemachine Virtual Cameras, virtual cameras are not Unity physical cameras, they're point of views between which the cinemachine brain, that contains a physical Unity camera, can blend. Take for instance a car game with a first person and third person view, you might want to be able to gently switch between both views or smoothly transition via an animation. The general workflow would then be to place to Virtual Cameras, one for the third person view and one for the first, then configure blending via a new custom blend and specify between which cameras and add a blend style. Cinemachine does all the heavy lifting in such a scenario, we never needed to switch between cameras so we didn't use blending features, however since we were using Virtual Cameras for the follow and shake features, we needed a Cinemachine Brain per camera for it to work (as they contain physical unity cameras), so each Brain only managed one virtual camera via the culling mask as described in the documentation that ignored the Virtual Camera of the other camera.

The features we used were to make a camera follow Lycaon and shake as he moved whilst Zeus's camera was manually moved forward. The viewport rects were configured to be able to split the screen in two.

2.6 Scripts Architecture

A global scripts architecture, communication links, main functions and responsibilities is presented in the diagram below. Note that we only include the non-trivial scripts we've written, there are others but we've omitted them here for the sake of being concise.

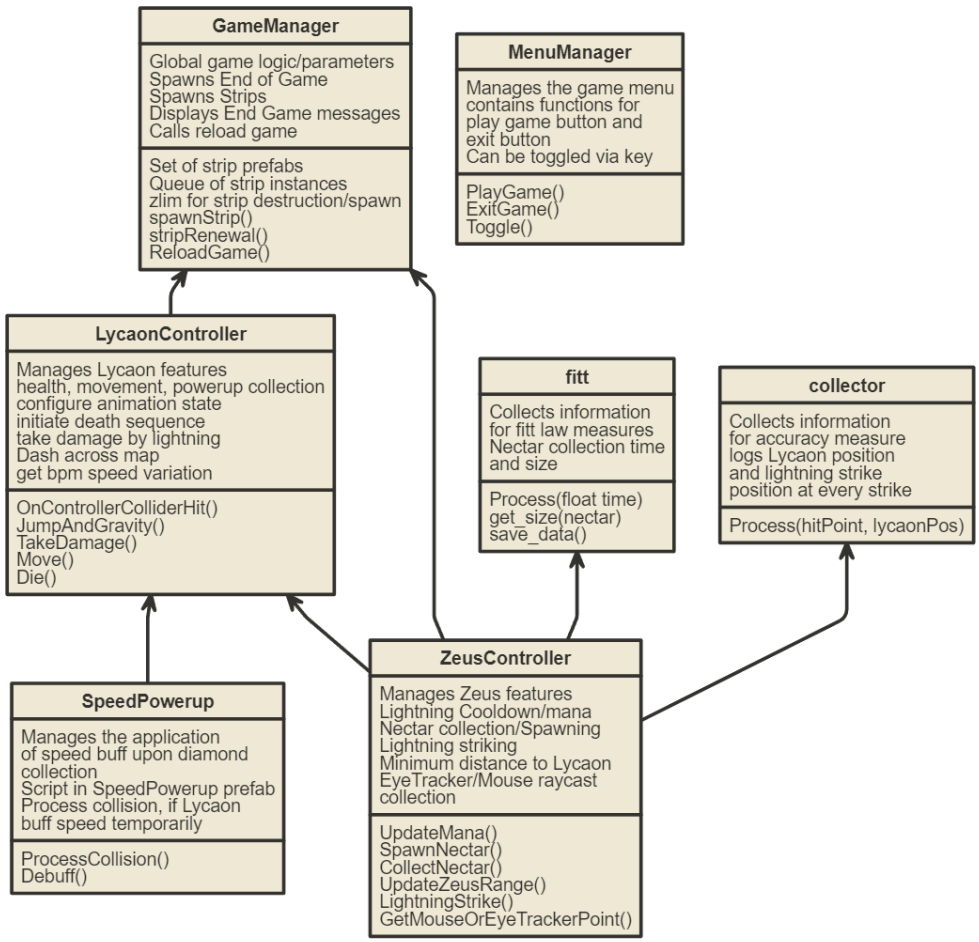


Figure 10: Main Scripts Architecture, every class is attached as a script component to a GameObject or Prefab of the same name in the hierarchy or Prefab folder, the arrows represent *script communication* via the use of `GameObject.Find(...).GetComponent<ScriptClass>()`.

We could talk extensively about implementation details, but we let the user consult the code for himself available at our git repository under MIT license.

2.7 Modality Integration

Integrating our modalities was the most painful part of the project. The wings of faith assigned us the Bitalino and Eye Tracker modality. We were initially working entirely on Linux, unfortunately, to be able to interface with our Tobii Pro Nano eye tracker we had to use a windows based machine. We tried interfacing via Linux, but gave up due to the lack of documentation and support. We had to forcefully borrow a VR gaming laptop from the faclab to be able to run our experiments and for F. Hategakimana to be able to work from home. A. Freeman had to transfer hig recreational PC to his office and configure it as a workstation. The general workflow to integrate a modality into our game is summed up in the activity diagram below.

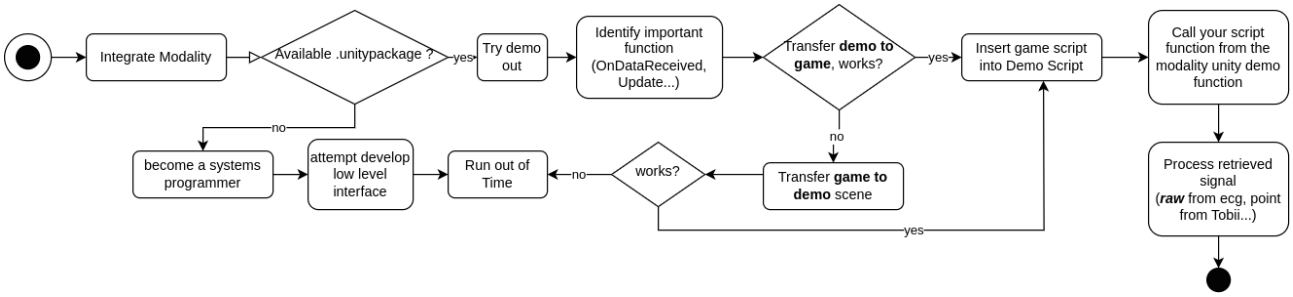


Figure 11: General steps followed to get signal acquisition from the Eye tracker and Bitalino into Unity. Note that the `Run out of Time` box is there for comic relief.

The first thing to do was to find a `.unitypackage` with a demo of the modality in use. We were extremely fortunate as one existed for the Bitalino thanks to pluxbiosignal's unity demo and one existed for the eye tracker thanks to Tobii's Unity SDK (download starts after four re-directs). Note that Tobii's website was very confusing to use, and finding what we were looking for took time.

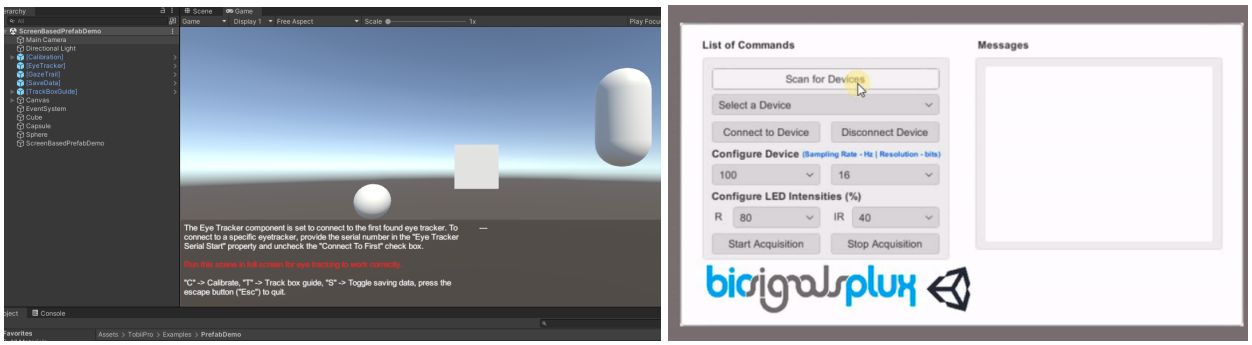


Figure 12: Tobii Unity SDK Demo Scene where looking at the objects draws green points where you're looking (left) and pluxbiosignal's unity demo Canvas GameObject containing functionality to connect to Bitalino and extract raw ecg signal (right)

2.7.1 Eye tracker

In the case of the eye tracker, we were initially overwhelmed by the gargantuan amount of scripts and assets that came with the SDK, we didn't know where to look and had to reverse engineer the above demo (after finding it in the first place!) to be able to understand where we could find the point being looked at. The code snippet we needed was in `GazeTrailBase.cs, Assets>TobiiPro>Common>Scripts`.

```
if (HasEyeTracker && _on) {
    Ray ray;
    var valid = GetRay(out ray);
    if (valid) {
        RaycastHit hit;
        if (Physics.Raycast(ray, out hit)) {
            latestHitPoint = hit.point;
            PlaceParticle(hit.point, _color, _particleSize);
            _latestHitObject = hit.transform;
        } else {
            _latestHitObject = null;
        }}}
```

From this scrip we see that the variable `_latestHitObject` contains the point of collision that interests us. From then on it was as simple as making a copy of that variable accessible publicly and getting the value into `ZeusController` via `GameObject.Find("[GazeTrail]").GetComponent<GazeTrailBase>().latestHitPoint`. To get this to work though we initially tried transferring the gameobjects from the left side of the above figure into our game, that did not work, no data was being obtained, so we had to resort to *transfering our entire game* to the demo scene and work from there. This was unpleasant but finally worked in the end. Note that to do the transfer we simply imported the unity package into our game and transferred our game assets from the game main scene to the demo scene.

2.7.2 Bitalino ECG

In the case of the ECG, we were faced with the simple problem of signal processing. We were able to obtain the numerical value of the collected signal every 10 seconds, but how do we extract a heart rate from it ? It turns out that work isn't implemented at a hardware level, and we weren't able to find a Bitalino API to do this for the PPG sensor we were using. There are implementations of signal processing algorithms for heart rate extraction such as the Pan-Tompkins algorithm but their implementation in csharp is one hefty task. Such implementations do however exist in python, notably in `scipy` which is the solution we opted for : simply extract the bitalino signal from python, process it there and write the BPM to a file, which is then read by Unity. This simplified things drastically and decoupled the bitalino from our game.

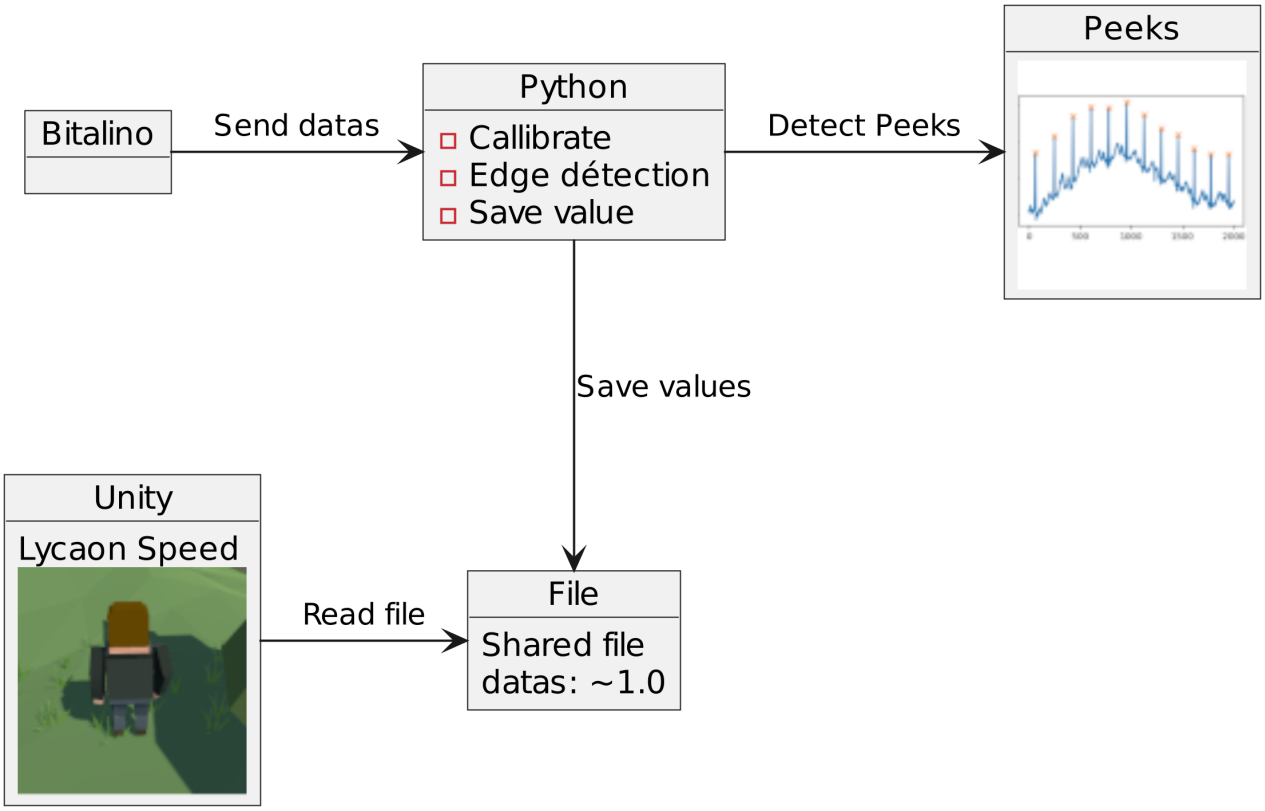


Figure 13: General Interaction between Unity, shared File, Python and Bitalino, note that the readout BPM is then scaled and added to Lycaon's speed in `LycaonController.cs`

.

The python scripts are all available in the `Assets` folder.

We had the arousal modality to add to our game. We tried using the bitalino to do this.

We ran into problems getting the sensor to work. In fact, we found very few sources mentioning it and the integration done by Unity only allowed us to capture the data. However, there is no specific C module for analysing cardiac data, so we had to implement the calculations ourselves.

We chose to do a 'polyglot' project. We used python, which has all the resources needed to integrate the bitalino and analyse the data generated.

We've written a mini-program that connects to the bitalino, reads the player's heartbeat, measures variations in heart rate and stores them in a file (value.txt). The value stored is a measurement relative to the pulse measured in relation to the calibration. This value is relative to the pulse at rest, so 1.0 is the value when the heart rate is normal and we should find values above 1.0 when the user's pulse increases and conversely below 1.0 when it decreases.

The game retrieves the value contained in the file to update its speed.

The player hangs the bitalino on the end of his little finger. This is done to avoid disturbing the player with his main fingers (thumb, index and middle fingers).

When we launch the program, we are presented with a shell into which we can enter our commands. To run a calibration, simply enter the command 'calibrate'. By default, this command measures the user's heart rate for 1 minute. We can also add numbers as parameters (such as "calibrate 15"), this value represents the number of seconds we want the heart rate to be collected. Once the frequency has been captured, the program automatically performs the calculations.

The program uses scipy's find˙peeks function, which is a vertex detection method using a comparison algorithm between neighbouring points. This algorithm has the advantage of letting the user define a threshold, allowing us to avoid vertices that are not part of the observed pulse. This threshold can be defined in the shell with the command "height [number]" (for example "height 700"). The program will use matplotlib to generate a graph representing the heart rate measured over the calibration time and will display a horizontal line in red to show the threshold.

You can perform as many calibrations as you like and adjust the detection of vertices according to the power and perception of the user's pulse. It is recommended that calibration is performed over 1 minute to obtain a good quality result, which is one of the reasons why 60 seconds is the default value for calibration, although this can be defined with the "calibrate [number]" command. It should be noted that a person's pulse normally varies between 60 and 100 bpm. We have therefore taken 80 bpm as the default value.

Once the calibration is complete, we have the user's bpm value. We can now run the scan over the long term. To do this, simply issue the 'start' command to begin the capture. This capture will continue indefinitely unless we stop it. It will capture the cardiac values for ten seconds and calculate the variation in relation to the default bpm. The capture will then create a relative value (1.0 if it falls on the same bpm) and store this value in the "value.txt" file.