Université de Bordeaux

Collège Sciences et Technologie

Projet de programmation Python - Virus Killer

Auteurs:

DOURTHE Cyril

JUNG Frédéric

THOUVENIN Arthur

Bordeaux
29-11-2017
Compte rendu rédigé en LaTeX

Sommaire

1 Introduction **2**

2 Analyse **2**

3 Conception **3**
3.1 Le terrain de jeu 4
3.2 L'affichage graphique 5

4 Réalisation **6**
4.1 Les différents import 6
4.2 La grille de jeu 6
4.3 Les déplacements 7
4.4 Le déplacement des virus 8
4.5 La génération des parois infranchissables 8
4.6 La génération aléatoire de l'ATP et des virus 9
4.7 La création et la gestion de l'inventaire 9
4.8 La gestion de l'explosion 10
4.9 Les conditions de victoire et de défaite 10
4.10 La gestion des commandes et le menu start 11
4.11 L'aspect visuel 11
4.12 La gestion d'erreur 13
4.13 Sauvegarde et Chargement 13

5 Conclusion **14**

6 Références **15**

1 Introduction

Virus Killer est un jeu développé intégralement sur Python dans le cadre de notre projet. Le joueur incarne un défenseur du système immunitaire qui a pour mission de détruire des virus qui ont infectés l'espace intercellulaire d'un patient. Tout en récupérant de l'énergie sous forme d'Adénosine Triphosphate (ATP) pour vous renforcer, vous devrez détruire les quatre virus du niveau à l'aide de bombes médicamenteuses qui peuvent s'avérer très puissante. Faites les bons choix de déplacements et faites en sorte de ne jamais tomber à cours d'énergie pour venir à bout des trouble-fête.

Par conséquent, le but de ce projet est de réaliser un jeu codé en langage Python, afin de mettre en pratique nos connaissances et nos compétences à coder en langage Python. Le joueur incarne le héros qui se déplace dans une grille de 10 cases sur 10.

2 Analyse

Le plus important pour un jeu vidéo est sa fonctionnalité. Il est donc nécessaire de définir quelles sont les fonctionnalités de base fondamentale au bon fonctionnement du jeu. Tout d'abord, il est obligatoire de définir une zone qui délimitera l'aire de jeu. Cela signifie qu'il faut un objet fixe qui sera à la base de toutes les fonctionnalités qui suivront.

Ensuite, le joueur doit pouvoir se déplacer selon un schéma prédéfini tout en restant dans l'aire de jeu et en respectant les contraintes de déplacement (parois cellulaires, déplacement unidirectionnel...). Cette base s'applique également aux virus qui doivent pouvoir se déplacer. Leur déplacement doit être aléatoire et donc être gérer par l'ordinateur tout en respectant également les contraintes de déplacement.

Le but du jeu étant d'éliminer les virus, le joueur doit pouvoir posséder les outils pour le faire. Il nous faut donc créer les armes qui seront les médicaments afin de détruire les virus. Il faut donc définir ensuite dans quelles conditions les bombes détruiront les virus. Les médicaments fonctionnent de la même manière que des bombes, c'est à dire qu'elles explosent et détruisent les virus.

A partir de ce point, nous obtenons un jeu qui est déjà fonctionnel et il ne reste plus qu'à implanter des fonctionnalités secondaires qui vont améliorer notre jeu, comme les molécules d'énergie sous forme d'ATP, la gestion de la puissance des bombes avant le début du jeu et pendant les tours de jeu,

l'explosion des bombes en fonction de la puissance et les conditions de victoire ou de défaite.

3 Conception

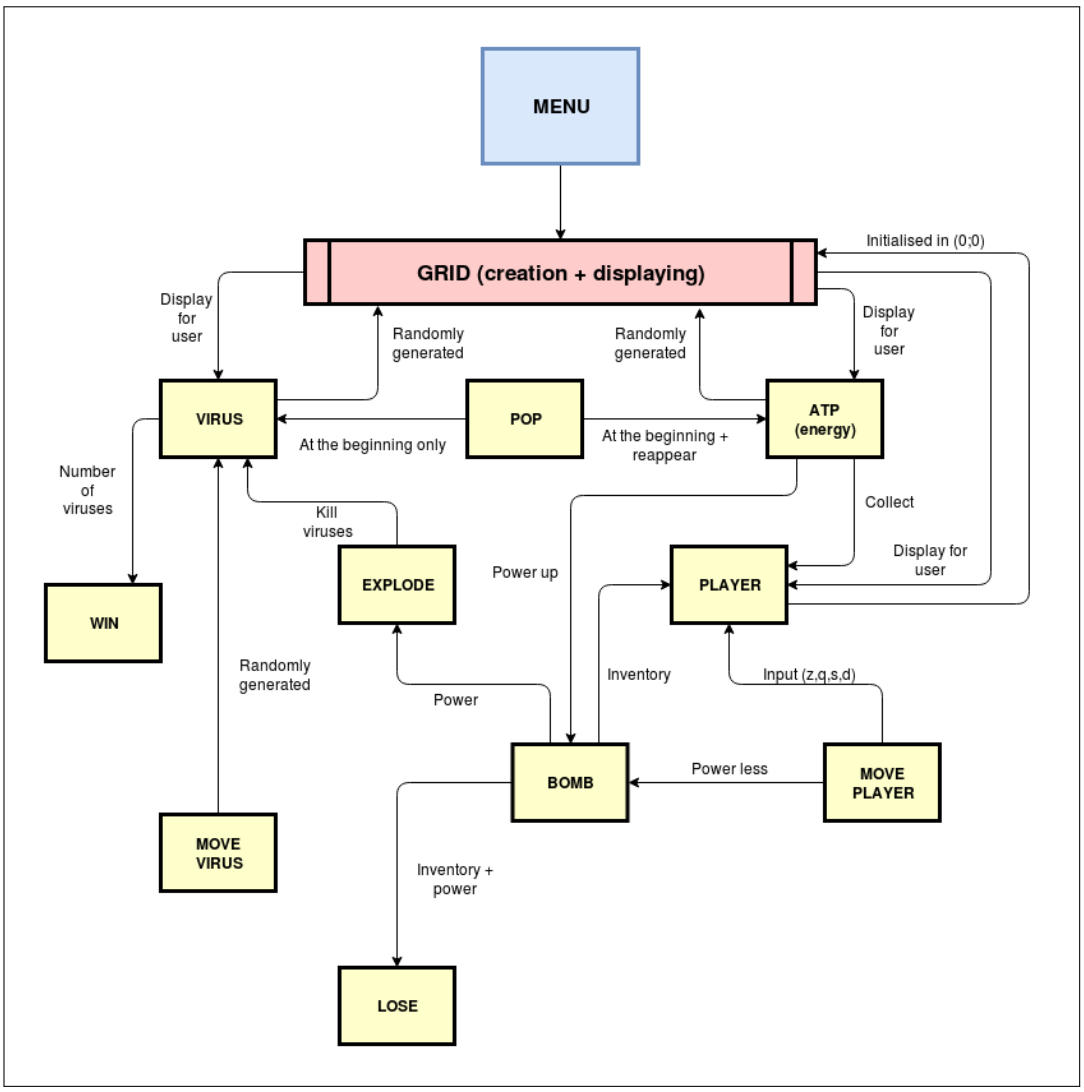

Figure 1: Conception du programme (workflow)

La conception a été pensée avant même de lancer le codage à proprement parler. Un workflow a tout d'abord été établi pour définir le sens que prendrait notre programme afin d'obtenir le code le plus logique possible

(Figure : 1), cela permet d'avoir une vue d'ensemble sur notre futur travail. Par la suite dans cette partie, nous allons développer en détail notre conception.

3.1 Le terrain de jeu

Pour commencer, nous allons créer la zone de jeu. Elle correspondra à une grille de taille 10x10. C'est sur cette grille que nous implanterons toutes les fonctionnalités du jeu (joueur, virus, bombes...), et pour cela il est nécessaire qu'elle soit modulable. Il existe plusieurs moyen que nous avons étudié pour stocker le contenue de chacune des cases de notre grille. Il faudra donc choisir entre une liste, un dictionnaire et une liste à deux dimensions. Ainsi un simple print du contenu de ces moyens de stockage nous permettrait d'afficher la grille.

Le deuxième aspect à concevoir est la génération des parois cellulaires. Elles vont être importantes notamment pour les déplacements du joueur et des virus. En effet, ce sont les parois qui vont arrêter le déplacement du joueur et celui des virus. De plus, les parois devront normalement être infrachissables pour l'explosion des bombes et obligera donc le joueur à réfléchir ses déplacements pour être bien positionné afin de détruire les virus avec les bombes. Concernant les parois, il est également envisagé de créer plusieurs niveaux de difficultés où le nombre de parois augmentera avec le niveau de difficulté. Nous envisageons de faire trois niveaux de difficultés: facile avec 10 parois, normal avec 20 parois et difficile avec 30 parois. Les parois ne seront pas placés au hasard mais selon un schéma que nous avons défini afin d'éviter les complications, notamment en mode difficile, comme le blocage total du joueur par les parois. Enfin, ces parois cellulaires seront infrachissables pour le joueur et pour les virus, et aucun objet (énergie ou bombe) ne pourra se trouver sur ces cases.

Après avoir initialiser l'aire de jeu et les objets immuables (parois),les huit énergies sous forme d'ATP devront être placés aléatoirement sur la grille de façon à ce qu'elles ne soient pas sur la même case que le joueur, les virus et surtout les parois infrachissables. Ces énergies auront pour rôle de redonner de la puissance à deux des quatre bombes du joueur. De plus, dès qu'une énergie est ramassée par le joueur, une autre réapparait sur la grille de manière aléatoire n'importe où sauf sur le joueur, un virus ou une paroi.

Ensuite, notre préoccupation sera tourné vers le joueur. Il faut définir un point d'apparition en début de partie pour le joueur qui sera pour nous la coordonnée (0;0). Le joueur doit se déplacer selon un nombre de cases qu'il

choisit dans une direction et une seule uniquement. Si le joueur rencontre une paroi avant la fin de son déplacement, il est stoppé sur la case avant la paroi. Pour se déplacer, le joueur doit chosir sa direction avec les touches qui seront configurer: "z" pour avancer, "q" pour aller à gauche, "d", pour aller à droite et "s" pour reculer. Après avoir choisi sa direction, le joueur définira de combien de cases il se déplacera entre 1 et 9. Il va falloir implémenter des conditions pour que le joueur ne traverse pas les murs et qu'il ne puisse pas terminer son déplacement sur une case qui contient un virus.

Le joueur dispose d'un inventaire dans lequel se trouve quatre bombes. Les quatre bombes ont chacune une puissance de 8, 6, 4 et 2. La puissance va influer sur le rayon de l'explosion de la bombe : la bombe explose seulement sur la colonne et donc une bombe de puissance 8 va exploser sur un diamètre de 8 (rayon de 4 de chaque côté de la bombe). Si à la fin de son tour le joueur ne pose pas de bombe, toutes les bombes de l'inventaire devront perdre 1 de puissance. Dés que le joueur pose une bombe, une nouvelle bombe avec une puissance aléatoire apparaît dans l'inventaire.

Enfin, en ce qui concerne les virus, nous laisserons l'aléatoire définir le positionnement des quatre virus au début du jeu, sous condition que les virus ne peuvent pas apparaître sur les énergies, les parois ou sur la case du joueur. Les virus se déplaçeront de manière aléatoire également dans une seule direction. Si le virus rencontre une paroi, il s'arrêtera sur la case juste avant la paroi. Le virus se trouvant dans le rayon d'explosion de bombe disparaît de la grille et ne réapparaît pas.

Pour les conditions de victoire, il faut donc que le joueur élimine tous les virus. Si le joueur se retrouve avec toutes ses bombes à 0 de puissance, il perd.

Un petit menu d'acceuil sera intégré au jeu afin de lancer ce dernier.

3.2 L'affichage graphique

Il faut que le joueur puisse comprendre comment jouer, sans affichage visuel, le jeu risque d'être incompréhensible. Dans la console, de simple print, peuvent nous permettre d'afficher la grille. Il faut aussi un moyen de distinguer les différents éléments, à savoir, les membranes, l'ATP, les virus et le joueur.

4 Réalisation

4.1 Les différents import

Lors de la réalisation de notre projet nous avons du recourir à plusieurs import, nous avaons cependant fait attention à n'utiliser que ceux nécessaire et à ne pas en abuser.

- import sys :
  Ce module nous a permis d'établir la sortie du programme dans les cas nécessaire.

- import os :
  Ce module nous a permis de réaliser des "clear" gâce à l'operating system.

- import random :
  Ce module nous a permis de tirer des chiffres aléatoirement dans des intervalles et ainsi parfois juste tirer un entier ou tirer un entier comme indice pour une liste.

- import codecs :
  Ce module nous a permis d'adapter les fonctions de Sauvegarde et Chargement avec Unicode en utf-8.

4.2 La grille de jeu

Nous avons fait le chois d'utiliser une liste à deux dimensionspour tracer la grille. C'est-à-dire que chaque élément de notre liste est une liste. Nous avons donc une première liste qui correspond à nos coordonnées en x et à l'intérieur de celle-ci d'autres listes (le nombre de liste est égale à la longueur de la grille) qui correspondent à nos coordonées en y. On trace ainsi un tableau de taille 10x10 (voir Figure : 2).

Ainsi, on peux facilement placer un élément dans la grille en déterminant ces coordonées x et y et G[x][y] nous renvoie le contenue de la case en position x,y. La position du joueur est définie aux coordonnées x=0 et y=0.

Avant toute chose, la grille est générée et un certain nombre d'éléments y sont ajoutés. Le joueur est placé selon sa position prédfinie ainsi que les murs. Les virus et les ATP apparaissent aléatoirement dans des cases vide.

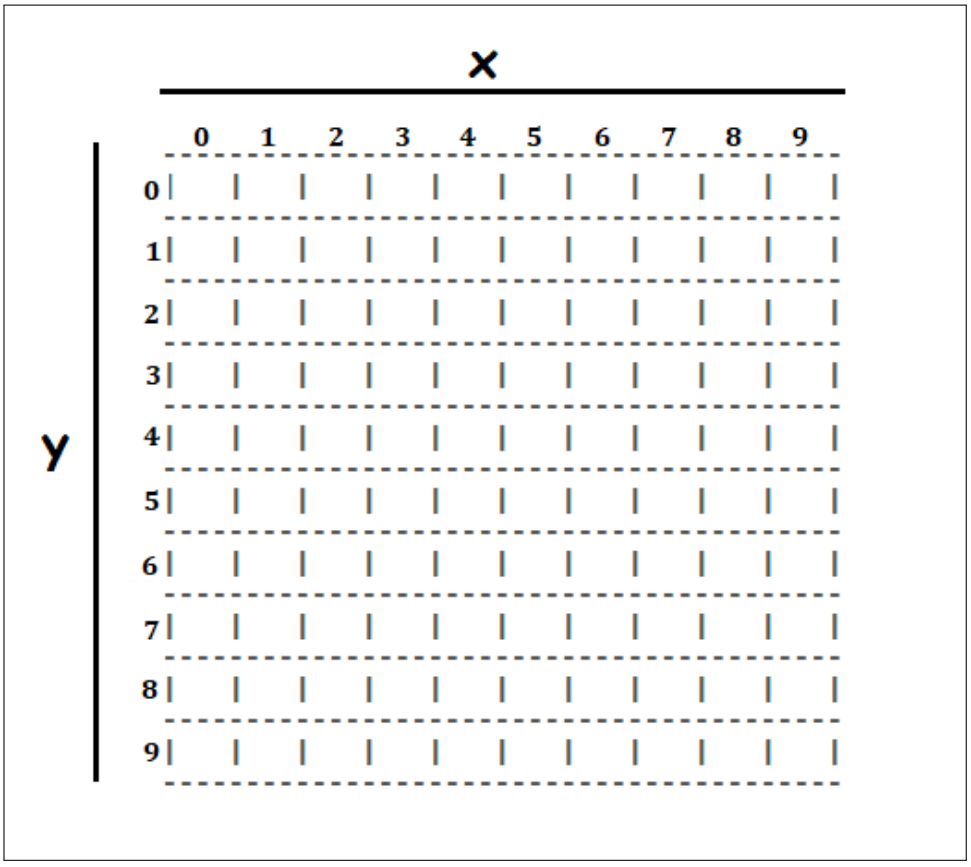

Figure 2: Grille de jeu

4.3 Les déplacements

Nous avons donc opté pour le système de déplacement "z", "q", "s", "d". Nous avons choisi de demander tout d'abord dans quel sens le joueur doit se déplacer et ensuite entrer le nombre de cases qu'il souhaite parcourir. Ces deux paramètres sont pris en compte dans une fonction qui nous permet de faire avancer le personnage de case en case jusqu'à atteindre le nombre de cases qu'il a choisi. La fonction regarde donc le contenu de la case à chaque déplacement et lui indique s'il peut continuer à avancer ou pas (Figure : 3).

Afin que le joueur ne puisse pas se déplacer en dehors de la grille nous avons imbriqué une condition qui dit que la position du joueur ne peut pas être en dehors de la grille. De la même manière une condition permet d'interdire au joueur de traverser un mur.

Par contre le joueur peut passer sur une case contenant un virus à condition et lui interdit de terminer son déplacement en contenant un. Il peut

également passer au dessus d'un ATP, mais celui-ci est consommé au même moment.

A la fin, l'ancienne position est remplacé par une case vide et le joueur prend la nouvelle position correspondant aux nouvelles valeur de x et y.

La focntion qui nous permet le déplacement a été optimisé de sort qu'elle puisse être utilisé selon les 4 directions de déplacement possible.

En résumé, les conditions d'un déplacement sont :

- Si la case suivante est un mur, ou se situe en dehors de la grille, le joueur recule et prend cette position.
- Quand le personnage doit s'arrêter, si la position du joueur correspond à celle d'un virus je reviens en arrière juequ'à trouver une case vide.
- Si le joueur passe sur une moléculé d'ATP, elle disparaît et les médicaments ont leur portée incrémenté de 1 (pour deux médicaments tiré au sort).

4.4 Le déplacement des virus

Les virus se déplacent de la même manière que le personnage. Nous utilisons donc la même fonction qui leur permettent de se déplacer selon les même conditions (Figure : 3), c'est à dire qu'ils ne puissent pas traverser des murs et sortir de la grille. On note quand même quelques différences concernant les règles de déplacement :

- Les virus ne peuvent pas s'arrêter sur une case contenant le joueur.
- Les virus ne peuvent pas ramasser les ATPs.
- La direction et le nombre de case de déplacement sont tirés au hasard.

4.5 La génération des parois infranchissables

Les parois infranchissables n'ont pas été défini de manière alétoire. En effet, il y aurait pu avoir des problèmes avec la génération aléatoire comme le cas où le joueur aurait pu se retrouver enfermer entre les parois extérieures et les murs infrachissables. Les murs auraient aussi bien pu être générer dans une petite zone et rendre inacessible cette zone. De plus, notre objectif est de créer plusieurs niveaux de difficulté qui, plus ils sont élevés, plus ils contiennent de parois, il est donc encore plus difficle de faire en sorte de faire apparaître aléatoirement nos murs.

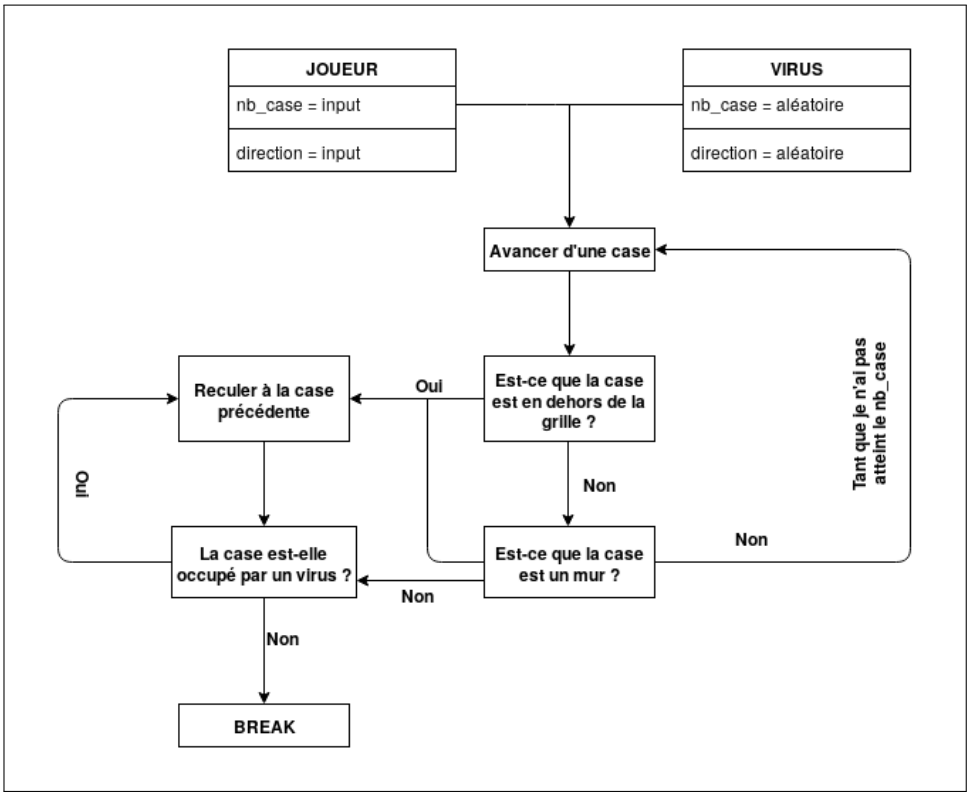

Figure 3: Schématisation des déplacements

4.6 La génération aléatoire de l'ATP et des virus

mljndfblkjsfdnvpkjqenvlkjdqsnvokjdsqnvkojn La génération de l'ATP et des virus se fait de manière aléatoire. Au début, nous avions distingué en deux fonctions la génération de chacune de ces entités. Finalement, une seule fonction regroupe les deux générations et permet d'être paramatrée pour faire apparaître le nombre de virus et d'ATP de notre choix. Les ATP et les virus ne peuvent pas apparaître sur des cases contenant le joueur, les murs, d'autres ATP ou d'autres virus. En ce qui concerne l'ATP, comme dit dans la partie déplacement, dés qu'une molécule d'ATP est ramassée par le joueur, une nouvelle apparaît immédiatement sur la grille en faisant appel à la fonction d'apparaition aléatoire de l'ATP dans la fonction de déplacement.

4.7 La création et la gestion de l'inventaire

Le joueur doit posséder en permanence quatre bombes sur lui. Ces bombes possèdent une certaine puissance qui va de 8 à 2 et qui décrémente à chaque fois que le joueur se déplace sur la grille. Une fonction a donc été

créé permettant de définir un inventaire pour le joueur. Il a été difficile de gérer tous les aspects concernant les bombes comme celui de la perte de puissance ou du fait qu'une bombe avec une puissance aléatoire réapparaît après la pose d'une bombe. Il a donc fallu créer plusieurs fonctions qui gèrent, une, la décrémentation de la puissance après le déplacement, une autre, l'augmentation de la puissance après le passage sur les molécules d'ATP et enfin une fonction générale qui gére l'inventaire de base et qui l'incrémente d'une bombe avec une puissance aléatoire après la psoe d'une des bombes. De même pour pouvoir faire réapparaître une bombe après utilisation, nous avons dpu recourir à une liste par défaut, peut être qu'un dictionnaire aurait été plus approprié.

4.8 La gestion de l'explosion

La gestion de l'explosion se fait via une fonction qui gére non seulement l'explosion et la destruction de virus. La partie délicate a été de penser au fait que les murs extérieurs arrêtent le rayon de l'explosion et que donc l'explosion ne doit pas commencer en haut de la grille et finir en bas et vice-versa. Pour cela cette fonction a été revue de nombreuse fois jusqu'à ce que l'on repère de grandes similarités avec le déplacement des virus et joueur, nous avons donc opté pour une version de la fonction plus orienté à la façon du déplacement notament grâce à des fonctions comme "blast".

4.9 Les conditions de victoire et de défaite

Les deux conditions étaient à la base de notre réflexion séparé en deux fonctions bien distinctes ne gérant qu'un seul aspect: soit la victoire soit la défaite. Pour la condition de défaite, une fonction a été spécialement conçue (fonction lose) et est appelée dans la fonction gérant les commandes. La fonction lose vérifie simplement si le joueur se retrouve avec toutes ses bombes à zéro de puissance. En ce qui concerne la condition de victoire, nous avons également décidé de faire une fonction dédiée à cette condition. Elle fonctionne en vérifiant une liste contenant les virus. Si les virus ont tous été effacés de la liste par les explosions (voir fonction explode), la condition de victoire sera remplie. Nous avons également revues ses fonctions jusqu'à la fin du projet en effet de nouvelles fonctions ont sans cesse était ajoutée modifiant parfois l'intégralité de nos conditions de victoire ou défaite.

4.10 La gestion des commandes et le menu start

Les fonctions permettant de faire tourner notre programme sont la plupart rassemblées dans une fonction commandes permettant d'exécuter dans le bon ordre toutes les fonctions. "Commandes" est appelée dans le MAIN, elle permet d'initialiser la partie mais aussi que son déroulement soit fait sans accroc. Un menu start (géré par la fonction éponyme) permet de lancer la partie après avoir sélectionner le niveau de difficulté. Les niveaux de difficultés sont gérés par une fonction à part ainsi que la présentation du jeu (speech). Tout au long de la réalisation cette partie centrale de notre jeux n'a cessé d'évoluer, avec l'ajout de fonction ou leur suppression, même jusqu'aux derniers jours où la fonction permettant un nouveau jeu fut implémentée.

4.11 L'aspect visuel

Le jeu a été pensé pour tourner sur la version 2.7.12 de python et lancé depuis un terminal sous linux. Une autre version pourrait entraîner des problèmes de compatibilités. Il est conseillé au joueur de lancer une partie avec un terminal avec fond noir en plein écran.

vspace0.25cm

Nous avons utilisé Unicode un standard informatique qui converti du texte avec des symboles pour que l'affichage soit plus esthétique et parlant, pour les couleurs cela fait appel à des fonctions bash au travers de python. Les différents éléments sont reconnus par leur forme et visuel. En effet, par convention, les ennemis (les virus) apparaissent rouge et les bonus (ATP) apparaissent en vert. Ces étapes ont nécessité une grande partie de recherche de plus des générateur d'art ASCII ont été utilisé pour certains motifs. Il a fallu repenser notre script qui était basé sur de simples string afin de déclaré ces strings en variable globales et ainsi pouvoir les modifier selon nos envies sans pour autant affecter nos fonctions.

L'interface du jeu est simple est complète, elle comprte trois zone (Figure 4). On distingue ainsi la zone de jeu (1), l'inventaire (2) où le joueur peut apercevoir la durabilité de ses bombes, le menu (3) qui indique au joueur les actions qu'il peut faire.

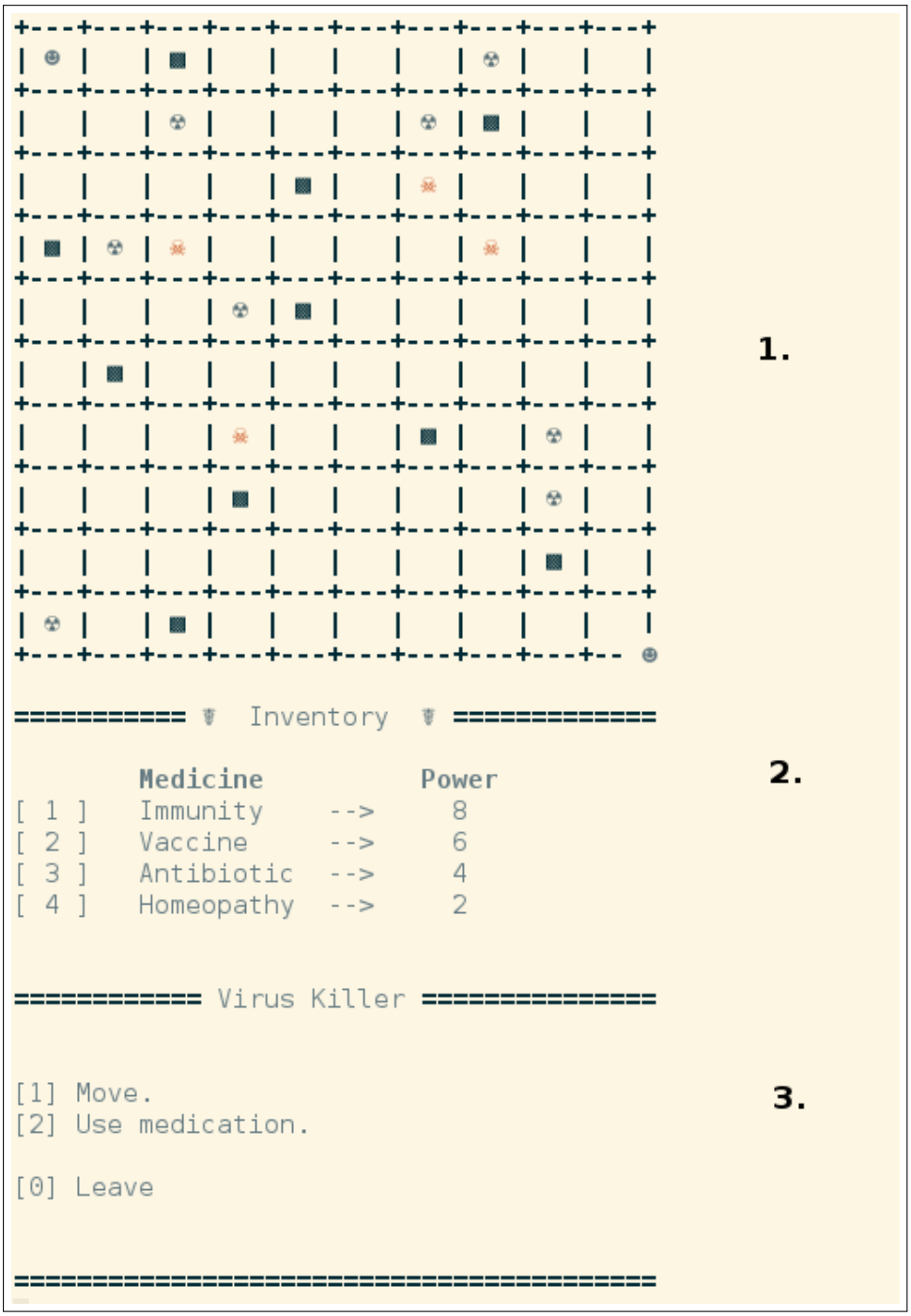

Figure 4: Interface du jeu

4.12 La gestion d'erreur

A chaque fois qu'un input est demandé au joueur, le programme passe par la fonction error qui permet au programme de ne pas s'arrêter en sortant une erreur. En effet, cette fonction est composé d'un try à la saisie de l'input et gère deux exceptions possible qui pourrait survenir à la saisie. Ces deux exceptions sont :

- Le programme s'arrête si l'input est une chaîne de caractère (NameError).
- Le programme s'arrête si l'input est un caractère spécial (SyntaxError).

4.13 Sauvegarde et Chargement

Ces fonctionnalités n'étaient pas demandées dans les consignes cependant nous avons pris leurs réalisations comme un défi personnel. Cependant le temps nécessaire pour simplifier le code de ces fonctions n'a pas été suffisant ce quiles rends quelque peu redondantes. Un vrai défi fut de jongler entre les caractères ASCII et UTF-8, ce fut un réel problème de réalisation, ce qui a nécessité une grande part de documentation. La partie est sauvegardé à chaque fois que le joueur quitte le jeux, car elle est implémentée dans notre fonction exit. Pour le chargement il se fait dans les menus selon le choix du joueur. Les données sont sauvegardées en UTF-8 dans un fichier save, puis elles sont chargées par après au format approprié. En revanche, par manque de temps, nous n'avons pas pu factoriser le code correspondant à la fonction save, afin de la simplifier au maximum.

5 Conclusion

Notre programme est globalement réussi et nous satisfait dans sa généralité. En effet, notre jeu est fonctionnel comme nous le voulions et nous avons réussi à le rendre plus esthétique et facile d'utilisation. Cependant, il pourrait être encore optimisé que ce soit au niveau du code (qui pourrait être plus court et mieux écrit) ou encore au niveau de la fonctionnalité. Les déplacements sont un peu lourd car le joueur doit réaliser des manipulations comme entrer la direction et le nombre de cases en étant obligé de toujours confirmer son choix en appuyant sur entrée, ce qui rend la manipulation plus lente et pouvant amener à des erreurs de saisie. Notre système de déplacement reste tout de même intuitif mais pourrait également rencontrer des problèmes dans le cas de joueur qui joueraient sur des claviers qwerty. Auquel cas les déplacements seront beaucoup moins intuitifs. En ce qui concerne le reste, les seuls difficultés ont été de simplifier au maximum notre code afin que n'importe quel autre développeur puisse le prendre et l'améliorer sans changer toutes les fonctions ou toutes les variables. Une règle était également sombre à propos de l'utilisation des médicaments, nous avons préféré laissé le choix au joueur d'utiliser le nombre de bombe qu'il désire.

Nous avons essayé autant que possible de faire en sorte que le programme soit le plus modulable possible de tel sorte que si l'on veut reprendre le code par la suite, pour améliorer le jeu par exemple, on puisse le faire assez facilement sans avoir à modifier tout le code. Ainsi la taille de la grille, l'aspect visuel des éléments de la grille, ou le nombre d'éléments sont des éléments parmis d'autres qui peuvent être changé sans répercussion sur les fonctions. De la même manière, tout au long du projet nous avons essayé de faire en sorte que si l'on décide à la dernière minute de changer la structure de la grille (c'est à dire plutôt passer par une simple liste ou un dictionnaire), que l'on puisse le faire sans trop de soucis. Par manque de temps nous n'avons pas pu tester les deux autres structures de grille et les comparer. La grille à deux dimensions nous est tout de suite venu à l'esprit, et fut très vite adopté pour la simplicité avec laquelle nous avons pu définir des coordonnées. Mais il est vrai qu'il aurait été intéressant de se pencher sur les autres cas.

Plus de temps nous aurait permis d'apprendre à utiliser une interface graphique. Tkinter est un module de Python qui permet de créer une interface graphique et notre projet s'est achevé avec la recherche documentaire autour de celui-ci. En effet, le fait de passer par le terminal pour jouer au jeu est assez contraignant et très limité.

6 Références

- Openclassroom :
  https://openclassrooms.com/courses/apprenez-a-programmer-en-python
- Un générateur d'art ASCII :
  http://www.ascii-fr.com/Generateur-de-texte.html
- Site officiel de Python
  https://www.python.org/
- Table des caractères unix :
  https://fr.wikipedia.org/wiki/Table_des_caract%C3%A8res_Unicode/U1F300
- API Python 2.7 :
  https://docs.python.org/2/c-api/index.html
- Stack Overflow :
  https://stackoverflow.com/
- Python Tutor :
  http://www.pythontutor.com/visualize.html#mode=edit