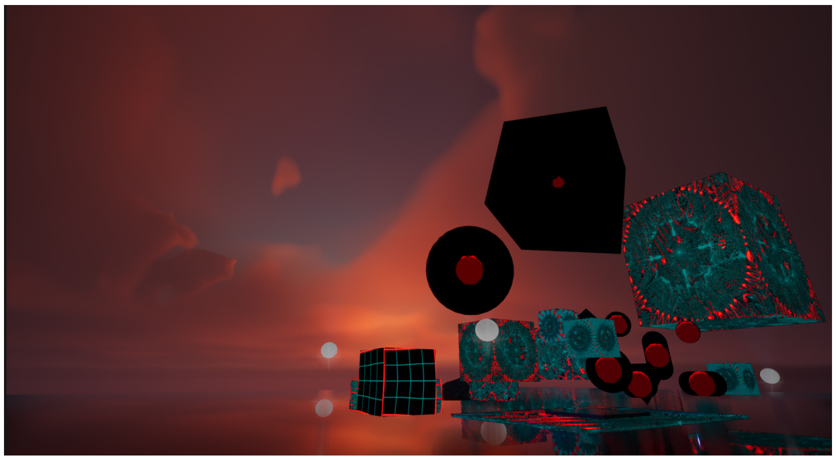
Concordia University

Self-Locator
An embodied ML interaction in VR

Leonardo Morales
Maya Marshel
CART 398

December 11, 2023

Abstract



Self locator is an embodied machine learning interaction built in Unreal Engine 5, with the InteractML plug-in (InteractML). It extends on work of the InteractML team by building from their template project available on GitHub. This asymmetrical collaborative VR experience is an exploration of gameplay as performance, and what real-time ML driven behavior can afford to the construction of a collaborative visual space.

Metaphorically the piece is meant to represent the continual act of self location in conscious experience, the way we navigate mental maps and models of reality using 'mechanisms of semantic descent' from one representational medium to another. Taking the notion of fixed-point representation, the exemplairs of a red dot on a map, and a frame of reference visualized as a grid in a representational domain from the theoretical work of cognitive philosopher Jennan Ismael in her book titled 'The Situated Self' (Ismael).

Self-Locator tries to answer, through a two player digital interaction dynamic, the question of how communication and partnership can affect the navigation and creation of mental maps and representational mediums.

Introduction

The world around us is a complex system of sensory stimuli. We constantly create simple shorthand representations of places, relationships, experiences and data in order to situate ourselves in the context of information spaces and understand our experience within the physical world. We continuously 'self locate' in order to fix these mental maps in a changing context. In our project we discuss the creation of these 'maps' by representing them as actual physical spaces.

The environment we created features dynamically changing configurations of colored lights and evolving shader effects depicting grids that consistently morph and distort. These dynamic elements are parametrized and driven by ML models that can be trained in real time on player position data. Two people are involved in this live performance that takes mechanical inspiration from asymmetrical VR games and interactive AI systems. One player is situated in the virtual space, wearing a VR headset, while the other controls the look and feel of the environment by cycling through light and texture value states from a computer. Both performers cooperate to craft a visual performance, the process of creation of which can be broadcast live to an audience. Self-Locator explores how communication can effect the creation of a Mental Map and consequently one's understanding of the world. The asymmetrical control over the stages of data capture, training, and visual state manipulation means that both players have a large say in the creation of the final 'Mental Map' and the subsequent exploration of spatial position to sensory configuration resulting from the interaction.

As described in Jenann Ismael's The Situated Self, a red dot within a map serves as an anchor, a fixed point, between the map representation of a space and the space itself, in relation to one standing at that location. In the same way, a red sphere and a 3D map or level may serve as a mental space for the manifestation of two players' intentions. One player in VR has control of the sphere while another viewing a 2D screen controls the visual state of the environment remotely through an interface. The two players must coordinate to define location-to-visual configuration mappings to iteratively train an ML model in real time.

Drawing off of past works, the surreal multiplayer puzzle and mystery experience "The Under Presents" showcases an interactive AI experience that uses machine learning technology as an artistic tool (Tender Claws). The multiplayer aspect of the work uses gameplay as a means of creating a new form of communication in a digital space. Effectively it takes the ones and zeros of a computer's medium of representation and transposes it to the fluid domain of conversation. Thereby emphasizing the element of 'humanness' in differently structured protocols of interaction between humans and machine learning technology.

Exploring the ways we can create new player driven game experiences using machine learning is at the forefront of this project. We have drawn from ideas of great thinkers and designers in order to create a rudimentary representation of this idea while also touching on certain human truths that this project represents.

Project Description

For the creation of our gamespace we decided to use the Unreal Engine (Epic Games). The Unreal Engine is a game engine that is more specialized to three dimensional games than an engine like Unity. Unreal Engine also employs a type of code interface called Blueprints. Blueprints are a form of visual coding that can be used in the same way as, for example, C# can, but are often more intuitive to new users (our two person team comprised a mixed level of experience). Figure 1 shows an example of Blueprints in Unreal Engine.

Instead of using a program like Wekinator we decided to use InteractML's plug-in ported for Unreal (originally developed as an interface for Unity). This allowed us to use more complex machine learning code since the algorithms were already built into the engine we decided to use. This also meant we were able to take a closer look into a more complicated use of machine learning algorithms and see how it worked with our program specifically. InteractML provides a series of demos that showcase different forms of machine learning inside the game engine. For our purposes we used a regression task.

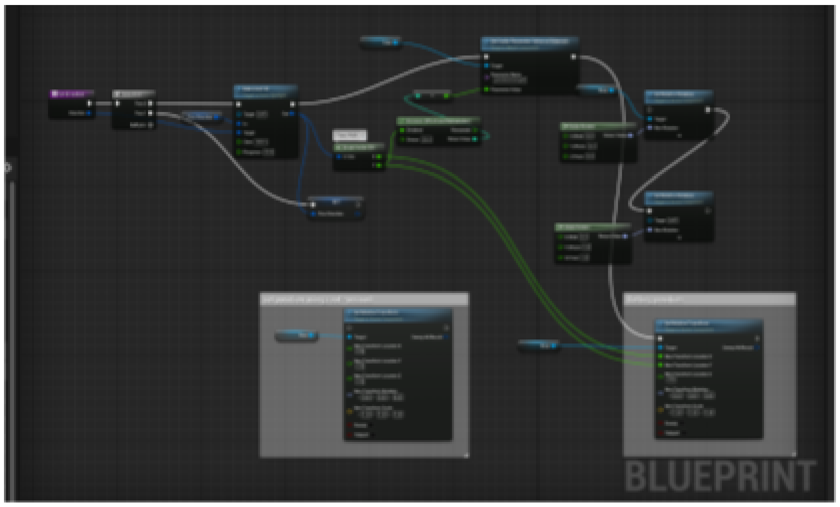


Fig. 1. Blueprints from Unreal Engine, written by InteractML, 2022, Modification of dynamic light blueprint, extending its ability to change position and material parameters according to label data state change

To make this into an 'embodied interaction' we decided to take the game off the 2D surface of the screen and put it into the real world using an Oculus Rift Virtual Reality (VR) headset. This headset comes with two hand controllers and a headset both of which are able to track the player's position in space without the need for sensors. To connect the Oculus with Unreal Engine we used Steam VR, and the Oculus app as well as their corresponding in-editor plugins for Unreal Engine, which allowed for the syncing of data from the PC monitor to the display of the headset and the controller input (Valve) (Luckey).

Like previously mentioned we used a simple regression task, starting from a demo scene included in the InteractML sample project. In this task the user would be able to train the position of spotlights based on the XY value of their digital avatar using first person character controls. The input was the position of the avatar in space using a 2D Vector coordinate (two float values representing X and Y) while the output was six different position states of the lights (with accompanying colors) which we would use to train the model. When the model is trained, then run, the sampling results in a seamless interpolation (or blending) between these different states.

For our project we wanted first to modify the code so that it would work the exact same way when using the Oculus as primary input instead of the computer, tracking the position of the player using the headset's XY coordinates instead of the digital player avatar. Once we got it working in VR we then set out to add new parameters to the output states so that we could change more aspects of the environment. Ideally we would have added area lights, sounds, texture warping, particle effects and more. Later in this paper the problems with this plan will be detailed. The goal, however, to modulate the demo so that the environment looked completely different from the original code, was achieved.

Finally we wanted to re-skin, and redesign, the gamespace from the original demo gamespace in order to emphasize the idea of finding oneself and creating one's view of the world. By deleting the initial stage of the demo, re-building and decorating a sci-fi, isolated, platform we were able to completely change the feel of the original demo. We also added a skybox and used in-built engine blueprints to customize the volumetric clouds.

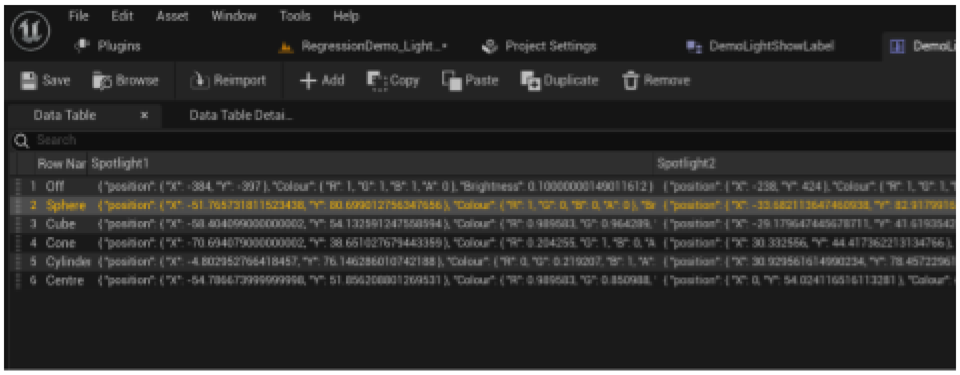
Our training data was quite small, only really including two values being recorded over and over again. Through many trials we found a few important features of data of this size. First of all we found that around 4-6 recordings in each position were ideal when training the model. This allowed a relatively precise description of the space we wanted recorded while keeping the machine from overfitting. We also found that in order to reduce noise and the overlapping of data points the algorithm worked best when we recorded different positions in spots that were as far away from one another as possible. If we had wanted the positions to be close together the amount of recorded examples per position would likely need to increase as a more detailed numeric description of each position would be needed. We, however, did not attempt this.

All these pieces worked together to create a relatively smooth player experience. During presentation day we were able to have a new user test the design and, with a little instruction, it worked great. The new user put on the head set and walked to a corner of the game space of their choosing. Then one of us at the computer set the lights to one of the six states. The new user clicked the record button a few times standing in places close to that position. They then repeated this process a few times. When the new user was done he clicked the train model button and was able to walk around the gamespace and have the model follow his trained movement. The model cycled through the states that were trained to the positions and was able to guess the many positions that were not explicitly trained.

Process

The conceptual framework for Self Locator was there from the beginning. Creating a work of art that discussed the creation of mental maps and sees consciousness as a continual process of self location was one of our first ideas. The parallels to machine learning algorithms and the exploration of the learning process from a meta-analysis perspective was a conscious choice. The idea to use Unreal Engine was also an initial concept being a way to focus our technical workflow investigation, and lean on our team's prior knowledge with the engine. In this way we would be able to create an interactive environment that could be manipulated thus achieving the goal representing this core idea. However, in order to make this into an 'embodied interaction' we decided to use Virtual Reality. Taking a user and having them physically move through the game space seemed like a very straightforward interpretation of the project mandate.

However, with the technologies we chose there was quite a learning curve. For the Virtual Reality aspect we decided to contact Marco Luna Barahona who is the head of the Immersive Storytelling Studio and taught the Narrative VR class at Concordia. He was able to teach us about what to look out for when making a game space that is meant for VR. For the InteractML portion we ended up contacting Phoenix Perry from the InteractML development team and she was able to point us towards relevant documentation that detailed the process of building the plugin's code from an IDE in the unreal template project. The template project is hosted on github and maintained by Sam Swain . We also needed to sit down and learn the basics of Unreal Engine and specifically the BluePrints feature so that we would be able to understand the InteractML code.



	Row Nar	Spotlight1	Spotlight2
1	Off	{"position": {"X": -384, "Y": -397}, "Colour": {"R": 1, "G": 1, "B": 1, "A": 0}, "Brightness": 0.100000001490116121}	{"position": {"X": -238, "Y": 424}, "Colour": {"R": 1, "G": 1,
2	Sphere	{"position": {"X": -51.765731811523438, "Y": 80.699012756347656}, "Colour": {"R": 1, "G": 0, "B": 0, "A": 0}, "Br	{"position": {"X": -33.682113647460938, "Y": 82.9179916
3	Cube	{"position": {"X": -58.404099000000002, "Y": 54.132591247558594}, "Colour": {"R": 0.989583, "G": 0.964289, '	{"position": {"X": -29.179647445678711, "Y": 41.6193542
4	Cone	{"position": {"X": -70.694079000000002, "Y": 38.651027679443359}, "Colour": {"R": 0.204255, "G": 1, "B": 0, "A	{"position": {"X": 30.332556, "Y": 44.417362213134766},
5	Cylinde	{"position": {"X": -4.802952766418457, "Y": 76.146286010742188}, "Colour": {"R": 0, "G": 0.219207, "B": 1, "A":	{"position": {"X": 30.929561614990234, "Y": 78.4572296
6	Centre	{"position": {"X": -54.786673999999998, "Y": 51.856208801269531}, "Colour": {"R": 0.989583, "G": 0.850988, '	{"position": {"X": 0, "Y": 54.024116516113281}, "Colour":

Fig.2. Blueprints from Unreal Engine, written by InteractML, 2022, example of a label table created by InteractML

One of the biggest learning curves came from both of our abilities to understand the InteractML code. This code was lengthy and involved functions that we had never seen before. We had decided to base our project off of a basic regression task demo which included an X and Y input and six different output states to be trained on. Our first idea was to add a new output state to the six initial ones so that we could customize it completely. After looking at the code for a while we learned that InteractML was using an aspect of Unreal Engine called Labels in order to categorize data types and sort out certain values from those data types. Figure 2 shows an example of the label data table and the floats that make up the different light position states. In general a user would categorize the data type of the label using a title such as 2D vector, float, or boolean but interactML had found a way to make their own title called a Parameter Value. This made it much more complicated to create a new parameter. Ideally, we could have spent time trying to figure out how to pick out and categorize our own parameter but we would have needed much more time to have made that work and likely would have needed help from the InteractML team to complete this.

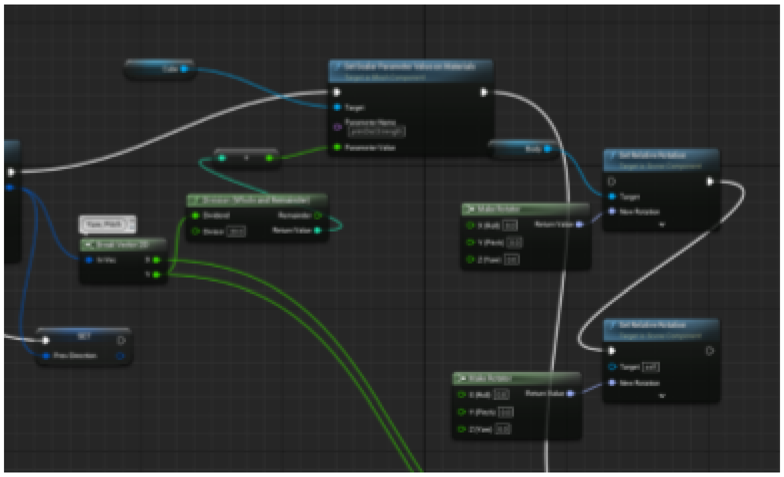


Fig.3. Blueprints from Unreal Engine, written by InteractML, 2022, example of nodes that translate the ending X and Y floats into pitch and yaw

Since we only had a few weeks to get this to work we needed to find a different way to go about it. While looking at the code we realized that the XY data was being run through this entire big series of functions but was being spit out as XY floats. Figure

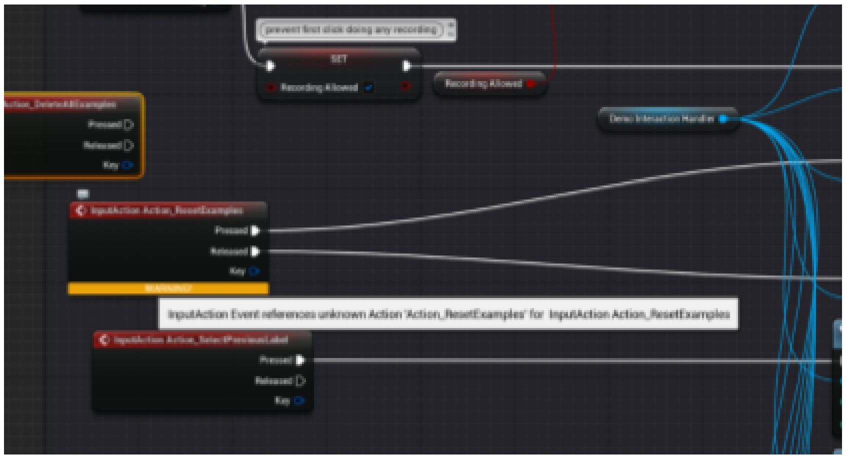
3 shows an example of the preliminary code. These floats were then going through a few more simple functions to change them into pitch and yaw values of the rotation of a spotlight. This gave us the idea to take the end floats and turn them into Translation functions instead of pitch and yaw so that the value would change the XY position of the element instead of rotating it. Then all we had to do was change the look of the spotlight (by manipulating the mesh) so that it would look like a floating orb instead of a spotlight. This ended up working very well as we could use the initial six output states but modify them to have the light move around like we wanted them to.

The only problem with this solution was that we could not change more things about the environment like how we had hoped. One of our goals was to manipulate an animated texture using this machine learning. However we quickly realized that the same method of using the float already generated by the algorithm would work as well. We took the value and plugged it into a texture distortion function (adjusting the float using a math node to work in the same scale as the distortion function ie. dividing 500 by 100 to work with the distortion rate from 1-10) and it worked!

Through this we learned that instead of finding out completely how this incredibly complex algorithm all worked (even though we both really wanted to) we could build on top of it to satisfy our needs in a time crunch.

Another group of problems we faced was in using the Interact ML code in Virtual Reality. We found that when running the code through the Oculus plug in, a few of the functions did not work the same because they were grabbing data from the computer and not from the Oculus.

Our first problem was finding the player's XY position in the game space. After looking at the code for a while we found out that the algorithm was using a node called 'Get Player Position' which would provide the position of the character avatar in the gamespace. While this sounds like it would work we did some digging and realized that we needed a specific node called 'Lock HMD' which would collect the XY data of the Oculus headset specifically. After some tweaking with the code around it this solution worked perfectly.



InputAction Event references unknown Action 'Action_ResetExamples' for InputAction Action_ResetExamples

Fig.4. Blueprints from Unreal Engine, written by InteractML, 2022, certain 'delete recording' commands not being attached to usable interfaces

Some of the controls also did not properly map from computer to Oculus. Namely the 'Delete Recordings' button didn't seem to work when we ran the project in VR. By comparing the first person demo game mode's implementation of the input mapping it was evident that the VR game mode's blueprint was using an action event that no longer existed. Perhaps a leftover product of migrating to the enhanced input system or such other revamp of the input mapping. We switched it for the action event in use on the first person game mode and the issue was fixed. We could now delete recording examples in VR in real time.

We also had to map a few of the controls to the Oculus hand controllers and while we were doing this we realized that one of our initial almost discarded concepts could easily be implemented. By mapping half of the controls to the computer and half to the Oculus we could effectively create a two person performance. One person would be in charge of manipulating the environment and the other would manipulate the input (their position in space).

There were a few other problems that included tweaking a few things or changing around settings. But those were the main technical issues we faced until presentation day.

In order to properly convey the artistic goal of this work we wanted to load the program onto a laptop and perform it in a different space. We had bought a cable that would enable us to do so but when we tried to use it on presentation day the screen was blank. We then had two options: to either present in the original room by streaming the screen with a TV from the VR lab, or to bring everyone upstairs to see the entirety of the performance. We decided that in order to properly show the interaction between two people and two different hardwares we would bring people up to the VR lab despite it being a bit more cramped than the other room. Through this we experienced the need for quick and decisive decision making with a very close time constraint (one hour) and had to decide what was really important to the presentation. We also learned that if we are using any sort of technology that we have not used before- even if it is just a simple cable- we have to test it before presentation day.

Future Work

Future iterations of this project have an endless amount of potential forms. The most interesting and promising aspect of this project was the idea that a user could not only create a game space using their own ideas but create a game space that is completely unique every time. In Mojang's popular sandbox game Minecraft, when a new world is generated all aspects of the world are unique to that single seed generation. The world also generates infinitely, creating new configurations of biomes and architecture using premade blueprints of individual assets. What if a user could generate a world based on their own will without the need to create specific assets while also keeping the element of surprise and exploration due to an inevitable uncertainty?

On a smaller scale (we are no Mojang) we could do a few things to increase the effect of complete world creation. First of all we would really like to learn how to add a new output state to the algorithm. For this the most effective way would be to contact someone at InteractML and get them to explain the process of Label creation in terms of their code. This would be time consuming but the effect would be immense. We would be able to create entirely new and diverse environment states. For instance we could have the user be able to change the lighting colors and positions with more precision, they could move around assets, and they could even change the look of the entire sky. The more states we could add to the output possibilities the more the user could put their own spin on things.

Another very important improvement we could make would be a remapping of the controls. When we had a random person come up and try out the project it was pretty difficult to explain which button did what as it was not very intuitively laid out. If we had more time to think this through and test it we could possibly create a mapping configuration that is more suitable to new ‘players’, making this a readily usable tool for anyone.

Finally to increase immersion yet again adding sound would be incredibly useful. Sound could add many atmospheric effects depending on the type of mood someone was going for. Hypothetically we would be able to match different states to different moods. One state could be a high energy mood with lots of red, sharp shapes, and fast music while another could be a calmer mood with blues, soft shapes, and relaxing music. We could have many of these different states which a player could then use to create entirely new experiences and even stories by training them to different positions.

Primarily the goal would be make the program as user accessible as possible while also making it extremely customizable.

Works Cited

Barahona, Marco Luna. Interview. Conducted by Maya Marshel. November 7th, 2023

Epic Games. “Unreal Engine.” 1998.

InteractML, interactml.com/. Accessed 11 Dec. 2023.

Ismael, Jenann. *The Situated Self*. Oxford University Press, 2009.

Luckey, Palmer. “Oculus App.” 2012.

Perry, Phoenix. Email Interview. Conducted by Leonardo Morales, November 5th, 2023

Polyarc. “Moss.” 27 Feb. 2018.

Tender Claws. “The Under Presents.” 19 Nov. 2019.

Valve. “Steam VR.” Apr. 2016.

Self-locator project On GitHub: https://github.com/lmorv/iml-self-locator

Iml-ue4 sample project on GitHub: https://github.com/Interactml/iml-ue4