Projet Informatique – Sections Electricité et Microtechnique

Printemps 2019 :*Obstructed Dodgeball* © R. Boulic

Rendu1 (jeudi 21 mars 23h59)

1. Buts, forme du lancement du programme, organisation du code et tests

But : test des fichiers, en particulier s'il y a des collisions initiales (section 3.1.1 de la donnée).

Syntaxe du lancement de l'exécutable pour ce rendu1 (section 7.3.1 de la donnée) :

```
./projet Error nom_fichier.txt
```

Le mode **Error** sert à détecter s'il y a une erreur dans le fichier dont le nom suit le mot-clef « Error ». Le programme cherche à initialiser l'état du jeu mais il s'arrête dès la première erreur trouvée dans le fichier.

Il s'arrête aussi après la lecture du fichier s'il n'y a aucune erreur ; dans ce mode il y a affichage d'un message indiquant le succès de la lecture.

Le mode **Error** n'a aucun affichage graphique.

Architecture à adopter (fig 9a de la donnée) :
Le module projet contient la fonction main() en charge d'analyser si la ligne de commande est bien conforme au mode **Error** qui est le seul exploité pour ce rendu (cours du sem1 Topic12).

Si c'est le cas elle délègue l'action de lecture et d'initialisation du jeu au module **simulation** (Fig 9a). Celui-ci, à son tour exploite les modules de plus bas niveaux pour initialiser les structures de données et effectuer certains tests.

Une stratégie simple avec un static vector pour mémoriser les éléments créés au moment de la lecture :

La détection des erreurs de collision entre éléments de nature différentes impose de mémoriser tous les éléments dans un premier temps, puis d'effectuer ces tests (ex : joueur-balle).

Vous êtes responsable du choix de cette mémorisation.

Nous demandons de mettre en œuvre **une structure de donnée dynamique** pour les joueurs, les balles et la grille des obstacles car leur nombre n'est pas fixé à l'avance. L'usage de **vector** ou d'allocation dynamique convient tout à fait. Par contre nous refusons la réservation de gros tableaux de taille connue à la compilation.

Vous pourrez changer d'approche après le rendu1 si vos idées évoluent entre-temps. Nous proposons ici une stratégie de mémorisation simple (appliqué aux joueurs) :

1. Le jeu doit gérer un seul ensemble de joueur
2. Cet ensemble de joueurs peut aussi être géré par le module **player** en charge de la classe **player**.
3. Il suffit de créer un **static vector<Player>** dans le module **player** ; ce module en aura la responsabilité pour ajouter/enlever des éléments.
4. Du fait de cette approche (un seul ensemble de joueur) les fonctions offertes par le module **player** n'ont pas besoin d'avoir un paramètre qui précise sur quel ensemble de joueurs elles travaillent. Comme il n'y en a qu'un, les fonctions travaillent sur l'ensemble défini par ce **static vector.**

Les tests à effectuer sur chaque fichier sont les suivants :
- la position du centre de tous les joueurs et balles doivent être dans le terrain de jeu
- cohérence des indices d'obstacle avec nbCell et absence de duplication des obstacles
- absence de collision joueur-joueur, joueur-balle, balle-balle, joueur-obstacle, balle-obstacle

=> nous fournissons le fichier en-tête **error.h** avec les messages d'erreur que vous DEVEZ utiliser pour chacun des types d'erreurs listés ci-dessus.

2. Forme du rendu1

Pour chaque rendu **UN SEUL membre d'un groupe** (noté **SCIPER1** ci-dessous) doit télécharger un fichier **zip** sur moodle (pas d'email). Le non-respect de cette consigne sera pénalisé de plusieurs points. Le nom de ce fichier **zip** a la forme :

SCIPER1_ SCIPER2.zip

Compléter le fichier fourni **mysciper.txt** en remplaçant 111111 par le numéro SCIPER de la personne qui télécharge le fichier archive et 222222 par le numéro SCIPER du second membre du groupe.

Le fichier archive du rendu1 doit contenir (**aucun répertoire**) :

- Fichier texte édité **mysciper.txt**
- Votre fichier **Makefile** produisant un executable **projet**
- votre code source (.cc et .h)

*On doit pouvoir produire l'exécutable **projet** à partir du Makefile après décompression du contenu du fichier zip.*

Auto-vérification : Après avoir téléchargé le fichier zip de votre rendu sur moodle (upload), récupérez-le (download), décompressez-le et assurez-vous que la commande make produit bien l'exécutable et que celui-ci fonctionne correctement.

Exécution sur la VM: votre projet sera évalué sur la VM du second semestre (disponible depuis le 6 mars).

Backup : Vous êtes responsable de faire votre copie de sauvegarde du projet. Il y a un backup automatique seulement sur votre compte myNAS. Sur la VM, vous devez activer vous-même le backup (icone « engrenage » en haut à droite, choisir system settings, choisir backup et activer cette fonction en précisant les paramètres). Une alternative est de s'envoyer la dernière version du code source par email.

Gestion du code au sein d'un groupe :
- vous pouvez envisager d'utiliser **gdrive.epfl.ch** pour définir un répertoire partagé par les 2 membres du groupe et pas plus. Il n'y a pas d'éditeur de code en mode partagé.
- L'option c4science n'est recommandée que pour ceux qui maitrisent déjà git. Si vous l'utilisez, vous devez supprimer l'accès public (par défaut) à votre code.

2. Date du rendu1

La date butoir est décalée du dimanche 17 mars 23h59 au jeudi 21 mars à 23h59