**2014/2015**

Casimir Raphaël

Champalier Mariane

ECE

ING1 TD2

Projet MERCURY



|  |
| --- |
| **SOMMAIRE** |
|  |
| [I. Introduction 3](#_Toc405149795)  [A) Présentation de l’équipe 3](#_Toc405149796)  [B) Présentation du jeu 3](#_Toc405149797)  [II. Conception du projet 4](#_Toc405149798)  [A) Organisation du groupe 4](#_Toc405149799)  [B) Planning 4](#_Toc405149800)  [C) Analyse Chronologique Descendante 4](#_Toc405149801)  [D) Découpage modulaire 5](#_Toc405149802)  [III. Réalisation du projet 7](#_Toc405149803)  [A) Structures utilisées 7](#_Toc405149804)  [B) Logiciels utilisés 7](#_Toc405149805)  [C) Graphe d’appel 8](#_Toc405149806)  [D) Test réalisés 8](#_Toc405149807)  [E) Difficultés rencontrées 8](#_Toc405149808)  [IV. Conclusion 9](#_Toc405149809) |

# Introduction

## Présentation de l’équipe



Raphaël Casimir Mariane Champalier

L’équipe est composée de Raphaël Casimir et Mariane Champalier, élèves en ING1. La formation du groupe s’est faite en fonction des notes au premier DS d’informatique du premier semestre. La formation des groupes a donc été imposée, malgré tout une très bonne entente nous a permis de nous répartir le travail de façon très naturelle.

## Présentation du jeu

Ce projet a pour objectif le codage en langage C d’un jeu de type Plantamiz ou Bejeweled, dont voici le principe:

Le jeu se présente sous la forme d’une matrice de caractères : M pour masters, S pour slices, O pour origins, P pour processes, F pour functions. L’utilisateur peut se déplacer dans la matrice grâce aux flèches directionnelles, et sélectionner chaque item en appuyant sur la barre d’espace. Il peut ensuite choisir de le déplacer à gauche, droite, en haut ou en bas. Lorsqu’il réalise un alignement de 3 items ou plus, ces derniers disparaissent et les trous sont comblés par chute des autres items par gravité, afin que la grille soit toujours remplie.

L’objectif est de remplir un contrat prédéfini, affiché à l’écran, avec un certain nombre d’items de chaque type à éliminer. Chaque fois que le joueur supprime des caractères, l’écran affiche l’état d’avancement du contrat. Le nombre de coup pour remplir le contrat est limité, et le joueur perd s’il n’arrive pas à remplir le contrat dans le nombre de coups défini.

Le jeu comporte plusieurs niveaux à difficulté croissante : si le joueur parvient à remplir le contrat d’un niveau, il passe automatiquement au niveau suivant.

L’écran affiche également le score du joueur, qui augmente à chaque nouvelle suppression d’items. Selon la forme des figures qu’il réalise et leur taille, le score ajouté peut se voir attribuer un coefficient multiplicateur pour valoriser les suppressions d’items plus importantes. Par exemple, une figure en forme de croix, de T ou de L augmente le score du nombre d’items supprimés multiplié par deux.

Lorsque le joueur désire quitter le jeu, il peut sauvegarder sa progression et son score, et donc reprendre le jeu quand il le désire.

# Conception du projet

## Organisation du groupe

Nous avions chacun commencé à préparer et coder, avant la répartition des élèves en groupes, des fonctions différentes, ce qui nous a permis de rassembler une partie de nos codes. Nous avons néanmoins commencé le travail de groupe sur le code de Raphaël, qui, ayant déjà fait de la programmation auparavant, avait utilisé certaines fonctions plus élaborées telle que l’allocation dynamique pour la création de la matrice. Nous nous sommes donc expliqués nos méthodes respectives, ce qui nous a permis d’avancer rapidement dans la conception du projet.

Nous avons ensuite fait une liste des fonctions à coder, en précisant leurs paramètres et leur rôle, que nous nous sommes départagées.

## Planning

**12 novembre** : Formation des équipes de projet.

**14 novembre** : Nous avons partagé nos codes respectifs précédemment réalisés et nous sommes expliqués les différentes méthodes utilisées.

**17 novembre** : Nous avons réalisé une liste des fonctions à coder et des structures à utiliser, avant de nous départager le travail de manière à peu près égale selon les capacités de chacun.

**20 novembre** : Nous avons réalisé le graphe d’appel du programme, ce qui nous a permis de coordonner le codage et l’appel de nos sous-programmes respectifs.

Nous avons par la suite fait de régulières mises en commun de l’avancement de nos codages.

**24 novembre** : Nous avons commencé à rédiger le rapport de projet.

**2 décembre :** Remise du code et du présent rapport.

## Analyse Chronologique Descendante

**Affichage du menu**

* Choix : commencer une nouvelle partie ou charger une partie sauvegardée

**Affichage de la matrice**

* Génération aléatoire de caractères dans la matrice
* Suppression automatique des suites d’items
* Chute des items par gravité

**Boucle évènementielle**

* Sélection d’un item par l’utilisateur
* Permutation avec un item voisin ou désélection de l’item
* Suppression des suites d’items
* Incrémentation des contrats et du score
* Chute des items par gravité
* Passage au niveau supérieur si tous les contrats sont remplis

**Fin du jeu**

* Demande au joueur d’entrer son pseudo
* Enregistrement du score et de l’état d’avancement des contrats

## Découpage modulaire

Pour que notre code sous le plus clair possible, et afin de pouvoir coder séparément sur des parties différentes du programme, la programmation modulaire s’est imposée à nous. Elle nous a notamment permis de prendre connaissance de nos codes respectifs sans trop de difficultés, puisqu’étant plus claire qu’un code dense composé d’aucun ou peu de sous-programmes.

Voici la liste des sous-programmes utilisés et leurs rôles dans le programme :

**Header. h :** contient les structures utilisées et les prototypes des sous-programmes.

**Void menu :** affiche le menu principal à l’écran au lancement du jeu.

* Sortie : Affichage du menu
* Entrée : Choix de de l’utilisateur

**Void initGrid :** crée l’emplacement mémoire de la matrice, remplis la grille à partir de caractères aléatoires et initialise les contrats à remplir.

**Void printGrid :** affiche une unique case de la grille.

* Sortie : Affichage de la matrice à l’écran

**Void game :** boucle évènementielle du jeu.

* Entrée : Obtention des choix de mouvements du joueur (droit, gauche, haut, bas) grâce aux flèches directionnelles

**Void switchLetter :** permute l’item sélectionné par le joueur avec l’un des items voisins.

* Entrée : Obtention de la direction dans laquelle le joueur souhaiter permuter l’item sélectionné, ou désélection de l’item en cas d’appui sur la barre d’espace

**Void searchPattern :** recherche les figures présentes dans le jeu avant et après l’intervention du joueur.

**Void gravity :** fait chuter les caractères pour combler les trous lorsque des items sont supprimés

**Char fillAlea :** renvoie un caractère aléatoire à la procédure gravity, qui le place à la première ligne lorsque les caractères sont descendus pour combler un trou.

**Void scores :** incrémente les points du joueur

* Sortie : Affichage des évolutions du score à l’écran

**Void contrat :** met à jour les contrats selon les figures supprimées par le joueur.

**Void menuPause :** affiche le menu pendant le jeu lors de l’appui sur la touche Echap.

* Sortie : Affichage du menu à l’écran
* Entrée : Choix de l’utilisateur

**Void saveAll :** enregistre le pseudo, le niveau, les scores et de la progression du joueur.

* Entrée : Obtention du pseudo du joueur
* Sortie : Affichage des emplacements mémoire disponibles à l’écran

**Void loadAll :** charge l’une des parties sauvegardées par le joueur.

* Sortie : Affichage des différentes parties sauvegardées
* Entrée : Choix de la partie à charger

**Void gotoligcol :** place le curseur à la ligne et à la colonne souhaitées par l’utilisateur.

* Sortie : Affichage du curseur sur une case de la matrice à l’écran

**Void initLevel :** initialise le niveau du jeu.

* Sortie : Affichage du niveau à l’écran

## Scénario du jeu

Nous avons choisi de créer un scénario pour notre jeu :

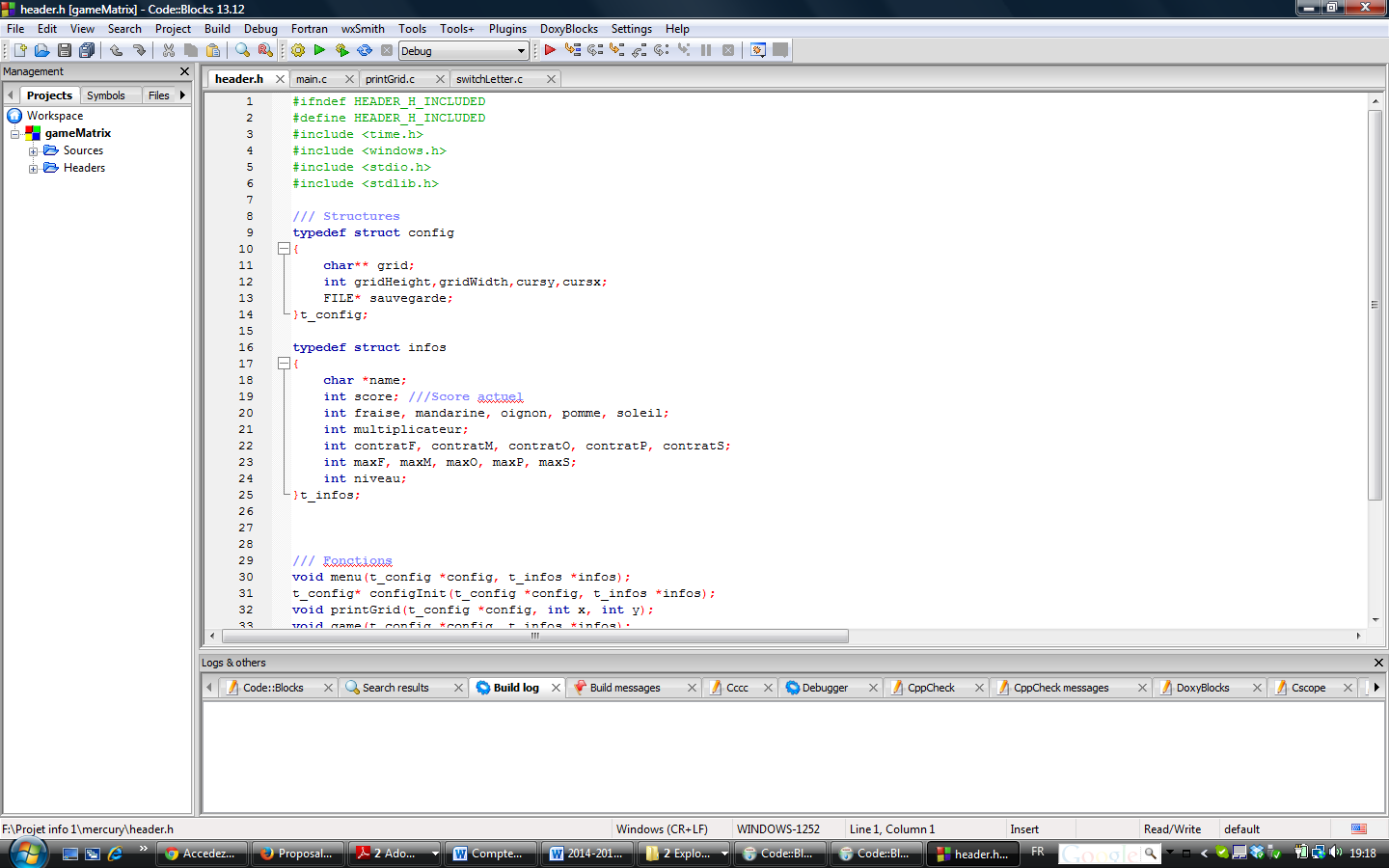
Inspiré du film « Code Mercury », où un enfant est capable de cracker un code hautement technique supposé indéchiffrable, notre jeu porte le nom de codeMercury. Ainsi, les caractères affichés par la matrice ne correspondent pas à des objets, mais à des éléments informatiques techniques : M pour masters, S pour slices, O pour origins, P pour processes et F pour functions. L’objectif du joueur est donc de « vaincre » le codage de la matrice, en arrivant au bout du jeu, et en supprimant des caractères par alignement.

Cela a pour but d’ajouter une histoire au jeu, comme pour un jeu d’aventure, afin que le joueur prenne plus de plaisir à jouer.

# Réalisation du projet

## Structures utilisées

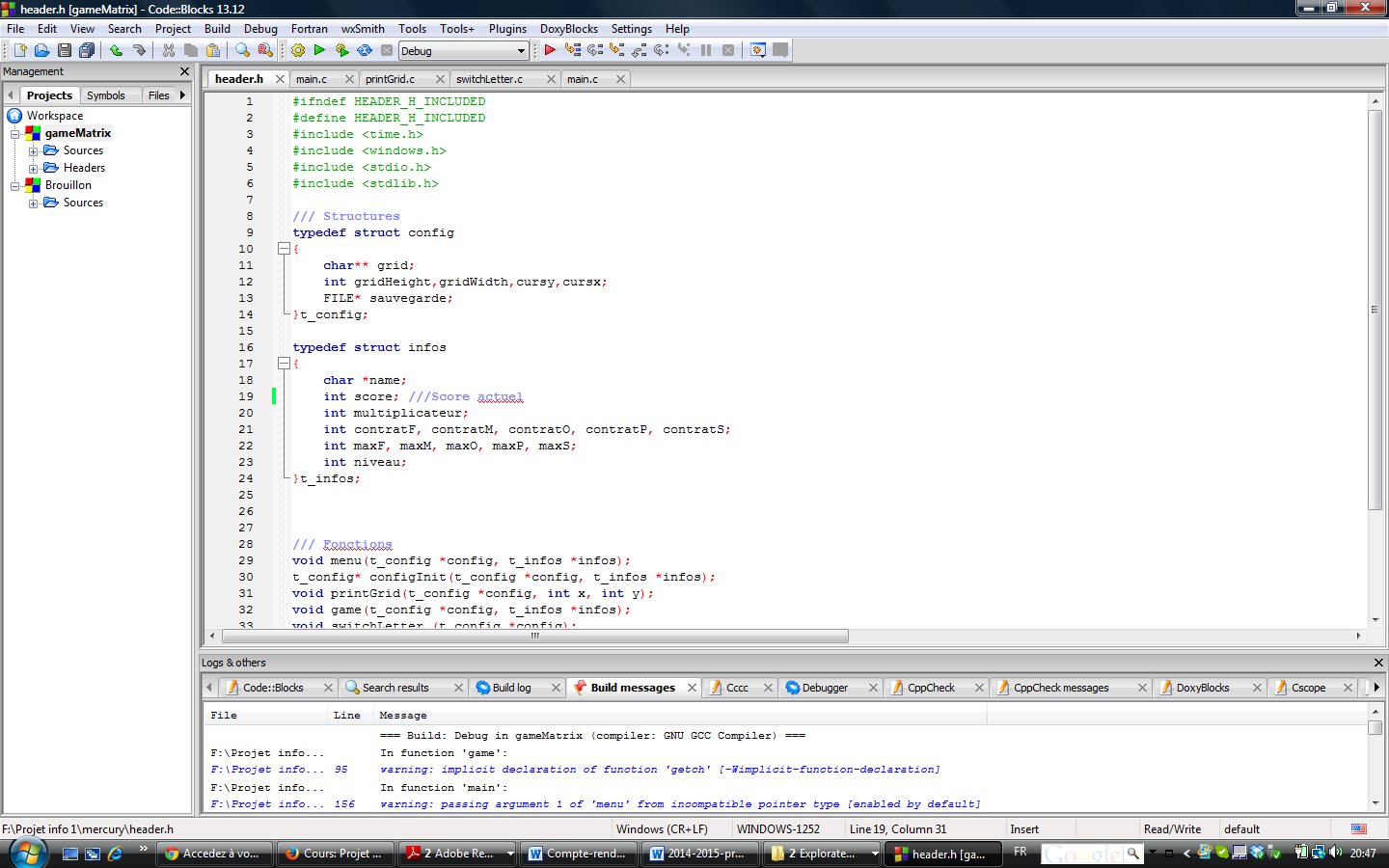
Nous avons choisi d’utiliser des structures pour faciliter le transfert de paramètres entre les différents sous-programmes. Nous utilisons deux structures différentes dans l’ensemble du sous-programme : t\_config et t\_infos.

La structure t\_config contient un pointeur sur la grille de jeu, envoyé à chaque sous-programme ayant besoin d’interagir directement avec la matrice.

Elle comprend également deux variables correspondant à la hauteur et la largeur de la matrice, et initialisées au début du jeu dans le sous-programme configInit.

Les variable cursx et cursy sont utilisées dans le programme pour désigner une ligne et une colonne de la matrice, cursx désignant les colonnes et cursy les lignes.

Enfin, elle contient également un pointeur sur le fichier de sauvegarde du jeu, qui enregistre le niveau, le score et l’avancement du joueur lorsqu’il désire quitter la partie.

La structure t\_infos contient les informations nécessaires à la sauvegarde de l’avancement du joueur et au comptage des points. Elle contient donc un pointeur sur le pseudo du joueur pour le programme de sauvegarde, et un entier recevant le score chaque fois qu’il est modifié.

Les entiers contratF, contratM, contratO, contratP, contratS servent à voir l’avancement du joueur par rapport au contrat qui lui a été affecté. Ces contrats sont indiqués par d’autres variables, maxF, maxO, maxP, maxM et maxS.

L’entier multiplicateur contient le coefficient multiplicateur du score en fonction du type de figure réalisé par le joueur (par exemple, le score ajouté est multiplié par 2 si la figure est une croix, un T ou un L, et 1 s’il s’agit d’un alignement simple).

Enfin, l’entier niveau prend en compte le niveau du joueur à un instant donné ou au moment où la sauvegarde a eu lieu.

## Logiciels utilisés

Le codage en langage C du projet s’est fait sur le logiciel Code::blocks en mode console.

La mise en commun de nos codes a été réalisée grâce au logiciel GitHub, qui nous a permis de compiler l’ensemble des nouveautés progressivement apportées au programme.

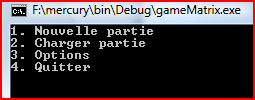
## C:\Users\champalier\Pictures\Diagramme%20appel.pngGraphe d’appel

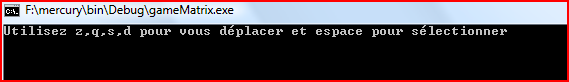
Voici le graphe d’appel des sous-programmes utilisés. Les cases rectangulaires indiquent un sous-programme en appelant d’autres, tandis qu’une case ovale indique un sous-programme ne nécessitant aucun appel pour son déroulement. Les lignes en pointillés indiquent que le lien entre les programmes ne se fait pas dans notre programme à l’heure actuelle, ou bien très rarement. Par exemple, le sous-programme loadAll appel à configInit uniquement si un fichier de sauvegarde est corrompu. D’autre part, le menuOptions n’ayant pas pu être codé, les programmes menu et menuPause auxquels il aurat dû être relié ne peuvent pas faire appel à lui.

## Test réalisés

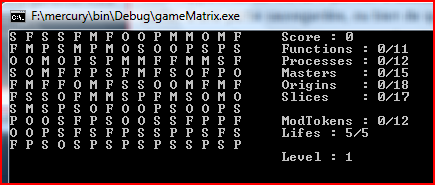
Nous avons effectué des tests à chaque ajout d’un sous-programme, et pendant les programmations des sous-programmes pour être sûrs que le code continue à fonctionner à chaque étape du codage. Cela nous a permis d’éviter au maximum les problèmes lorsque nous avons compilé nos codes et rassemblé nos sous-programmes.

Voici les tests finaux réalisés :

* Voici l’apparence du menu. Il permet d’accéder à une nouvelle partie, à une partie sauvegardée, ou bien de quitter le jeu. Il ne donne pas accès aux options comme cela est indiqué, car nous n’avons finalement pas eu le temps de rajouter les options que nous envisagions à l’origine.



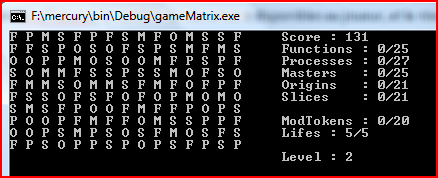
* Le jeu affiche ensuite les touches nécessaires au déplacement du joueur dans le jeu.



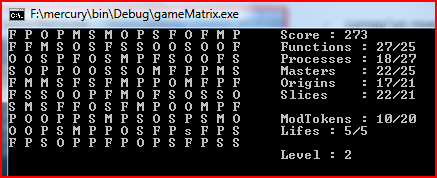
* La matrice s’affiche, puis vide progressivement le tableau des figures existantes, et comble les trous par gravité. Le joueur obtient alors un tableau vide de toute figure, dans lequel il peut se déplacer. Le score et les contrats s’affichent à gauche de la matrice ; sont également indiqués le nombre de mouvement limite pour remplir les contrats(ModTokens), le

nombre de vies encore disponibles au

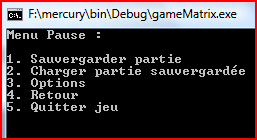
joueur, et le niveau du jeu.



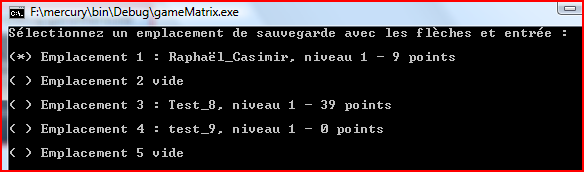
* Lorsqu’un niveau est complété, on accède automatiquement au niveau supérieur. Le nombre d’items à supprimer augmente, ce qui accroit la difficulté. Le score affiche alors le nombre d’items supprimés au niveau précédent.



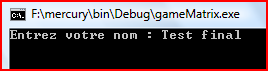
* Tant que tous les contrats n’ont pas été complétés, le niveau ne change pas. Le nombre d’items supprimés peut dépasser les contrats demandés.



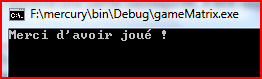
* En cas d’appui sur la touche Echap pendant le jeu, le Menu Pause s’affiche, et le joueur a la possibilité de sauvegarder sa partie, de charger une autre partie sauvegardée, de retourner au jeu ou de quitter le jeu.



* Lorsque le joueur appuie sur la touche 1 pour sauvegarder sa partie dans le menuPause, un nouveau menu s’affiche. Le joueur a la possibilité de choisir l’emplacement mémoire dans lequel il veut enregistrer sa partie. Il peut se déplacer avec les flèches directionnelles et appuyer sur entrée pour choisir son emplacement. La sauvegarde enregistrera le pseudo du joueur, son niveau, son score et le nombre de vies qu’il lui reste. Lorsque le joueur reprend une partie sauvegardée, il doit recommencer son niveau.



* Le joueur rentre ensuite son pseudo. Les espaces seront remplacés en mémoire par des tirets.



* Si le joueur décide de quitter le jeu, un message de fin apparait et le jeu se termine.

La phase de test finale n’a rapporté aucune erreur majeure du code. Il fonctionne correctement, et il est tout à fait possible d’y jouer.

## Difficultés rencontrées

Le codage du sous-programme searchPattern de recherche des figures dans le programme a été particulièrement problématique, car étant assez répétitif de petites erreurs concernant les numéros de lignes ou colonnes se glissaient dans le code de bloquaient l’élimination de certaines figures. De plus, le codage de ce sous-programme n’a pas été idéal car la méthode utilisée n’est pas la plus pratique : elle consiste à énumérer tous les cas possibles dans l’ordre. Néanmoins, après avoir corrigé les problèmes, la fonction a correctement joué son rôle.

Le sous-programme d’échange de lettres switchLetter a également été difficile à coder. En effet, nous avons choisi d’espacer les items les uns des autres en ajoutant un espace entre chacun d’entre eux. Cela a malheureusement eu des conséquences sur le placement de la fonction gotoligcol et le sous-programme d’affichage printGrid, et il a fallu multiplier le numéro de colonne reçu par printGrid par 2 afin qu’il affiche bien la bonne case à la bonne colonne.

La fonction de sauvegarde saveAll nous a pris du temps à coder et à tester. En effet, nous ne connaissions pas toutes les fonctions nécessaires à la sauvegarde de fichiers, et nous avons dû faire quelques recherches complémentaires. De plus, cette étape demande une grande précision pour avoir accès aux fichiers contenant la sauvegarde et les relier au programme. La procédure loadAll, allant de pair avec saveAll, a quant à elle été plus simple à coder car elle dépend entièrement de la façon dont a été enregistré le fichier, donc une fois que saveAll a été opérationnel, loadAll a également fonctionné.

Enfin, la bouche évènementielle game s’est construite petit à petit, puisqu’elle appelle les sous-programmes les uns après les autres. A chaque nouveau codage d’un sous-programme, nous devions donc mettre à jour la boucle, et il arrivait que nos modifications combinées posent de légers conflits, qu’il a été plutôt facile de résoudre.

# Conclusion

Le codage de ce programme nous a permis de faire une application directe des connaissances acquises en cours. Cela a constitué un entrainement formateur au codage et à la réalisation d’un projet dans un laps de temps relativement court et nous a permis de mieux approfondir le développement en langage C.

Au niveau collectif, cela nous a permis de réaliser l’importance de la réalisation d’un projet en groupe. Nous avons ainsi pu mettre en commun nos connaissances et prendre conscience de nos compétences respectives dans ce domaine. Nous avons également été amenés à mettre en place une démarche de projet, en définissant différentes étapes dont la conception et la réalisation, et à organiser notre travail afin d’affecter à chacun des tâches très précises. Une étape importante a été la phase d’intégration des développements afin de tester que l’application fonctionnait correctement, et présentait l’ensemble des fonctionnalités attendues.

La rédaction de ce rapport a permis de synthétiser notre organisation et notre approche du projet et a permis de fait ressortir les atouts d’un projet mené en équipe.