HaptiEditor: Haptics Integrated Virtual Terrain Editing Tool

CÉLESTE MARUEJOL*, École de technologie supérieure, Canada
SEAN BOCIRNEA*, University of British Columbia, Vancouver, Canada
RISHAV BANERJEE*, University of British Columbia, Okanagan, Canada

The contemporary landscape of virtual world design is characterized by the ubiquity of diverse tools and terrain design engines, which have significantly reduced the barriers to entry in this domain. Despite this progress, the predominant input modalities of mice and keyboards fail to provide users with haptic feedback during the processes of designing, testing, and experiencing virtual environments. Addressing this limitation, we introduce HaptiEditor, a virtual terrain editing tool that integrates haptic feedback through the utilization of force-feedback capabilities offered by the Haply 2Diy device, implemented within the Unity game engine framework. Our primary objective is to enhance both the design process and the evaluative capacity of designers, as well as to enrich the immersive engagement of users or players navigating these virtual worlds.

Additional Key Words and Phrases: Haptics, Tool Design, Unity, Haply, ForceFeedback, Haptic Texture Rendering

1 INTRODUCTION

Haptics interfaces are already often used for enhancing the sculpting experience, in multiple fields, notably, medical braces. However, there hasn't been any significant implementation of force-feedback interfaces in the case of terrain editing and terrain painting. HaptiEditor is a project intending to explore this application of the haptics, in the hope of seeing wether or not it is promising to invest more effort into this idea.

HaptiEditor is a Unity application which has for objective to edit maps and terrain by painting textures and objects on different scales. By using the Haply 2DIY we hope to create an interesting approach to terrain creation and edition through haptic feedback. The goal is to allow real time editing of a virtual space, and simultaneously feel the effect of the changes right away.

The development period span over a period 3 months during the Winter session of 2024, in the course entitled CanHap501. CanHap501 is a program which covers multiple Canadian universities in the hope of introducing graduate MSc students to haptic interfaces. This course teaches us how to conceptualize, prototype, develop and do user evaluation with multimodal human-computer interfaces and haptic exeperiences. This project is being developed in a team of three, each of us in different location (i.e. Montreal, Okanagan and Vancouver) with the management issues it entails. For instance, we had to deal with timezone issues as Okanagan and Vancouver are on a different timezone than Montreal. Or, the version of the hardware given to each student could be different depending on the node they are coming from.

Since the development timeline of this project is very short we decided to go with a rapid prototyping approach as learned in parallel during this course. Moreover, since we didn't really have enough time to create a fully fledged terrain editor software, we agreed to use Unity to simplify a lot of the designing and implementation to get to experiment with the haptic side of the project quicker. Especially since Unity, as a game engine, already implements some solid systems for collision, forces and texturing.

*All authors contributed equally to this project

Authors' addresses: Céleste Maruejol, École de technologie supérieure, 1100 R. Notre Dame O, Montréal, Canada, celeste.maruejol.1@ens.etsmtl.ca; Sean Bocirnea, University of British Columbia, Vancouver, 2329 West Mall, Vancouver, Canada, seanboc@student.ubc.ca; Rishav Banerjee, University of British Columbia, Okanagan, 3333 University Way, Kelowna, Canada, rishav.banerjee@ubc.ca.

During the development of HaptiEditor, a system to zooming in and out has been implemented. Using a system of sampling existing textures form the surface's material in order to create haptic feedback and Unity's collision system, HaptiEditor is able to provide different haptic feedback depending on the scale of the end effector scale in the scene. This allows a haptic continuum to enable the user to feel the terrain they are editing at any scale they would potentially need.

The present report is a statement of the advancements and findings made during the development of HaptiEditor. It will go over the different avenues tried in order to create HaptiEditor, what was prototyped and how it shaped the current software. Then we will examine the results gathered through semi-formal user testing and the overall appreciation the porject received. The results of the user testing and our implementation will thouroughly be discussed in the Discussions section. There will be an appendix going over the details of the code judged the most important for our software to work.

2 RELATED WORK

Lorem ipsum dolor sit amet, consectetur adipiscing elit. Ut accumsan arcu bibendum purus laoreet rutrum. Integer aliquam arcu gravida dignissim vestibulum. Aenean ut congue purus. Nullam eleifend, justo non bibendum efficitur, odio felis porta ligula, semper commodo massa erat vel tellus. Nullam non nibh vel eros imperdiet vestibulum sit amet a ex. Proin mattis dui at tortor ultricies, id pulvinar nisi euismod. Duis sed massa in dui aliquet hendrerit. Praesent aliquam luctus nisi non lobortis. Sed bibendum magna orci, vitae pharetra nibh mattis ac. Mauris quis lacus vitae est posuere blandit. Vestibulum ante ipsum primis in faucibus orci luctus et ultrices posuere cubilia curae; Sed quis augue id lacus egestas tincidunt. Duis et risus vel justo vestibulum feugiat. Donec neque nisl, luctus vitae leo sit amet, vulputate maximus nunc. Cras et velit ut ex ultrices venenatis eget vel odio. Proin fringilla, ante vel fermentum bibendum, ipsum neque dictum dolor, nec semper urna felis sit amet nibh. [1]

Fusce eget tristique ante. Proin rutrum est eget semper iaculis. Morbi pulvinar urna non dui dapibus faucibus. Suspendisse tincidunt, sapien non luctus varius, justo ante consectetur erat, quis efficitur ante felis ultricies urna. Sed finibus justo a fringilla finibus. Praesent vitae elit ac nisi maximus tristique. In ac orci volutpat, bibendum quam in, tristique dui. Integer eget nulla id nulla ultricies molestie nec id lacus. Donec eleifend rutrum risus et tempor. Morbi lobortis maximus vestibulum.

Fusce ultrices at sapien tristique semper. Ut congue, ex quis ultrices dictum, nunc ligula ornare sem, in pellentesque felis risus in diam. In hac habitasse platea dictumst. Pellentesque ultricies lectus et congue mattis. Integer a ligula non est dapibus mattis eget ut diam. Sed et ex purus. Aliquam pretium, lacus nec molestie tempus, erat magna mollis tortor, quis pulvinar nulla ipsum eu turpis. Sed nibh erat, posuere in dolor ut, accumsan luctus felis. Nam sit amet elit purus. Nulla rhoncus turpis vitae odio viverra commodo pretium nec libero. Lorem ipsum dolor sit amet, consectetur adipiscing elit. Integer in mi quis odio gravida lacinia. Nam massa mauris, tempor consequat felis id, blandit tincidunt nibh. Integer elementum ex vitae nibh fermentum, ac tincidunt libero semper.

3 APPROACH

This project attempts to deliver on a haptics focused terrain editing experience first and foremost. The overaching approach was to first establish a reliable coupling via a virtual proxy to Unity's physics system, and then create a proof of concept terrain editing tool which is driven by said proxy.

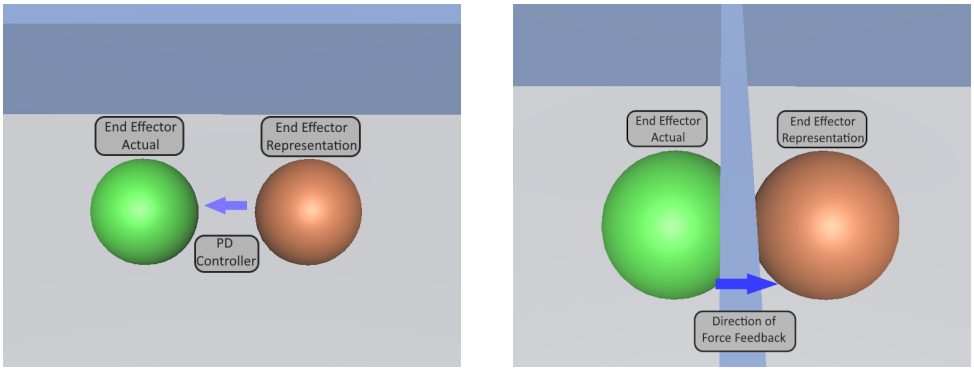
The three fundamental questions we had were as follows:

(1) How would we design a generic virtual coupling between Unity's physics engine and the Haply's force feedback mechanisms?
(2) How would we detect and render textures in realtime?
(3) How would we design the tool itself, with the main mode of interaction being through the Haply?

3.1 How would we design a generic virtual coupling between Unity's physics engine and the Haply's force feedback mechanisms?

We initially employed the Unity template obtained from the Haply GitLab repository as the foundation for our implementation. However, upon closer examination, it became evident that the forces applied were hardcoded, prompting us to adopt a PD controller model [3] facilitated by a virtual proxy (*See 1*), as elaborated below.

In our implementation, we utilized Unity game objects, specifically one designated as the "**End Effector Actual**" to track the ideal positional data of the Haply, and another marked as the "**End Effector Representation**" equipped with a built-in sphere collider. This was to allow it to interact with Unity's physics engine. Subsequently, we established a PD controller relationship between these two entities. The underlying operational logic mandated the "**End Effector Representation**" to consistently attempt to minimize the euclidean distance between itself and the "**End Effector Actual**". This behavior was governed primarily by the proportional component of the PD controller, supplemented by the derivative component to offer additional smoothing (*See 1a*). In instances where the "**Representation**" detected any physical collisions, it directed the Haply to exert a force in the direction of the "**Representation**" from the "**Actual**" (*See 1b*). This would happen in parallel to the distance minimization attempts of the "**Representation**". Consequently, this establishment facilitated an adaptable virtual coupling mechanism, subject to the influence of Unity's physics engine via the intermediary proxy.



(a) PD controller moves Representation to Actual if there is no obstruction

(b) Force Feedback Rendered if physics collision is detected

Fig. 1. Virtual Coupling between Representation and Actual End Effector. Note that the "**End Effector Actual**" was never visualised in the tool itself.

Notably, while the direction vector was three-dimensional, only the X and Z components of this vector were translated to the Haply to render force feedback. The selected axes were simply an artifact of our design decision for editing on a terrain lying in the X-Z plane.

3.2 How would we detect and render textures in realtime?

Tangentially influenced by the research conducted by Li et al. (2010) [2], we opted to leverage the inherent image data embedded within the texture for the application of small directional jitters. The procedural approach to texture rendering works by sampling a three by three pixel window beneath the end effector representation. Subsequently, grayscale conversion is applied to this sampled data, facilitating the extraction of pixel brightness values. The brightest pixels then exert the strongest forces away from itself. This mechanism imparts the perceptual impression of being coerced towards regions of lower luminosity, which can intuitively be mapped to "lower points" in the texture. Critically, this force modulation only manifests during end effector movement, thereby avoiding any undesirable tremors during static user positioning. By recalculating this tug force on a per-frame basis, an appreciable frequency modulation is introduced to the end effector, directly mirroring the texture's visual attributes (*See 2*).

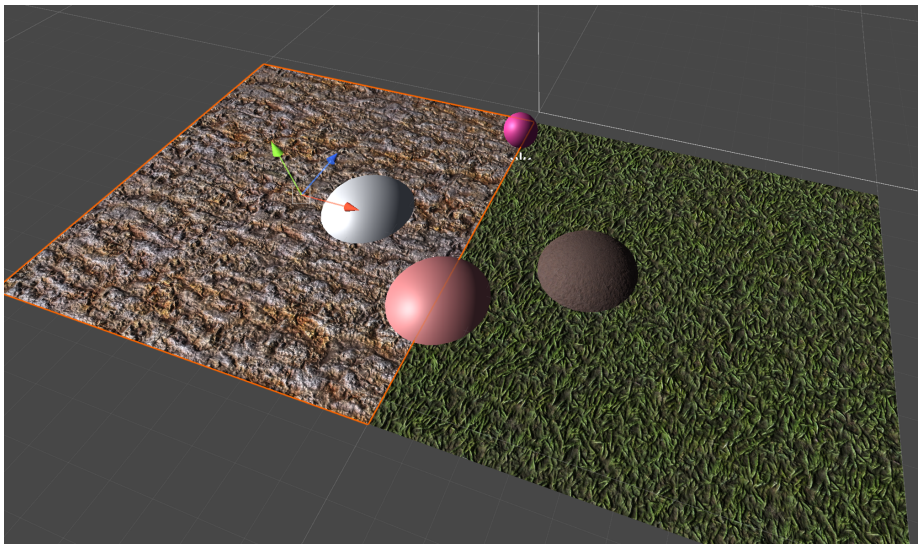


Fig. 2. Two different textures providing different jittery sensations

Although our initial goal was the utilization of heightmap data instead of the texture's pixel values, we found the latter approach yielded satisfactory outcomes during informal tests, prompting its adoption as the preferred methodology.

3.3 How would we design the tool itself, with the main mode of interaction being through the Haply?

Game engines typically offer a range of tools for terrain editing, which are primarily oriented towards editing within the editor environment rather than functioning during runtime. Consequently, it became necessary for us to reconstruct the fundamental components of a terrain editing tool within Unity, essentially embedding them as "game mechanics". This endeavor primarily involved leveraging Unity's Terrain game object [4], which is specially built to optimally encode localised texture and static object placement data. A straightforward user interface facilitates the selection among textures, objects, and an eraser tool, further categorized into sub-menus delineating texture types (e.g., grass, sand) and object variants (e.g., trees, rocks) (refer to Figure 3). The user can then designate their preferred object or texture through mouse interaction. Upon selection, the haply device acts akin to a virtual paintbrush, allowing object placement or texture painting at the location of the end effector. Activation of the painting action is achieved either

through the stylus button integrated with Gen3 DIY devices or, alternatively, by utilizing the spacebar in cases where the stylus button is unavailable.

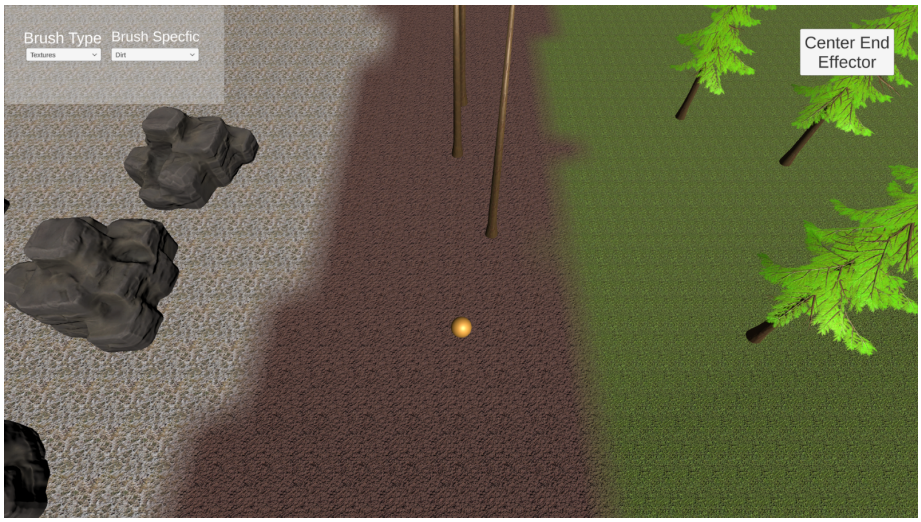


Fig. 3. The final terrain painting interface

The central utility of this approach lies in its capacity to provide users with tactile feedback with their interactions with the terrain in real-time. Objects impart a rigid collider-based resistance as a consequence of the virtual coupling (see Subsection 3.1), while textures provide haptic modulation based on their visual characteristics and behavior in accordance with haptic texture rendering principles (see Subsection 3.2).

4 PROTOTYPING

Lorem ipsum dolor sit amet, consectetur adipiscing elit. Ut accumsan arcu bibendum purus laoreet rutrum. Integer aliquam arcu gravida dignissim vestibulum. Aenean ut congue purus. Nullam eleifend, justo non bibendum efficitur, odio felis porta ligula, semper commodo massa erat vel tellus. Nullam non nibh vel eros imperdiet vestibulum sit amet a ex. Proin mattis dui at tortor ultricies, id pulvinar nisi euismod. Duis sed massa in dui aliquet hendrerit. Praesent aliquam luctus nisi non lobortis. Sed bibendum magna orci, vitae pharetra nibh mattis ac. Mauris quis lacus vitae est posuere blandit. Vestibulum ante ipsum primis in faucibus orci luctus et ultrices posuere cubilia curae; Sed quis augue id lacus egestas tincidunt. Duis et risus vel justo vestibulum feugiat. Donec neque nisl, luctus vitae leo sit amet, vulputate maximus nunc. Cras et velit ut ex ultrices venenatis eget vel odio. Proin fringilla, ante vel fermentum bibendum, ipsum neque dictum dolor, nec semper urna felis sit amet nibh. [1]

Fusce eget tristique ante. Proin rutrum est eget semper iaculis. Morbi pulvinar urna non dui dapibus faucibus. Suspendisse tincidunt, sapien non luctus varius, justo ante consectetur erat, quis efficitur ante felis ultricies urna. Sed finibus justo a fringilla finibus. Praesent vitae elit ac nisi maximus tristique. In ac orci volutpat, bibendum quam in, tristique dui. Integer eget nulla id nulla ultricies molestie nec id lacus. Donec eleifend rutrum risus et tempor. Morbi lobortis maximus vestibulum.

Fusce ultrices at sapien tristique semper. Ut congue, ex quis ultrices dictum, nunc ligula ornare sem, in pellentesque felis risus in diam. In hac habitasse platea dictumst. Pellentesque ultricies lectus et congue mattis. Integer a ligula

non est dapibus mattis eget ut diam. Sed et ex purus. Aliquam pretium, lacus nec molestie tempus, erat magna mollis tortor, quis pulvinar nulla ipsum eu turpis. Sed nibh erat, posuere in dolor ut, accumsan luctus felis. Nam sit amet elit purus. Nulla rhoncus turpis vitae odio viverra commodo pretium nec libero. Lorem ipsum dolor sit amet, consectetur adipiscing elit. Integer in mi quis odio gravida lacinia. Nam massa mauris, tempor consequat felis id, blandit tincidunt nibh. Integer elementum ex vitae nibh fermentum, ac tincidunt libero semper.

5 EVALUATION AND RESULTS

Lorem ipsum dolor sit amet, consectetur adipiscing elit. Ut accumsan arcu bibendum purus laoreet rutrum. Integer aliquam arcu gravida dignissim vestibulum. Aenean ut congue purus. Nullam eleifend, justo non bibendum efficitur, odio felis porta ligula, semper commodo massa erat vel tellus. Nullam non nibh vel eros imperdiet vestibulum sit amet a ex. Proin mattis dui at tortor ultricies, id pulvinar nisi euismod. Duis sed massa in dui aliquet hendrerit. Praesent aliquam luctus nisi non lobortis. Sed bibendum magna orci, vitae pharetra nibh mattis ac. Mauris quis lacus vitae est posuere blandit. Vestibulum ante ipsum primis in faucibus orci luctus et ultrices posuere cubilia curae; Sed quis augue id lacus egestas tincidunt. Duis et risus vel justo vestibulum feugiat. Donec neque nisl, luctus vitae leo sit amet, vulputate maximus nunc. Cras et velit ut ex ultrices venenatis eget vel odio. Proin fringilla, ante vel fermentum bibendum, ipsum neque dictum dolor, nec semper urna felis sit amet nibh. [1]

Fusce eget tristique ante. Proin rutrum est eget semper iaculis. Morbi pulvinar urna non dui dapibus faucibus. Suspendisse tincidunt, sapien non luctus varius, justo ante consectetur erat, quis efficitur ante felis ultricies urna. Sed finibus justo a fringilla finibus. Praesent vitae elit ac nisi maximus tristique. In ac orci volutpat, bibendum quam in, tristique dui. Integer eget nulla id nulla ultricies molestie nec id lacus. Donec eleifend rutrum risus et tempor. Morbi lobortis maximus vestibulum.

Fusce ultrices at sapien tristique semper. Ut congue, ex quis ultrices dictum, nunc ligula ornare sem, in pellentesque felis risus in diam. In hac habitasse platea dictumst. Pellentesque ultricies lectus et congue mattis. Integer a ligula non est dapibus mattis eget ut diam. Sed et ex purus. Aliquam pretium, lacus nec molestie tempus, erat magna mollis tortor, quis pulvinar nulla ipsum eu turpis. Sed nibh erat, posuere in dolor ut, accumsan luctus felis. Nam sit amet elit purus. Nulla rhoncus turpis vitae odio viverra commodo pretium nec libero. Lorem ipsum dolor sit amet, consectetur adipiscing elit. Integer in mi quis odio gravida lacinia. Nam massa mauris, tempor consequat felis id, blandit tincidunt nibh. Integer elementum ex vitae nibh fermentum, ac tincidunt libero semper.

6 DISCUSSIONS AND LIMITATIONS

6.1 Discussions

There are 2 major points of discussion to be had with regards to this project:

6.1.1 The Haptic-Editing process: (TODO: Put information here about the feedback we have recieved and how that improves the experience of editing virtual spaces)

6.1.2 Unity as a tool for haptics design: Throughout history, various art forms, such as music, drawing, photography, film, video editing, game development, and XR development, have experienced significant advancements whenever developers have embraced user-friendly tools. These advancements have been facilitated by the availability of accessible cameras, freely available software for music composition and video editing, as well as widely supported game engines like Unity and Unreal. However, when the primary means of entry into these fields is limited to Java and Processing code,

the potential for designing experiences becomes constrained by the technical skills of developers, creating a bottleneck effect. It is imperative to encourage the haptics community to explore beyond mere code frameworks and instead adopt a singular, user-friendly engine (such as Unity) that can empower designers to focus on the user experience aspect of haptics, rather than being bogged down by technical jargon. Such an approach would enable individuals to specialize in various aspects of haptics, be it hardware, software, or design, akin to the specialization seen in the game development industry, thereby enhancing the overall quality of output.

6.2 Limitations

There are quite a few limitations with our final project, given how many unique concepts and implementations we tried to explore. Some of the primary ones are discussed below:

6.2.1 Movement in an infinite space. With our current implementation, the end effector can only move in the virtual space a proportional amount to the haply device's arm length. While we can change the movement scale in the engine, the end effector will eventually get stuck due to the haply's arm pantograph constraint. The primary difficulty lies in changing the relative positioning of the proxy with respect to the world, and what the underlying user experience design philosophy should be for the same without disorienting the user.

6.2.2 Fine grain control of placed objects.

7 ACKNOWLEDGMENTS

blah blah

```
\begin{acks}
...
\end{acks}
```

ACKNOWLEDGMENTS

To bob.

REFERENCES

[1] Roberta L Klatzky, Dianne Pawluk, and Angelika Peer. 2013. Haptic perception of material properties and implications for applications. *Proc. IEEE* 101, 9 (2013), 2081–2092.

[2] Jialu Li, Aiguo Song, and Xiaorui Zhang. 2010. Image-based haptic texture rendering. In *Proceedings of the 9th ACM SIGGRAPH Conference on Virtual-Reality Continuum and its Applications in Industry*. 237–242.

[3] MathWorks. 2011. What is PID Control? — mathworks.com. https://www.mathworks.com/discovery/pid-control.html. [Accessed 18-04-2024].

[4] Unity Technologies. 2014. Unity - Manual: Terrain — docs.unity3d.com. https://docs.unity3d.com/Manual/script-Terrain.html. [Accessed 18-04-2024].

A INDIVIDUAL CONTRIBUTIONS

A.1 Rishav's Contributions

For the final iteration, we wanted to build an executable that anyone with a haply 2diy board can plug and play. Expecting everyone to install Unity would be quite an ask, so we needed to build a live board configurator.

The live board configurator would need to modify the following parameters:

- Board Presets

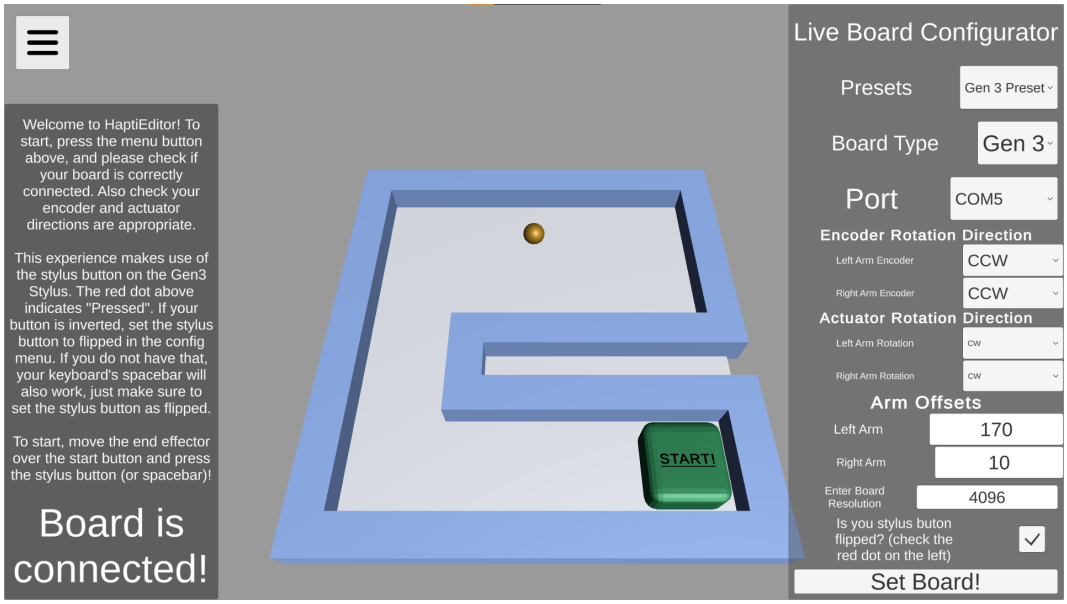


Fig. 4. HaptiEditor configuration menu

- Gen2 or Gen3 (for arm distance offset)
- Encoder rotation direction
- Actuator rotation direction
- Arm offsets for base position
- Encoder resolution
- Flipped Stylus Button (Some Gen3 boards have flipped sensor data for the stylus port)

Actually passing the appropriate data and reloading the board was significantly more difficult. In a nutshell, I had to do the following:

- Cancel the worker thread simulation task gracefully
- Flush all forces
- Delete the existing instance of the board (along with the encoder, actuator and sensor parameters)
- Create a new board instance with the new parameters
- Attempt a connection to the new specified port based on the user's selection from current active ports
- Launch a new worker thread.
- Potentially connect to a Button Handler if the scene had one present.

This required a significant amount of refactoring of the core hAPI, but in the end I was successfully able to load and reload new user specified board configurations. The main chunk of this was happening in the `EndEffectorManager.cs` as follows:

```
public void ReloadBoard(DeviceConfig customConfig, string targetPort)
{
    // Destroying existing setup
    haplyBoard.DestroyBoard(); // added function to close port and destroy this board
```

```
    CancelSimulation();
    SetForces(0f, 0f);
    Destroy(pantograph.gameObject.GetComponent<Device>());
    // New Setup
    device = pantograph.gameObject.AddComponent<Device>();
    device.Init(); // re-establishes basic connections with pantograph and board
    LoadBoard(customConfig, targetPort);
    // Checking for button handler
    ButtonHandler buttonHandler = gameObject.GetComponent<ButtonHandler>();
    if (buttonHandler == null) return;
    buttonHandler.SetButtonState(customConfig.FlippedStylusButton);
}

private void LoadBoard(DeviceConfig customConfig = null, string targetPort = null)
{
    device.LoadConfig(customConfig);   // loads new config if custom config is not null
    haplyBoard.Initialize(targetPort); // attempts connection with new port
    device.DeviceSetParameters();
    // ...
    simulationLoopTask = new Task( SimulationLoop );
    simulationLoopTask.Start();
}
```

After this, I decided to improve the visual experience of the project. Users will be expected to understand that they have to configure their board, and that their board might be set up different to our dev setups, so the menu scene should ideally be different from the actual terrain editing scene. Additionally, the stylus button (or space bar) is critical to the experience, so the users should be aware of how to do that as well.

To let the user learn this separately, I worked on a simple menu scene with a bit of flair, and added scene transitions. Users will now be expected to first configure their board, be able to move over a physical button in the world, and then click on the stylus button to enter the terrain painter tool.

I fished out a basic scene manager and transition handler from an older project, and after some tweaking, blender modelling and building an executable for Windows, I produced the menu scene in Figure 4.

A.2 Sean's Contributions

This iteration, I finalized work on the texture pipeline and integrated it with Pierre's work from iteration 2 on texture and object painting on the terrain object. Pierre wrote some logic to get world position into a corresponding position on the surface of the terrain object, removing the need for our expensive physics ray-cast. The rest of the process involved the following changes:

- Detecting which terrain texture was currently painted onto the surface of the terrain underneath the end effector. This required fetching the blend ratios of each material at the EE location and determining which was present:

```
float[,,] swatch = terrain.terrainData.GetAlphamaps((int)pixelUV.x, (int)pixelUV.y, 1,1);
```

- Once I have the ID of the texture to sample, we fetch color from its normal texture, giving us both a direction and intensity of force we immediately apply to our end effector. This greatly simplified the sampling code:

```
Color normalPixel = prot.normalMap.GetPixel((int)normalUV.x, (int)normalUV.y);

forces.x += normalPixel.r - 0.5f
forces.y += normalPixel.g - 0.5f;

```

```
6   forces *= intensity;
7   previousPosition = eeTransform.position;
```

- I introduced a variable `swatchscale` to allow us to control the ratio of world movement relative to movement on the normal map, controlling the linear density of our texture. Normal sampling also greatly improved the accuracy of forces, as we're no longer estimating heightmaps from surface color but now have geometry-accurate normal maps.
- Selected texturally distinct normal maps for our materials for a clearer distinction between painted materials.
- Added back space-bar support for painting, so a user has an alternative to the stylus button.

I then tuned the scaling code Rishav worked on in iteration 2 and added it to our scene. The following changed:

- I added a scale factor for the movement range of the end effector, so it expanded as the scene zoomed out, covering the new area.
- I then tuned the ratios for movement range, end effector size and camera zooming so that they scaled in tandem.
- I changed the painter code so that the painted objects had their appropriate colliders, allowing the large end effector to skate over rocks as we'd intended it to, rather than getting stuck like before.

Lastly, I tuned the texture sampling code again so that it would play nice with the zooming feature we'd introduced. I chose a `swatchscale` such that at high zoom levels, individual surface features like pebbles were quite large and discernible, but with a wider camera, surface features became higher frequency noise and force feedback from geometry became the primary force on the end effector. Textures representations were different depending on the painted texture on the terrain and could be rendered in tandem with terrain objects placed during use. We learned that these pipelines could all coexist in a cohesive way and were pleased to see our parallel approach to developing them paid off.

A.3 Pierre's Contributions

As stated in our blog posts for iteration 2, the goal for the third was to merge every concepts and prototypes into one single, preferably enjoyable, experience. In this iteration we ended up working together way more and in closer collaboration by helping each others a lot more. This can be easily explained because we didn't prototype on a different aspect of the experience and were actually making something unique together. This is why sometimes the lines of contributions of what I did and what a team member did will blur into what we did this feature together.

While we knew that we would merge all of our progress into the Terrain scene because the end goal is to use Unity's Terrain game object has our editor, the `TerrainScript.cs` from iteration 2 wasn't usable as is. Firstly, we need to fix the lack optimizations. Secondly, the user should have a way to change the brush type and their specifics without using Unity editor menus (it is necessary if the user interacts with our software through an executable instead of the Unity Project). Thirdly, Lastly, it is unlikely the code we used for texture sampling for haptic feedback would work on the Terrain game object, because it has the particularity to not have a `MeshRenderer` (a component that allows a game object to render a mesh). Finally, there needs to be a way paint using the stylus button instead of the mouse.

During this iteration Rishav did an amazing work at going over the code that has been made and telling us what should be avoided in the future. Following those practices we started a refactoring on `TerrainScript.cs`. During this time, I found a way to optimize the management of the `TreeInstances` what were giving us issues during the second iteration, basically, deleted `TreeInstances` would never really be deleted but replaced with empty version of themselves.

This is problematic because with the core logic of the problem they would be given another collider the next time the software is running even though there is nothing to delete.

```
private void OnApplicationQuit()
{
    //test
    List<TreeInstance> trees = new List<TreeInstance>(terrain.terrainData.treeInstances);
    List<TreeInstance> trees_cleaned = new List<TreeInstance>();
    TreeInstance empty_tree = new TreeInstance();
    for (int i = 0; i < trees.Count; i++)
    {
        if (!trees[i].Equals(empty_tree))
            trees_cleaned.Add(trees[i]);
    }
    terrain.terrainData.SetTreeInstances(trees_cleaned.ToArray(), true);
}
```

Using this code when the application is closed we remove all of the `TreeInstances` that we could consider empty. The way it works is by going through all the objects in the Terrain `TreeInstances` and comparing it with an empty object. If they are different we add them to the new list that will contain all the non-empty `TreeInstance`.

Then using the `ObjectPlacer.cs` as a reference I created two co-routines one for painting object on the terrain and the other for painting texture. We implemented an enumerator containing all the brushes types (i.e. Texture, Object and Object Eraser) and depending on this brush type one of the co-routine or the object deletion will be enabled.

From then on Rishav worked with me on a basic UI so we can change the brush types and what texture/object is being painted.