

***Class* Group Project**

***Development/Architecture***

Delivery

Group members

Documentation

Version 1.0

Last update: 06/06/2013

Use: Students

Author: Samuel CUELLA

|  |
| --- |
| **Conditions d’utilisations :** SUPINFO International University vous permet de partager ce document. Vous êtes libre de :   * Partager — reproduire, distribuer et communiquer ce document * Remixer — modifier ce document   **A condition de respecter les règles suivantes :**  Indication obligatoire de la paternité — Vous devez obligatoirement préciser l’origine « SUPINFO » du document au début de celui-ci de la même manière qu’indiqué par SUPINFO International University – Notamment en laissant obligatoirement la première et la dernière page du document, mais pas d'une manière qui suggérerait que SUPINFO International University vous soutiennent ou approuvent votre utilisation du document, surtout si vous le modifiez. Dans ce dernier cas, il vous faudra obligatoirement supprimer le texte « SUPINFO Official Document » en tête de page et préciser notamment la page indiquant votre identité et les modifications principales apportées.  En dehors de ces dispositions, aucune autre modification de la première et de la dernière page du document n’est autorisée.  **NOTE IMPORTANTE :** Ce document est mis à disposition selon le contrat CC-BY-NC-SA Creative Commons disponible en ligne http://creativecommons.org/licenses ou par courrier postal à Creative Commons, 171 Second Street, Suite 300, San Francisco, California 94105, USA modifié en ce sens que la première et la dernière page du document ne peuvent être supprimées en cas de reproduction, distribution, communication ou modification. Vous pouvez donc reproduire, remixer, arranger et adapter ce document à des fins non commerciales tant que vous respectez les règles de paternité et que les nouveaux documents sont protégés selon des termes identiques. Les autorisations au-delà du champ de cette licence peuvent être obtenues à support@supinfo.com.  © SUPINFO International University – EDUCINVEST - Rue Ducale, 29 - 1000 Brussels Belgium . www.supinfo.com |

Table of contents

Table des matières

[1 Résumé du groupe 4](#_Toc10761412)

[2 Rapport du projet 5](#_Toc10761413)

[3 Manuel d’utilisation 5](#_Toc10761414)

[3.1 Sommaire 6](#_Toc10761415)

[3.2 Introduction 7](#_Toc10761416)

[3.3 Configuration Requise 8](#_Toc10761417)

[3.4 Instructions d’installation 8](#_Toc10761418)

[3.5 Comment jouer ? 9](#_Toc10761419)

[3.6 Support technique 9](#_Toc10761420)

[4 Documentation technique 10](#_Toc10761421)

[4.1 FONCTIONNEMENT 10](#_Toc10761422)

[4.1.1 Déplacement du curseur 10](#_Toc10761423)

[4.1.2 Appui sur une touche 10](#_Toc10761424)

[4.1.3 Déplacement du joueur 11](#_Toc10761425)

[4.1.4 Objets et clés 12](#_Toc10761426)

[4.1.5 Portes 13](#_Toc10761427)

[4.1.6 Score et écran de fin 15](#_Toc10761428)

[4.2 AFFICHAGE 16](#_Toc10761429)

[4.2.1 *Exemples de « sprites » envisagés* 16](#_Toc10761430)

# Résumé du groupe

Campus: **Strasbourg**

Classe : **A.Sc.1**

|  |  |  |  |
| --- | --- | --- | --- |
| ID Open Campus | Nom de famille | Prénom | Photo |
| 296188 | GRUBER | Quentin | C:\Users\Megaport\Desktop\296188.jpg |
| 295006 | BIER | Maxime | https://media.discordapp.net/attachments/501657572315496460/586318899838976010/295006.png |
| 291960 | CUSIMANO | Laurent | https://cdn.discordapp.com/attachments/497682712652414978/586319063626547221/291960.png |
| 280120 | DUPREZ | Aurélie |  |

# Rapport du projet

Quentin a pris en main le fonctionnement du jeu en programmant les mécaniques centrales du jeu : les déplacements du joueur, les collisions et interaction du joueur, et l’affichage de l’inventaire du joueur et des dialogues.

Aurélie a programmé le système de score du jeu, ainsi que l’affichage du menu et des différents écrans de fin.

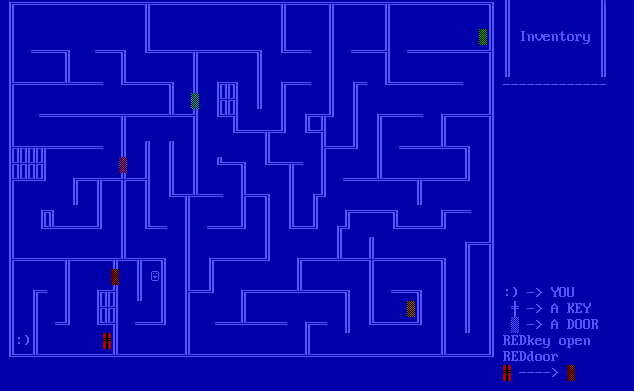
Maxime a conçu dans son intégralité le labyrinthe du jeu, et a participé à sa génération.

Laurent a programmé la génération du labyrinthe dans le jeu.

# Manuel d’utilisation

MAZE GAME

Retro’GameZ



## Sommaire

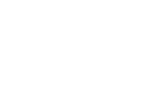
[Introduction 7](#_Toc2478)

[Configuration Requise 8](#_Toc2479)

[Instructions d’installation 8](#_Toc2480)

[Comment jouer ? 9](#_Toc2481)

[Support technique 9](#_Toc2482)



3

Maze Game – Retro’GameZ

## Introduction

Maze Game est un jeu de labyrinthe où l’on incarne un personnage souriant. Le but du jeu est de s’échapper du labyrinthe en ouvrant successivement les 5 portes qui nous séparent de la liberté, et ce en récupérant pour chaque porte une clé de la couleur qui lui correspond.

Une fois échappé du labyrinthe, le joueur est informé du nombre de déplacements effectués. L’objectif est maintenant de s’échapper du labyrinthe en un minimum de mouvements.

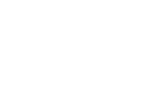
Il existe également une manière cachée de gagner la partie, à vous de la trouver !

## Configuration Requise

* Un ordinateur permettant de lancer l’émulateur DOSBox
* Un clavier
* Un écran

## Instructions d’installation

1. Lancer DOSBox
2. Entrer l’instruction « *mount c C:\[****votre chemin d’installation****]* »
3. Entrer l’instruction « *C:* » pour se déplacer sur le disque C :
4. Entrer l’instruction « *cd [****votre chemin d’installation****]* » pour naviguer jusqu’au dossier contenant le jeu.
5. Entrer l’instruction « *MAZEGAME.COM* » pour lancer le jeu



5

Maze Game – Retro’GameZ

## Comment jouer ?

Pour se déplacer, le joueur peut utiliser deux ensembles de touches selon sa préférence :  Les flèches directionnelles :

* ↑ a
* ↓ b
* ← c
* → d

 Les touches de déplacement habituelles :

* Z (haut)
* S (bas)
* Q (gauche)
* D (droite)

## Support technique

En cas de problème, n’hésitez pas à contacter le support technique à l’adresse suivante :

**296188@supinfo.com**

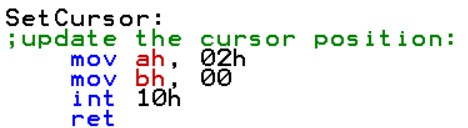
# Documentation technique

## FONCTIONNEMENT

### Déplacement du curseur

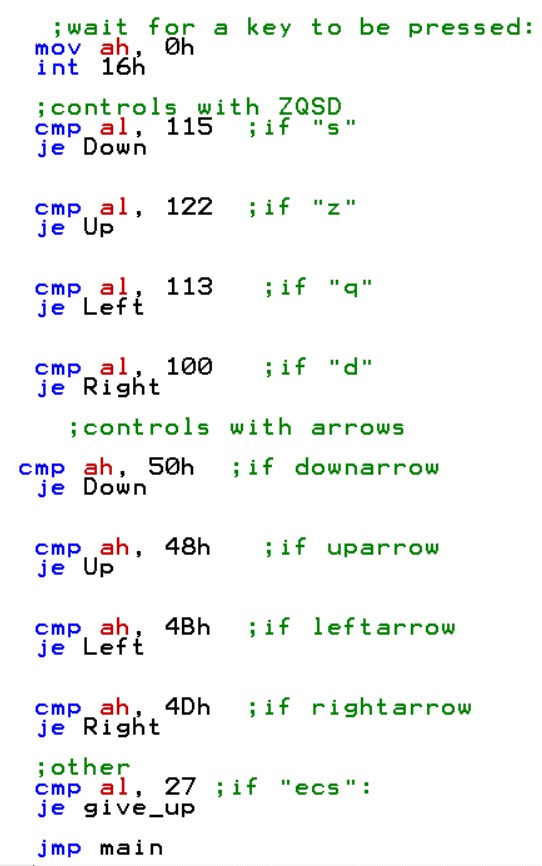
La gestion du jeu est basée sur le système de curseur de l’assembleur 8086. Celui-ci n’est pas visible dans le jeu grâce à la commande “CURSOROFF” présente dans la librairie “emu8086.inc “. Cette dernière est la seule librairie utilisée pour ce programme.

Pour déplacer le curseur, il faut modifier les registres dl et dh qui seront ses coordonnées et appeler notre fonction “SetCursor” qui met à jour la position du curseur.



### Appui sur une touche

Le programme attend la pression d’une touche de déplacement afin de lancer une fonction qui va déplacer le personnage dans la direction souhaitée.



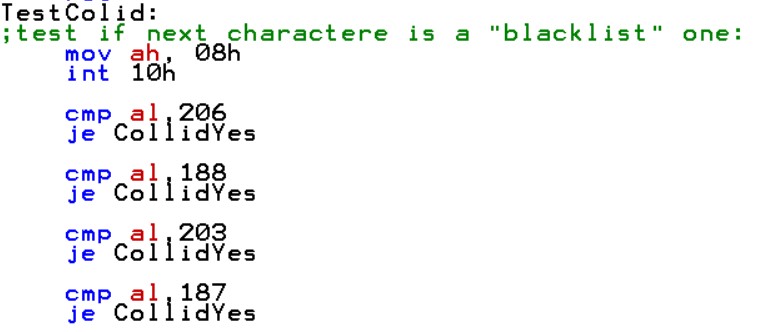
### Déplacement du joueur

Pour effectuer un déplacement la fonction “clear\_player” est appelée pour effacer le joueur. Par la suite, le curseur est déplacé dans la direction choisie et la commande “PRINT ‘:)’” affiche le joueur à sa nouvelle position.



En revanche, avant chaque déplacement, le programme vérifie le contenu de la case dans laquelle le personnage s’apprête à se déplacer :

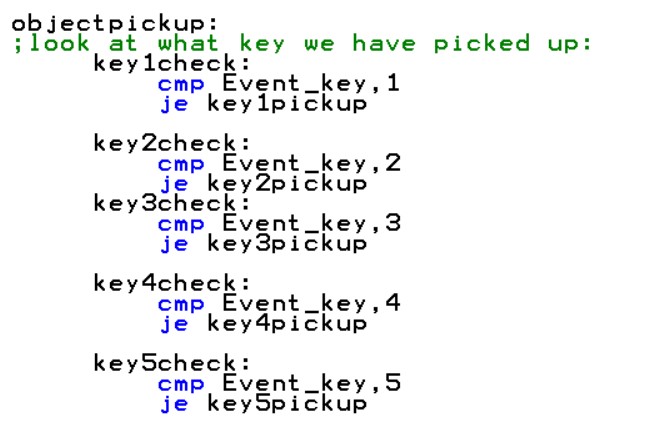
* Si cette dernière contient un caractère représentant l’un des murs du labyrinthe, il empêche le déplacement du personnage.
* Si elle contient un caractère avec lequel on doit interagir (porte, clé…), le programme va exécuter la fonction associée à ce caractère.



*Ici, 206, 188, 203 et 187 sont des caractères représentant des murs.*

### Objets et clés

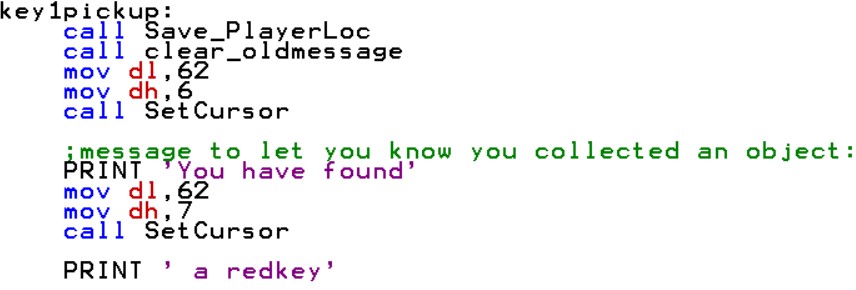
Les clés apparaissant au fur et à mesure de la partie et toujours dans le même ordre. La fonction “objectpickup”, qui est exécutée après un déplacement sur une clé, vérifie dans quelle partie du labyrinthe le joueur se situe et exécute la fonction de la clé correspondante.



Lors du ramassage d’un objets/l’ouverture d’une porte, un message est affiché sous l’inventaire du joueur :



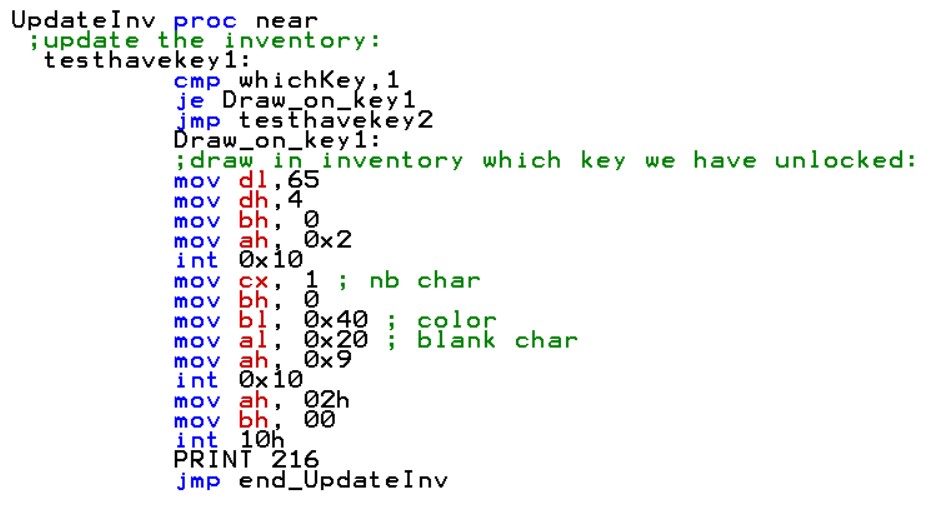
Pour afficher ce dialogue, la fonction “clear\_oldmessage” est appelée. Celle-ci va effacer le dernier message sur l’interface avant d’afficher le texte de remplacement via la commande “PRINT”.



Chaque clé est associée à sa propre fonction. Elles suivent toutes la même procédure :

* Afficher le message d’obtention de la clé
* L’ajouter à l’inventaire du joueur
* Stocker l’information dans une variable

L’inventaire est mis à jour via l’appel de la fonction “UpdateInv”. Cette fonction vérifie quelles clés sont possédées par le joueur et l’affiche dans son inventaire.



### Portes

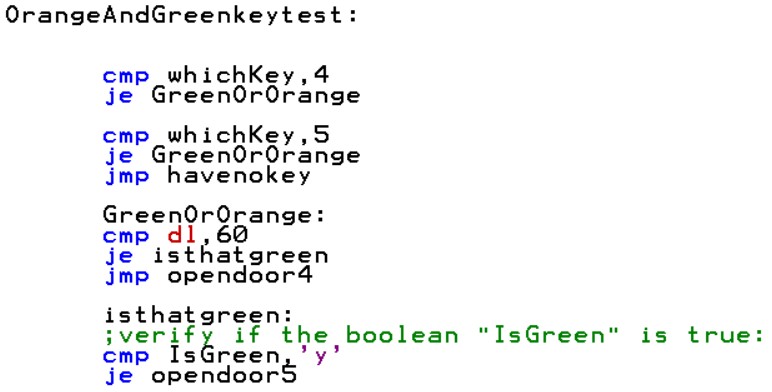
Lors de la collision avec une porte, une première fonction se lance : “keytest”. Elle va tester quelle(s) clé(s) sont dans l’inventaire du joueur.



La méthode de vérification de clé varie selon les portes afin de s’assurer que le joueur ouvre les portes dans l’ordre. Pour les trois premières portes, c’est très simple : si le joueur veut ouvrir une porte, il doit avoir la clé correspondante.

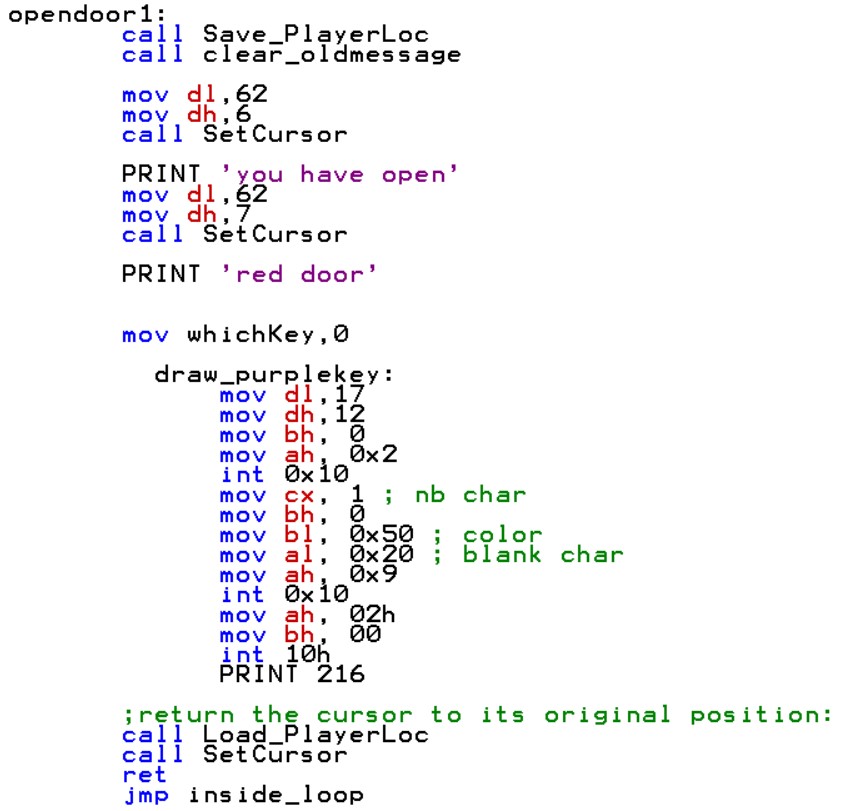
Pour les deux dernières, la logique varie. Ces deux portes devenant accessibles en même temps, il faut s’assurer que le joueur les ouvre dans le bon ordre, orange puis verte. Cette vérification s’effectue via la fonction “GreenOrOrange” qui vérifie la position du joueur.

* S’il se trouve face à la porte verte, la fonction “isthatgreen” vérifie si la clé verte est présente dans l’inventaire. Ceci n’est possible qu’après ouverture de la porte orange : c’est cet « évènement » qui déclenche l’apparition de la clé verte.
* Sinon, le joueur se trouve en face de la porte orange : l’ouverture suit la même logique que les autres portes.



Chaque porte possède sa propre fonction qui effectue les opérations suivantes à son ouverture :

* Afficher un message au joueur à propos de l’ouverture de cette porte  Déclarer via la variable “whichkey” l’utilisation de la clé de la zone précédente  dessine la clé de la zone suivante.



### Score et écran de fin

À chaque déplacement, le programme incrémente une variable. Cette variable est affichée sur l’écran de fin de partie, généré à l’ouverture de la dernière porte, et permet au joueur d’avoir un retour de sa performance.



*Il existe cependant un deuxième écran de fin, caché dans le jeu…*

## AFFICHAGE

Pour l’affichage du jeu, la première solution envisagée a été d’afficher différents « sprites » pour les objets et le personnage :

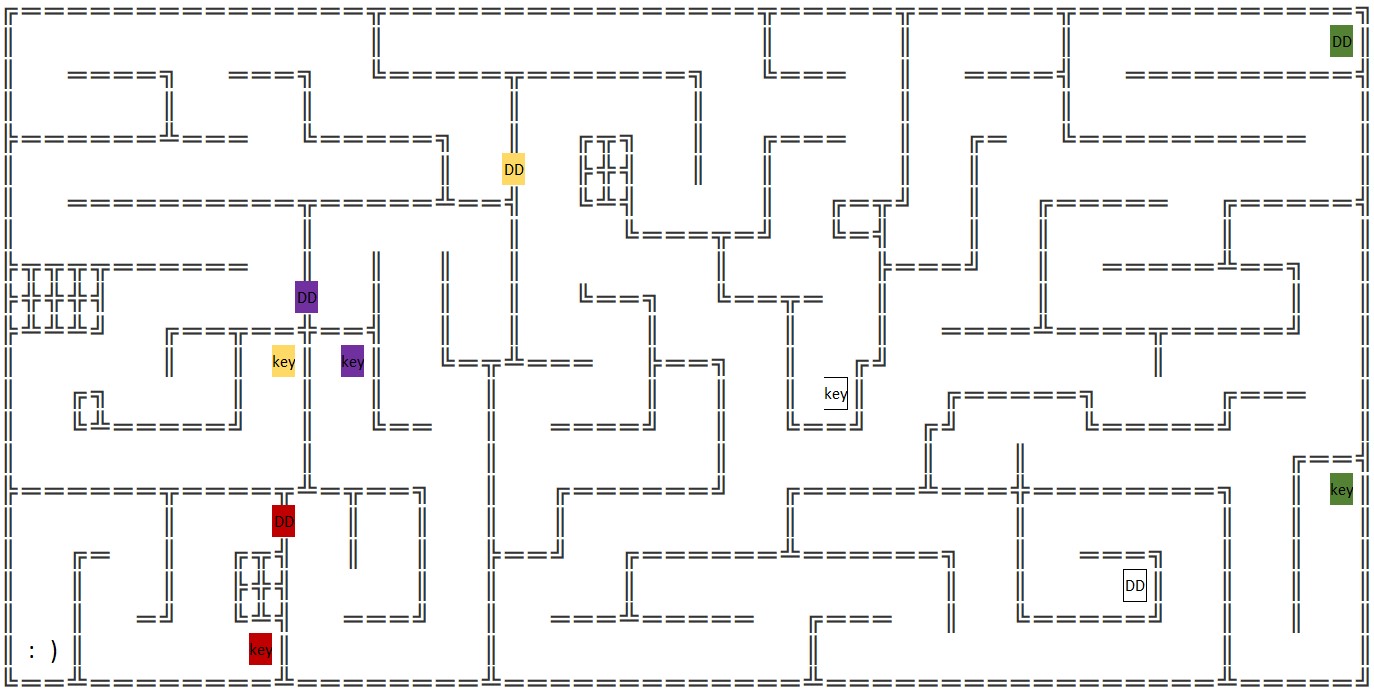


### *Exemples de « sprites » envisagés*

Cette solution a cependant été rapidement abandonnée, au vu de sa complexité d’implémentation et de maintenance.

La deuxième solution, celle-ci adoptée, est de modéliser le labyrinthe en caractères ASCII, afin de faciliter son affichage. L’équipe a donc choisi de réaliser un labyrinthe de dimensions 60x22, afin de tirer pleinement partie des possibilités de l’environnement imposé. Bien que le nombre de couleurs disponibles pour les éléments soit limité par les capacités techniques de l’assembleur 8086, ils possèdent tous leur propre couleur.

Avant d’implémenter le labyrinthe dans le jeu, un plan réalisé à l’aide d’un tableur a permis de le préparer et de pouvoir y apporter toutes les corrections nécessaires



Les caractères ASCII utilisés sont les suivants :

|  |  |  |  |  |  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- |
| ║ | ═ | ╝ | ╗ | ╔ | ╚ | ╠ | ╣ | ╩ | ╦ | ╬ | Ï | ▒ |
| 186 | 205 | 188 | 187 | 201 | 200 | 204 | 185 | 202 | 203 | 206 | 216 | 177 |

Résultat en jeu :

