Projet de C++ :

Last Day on Earth

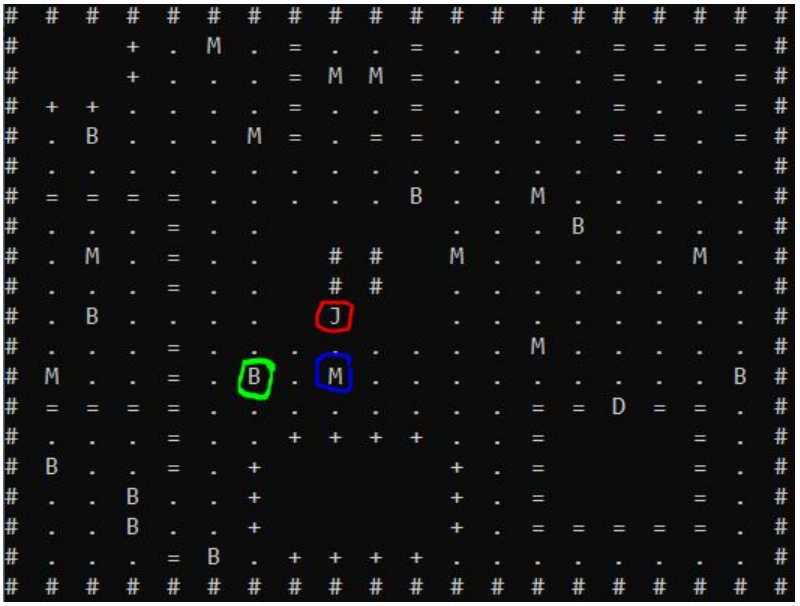
Élèves-ingénieurs : Hadrien Gourdet, Florent Hamelet-Delval
Encadrante : Mme Braunstein
Institution : Polytech Sorbonne (campus Jussieu)
Spécialité : Electronique et Informatique en systèmes embarqués (année 4)
Années : 2020 - 2021

Sommaire :

I) Introduction sur le jeu

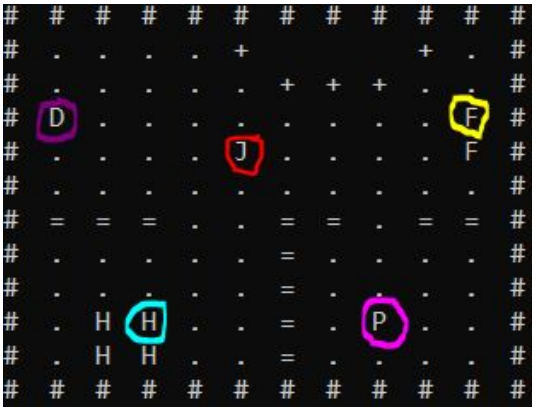
Dans le cadre du module de programmation C++, nous devions développer une application respectant le thème "Le monde d'après". Nous avons donc décidé de réaliser le jeu "*Last Day on Earth*". Ce dernier est un jeu 1D de type survie. Son affichage se fait sur un terminal, une fois que la commande d'exécution de l'application est rentrée.

Dans le principe, vous incarnez Ken, survivant d'un terrible bombardement nucléaire ayant ravagé une partie de la planète Terre.



Au sein de la région du WasteLand, Francis doit survivre à tout prix. A cette fin, il doit chercher des ressources vitales (eau et nourriture) pour ne pas finir six pieds sous terre. Sur son chemin, il sera amené à affronter des bandits armés jusqu'aux dents et à combattre d'horribles monstres mutants. Heureusement pour lui, au cours de sa quête, il aura à disposition des armes et des vivres qu'il aura notamment récupérées sur les corps de ses victimes.

Au sein du WasteLand, Francis a monté une base qui lui sert de campement pour se reposer avant la prochaine exploration. C'est une zone où il peut se sentir en sécurité : aucun ennemi ne peut y pénétrer pour l'attaquer lorsqu'il sommeille.



Plusieurs icônes majeures sont représentés sur les modèles du WasteLand et de la base :

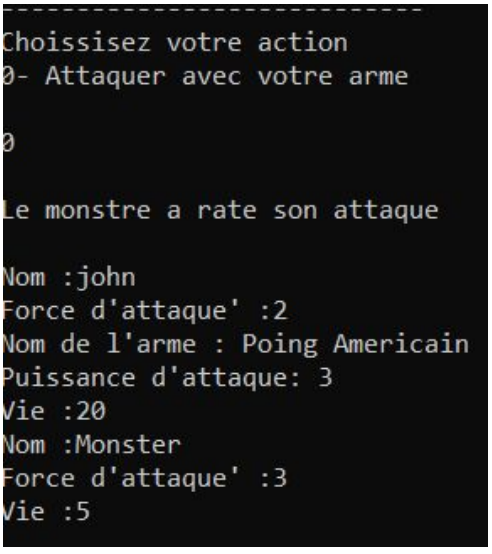
- **J** (Joueur) ⟶ il est doté de 20 PV, 40 points de nourriture et d'eau et se déplace par appui sur les touches **Z** (haut), **S** (bas), **Q** (gauche) ou **D** (droite) suivi d'une commande **Entrée**.
- **B** (Bandit) et **M** (Monstre) ⟶ ils sont dotés de 15 PV et lorsque le joueur les bat, ce dernier récupère soit une arme soit de la nourriture.
- **D** (Délocalisation) ⟶ portail reliant la base et le WasteLand.
- **P** (PNJ) ⟶ informations sur les quêtes (intitulés et nombre restantes).
- H (Home) ⟶ emplacement pour récupérer dormir, le joueur récupère de la santé.
- F (Food) ⟶ emplacement pour manger, le joueur récupère de la nourriture.
- **+** ⟶ délimitations d'une zone pour boire, le joueur récupère de l'eau.

D'autres icônes mineures sont :

- **=** ⟶ fondations de la base.
- **#** ⟶ bords du WasteLand ou de la base.
- **.** ⟶ zone accessible par le joueur.

Par ailleurs, certaines commandes peuvent être effectuées : **M** (accéder au menu du jeu) ou **E** (quitter un menu ou une notification).

Notamment, le menu propose plusieurs actions : **A** (affichage des PV et des niveaux de nourriture et d'eau du joueur), **Z** (liste des armes disponibles), **R** (quitter le jeu), **T** (intitulé de la quête actuellement active), **Y** (intitulé de l'arme active), **U** (changer d'arme active ⟶ si la première, rentrer indice 0 ; si la seconde, indice 1 ; ect.) et **O** (manger une nourriture ⟶ même principe).

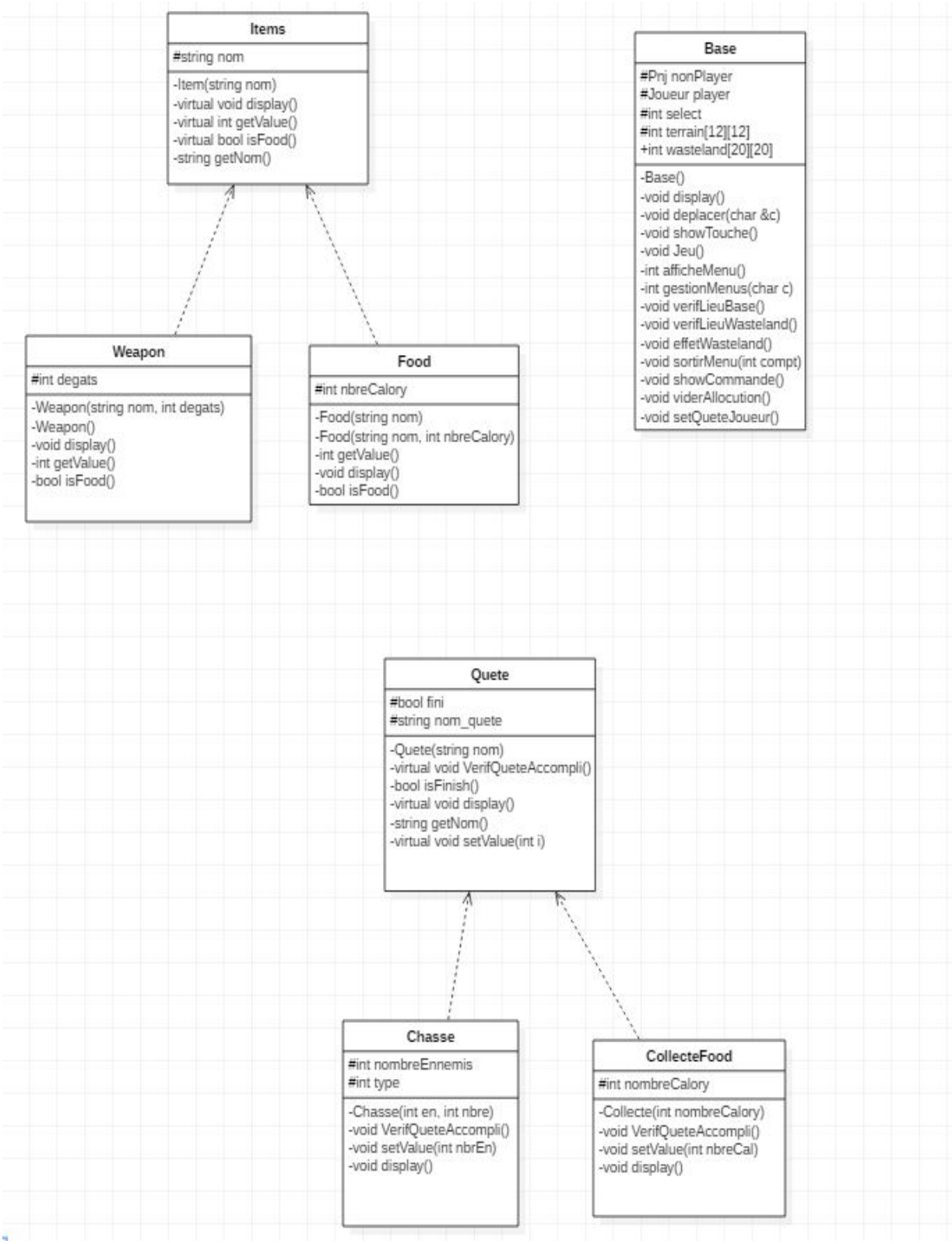


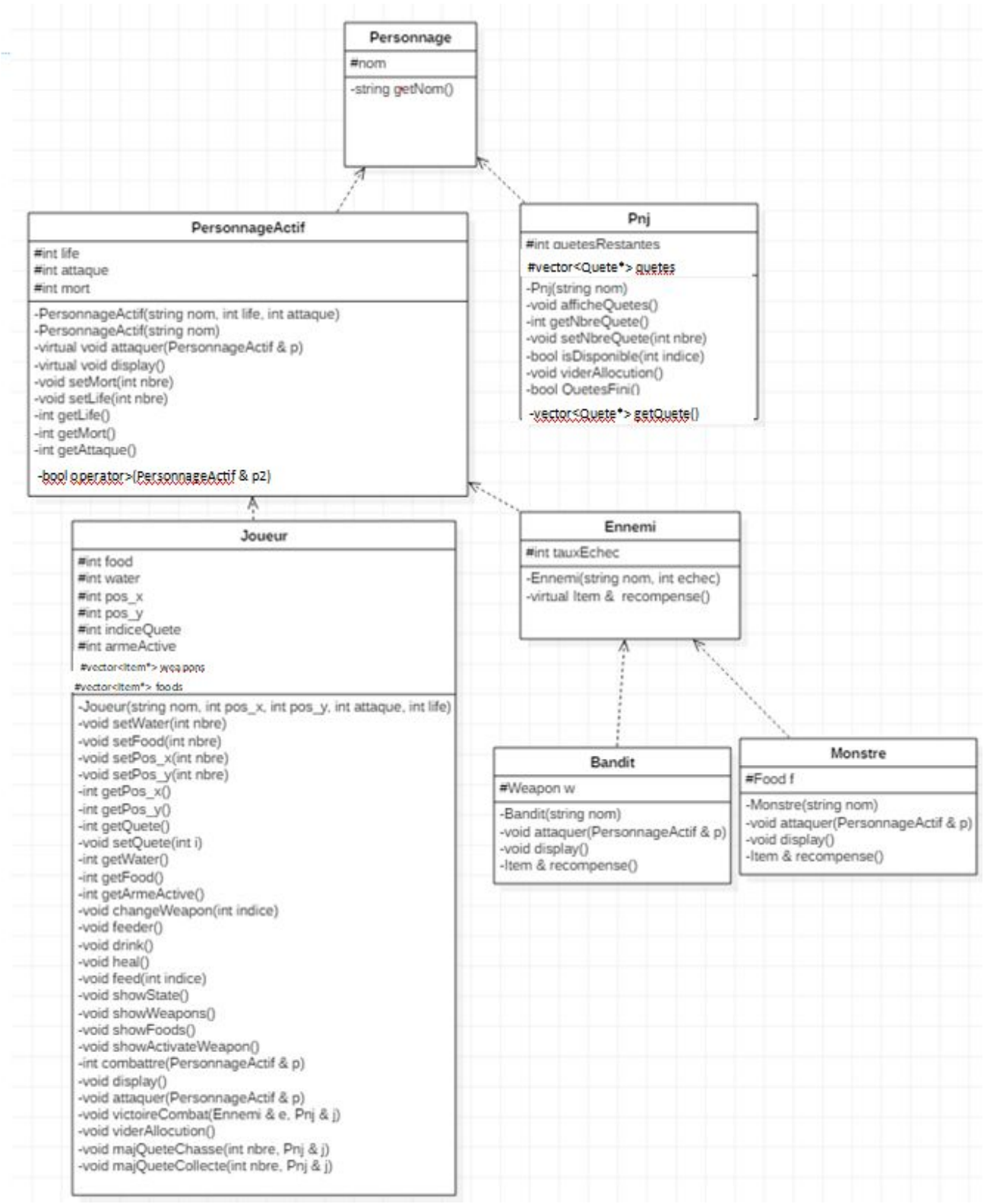
En phase d'exploration, le joueur perd une unité de nourriture et d'eau par déplacement effectué (excepté dans sa base). Lorsqu'il manque de l'un ou de l'autre, il perd un PV à chaque déplacement.

En phase de combat, les informations propres au joueur (PV, arme actuelle et dégâts d'attaque = force du joueur + puissance de l'arme) et à son adversaire (PV et force d'attaque) sont affichées à chaque tour. Le joueur joue en premier. Une seule action est réalisable : l'attaque. S'il lui reste moins de PV, ce dernier se voit octroyer un bonus de dégâts x2. Par ailleurs, l'attaque de son adversaire peut échouer.

II) Diagramme UML

Nous ne représenterons ici que la partie statique du diagramme UML, c'est-à-dire le diagramme de classes. Nous ne nous intéresserons pas aux aspects temporels et dynamiques. Le diagramme UML vise à décrire clairement la structure d'un système particulier en modélisant ses classes, ses attributs, ses opérations et les relations entre ses objets. Pour le "*Last Day on Earth*", le diagramme de classes UML sera le suivant.




Personnage
#nom
-string getNom()
PersonnageActif
#int life
#int attaque
#int mort
-PersonnageActif(string nom, int life, int attaque)
-PersonnageActif(string nom)
-virtual void attaquer(PersonnageActif & p)
-virtual void display()
-void setMort(int nbre)
-void setLife(int nbre)
-int getLife()
-int getMort()
-int getAttaque()
-bool operator>(PersonnageActif & p2)
Pnj
#int quetesRestantes
#vector<Quete*> quetes
-Pnj(string nom)
-void afficheQuetes()
-int getNbreQuete()
-void setNbreQuete(int nbre)
-bool isDisponible(int indice)
-void viderAllocution()
-bool QuetesFini()
-vector<Quete*> getQuete()
Joueur
#int food
#int water
#int pos_x
#int pos_y
#int indiceQuete
#int armeActive
#vector<Item*> weapons
#vector<Item*> foods
-Joueur(string nom, int pos_x, int pos_y, int attaque, int life)
-void setWater(int nbre)
-void setFood(int nbre)
-void setPos_x(int nbre)
-void setPos_y(int nbre)
-int getPos_x()
-int getPos_y()
-int getQuete()
-void setQuete(int i)
-int getWater()
-int getFood()
-int getArmeActive()
-void changeWeapon(int indice)
-void feeder()
-void drink()
-void heal()
-void feed(int indice)
-void showState()
-void showWeapons()
-void showFoods()
-void showActivateWeapon()
-int combattre(PersonnageActif & p)
-void display()
-void attaquer(PersonnageActif & p)
-void victoireCombat(Ennemi & e, Pnj & j)
-void viderAllocution()
-void majQueteChasse(int nbre, Pnj & j)
-void majQueteCollecte(int nbre, Pnj & j)
Ennemi
#int tauxEchec
-Ennemi(string nom, int echec)
-virtual Item & recompense()
Bandit
#Weapon w
-Bandit(string nom)
-void attaquer(PersonnageActif & p)
-void display()
-Item & recompense()
Monstre
#Food f
-Monstre(string nom)
-void attaquer(PersonnageActif & p)
-void display()
-Item & recompense()

III) Description de la classe majeure

La classe majeure dans notre projet de jeux vidéo sur le thème “Le monde d’après” est la classe *Base*. Ce qui rend cette classe intéressante est qu’elle utilise l’ensemble des objets que nous avons définis précédemment de façon directe ou indirecte.

De plus, cette classe possède une méthode *jeu()* qui permet à l’application de tourner. Pour continuer, il est important d’indiquer que c’est la classe qui a demandé le plus de temps. Pour conclure, cette classe est très importante à nos yeux car le jeu a fonctionné pour la première fois suite à l'aboutissement de celle-ci.

IV) Procédure d’exécution du code

- Pour l'exécution du code, il faut lancer le fichier nommé *LastDayOnEarth.exe,* soit par l’intermédiaire d’un terminal (dans le répertoire du jeu, commande *./LastDayOnEarth)* ou par le biais d’un explorateur de fichiers (double-clique sur l’exécutable).

Il est important de notifier que nous n’utilisons pas de librairie spécifique. Par conséquent, il n’y a aucune bibliothèque à installer de votre côté.

- Pour la compilation, il y a dans l’archive de jeu envoyée un Makefile, sans tests unitaires. Il vous suffit de faire la commande Make dans le répertoire du jeu sur un terminal, pour créer un nouvel exécutable. Autrement, vous pouvez ouvrir l’archive du jeu sur le logiciel Dev C++ sur lequel nous avons développé “Last Day on Earth”.