MERKACHE Mohammed
RAVELONJATO Faly

EISE-4

RAPPORT: PROJET C++

Sommaire

I. Description de l'application

L'application Wall-E The Last Hope est un jeu basé sur un jeu sokobox ainsi que le film Wall-E de Pixar. En accord avec le thème de cette année « There is no Planet B », le jeu consistera à aider à rétablir l'environnement naturel de la terre avant sa destruction en plantant des arbres avec Wall-E et de faire comprendre que même les actions minimes peuvent avoir des résultats inimaginables sur le long terme.

Les règles du jeu :

- Le joueur a des points de vie et peut prendre des dégâts s'il est exposé à de la radioactivité.
- Si les graines sont exposées à la radioactivité, elles seront détruites donc fin de partie
- Si les points de vie du joueur sont inférieurs à 0, la partie est terminée
- Le score diminue à chaque déplacement et la partie se termine quand le score devient 0.

II. Diagramme UML

1. Les classes principales :

La classe Game :

La classe game est celle qui permet de faire tourner le jeu, elle contient les objets des autres classes. Elle contient des variables que l'on peut catégoriser :

- relatives à la fenêtre graphique donc dérive de la bibliothèque sfml : window, videoMode, Icon , Sprite, Texture Font.
- relatives au jeu : des objets des classes character et level.

La classe Character:

- Les objets de la classe **character** sont ceux représentés par des sprites dans le jeu.
- La classe contient aussi les attributs pour les positionnements et les animations des sprites / des blocs dans le jeu.

La classe Level:

La classe level est une classe qui permet de créer les maps de chaque niveau à partir d'un fichier texte.

- Les attributs principaux sont la taille de la grille et le nombre de graines à planter pour chaque niveau.

La classe Player:

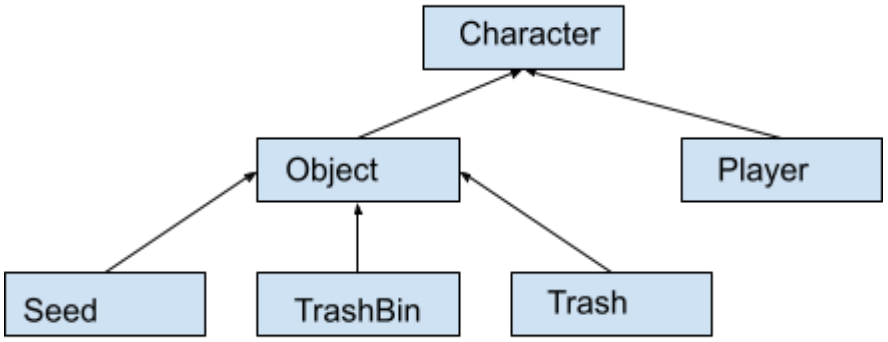
La classe Player découle de la classe character avec des attributs et méthodes spécifiques au joueur, notamment par rapport à ses points de vie.

La classe Seed:

La classe Seed découle de la classe character avec des attributs et méthodes spécifiques à la graine en rapport avec les animations de la graine et sur ses possibles états: plantés ou infectés.

2. Les héritages :

On a une hiérarchie en 3 étages :



Les classes object et player héritent de la classe Character et les classes seed , trash et trashbin hérite de la classe Object.

III. Procédure d'installation des bibliothèques et lancement du jeu

Notre jeu a été développé en C++ et on a utilisé la bibliothèque SFML pour la partie graphique.

Pour installer la bibliothèque sur Ubuntu, on l'installe avec le gestionnaire de paquets, par exemple, en tapant la commande `sudo apt-get install libsfml-dev`. Il faut s'assurer que la bibliothèque soit installée dans le chemin standard sinon il faut changer le makefile existant dans le dossier zip de l'application pour le faire fonctionner.

Pour compiler le programme, il faut dézipper tout le dossier et se placer dedans, puis saisir dans votre terminal la commande « make » puis l'exécutable « ./Wall-e ».

En parlant du jeu en lui-même, il y a d'abord le StartMenu, puis le jeu et l'image de fin.

Le StartMenu :

Pour lancer le jeu, il faut cliquer sur le bouton start ou appuyer sur la touche entrée.

Le jeu :

a) Aspect du jeu :

b) Les commandes :

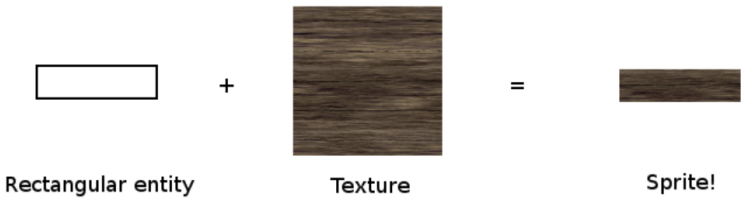
-Les directions : on utilise les touches UP/DOWN/LEFT/RIIGHT ou Z/S/Q/D.
-La touche « échap » permet de fermer le jeu
-La touche « R » permet de restart le jeu
-La touche « U » permet d'annuler le mouvement précédemment effectuer

IV. LES IMPLÉMENTATIONS PERTINENTES

1. Les animations

- Wall-E :

On a utilisé plusieurs images pour rendre de plus en plus vivides l'animation du personnage. De ce fait, à chaque mouvement effectué, on utilise un compteur pour modifier le sprite et chaque sprite a une dimension de 50*50 pixels. Pour cela, l'image sera mise à jour avec un déplacement du sprite de 10 pixels. Pour être plus efficace, au lieu d'utiliser une dizaine d'image, on condense tout sur une seule image qui sera transformée en texture puis rognée pour ne sélectionner que le sprite de 50*50px voulu pour l'animation comme nous le montre la documentation sur le site SFML:



On applique donc ce principe pour notre personnage Wall-e:

- La graine et la plante :

Chaque fois que la graine est plantée dans un bloc fertile, elle se transformera en une plante qui va grandir jusqu'à maturité.

- On peut changer les sprites avec ce que l'on veut en utilisant d'autres images dans les dossiers "images" et "sprites".

2. Les mécanismes du jeu

-Le principe de perte de point :

Chaque fois que l'on se déplace le joueur perd des points (10pts) jusqu'à ce que ces points atteignent 0 et la partie se termine.

-La radioactivité :

On utilise un bloc spécifique pour la radioactivité.
Chaque fois que le personnage met un pied dessus il perd des points de vie jusqu'à en mourir. Il y a une animation où on voit le personnage prendre des dégâts en devenant rouge et retourner à la case précédente pour éviter d'être infecté (undo automatique = retour en arrière automatique).

Si une graine a été poussée sur du sol radioactif, la partie sera terminée et le joueur a perdu.

-Retour en arrière possible:

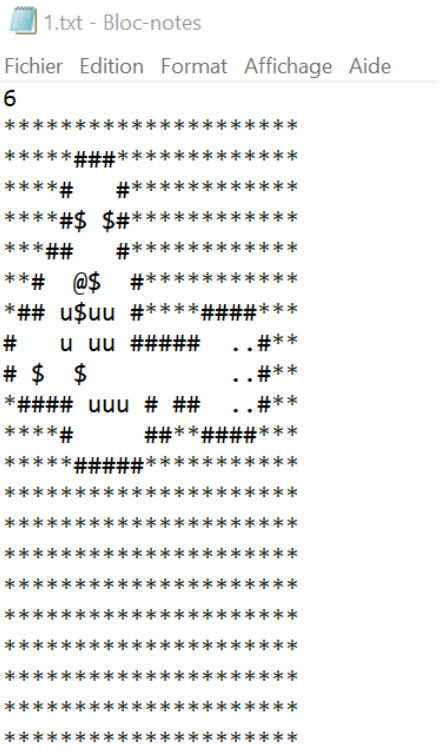
On peut refaire un retour en arrière sur la case précédente, par contre chaque fois qu'on fait cela, on perd 5 pts de vie.

3. Des niveaux implémentables :

On peut modifier les niveaux de jeu, pour cela il faut modifier les fichiers texte dans le dossier "levels" en leur donnant le numéro du niveau que l'on peut ajouter. Pour que le jeu marche correctement, le fichier doit commencer par le nombre de graines que l'on doit planter et la carte doit être sur 21*21 caractères. Puis on construit la carte avec des blocs symbolisés avec des caractères ou des lettres:

- En dehors de la carte : '*'
- murs : '#'
- Joueur : '@'
- Graines : '$'
- partie radioactive : 'u'
- But : '.'
- Espace : '+'
- Vide : ' '

exemple:



4. Amélioration de la fluidité du jeu

Dans la mesure où chaque implémentation entraîne l'augmentation d'évènement dans la boucle de jeu (fonction gameLoop()), par l'ajout de nouvelles conditions à vérifier dans la loop du jeu, le jeu devient de plus en plus lent car il doit vérifier les conditions tant que la fenêtre est ouverte. Afin de rendre le jeu plus fluide, on utilise un booléen qui est à VRAI lorsque le joueur est en train de bouger et FAUX sinon.

L'astuce est de vérifier les évènements que lorsque le joueur ne bouge plus, c'est-à-dire lorsque le booléen est à FAUX. Dans le cas où le booléen est vrai, on ne fait que la mise à jour du joueur (méthode update_player()). Sinon, on vérifie chaque événement (clavier, souris, et les mécanismes du jeu).

5. La fin du jeu

Bouton de retour au menu principal

Bouton pour passer au prochain niveau

Projet

Projet C++

Date de rendu le **TBD**

Barème provisoire:

Rapport /5					Code /20													Fonctionnalités /14						
Mise en forme	Install	Description de l'appli	Fiertés	diag UML	Propre	Commenté	erreur valgrind	Compil sans err/sans warning	Makefile	8 classes	3 niveaux de hierarchie	2 fonctions virtuelles	2 surcharges d'opé	conteneurs	Test Unitaires	Plus	Jouable	Beau	Intéressant	Complex	Drôle	Conception (hierarchie a un sens)
1	1	1	1	1	1	1	1	1	1	2	2	2	2	3	2	2	2	2	2	2	1	5