Bordeaux INP
ENSC

Programmation avancée - Projet 2021

Création d'un Roguelike

- 1ère Année FILIÈRE INGÉNIEUR -
Livrable - Semestre 6

Emma Marqueton
&
Angéline Rondeau

Synthèse technique de l'application

Création d'un Roguelike : PfeverQuest

Notre jeu est un Roguelike inspiré de l'épidémie de Covid-19. Le joueur incarne un anticorps dont l'hôte se voit administrer une dose d'un vaccin (dont nous tairons le nom) contre le virus. L'objectif est de récupérer les fragments de virus à travers différentes parties du corps de l'hôte afin de développer son immunité. Il lui faudra éviter des hordes de bactéries et de caillots sanguins tout en maîtrisant son nombre de déplacements, qui est limité. Pour cela, le joueur aura la possibilité de récolter différentes ressources / items qui augmenteront sa force d'attaque, son nombre de déplacements disponibles, ou sa résistance au cours de la quête.

Ce jeu a été conçu en langage C# et suit les règles de la programmation orientée objet. Il a été testé comme fonctionnel sur **Visual studio 2019.** Nous exposerons dans ce rapport les choix techniques, les différents tests effectués et nous présenterons une brève notice d'utilisation du jeu.

I. Déroulement du jeu

La partie débute par le choix de l'avatar de notre anticorps. Le joueur est ensuite confronté à sa quête des fragments de virus réparties en 5 niveaux de difficultés croissantes.

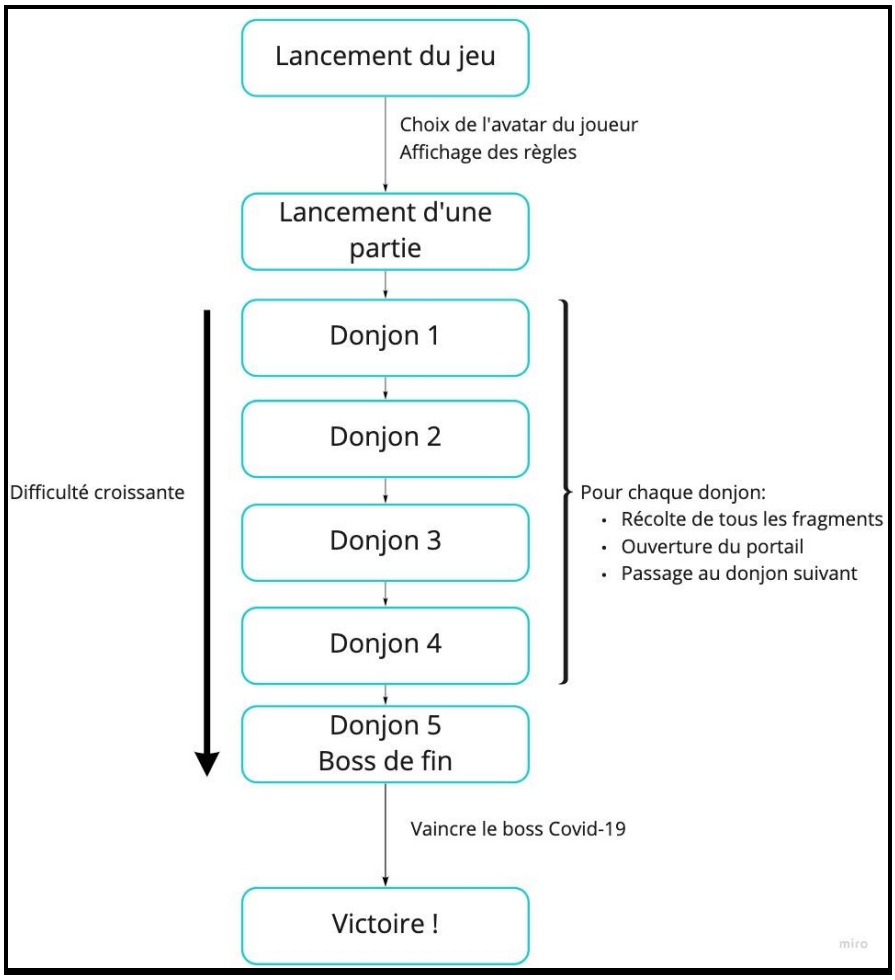

Figure 1: Schéma du déroulement d'une partie complète

Le joueur commence la partie avec 20 points de vie, et doit récolter tous les fragments d'un niveau avant de pouvoir accéder au suivant. Les attaques ennemis lui font perdre de la vie, mais il a la possibilité d'attaquer de son côté, et de récupérer de la vie en récoltant de la nourriture.

La partie prend fin dans 3 situations différentes: si le joueur a récupéré l'ensemble des fragments des 5 niveaux et battu le boss de fin (c'est une victoire), si le joueur n'a plus de vie ou plus de déplacement disponible (c'est un échec). À la fin d'une partie, le jeu propose d'en recommencer une.

II. Choix techniques

Dans cette partie, nous allons présenter les différents choix techniques que nous avons fait pour réaliser ce jeu.

A. Diagramme des classes

Diagramme de classe en annexe

Notre jeu se lance dans ***Programme.cs*** qui contient l'exécutable. Ce fichier crée une partie et lance sa simulation (*jouerPartie()*). Il propose au joueur de relancer une partie en appuyant sur espace dès qu'une vient de se terminer (victoire ou défaite).

Notre jeu est articulé autour de la classe ***Partie***. Lorsqu'on crée une nouvelle Partie, le constructeur de la classe crée un joueur Anticorps (relation de composition) après avoir demandé au préalable le symbole que le joueur souhaiterait utiliser comme avatar. Il crée et range ensuite dans une liste 5 donjons dont les niveaux vont croissant, le dernier contenant le boss de fin. Il y a là aussi une relation de composition entre la classe ***Donjon*** et ***Partie***. Cette dernière contient des méthodes qui vont permettre d'afficher du texte en début et en fin de partie. Elle contient aussi la méthode *JouerPartie()**,*** qui affiche les règles puis lance la simulation de chacun des donjons en appelant les méthodes *JouerDonjon()* de la classe ***Donjon***. Cette méthode permet aussi d'articuler les différents donjons entre eux (sortir de la partie quand on perd, introduire le boss final, féliciter le joueur à la fin de la partie en cas de victoire).

Chaque niveau est articulé autour de la classe ***Donjon***. Le constructeur de la classe se charge de générer un donjon avec un certain niveau de difficulté, d'y associer un joueur et d'y intégrer les items (***FragmentVirus***, ***Nourriture***, ***Arme***) et les ennemis (***CaillotSanguin***, ***EscherichiaColi***). Le niveau de difficulté va impacter la taille du donjon et le type d'ennemi (plus on augmente dans les niveaux, plus les ennemis qui font le plus de dégâts ont une probabilité importante d'apparaître). La méthode *JouerDonjon*() permet la gestion du déroulement d'un niveau, avec les déplacements des personnages et leurs interactions avec l'environnement, comme la récolte d'items et les combats PVE (Player vs Environment).

De plus, les déplacements du joueur et des ennemis sont gérés différemment: les mouvements des ennemis sont rythmés par un Timer déclaré dans *JouerDonjon*(), tandis que le joueur peut se déplacer sans contrainte de temps simplement à l'aide des flêches directionnelles.

La gestion du dernier donjon qui contient le boss final est légèrement différente : aucun item n'est généré, seul l'ennemi ***Covid19*** est ajouté à la liste, la taille du donjon est prédéfinie, une barre de vie pour l'ennemi s'affiche et il n'est pas nécessaire de passer un portail pour gagner.

La classe abstraite ***Personnage***, qui contient la position, le niveau de vie, les dégâts infligés et symbole d'un personnage, est dérivée en 2 classes :

- La classe ***Ennemi***, elle-même dérivée en ***CaillotSanguin***, ***EscherichiaColi*** et ***Covid19***. Ces dernières instancient le symbole, les dégâts et la vie des ennemis. De plus, la méthode *Action*() de ***Personnage*** est override dans les trois classes filles pour déclarer un pattern de déplacement unique à chaque ennemi. Le Caillot sanguin se déplace en ligne tandis que Escherichia Coli suit notre personnage. Covid19 se comporte comme Escherichia Coli mais peut se déplacer en diagonale et possède beaucoup plus de points de vie car il est le boss de fin de notre jeu.

- La classe ***Anticorps***, qui est la classe du joueur. Elle possède comme attributs le nombre de pas, ses dégâts de base (qui pourront augmenter par la suite avec les armes), et sa vie. La méthode *Action*() est override pour permettre au joueur de se déplacer avec les flèches directionnelles du clavier, et d'attaquer avec la barre espace. Afin d'attaquer, la méthode prend en paramètre un donjon (le donjon dans lequel le joueur se trouve) pour vérifier si un ennemi se trouve sur une case à côté du joueur en faisant le lien avec les positions ennemis stockées dans la classe Donjon.

La classe ***Item***, qui contient la position d'un item, est dérivée en 3 classes :

- La classe ***Arme***, elle-même dérivée en ***CrisprCas9***, ***CrisprCas13*** et ***Transfection***, instancie la propriété de GainDegat. Les trois classes filles instencient la valeur de GainDegat, qui représente ce qu'on ajoute au dégât infligés par le joueur s'il récupère une arme.
- La classe ***Nourriture*** instancie la propriété GainVie = 3, qui représente ce qu'on ajoute à la vie du joueur s'il récupère un item nourriture.
- La classe ***FragmentVirus*** instancie et la propriété GainNombreDePas = 15, qui représente ce qu'on ajoute nombre de pas restants du joueur s'il récupère un item fragment.

B. Tests

Pour tester notre jeu, nous avons testé chaque méthode des donjons individuellement afin de nous assurer que les méthodes soient fonctionnelles, avant d'intégrer toutes les épreuves à la partie. Un test global nous a ensuite permis de tester notre application dans son ensemble afin de déceler d'éventuelles erreurs de conception ou des erreurs lors de la création des différents objets du jeu.

1. Test des donjons

Le tableau ci-dessous résume les tests effectués pour vérifier le bon fonctionnement de chaque donjon.

Fonctionnalité testée	Méthode
Respect des règles	Tester plusieurs fois le jeu afin de s'assurer que tous les cas possibles ont été pensés. Par exemple, bien générer les personnages et items à l'intérieur des murs des donjons.
Gestion des erreurs	Appuyer sur différentes touches du clavier pour tester la gestion des erreurs lors des saisies, notamment lorsque le joueur attaque les ennemis.
Fin de la partie	Perdre et gagner pour vérifier que le jeu affiche bien le portail dans le cas où tous les fragments ont été collectés.
Affichage	Nous avons affiché toutes les caractéristiques d'un donjon (taille, nombre d'ennemis et d'items ...) et affiché la grille de jeu pour vérifier la cohérence entre les valeurs prises par ces caractéristiques et l'affichage de la grille de jeu

Ces tests nous ont permis de trouver différentes failles dans nos donjons puis de les combler en fonction des situations rencontrées.

2. Test des personnages

Pour les ennemis, nous les avons fait apparaître dans un donjon et avons vérifié qu'il respectait bien leur pattern de déplacement respectif (ligne horizontale pour ***CaillotSanguin***, suivi du joueur pour ***EscherichiaColi***). Nous avons aussi testé les attaques et leur efficacité sur le joueur. Enfin nous avons vérifié que les ennemis effectuaient l'ensemble de leurs actions en fonction du Timer.

Pour le joueur, nous avons testé les déplacements et les interactions avec l'environnement: récolte d'item, attaque des ennemis, passage à travail le portail.

3. Test de la partie

Les tests sur la partie nous ont permis de tester l'application dans sa globalité.

Partie testée	Comment
Enchaînement des donjons	Lancement de plusieurs parties en vérifiant l'affichage des informations et que le niveau de difficulté augmente au fil des donjons.
Fin de partie	Perdre ou gagner afin de voir la gestion des fins possibles. Si le joueur n'a plus de vie ou de déplacement, la partie doit s'arrêter et proposer de rejouer.

Ces tests nous ont permis de rectifier les derniers problèmes auxquels nous avons été confrontés dans l'application, afin de proposer un jeu de type Roguelike fonctionnel.

III. Notice d'utilisation

A. Donjon et items

Le joueur se déplace librement au sein de 5 différents donjons, limités par des murs. Dans ces donjons, trois types d'items peuvent être récupérés pour augmenter les statistiques du joueur. En bleu, les armes qui augmentent les dégâts infligés, en vert la nourriture qui fait regagner les points de vie perdus (ils ne peuvent pas dépasser la valeur maximale déclarée dans la classe ***Anticorps***), et en jaune les fragments de virus qui redonnent des déplacements au joueur.

Dans les donjons, on peut aussi trouver deux types d'ennemis. Les caillots sanguins se déplacent en ligne droite et infligent peu de dégâts au joueur. Escherichia Coli fait en sorte de suivre le joueur, en inflige davantage de dégâts que les caillots sanguins. Il est possible d'attaquer ou d'éviter les ennemis pour compléter un niveau.

B. Début du jeu

En lançant notre jeu, le joueur doit choisir le symbole qui le représentera au cours de son aventure. Ensuite un texte apparaît pour expliquer les règles au joueur, et propose de lancer une partie. Une fois la partie lancée, notre joueur se retrouve dans le premier donjon.

C. Les épreuves

Le but du joueur est de récupérer tous les fragments de virus en jaune d'un donjon, sans mourir. Il meurt s'il perd toute sa vie ou s'il a dépensé tous ses déplacements. Une fois tous les fragments récupérés, un portail en bas du donjon apparaît pour amener le joueur au donjon d'après. La difficulté augmente au fur et à mesure: davantage d'Escherichia Coli sont susceptibles d'apparaître, et les donjons sont plus grands, ce qui demande une meilleure gestion du nombre de pas disponibles.

D. Boss de fin

Au dernier donjon, le boss final apparaît: le Covid-19. Le joueus doit le vaincre tout en évitant ses attaques.

E. Fin de Partie

La partie prend fin en cas de mort du joueur (plus de vie ou de déplacement) ou lorsque le boss de fin a été vaincu.

IV. Gestion de projet

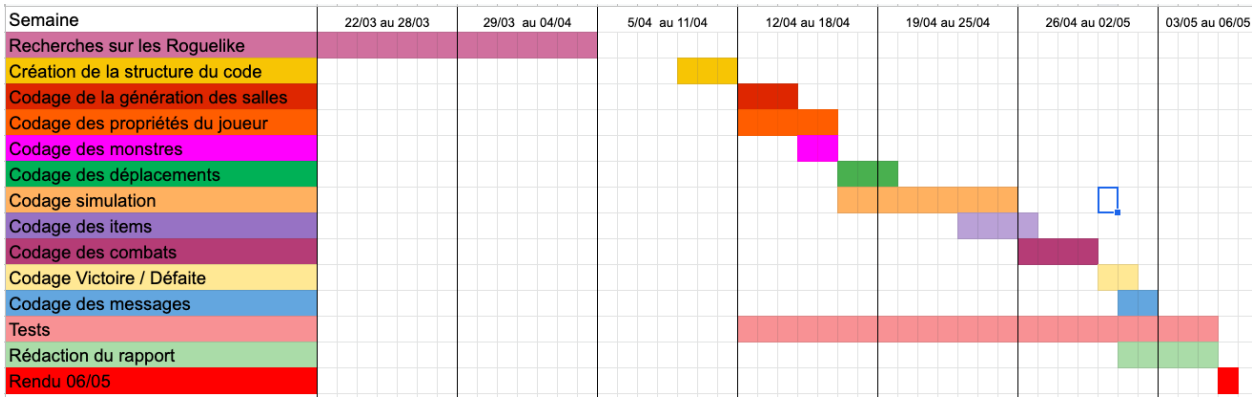

Figure 2: Planning prévisionnel

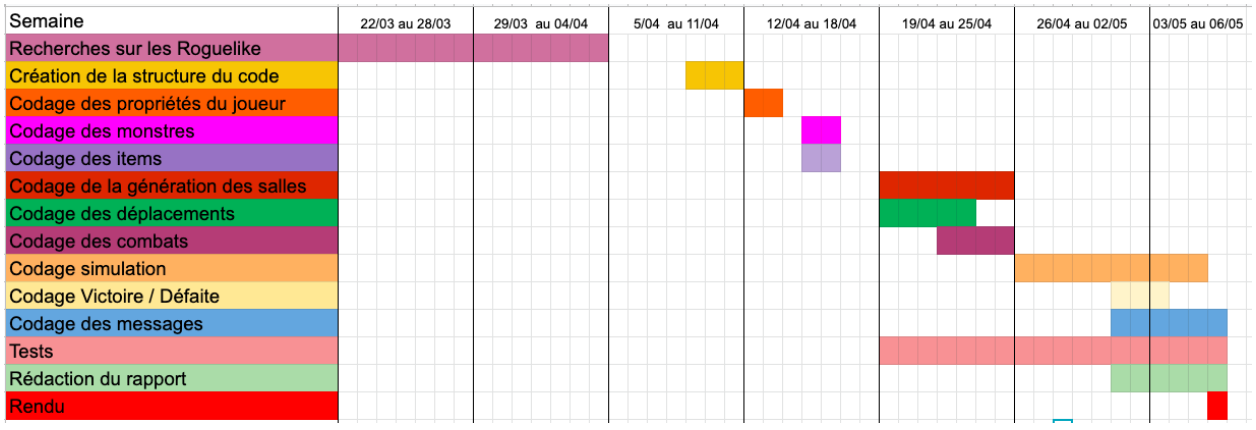

Figure 3: Planning effectif

Le projet a été réalisé en binôme avec GitHub comme plateforme obligatoire de partage du code.
Nous nous sommes répartis équitablement les fonctionnalités à développer lorsque nous étions en distanciel, mais avons favorisé le pair-programming lors des séances de TP à l'école. Cela a permis de régler plus efficacement les problèmes les plus importants que nous rencontrions au fur et à mesure.

Nous avons choisis de faire un planning prévisionnel avec une granularité de tâches fine, ce qui nous a permis de nous rendre compte précisément de la quantité de travail à réaliser. Nous avions plutôt bien évalué cette quantité de travail grâce à la longue phase de réflexion réalisée au préalable. Un travail chronophage qui s'est ajouté était lié à des débugages et des soucis de communication entre le Windows d'Angéline et le Mac de Emma via Git.

Durant le projet, nous nous sommes organisées en séances de travail intensives et avons finalement travaillé sur moins de jours que prévu. Cependant nous n'avions pas anticipé le fait que coder le donjon serait primordial afin de pouvoir faire interagir les différentes classes entre elles et tester leurs interactions. Ceci explique le remaniement de l'ordre des étapes entre le planning prévisionnel et le planning effectif. C'est seulement une fois la classe ***Donjon*** terminée que nous avons pu nous lancer sur la simulation d'une partie complète.

Enfin, nous avons profité du temps qu'il nous restait pour corriger certaines erreurs et enrichir notre jeu de fonctionnalité et de détails. Le boss de fin n'était à l'origine pas prévu par exemple.

V. Conclusion

Finalement, nous avons réalisé un jeu de type Roguelike fonctionnel en programmation orientée objet. Ce projet nous a permis de comprendre l'importance de ce paradigme et surtout la nécessité de la phase de modélisation : bien réfléchir à la hiérarchie des classes et aux différentes relations entre elles avant de commencer à programmer permet d'être plus efficace et d'éviter tout problème nécessitant la modification de l'architecture initiale.

Par la suite il serait envisageable d'enrichir notre jeu, en ajoutant par exemple de nouveaux types de personnage ou items, ce qui est simplifié par nos classes. De plus, nous aimerions que la méthode AfficheDéfinition() soit appelée lors des interactions du joueur avec l'environnement, car actuellement elle est simplement appelée en fin de partie. Par exemple, la définition de CRISPR Cas9 apparaîtrait et mettrait le jeu en pause lorsque le joueur collecterait l'arme ***CRISPRCas9***. Il serait aussi intéressant de rajouter des décorations en ASCII art autour des donjons pour coller davantage au thème "corps humain" de notre jeu.

Une autre idée que nous aimerions incrémenter dans une version future est la conservation des scores dans un fichier mis-à-jour à chaque partie et consultable avant ou après une partie. L'enregistrement d'une partie non terminée afin de la reprendre à la prochaine ouverture du jeu pourrait également être une piste d'amélioration à envisager.

VI. Annexe

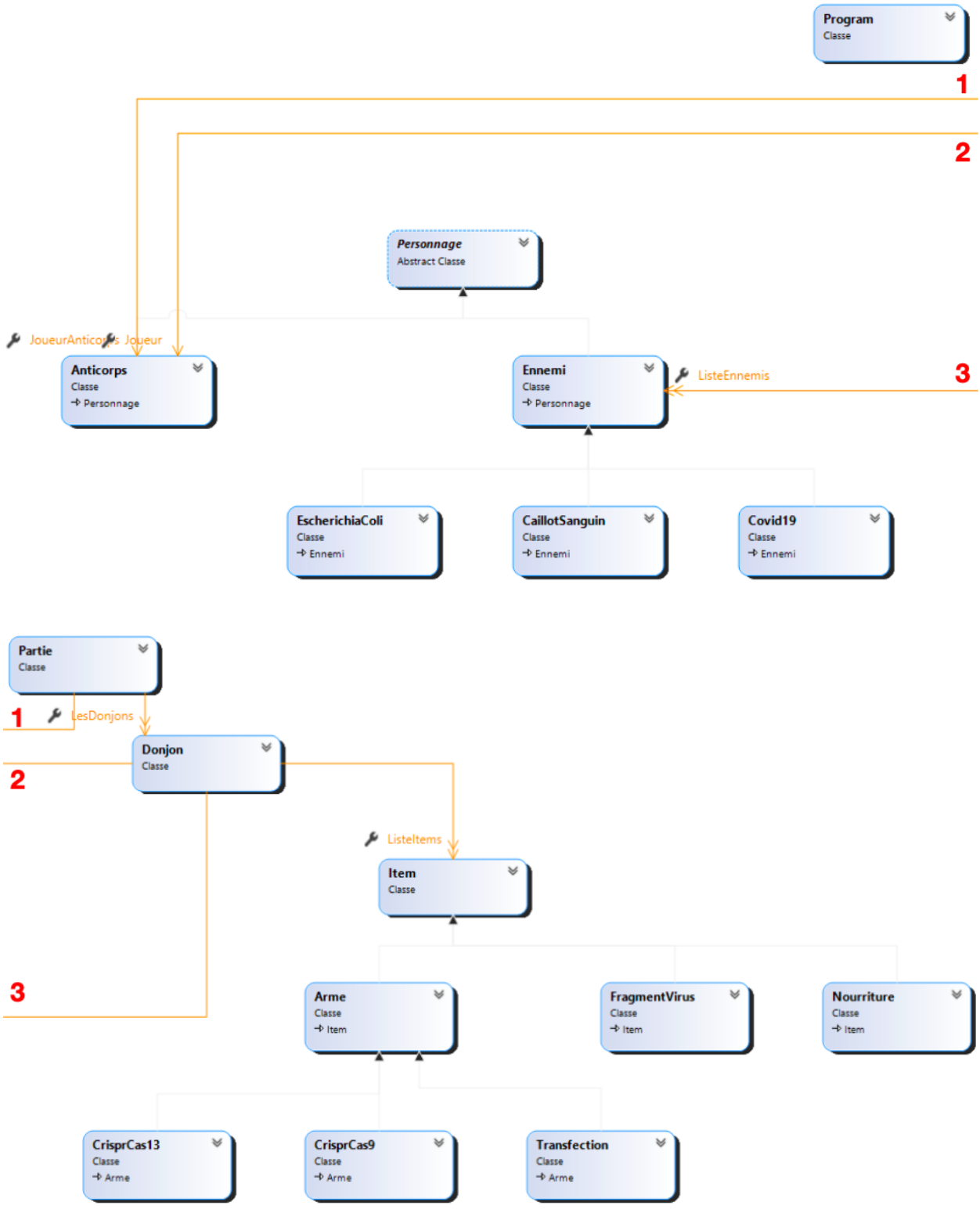

Figure 4: Diagramme des classes