Github link : https://github.com/Totocmoi/ComputerGamesFinal

UNIVERSITÀ DEGLI STUDI
DI GENOVA

Electric circuit simulator

Computer Games Final Project

Tom Lorée

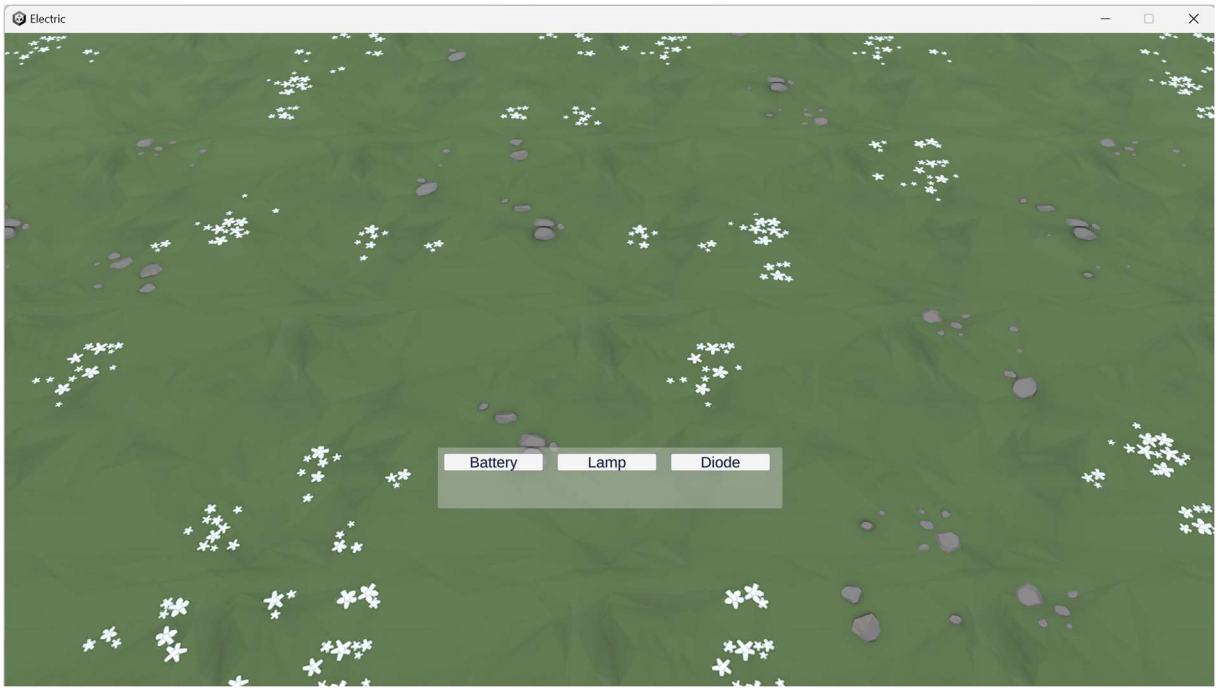
Introduction

The main goal of the game is to be able to build and simulate an electric circuit as if the components were here. The game is created using minimal package and is designed to be as modulable as possible. For now, the components available are a battery, a diode, a LED, and a flip-flop. It doesn't consider the tension nor the intensity of the current, and all components are supposed to have an electrical resistance.

In this report, I will detail how to play the game in the first part, and then how the game works in detail.

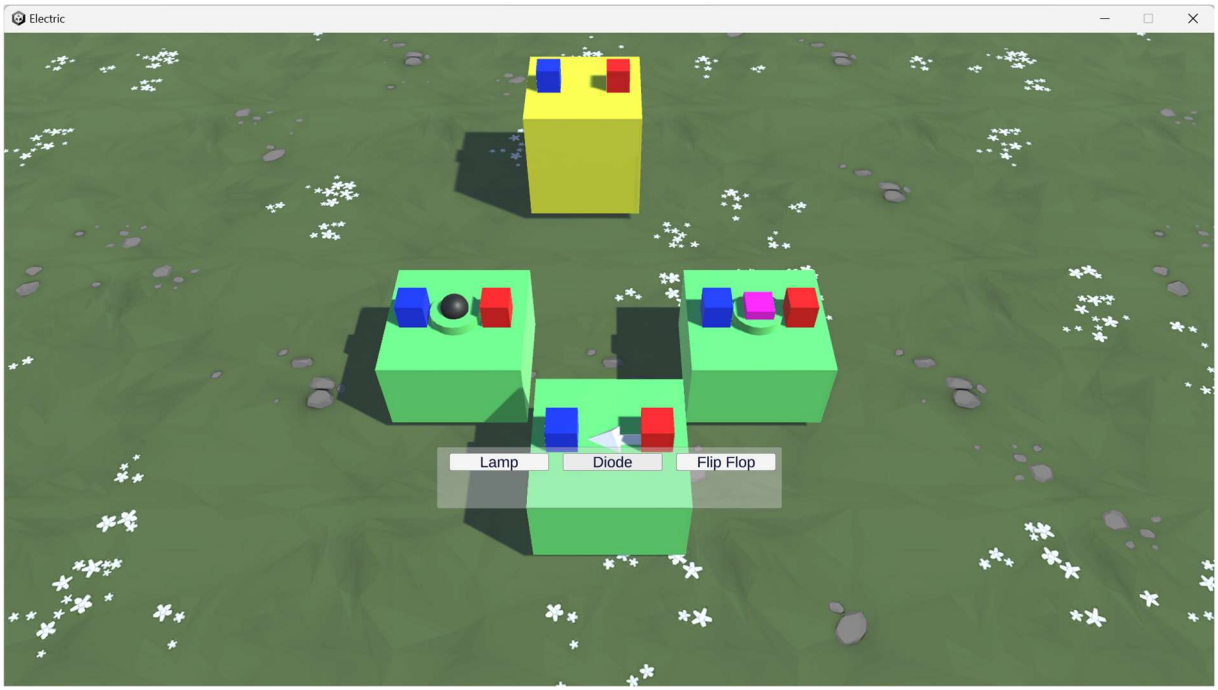
How to play

When launching the game (be it as a unity project or by launching the Electric.exe program inside the "Build" folder), the player is greeted by this view:

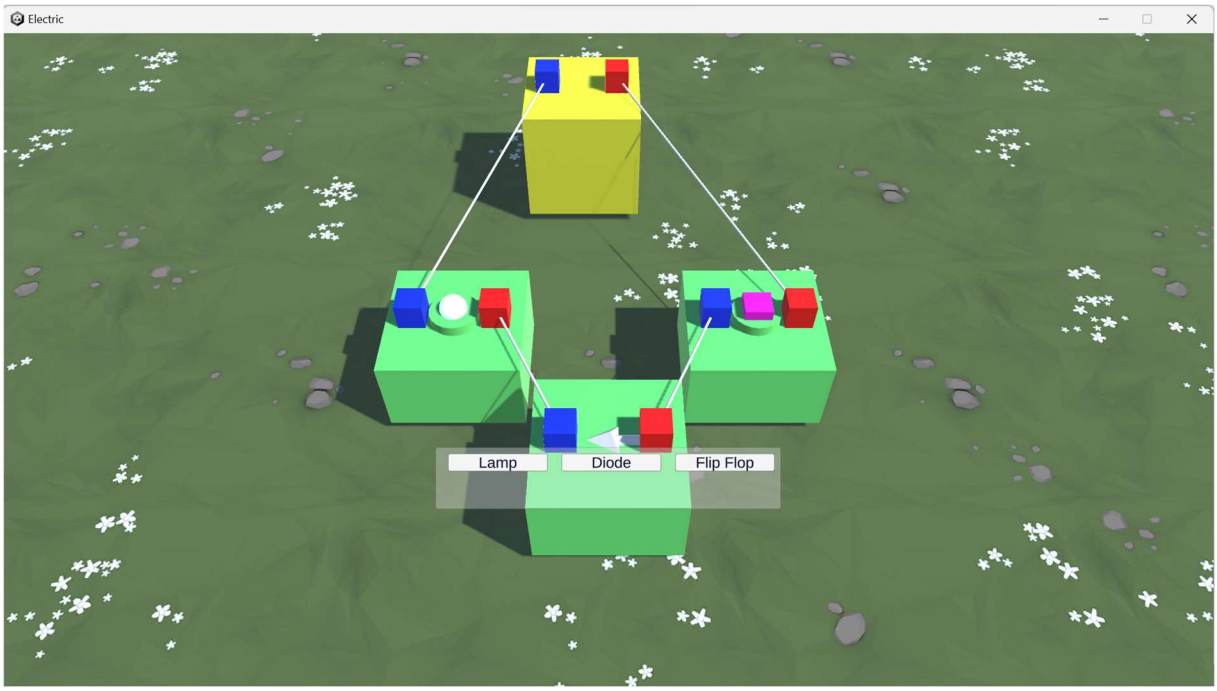


While it doesn't explain anything, the idea is quite clear. What you see is a field (from the course package) and buttons. Please note that you can make the component selector slide to discover other components. In any case, a standard way to start an electric circuit is to add a battery. So, after clicking on the battery button:

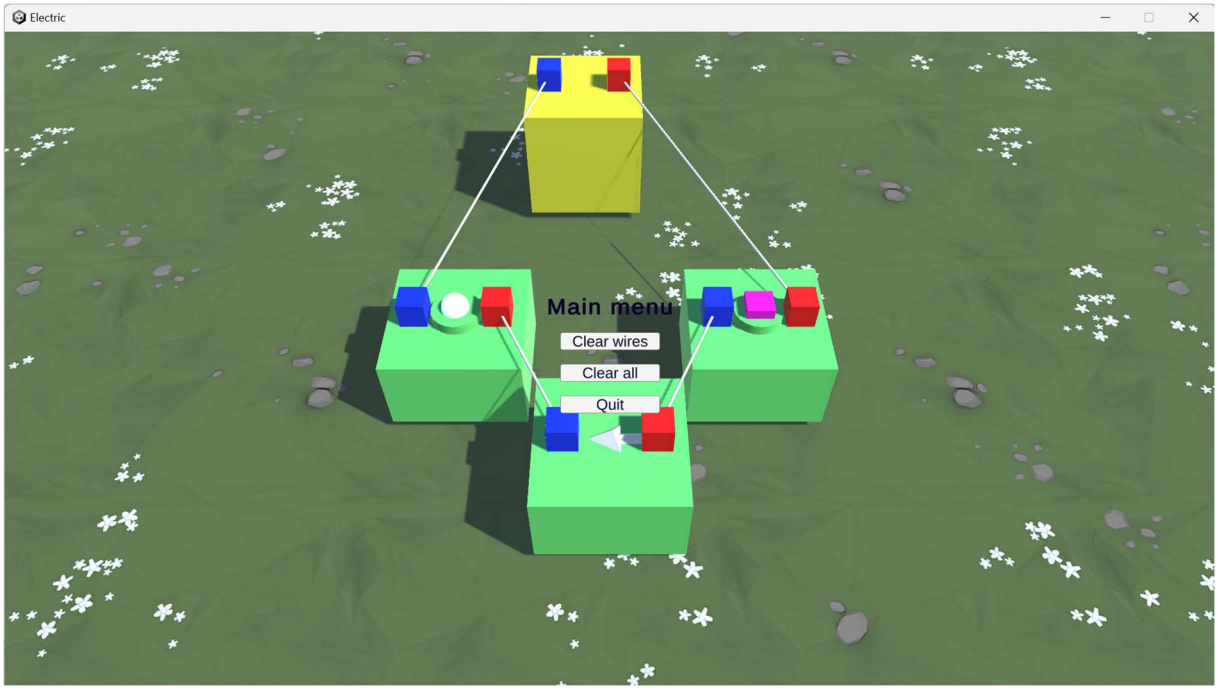
A battery appeared. So now, what we want is that the lamp that we will connect to it is not placed on the same place. It is important to note that a component cannot be moved. So, using the arrows keys or ZQSD, we will move the camera and add a lamp:



To save time, I also added a diode and a flip-flop. So now, how do we connect it all? The answer is quite simple; we just need to click on the red and blue cubes that represent the terminals of each component. For example, we could do this:



We can see that the lamp went on since the diode is on the right side, and the flip-flop is on. Please note that clicking on the violet button would turn the flip-flop off and would also turn the light off. But now, we want to build a different circuit. At this point, pres ESC to open the menu:



It is important to note that while the main menu is open, it is not possible to interact with the components in the scene. The buttons are quite straightforward: “clear wires” will erase all the wires; “clear all” will reload the scene (and thus erase all wire and all components); “quit” will close the game (only in the Electric.exe build).

With this, you are now a master of the game. Go and build circuits!

Technical details

In this part, I will describe in more details how each part of the project works.

I) Components

The first thing to talk about is the creation of components. Each component have a script on their own, but each one of them is an extension of the CompoScript class. This class is used to have a common ground on the components. It is an abstract class, and have the abstract methods Activate() and Deactivate(). These are the methods used by the circuit to turn on and off the components. It also has the virtual methods Explode() and Button(), that are used by components that needs to explode (in the current build, the battery), and those that have a button (in the current build, the flip-flop). Lastly, it has four Booleans that define if the component is active, if it is a power supply, and if the electricity can pass through (one for each direction). This allow the circuit to check the nature of a component without asking for its name or test each possible parent script. It also allow for new components to be added without problem.

II) CableManager

CableManager is the most important script of all the project. Not only does it manage the cables like the name suggest, but it also serves as a circuit manager by activating and deactivating the components, and even as a game manager since it has the functions the main menu buttons use. To do all of this, it has six functions. The first function, that is his main one, is AddPoint(GameObject). This function is used when a terminal is clicked on. It takes this terminal as an argument and try to add a wire between the previously clicked terminal and the argument. If there is no previously clicked terminal, it will store this one, otherwise it will check if the two terminals are distinct. If not, it will create a wire and store it according to the tension it should be at (in other words, the terminals that are linked together are grouped inside a “supposed tension value”). In any case, it will forget the two terminals and update the electrical components (by calling the second function).

The second function is UpdateElectricThings(). It will start by deactivating all components that ever went inside the CableManager script (since it is the script that activate components, it deactivates every activated component). Then it will fill a dictionary of all paths possible for electricity from one group to another, while considering the capacity of the component to let electricity pass. It will, at the same time, fill a list of all the terminal groups of the power suppling components. Then, for each of these power source, it will search for every path from the first group to the other, trough the dictionary. To do that, it will call recursively the third function, that will activate the components that are on a way from the plus to the minus terminal of a power component. Please note that this way of activating components suppose that each component has an electrical resistance, but it allows to ignore short-circuited components.

The last three function are the functions used by the main menu, so they do exactly what the main menu buttons do (erase all the wires, reload the scene and quit).

III) Other interesting scripts to note but that can’t make whole parts

In this part, I will explain the script for the camera, the terminal (BorneScript), and the component button.

The script for the camera is quite simple: when the player press escape, it will change the Boolean isMenu that define if there is the main menu and set as active the corresponding menu. It will also set as active the Raycaster that allow to click on the elements. Last thing it does, is to move the camera depending on the user input.

Then the BorneScript. When the terminal it is attached to is clicked on, it will call the AddPoint function of the CableManager.

As for the component button script, it will instantiate the given component when the button it is attached to is clicked on. The component prefab is given through the editor.

Conclusion

To conclude I will say that this electrical simulator is not a revolution on its own, but its modularity, ease of use and ease of extend make it something very interesting. It could be interesting to add, in future updates, other components to extend the modularity and usefulness of this project.