esiea
ECOLE D'INGENIEURS
DU MONDE NUMERIQUE

TANK BATTLE

PROJET C - 2018

VICE CITY

Auteurs :

Joseph-Emmanuel Banzio

Lenny Daho

Professeur encadrant :

Michael François

SOMMAIRE

INTRODUCTION

Le projet sur lequel nous avons travaillé pendant près de trois mois a été choisi par le professeur de programmation en C, Mr. Michael François. Il était question de développer un jeu de tanks dont l'objectif est de protéger un oiseau en voie de disparition des attaques de tanks ennemis. Nous avons décidé de nommer notre projet "Tanks Battle - Vice City" et nous avons orienté le design vers l'univers de Grand Theft Auto Vice City.

La complexité de ce projet réside non seulement dans la disparité au niveau des compétences en programmation dans le groupe de projet et du choix de développement (SDL ou en mode console). Le temps est également un facteur principal et difficile à gérer, puisque créer un jeu en mode console nécessite beaucoup de réflexions sur le design afin d'avoir un bon rendu.

Afin de vous présenter notre projet dans son ensemble, nous vous présenterons dans un premier temps l'aspect graphique notamment par rapport aux différents choix que nous avons effectué pour les modèles des tanks et de la map. Ensuite, nous vous expliquerons en détail toutes les fonctionnalités que nous avons développé et comment nous avons procédé pour le faire.

GAMEPLAY

Comme expliqué dans l'introduction, le jeu gravite autour d'une bataille de tanks.Le rôle du joueur est de protéger Piou Piou des tank ennemis coûte que coûte. Le tank est de niveau 1.

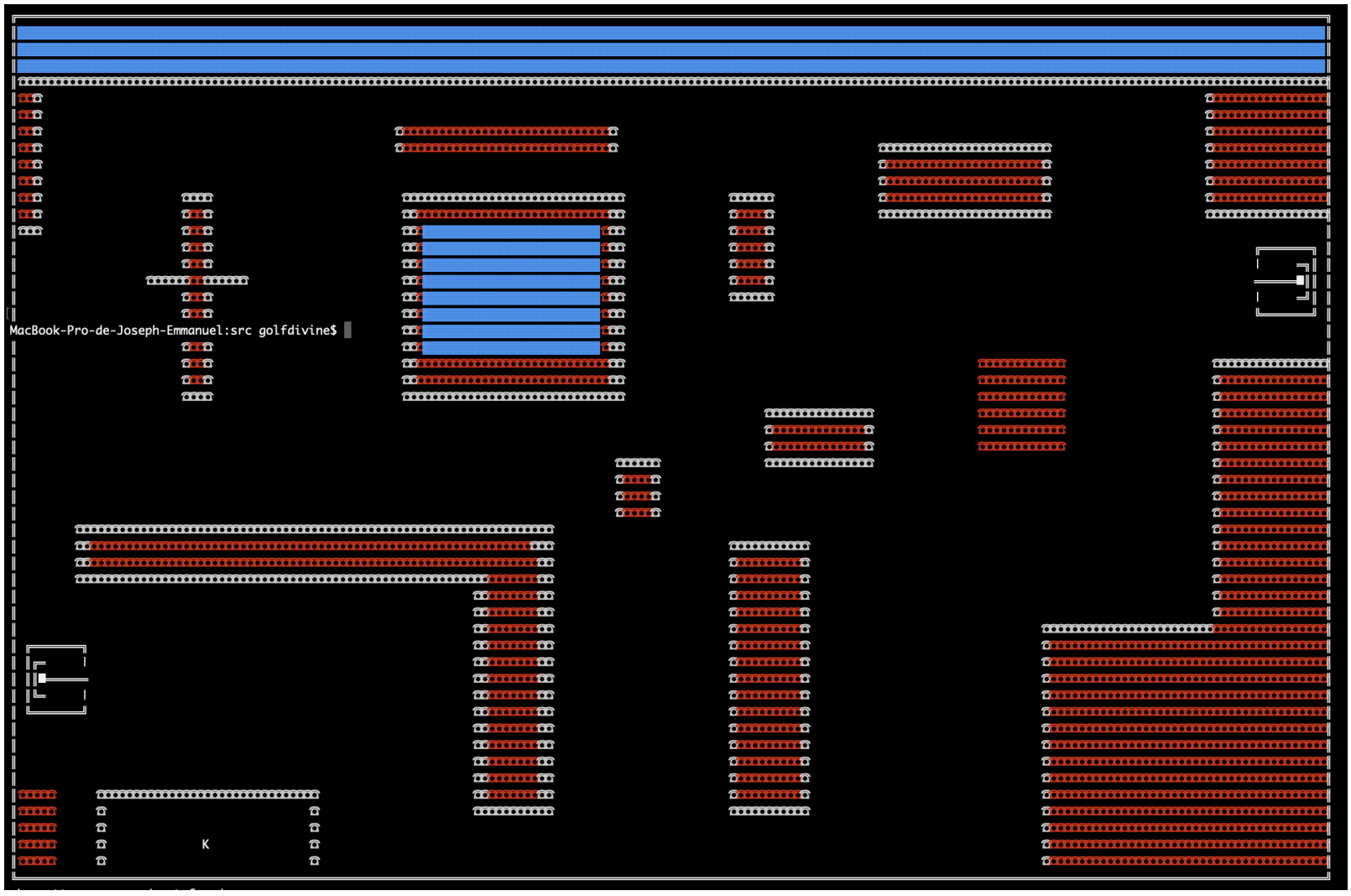
Puisque le jeu a été développé sur Mac, nous avons choisi comme différentes touches les touches, les flèches directionnelles du clavier et la touche G pour tirer.

Les tanks ennemis se déplacent et tirent de façon pseudo-aléatoire. Pseudo-aléatoire dans le sens où un nombre est généré entre 1 et 5 avec la fonction rand() du C. Ce n'est donc pas une probabilité exacte puisque le nombre 2 par exemple peut être générée plusieurs fois d'affilée. Au début, Joseph-Emmanuel avait pensé au développement et la mise en place d'une certaine intelligence chez les ennemis. Faute de temps, nous n'avons pas pu le faire. Dans l'idée, il souhaitait utiliser l'algorithme de Djikstra qui permet, par le biais de la théorie des graphes, de calculer le plus court chemin d'un point A à un point B. L'objectif (pour les ennemis) est de tuer Piou Piou, l'idée aurait donc été d'utiliser les coordonnées de ce dernier pour simuler Djisktra. Tous les différents chemins auraient été stockés dans un tableau à deux dimensions puis on aurait fait un rand pour éviter que le déplacement des tanks soit répétitif. Une autre idée aurait été d'empêcher les tanks de tirer tant que sur la ligne ou sur la colonne il ne voit pas le tank du joueur.

Le joueur et les ennemis ont la possibilité de casser des briques qui sont des obstacles qui bloquent le passage. Piou Piou est protégé par une muraille de briques rouges.

Ce qu'il faut savoir c'est qu'un obus suffit à détruire une brique blanche. Concernant les briques rouges, lors de l'impact avec les briques blanches, elles se transforment en blanches. Il faut donc deux obus pour les détruire.

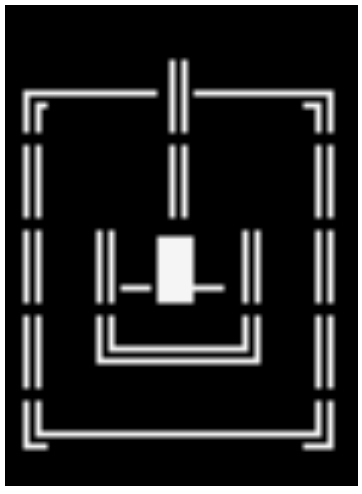
DESIGN DU JEU



Map du jeu (J01)

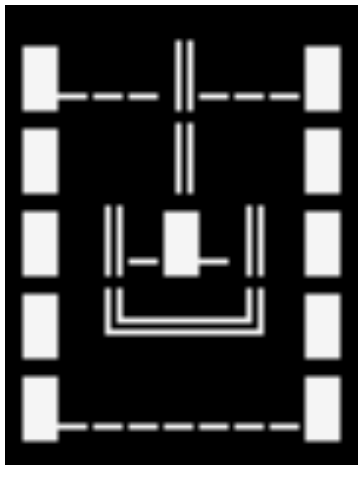
La map du jeu est inspirée de la ville fictive Vice City. Elle représente une ville un peu old school des années 80. Etant limité sur le design en mode console, nous avons considéré les briques rouges entourées de briques blanches comme des immeubles. Les briques blanches sont faciles à casser mais pour casser une brique rouge il faut tirer deux fois. Le premier tir convertit la brique rouge en brique blanche. Pour voir les différents menus du jeu, il faudra y jouer...

DIFFERENTS MODELES DE TANKS



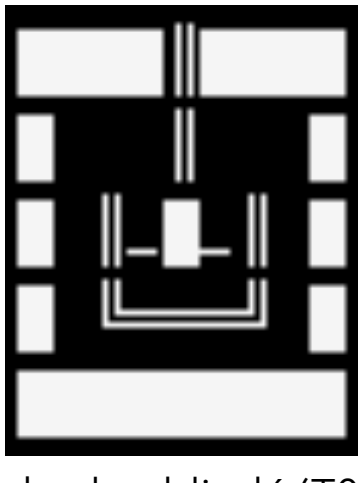
Ce tank est le tank de niveau 1. C'est avec ce tank que le joueur débute la partie. Puisqu'il est de niveau 1, il explose dès qu'il y a collision avec un obus.

Tank simple (T01)



Ce tank est le tank de niveau 2. Nous avons mis en place une probabilité d'apparition de 1/4 pour ce tank ennemi dans le mode facile; concernant le mode difficile, une probabilité de 1/2 a été mise en place pour qu'il y en ait beaucoup plus.

Tank blindé (T02)

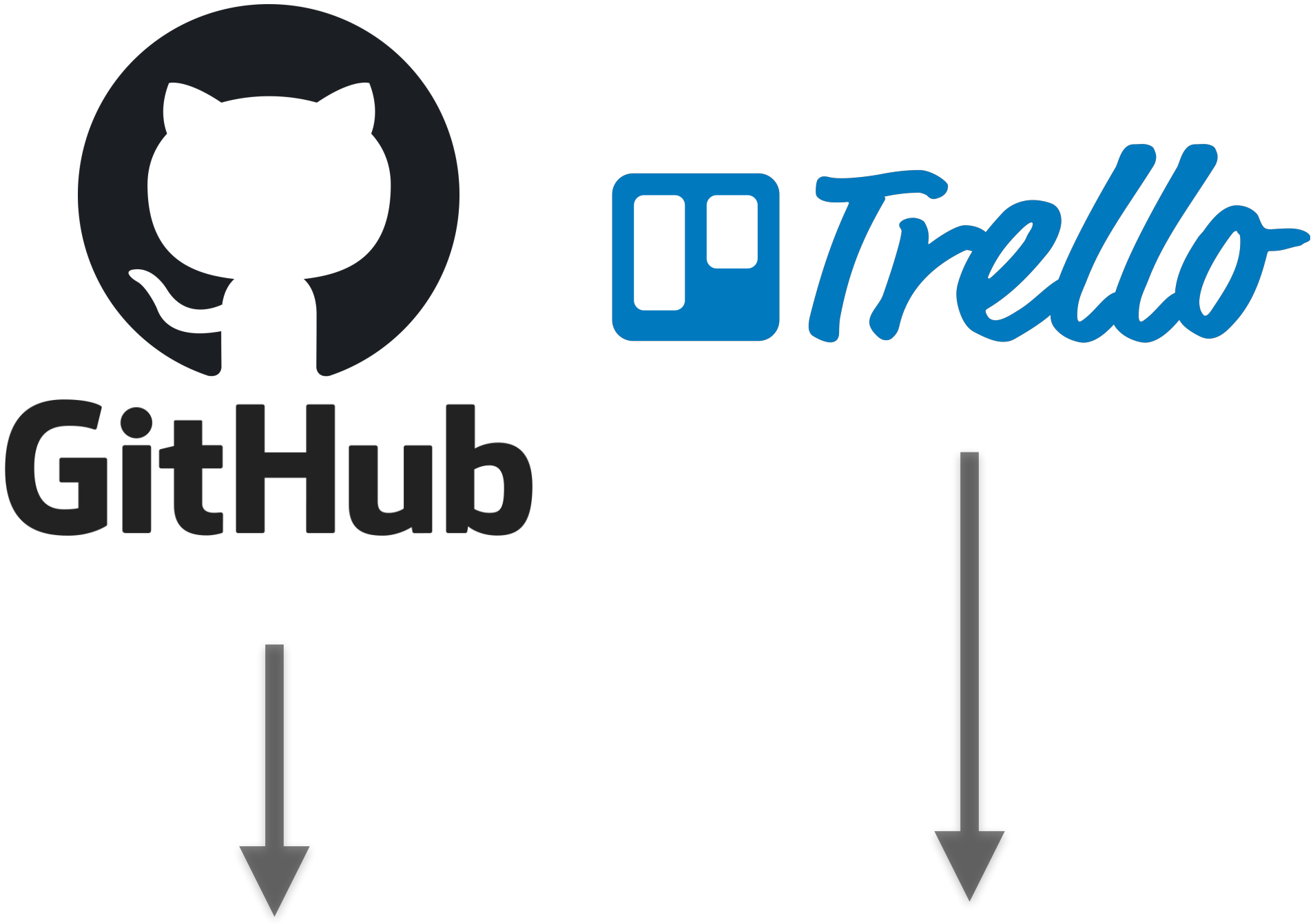


Ce tank est le tank de niveau 3. Il est très rare dans le mode facile car ultra blindé donc 3 niveaux de vie. Nous avons mis une probabilité de 1/5 dans le mode facile et de 1/3 dans le mode difficile pour que la difficulté soit apparente.

Tank ultra blindé (T02)

DEVELOPPEMENT DU JEU

Dans cette partie, nous vous expliquerons en détail comment nous avons procédé pour coder ce jeu de A à Z. Du pseudo-orienté objet en C avec les structures et des listes chaînées. Nous vous parlerons aussi du temps que nous avons passé à afficher des adresses et à utiliser lldb (équivalent de gdb sur Mac) pour débugger le code...



Afin de travailler de façon optimale et de pouvoir gérer notre temps, nous avons utilisé l'outil en ligne Trello pour organiser nos tâches dès le début. Github nous a permis de travailler sur le projet chacun de son côté sur différentes versions. Cette façon de bosser nous a permis de revoir/apprendre à utiliser Github qui est très utilisée dans le monde du travail. Github nous permet d'avoir une sauvegarde et de retourner en arrière dans le cas où nous avons créé une erreur dans la version actuelle.

Les listes chaînées et les structures occupent une place importante dans le bon fonctionnement de ce projet. Sans elles, le jeu aurait été beaucoup plus difficiles. Les structures permettent de faire du pseudo-orienté objet en C. Nous avons créé quatre structures, une structure de tank contenant toutes les informations sur les tanks (type, armure, la carrosserie...), une pour les obus et les deux autres correspondent aux listes chainées des obus (ObusList) et des tanks (TankList). Chaque tank a une bodyWork dans sa structure qui correspond à la matrice qui contient le tank.

```c
typedef struct Obus Obus;

struct Obus {

    int posX;
    int posY;
    int id;

    char direction;
    int type; //E
    int timer;
    int provenance;

    Obus *next;
};
```

```c
struct Tank
{

    char direction;   //B vers le bas, H vers le haut, G vers la gauche , Droite vers la droite
    int posX;
    int posY;

    int armor; // 0 => pas de blindage, 1 => tank blindé, 2 => tank ultra blindé

    //int blindage_orig;

    char **bodyWork; //Servira pour l'affichage du tank

    char type; //P player, E enemy
    int timer;
    int condition; //3 on décrémente ensuite
    int sameDir;

    int id;

    Tank *next;

};
```

```c
struct ObusList{

    Obus *firstObus;
    int id;
};
```

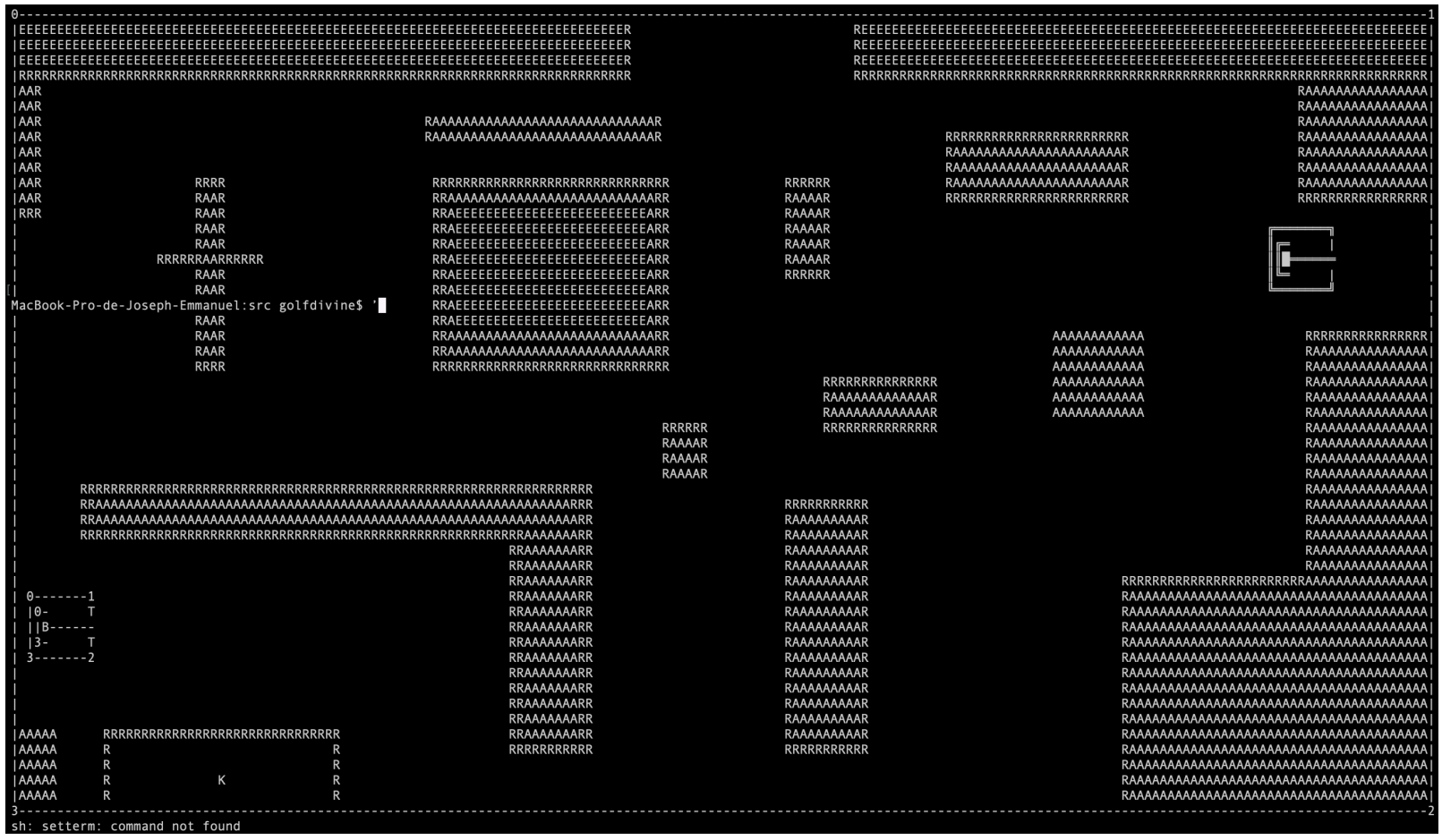
```c
struct TankList{

    Tank *firstTank;
    int id;
};
```

Extraits du fichier headers.h (T04)

Nous avons développé toutes les fonctions permettant de manipuler les listes chainées : de l'ajout d'un tank ou d'un ennemi à la suppression par ID. Nous avons mis en place des ID pour faciliter la suppression des tanks quand ils explosent. Le tank du joueur a l'ID 1, les numéros sont ensuite incrémentés à la suite.

Concernant l'affichage des tanks et de la map, nous avons tout simplement fonctionné de la manière suivante. La map est contenue dans un fichier txt dans le répertoire models/map et les tanks dans models/tanks. On charge ensuite les différents fichiers dans des matrices. Avant de lancer le programme, nous faisons appel à notre fonction qui nous permet d'allouer l'espace mémoire pour toutes les matrices de départ avant de les charger en les remplissant avec le contenu des fichiers. Par exemple, puisque nous avons trois différents types de tanks, nous chargeons au départ 12 (4*3) matrices pour les modèles avec les différentes directions. On initialise bien évidemment le tank du joueur en lui attribuant la position (x,y) de départ. Une fois que le chargement et que toutes les étapes nécessaires au bon fonctionnement ont été effectué, nous faisons appel à la fonction loadTankInMatrix() qui permet de charger le tank dans la matrice par rapport à sa position.



Screen de la map (M01)

Après ça, nous utilisons la fonction dispMatrix() qui permet d'afficher une matrice en convertissant les caractères par les émoticones/caractères ASCII étendus. Sans ce switch, nous aurions affiché le fichier de la map comme nous pouvons le voir dans l'image **M01**.

Le déplacement des tanks est beaucoup moins compliqué qu'il ne parait finalement. On déplace graphiquement le tank et dans la matrice aussi. Pour ce faire, on a utilisé trois fonctions.

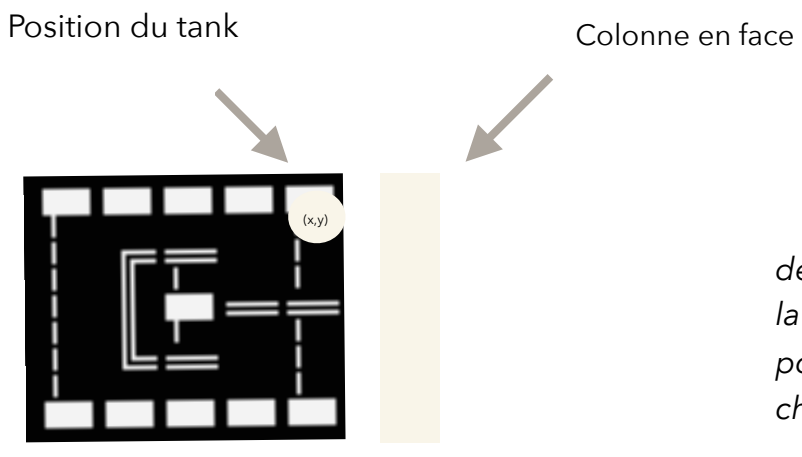
=> MoveTankPlayer() qui permet de déplacer le tank de façon générale par rapport aux touches appuyées par l'utilisateur. Si il appuie direction à droite par exemple, il ira vers la droite. Cette fonction fait appel à toutes les autres dont isFree() que j'expliquerais plus bas.

=> MoveTank() qui permet de déplacer le tank.

=> DeleteTank() qui permet de supprimer le tank.

Nous avons utilisé une des fonctionnalités offertes par le printf du C qui permet de déplacer le curseur à une position X et Y. On déplace ensuite le curseur à chaque fois que le tank du joueur bouge. Avec ça, les déplacements sont beaucoup plus faciles. Dans moveTank, on déplace le curseur à la position du tank +1 + i pour X et +1 +j pour Y pour la prochaine position. Ensuite, on affiche le contenu de la matrice stockée dans la structure Tank. Dans deleteTank(), on affiche juste des espaces et on stocke des espaces à l'ancienne position du tank pour qu'il puisse y retourner.

La partie la plus intéressante est la gestion des collisions qui est gérée par la fonction isFree qui va vérifier si le tank peut avancer et retourne 1 si oui 0 si non.



On fait varier i (initialisée à 0) par rapport aux nombres de lignes du tank. Et à chaque tour de boucle, on vérifie si la position du tank en X +1 (pour descendre avec Y - 1 pour la colonne en face est différent d'un espace à chaque tour de boucle.

Schéma représentatif des collisions vers la droite (SCHM1)

Par exemple, prenons le cas d'un tank dont la direction est "D", pour pouvoir avancer, on doit vérifier que toute la colonne en face de lui est libre sinon il ne pourra pas avancer.

Pour ce faire, nous avons les coordonnées (x,y) du tank qui correspondent à la dernière ligne du tank qui est sur la même colonne que le canon.

Les fonctions de déplacements des tanks ennemis sont plus ou moins similaires à celles du tank du joueur à quelques exceptions près. C'est par rapport à la liste chainée que l'on va déplacer les tanks, on parcourt la liste chainée en faisant attention à ce que le premier tank (celui du joueur) ne soit pas pris en compte sinon son déplacement se retrouvera affecté. On génère d'abord un nombre de tanks défini dans la fonction generateTankEnemy(), 10 pour le mode facile et 15 pour le difficile. Un rand entre 5 et 1 fait largement l'affaire pour gérer aléatoirement le déplacement et les tirs des tanks. Cependant, nous avons fait en sorte que les tanks ennemis ne tirent que quand ils ont le tank du joueur en ligne de mire et qu'ils avancent au moins 15 fois dans la même direction avant de tourner pour leur donner un semblant d'intelligence. Comme expliqué auparavant, nous n'avons pas eu le temps de développer une IA plus poussée. On l'appelle ensuite dans la boucle infinie avec le moveTankEnemy() et tout est géré par la liste chainée. Nous avons mis en place un timer pour le déplacement des tanks ennemis pour que le déplacement ne soit pas trop rapide et pour la génération qu'ils ne spawnent pas tous en même temps.

Les listes chainées ont eu la même importance dans le déplacement des tanks ennemis pour les obus. Nous avons utilisé trois fonctions pour les obus.

=> generateObus() permet de gérer un obus en face du canon d'un tank quelque soit sa direction

=> moveObus() déplace ensuite l'obus par rapport à la liste chainée

=> collision permet de gérer les collisions notamment tout ce qui concerne les dégâts entre tanks, sur les briques (brique rouge => devient blanche + 2 obus, blanche => 1 obus), sur les sorties de la map pour éviter les **segmentations faults.** Si un obus touche Piou Piou, l'écran de Game Over s'affichera et on sortira du jeu.

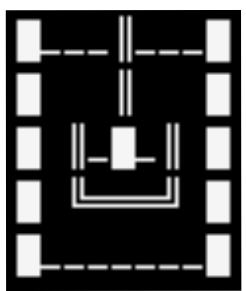
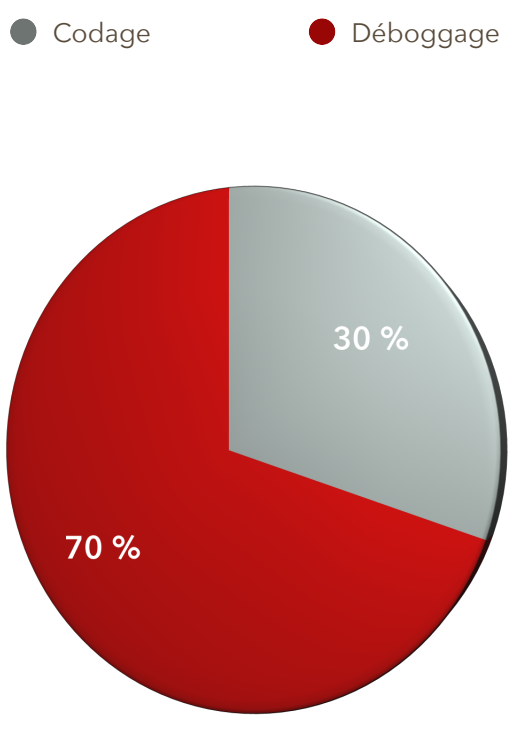


Schéma représentatif de la génération d'un obus (SCHM2)

L'obus est généré en face du canon. En sachant que nos tanks font 9 colonnes et 5 lignes pour se placer à cette position on fait x-1 (aller au dessus) et y+4 (décaler vers la droite de 4 colonnes). On l'insère ensuite

DIFFICULTEES RENCONTREES

Comme vous avez pu le constater malgré le fait que tout soit plus ou moins déjà développé, nous n'avons malheureusement pas pu tout mettre en place à cause des nombreux problèmes que nous avons rencontrées. Le plus gros problème que nous avons rencontré était un bug de téléportation. Dans notre fonction moveObus(), nous envoyions en paramètre un pointeur sur un tank cependant quand on tirait plusieurs fois notre tank se téléportait ensuite à la position de l'obus. Nous avons engendré un certain retard à cause de ce bug. La solution était de faire un passage en valeur ce qui bien entendu n'a pas forcément beaucoup de sens puisque le passage d'un pointeur n'est pas censé généré ce genre de bugs.

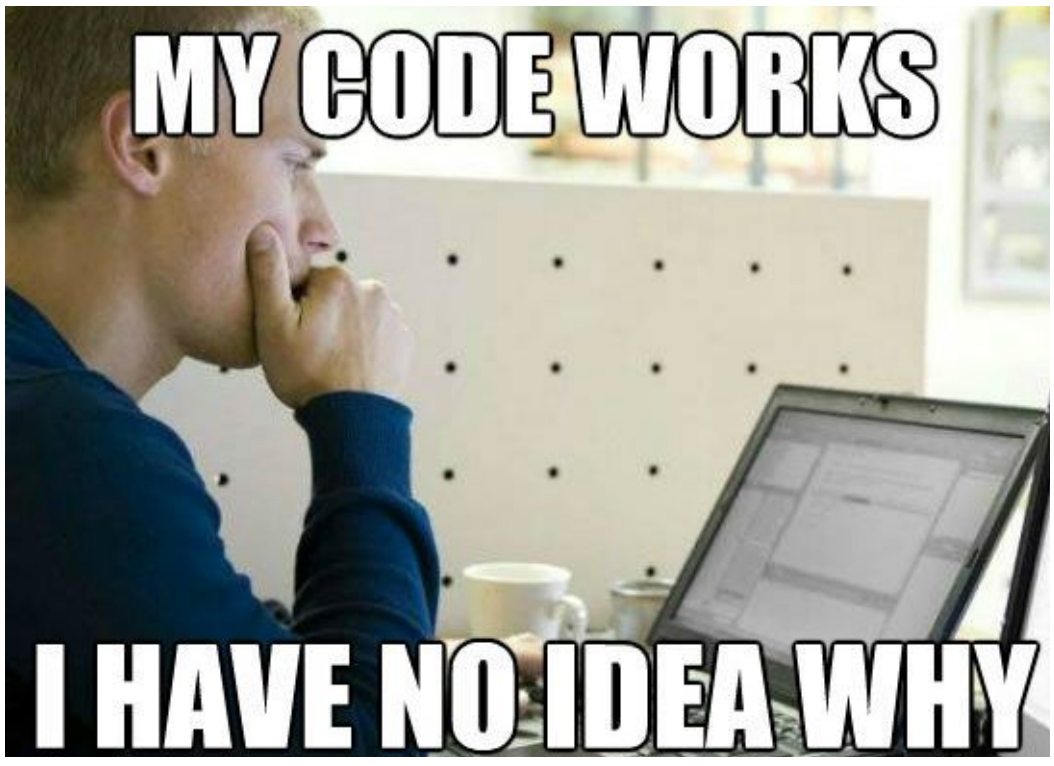


Pie Chart représentant le pourcentage de temps entre le codage et le déboggage (SCHM3)

Le pourcentage de temps est très inégale comme vous pouvez le constater cependant il représente parfaitement notre situation. Le problème est que nous avons tout codé comme en objet et ensuite essayer de tout mettre ensemble. Ce mode de fonctionnement n'est pas adapté au C car procédural. Nous avons donc découvert des bugs récurrents progressivement que nous avons dû régler (des malloc en trop parfois).
L'autre problème assez récurrent énoncé dans l'introduction était au niveau de la disparité au niveau des compétences en programmation.

Le travail n'a donc pas pu être optimal puisque l'un de nous n'avait jamais fait de programmation. Il s'est donc focalisé sur la partie design tandis que l'autre s'occupait du code.

CONCLUSION ET PERSPECTIVES



Malgré le fait que nous soyons assez déçus de ne pas avoir entièrement terminé le projet pour pouvoir nous focaliser sur ce qui nous intéressaient le plus, l'intelligence artificielle(rendre les ennemis intelligent avec Djisktra ou de A*), ce projet nous a permis de revoir le C dans son ensemble : des structures aux listes chaînées. Il nous a également permis d'apprendre à gérer son temps pour respecter des deadlines, s'adapter aux compétences d'une équipe de projet qui peuvent s'avérer être très disparates...