Ligony
Charles

TS

Lycée Alfred Kastler
année 2015/2016

Rapport de projet d'ISN

« Zombies. »

Projet présenté par :

- LIGONY Charles
- HULOT Stéphane
- SCHARFF Raphaël

Dans le cadre de la spécialité ISN, un groupe composé de trois personnes, moi compris, a codé un jeu qui s'appelle « Zombie ! » et qui a entièrement été codé sous python en utilisant la librairie Tkinter intégré à celle-ci.

SOMAIRE

I. Le but visé

Pourquoi ce projet :
Nous avons observé que durant les années précédente, beaucoup de jeu fait, étaient des reprises améliorées de jeu existant et populaire. Nous voulions donc faire un jeu qui n'avait été, à notre connaissance, pas fait, et qui nous demandais une réflexion sur tout les principes et dynamique du jeux.

Les principes du jeu :
Le jeu se passe en tour par tour. Pour finir un niveau, un des deux joueurs doit d'échapper d'un labyrinthe de 21x21 cases (murs compris) en se rendant jusqu'à un drapeau.
Un nombre de zombies variant de 3 à 5 se trouvent dans les niveau et si l'un d'eux attrape un joueur, la partie est perdu pour les deux joueur. Le jeu est constitué de 5 niveaux de difficulté croissante, d'un didacticiel (donc 6 en tout) ainsi qu'un éditeur de niveau.

Déroulement d'une manche :
Lorsque c'est aux tour des joueurs, deux déplacements leurs sont attribué pour les deux. C'est-à-dire que si un joueur à besoin de se déplacer deux fois, il pourras le faire en un tour mais l'autre joueur ne se déplacera pas du tour.
Lorsque c'est aux tour des zombies (intelligence artificielle), ils bougent alors chacun d'une case vers le joueur le plus proche. Si un joueur ou un zombie sont sur un bouton, des portes s'ouvrent.

Contrôles :

Les joueurs ne peuvent que se déplacer. Pour cela ils doivent utiliser **Z, Q, S, D** pour le **joueur 1** et **les touches directionnelles** pour le **joueur 2**. Dans l'éditeur de niveau, **p**our sélectionner les différentes parties que constitue un niveau, il faut appuyer sur 1, 2, 3, 4 ou 5 afin de sélectionner le type de bloc et faire un clic gauche de la souris pour poser un bloc et clic droit de la souris pour le retirer.

→ ANEXE 1

II. La structure du logiciel

Le programme est codé en Python et utilise Tkinter. Il est découpé en quatre documents différents. C'est donc du langage dit objet.

Chaque document a une particularité différente qui sont :

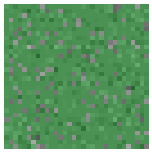
Non du document	Ce qu'il permet	Nombre de ligne de code
Principal.py	- Permet de lancer le jeu - gère le menu du jeu	Environ 50
Jeu.py	- gère les différents niveaux - importe les texture des niveaux - gère le jeu en temps réel dont: - Les collisions avec les joueurs, les murs, les porte et les zombie - L'ouverture des portes - La mort des joueurs - La victoire des joueurs - Les déplacements des joueurs et des zombies	Environ 550
Joueur.py	- teste les collisions entre le terrain et le joueur - déplace le joueur	Environ 30
Zombie.py	- s'occupe du déplacement des zombie pour piéger le joueur en : - suivant le plus proche - résolvant des cas (1 bloc autour de lui, 2 blocs autour de lui...)	Environ 120

Mais chaque document dans le code qui est séparé à besoin des autre document pour fonctionner :

→ **ANEXE 2**

III. Problème rencontrés

Notre premier problème a été l'affichage des images qui composent ce que l'on voit dans le jeu. C'est à dire :

    …

Le problème était en faite que les images s'affichaient bien mais elles se supprimaient directement après. Le problème a été corrigé dès la première séance. Il s'agissait seulement du manque d'un 'self' dans le code.

→ **ANEXE 3**

De plus, nous avons eu deux gros problèmes avec l'interface.
Tout d'abord, lorsque nous changions de niveau via le menu, celui-ci disparaissais. En fait celui-ci effaçais le niveau d'avant et affichais le nouveau en dessous c'est-à-dire dans une zone non visible. La solution a été d'effacer ce qui se trouver dans le canevas pour créer ceux du niveau que l'on veux charger.

→ **ANEXE 4**

Le second problème rencontré avec l'interface a été un problème avec les boutons. En effet lorsque l'on mourais, en recommençant le niveau, les portes ne s'affichaient plus. Nous avons du chercher la solution sur internet car on devait utiliser 'deepcopy' qui n'avais pas été vu en cours.

→ **ANEXE 5**

Enfin, le plus gros problème que nous avons eu dans le code a été l'IA, c'est-à-dire le déplacement des zombies. En effet, au début nous n'arrivions pas à les faire bouger. Le code a été vite corrigé et les zombies ont été capable de se déplacer de haut en bas sur le terrain se qui étais totalement inutile pour jouer. De ce fait nous avions opté pour un déplacement totalement aléatoire du zombie qui avais le problèmes de rester bloquer et de rendre le jeu beaucoup trop facile. Nous avions ensuite réfléchis à une solution en fonction de la distance entre le zombie et le joueur le plus proche pour enfin décider de faire un code beaucoup plus complexe en répondant au cas par cas. Tout les cas on donc été pensé et même le cas ou le zombie est pris entre 4 blocs car celui ci avais un bug, il montais vers le haut.

→ **ANEXE 6**

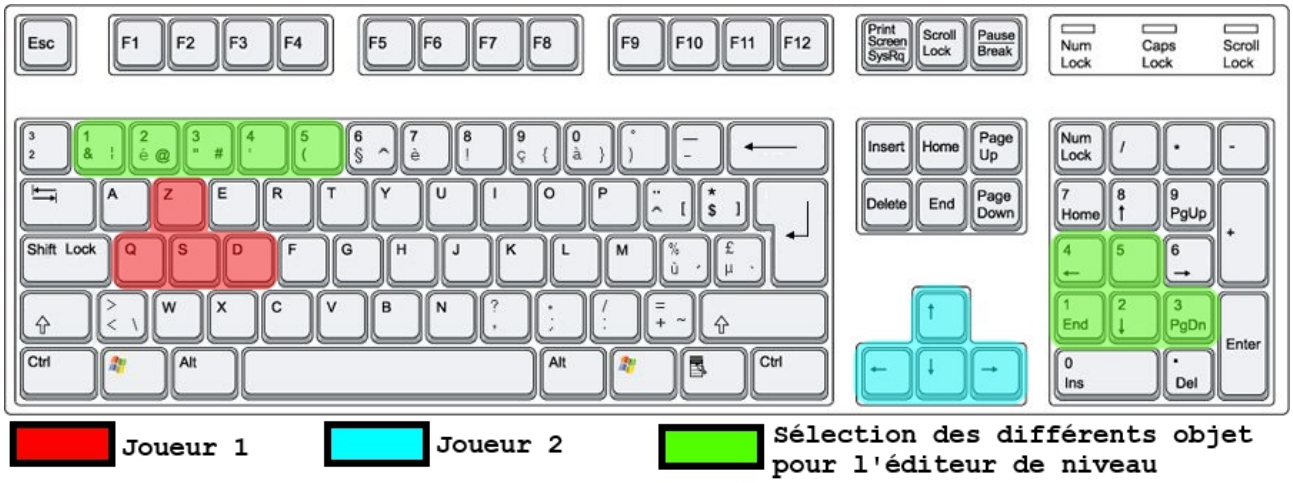
IV. Conclusion

Dans le cadre de notre projet, nous devions concevoir un logiciel dans un thème choisi et dont le but était de faire un travail de groupe affins de créer un logiciel valable.
Nous avons commencer par faire une version papier des idées que nous avions du logiciel avant de nous lancer. Ceci fût une erreur de notre part car nous n'avions pas organiser totalement et séparer toute les tâches. Je pense qu'un organigramme aurait été plus judicieux.

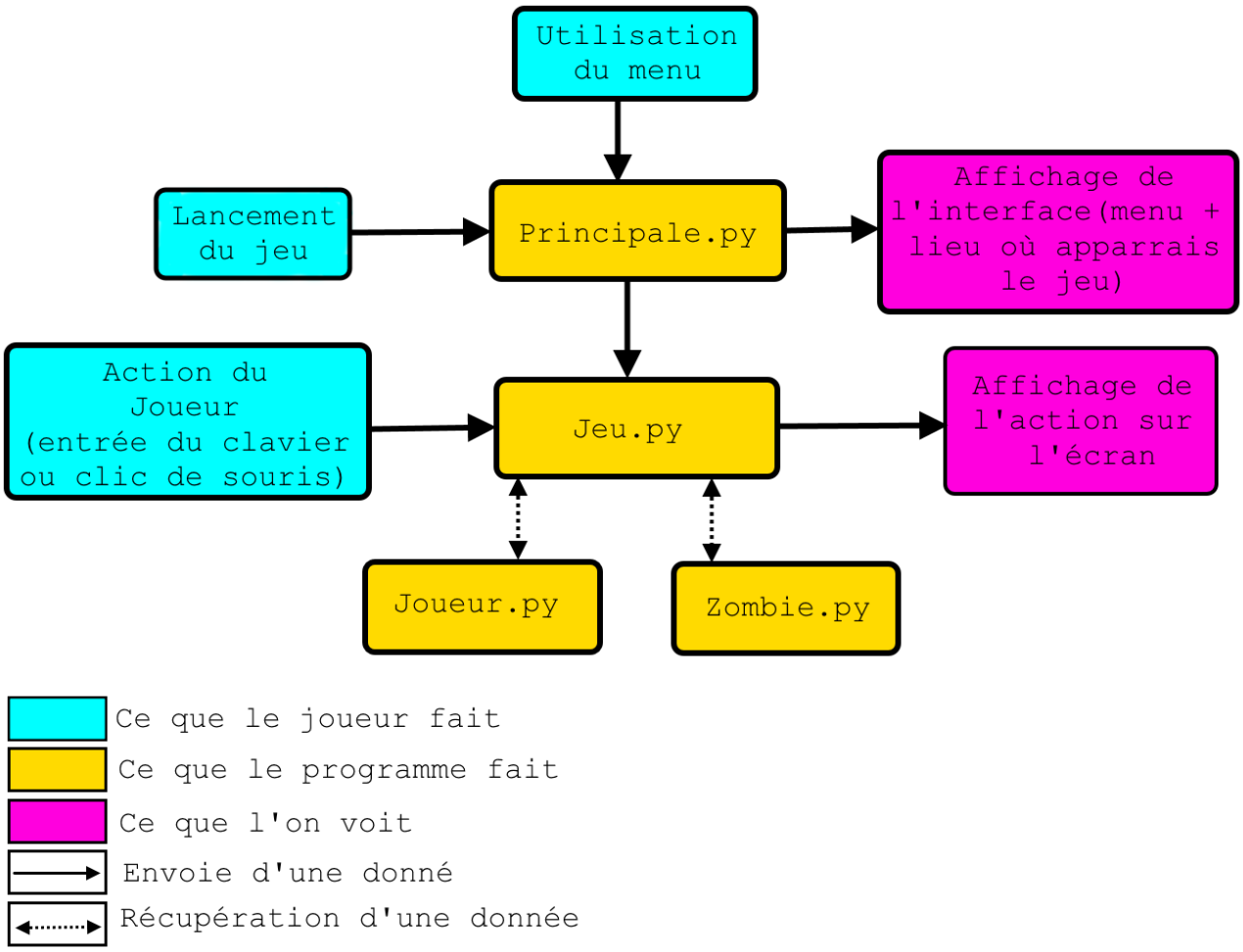
La gestion des tâches fût après assez bien gérée et le langage tourné objet fût une bonne idée afin que tout le monde puisse travailler sur une partie du code différente bien qu'il y avait quelque problème, c'est-à-dire qu'une partie du logiciel (ex. zombie.py) pouvais interagir avec une autre partie du logiciel (ex. principale.py) mais l'inverse ne fonctionnais pas.

D'un point de vue plus humain, le travail en groupe est une bonne chose. En effet cela permet à la fois de travailler en équipe mais aussi de pouvoir avoir d'autre point de vu sur une même idée et donc d'interagir avec autrui.

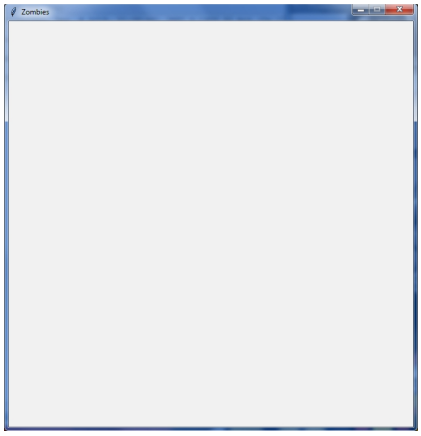
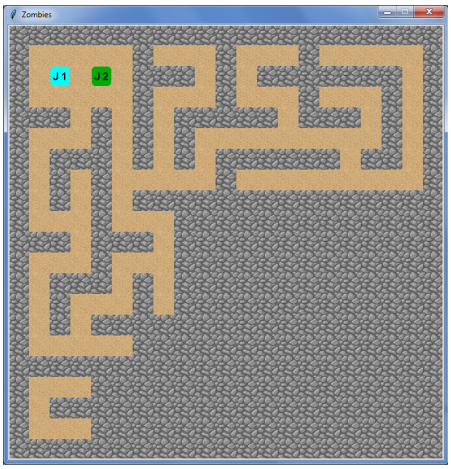
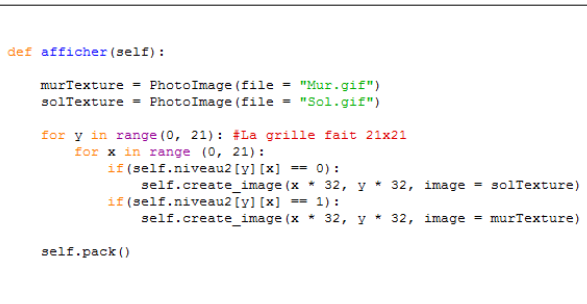
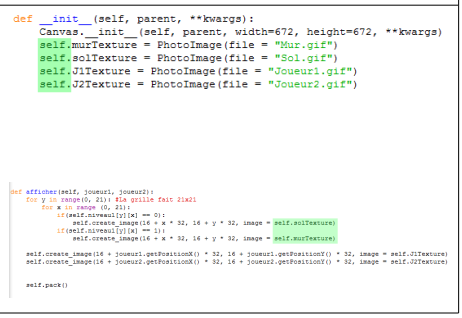
V. Anexes



ANEXE 1 : Image représentant les différentes touches active du clavier dans le jeu



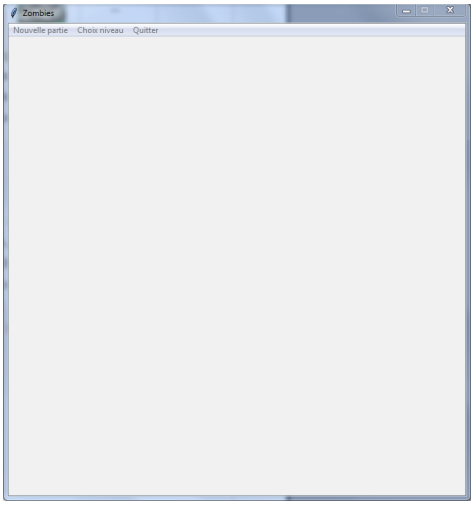
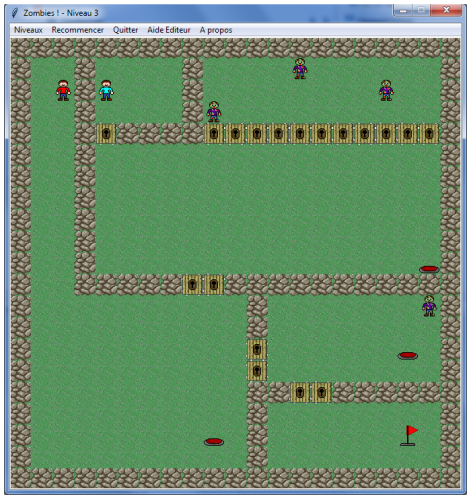
ANEXE 2 : Organigramme expliquant le fonctionnement du logiciel

	Avant	Après
Vision du logiciel		
code		

```
def afficher(self):

    murTexture = PhotoImage(file = "Mur.gif")
    solTexture = PhotoImage(file = "Sol.gif")

    for y in range(0, 21): #La grille fait 21x21
        for x in range (0, 21):
            if(self.niveau2[y][x] == 0):
                self.create_image(x * 32, y * 32, image = solTexture)
            if(self.niveau2[y][x] == 1):
                self.create_image(x * 32, y * 32, image = murTexture)

    self.pack()
```

```
def __init__(self, parent, **kwargs):
    Canvas.__init__(self, parent, width=672, height=672, **kwargs)
    self.murTexture = PhotoImage(file = "Mur.gif")
    self.solTexture = PhotoImage(file = "Sol.gif")
    self.J1Texture = PhotoImage(file = "Joueur1.gif")
    self.J2Texture = PhotoImage(file = "Joueur2.gif")
```

ANEXE 3 : Problème et résolution de l'affichage

<table>
<tr><th></th><th>Avant</th><th>Après</th></tr>
<tr><td>Vision du logiciel</td><td> 
</td><td> 
</td></tr>
<tr><td>Code</td><td>

```
global niveauCharger
niveauCharger = False

def verification():
    global niveauCharger, jeu
    if niveauCharger == True:
        jeu.delete(ALL)
        niveauCharger = False
    else:
        niveauCharger = True

def niveau0():
    global jeu
    verification()
    jeu=Jeu(fenetre, 0)

def niveau1(): #Lance le jeu 1
    global jeu
    verification()
    jeu=Jeu(fenetre, 1)

def quitter():
    fenetre.destroy()
```

Provient de principale.py

```
def afficher(self):
    for y in range(0, 21): #La grille fait 21x21
        for x in range (0, 21):
            if(self.niveauActuel[y][x] == 0):
                self.create_image(16 + x * 32, 16 + y * 32, image = self.solTexture)
            if(self.niveauActuel[y][x] == 1):
                self.create_image(16 + x * 32, 16 + y * 32, image = self.murTexture)
            if(self.niveauActuel[y][x] == 2):
                self.zombies.append(Zombie(x, y, len(self.zombies), self))
                self.create_image(16 + x * 32, 16 + y * 32, image = self.solTexture)
            if(self.niveauActuel[y][x] == 3):
                self.create_image(16 + x * 32, 16 + y * 32, image = self.murTexture)
            if(self.niveauActuel[y][x] == 4):
                self.porte = self.create_image(16 + x * 32, 16 + y * 32, image = self.murTexture)
                self.porteX = x
                self.porteY = y
```

Provient de Jeu.py</td><td>

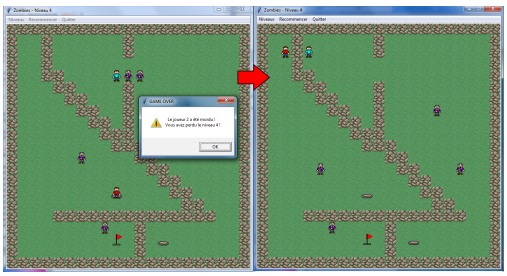
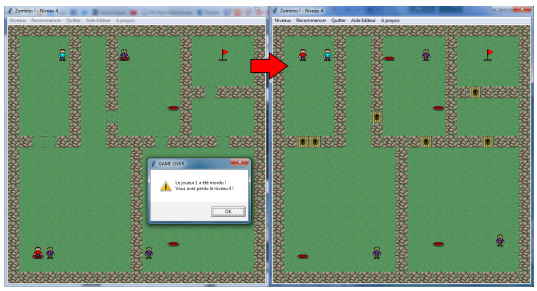
```
def chargerNiveau(niveau): #Charge le niveau
    jeu.afficher(niveau)

def recommencer(): #Recomence le niveau
    jeu.recommencer()
```

Provient de Jeu.py

```
def afficher(self, niveau): #Affichage et mise en place d'un niveau
    self.delete("all")
    self.niveau = niveau
    #Chargement du niveau
```

Provient de principale.py</td></tr>
</table>

ANEXE 4 : Problème et résolution du chargement des niveau

<table>
<tr><th></th><th>Avant</th><th>Après</th></tr>
<tr><td>Vision du logiciel</td><td>  </td><td>  </td></tr>
<tr><td>Code</td><td>

```
def afficher(self, niveau):
    self.delete("all")
    self.niveau = niveau
    #Chargement du niveau
    if niveau == -1:
        self.niveauActuel = self.debug
    if niveau == 0:
        self.niveauActuel = self.tuto
    if niveau == 1:
        self.niveauActuel = self.niveau1
    if niveau == 2:
        self.niveauActuel = self.niveau2
    if niveau == 3:
        self.niveauActuel = self.niveau3
    if niveau == 4:
        self.niveauActuel = self.niveau4
    if niveau == 5:
        self.niveauActuel = self.niveau5
```

</td><td>

```
def afficher(self, niveau): #Affichage et mise en place d'un niveau
    self.delete("all")
    self.niveau = niveau
    #Chargement du niveau
    if niveau == -1:
        self.niveauActuel = deepcopy(self.custom)
    if niveau == 0:
        self.niveauActuel = deepcopy(self.tuto)
    if niveau == 1:
        self.niveauActuel = deepcopy(self.niveau1)
    if niveau == 2:
        self.niveauActuel = deepcopy(self.niveau2)
    if niveau == 3:
        self.niveauActuel = deepcopy(self.niveau3)
    if niveau == 4:
        self.niveauActuel = deepcopy(self.niveau4)
    if niveau == 5:
        self.niveauActuel = deepcopy(self.niveau5)
    if niveau == 6:
        self.niveauActuel = deepcopy(self.fin)
```

</td></tr>
</table>

ANEXE 4 : Problème et résolution de l'affichage des portes

```
def deplacer(self, carte, joueur1, joueur2):
    distJoueur1 = sqrt((joueur1.getPositionX()+self.x)**2 +(joueur1.getPositionX()+self.x)**2)
    distJoueur2 = sqrt((joueur2.getPositionX()+self.x)**2 +(joueur2.getPositionX()+self.x)**2)

    direction = 0
    dx = 0
    dy = 0

    if(distJoueur1 < distJoueur2):
        dx = joueur1.getPositionX() - self.x
        dy = joueur1.getPositionY() - self.y
    else:
        dx = joueur2.getPositionX() - self.x
        dy = joueur2.getPositionY() - self.y

    if(dx <= dy and dx != 0):
        if(dx > 0):
            direction = 1
        else :
            direction = 3

    else:
        if (dy > 0):
            direction = 0
        else :
            direction = 2

    if(direction == 0 and carte.niveau1[self.y + 1][self.x] != 1):
        self.y += 1
        carte.afficherDeplacementZombie(0, 1, self.numero)
        deplace = True
    if(direction == 1 and carte.niveau1[self.y][self.x + 1] != 1):
        self.x += 1
        carte.afficherDeplacementZombie(1, 0, self.numero)
        deplace = True
    if(direction == 2 and carte.niveau1[self.y - 1][self.x] != 1):
        self.y -= 1
        carte.afficherDeplacementZombie(0, -1, self.numero)
        deplace = True
    if(direction == 3 and carte.niveau1[self.y][self.x - 1] != 1):
        self.x -= 1
        carte.afficherDeplacementZombie(-1, 0, self.numero)
        deplace = True
```

```
def deplacer(self, joueur1, joueur2):
    distJoueur1 = sqrt((joueur1.getPositionX()-self.x)**2 +(joueur1.getPositionY()-self.y)**2)
    distJoueur2 = sqrt((joueur2.getPositionX()-self.x)**2 +(joueur2.getPositionY()-self.y)**2)

    if distJoueur1 == 1:
        self.jeu.gameOver(1)

    if distJoueur2 == 1:
        self.jeu.gameOver(2)

    direction = ""
    dx = 0 #Position X relative
    dy = 0 #Position Y relative

    if(distJoueur1 < distJoueur2): #Si le joueur 1 est le plus proche
        dx = joueur1.getPositionX() - self.x
        dy = joueur1.getPositionY() - self.y
    else: #Si le joueur 2 est le plus proche
        dx = joueur2.getPositionX() - self.x
        dy = joueur2.getPositionY() - self.y


    if(abs(dx) > abs(dy)): #Si la distance x a parcrourir est plus grande que la distance y a parcourir
        if(dx > 0):
            direction = "E" #Droite
        elif(dx < 0):
            direction = "W" #Gauche
    else: #Si la distance y a parcrourir est plus grande que la distance x a parcourir
        if(dy > 0):
            direction = "N" #Haut
        elif(dy < 0):
            direction = "S" #Bas

    if(direction == "N" and self.jeu.testColision(self.x, self.y + 1)):
        self.y += 1
        self.jeu.afficherDeplacementZombie(0, 1, self.numero)
    if(direction == "E" and self.jeu.testColision(self.x + 1, self.y)):
        self.x += 1
        self.jeu.afficherDeplacementZombie(1, 0, self.numero)
    if(direction == "S" and self.jeu.testColision(self.x, self.y - 1)):
        self.y -= 1
        self.jeu.afficherDeplacementZombie(0, -1, self.numero)
    if(direction == "W" and self.jeu.testColision(self.x - 1, self.y)):
        self.x -= 1
        self.jeu.afficherDeplacementZombie(-1, 0, self.numero)
```

```
def deplacer(self, joueur1, joueur2): #Déplace le zombie en fonction de la direction du joueur sinon d'un chemin possible
    self.distanceJoueur(joueur1,joueur2) #Calcule les distances entre le zombie et chaque joueur

    if self.distJoueur1 == 1:
        self.jeu.joueurMort = 1
        return

    if self.distJoueur2 == 1:
        self.jeu.joueurMort = 2
        return

    move = False
    direction = self.suivreJoueur() #Suis le joueur en fonction de sa potion

    if(direction == "N" and not self.jeu.testCollision(self.x, self.y + 1)): #Déplacement Haut
        self.y += 1
        self.jeu.afficherDeplacementZombie(0, 1, self.numero)
        move = True

    if(direction == "S" and not self.jeu.testCollision(self.x, self.y - 1)): #Déplacement Bas
        self.y -= 1
        self.jeu.afficherDeplacementZombie(0, -1, self.numero)
        move = True

    if(direction == "E" and not self.jeu.testCollision(self.x + 1, self.y)): #Déplacement Droite
        self.x += 1
        self.jeu.afficherDeplacementZombie(1, 0, self.numero)
        move = True

    if(direction == "W" and not self.jeu.testCollision(self.x - 1, self.y)): #Déplacement Gauche
        self.x -= 1
        self.jeu.afficherDeplacementZombie(-1, 0, self.numero)
        move = True

    if(move == False): #Si il n'y a aucun déplacement (obstacle rencontré), le zombie analyse la situtation
        chemin = self.cheminPossible()
        if(chemin == "N"): #Deplacement Haut
            self.y += 1
            self.jeu.afficherDeplacementZombie(0, 1, self.numero)

        if(chemin == "S"): #Deplacement Bas
            self.y -= 1
            self.jeu.afficherDeplacementZombie(0, -1, self.numero)

        if(chemin == "W"): #Deplacement Gauche
            self.x -= 1
            self.jeu.afficherDeplacementZombie(-1, 0, self.numero)

def distanceJoueur(self,joueur1,joueur2): #Calcule la distance entre le joueur le plus proche et un zombie
    self.distJoueur1 = sqrt((joueur1.getPositionX()-self.x)**2 +(joueur1.getPositionY()-self.y)**2)#Th de Pythagore
    self.distJoueur2 = sqrt((joueur2.getPositionX()-self.x)**2 +(joueur2.getPositionY()-self.y)**2)


    self.dx = 0 #Position X relative
    self.dy = 0 #Position Y relative

    if(self.distJoueur1 < self.distJoueur2): #Si le joueur 1 est le plus proche
        self.dx = joueur1.getPositionX() - self.x
        self.dy = joueur1.getPositionY() - self.y
    else: #Si le joueur 2 est le plus proche
        self.dx = joueur2.getPositionX() - self.x
        self.dy = joueur2.getPositionY() - self.y

def suivreJoueur(self): #Retourne la direction du joueur le plus proche
    if(abs(self.dx) > abs(self.dy)): #Si la distance x à parcrourir est plus grande que la distance y à parcourir
        if(self.dx > 0):
            return "E" #Droite
        elif(self.dx < 0):
            return "W" #Gauche
    else: #Si la distance y à parcrourir est plus grande que la distance x à parcourir
        if(self.dy > 0):
            return "N" #Haut
        elif(self.dy < 0):
            return "S" #Bas

def cheminPossible(self): #Retourne une direction possible en fonction de la situation
    #Cas 4 blocs
    if(self.jeu.testCollision(self.x,self.y + 1) and self.jeu.testCollision(self.x - 1,self.y) and self.jeu.testCollision(self.x + 1,sel
       return ""

    #Cas 3 blocs
    #Si il y a qqch Nord + Ouest + Est, prendre Sud
    if(self.jeu.testCollision(self.x,self.y + 1) and self.jeu.testCollision(self.x - 1,self.y) and self.jeu.testCollision(self.x + 1,sel
        return "S"
    #Si il y a qqch Sud + Ouest + Est, prendre Nord
    if(self.jeu.testCollision(self.x,self.y - 1) and self.jeu.testCollision(self.x - 1,self.y) and self.jeu.testCollision(self.x + 1,sel
        return "N"
    #Si il y a qqch Nord + Ouest + Sud, prendre Est
    if(self.jeu.testCollision(self.x,self.y + 1) and self.jeu.testCollision(self.x - 1,self.y) and self.jeu.testCollision(self.x,self.y
                """
def cheminPossible(self): #Retourne une direction possible en fonction de la situation
    #Cas 4 blocs
    if(self.jeu.testCollision(self.x,self.y + 1) and self.jeu.testCollision(self.x - 1,self.y) and self.jeu.testCollision(self.x + 1,se
       return ""

    #Cas 3 blocs
    #Si il y a qqch Nord + Ouest + Est, prendre Sud
    if(self.jeu.testCollision(self.x,self.y + 1) and self.jeu.testCollision(self.x - 1,self.y) and self.jeu.testCollision(self.x + 1,se
        return "S"
    #Si il y a qqch Sud + Ouest + Est, prendre Nord
    if(self.jeu.testCollision(self.x,self.y - 1) and self.jeu.testCollision(self.x - 1,self.y) and self.jeu.testCollision(self.x + 1,se
        return "N"
    #Si il y a qqch Nord + Ouest + Sud, prendre Est
    if(self.jeu.testCollision(self.x,self.y + 1) and self.jeu.testCollision(self.x - 1,self.y) and self.jeu.testCollision(self.x,self.y
        return "E"
    #Si il y a qqch Nord + Est + Sud, prendre Ouest
    if(self.jeu.testCollision(self.x,self.y + 1) and self.jeu.testCollision(self.x + 1,self.y) and self.jeu.testCollision(self.x,self.y
        return "W"

    #Cas 2 blocs
    #Si il y a qqch Nord + Ouest prendre Est
    if(self.jeu.testCollision(self.x,self.y + 1) and self.jeu.testCollision(self.x - 1,self.y)):
        return "E"
    #Si il y a qqch Nord + Est prendre Ouest
    if(self.jeu.testCollision(self.x,self.y + 1) and self.jeu.testCollision(self.x + 1,self.y)):
        return "W"
    #Si il y a qqch Sud + Ouest prendre Est
    if(self.jeu.testCollision(self.x,self.y - 1) and self.jeu.testCollision(self.x - 1,self.y)):
        return "E"
    #Si il y a qqch Sud + Est prendre Ouest
    if(self.jeu.testCollision(self.x,self.y - 1) and self.jeu.testCollision(self.x + 1,self.y)):
        return "W"

    #Cas 1 bloc
    #Si il y a qqch Nord/Sud prendre Est ou Ouest (en fonction de la direction du joueur)
    if(self.jeu.testCollision(self.x,self.y + 1) or self.jeu.testCollision(self.x,self.y - 1)):
        if(self.dx < 0):
```

ANEXE 6 : évolution du code Zombie.py