CHILI
computer-human interaction
in learning and instruction

LEARN
Center for Learning Sciences

Design a human interaction activity with *Cellulo-Moris*

Author:
Jean-Etienne Charbonnet

Supervisor(s):
Hala Khodr (PhD student),
Kevin Holdcroft (PhD student),
Barbara Bruno(PostDoc)

Professor:
Pierre Dillenbourg

January 7, 2022

Contents

1 Introduction

The purpose of this work was to design a learning activity on emergence using *Cellulo*s and *Mori*s, robots either developed at CHILI[1] or RRL[2]. The **emergence** is the ability of multiple elements to develop a property or an order that they don't have on their own (e.g. the organisation of an ant colony or arrangement of atoms in a solid).

Altogether, we seeked to create a simulation which lets the user discover what the emergence is by using our robots: to have a common goal that you can only achieve by joint efforts of all robots or users which are taking part in the simulation.

This project and the project of Antoine Clivaz were done simultaneously and are part of the *Grass Root project*, as it will be explained in the section below.

2 General background and expectations

First of all, let us recall what was at stake and why this project was born. According to the **Grass Root proposal**,[3] the main goal of this research is to provide a validation on a theory based on the idea of Piaget, Montessori and Frobel that hands-on and physical manipulation have lots of benefits in learning and education.

Using *Mori*s, that are self-reconfigurable modular robots (SRMR) and the educational *Cellulo* robots, we would be able to create a learning activity that allows users to perform local actions by using the *Cellulo*s and concurrently see their global effects through the *Mori*s 3D topologies, thus grasping the concept of emergent behaviors.

Before going further, here are representations of the robots that we will use for this work. From left to right: a *Cellulo*, a *Mori* and a *Cellulo-Mori* (CM). The *Cellulo-Mori* is constituted of a *Cellulo* as a base, which gives us some abilities. For instance: moving, communicating with others and displaying colors on LED's. A *Mori* is attached on top of the *Cellulo-Mori*, which gives us the ability to connect with other robots (*Mori*s and *Cellulo-Mori*s).
Each robot has specific properties that bring a lot to the activity: *Cellulo*s were designed as robots for education and learning, thus have several components that can be used to make visible and tangible what is invisible and intangible. This provides us with haptic feedback to interact with the user, motors allowing us to move the robot in a plane, LED's for visual feedback, etc.

[1]Computer-Human Interaction in Learning and Instruction: `https://www.epfl.ch/labs/chili/fr/chili/`

[2]Reconfigurable Robotics Lab: `https://www.epfl.ch/labs/rrl/`

[3]Hala Khodr et al. *Cellulo-Mori: Reconfigurable Modular Swarm Robots.* URL: `https://github.com/DeepGreen1/cellulo-mori/blob/main/Modular%20Cellulo%20-%20Grass%20Root%20Proposal.pdf`. (accessed: 2021).

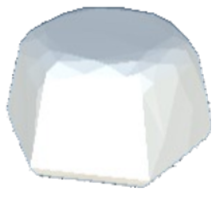


Figure 1: Cellulo

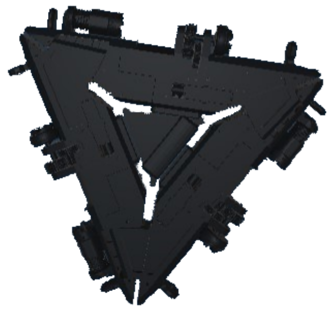


Figure 2: Mori

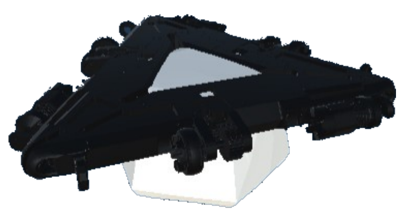


Figure 3: Cellulo-Mori

On the other hand, *Mori*s are *SRMR* robots i.e. modular robots that can take different shapes by connecting with others. This allows us to interact with the 3D world and brings a new dynamic to the activity as the *Cellulo*s were bound to only act in a horizontal plane.

A combination of those robots provides us with a really interesting tool for hands-on and physical manipulations. Secondly, recall that this project is part of another one: the Grass Root project. What were our goals and expectations?

The *Grass Root* project had three main goals: redesign the *Cellulo* robots for modularity; integrate *Mori* robot as an interaction layer; design and validate an activity on emergent behaviours that relies on *Cellulo* and *Cellulo-Mori* robots.
In other words, this work and the one of Antoine Clivaz are bound together and take place as the third goal of the Grass Root project.

Our objectives were hence to find an idea of a game that meets the third goal's objective and to implement it.

3 Game realisation

3.1 Game Design

In order to create the activity, we had to define more clearly the goals of our projects. We had to think about a game that aimed to couple **physical objects** with which you can interact, that can change in shape and all the possibilities that a **digital simulation** offers. We thought of different games that could make use of our robots to highlight emergent behaviors and teach this concept to the user. Once we settled for a game and we agreed on a typical scene, it became much more clear which features had to be implemented or not. Screenshots of the game can be found in appendix and Antoine's work speaks in details about it.

3.2 Development steps

We split the work in several steps that were done almost linearly during the weeks of development. This offered us a global view on the project and helped us to coordinate our efforts as we were two people to be working on it.

- Learn to use Unity
- Think about the game: game creation
- Update the physics engine and implement some user interface (UI)
- Implement features for the game
- Have a preliminary evaluation on learning and gain user experience (UX)

The following sections will explain in more details each big topic encountered during the development of this project.

4 Getting started with Unity and game creation

For this project, we used the Unity game engine. This choice was made by our supervisors as they were already using software on Unity to link a digital simulation containing *Cellulo*s with real robots. Doing so, we can use the simulation to test the feasibility of the activity before trying to do it in a physical world.

Before going further and for vocabulary concerns, here are some concepts of Unity

- An *activity* is another word for the game. In our case, it's a game which aims to teach the emergent behavior concept to the user.
- A *scene* is another word for a level of the game.
- A *gameobject* is the base entity that can be used in a scene. All components inherit from it and can thus be considered generically as a gameobject.
- A gameobject can contain several *components* that are fundamental pieces of every gameobject. Each component will give different functionality to the gameobject. *Box Collider*, *RigidBody*, *NavMeshAgent*, *NavMeshObstacle* are examples of such components that you will see below.
- A *prefab* is a system of Unity that allows you to create and store an object of the game with all its component.
- In Unity, we use C# scripts in order to create new components and hence new functionalities for gameobjects.

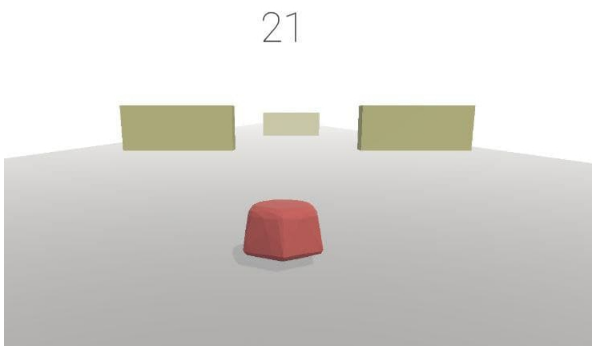


Figure 4: First game inspired by Brackeys

We were provided with some C# scripts and an example scene. This was really useful to start as the engine is really powerful but not so intuitive at a first glance. I began by creating a small racing game using *Cellulo*s to grasp the main concepts of Unity (inspired by a YouTube video[4]) that you can see in figure 4. I also played with the provided scene. In parallel, we thought about the activity and found different ideas:

- Use robots as a slide for a small ball, this game can be cooperative.
- Create a version of the *game of the wolf, the goat and cabbage*[5] with our robots. One person controls a *Cellulo-Mori* as a boat and the other can connect the actors represented as *Mori*s (wolf, goat, cabbage) to the boat in order to make them cross the river.
- Place a robot in a maze and use another one (that you can move with your hand) to move the first robot, that one would move in mirror direction.
- Simulate an egg race between multiple players. A *Cellulo-Mori* connected to a *Mori* can act as the spoon, on which we place an object that must not fall on the floor.

From those ideas, we kept the elements that we thought the most interesting for the activity: use the connections between *Cellulo-Mori*s; have a human driven control on the robots; use collaboration to overcome a problem. This last point can be seen in multiple ways: either the control of the robot is split in different roles (e.g. one user can move the robot, the other can change the orientation of the joints, thus having collaboration within the human driven control) or either the need of different robots to act together to achieve the goal (e.g. A *Cellulo-Mori* has to give a *Mori* to another *Cellulo-Mori* that can only rotate but not move).

With that in mind, we could start to create a scene for the activity. Antoine made a first level and we worked on it from then on.

[4]`https://www.youtube.com/watch?v=IlKaB1etrik&list=PLPV2KyIb3jR53Jce9hP7G5xC4O9AgnOuL`
[5]`https://en.wikipedia.org/wiki/Wolf,_goat_and_cabbage_problem`

5 Physics & MuJoCo

Physics In the first weeks, we had to split the work between *physics* and *UI* (User Interface) so that we did not get in the way of one another. I'll focus this part on the physics as I was assigned this part[6]. The goal of this part was to have a simulation that is as close to reality as possible. Recall that we use the simulation to test if the activity is feasible before creating it with real robots. Thus, it has to mimic reality.

I noted several improvements that could be done: the relative weights of the robots were not the same as the ones in real life; the friction of the floor was not correctly set; the robots were tilting when moving around; a gap was created between connected robots. I corrected those issues by changing the masses and checking with videos of the real robots in order to have a similar behavior. Adding a *Material* in the collider of the floor parts let me add and adjust the coefficient of friction. Adapting the scripts to change the way in which robots were moving was also a key element: previously I was only changing the velocity and not adding a force in the desired direction of movement. This had a bad effect on the physics as changing velocity does not take the masses into account.
I tried different kinds of joints to deal with the gap between connected robots (hinge joint, fixed joint and thought of using a configurable joint, but didn't go too far with it as this is too specialised compared to what we wanted to do). The best option to correct the problem was actually to keep the hinge joint but to change its parameters.

Playing with connections raised several questions concerning groups of robots: how should a collection of two or more connected robots move and rotate? Also, with a lot of robots connected, loops could appear: we should detect it so that the global behavior can change accordingly. Finally, we need some UI elements to control the connections: select which one to move up or down, which one to delete.

This led me to add features to display connected elements of a *Mori*, to add a UI capsule on each connection so that we can work with it and to write a Python script that detects and lists all cycles among connected robots. The behavior of a collection of robots is still an open question and I'll speak more about it in the future ideas for this project[7]. See the UI in figure 5: select the joint by a mouse click on the blue/pink cylinder, then play with it.

MuJoCo Halfway into the development of the activity, an advanced physics engine that was acquired by *DeepMind* became freely available as it's now open source: **MuJoCo**[8]. A physics engine enables objects to approximate universal forces in nature such as gravity, velocity, acceleration, and friction. The question to use it for our project emerged: would it be worth changing the current scripts and simulation to use MuJoCo, in order to have

[6]See Antoine's work for more on the *UI*.
[7]For now, a *Cellulo-Mori* connected to at most one another *Cellulo-Mori* can not move.
[8]`https://mujoco.org/`

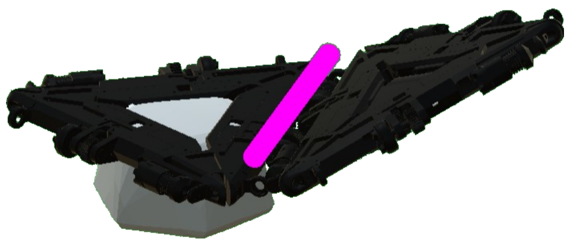


Figure 5: Connection UI (in pink)

more realistic physics?

Sadly, as the engine was recently bought, all the documentation required to use it was currently removed from the internet. Adding it in Unity for our activity was thus a complicated task. I however managed to download a working version and Unity plugin file, in order to test it.
After having played with the new physics engine, I could see that it's a powerful one but that it will not provide us with enough advantages to counter the time it would take us to re-implement the scripts and find the documentation needed to use it. The main reason behind this decision is that we are mainly using joints between separated objects and that MuJoCo focuses more on joints as the Unity configurable joint, thus providing nothing more as the default Unity engine but only a bigger overhead.
We chose to keep the project as it was and not to use MuJoCo. It can however be a good idea to sometimes keep it in mind for the project.

6 Implemented features

With working physics, the idea of the activity in mind and some UI to interact with the simulation, we thought of different features that will help the user and make the playing time (and development) more enjoyable: a feature to **align robots in an autonomous way** as the detection of two robots connecting was really delicate; a feature to play almost without the keyboard that we called **self-motion**: set the destination and the robot will avoid obstacles to go there, following the shortest path; a feature that extends the self-motion feature called **self-connection**: select the side on the target robot and the current selected robot will connect with it on the chosen side. Finally, I had to reprogram the scripts for **manual connection** as the way to connect robots and to move robots has changed due to the previous features.

Before going in details for those features, let's speak about a difficulty induced by Unity: the development of the features took more time than expected especially due to the different kind of robots and ways of using it. When developing a feature, I chose to test extensively on *Cellulo-Mori*s as they are the more complex one. When the features were working on them, I had to rethink about the process and integrate particular behaviors for the *Mori*s, due to the difference in their constitution in Unity (the elements of the

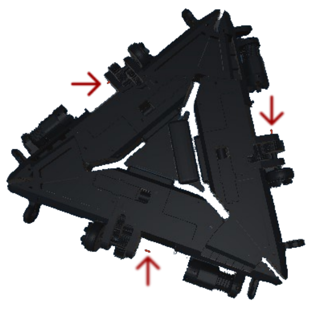


Figure 6: The colliders that trigger a connection (in red)

Prefab are different, so we have to work in a different way).

In the following sections, I'll speak in more depth about the choices of implementation that I had to make in order to develop the wanted features.

6.1 Self-alignment

As explained above, connecting two robots was not an easy task. The colliders that trigger a connection were small and we had to be at a very precise spot to initiate the connection, see figure 6. We thought that having it autonomous would make everything much easier. For that, I had to modify the Prefab of the *Mori* and add some *detection pads* for each side of the robot. By writing different scripts and by using the trigger property of those detection pads (that are actually box colliders), it was possible to detect which robot was in range of which side to initiate a connection. With this feature enabled, passing close enough to another robot (i.e. on it's detection pad) will start the connection. To prevent this from happening unexpectedly, I added some UI to enable or disable it. The detection pads are presented in figure 8.

Having detection pads was a good choice for its reliability compared to the Raycasts and for an indirect feature that it provides: we can simulate having a non-working side of a robot. For that, we only have to disable the box-collider so that no robot can connect with it.

You can see a *Cellulo-Mori* connecting to another one at the computed position (on a spot of the red line) in figure 7.

Concerning the algorithm, here is some insight of its behavior. First of all, let us define some terms: we will call *robot* the moving robot and *other robot* the one to which we want to connect. The algorithm follows the following workflow:

1. The robot collides with a detection pad which triggers the execution of a script. User inputs are disabled and the robot will go through three stages: move to a safe distance of the other robot; rotate to be parallel to the side of the other robot,

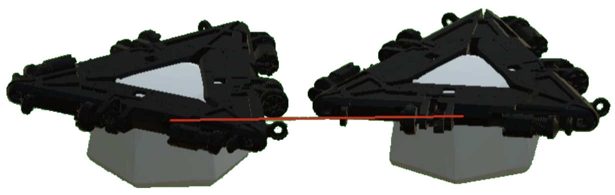


Figure 7: Two CM aligning and connecting automatically

accordingly to the detection pad; get close to the other robot until reaching a threshold.
Note that with the manual feature, none of those stages are used and the connection is triggered with the colliders presented in figure 6. More on that below.

2. *Move to a safe distance*: The seeked position is computed with the position of the other robot. Depending on the detection pad we triggered, we know in which direction we want to place the robot, relatively to the other robot. The wanted position is computed with $other\_position + offset * direction$.

3. *Rotation*: Here, we have to compute the orientation in which we want to set the robot. As *Mori*s are equilateral triangles, there is three rotations that results in a similar orientation namely rotations of: 60, 180 and 300 degrees. With respect to the position of the other robot, we have to select the smallest difference of angle between those possibilities. Then, the sense of rotation matters as it's useless to make a full rotation. Thus, we have to select the rotation that results in the smallest angle between the orientation of the other robot and the orientation chosen in the previous step.

4. *Reaching the threshold*: When the robot is at the safe position and parallel to the other robot, it'll move towards the center of the *Mori* of the other robot. When reaching a threshold that corresponds to the minimal distance between the two robots, another script is called in order to create the connection. The choice not to use the colliders of figure 6 was made because of the problems with the mesh of the *Mori*s (see below). As the colliders are very small, they would not be triggered if the robot was not exactly at the right place.

Main problems encountered

- Raycasts: before doing it this way with box colliders, I tried to use Raycasts. This version was interesting as I never worked with it before, but was more bug prone than box colliders as rays can sometimes pass through a collider without triggering it. (e.g. if the source of the ray is partially inside of it)

- Relative position of elements inside of a prefab: I had to compute the final position of the connection in order to add the HinjeJoint (and later on, to add the connection

UI). This was not straightforward as the positions given for elements inside of a prefab are not given in absolute position at run-time. I had to find some workaround.

- Quaternions: I wanted to make the automated alignment as quick as possible, i.e. choose the best sense of rotation and the shortest angle to rotate. For that, I had to work with angles in Unity that are represented in Quaternion. This was also my first time.
- The Mesh on their own: Mesh of the *Mori*s were not centered, so we had to add an offset to recompute the correct center during run-time, until being provided with a good version of the mesh. Without this correction, the robots were drifting when rotating and the colliders of connection were not triggered as explained above.
- Get the correct side: when connecting or disconnecting, we kept track of the connected side of each robot. At first, we did not store the right one, resulting in a undefined state of the simulation. Correcting this took more time than expected due to the shared portions of code between robots.

6.2 Self-motion & NavMeshAgent

To implement the self-motion feature, I had to change the Prefab once again and add a NavMeshAgent along with a NavMeshObstacle. This forced me to change the way in which robots move: I can no longer add forces on the robot but must use the methods provided with the agent. This had a bad impact on the physics at first but was corrected with some adjustments on the parameters of the agent and obstacle.

The NavMeshAgent is a component of each *Cellulo-Mori* and NavMeshObstacle of each *Mori*. Both are not part of *Cellulo-Mori* as we use *Mori*s alone in the activity (the self-motion feature has to avoid colliding with them).

Main problems encountered

- An object cannot have a NavMeshAgent and NavMeshObstacle enabled at the same time: when selecting a robot, I had to disable its obstacle so that the agents can work properly.
- Avoiding connected robots: when connected together, I could not re-activate the NavMeshObstacle of the robots as they would repel each other. Consequently, when another robot is moving in the simulation, if we target a position that is behind the group of robots, it wouldn't consider it as an obstacle and would try to go through the group. This issue should be handled with the problem pointed out before with connected robots, concerning the center of rotation of the group or the way that they would move. We decided to keep this for the future of the project and focused on the development of the other features and on the rest of the game.
  By being aware of this problem, we can move around the group of robots by targeting a position that goes by the side of the group before targeting the final seeked position.

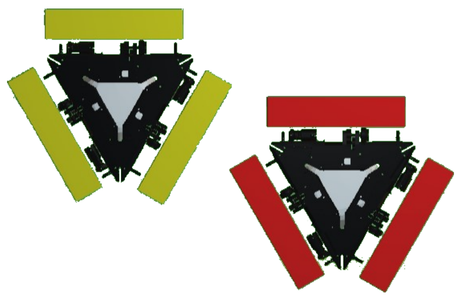


Figure 8: Feature displaying the detection pads

- NavMeshAgent causes stuttering: as we were updating the RigidBody when moving around, using an agent was inducing a stuttering when the RigidBody and agent did not agreed on the positions. This problem was solved by using the agent's position to move our robots, even when the feature was disabled.

6.3 Self-connection

As explained before, this feature is an extension of the two previous ones: we use the self-motion feature to move the robot to the selected detection pad, which triggers the self-alignment feature (Note that we have to disable the obstacle before trying to connect).

Main problem encountered As we are using the self-motion feature, we are suffering from the same constraint as the one explained above: if the path of the targeted detection pad goes through a group of connected robots, we have to move our robot before initiating the connection.

6.4 Manual connection

Adding all the features changed the way in which we are connecting the robots. More precisely, adding NavMeshObstacles prevents us from being close enough from the triggers that would normally start a connection. For a manual connection, we thus need to specify to which robot we want to connect in order to disable the obstacles. For that, I added a UI feature that displays the detection pads. They are in *red* if the obstacle is disabled and in *yellow* if it's enabled, as displayed in figure 8.

Hence, to connect a robot manually you need to disable the obstacle of the other robot and then move close enough to trigger the connection: place yourself exactly where you want the robot to be connected. As said before, we added the self-alignment feature because it was delicate to perform such a procedure. I changed the Prefab so that the detection is more sensitive, with bigger and well placed Capsule Colliders around the *Moris*. Recall the figure 6. This manual connection feature only uses the colliders to initiate a connection, contrary to the self-alignment feature presented above.

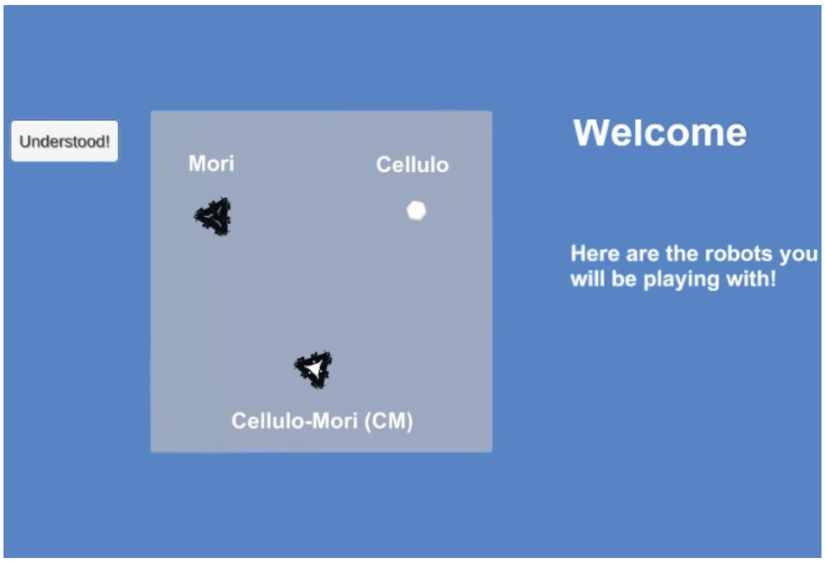


Figure 9: A preview of the tutorial

7 Preliminary evaluation and future steps

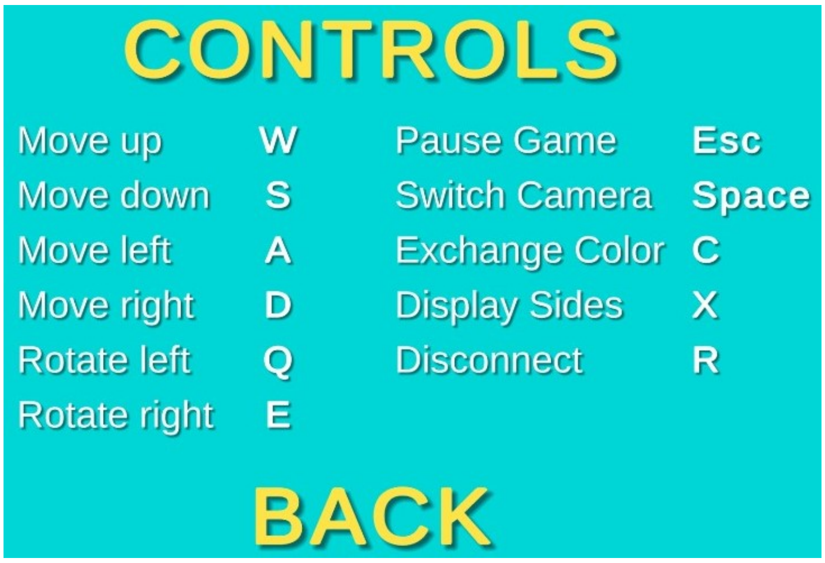
The third goal of the *Grass Root Project* was stated as *design and preliminary validate a learning activity on emergent behaviours that relies on Cellulo and Cellulo-Mori robots [...]*.[9] Testing the current version of the activity by having user testing is a good way of getting a direct return on our work. We did not have time for extensive tests but this already gave us a direction for the activity. In consequence, I created a tutorial: in order to use our features the user must be aware of the different manipulations. Having a hands-on scene is much more understandable than text, as displayed in the controls menu of Antoine, which is now used as a cheat sheet for the controls. See figure 9 and figure 10.

Making more of those user driven tests can be a good starting point for the future of the project: What must be done differently? What works well? What is not understood by the user? Is the emergent behavior really pointed out?

Concerning the future steps and possible updates, here are a list of changes that we already thought of and that can make the activity better:

- Discuss the way in which a group of robots should behave: as explained a few times above, we have to decide how we want a group of robots to rotate or move. Considering it as an unique component instead of several robots connected to each other can help for the self-motion feature and can bring new mechanics for the activity.
  Think about having two *Cellulo-Mori* connected to a *Mori*. How should the rotation

[9] Hala Khodr et al. *Cellulo-Mori: Reconfigurable Modular Swarm Robots*. URL: `https://github.com/DeepGreen1/cellulo-mori/blob/main/Modular%20Cellulo%20-%20Grass%20Root%20Proposal.pdf`. (accessed: 2021).



CONTROLS

Move up	W	Pause Game	Esc
Move down	S	Switch Camera	Space
Move left	A	Exchange Color	C
Move right	D	Display Sides	X
Rotate left	Q	Disconnect	R
Rotate right	E		

BACK

Figure 10: The controls cheat sheet

take place: around one of the *Cellulo-Mori* or with a new center of rotation in the middle of the group (in this case, in the middle of the *Mori*)? Also, how should a movement act: should one of the two *Cellulo-Mori* drag the other robots or should both *Cellulo-Mori*s move in the same direction? Those questions depend on the activity and discussing it can change the mechanics of the simulation.

- Update the activity or add a new level that could make use of the cycle detection. This would allow us to use the third dimension as a fundamental part of the activity and make a more intense use of the properties brought by the *Mori*s.

- Add a UI interface to change the way of using the joints. For now, we can select one particular joint and change the angle between the connected robots. Think about having three *Mori*s connected to a *Cellulo-Mori*: it would be easier to use a pre-defined behavior if we want to make a flat plane with the *Mori*s or to create a particular shape.

- Re-think about MuJoCo. With more documentation and a more developed API, it could be worth using it.

- Change another time the mesh of the *Mori* prefab in order to have more accurate self-connection feature.

A lot more can be done and the constraints are part of the activity in which those features would be used. Thinking about the activity before implementing them would be a good idea.

8 Conclusion

This project was really interesting and made me discover Unity as well as the teaching power of such robots. Working on a project that grows as time goes on and in particular working with tangible objects is really exciting.
This simulation was done in parallel with another student, this was really a good thing even with the problems that occurred due to merging. Helping each other made us learn more about Unity than what we would have if we were alone. Also, creating an activity needs a lot of imagination, working with someone else brings up ideas and helps to find good ones.

Thinking about the activity before starting to create the game was a good idea, it would also have been great if we had thought about all the features before starting to implement them. On multiple occasions I had to recreate a feature or part of it, due to a new one that changed how the scripts worked together. Also, having mixed-up features made the work of debugging nontrivial and I think that with more experience and knowledge we could have improved the final version of the activity. Nevertheless, it was really pleasant to learn on the go and to see every developed piece working together.

In the appendix, you can find some screenshots of the final activity that we created and that make use of the work presented in this paper, see figure 11.

The code is available on GitHub[10].

Acknowledgments
I would like to thank my supervisors Hala Khodr, Kevin Holdcroft, Barbara Bruno and all the people that were present on the weekly meetings. They helped me through the process of learning Unity and working with the robots during the semester.

Also, a great thank you to Antoine Clivaz that worked with me in parallel on this project, as well as my family and friends for their support and for their proofreading.

[10]`https://github.com/DeepGreen1/cellulo-mori/`

9 Appendix

Some screenshots of the final activity.

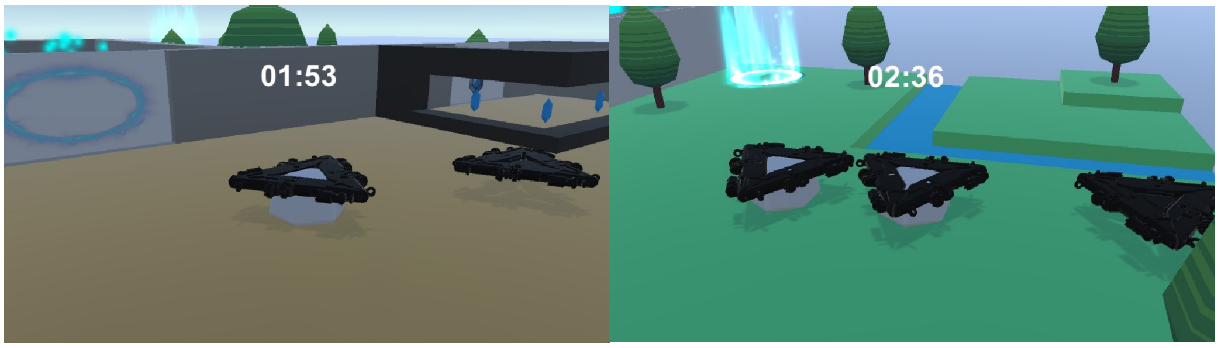


Figure 11: First person view of the activity

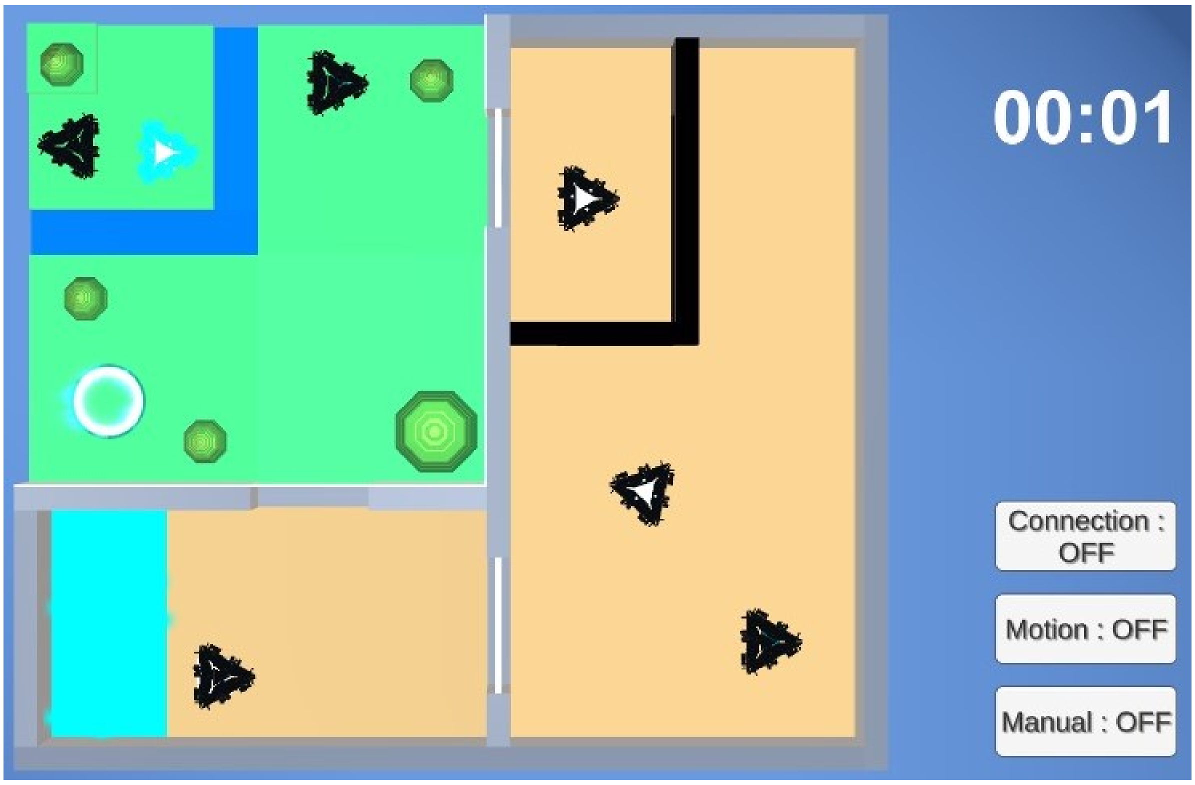


Figure 12: Global view of the activity

List of Figures