Compte rendu du projet « blocus »

Par Yann Pascoet (groupe 4) et Victorien Blanchard (groupe 1)

Table des matières

[I/Introduction 2](#_Toc469585971)

[II/Structure du programme 2](#_Toc469585972)

[a)3 étapes clés 2](#_Toc469585973)

[1 – L’initialisation 2](#_Toc469585974)

[2 – Affichage 3](#_Toc469585975)

[3 – Calcul d’un tour 3](#_Toc469585976)

[b)Fonctionnalités supplémentaires 4](#_Toc469585977)

[1)Mode histoire 4](#_Toc469585978)

[2)Succès 4](#_Toc469585979)

[III/Explication des tours de jeux 4](#_Toc469585980)

[IV/ Découpage du code 6](#_Toc469585981)

[V/ Conclusions personnelles 7](#_Toc469585982)

[Victorien 7](#_Toc469585983)

[Yann 7](#_Toc469585984)

# 

# I/Introduction

Le jeu qui va vous être présenté dans ce dossier a été développé dans le cadre du projet tuteuré du premier semestre. Il s’agit d’un jeu dont le but consiste à bloquer l’adversaire, c’est-à-dire de faire en sorte à ce que l’adversaire ne puisse plus bouger. Le jeu se déroule au tour par tour sur une grille dont la taille est choisie par l’utilisateur. Chaque tour, chaque joueur peut se déplacer vers une case adjacente et boucher une case de la grille, de sorte que la case ne soit plus accessible. La partie s’arrête quand un joueur doit déplacer son pion et que celui-ci ne peut plus se déplacer vers une case adjacente.

L’utilisateur a donc le choix entre 2 modes de jeux principaux :

- le mode 1 joueur où il jouera contre une intelligence artificielle (IA) qui se déplacera de façon aléatoire et qui bloquera chaque tour une case autour du joueur ennemi.

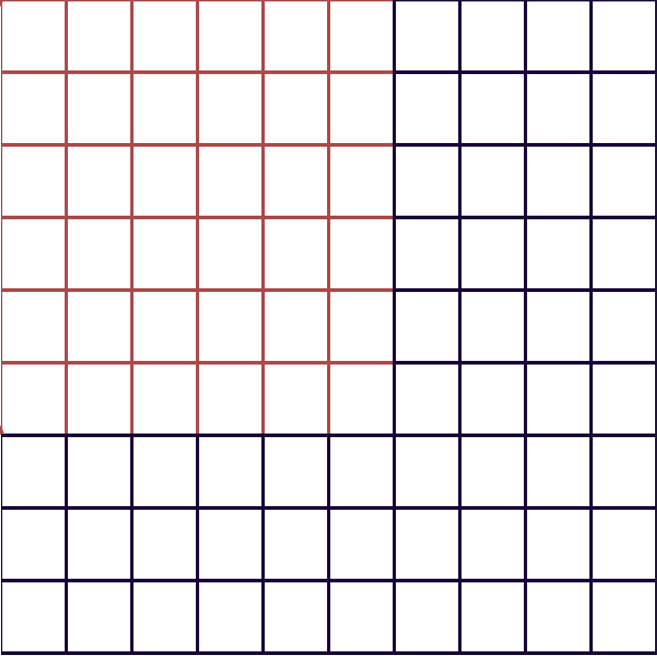
- le mode 2 joueurs où il jouera contre un autre utilisateur.

Le jeu dispose d’un mode supplémentaire : le mode histoire, dans lequel l’utilisateur devra se battre contre des oppresseurs en tant que représentant des mineurs d’une mine remplie de richesses.

# II/Structure du programme

## 3 étapes clés

### 1 – L’initialisation

Afin de réaliser ce projet, nous avons eu besoin de créer le moteur du jeu, composé de plusieurs fonctions assez simples. Pour ce qui est de la partie purement logicielle, le jeu se déroule dans une fenêtre de taille 800x800 dans laquelle le joueur choisit le mode de jeu auquel il souhaite jouer, ainsi que la taille du plateau. Une fois ce choix fait, la partie s’initialise.

***Schéma 1***

***En rouge, le tableau tel qu’il est compris par le programme.***

Pour initialiser une partie, il nous a suffi d’initialiser un tableau à double dimension de taille 9, qui est la taille maximum que l’utilisateur peut choisir, et de le remplir de 0 à chaque nouvelle partie, afin de le réinitialiser. Les fonctions du reste du programme se servirons de ce tableau, mais prendront en compte la taille voulue par l’utilisateur. Ainsi, si l’utilisateur choisit une taille de 6, le tableau sera initialisé comme ayant une taille de 9, mais le reste du programme considérera que le tableau ne fait que 6x6 cases, comme montré sur le ***schéma 1***: la partie rouge représente le tableau tel qu’il sera lu par le programme. A l’affichage, c’est la taille des cases qui s’adaptera en fonction de la taille de la fenêtre et de la taille de la grille, afin de remplir la fenêtre. Cette même taille est calculée par une fonction lors de l’initialisation du plateau de jeu.

### 2 – Affichage

Après l’initialisation du tableau nous avons créé une fonction qui prend les informations de l’initialisation du jeu : la taille de la grille, et va dessiner le jeu sur un écran virtuel qui sera ensuite copié sur l’écran principal afin d’éviter des problèmes d’affichage. Le fonctionnement sera détaillé plus loin.

Pour calculer chaque image, on affiche tout d’abord le fond du plateau (l’image est la même pour tous les modes, la fenêtre ayant une taille constante de 800x800), puis la grille apparaît au-dessus du fond, le nombre de cases et leurs largeurs sont calculés en fonction de la taille de grille, ce qui a pour effet de tracer un segment tous les x pixels et ce x fois.

### 3 – Calcul d’un tour

Une fois la partie initialisée, on peut commencer la partie. Une partie est une fonction à laquelle on donne le nombre de joueur IA et humains. La partie crée alors des joueurs qui seront caractérisés par : leur numéro, un indicateur afin de savoir s’il s’agit d’une IA ou non, et enfin du nombre de cases libre autour d’eux. Lors de la partie, on calcule chaque action de chaque utilisateur : son déplacement, les emplacements qu’il bouche, ses clics de souris. Après chacune de ces étapes, on recalcule le nombre de cases libres restantes autour du joueur actuel, mais aussi du joueur suivant. Ainsi, lorsqu’un joueur doit se déplacer, si le nombre de cases libre est de 0, la partie est terminée.

La partie ne devant être jouée qu’à la souris, il nous faut définir quelques fonctions de base avant de continuer.

* Il faut calculer la case sur laquelle l’utilisateur a cliquée en fonction de la taille d’une case calculée à l’aide de la fonction précédente.
* Il faut vérifier que la case cliquée par l’utilisateur est bien libre, c’est-à-dire que l’utilisateur ne clique pas autre part que sur une case libre.
* Il faut vérifier que l’utilisateur ne tente pas de se déplacer trop loin si la phase de jeu est une phase de déplacement.

## Fonctionnalités supplémentaires

### Mode histoire

Nous avons intégré au jeu un mode histoire contenant deux chapitres : un premier où l’on joue contre deux IA et un second où l’on joue contre une seule IA qui représente un boss. La progression est sauvegardée si l’on gagne un chapitre, et on peut la réinitialiser dans le menu. Le déroulement d’une partie est géré de la même façon qu’une partie standard, à l’exception près que le jeu doit gérer deux IA.

### Succès

Des succès ont été ajoutés pour le plaisir de jeu, à chaque lancement du jeu on lit dans un fichier les succès débloqués, on peut les réinitialiser en même temps que la progression dans le mode histoire depuis le menu. Chaque fin de partie vérifie si les succès sont débloqués et s’ils doivent être débloqués le cas échéant. On peut d’ailleurs vérifier les succès déjà débloqués à chaque visite du menu.

# III/Explication des tours de jeux

Afin d’expliquer au mieux le fonctionnement des tours de jeu, nous allons prendre un exemple de deux tours.

Image d’un jeu en cours

Chaque tour, si le joueur actuel n’est pas une IA, le programme attend un clic de souris. Autrement, l’IA joue toute seule avec un temps de pause d’une demie seconde entre chaque action, dans un souci de visibilité.

Une fois ce clic réalisé, on examine la phase de jeu actuelle, ici nous sommes à une phase de déplacement, comme écrit dans le rectangle informatif en bas de l’écran de jeu.

On agit alors on conséquence. Lorsque la phase de jeu est une phrase de déplacement, le programme attend que l’utilisateur clique sur le terrain. Lors de chaque phase, quel que soit le soit le type de joueur (humain ou IA), la case cliquée est analysée et le programme détermine si la case cliquée est « valide », c’est-à-dire si elle a passé des tests. En l’occurrence ici, le programme vérifie si la case cliquée se situe à une case de distance maximum de la position du joueur, et que cette case est libre.

*Comment savoir si une case est libre ?*

*Grâce aux fonctions présentées dans la partie précédente, on peut déterminer la case sur laquelle le joueur a cliqué. Lorsque le tableau est vide, celui-ci est remplis de « 0 ». Ensuite, lors du placement des joueurs, le joueur 1 se voit attribuer le numéro « 4 » dans le tableau, et les autres joueurs se voient attribués le numéro « 8 ». Chaque case cochée sera caractérisée par un « 1 », jettes un coup d’œil au schéma 3 !*

*Ainsi, on peut aisément déterminer si une case est libre ou non en vérifiant si la valeur de la case est bien « 0 ».*

*Et si la case ne respecte pas les conditions ?*

Si la case ne respecte pas les conditions, on attend un nouveau clic. Si le joueur actuel est une IA, le programme tente de déplacer le joueur IA de façon aléatoire, tout en respectant les règles. Si le déplacement généré correspond à une case qui n’est pas « valide », le programme génère un nouveau déplacement, jusqu’à ce que la case choisie soit valide. En revanche, à l’inverse, on effectue le déplacement si la case est valide, et on affiche à nouveau le plateau de jeu, afin d’afficher le déplacement. Pour cela, on affiche le morceau de terrain sur lequel se situait le joueur à son ancienne position, et on affiche le joueur à son nouvel emplacement.

Une fois la case validée, et le déplacement réalisé, c’est-à-dire qu’on supprime le joueur du tableau, et on le remet à la case voulue. On passe alors à la phrase de jeu suivante, qui est la phase où le joueur doit cocher une case. Lors de cette phase, la case sélectionnée devra également être « valide », et donc passer certains tests. Dans cette phase de jeu, la case sélectionnée devra être une case totalement libre. Encore une fois, si la case sélectionnée n’est pas libre, on attend un nouveau clic jusqu’à ce que le joueur clique sur une case libre. Pour ce qui est de l’IA, elle cherche tout d’abord à boucher une case autour du joueur humain de façon aléatoire. Si le joueur humain est dans une situation où il ne peut pas se déplacer, l’IA coche alors une case aléatoire libre sur le plateau.

Une fois la case sélectionnée cochée, on revient à la phase déplacement, et on change de joueur actuel. A la fin de chaque phase de jeu, on recalcule le nombre de cases libres du joueur qui va jouer le tour suivant. Si nous sommes dans une phase de déplacement et que le joueur actuel ne peut plus se déplacer d’après les calculs, on n’attend pas de clic et on arrête la partie pour afficher que ce joueur a perdu.

*J’ai vu qu’il y avait deux IA dans le mode histoire qui jouaient contre nous. Mais alors, si l’une d’entre elles est battu, on arrête le jeu ?*

*Eh bien non ! Certaines fonctionnalités ont été rajoutées, afin de prévoir plusieurs IA dans une même partie !*

En effet, lorsque le jeu comporte au moins deux IA et un joueur humain, les conditions de « game over » ne sont pas les mêmes. Lorsqu’une IA est morte, et que l’on détecte qu’une autre IA est toujours en vie, le jeu continuera.

Schéma 4 ci contre

Pour mieux comprendre, prenons un exemple (voir schéma 4 ci-contre) : le joueur 1 est humain et il vient de cocher une case. Le programme vérifie donc le nombre de cases libres pour le joueur suivant. Celui-ci détecte que l’IA qui était censé jouer le tour suivant a déjà perdu, mais il détecte également qu’il reste une autre IA en vie. Le programme décide alors que le joueur suivant ne sera pas

le joueur 2 comme prévu, mais le joueur suivant, soit le joueur 3.

A tout moment, le joueur peut quitter la partie en appuyant sur la touche « ECHAP ».

# IV/ Découpage du code

Afin de découper efficacement le code, nous l’avons séparé en plusieurs fichiers qui correspondent aux étapes essentielles du code, c’est à dire qu’on a un fichier pour la boucle de jeu (partie.c) , un qui gère toute la partie affichage (affichage.c), et deux qui font les calculs, un pour les joueurs humains (deroulement.c), et un pour l’IA (ia.c).

Nous avons également découpé les fichiers selon les différents écrans de jeu. Ainsi nous avons également un fichier menu.c, et un main.c.

Quelques fichiers supplémentaires ont été rajoutés pour les fonctionnalités qui ont été rajoutées en plus de la consigne de base. Nous avons donc un fichier achievements.c dans lequel sont stockées les fonctions qui s’occupent de la gestion des succès; et un fichier histoire.c qui permet de la gestion du mode histoire.

Le fichier partie.c est le fichier “principal” du jeu. C’est de celui-ci que l’on se sert pour jouer une partie. C’est également celui-ci qui se sert de presque tous les autres fichiers : il se sert du fichier qui gère l’affichage, mais aussi de celui qui gère les calculs et l’initialisation.

Le fichier main.c permet d’avoir un aperçu global des écrans de jeux en cours. Il contient la boucle de jeu globale. C’est dans ce fichier que l’on appelle le menu qui permet à l’utilisateur de choisir un mode de jeu, qui est ensuite utilisé par la main pour lancer la partie correspondant au choix.

Le fichier histoire.c ouvre un fichier “save” dans lequel il va lire la sauvegarde qui va lui permettre de déterminer quel chapitre de l’histoire lancer en fonction des chapitres déjà gagnés.

Le fichier achievements.c ouvre un fichier contenant des informations à propos des succès, le lis et exécute ce qu’on lui demande de faire. Presque tous les fichiers utilisent ce fichier achievements.c, car on peut avoir besoin d’afficher les succès pendant une partie, ou encore d’informer l’utilisateur qu’il a débloqué un succès.

# V/ Conclusions personnelles

## Victorien

Ce projet a pour moi été très intéressant au sens où je m’en suis servis afin d’appliquer directement les notions de cours tel que la gestion des fichiers (dans les fichiers concernant les sauvegardes), ou encore les structures. Cela permet aussi de développer son esprit « algorithme », car ce projet demande une réelle rigueur algorithmique (que j’ai parfois bafouée je dois l’admettre, oh vous savez à 1h du matin on sait plus bien ce qu’on fait…). Le vrai défi a été de créer une boucle de jeu qui puisse gérer plusieurs IA, et de faire une structure de joueur utilisable par la boucle de jeu. En résumé, ce qui est à mon sens le plus compliqué est de penser le programme pour qu’il fonctionne dans tous les cas : j’aurai pu créer un mode où deux IA se combattent bêtement sans même avoir à toucher à la boucle de jeu.

Pour conclure, je trouve que ce genre de petit projet apporte un vrai plus à notre formation, il nous permet de travailler à fond sur des notions qu’on aurait un peu survolées en cours.

## Yann

J’ai trouvé ce projet assez enrichissant, dans le sens où c’était une première occasion de mettre en accord un grand nombre de notions rencontrées jusqu’à présent, mais aussi un certain défi à réaliser ainsi qu’un moyen d’expression, d’une certaine façon. L’intitulé du projet offrait assez de liberté pour qu’on puisse prendre plaisir à réaliser un jeu selon nos préférences, mais demandait aussi une certaine maîtrise technique sous certains aspects, ce qui le rendait plutôt intéressant à réaliser.

Puisqu’on était en binôme, cela nous a aussi permis d’apprendre à nous organiser pour travailler en équipe, surtout pour le partage de fichier et la répartition des tâches (et de prendre un petit peu d’avance sur le TP de git, hehe).

J’ai donc pris beaucoup de plaisir à réaliser ce projet.