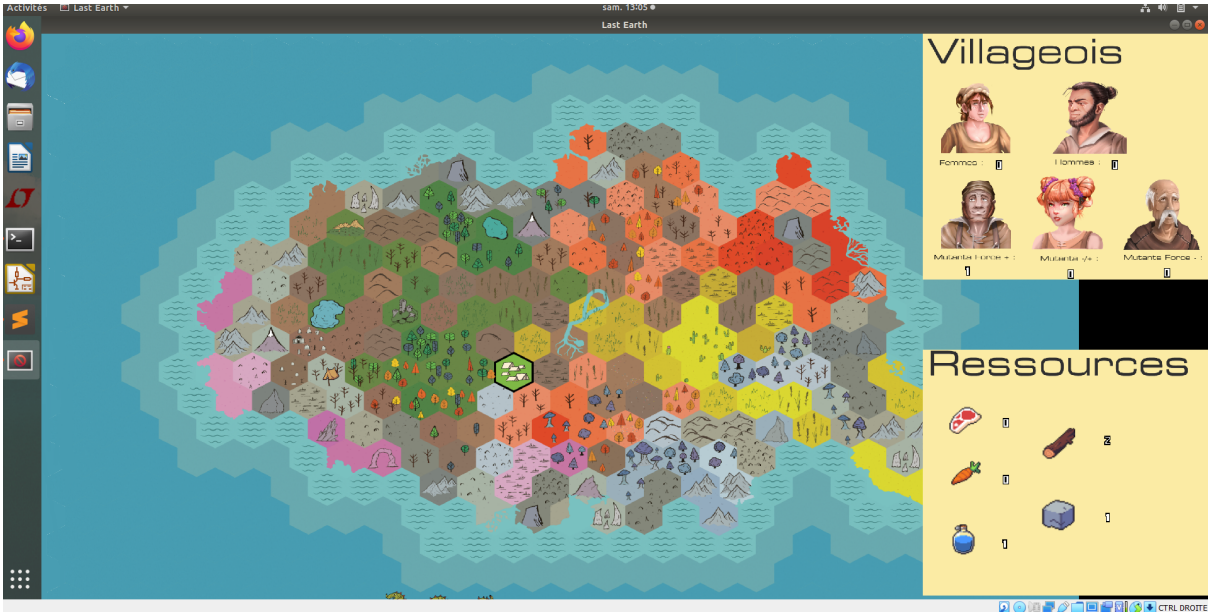
Maï Ghalioun & Hadrien Gourdet
EISE - 4

Last Earth :
Un jeu de gestion post-apocalyptique

POLYTECH SORBONNE

Maï Ghalioun & Hadrien Gourdet
EISE - 4

L'Application :



Last Earth est un **jeu de gestion** qui se joue par tour. Le joueur contrôle une caravane de **villageois** (initialement composée d'un homme et d'une femme). Cette caravane possède un certain nombre de **ressources de 5 types** possibles (viande, légume, eau, bois et pierre).

A chaque tour, la carte se découvre un peu plus (ici un screen de la fin du jeu). Le joueur découvre donc qu'il est sur une **île aux ressources limitées…**
Un **tour** est constitué de plusieurs étapes :

- Le joueur **choisit une direction** où se diriger via les touches **Z,E,D,X,W,Q.**
- Le joueur **récolte** automatiquement les ressources présentes sur la case (ce n'est possible qu'une seule fois, les ressources s'épuisent).
- Le joueur **nourrit ses villageois** (chaque type ayant ses spécificités, que nous discuterons plus bas) en appuyant sur le **gros bouton bleu** apparu
- Le joueur **choisit de reproduire** ses villageois ou non (**petit bouton bleu ou bouton rouge**). **Il faut appuyer sur les personnages dans la zone Villageois, une fois par personnage** (par exemple une fois sur **l'image de la femme** puis une fois sur **l'image de l'homme**). Après deux petites secondes, on voit qu'un nouvel individu s'est ajouté si la reproduction a fonctionné, si un paramètre n'est pas bon, un message apparaîtra dans le terminal et le jeu continuera.
- Les villageois qui n'ont pas été nourris ou qui ont atteint la fin de leur vie **meurent** et le prochain tour commence.

Il y a au total **8 tours**, le nombre de tours peut être adapté via l'argument *tours* du constructeur de la classe Game (dans le fichier *main.cpp*).

Chaque type de villageois a un comportement différent de reproduction et de consommation des ressources :

- Les hommes et les femmes ne consomment **que des ressources alimentaires**. Et leur **reproduction peut donner les 5 types** avec une probabilité **plus élevée pour un humain**.
- Les mutants_pm peuvent consommer n'importe quelle ressource, de préférence les ressources alimentaires, ce qui les met en concurrence avec les humains, mais dans le pire des cas ils **peuvent se contenter de bois ou de pierre** (comme des termites du futur).
- Les mutantsF_minus consomment n'importe quelle ressource de la même manière que les mutants_pm, mais **deux ressources** à la place d'une.
- Les mutantsF_plus **ne consomment pas** de ressources et ne meurent que de vieillesse.

Chaque type de personnage a une **espérance de vie différente**, qui varie de 5 tours pour un humain et un mutant_pm, à 2 tours pour un mutant_minus.
Elle est de 4 tours pour un mutant_plus.
Le jeu est **gagné** si à la fin du nombre de tours, il reste **au moins un personnage en vie**. Le jeu est **perdu** si **tout le monde meurt**.

Le jeu peut être **mis sur pause** et les **règles du jeu** peuvent être lues sur l'écran de pause accessible par la touche **Echap.** Le jeu est réintégré en appuyant sur le **bouton en forme de planète Terre**.

Installation du jeu :

La partie détaillant les règles du jeu et la méthode d'installation sont aussi détaillées dans le **README.txt** (qui contient une partie additionnelle détaillant les tests).
Le projet a été réalisé à l'aide de **SFML en C++**.
Il utilise des appels systèmes POSIX et est donc **uniquement compatible environnement Linux**. Le jeu a été développé sous Ubuntu 18.04.6 LTS.

Sur votre environnement Linux, il faut installer les paquets SFML permettant de compiler correctement le projet.
Entrez la ligne suivante sur le terminal pour permettre l'installation :

sudo apt-get install libsfml-dev

Cela devrait fonctionner sur la plupart des systèmes Debian. Au cas où l'installation ne se ferait pas correctement, il est possible de télécharger manuellement les paquets puis de les décompresser à l'endroit souhaité pour pouvoir ensuite les compiler en spécifiant le chemin des librairies.
Il faut de préférence les copier dans le chemin standard **"/usr/local"** pour pouvoir utiliser le Makefile fourni.
Voici le lien de téléchargement :

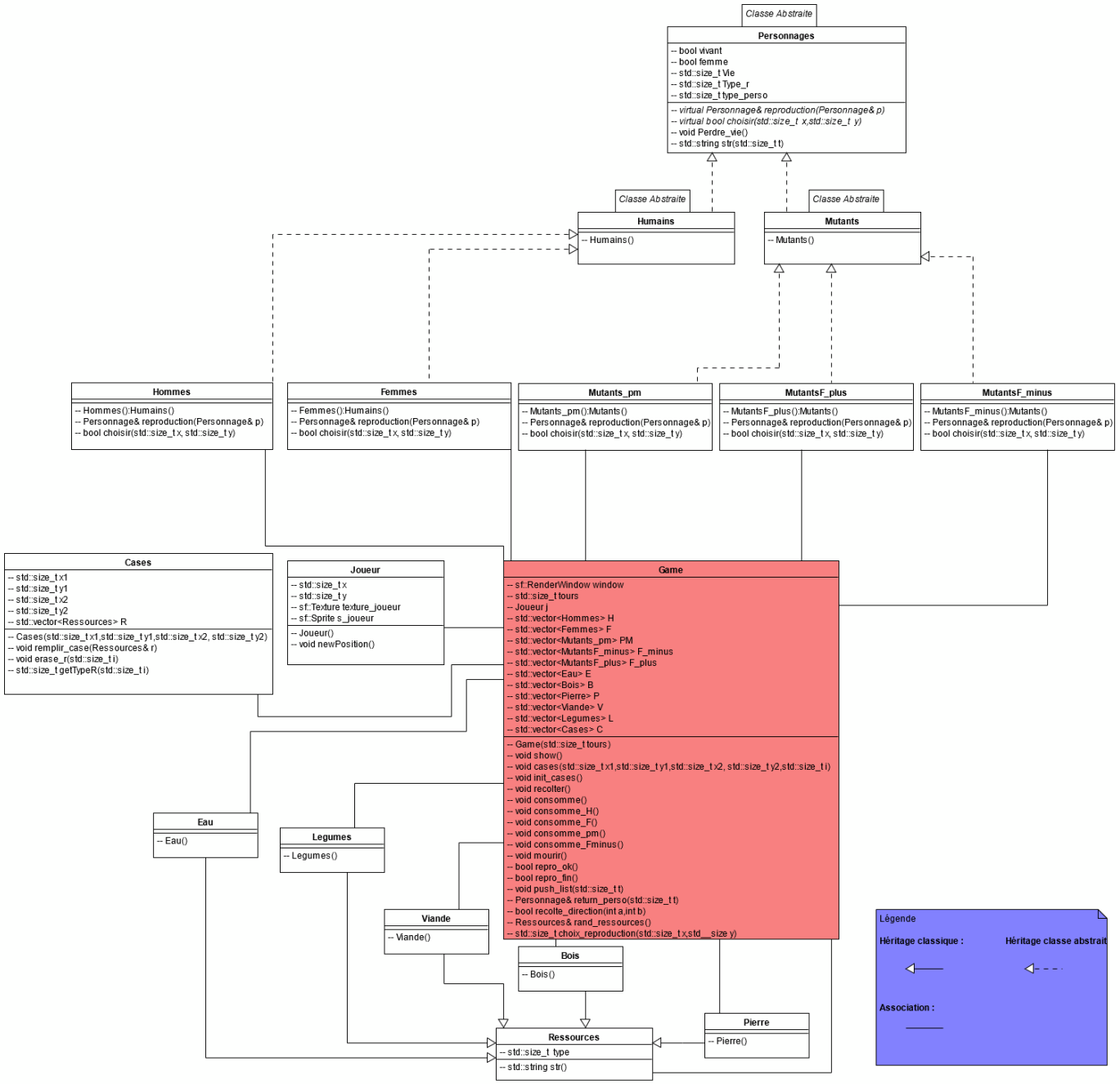
https://www.sfml-dev.org/download-fr.php

Une fois l'installation terminée, il faut télécharger le dossier du jeu et le décompresser. Un **Makefile** est fourni permettant de compiler le jeu avec l'instruction **"make"**. Le Makefile permet de compiler les testcases avec l'instruction **"make testcases"**.
Il suffit de lancer le jeu par **"./Last_Earth"**. Pour lancer les testcases, il faut entrer **"./testcases"**.

Maï Ghalioun & Hadrien Gourdet
EISE - 4

Diagramme UML :

Pour ne pas encombrer le diagramme plus que de raison, les getter et les setter ne sont pas spécifiés dans les méthodes. Le diagramme UML est aussi accessible directement sur le git.



Maï Ghalioun & Hadrien Gourdet
EISE - 4

Contraintes imposées :

8 Classes et 3 Niveaux de hiérarchie :

- **Ressources -> Eau, Legume, Viande, Bois et Pierre**
  Ici, les 5 classes héritent de la classe mère Ressources. Ressources est utile pour instancier un vecteur de Ressources pour chaque case qui soit aléatoirement rempli (fonction *rand_ressources()*). Je ne l'ai pas rendu abstraite car je ne voyais pas spécialement l'utilité d'une fonction virtuelle dans ce cas. Chaque classe fille représente un type différent de ressources, qui sont gérées dans le jeu par une liste propre à chacun.
- **Personnages -> Humains & Mutants -> Hommes, Femmes, Mutants_pm, MutantsF_minus, MutantsF_plus**
  Personnages est la classe abstraite dont héritent Humains et Mutants, elles-mêmes des classes abstraites. Ces classes contiennent deux méthodes virtuelles et sont donc non instanciables. Les classes filles Hommes, Femmes, Mutants_pm, MutantsF_minus et MutantsF_plus contiennent chacun une redéfinition des méthodes virtuelles. Ces trois niveaux de hiérarchie permettent de spécifier le comportement de chaque type de villageois différent.
- **Game**
  La classe Game est la "Fat Class" du programme. Elle contient les fonctions nécessaires au jeu et à la résolution de chaque tour mais aussi des vecteurs et des listes de Personnages, Cases, Ressources et bien sûr le joueur. Le jeu est lancé par la fonction *show()* de cette classe dans le main.cpp.
- **Cases**
  La classe Cases contient les coordonnées de la case mais aussi les ressources que nos personnages peuvent récolter. Un vecteur de Cases permet de modéliser la carte dans le programme.
- **Joueur**
  La classe Joueur contient le Sprite ainsi que les coordonnées du joueur. Elle permet de le modéliser dans le programme.

2 conteneurs différents de la STL :

- **vecteur**
  Les personnages sont gérées par des vecteurs de chaque type : Hommes, Femmes, Mutants_pm, MutantsF_minus et MutantsF_plus. Les cases sont aussi gérées par un vecteur, en parcourant ce vecteur on peut savoir où se trouve notre joueur sur la carte. La classe Cases contient aussi un vecteur de Ressources qui stockent les ressources que nos villageois peuvent trouver dans cette zone.
- **liste**
  Les ressources Eau, Legumes, Viande, Bois et Pierre sont gérées par la Fat Classe Game via des listes. Les fonctions de la classe Game sont réfléchies pour des vecteurs et donc aussi des listes, car ces deux conteneurs ont beaucoup de fonctions en commun. Le choix de deux conteneurs similaires a été fait pour faciliter le code.

2 fonctions virtuelles :

- **Personnage& reproduction(Personnage& p)**
  Cette méthode virtuelle présente dans la classe Personnages permet de récupérer un nouveau personnage en en choisissant deux. Chaque type de personnage ne présente pas les mêmes probabilités de donner tel ou tel type de Personnages. Les mutants ne peuvent donner que des mutants alors que les humains peuvent donner les 5 types avec une probabilité plus élevée pour les humains.
- **bool choisir(std::size_t x,std::size_t y)**
  Cette méthode est complémentaire de celle de la reproduction, elle permet de savoir sur quel personnage le joueur a appuyé.

2 surcharges d'opérateurs :

- **std::ostream& operator<<(std::ostream& out, Cases& c)**
  Cette surcharge d'opérateur permet d'afficher les ressources à l'intérieur de la zone par leur nom dans le terminal. Si la zone est vide, un message l'indiquant apparaît.
- **template<class T> std::ostream& operator<<(std::ostream& out, std::vector<T> p)**

Cette surcharge d'opérateur est sous forme de template pour pouvoir être directement utilisée avec les classes filles de Personnage. Cette fonction permet d'afficher après l'étape de reproduction la liste de personnages du type du nouveau personnage créé ainsi que leur vie restante.

Tests unitaires :

Le but de ces tests unitaires est de couvrir un maximum des fonctions du code. Le Testcase est composé de 9 tests différents soit 19 asserts. Les tests permettent de vérifier le bon déroulement des fonctions de la classe Personnage, de la classe Cases et bien sûr de la classe Game. Ces tests ont été volontairement réalisés sur les fonctions au fonctionnement le plus critique pour le jeu. Un compte-rendu plus détaillé du but de chaque Test est accessible sur le README.txt

Fiertés :

Je n'avais jamais réalisé un jeu en entier en orienté objet avec une implémentation graphique. Je n'avais donc non plus jamais utilisé de librairies graphiques ni SFML. Pour une première fois, je suis plutôt fière du résultat graphique. J'ai dessiné la carte moi-même à l'aide d'un logiciel d'aide à la réalisation de cartes pour jeu de rôle. Et j'ai essayé de trouver des sprites libres de droit pour l'implémentation graphique.
Le jeu est entièrement jouable en cliquant sur les boutons ou les personnages et sans passer par le terminal, et le mouvement du joueur se fait par touche ce que je trouve plus agréable qu'appuyer sur un bouton "est", "ouest", etc.
Je suis assez satisfaite de l'ambiance générale du jeu avec la musique et l'interface graphique, aussi les petits sprites de nombre qui se mettent à jour à chaque tour.

Pour ce qui est du code, j'ai réussi à découper le tour en plusieurs parties à l'aide de flags à la manière un peu d'une machine à état, ce qui a permis d'afficher les bons sprites à chaque étape.

Avec tous les projets que nous avions en EISE, finir à temps est déjà ma plus grande source de fierté. Je pense avoir intégré toutes les contraintes, et utiliser à bon escient template et surcharges d'opérateurs.