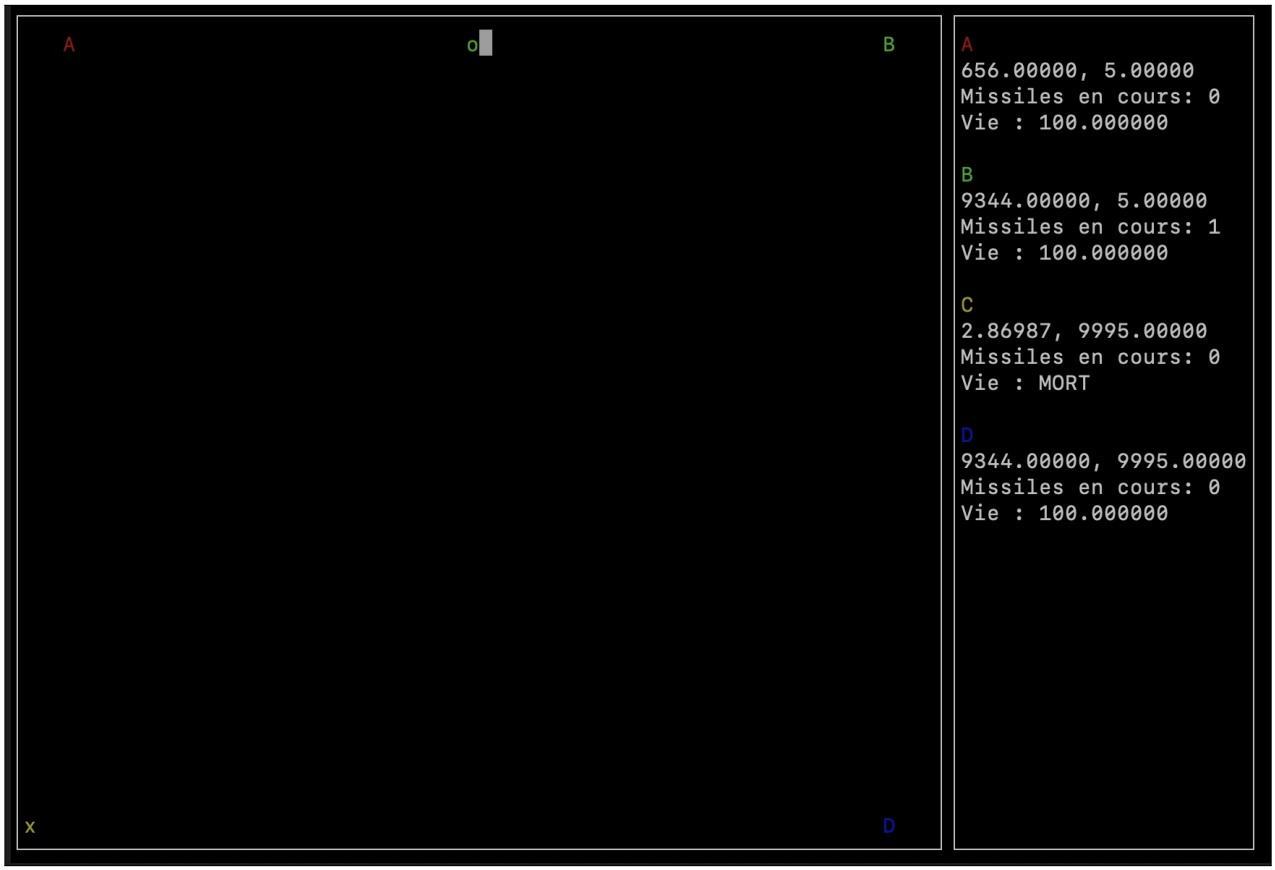

PROJET DE CONDUITE DE PROJET

PRÉPARÉ PAR : ANTOINE BERTHEL, DAMIEN HERBERT, TOM PAYET

15 MAI 2020

URL du dépôt : https://gaufre.informatique.univ-paris-diderot.fr/antomdams/cpproject

Description des fonctionnalités implémentées

Dans le cadre du projet, nous avons créé un programme de «combat de robot».
Pour se faire, nous avons séparé l'implémentation en trois parties :

- le parsage des scripts
- le moteur physique
- l'affichage

Notre programme est donc capable de d'analyser et d'interpréter un code «BURP» afin de donner des instructions à chaque robot.
Les scripts sont lu et stocké par les robots au début du programme, évitant ainsi d'avoir une erreur au milieu du combat à cause d'une instruction mal spécifiée.
Vis-à-vis de la physique la plupart des collisions possible sont gérées par notre code :

- collision robot / robot (avec dégâts de collision)
- collision robot / mur (avec dégâts de collision)
- collision robot / missile (avec dégâts d'explosion)
- collision missile / mur (avec dégâts d'explosion)

Nous avons eu un problème de collision entre les robots et les missiles du fait que les missiles se déplacent de 500 unités par cycle. Pour régler cela nous avons augmenté le nombre de rafraîchissement de la physique (sans changer le nombre de cycles par seconde).
De plus, la vitesse d'exécution peut être modifiée à l'aide des touches 'p' (pour accélérer) et 'm' (pour ralentir).
Ensuite, viens la phase d'affichage qui affiche à la fois l'arène et des informations sur l'état des robots. Chaque robot a sa couleur et sa lettre. Lorsqu'ils meurent, ils deviennent des 'X'. L'affichage est dynamique : on peut modifier la taille de la fenêtre et l'actualiser en fonction d'une constante AFFICHAGE_PAR_SECONDE.

Lancement du projet

(voir README)

Description d'architecture

modele.h : - déclaration des structures permettant de gérer les positions, les robot, les missiles

arene.c : - initialisation des robots
- gestion des collisions et des dégâts qui en résultent

missile.c : - gestion des explosions
- mis à jour des positions

robot.c : - gestion des instructions (script)
- mis à jour des positions
- gestion des points de vie

parser.c : - parsage des scripts fournis

plateau.c : - gestion de l'affichage de l'arène avec NCurses

Nous avons fait le choix de ne pas utiliser la bibliothèque «Glib» car le parsage était suffisamment simple pour le faire nous même.
De plus, une fonctionnalité de debogage a été ajoutée afin de pouvoir afficher la lecture du parser pour analyser les potentielles erreurs.

Organisation du travail

Avant le confinement, il était question de se retrouver chaque semaine en cours pour discuter de ce qui avait été fait dans la semaine et de ce qu'il nous restait à faire.

Ensuite, nous avancions un maximum, que ça soit sur les problèmes rencontrés pendant la semaine ou sur l'implémentation de nouvelles fonctions.

Suite au confinement, le projet était déjà bien avancé, il ne restait plus qu'a faire le parser et éventuellement corriger ou améliorer le projet.

On se retrouvait via discord pour discuter et programmer ensemble. Nous avons utilisons Visual Studio Code pour réaliser ce projet car il possède une grande bibliothèque d'extensions.

Pour pouvoir compiler, nous avons mis en place un makefile.

Couverture

Le code couvre la partie 'obligatoire' du projet.

De plus, nous avons fait en sorte d'augmenter le nombre de rafraîchissement par seconde sans changer le nombre de cycle par seconde. Cela permet d'une part un affiche beaucoup plus fluide et d'autre part de gérer certaines collisions qui étaient a l'origine presque impossible (les missile se déplaçaient trop vite pour pouvoir entrer en collision avec les robots).