aMAZEing

Adrien Nichols, Maxime Encrenaz, Timothée Malossane & Nolan Francois

You've always dreamed o itf. We did it. Put yourself in the skin of a rabbit. A rabbit who must escape from a maze for its survival. A rabbit pursued by a wolf. A rabbit struggling for its survival... You get it? Then aMAZEing is for you. **www.gameone.net**

aMAZEing is the kind of game that will never bore you. Single player or multiplayer, you will spend hours of fun on this game. **www.gamekult.com**

Introduction

First of all, the development was done under Windows. And despite all efforts to make a Linux Portability as perfect as possible, some problems still remain at execution. During this year, we tried to push the project to the maximum of it's possibilities, to improve our knowledge in Lazarus and developing in general.

1. Objectives (In order of resolution)

1.1. Generate random mazes.

To generate a random maze, we wanted to understand and use a simple algorithm. Unfortunately, we quickly realized that it was not as easy as it seemed. A website helped us a lot: http://ilay.org/yann/articles/maze/. We used the method "Uniqueness and random path" that performs better on large mazes. This work has been done on unit <ulabyrinthe> with procedures *CLabyrinthe.NettoieCellules* and *CLabyrinthe.GenererLab.* The difficulty was that we wanted to write mazes on txt files, to eventually create an editor. We needed to adapt the pattern of our mazes to be easily editable.

1.2. How to find the shortest path in a Maze.

For some bonuses, we needed to find an algorithm that gives the shortest path between two points in the mazes at any moment. Our research has led to the Dijkstra algorithm. It is ultimately a mix between A * and Dijkstra we developed, using the speed of the first and the robustness of the second to get a strong algorithm. We also added a heuristic procedure (based on probabilities) to accelerate the speed of our calculations. This work has been done on unit <ulabyrinthe> with procedures *CLabyrinthe.heuristique, CLabyrinthe.get_nb_chemins, CLabyrinthe.get_last_noeud* and *CLabyrinthe.get_chemin.*

1.3. Create a game

What are good ideas without a good presentation? Nothing. That's why we focused on the graphical aspect of our program. First of all, our game needed to be full-screen, job done with the procedure *TFFenetre.SwitchFullScreen*. Some esthetics and we had everything. Or almost… That's why we added music, music that would immerse the player in the game.

Here Lazarus showed its first weaknesses with screen flickering. After hours of work, we changed our way to create graphics and avoided moving Timages on screen to avoid flickering. We could have used an external library called bgraBitmap but at this step we didn't really need it.

And last but not least, we created a splash screen which popped up at each start. Hence we developed a new window called <Ufenetre_splash> that displays itself before being deleted and calling our main window.

1.4. Game Mode

We wanted our game to have several game modes, in order to give it a longer playing "life". We needed to create inherited classes from <Ulabyrinthe>, such that <ulabyrinthe_survivor> or <ulabyrinthe_FirstOut> which change some parameters in the game.

We created a new class called <Upersonnage> able to manage users. Names, avatars, scores, everything can be stored in a file to avoid creating a new character each time. This class has never been finished due to lack of time, it will be one of the first changes in next versions.

1.5. Online Mode

We wanted to create an Online Mode to play with friends. Such a mode was the biggest challenge of our project. We had to start from nothing, create a servor, an exchange protocol for datas, and communicate information from our program to our servor. We decided to use a simple HTTP Protocol, with GET requests. A simple Apache+PHP servor has been used as our host '*http://malossane.fr/services-amazing.html*'. Calling this page with parameters is enough to create a servor. We used a MySQL database to store datas with the structure below:

Database db308952592

Structure de la table s_lazarus

Champ	Type	Null	Défaut
id	int(8)	Oui	NULL
name	varchar(30)	Oui	NULL
ip	varchar(45)	Oui	NULL
ready	int(1)	Oui	NULL
ping	datetime	Oui	NULL
playing	varchar(10)	Oui	NULL
start	datetime	Oui	NULL
top	int(2)	Oui	NULL
loup	int(3)	Oui	NULL

Some optimizations are needed to avoid cheats for example, but it's already functional. The bad surprise we had is that sometimes servors take time to respond. During this time, the game was sleeping and lags happened. We fixed this using Multi-Threading: All our requests are sent with another thread, hence the main thread isn't waiting for an answer and you can move your character

while we communicate with the servor and get informations about other players. The problem was that it's very complicated to develop a program when you don't exactly know when your informations will be available. The time lapse can go from 1 sec to 20 secs, you need to check information each time you use it hence it was very hard to fix it. It seems there are still some troubles on Linux.

2. The final product: aMAZEing

2.1. The Game.

After months of developing we have finally got the end product. A fun and challenging game.

In single player, you are a rabbit that must get out of the maze as fast as he can to escape from a wolf, and in multiplayer you can challenge your friends to race through the mazes while using bonuses to slow their progress. We hope to have made the game interesting enough so that it could entertain players over time as we have tried to polish every aspect of the game, the random mazes work great as you can never use twice the same path making the game exciting even after a few attempts, and added features to extend the games lifespan.

Unfortunately we were unable to include weapons into our bonuses as the developing of this aspect would have been a long process and preferred finishing the main game but we will be sure to add weapons if we have the chance to make another version of the game.

2.2. The team work.

Our team worked very well together. We all had different ideas of what we should do but we all came to an agreement over our final idea: aMAZEing.

We all helped with the developing of the program, the ones with more experience in computer science helping the weaker members of the team and even then we used many internet tutorials to compensate for our lack in knowledge. But the teamwork was most appreciated when we had to rethink several times our ways to approach a problem or an obstacle to our game play, as we say: "Two heads are better than one." and in this case we had four.

Everybody benefited from this experience, would it be in computer science knowledge or experience in management and cooperation within a team, and understands better the effort one has to put into a program to make it the best possible.