WebGL simulation tool for Micromouse contest

Tiago Madeira 76321 Diogo Duarte 77645

***Abstract* –This paper showcases the final product of an original project developed in the field of Visual Computation, a subject that is part of the fourth year of University of Aveiro's course Engenharia de Computadores e Telemática, as well as details of its main aspects, the main features, the user interface, as well as a brief description of the work done, what was considered most important, the main goals and the problems and solutions found in order to develop this solution. The idea of creating a 3D WebGL simulation environment for the Micromouse contest was suggested and accepted as a final assessment item for the first set of classes, which focus on WebGL. Through an approach that is intended to be simple and organised, this paper starts by setting the context, scope and motivation that back this project up. Various features, ideas and problems are then presented, following roughly the real chronological order in which they came about throughout the development of the project. Finally the main conclusions are discussed, going over the acquired knowledge, the main goals and their completion and the enriching characteristics of the project, technical-wise and social-wise.**

***Resumo* –O presente artigo apresenta o produto final de um projeto original desenvolvido no ambito da unidade curricular de Computação Visual, que faz parte do quarto ano do curso de Engenharia de Computadores e Telemática da Universidade de Aveiro, e detalhes sobre os aspetos principais do mesmo, as funcionalidades mais importantes, a interface, bem como sobre o trabalho efectuado, o que foi considerado mais relevante, os objetivos principais e os problemas e soluções encontradas, para que a solução fosse desenvolvida. A ideia de criar um ambiente de simulção 3D para o concurso Micromouse foi sugerida e aceitada como elemento de avaliação final para o primeiro conjunto de aulas, que se foca em WebGL. Usando uma abordagem que se pretende simples e organizada, o artigo começa por estabelecer o contexto em que o projeto surgiu, o ambito e a motivação por trás do mesmo. Em seguida são apresentadas, tirando partido da ordem cronológica real aproximada pela qual surgiram, as várias funcionalidades, ideias e problemas ao longo do desenvolvimento do projeto, colmatando com a discussão das principais conclusões tiradas do projeto, que passam pelo conhecimento adquirido, pelos principais objectivos e o seu cumprimento e pelas caracteristicas enriquecedoras do projeto, tanto do ponto de vista técnico, como do ponto de vista social.**

***Keywords* – MicroMouse, WebGL, Simulation**

I. Introduction

In the context of developing an original project in the field of Visual Computation, a subject that is part of the fourth year of University of Aveiro's course Engenharia de Computadores e Telemática, this idea of creating a 3D WebGL simulation environment for the Micromouse contest was suggested and accepted as a final assessment item of the first set of classes, which teaches the usage of the Web Graphics Library or WebGL. It is interesting to create projects that have real use across different areas of knowledge and expertise, and the community around the contest Micromouse is known for its enthusiasm and for being welcoming. Being closest to the world of keyboards and computer screens, and further from the electronics and wiring, the idea of the project is, not only to build something useful to those who already participate in the Micromouse, but to also bring both worlds closer together. The Micromouse is a contest where small sized robots navigate a maze and try to solve it. It began around the late 1970s and the events are held worldwide, being most popular in the UK, U.S., Japan, Singapore, India and South Korea.

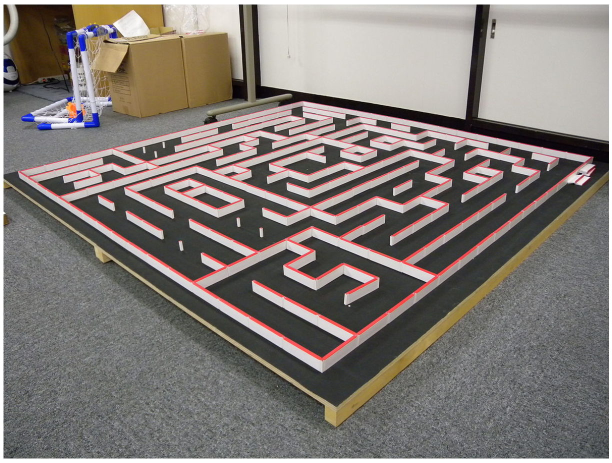

Fig. 1 - Micromouse maze [1]

II. Building blocks

The first approach to the problem was to research the ratios of the pieces of the official maze in order to be able to build it to scale. According to the Robotics Society of America [2] the whole maze's area is to be at 2.88m x 2.88m and the separations are regulated as 5cm high and 1.2cm thick. This information was used to scale the pieces in the 2unit x 2unit canvas, creating the floor in the size of the maze and the emblematic `walls` and `posts` that connect them, as seen in figure 1. Textures were used on the `floor`, `walls` and `posts` in the style of the real life setting. The background was given a grey tone, in order to not be confused with the walls, depending on the angle of the lighting later on, and to make a scene not as harsh on the eye. The official `Micromouse` regulations on the `mouse`'s shape aren't very strict at all, the only rule to it is that it must be smaller than 16cmx16cm, as such it was decided that the `mouse` would be a medium sized rectangular parallelepiped shape with the texture of a generic mouse circuit board, bearing a round front, and the edges cutout in the back, this would be considered while calculating collision. The basic models used to create the whole environment can be seen in figure 2.

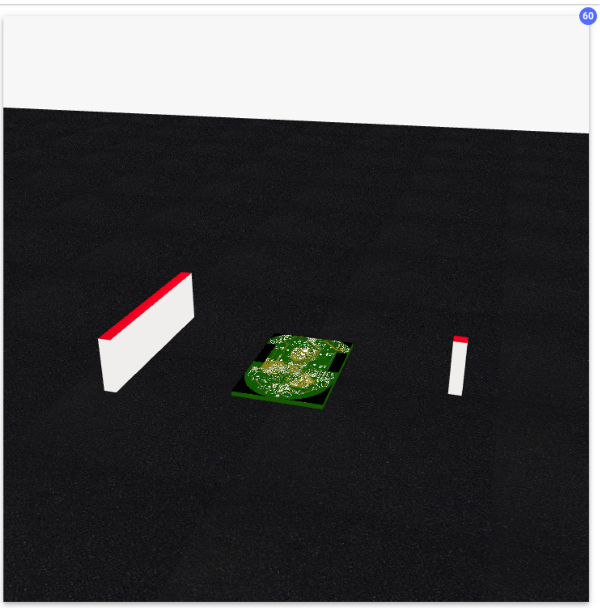

Fig. 2 - Wall, mouse and post

III. MAP INPUT

The aspect of the application that is most important for users to be able to temper with is the map's configuration. There are some rules of course that serve as guidelines for the creation of the simulation's environment and the validation of the mazes inputted by the user. There are a lot of variations of these rules, but currently the most classic and well-known version of the contest is the one being used. Ergo the maze must be described as 16x16 cells where the finish line is in the centre 2x2 cells square and the mouse always starts in the bottom left corner. Two different formats of text file are allowed to be used to describe the maze and these are checked to see if the maze truly is 16x16, if all the outer `walls` are placed and if every possible place for a `wall` is described as having a `wall` or not, this validation is done through the use of regular expressions (regex). All the `posts` are always drawn, even if not between `walls`, as a way to visualise easily the cells of the map, they are also kept a lot of times in real-life as seen in Figure 1. During the loading of the maps, it's added to the dictionary of variables the coordinates of all the `walls`, centres and corners, this is used to draw them and to check for collisions with the `mouse`.

A. Format A

In format A all the pieces area represented, the `walls` and `posts` as *'#'* and the free cells and empty walls as *'-'*, this is merely as a way to visualise the `plain text` more easily. An example of a maze in format A can be seen in Figure 3.

```
# # # # # # # # # # # # # # # # # # # # # # # # # # # # # # # # #
# - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - #
# - # # # # # # # # # # # # # # # # # # # # # # # # # - # # # - #
# - # - - - - - - - - - - - - - - - - - - - - - - - - - - - - - #
# - # - # - # - # - # - # # # # # - # - # # # # # - # # # # # - #
# - # - # - # - # - # - # - - - # - # - # - - - # - # - - - # - #
# - # - # # # - # # # # # - # - # # # # # - # - # # # - # - # # #
# - # - - - # - # - - - - - # - - - # - - - # - # - - - # - - - #
# - # # # - # # # - # # # # # # # - # - # # # - # - # # # - # - #
# - # - # - - - # - - - - - # - # - - - # - # - - - # - - - # - #
# - # - # # # - # # # - # - # - # - # - # - # # # # # # # # # - #
# - # - - - # - - - # - # - - - # - # - - - # - - - # - - - - - #
# - # - # - # # # - # # # - # # # - # - # - # - # - # - # - # # #
# - # - - - - - # - - - - - # - - - # - # - # - # - # - # - - - #
# - # - # - # - # # # - # - # # # # # - # - # - # - # - # - # # #
# - # - - - - - - - # - # - # - - - - - # - - - # - - - # - - - #
# - # - # - # - # - # # # - # - # - # # # # # # # # # # # # # - #
# - # - - - - - - - # - - - # - - - # - - - # - - - - - - - - - #
# - # - # - # - # - # - # # # # # # # - # - # # # # # # # # # - #
# - # - - - - - - - # - - - - - # - - - # - - - - - # - - - # - #
# - # - # - # - # - # # # - # - # - # - # # # - # # # - # - # - #
# - # - - - - - - - # - - - # - # - # - - - # - - - # - # - # - #
# - # - # - # - # - # - # # # # # # # # # - # # # - # - # - # - #
# - # - - - - - - - # - - - - - # - - - # - - - # - - - # - - - #
# - # - # - # - # # # # # - # - # - # - # # # - # # # - # - # - #
# - # - - - - - # - # - - - # - - - # - - - # - - - # - - - # - #
# - # - # - # # # - # - # # # # # - # - # - # # # - # # # - # - #
# - # - - - # - - - - - - - # - - - # - # - - - # - - - # - - - #
# - # # # # # # # # # # # # # - # - # - # # # - # # # - # # # # #
# - - - # - # - - - - - # - # - # - # - # - # - - - # - - - - - #
# - # - # - # - # - # - # - # - # - # - # - # # # - # # # - # # #
# - # - - - - - # - # - - - - - # - # - - - - - - - - - - - - - #
# # # # # # # # # # # # # # # # # # # # # # # # # # # # # # # # #
```

Fig. 3 - Map input format A

B. Format B

In this format the `posts` are omitted, since they aren't defined by the user, as well as the empty spaces and the `walls` are represented as *'—'* for the vertical walls and *'_'* for the horizontal walls. This format, despite being more appealing in the `plain text`, might be harder to edit a map. An example of a maze in format B can be seen in Figure 4.

IV. MOUSE MOVEMENT

Three different ways to move the `mouse` within the maze were implemented, `Free`, `Constrained` and `Script`.

A. `Free Movement`

This way of moving allows the user to rotate the `mouse` in any angle and go forwards or backwards, moving

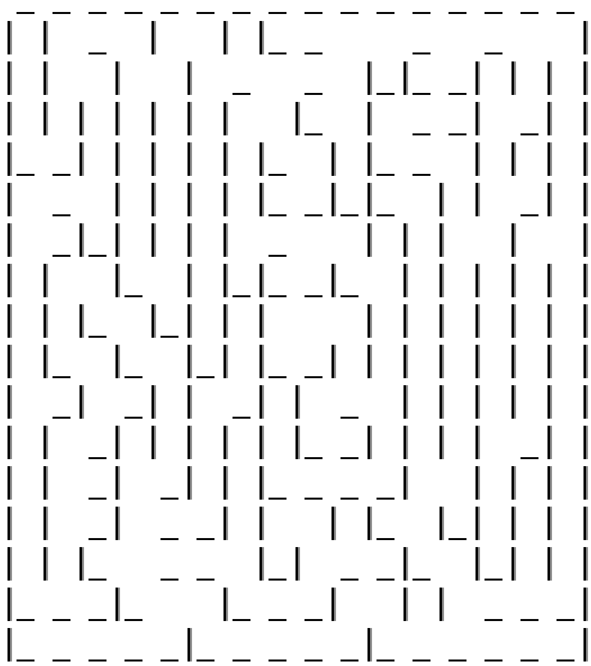

Fig. 4 - Map input format B

freely in any allowed space. In this form of movement the collision calculation was quite a challenge, and some simplifications had to be considered since the logic was to be built from the ground up. This mode is also the most demanding in terms of hardware, on account of all these calculations. The `posts` alone are not considered for collision, since their primary purpose is to mark the boundaries of the blocks of the map and, in real-life situations, to help put together the walls, in which setting it may be decided either way if the posts are to be kept or taken off, however it is generally agreed that they don't bear the purpose of blocking the way of the `mouse`. Collision-wise all the `walls` are considered wider, covering for the space occupied by the `posts` between them, and the `mouse` is boiled down to three points in the front, these are placed in a slight curve emulating the rounded front of the circuit texture, and two points in the back which are positioned along the cutout corners. When the `mouse` collides with a wall at an angle, since the movement is treated as two separate components, one of them is still applied, allowing the mouse to slide along the side of the wall, and avoiding "sticky walls". The rotation movements are also checked for collision, not allowing the front of the `mouse` or the back to rotate into a wall.

B. *Constrained Movement*

`Constrained Movement` is the most efficient way to control the `mouse`. Translation is "block-by-block", forwards and backwards, and rotation left and right is done ninety degrees at a time, taking advantage of an animation to smooth it out through time, this animation's duration can be controlled by the user through a slider. Given that in this case, the `mouse` travels a fixed distance, and that there are only four angles allowed to travel in, the calculation for collisions is simplified to only checking weather there's a wall between the `cell` where the `mouse` is, and the `cell` it's trying to travel to, being less demanding compared to `free movement` control, allowing for a better user experience.

C. *Script Movement*

This type of movement is perhaps the most ambitious, the `mouse` itself moves similarly to the `Constrained Movement` version but the user is allowed to control the movement through inputted JavaScript code through a text box, using the functions provided they can create an original algorithm to guide the `mouse` and learn information, facilitating a user to test their solving algorithm in multiple mazes. This is made possible by running the code given by the user in a loop that returns the current state of the mouse, i.e., it returns if there are walls at the left, front or right side of the mouse in that instance, through the array `pathIsClear`. After receiving this information the user can store the maze information in the provided variable `maze` and then decide its next move, using the following functions, `forward`, `back`, `right` and `left`, that move the mouse forwards, backwards and rotate the mouse to the right and left, respectively. An example of some valid code can be seen in figure 5 and its result, with `Bread Crumbs` enabled in figure 6

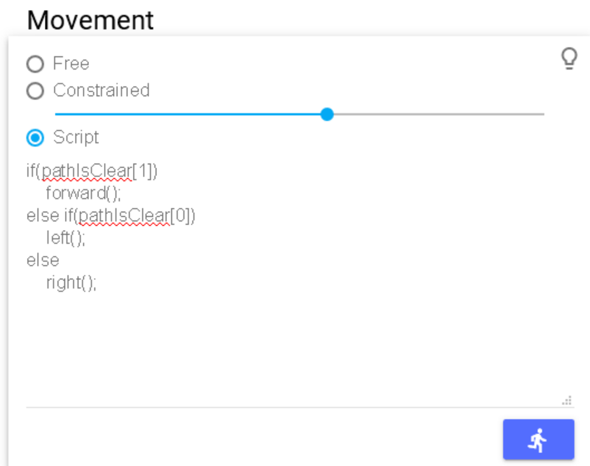

Fig. 5 - Example of script inputted in the code box

V. Camera

The camera options are well-known and therefore pretty straightforward. There's a `Free Camera` option, as seen in figure 7, allowing the user to spin the map around the vertical axis, and along the horizontal axis, tilting it towards or away from them, as well as zoom in and zoom out fixed in the centre of the maze. A `Top View Camera`, as shown in as seen in figure 8, was also implemented, this one is fixed and shows the entire maze directly from above, *Bird's-eye view* style. Finally, there's a `First-person Camera`, figure 9, al-

Fig. 6 - Result of the execution of code in figure 5

lowing the user to *be the mouse*, not allowing sight above the walls.

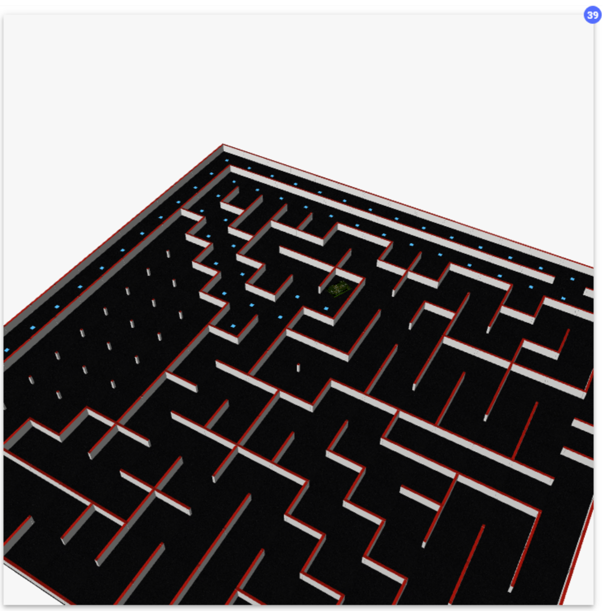

Fig. 7 - `Free Camera`

VI. Illumination

In terms of illumination, it was decided that `point lighting` was to be used, instead of `Phong Illumination`, which was taught in the subject's lessons. This was mainly due to performance issues, since the `Scene` has a reasonably high number of models being drawn at the same time, five hundred plus models in average, our `Phong Illumination` implementation made the simulation too slow, especially if in `free movement`, running in a remarkably low frame rate. So, to mediate this problem, it was implemented

Fig. 8 - `Top View Camera`

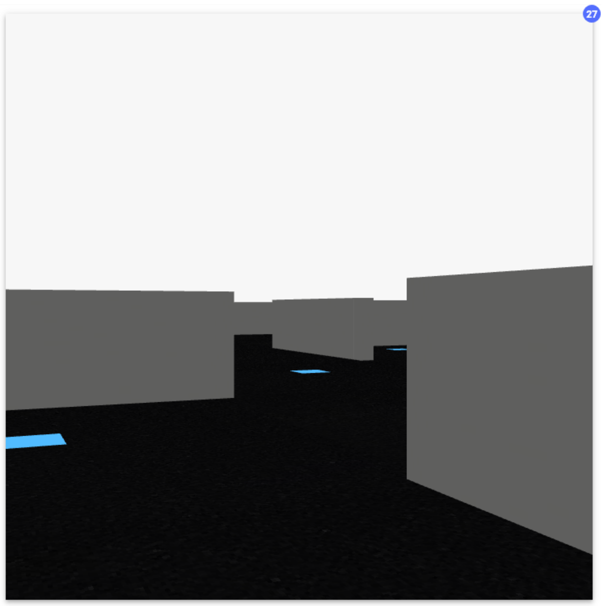

Fig. 9 - `First-person Camera`

`point lighting` as taught in *Learning WebGL* [3] Lesson 12. Even though it doesn't has as many options in configuration as the `phong` implementation being used previously and might be not as realistic, it's less demanding and it worked fine in this case. The effect of the illumination can be seen by rotating the maze in the `Free Camera` mode, as well as by turning the lighting effect ON or OFF using the respective button, as seen in figure 10.

VII. Bread Crumbs

The bread crumbs came about from the need of being able to localise the mouse more easily when in `first-person` view, this tends to be quite confusing, as expected from a maze setting, it also proves useful in

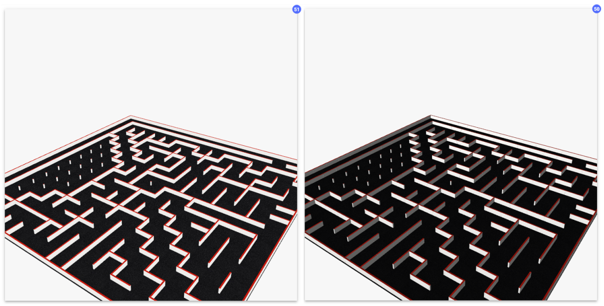

Fig. 10 - `Lighting OFF and ON`

the context of being able to spot easily the path taken by the `mouse`, particularly while using a script, which may be crucial whilst testing an algorithm. So, when turned ON in the settings, blue squares will be drawn in the centre of the cells when the `mouse` has already visited, as shown in figure 11

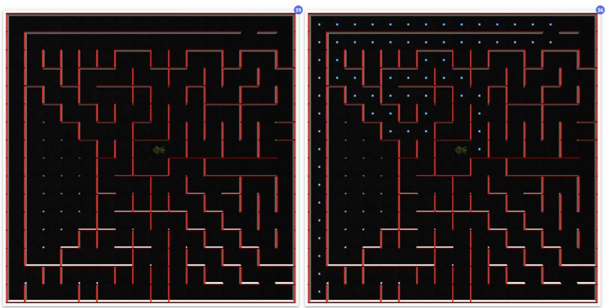

Fig. 11 - `Breadcrumbs OFF and ON`

VIII. User interface

The project's main focus is to replicate the setting of a Micromouse contest and to build it in a way that would allow for user input in various ways. Therefore, a lot of work was also put into creating a user-friendly interface that would allow the intuitive use of all the implemented features. Even so, most of the elements of the page display information about them on `mouse-over`. The web page that serves as the interface was implemented taking advantage of `Material Design Lite MDL [4]`, separating the main controls, canvas and information displayed and presenting it in a clean minimalist way. On the left side of the canvas there's one `card` containing a `radio-button` style menu for the selection of the camera mode, a group of controls to upload the map file, reset the `mouse`, turn the lighting ON and OFF and finally to enable or disable the `Bread Crumbs` feature, this can be seen clearly in figure 13. Right next to these controls is placed the timer, showing the elapsed time since the start of the current attempt at solving the maze. Beneath all these is a bigger `card` containing the movement options, again in the form of `radio-button`, a slider to control the animation speed when in `constrained` or `script` movement and a `text box` allowing the user to input code to be run, using the bottom button, when the `Script` mode is selected, as shown in figure 14.

The canvas fits nicely inside a card of its own and has a `badge` style counter for the frame-rate display. To the right of it is an ordered list of the last ten times taken to complete a maze, kept in the session, see figure 12.

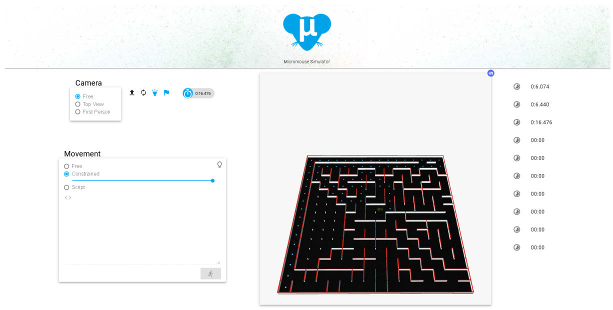

Fig. 12 - User interface

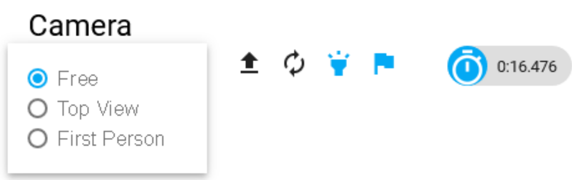

Fig. 13 - Camera and main controls

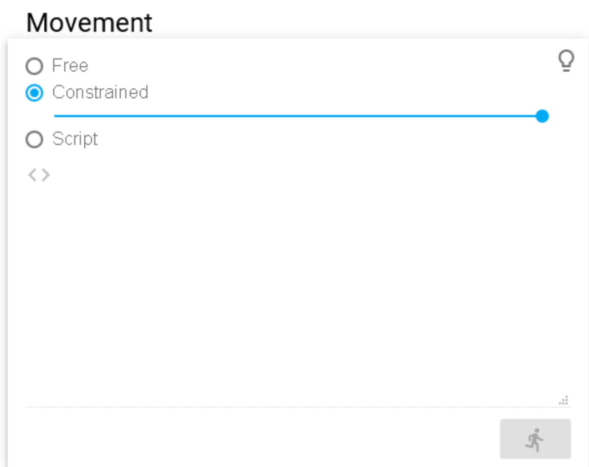

Fig. 14 - Movement controls

IX. Conclusions

Throughout the development of this project, it was possible to learn how a multitude of techniques work and match them together in a greater scale than merely lessons and tutorials can provide, with a greater motivation backing it up. It was all planned with a target audience in mind but was kept intuitive and `fun` enough, through the implementation of features like the `First-person Camera`, that it hopefully can be enjoyed by a wider group of individuals, not only `Micromouse` enthusiasts. Furthermore it is important to highlight that, from the start, it was an end-goal to create something more than a thematic simulation game, but rather a tool that would be valuable for others, namely through the possibility of testing maze solving algorithms in a simulated environment as discussed in section IV-C. Having that said, the goals set for the project were achieved. It is also of course important to go over the whole social aspect of working in a team, even if only a two-man one, soft-skills become imperative, cooperation and understanding is crucial and something to always be worked on granted the opportunity, which this project has.

X. Grades

Estimation of the contribution of each element of the group in percentage, considering the different nature of the tasks selected:

Diogo Duarte	Tiago Madeira
54	46

Estimation of the relative effort of each element of the group:

Diogo Duarte	Tiago Madeira
45	55

References

[1] "Wikipedia - micromouse".
URL: wikipedia.org/wiki/Micromouse

[2] Robotics Society of America, "Maze solving / micromouse rules".
URL: http://robogames.net/rules/maze.php

[3] Learning WebGL, "3d programming for the web".
URL: http://learningwebgl.com/blog/?page_id=1217

[4] "Material design lite".
URL: https://getmdl.io/

[5] J. Madeira, "Webgl a quick introduction".
URL: http://sweet.ua.pt/jmadeira/WebGL/20170913_WebGL_JM.pdf

[6] The Khronos Group, "Opengl es for the web".
URL: https://www.khronos.org/webgl/

[7] Ed Angel and Dave Shreiner, "An introduction to webgl programming".
URL: https://www.cs.unm.edu/~angel/SIGGRAPH14/Introduction%20to%20WebGL%20Programming.pdf

[8] "Micromouse online".
URL: http://www.micromouseonline.com/

[9] "Micromouse usa".
URL: http://micromouseusa.com/

[10] "Micromouse utad".
URL: http://www.micromouse.utad.pt/

[11] "W3 school - javascript".
URL: https://www.w3schools.com/Js/