REPORT: Projet 2A S3 SDD

EFREI 2022

Chicky Graphs

Team Tardigradus

Gabriela de Almeida — Teaching Manager
Chloé Carayon — Communication Manager
Vincent Mouillon — Project Manager
Victor Taillieu — Technical Manager
Luca Vaio — Design Manager

Figure 1: Chicky Graphs logo

December 18, 2018

Contents

Introduction

For our third semester at EFREI, we had to create an original project entirely coded in C using trees or lists. We decided to go further by using graphs as trees and lists are specific kinds of graphs.

What is the theory of graphs? The theory of graphs covers everything about graphs: how to deal with them, implementation, traversals, spanning forests etc...

We currently have the first level of difficulty by the end of which the user learns what Adjacency Matrix and Adjacency Lists are. We plan on continuing the project until the last level of difficulty where the user learns about traversals and spanning forests and models a riddle with a graph.

1 Our Project: Chicky Graphs

1.1 The Team

As Gabriela had already studied the theory of graphs and as it is a subject which seemed quite complicated, we thought of a learning platform for it. We thought it would be a good idea to make a game to teach the theory of graphs in an easy, playful and fun way. Vincent took the lead on this project and with Victor, they had pretty heavy knowledge in programming so they were responsible for it. Luca created our main character, Poussin and all the other assets. Chloé and Gabriela mainly worked on the learning experience.

1.2 Subject

We had in mind a project aimed to teach and be useful to people. To make the project even more interesting, the subject should appear to be slightly complicated, which will lead to a real apprenticeship.

1.3 Goals of Chicky Graphs

Our idea is to develop a learning software dedicated to learning how graphs work and how to use them. A tree is in fact a particular graph. Thus, it is a way for us to approach this notion and transmit what we learned through the different stages of creation of this project.

The public aimed is students like us or even younger, aged from 15-16. Indeed, we wanted Chicky Graphs to be a fun way to approach a the complicated subject of the theory of graphs.

1.4 Challenges

Our project is playful and interactive, well designed and done so that the user doesn't get lost and enjoys using it: the software gets the user's attention at first sight. Thereby, through this project we stepped in both students' and teachers' shoes, thus leading us to confront ourselves to the difficulties of teaching and point them out to maximize the user experience. We created it in hopes that it will become one day a learning tool for students.

2 Skeleton of Chicky Graphs

2.1 Website

The website was entirely coded in HTML and CSS (and a tiny little bit of Java Script).

There is a menu in which you can choose between four sections: home, team, news and media.

You can download the game as well as this report and the synopsis directly from the website.

The overall design is quite childish but it is a choice. We wanted to make the theory of graphs appear silly, not complicated.

The website is hosted on www.maxdecours.com (only available for a few days...).

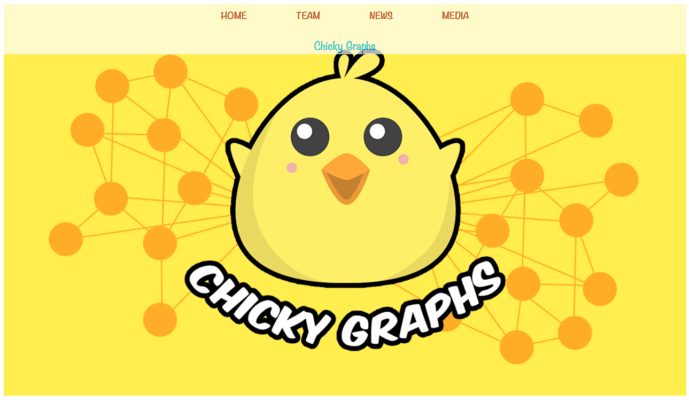

Figure 2: Website, Home section

2.2 Game

For the game, we have decided to do all of our work using C. Also, we use SDL 2 for the graphical interface because of the better performances and text management.

In order to have an organized code, we wanted to follow the MVC architecture (Model View Controller).

As we wanted to create an interactive learning program, we thought about possibilities in order to create different levels of difficulties for our missions. We had to manage the graphs parts, the rendering one and the data of the missions. One of our main structure was the game one which is in our rendering part, in the sdl.h file.

Everything depends on it, it stores all the information utilized in the game loop to avoid many parameters in the game loop which is in the main.c file. We can store the mission in which we are currently in, all graphic related structures, as well as the graph structures and other useful data.

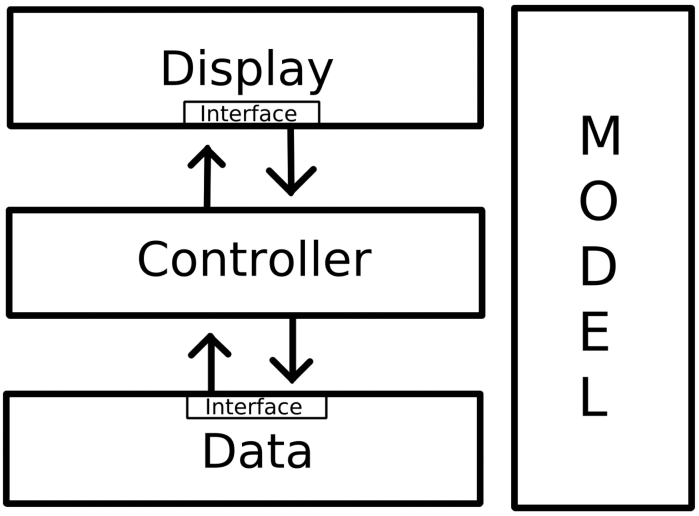

Figure 3: Structure of our code

Controller This part manages the data by transforming it or updating it. The controller is the brain of our program, it combines all of our work, from the data to the rendering.

In our controller folder, we have the main and our graphs.h file.

Firstly, in order to have a clear main, we use the structure game, presented above and we initialize the SDL, then we enter the game loop and the user can have access to the menu, and go to a new adventure, the resume game or the sandbox.

Then we also have the graphs.h, this file gives us the possibility to initialize, create graphs, nodes and edges, delete them, or add some, search and contaminate a node. It gave us the opportunity to learn graphs and then be able to explain them. We used these functions during all of our program for example to compare the user graph to one of our graphs. Graphs will be the main part of our program since this is what we want to teach. It also called the SDL related functions for the rendering part of the program.

Display Finally, the display, was one of the most important part of our work. In the folder Rendering, we had four important files : sdl.h, renderringSLL.h, texTree.h and text.h. Firstly, sdl.h is the core of the display implementation. It has the two main function of the game loop: handleEvents() and render(). The first function takes care of all input, and the second one of all output. There is a multitude of helper functions for each of these functions for each action to have a cleaner code base and avoid repetition. All the input is processed in sdl.c and calls a multitude of other functions to update the data of the game. Then, we have the render function. the render function is here to display the necessary information to the user, such as the graph, the text, or the UI. The main two functions that are called in this function are: renderRenderingSLL() and renderText(). The first one renders every element of the renderingSLL in a sequenced manner, but, what is a renderingSLL ? Well, it is a structure that contains every elements that have to be rendered in multiple Single Linked Lists. There is three SLLs, the first one is to contain all edges, the second one is for the nodes and the last one is for the ui elements. the renderRenderingSLL() function will first display the edges, then the nodes on top of it, and then the UI. The edges and nodes are automatically added when the related controller functions are called. the UI is initialized at the beginning of each screen by the data-layer. The renderText() function will render the first text contained in the game TextSLL.

Each element of the screen has a different texture. We wanted a fast system, with no duplicate textures for each nodes. So, Vincent created the texture tree. The idea is to load the different textures in a tree when starting the game, and then link them to each node with a pointer instead of recreating the texture for each node. This is helpful for memory management as it is simpler to free, and for speed as the loading is done just a single time.

The text.h and text.c are very small files, but help make the connection between our program and a library called SDL_FontCache. This is also why we have two SDL_FontCache files in our project.

Data We have two folders which are important for our Data. Firstly, we have 'Data' where we have put all our data: assets, fonts, missions and the website. The website allows the user to easily download the software. The assets and the font are used for the display and finally the missions where we find all the initial conditions in order to allow the good functioning of the missions in the game. In fact every mission is divided in three parts : Nodes, Edges and UI. Nodes allowed us to know the size of the graph which will be used for the mission, if it is directed or undirected, the number of nodes (chickies or poussin), their textures, coordinates and size for the display. Edges give the link which exists between chickies and finally the UI allows us to know if there is some images, and texts to display. In order to read the information of each missions, we use a data-layer folder with data.h, where we load the data for a mission. It analyzes the mission file and call all the necessary graphs and rendering functions to have the specified graph and data loaded. This file acts like a bridge in

order to connect data to our features.

Finally, the data interface saves the state of the game in order to allow the user to try some features on the Sandbox for example and come back in the mission they were previously in.

3 Main problems encountered

Firstly, the skeleton of our program was quite hard to find. The project manager worked hard and found the MVC architecture, which allowed a fair distribution of the work inside the team. The distribution of tasks was difficult as everyone had a different levels in programming and did not progress at the same speed.

Secondly, as we used SDL 2, we encountered some problems at the beginning of our project. Some of us had difficulties in order to put SDL 2 on their computer. Moreover, as the graphical interface was at the center of our interest, we had to manage everything we wanted to display, knowing where we had to put it, ect... For example, we encountered problems with the text, we had difficulties to implement, store and delete it.

As said above, we had to share the work. The ones working on the rendering part had a lot of investigation time as this part was crucial in our project.

Finally, time was our enemy. Even if we accomplished our first goals, we designed enough missions for the whole game. We even had a final mission where the user is supposed to model a riddle with a graph on his own. We knew we weren't going to be able to finish the project, but had no idea it was going to be so long.

Conclusion

This project was a new experience. Working in a team of 5 people was sometimes difficult to handle but overall it was a great way to learn to work as future engineers. During this project we worked on our technical skills and learned a lot in terms of theory of graphs obviously, but also in programming in general.

Acknowledgments

We would like to thank EFREI for giving us the opportunity to make such a project. We are looking forward to continue it on our own.

We want to give a special thanks to sir Marshall-Breton, our magnificent teacher who used all of his patience and his best teaching methods to teach us so much this semester and helped us throughout the creation of Chicky Graphs.

List of Figures