Groupe : Les laitiers

Allemand Valentin
Denis Thomas
Fayet Fabio
Arrahmani Amin

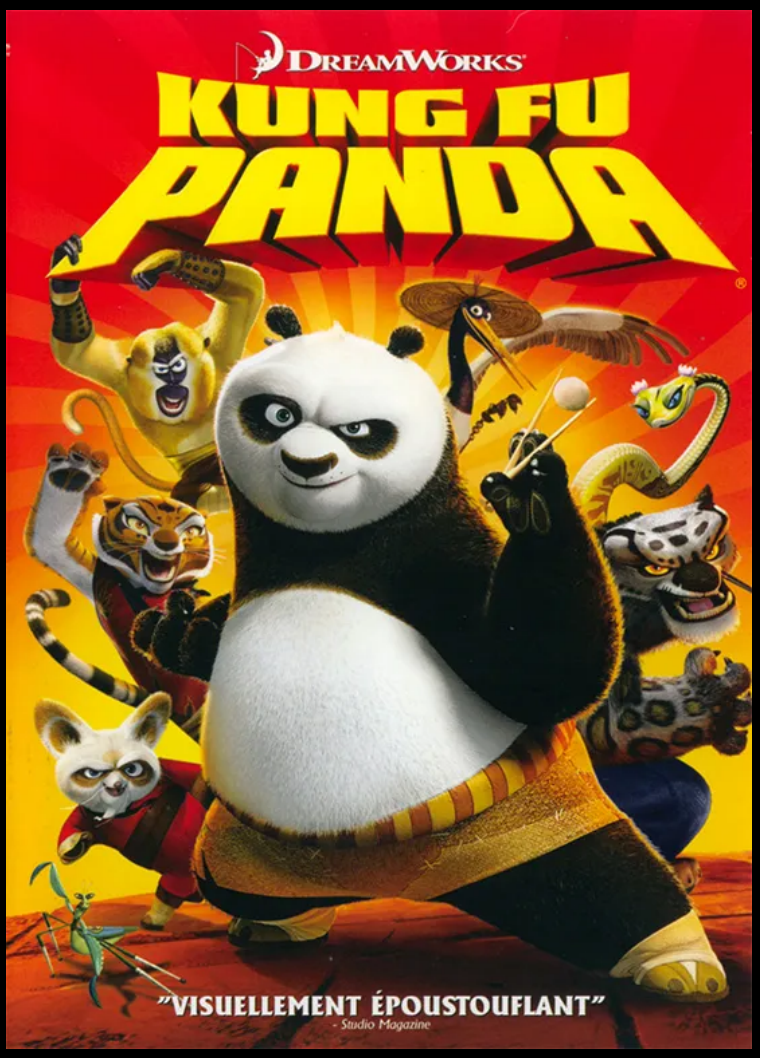
Document d'architecture

Projet: BomberMonkeys

1°) Présentation du projet

1° Introduction

Nous avons choisi pour notre groupe la réalisation d'un jeu BomberMan en 2d. Nous avons également pris comme thème pour revisiter le BomberMan de nous inspirer du célèbre film "Kung-Fu-Panda". Les graphismes seront donc tous tirés de ce film.



2° Objectif

Pour considérer le projet comme réussi, il est primordial de définir certains points qui seront obligatoires lors du rendu. Premièrement, le jeu doit être fonctionnel, que ce soit en 1 contre 1 où contre un ordinateur qui simulera le deuxième joueur. Il doit être également possible de sauvegarder/charger une partie et reprendre le cours de celle-ci. Le projet doit pouvoir indiquer à la fin de la partie qui est le vainqueur et offrir la possibilité de pouvoir rejouer. Concernant les solutions de mise en place, les groupes étaient assez libres. Pour notre part, nous avons choisi d'utiliser SDL2 pour les graphismes dans un souci de performance/utilité. Nous avons également utilisé de nouvelles bibliothèques telles que pthread dont nous expliquerons le fonctionnement en temps voulu dans le document.

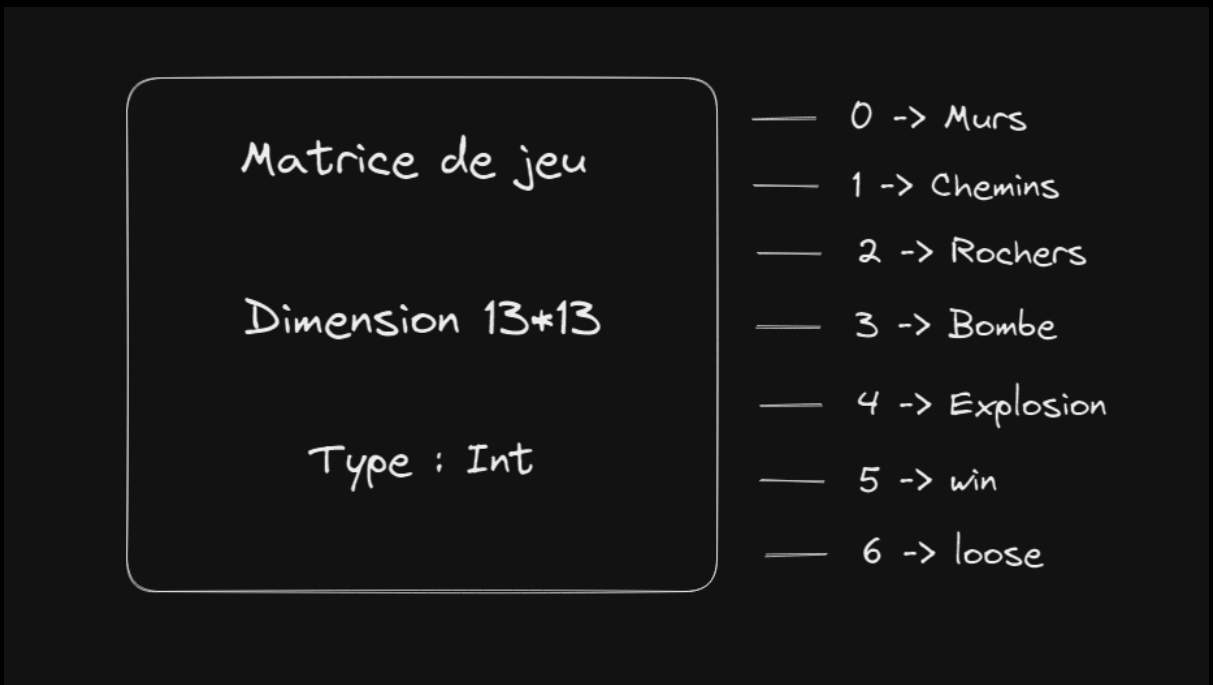
2°) Découpage de l'équipe pour les rôles

Notre équipe comporte 4 personnes: Denis Thomas le chef de projet, Allemand Valentin, le développeur graphique. Fayet Fabio, le testeur et enfin Arrahmani Amin le développeur moteur.

3°) La partie Moteur

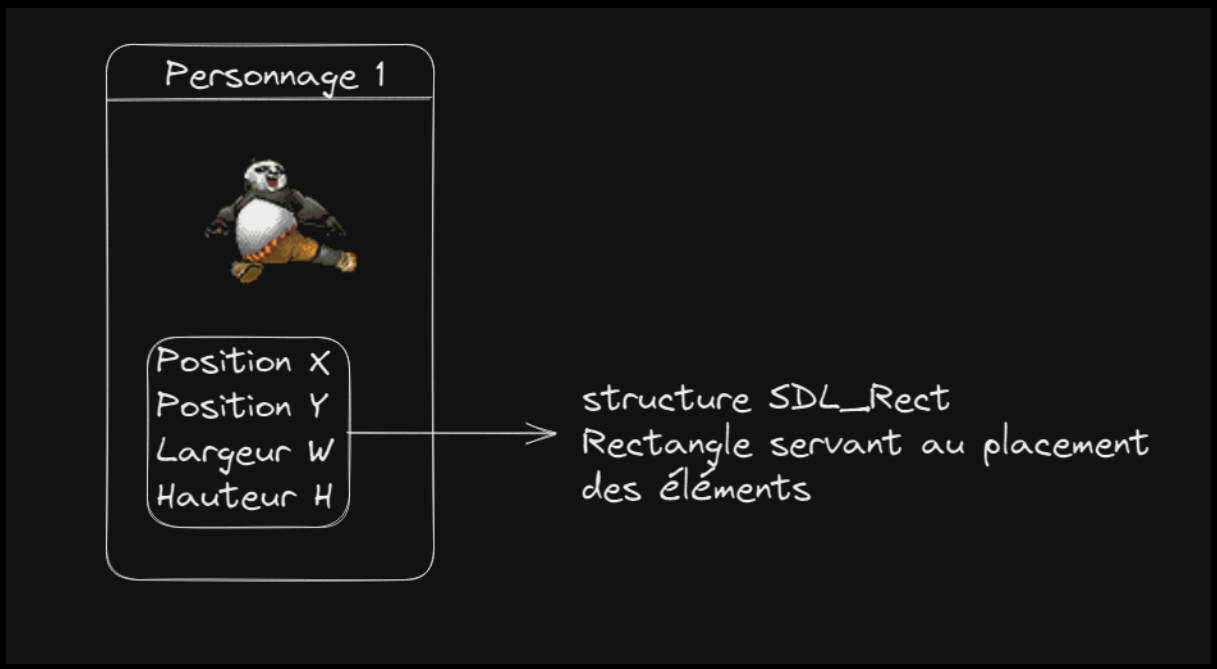
1° La matrice du jeu

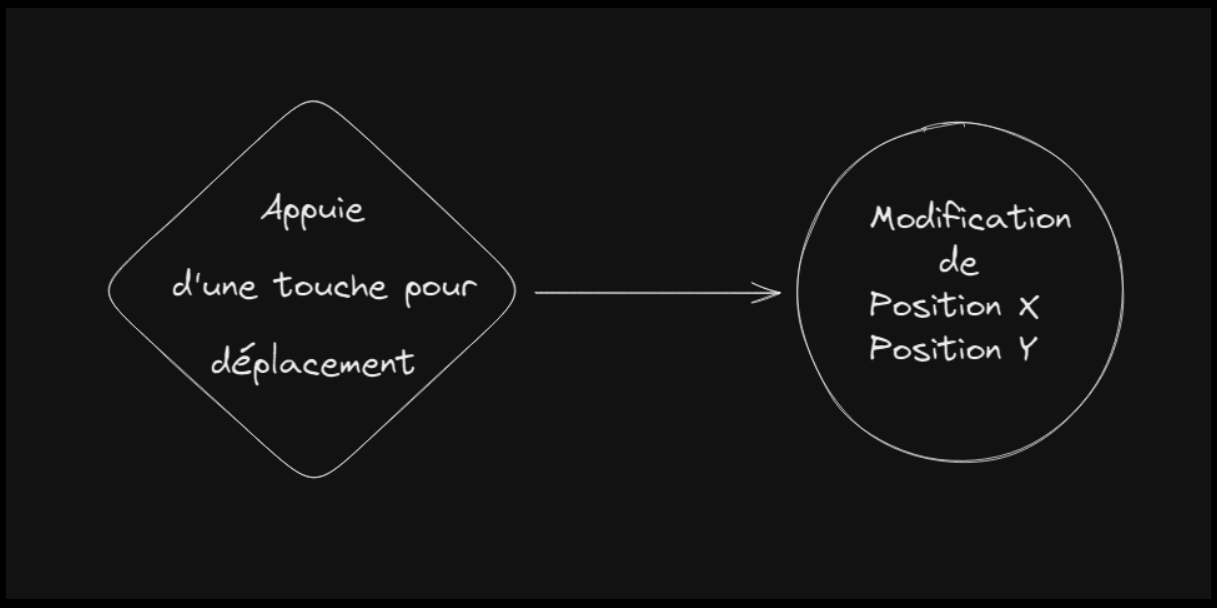
Le jeu fonctionne sur la base d'une matrice 13*13, contenant seulement des int entre 0 et 6. Chaque int représente un élément du jeu; respectivement 0 pour les murs, 1 pour le chemin, 2 pour les rochers , 3 pour les bombes. Pour le reste, nous les évoquerons dans les conditions de victoire.



2° Déplacement des personnages

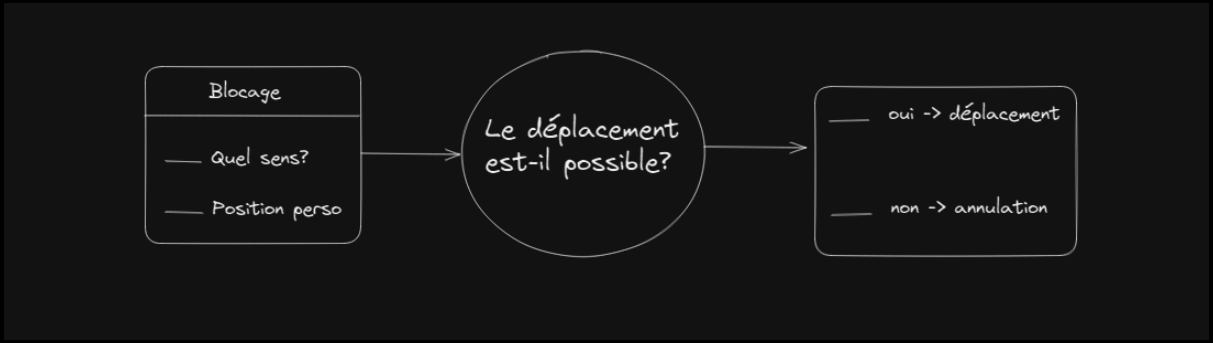
Il faut savoir que les personnages ne sont pas présents dans la matrice, on peut donc se demander comment les personnages peuvent se déplacer. Chaque personnage possède un vecteur de déplacement, une structure contenant une coordonnée en X et en Y. Ainsi, en divisant ces coordonnées par 13 et en ajoutant à chaque déplacement la taille de l'image, on peut simuler un déplacement comme si le personnage était présent dans la matrice. Grâce à cette solution, les cases du jeu ne disparaissent pas après le déplacement.





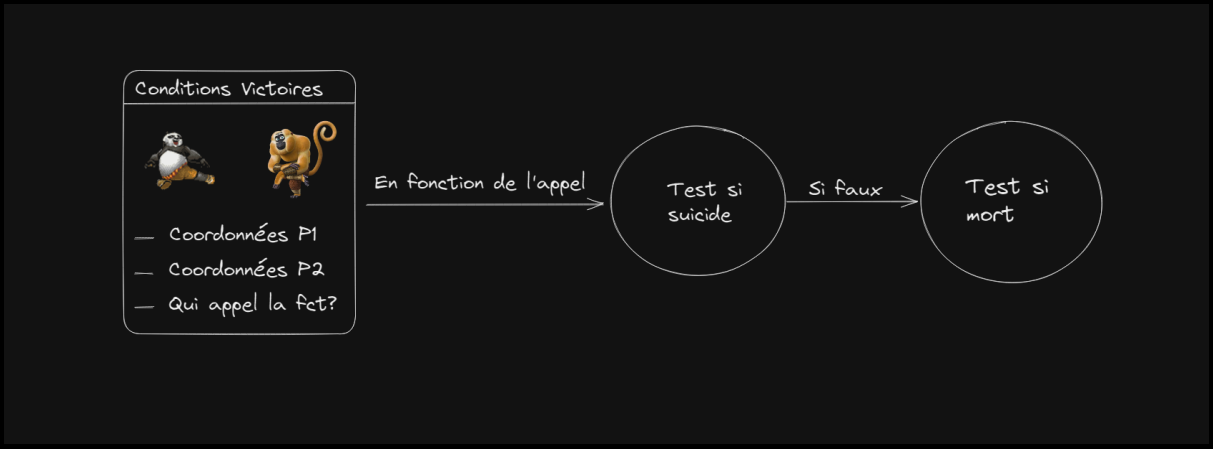
3° Le blocage des personnages

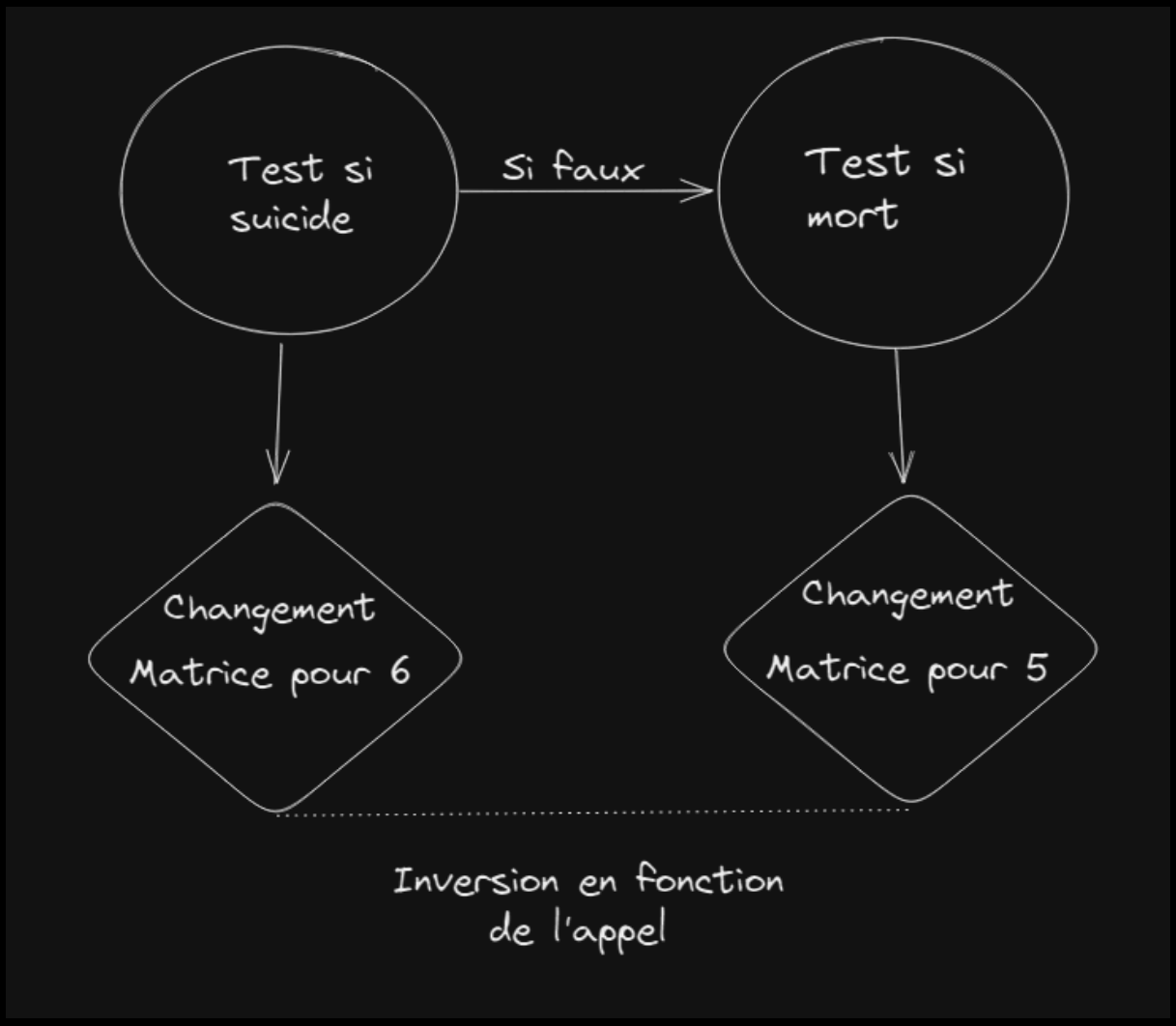
Pour limiter les déplacements des personnages sur la carte de jeu, nous avons créé des fonctions blocages qui testent la case qui est visée par le déplacement. SI celle-ci est différente d'un chemin, alors le déplacement est annulé, sinon on modifie le vecteur position.

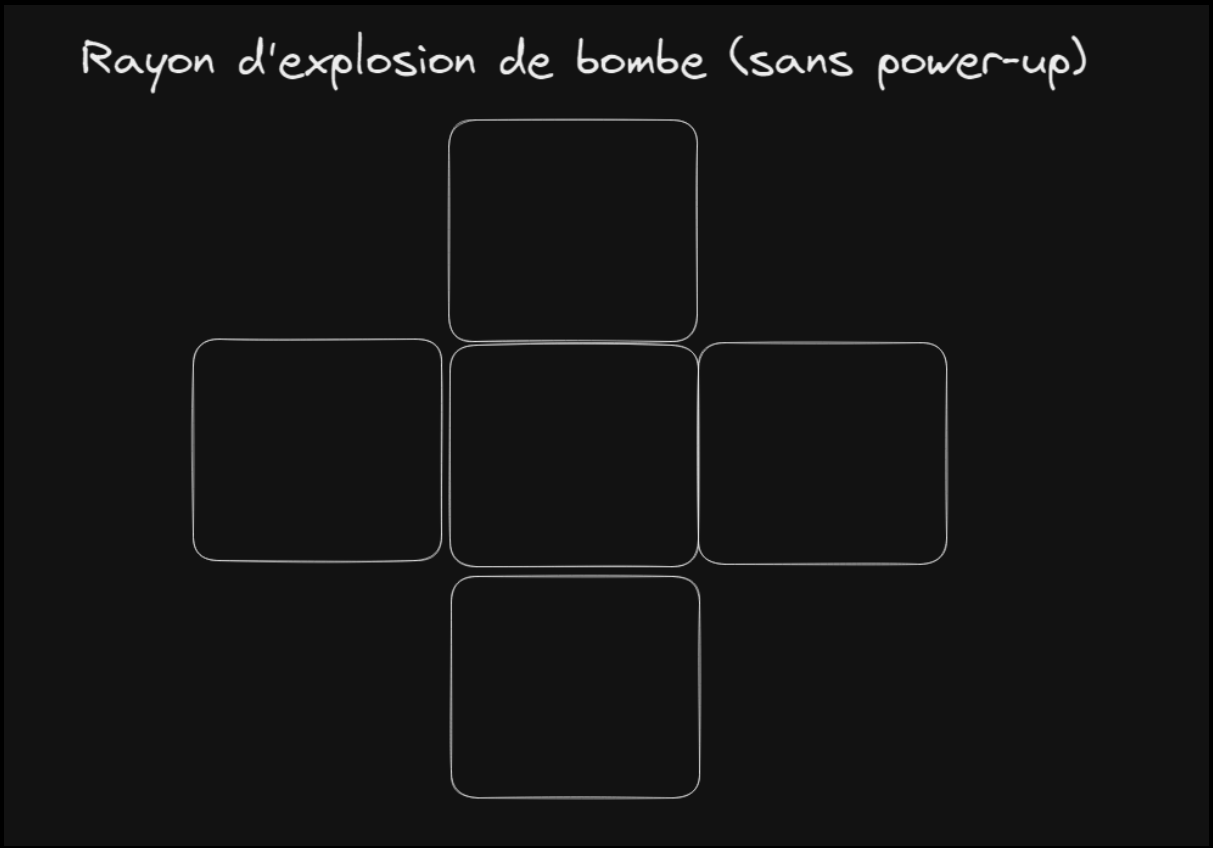


4° Les conditions de victoires

A chaque explosion de bombe, on teste si un des perso est mort. Mais comment réaliser ces tests. Il faut savoir tout d'abord que l'on test toujours le le personnage qui appelle la fonction en sais pas suicider en premier. Pour ce faire on récupère les coordonnées des personnages quand la bombe explose on test les cases autour du personnages de cette façon:



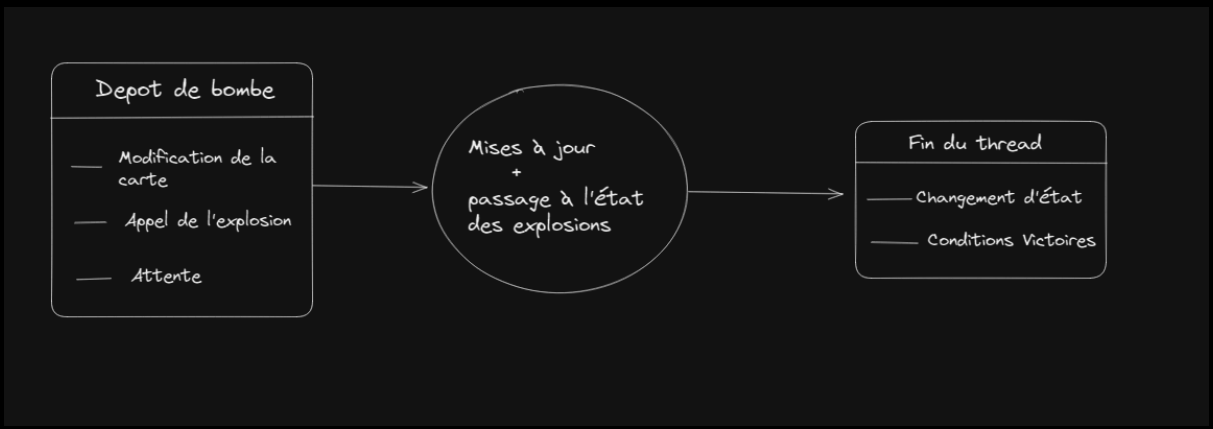

Test si suicide
Si faux
Test si mort
Changement Matrice pour 6
Changement Matrice pour 5
Inversion en fonction de l'appel



On réitère ensuite les mêmes conditions mais avec l'autre personnages pour voir si il n'est pas mort

5° Explosions et bombes

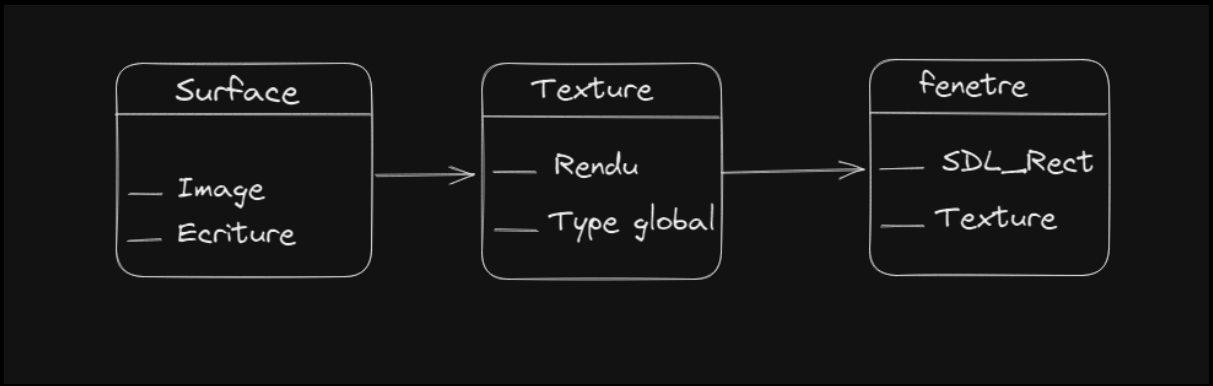
Pour gérer le dépôt de bombe, nous avons utilisé un autre thread (explication plus bas). Tout d'abord, lors de l'appuie de la touche, on modifie la carte pour passer l'état de la case à 3 (la modification graphique se fera automatiquement). On impose ensuite au thread, d'attendre pendant 2 secondes le temps que le joueur se mette à l'abri. On passe ensuite l'état des cases soumises à l'explosion à 5 pour afficher l'explosion puis on re-attend 750 millisecondes. Ensuite on remet les cases à l'état chemin puis on teste si quelqu'un a gagné.



4°) Graphismes avec la SDL2

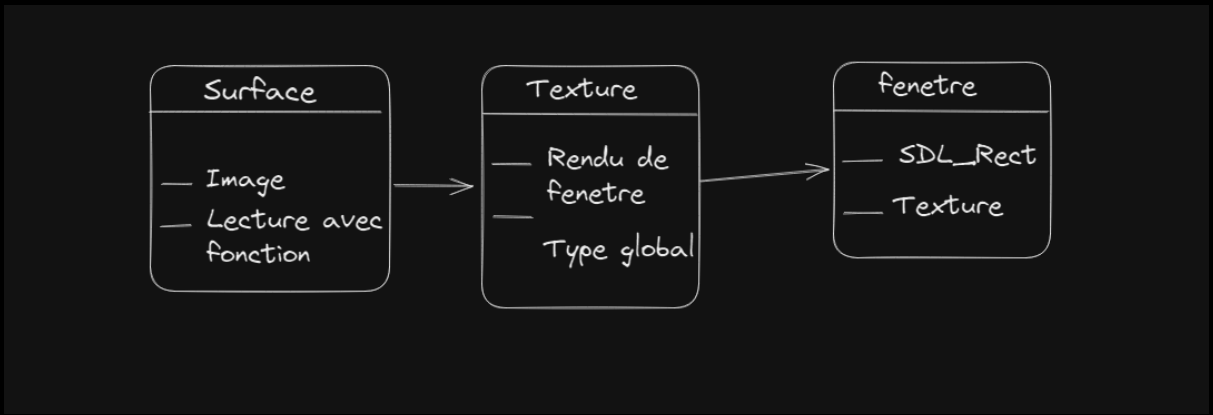
1°Introduction

Avant de commencer les explications, nous allons expliquer très brièvement comment la SDL fonctionne en C. Cette bibliothèque permet de charger des éléments sur une fenêtre. Pour ce faire, elle utilise des textures et des surfaces; des structures propres à la SDL. Pour charger un élément, on commence par créer une surface contenant l'élément dit. On crée ensuite une texture grâce à cette surface mais également grâce à un renderer (le renderer est ce qui permet d'afficher les éléments sur la fenêtre). On place ensuite la texture grâce à un "Rect", un rectangle qui permet de placer la texture en X/Y, mais également définit la largeur et la hauteur.



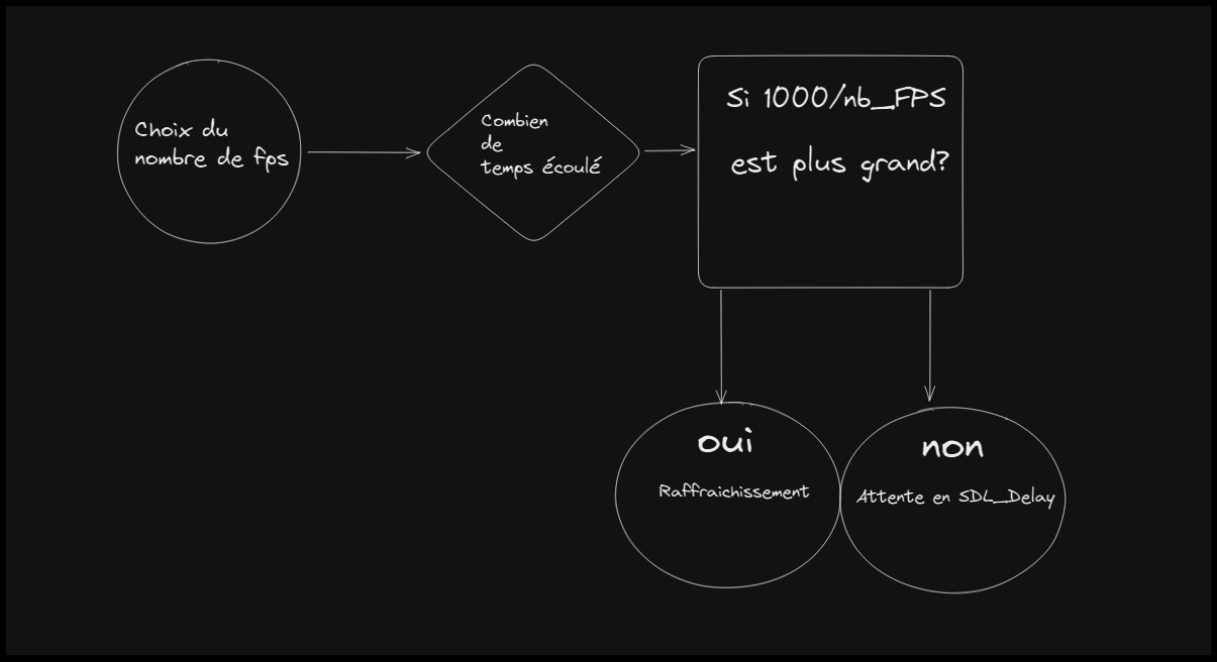
2° Chargement des images

Pour charger des images, on suit le même fonctionnement . On met notre image dans une surface, puis on crée une texture. On la place dans le rendu grâce au rectangle et on force l'update sur la fenêtre. Il suffit ensuite de réitérer l'action pour chaque image de notre jeu.



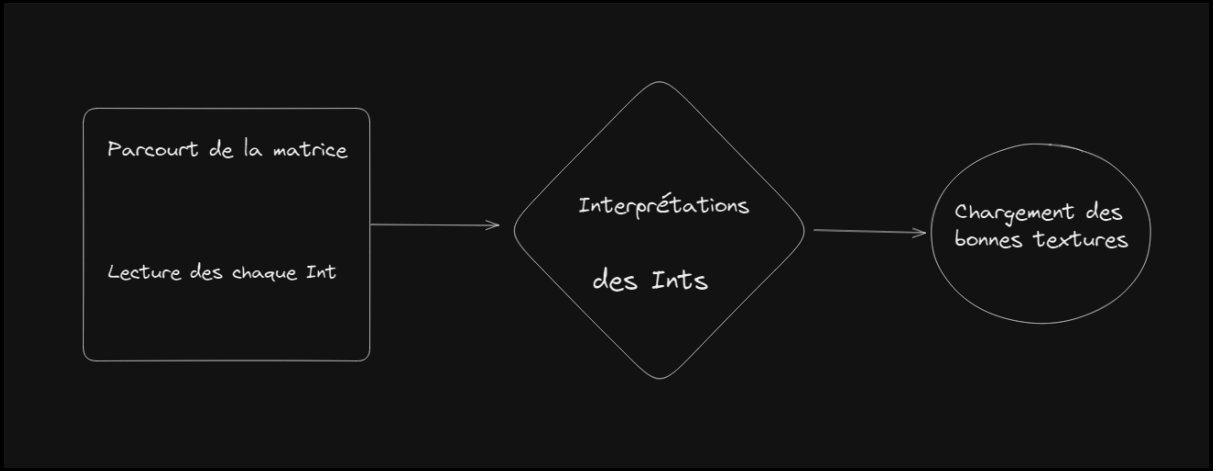
3° Rafraîchissement de la fenêtre

Pour améliorer la fluidité du jeu, il faudrait que le rafraichissement de la fenêtre se fasse automatiquement. Pour se faire , il faut définir une fonction qui, pour un nombre voulu par l'utilisateur, demandera à la fenêtre de s'updater. On définit donc le nombre de 1000 images par seconde que l'on divise par le nombre de rafraichissement que l'on veut. Ensuite, on demande à chaque instant le nombre de ticks qui s'est passé entre le lancement du programme et maintenant. Grâce à cela, il suffit de retirer ces ticks à 1000 puis de tester si c'est inférieur au nombre de fps que l'on veut. Si oui, on force le rafraichissement ce qui nous donnera donc un update automatisé de la fenêtre



4° Initialisation de la carte

Pour initialiser la carte, on commence par mettre le int de notre choix dans la matrice expliquée précédemment. On parcourt ensuite cette matrice en entier et en fonction du nombre testé on copie dans le rendu la texture adéquate. Il suffit ensuite d'augmenter les coordonnées du rectangle de la taille de l'image pour construire notre carte de jeu dans la fenêtre.

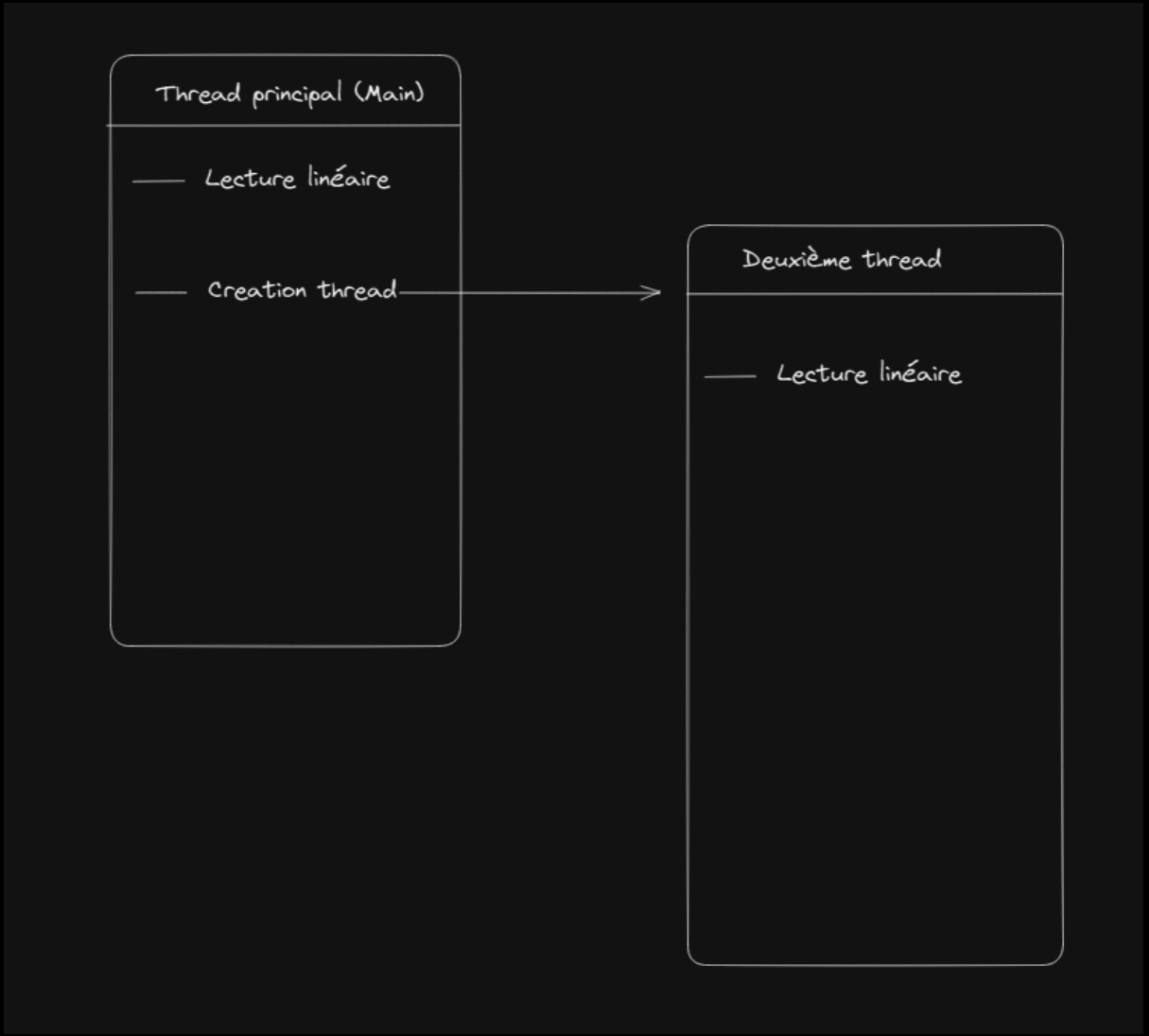


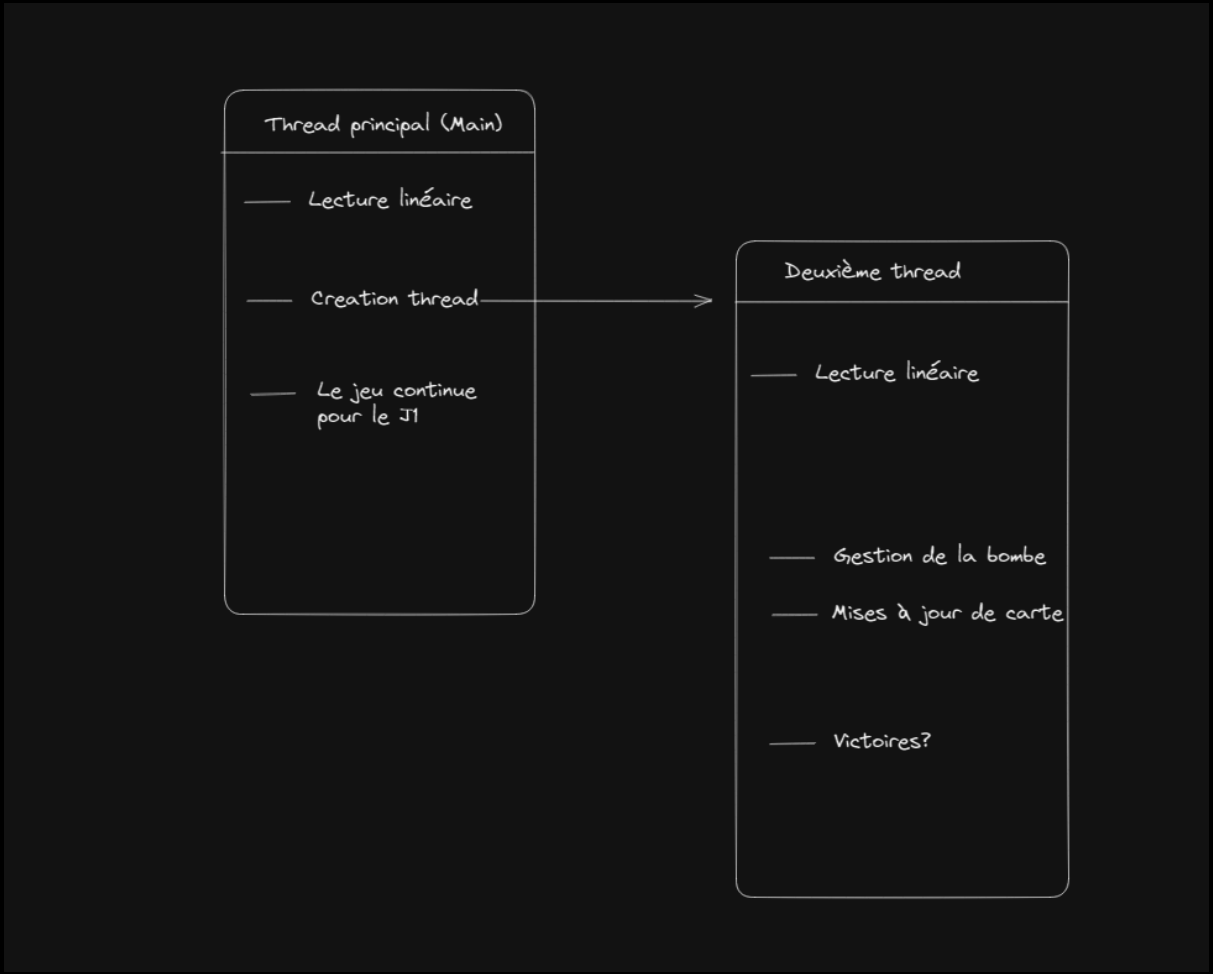
5° Prise en compte des touches

SDL permet de prendre en compte les touches du clavier dans des variables DEFINE dans ses fichiers headers, il suffit donc de créer un switch en testant toutes les touches possibles d'augmenter les coordonnées du VecteurPosition du personnage en fonction de la touche pressé.

5°) Threads

Pour garantir le bon fonctionnement du jeu, nous avons dû nous initier au thread en C. En effet, lors du dépôt de la bombe sur le jeu, il faut simuler une attente d'environ 2 secondes pour laisser le temps au joueur de se mettre à l'abri. Cependant , sans l'utilisation de thread, cette attente nous bloquerait complètement dans le jeu, et rendrait donc impossible le déplacement de notre personnage. Un thread est un fil conducteur qui exécute un algorithme de façon linéaire. On peut prendre par exemple le main qui est un thread. Pour résumer, lors de l'appel de la bombe, on crée un thread qui exécutera le programme de la bombe donc le dépôt, l'attente, l'explosion et enfin les conditions de victoire.

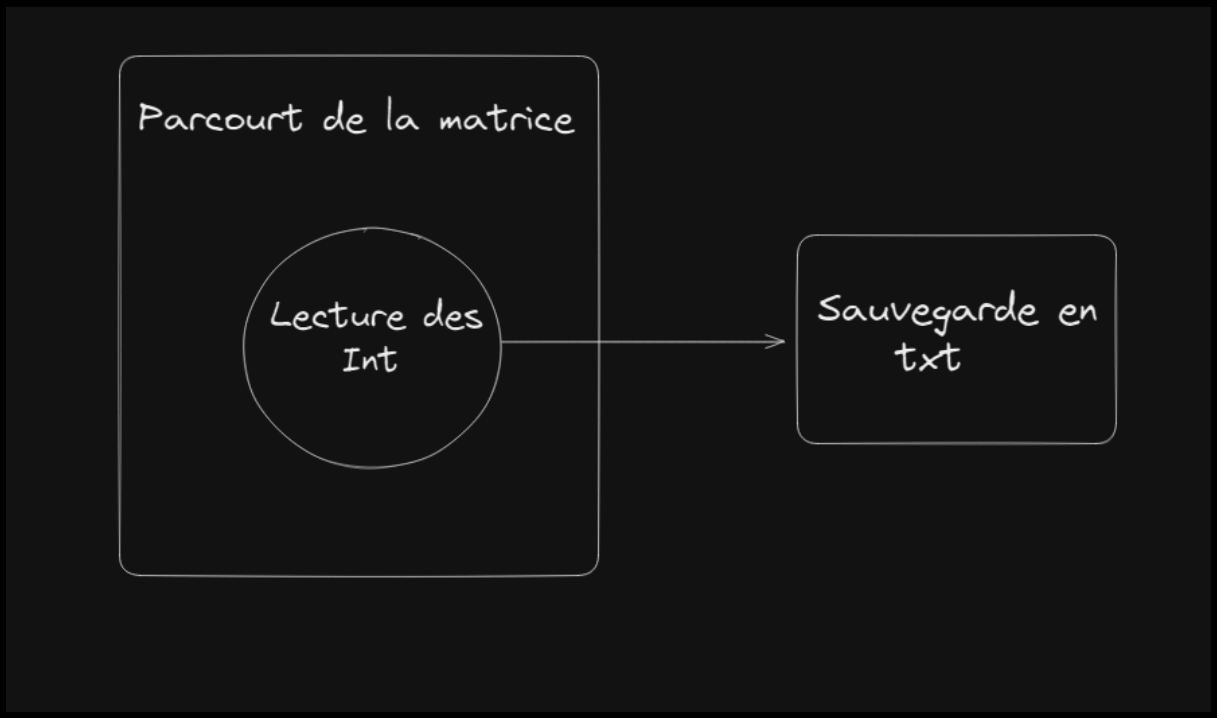

Thread principal (Main)
Lecture linéaire
Creation thread
Deuxième thread
Lecture linéaire



6°) Sauvegarde et Chargement

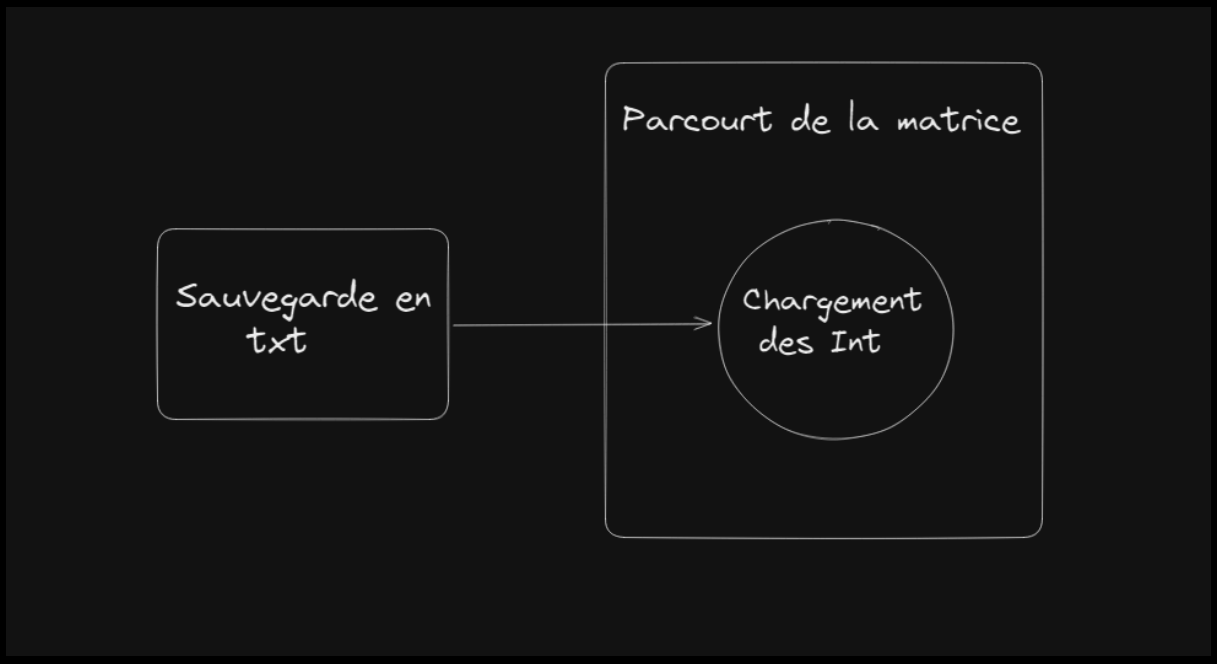
1° Sauvegarde

Nous avons opté pour une sauvegarde en txt, car la seule chose à sauvegarder était les différents états de la carte. C'est-à-dire les int qui sont présents dans la carte. Le reste sera automatiquement régénéré grâce au rafraîchissement automatique.



2° Chargement

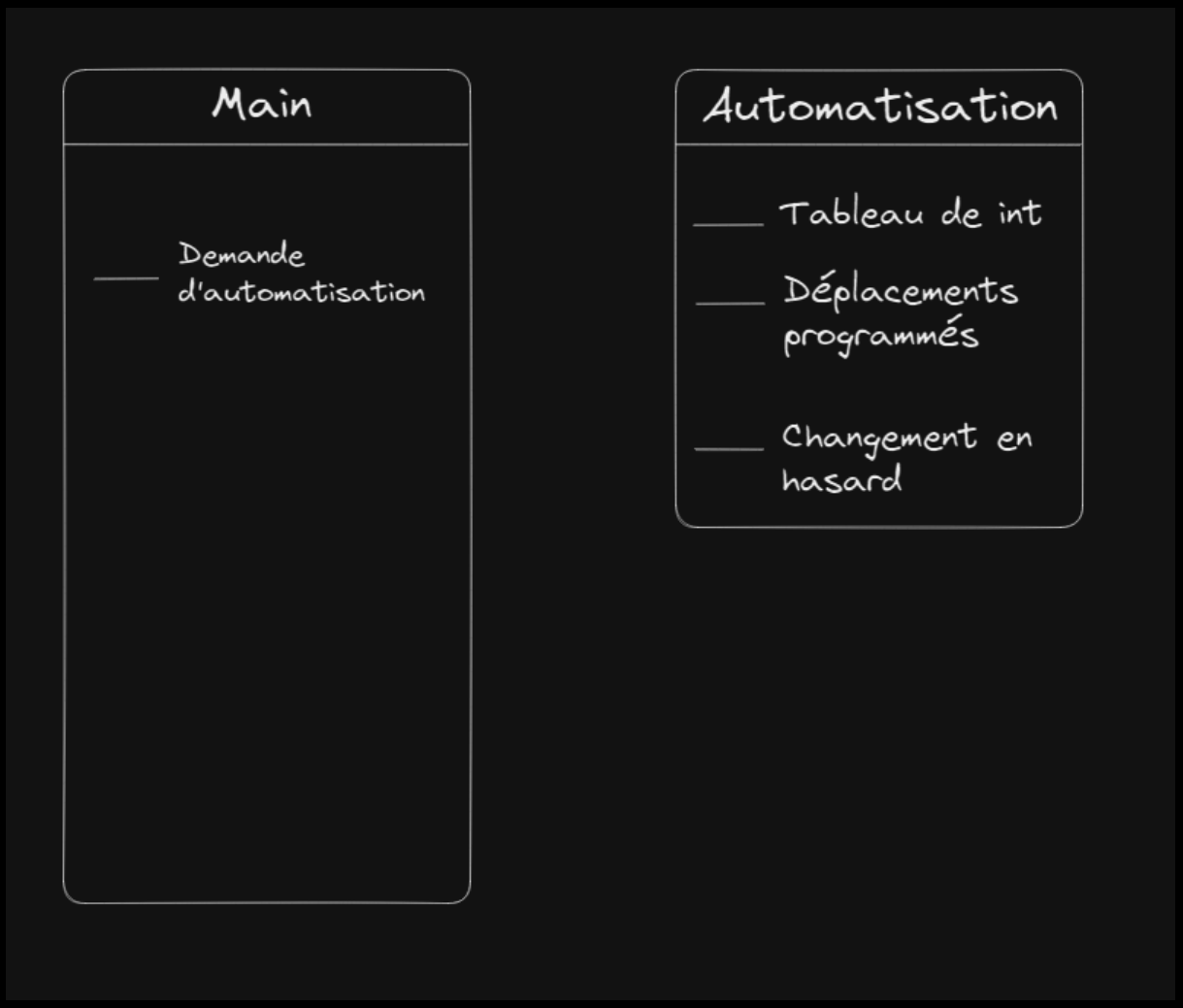
Pour le chargement, il suffit de lire les données et de les remettre dans la carte de jeu actuelle. Encore une fois le rafraichissement automatique se chargera de modifier l'affichage en fonction de la save



7°) Création d'une IA

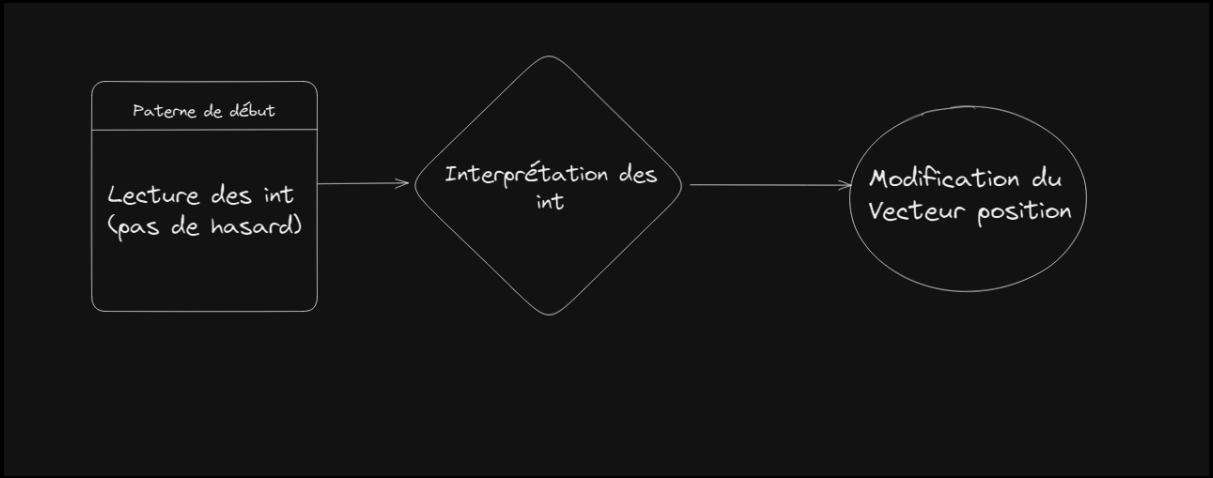
1°Initialisation d'un thread

Pour simuler le déplacement d'un automate, il est nécessaire que celui-ci s'exécute dans un autre fil que celui du main (pour ne pas bloquer complètement le cours du jeu). La première étape est donc de créer un thread à chaque fois que le joueur automatise le déplacement (appuie de la touche A). On passe une fonction qui s'occupera du pattern pour que le joueur ne suicide pas au début mais également le fait que les déplacements soit hasardeux après la fin du pattern.



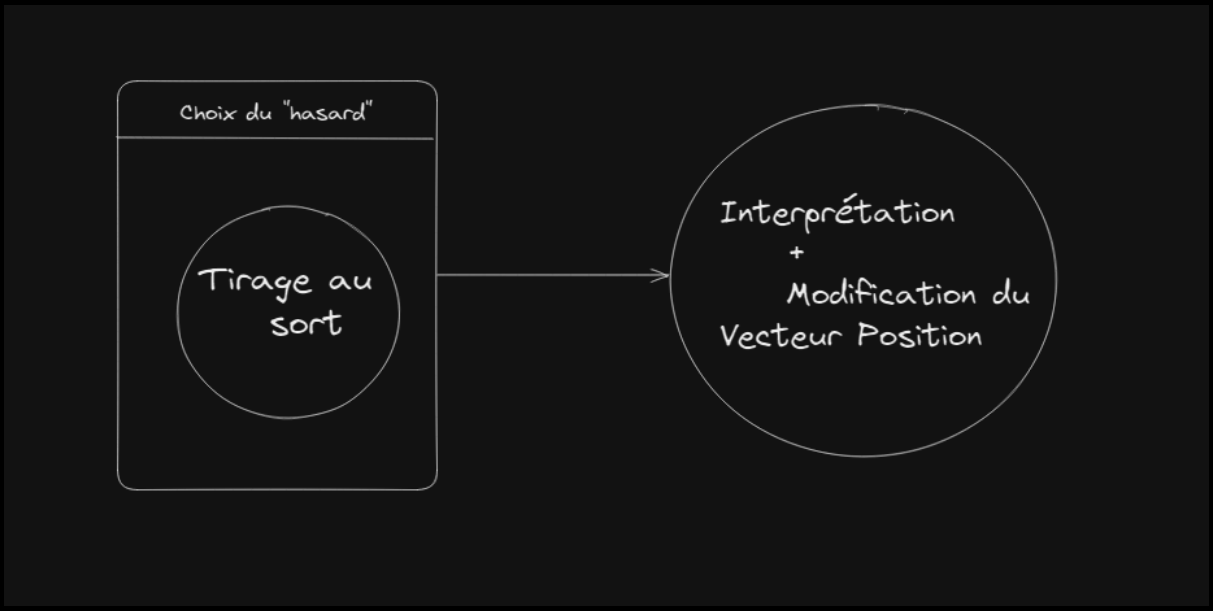
2° Paterne de début

Pour eviter que le 2ème joueur automatisé ne se suicide des le début, nous avons implementé une petite fonctionnailité qui lui permet de suivre un parcours défnit au début de la partie avant de lui laisser la main. On a créé un tableau 2 dimensions remplit de int qui représente les déplacements/actions. Une fois le tableau fini, le hasard rentre en compte.



3° Hasard des déplacements

Les déplacements de l'intelligence artificielle sont complètement aléatoires, en tout cas dans l'idée. En effet, la fonction rand() n'est pas thread safe car elle utilise un état caché qui est modifié à chaque appel". Le thread choisit donc une seed par défaut et répète tout le temps les mêmes actions. En théorie, si le rand() marchait réellement, l'IA serait fonctionnelle. Pour en revenir au déplacement, en fonction du nombre choisi, on interprète celui-ci par une action concrète comme le déplacement ou encore le dépôt d'une bombe.



8°) Conclusion

Pour conclure ce projet, nous allons tous faire une petite conclusion personnelle sur ce que nous a apporté ce projet:

Allemand Valentin : Ce projet a été une réelle source d'inspiration pour moi. En effet, nous avons dû effectuer un réel produit fini , qu'un acheteur aurait pu acheter. De plus, la découverte de la bibliothèque graphique SDL2 m'a réellement plus. J'ai dû m'investir pour développer la partie graphique de ce jeu, ce que je n'avais jamais fait auparavant. J'ai également bien aimé le fait qu'on ait créé une réelle "team" comme on peut voir en entreprise.

Fayet Fabio: Ma tâche était de réaliser les tests de fonction du programme, mais aussi d'aider les programmeurs. Pour chacune des fonctions du programme des conditions pour vérifier et éviter tout problème lors de la réalisation et l'exécution du code. Les plus grandes difficultés auxquelles nous nous sommes confrontés sont les conditions de victoires avec le refresh map, le principe de multithread mais aussi le créer une IA (qui ne se suicide pas). Ce projet m'a permis tout d'abord de découvrir une nouvelle librairie graphique, la SDL, et de pouvoir l'utiliser. L'application du multithread et l'utilisation des frames sont aussi des utilitaires pratiques suivant le programme que l'on veut construire.

Arrahmani Amin : Ce projet m' a permis de mettre en pratique mes connaissances et m' a également permis d' améliorer mes compétences de travail en groupes. Les points d' avancement réalisés régulièrement était un vrai plus pour lier le travail sur la partie moteur et celle sur la partie graphique et vérifier le bon fonctionnement du tout grâce aux fonctions de test. Pour réussir davantage dans le futur il serait préférable de plus commenter le code de chacun comme nous l'avons fait vers la fin du projet afin de faciliter la compréhension des fonctions complexes pour ces compères de travail.

Denis Thomas: Pour ma part, le projet m'a permis de gagner beaucoup de compétences en matière de multithreading, en particulier, ainsi que d'intégrer plus efficacement les notions liées à l'environnement en langage C. En tant que chef d'équipe, j'ai pu assumer davantage de responsabilités et prendre des décisions qui ont permis de faire avancer le projet avec succès.