Promo 2018

CPO – Zombie Shooter

Projet Tower Defense

Destinataire : Marie-Rose Goncalves

Nicolas Breton et Cécile Tran
EPF Sceaux – Groupe A
Promo 2018

Table des matières

Remarque : Les textes surlignés en jaune sont des parties de code ajoutées ou modifiées depuis le dernier rapport. Lorsque le nom de la classe est surligné, cela veut dire que l'ensemble de la classe est nouvelle.

1. Introduction

Le projet de CPO a pour objectif de concevoir un logiciel comportant une interface graphique programmée en Java sur le logiciel *Netbeans*. Le sujet que nous avons choisi est « Zombie Shooter » ; il s'agit d'un jeu de stratégie dans lequel le joueur doit placer des tours dans le but de défendre un point d'arrivée de différents ennemis apparaissant aléatoirement dans une zone de la carte.

Ce rapport décrira dans un premier temps le programme de manière générale en précisant les règles du jeu choisies. Dans un second temps, les modifications principales du programme seront présentées. Finalement, les jeux d'essai ainsi que le listing du code seront expliqués.

2. Description générale du programme

Les règles définissant notre projet sont les suivantes :

- Le jeu se présente sous la forme d'un plateau composé d'un chemin où seuls les zombies seront présents. Le joueur a la possibilité d'acheter et de placer des tours aux abords du chemin de manière à empêcher les zombies d'arriver au bout de celui-ci. Le parcours des zombies se fait de la gauche vers la droite.

- Le jeu est composé de 3 niveaux de difficulté qui se différencient par la forme et la longueur du chemin que parcourent les zombies.

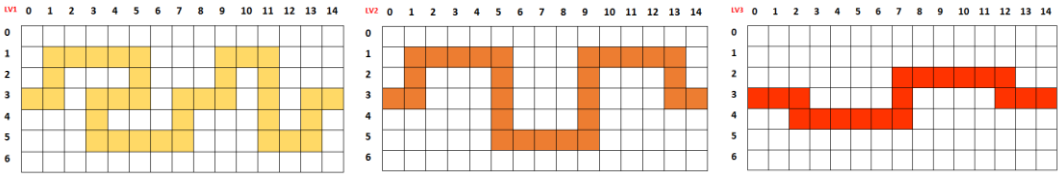


Figure 1 : Schémas des 3 chemins du plus facile au plus difficile

- Le jeu propose 2 types de tours : le premier est composé de 3 tours dites « offensives » qui infligent des dégâts aux zombies ; quant au second, il s'agit d'une tour ralentissante qui réduit temporairement la vitesse des zombies.

- Le joueur peut effectuer plusieurs actions au cours d'une partie : **Acheter tour** qui autorise le joueur à placer une tour en échange d'un paiement, **Amélioration tour** qui permet de renforcer les tours préexistantes sur le terrain, **Supprimer tour** qui propose au joueur de retirer des tours si nécessaire, **Sauvegarder** qui donne la possibilité d'enregistrer les données du joueur et du terrain pour pouvoir reprendre la partie plus tard en sachant qu'il n'est possible de sauvegarder qu'entre 2 vagues.

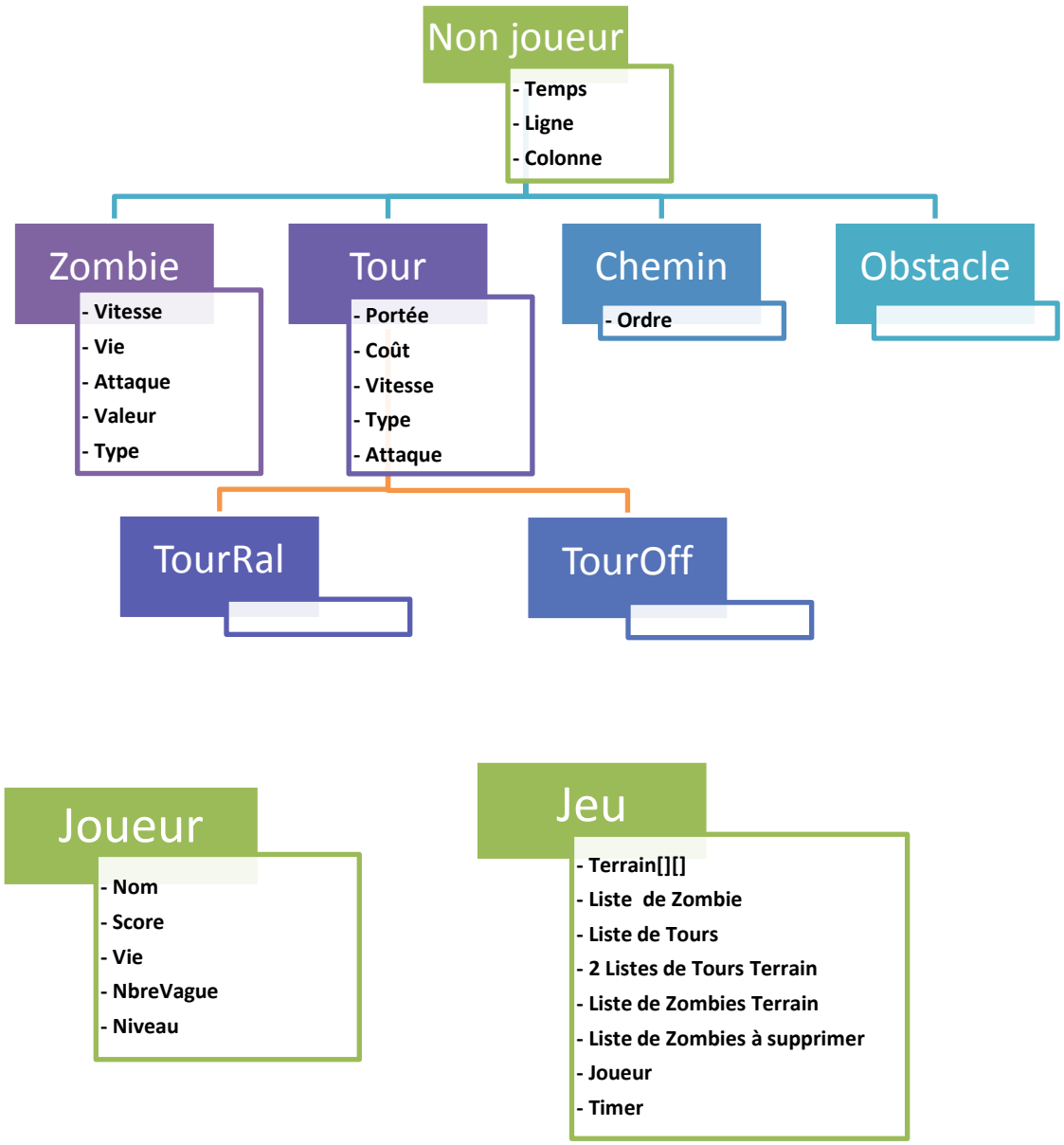
- Le joueur a le temps qu'il souhaite pour placer et améliorer ses tours dans la mesure où c'est lui-même qui lance la vague d'ennemis en appuyant sur le bouton éponyme. A noter que la constitution des vagues se fait aléatoirement dans la quantité et dans la composition. De plus, à la dernière vague, seul un boss apparaît et sa mort entraîne la fin de la partie.

- Le joueur voit son score incrémenté d'un nombre définit selon le zombie tué. Si un zombie atteint le point d'arrivée, la vie du joueur est réduite d'une valeur associée au type du zombie. Ainsi, si le joueur n'a plus de point de vie, la partie est terminée et le score est sauvegardé si celui-ci fait partie des 10 meilleurs.

- Le jeu propose un tableau des scores qui répertorie l'ensemble des 10 meilleurs scores réaliser depuis le début du jeu.

3. Modifications des unités de traitement

La hiérarchie définitive de notre programme est la suivante :



Remarque : Nous avons aussi utilisé une classe appelée JoueurComparator qui a pour rôle de trier les joueurs par ordre décroissant en redéfinissant la méthode Compare de l'interface Comparator<>.

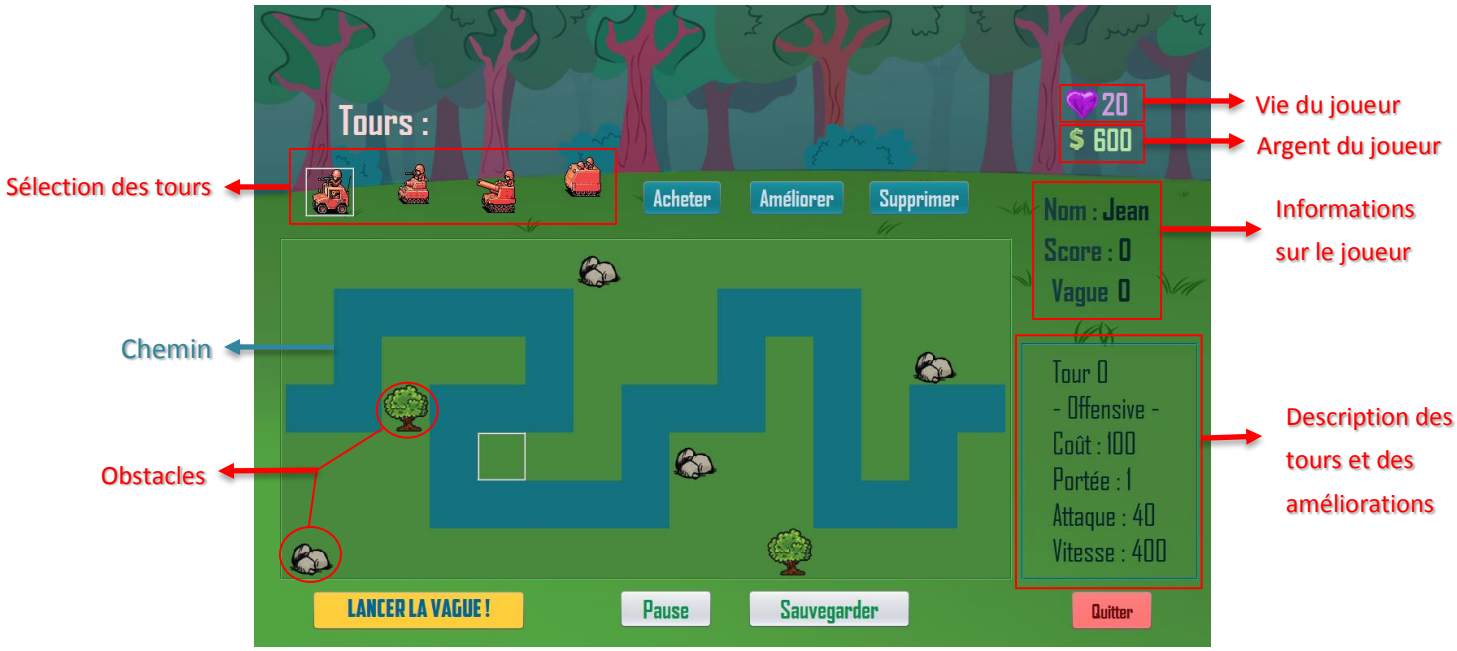
4. Jeu d'essais

4.1. Rôle des différentes fiches

✓ **Fiche Accueil : La fiche accueil est la première fiche qui apparait lors de l'exécution du programme, elle permet d'accéder et d'initialiser les fiches des scores et de jeu.**

✓ **Fiche Scores : Elle affiche le tableau des 10 meilleurs scores triés par ordre décroissant.**

✓ **Fiche Jeu : Elle est la fiche principale où se déroule le jeu.**



Rôle des boutons :

- **Acheter :** Lorsqu'une tour et une case du terrain sont sélectionnées, ce bouton permet d'acheter et de placer la tour choisie sur le terrain si c'est possible : une tour ne peut être acheté si le joueur n'a pas assez d'argent et ne peut être placé sur le chemin ou sur un obstacle.
- **Améliorer :** Lorsqu'une tour est sélectionnée sur le terrain, il est possible de l'améliorer si le joueur a assez d'argent et si une évolution de la tour existe.
- **Supprimer :** Supprime la tour sélectionnée sur le terrain.
- **Lancer la Vague :** Permet de lancer une vague de zombie lorsque le joueur est prêt.
- **Pause :** Met le jeu en pause : stop l'activité sur le terrain puis permet de la reprendre.
- **Sauvegarder :** Permet de sauvegarder les données du joueur et le terrain seulement entre les vagues d'ennemies.

4.2. Données joueurs

Dans cette partie, nous allons tester les différentes évolutions des données liées aux joueurs. Ces données sont utilisées lors de la partie du joueur elle-même, mais aussi lors de la sauvegarde et du chargement et enfin lors de l'établissement du tableau des scores.

4.2.1. Attributs du joueurs

Le premier attribut qui est initialisé est le nom. En effet, en ouvrant la fiche Accueil, le bouton Nouvelle Partie est disponible, mais si le joueur clique dessus alors que le nom n'est pas saisi, le

logiciel lui rappelle que le nom est obligatoire pour commencer une nouvelle partie ou pour lancer le chargement de l'ancienne partie.

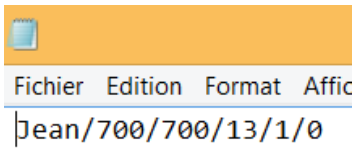
Ensuite, lorsque le joueur a choisi un nom et appuyé sur le bouton Nouvelle Partie, 3 choix s'offrent à lui : Facile, Moyen, Difficile. Ce choix est aussi un attribut du joueur utilisé principalement par la suite pour la sauvegarde et le chargement.

A partir du moment où le joueur a appuyé sur le bouton Jouer, la fiche Jeu s'ouvre. A ce moment précis, plusieurs attributs s'initialisent « automatiquement » : l'argent, la vie du joueur, le score et le nombre de vagues franchies. Ces différents attributs sont affichés au joueur sur la fiche Jeu comme il est montré dans la partie précédente intitulée : *Rôle des différentes fiches*.

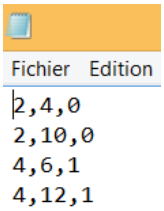
4.2.2. Sauvegarde et chargement

La sauvegarde et le chargement sont 2 composants essentiels de ce programme. Ils permettent de sauvegarder la progression entre 2 vagues, de quitter le jeu et de reprendre la partie lors de lancement du programme grâce à la réécriture du nom choisi dans la partie précédente.

Dans un premier temps, nous allons regarder comment la sauvegarde fonctionne. Lorsque le joueur clique sur ce bouton entre deux vagues, 2 méthodes sont appelées pour sauvegarder la position et les types des tours placées par le joueur et pour sauvegarder les attributs énoncés précédemment du joueur. Les sauvegardes du terrain et des données du joueur se font sur 2 fichiers textes différents.



Les caractéristiques du joueur sont alors enregistrées dans le fichier « joueur.txt » dans l'ordre suivant : Nom/Score/Argent/Vie/Vague/Niveau

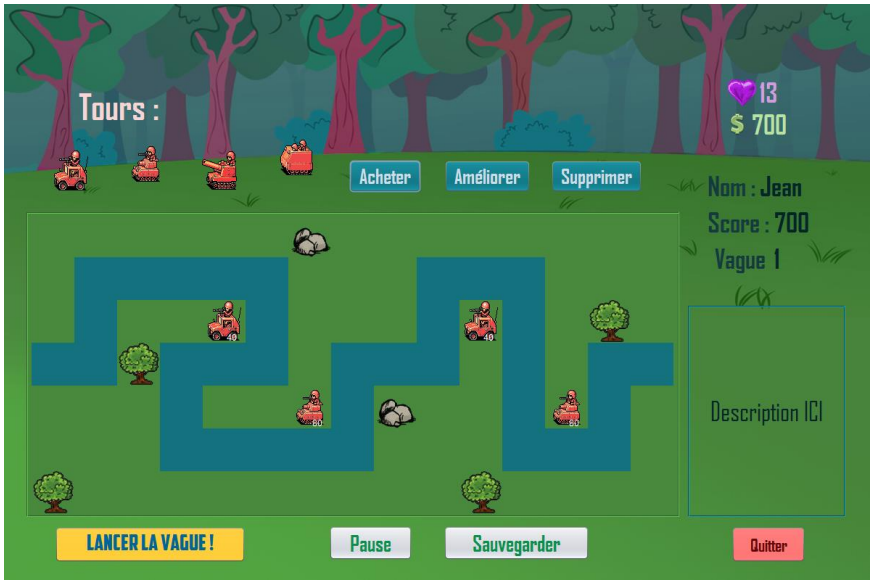


La sauvegarde du terrain est faite grâce au niveau sauvegardé dans le fichier joueur qui permet de récupérer le chemin et les obstacles du terrain selon la difficulté précédemment choisie. Chaque tour du terrain est ensuite sauvegardé ligne par ligne dans le fichier « Terrain.txt » avec les caractéristiques suivantes : Type, Ligne, Colonne

Maintenant, regardons le fonctionnement du chargement du jeu. Lorsque le joueur clique immédiatement après le lancement du programme sur le bouton chargement, un message s'affiche au-dessus de la zone de saisie du nom. Ce message indique quel est le nom à saisir avant de cliquer sur le bouton Chargement pour que la partie soit chargée. Ainsi, quand le nom saisi est identique au

nom de la sauvegarde du joueur, une méthode de la classe Jeu lit le fichier de sauvegarde des attributs du joueur et lance la partie comme elle a été sauvegardée.

Après le chargement, on retrouve bien les caractéristiques du joueur et les tours placées durant la dernière partie.



Remarque : Les images des obstacles sont générées aléatoirement à chaque lancement de la fiche de jeu.

4.2.3. Gestion des scores

La gestion des scores, comme dans tout jeu qui se respecte, est l'objectif ultime pour le joueur. En effet, être le premier du tableau des scores est ce que nous pourrions appeler la « carotte » qui donne le sentiment au joueur qu'il doit jouer pour être le meilleur. Ainsi, c'est une

composante essentielle de notre jeu. La gestion des scores est réalisée à plusieurs moments du jeu pour aboutir au classement tant attendu.

Tout d'abord, il faut noter que le score au début de la partie dans notre programme commence à 0. Ensuite, le score est incrémenté d'une valeur définie pour chaque type de zombie à chaque fois que l'un d'entre eux est tué. Ce score ne peut être diminué dans notre jeu ; en effet, nous aurions pu décider de diminuer le score lorsqu'un zombie atteignait la zone d'arrivée mais seule la vie diminue.

Ensuite, lorsque le joueur finit une partie, pour observer son classement dans la fiche Score, il doit fermer le programme et relancer ce dernier. Ensuite, il clique sur le bouton Score et l'ensemble des 10 meilleurs scores est affiché du plus grand au plus petit avec à chaque fois le nom du joueur qui a réalisé la performance. Le tri des scores se fait grâce à une classe JoueurComparator qui définit une relation d'ordre dans les joueurs à partir de l'attribut score.

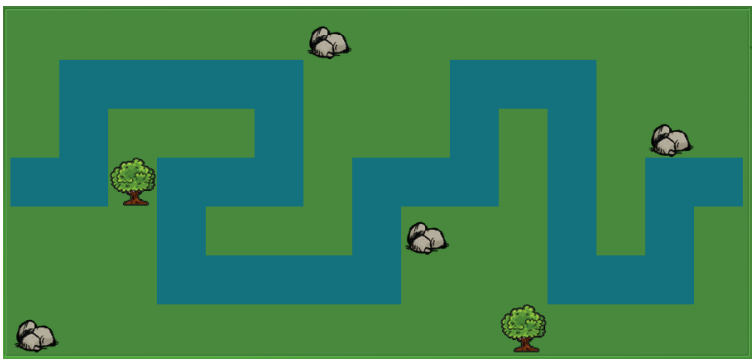
4.3. Zombies

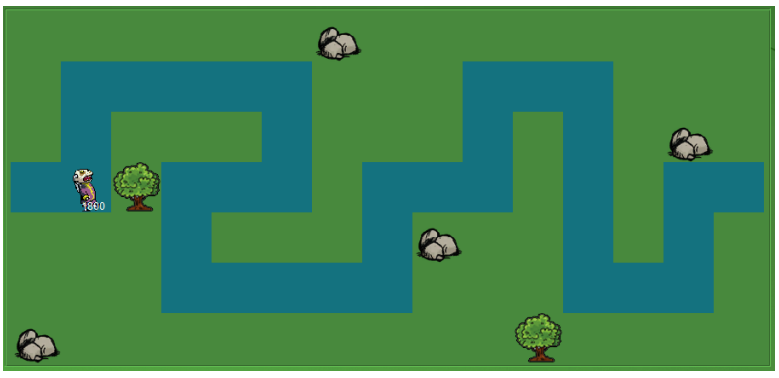
Dans notre jeu, les ennemis ne sont autres que des zombies. Ces derniers sont créés aléatoirement dans le nombre et dans le type à chaque vague et se déplacent avec des vitesses différentes selon le type. A noter que la dernière vague voit l'apparition d'un boss.

4.3.1. Création de zombie et lancement de vague

La création du zombie se fait au moment du lancement de la vague et pendant toute la durée de celle-ci. Le type du zombie est déterminé aléatoirement et celui-ci définit la position du zombie dans la *Linked List* des zombies possibles dans le jeu. Lorsque le zombie est choisi, celui-ci est crée et placé dans la case de départ du chemin. Tout se processus se fait automatiquement dès que le joueur clique sur le bouton Lancer Vague.

Le lancement de vague est en réalité un ensemble de création de zombies. En effet, lorsque le joueur appuie sur le bouton Lancer Vague, un nombre est généré aléatoirement entre 2 valeurs liées au numéro de la vague. Ce nombre définira le nombre de zombies à créer lors de la vague. Ensuite, le processus de Création de zombie décrit précédemment se réalise toutes les 2 secondes un nombre de fois égal au nombre aléatoire.

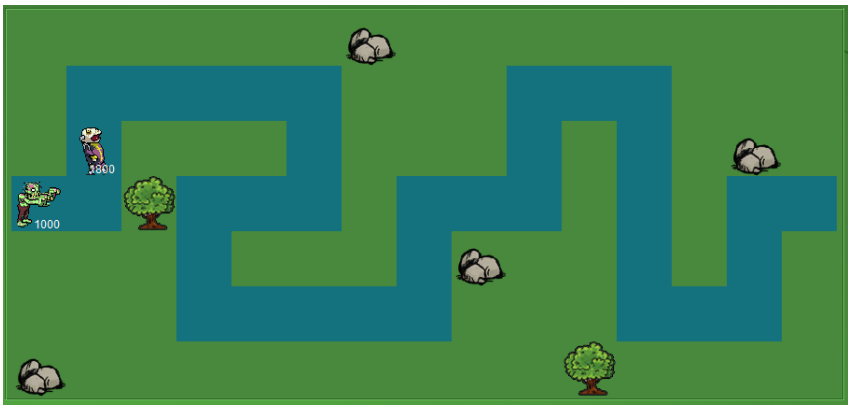




4.3.2. Déplacement des Zombies

Le déplacement des zombies se fait grâce à l'attribut Vitesse de ces derniers. Ce processus se fait entièrement automatiquement sans intervention du joueur. Un attribut Timer dans la classe Jeu

est incrémenté à chaque fois que le Timer de l'interface graphique appelle les méthodes du *Run*. Ensuite, si la somme de la vitesse du Zombie et de l'attribut Temps de la super classe de zombie est égale au Timer, alors le zombie avance d'une case. L'attribut temps est, si et seulement si le Zombie vient d'avancer, mis égal à la valeur de l'attribut Timer.



Pour que le zombie ne recule pas au lieu d'avancer dans le chemin, un système de numéro a été mis en place dans l'ordre croissant. Ainsi, un zombie ne peut aller que dans une case avec le nombre le plus élevé.

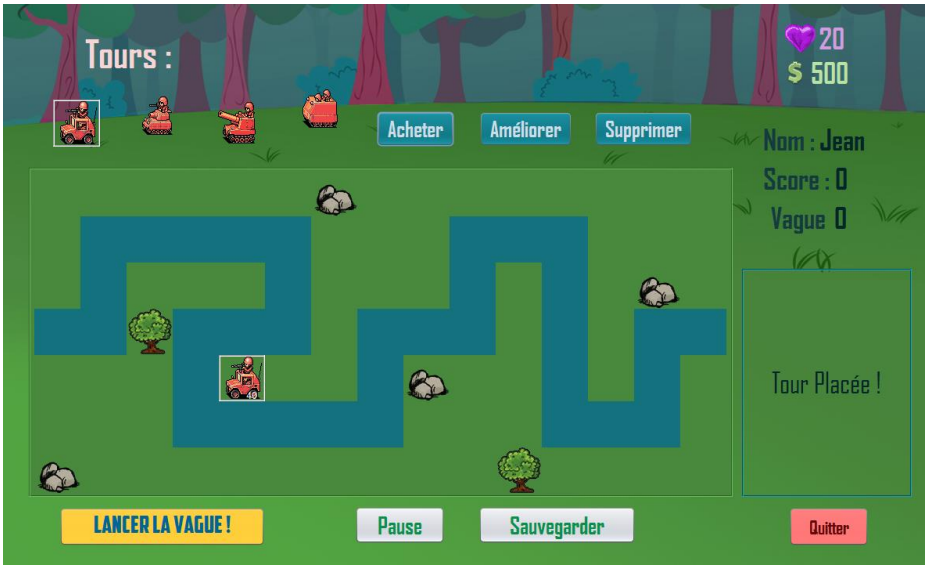
4.4. Tours

Dans notre jeu, nous avons décidé de créer deux types de tours avec des attaques différentes : certaines réduisent la vie des zombies et d'autres les ralenties. Celles-ci peuvent être achetées, améliorées et supprimées, avant et pendant les vagues d'ennemies.

4.4.1. Achat des tours

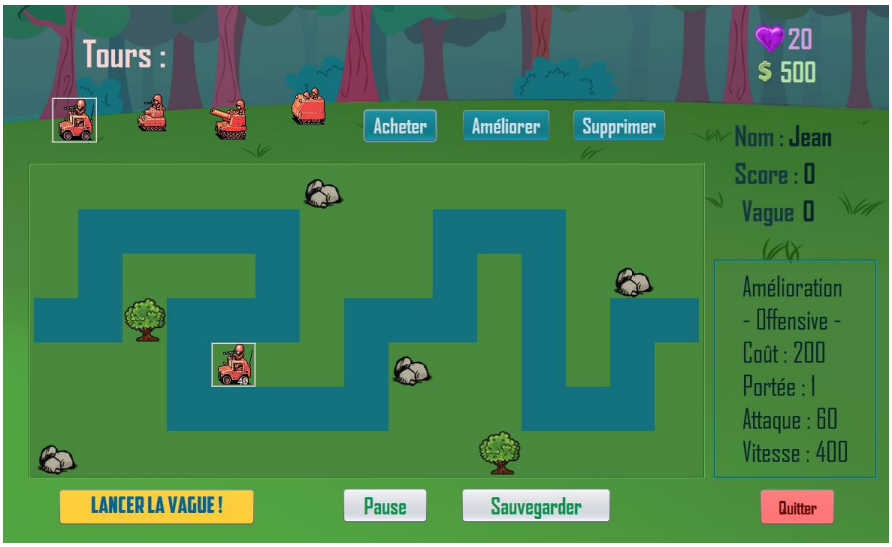
Chaque tour doit d'abord être achetée avant d'être placée sur le terrain. Il faut alors sélectionner la tour à utiliser et celle-ci sera alors placée sur la case sélectionnée sur le terrain une fois que le joueur l'aura acheté s'il le peut.

La tour offensive choisie coûte 100$, l'argent du joueur est donc déduit et la tour s'est bien placée sur la case choisie.

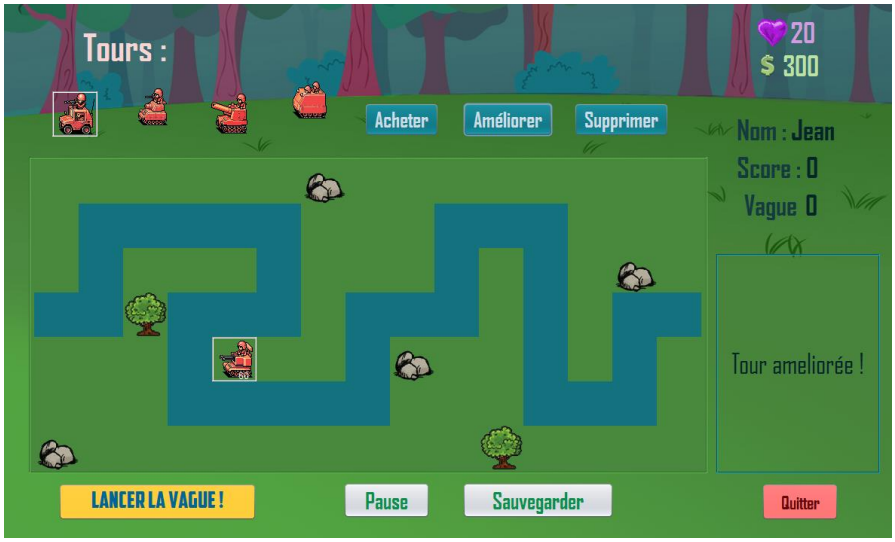


4.4.2. Amélioration des tours

Il est possible au cours de la partie d'améliorer les tours qui sont déjà placées sur le terrain. Les caractéristiques de leurs améliorations sont affichées lorsque l'on sélectionne une tour dont l'amélioration est possible. Ici la tour de type 0 possède une amélioration coûtant 200$ qui lui permettra d'augmenter ses dégâts.



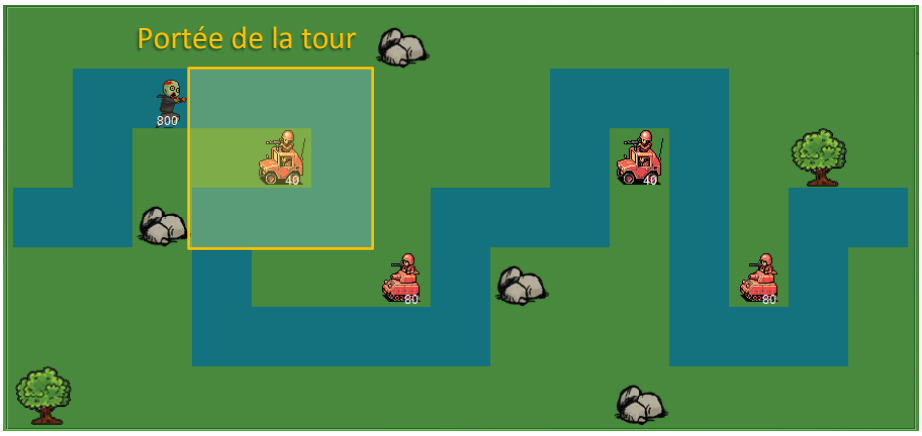
Lors de l'achat de l'amélioration, la nouvelle tour remplacera l'ancienne sur le terrain.



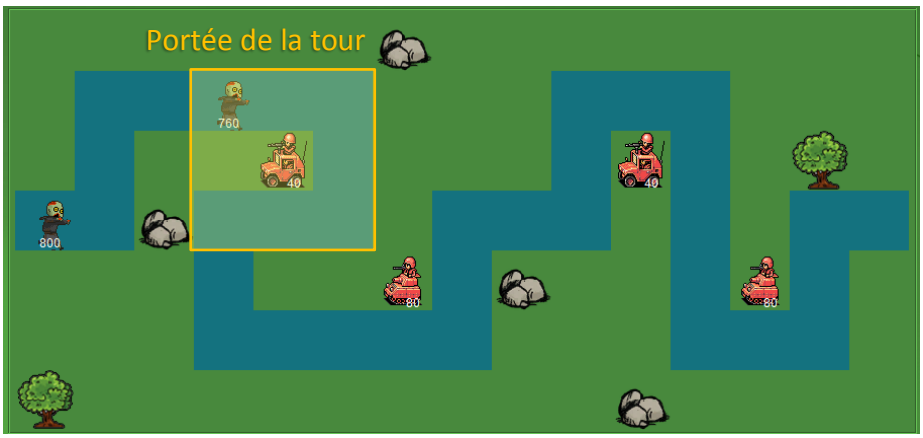
4.4.3. Attaque des tours

4.4.3.1 Tours Offensives

Les tours offensives ont pour objectifs de détruire les zombies en leur infligeant des dégâts sachant que le zombie meurt lorsqu'il n'a plus de point de vie. Il existe différentes tours offensives dans le jeu et leur attaque ainsi que leur portée peuvent varier selon le type de tour. Par exemple, pour une tour de type 0 la portée est de 1 ; le zombie ne sera attaqué que lorsqu'il rentrera dans le champ d'attaque de la tour.

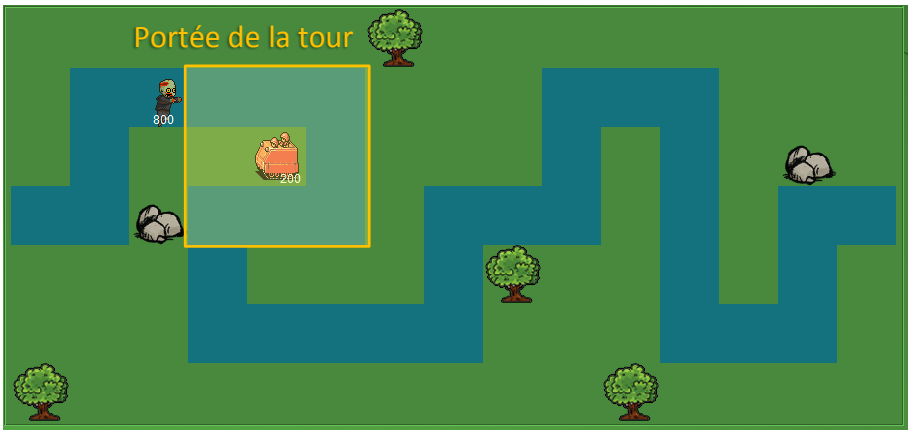


Lorsque le zombie est à portée de la tour, la tour lui inflige autant de dégâts qu'elle a d'attaque jusqu'à ce qu'il sorte du champ ou qu'il meurt. Nous avons affiché la vie de chaque zombie pour pouvoir suivre le déroulement du jeu.

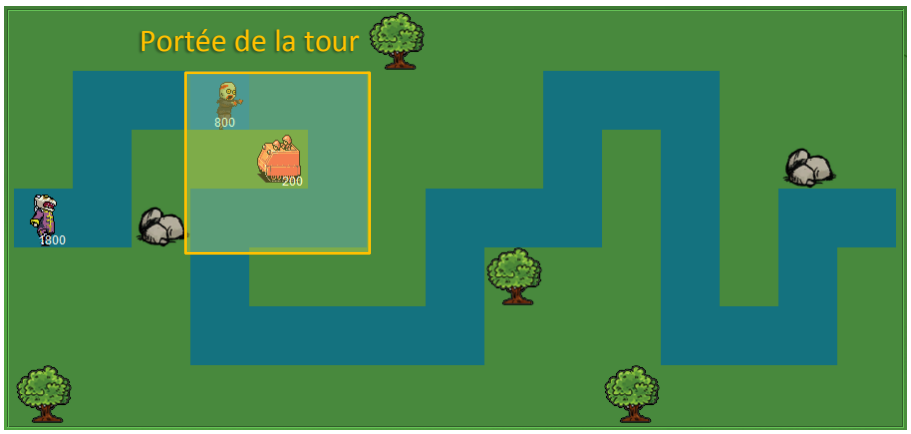


4.4.3.2 Tours Ralentissantes

De la même manière, les tours ralentissantes diminuent la vitesse des zombies ce qui permet aux tours offensives de les attaquer plus longtemps. Le zombie sera alors ralenti pendant un certain temps avant de reprendre sa vitesse de départ.



Lorsque le zombie est ralenti, nous changeons le fond de la case pour l'indiquer au joueur.



5. Listing

5.1. Classe Jeu

```
1 /*
 2  EPF 2A
 3  Auteurs : Nicolas Breton et Cecile Tran
 4  Zombie shooter
 5  Groupe A
 6  Date : 02/02/15
 7  Edit : 15/05/15
 8  */
 9 package bretontran_projetcpo;
10
11 import java.io.BufferedReader;
12 import java.io.FileNotFoundException;
13 import java.io.FileReader;
14 import java.io.FileWriter;
15 import java.io.IOException;
16 import java.util.Iterator;
17 import java.util.LinkedList;
18 import java.util.NoSuchElementException;
19 import java.util.Random;
20 import java.util.logging.Level;
21 import java.util.logging.Logger;
22 import javax.swing.JLabel;
23 import pfiches.MonLabel;
24
25 public class Jeu {
26
27     final int CHEMIN = 2;
28     final int COLONNES = 15; // Nombre de colonnes du plateau
29     final int LIGNES = 7; // Nombre de lignes du plateau
30     final int CASES = 105; // Nombre de cases dans le plateau
31     final int FIN = 5; // Nombre de vagues maximum
32     final String NOM_FICHIER = "Terrain.txt"; // Fichier de sauvegarde des tours
33     public static final String DOSS_IMAGES = "src/pimages/";
34
35     // Linked List a chaque case du tableau a 2 dimensions
36     private LinkedList<NonJoueur> terrain[][] = new LinkedList[LIGNES][COLONNES];
```

```
37  private LinkedList<Zombie> listeZombies;// Liste des differents types de Zombie
38  private LinkedList<Tour> listeTours;// Liste des differentes tours
39  private LinkedList<TourOff> listeToursTerrain;// Liste des tours offensives sur le terrain
40  private LinkedList<TourRal> listeToursTerrain2;// Liste des tours ralentissantes sur le terrain
41  private LinkedList<Zombie> listeZombTerrain;//Liste des zombies sur le terrain
42  private LinkedList<Zombie> listeZombSuppr;//Liste des zombies sur le terrain
43
44  private Joueur joueur;
45  private int timer = 0; // Compteur
46  private int nbreAlZ, nbreAjoutes = 0; // Valeurs tirees aleatoirement
47  private Zombie terrainAv[][] = new Zombie[LIGNES][COLONNES];
48  public int vagues = 0;
49
50  public Jeu(String nom) {
51
52      joueur = new Joueur(nom);
53
54      // Initialisation de toutes les linked list du terrain
55      for (int i = 0; i < terrain.length; i++) {
56          for (int j = 0; j < terrain[i].length; j++) {
57              LinkedList<NonJoueur> maListe = new LinkedList<>();
58              terrain[i][j] = maListe;
59          }
60      }
61
62      this.listeToursTerrain = new LinkedList<>();
63      this.listeToursTerrain2 = new LinkedList<>();
64      this.listeZombTerrain = new LinkedList<>();
65      this.listeZombSuppr = new LinkedList<>();
66      this.listeZombies = new LinkedList<>();
67
68      // Type, Vitesse (ms), vie, attaque, valeur
69      Zombie z = new Zombie(0, 400, 800, 1, 100); // Zombie rapide et faible
70      Zombie z2 = new Zombie(1, 800, 1000, 2, 200); // Zombie moyen
71      Zombie z3 = new Zombie(2, 1000, 1800, 3, 300); // Zombie costaud
72      Zombie zb = new Zombie(3, 1200, 20000, 15, 1000); // Zombie boss
73
74      listeZombies.add(z);
75      listeZombies.add(z2);
76      listeZombies.add(z3);
77      listeZombies.add(zb);
78
79      this.listeTours = new LinkedList<>();
80
81      //Type, portee, cout, vitesse(ms), attaque, ligne, colonne
82      TourOff to1 = new TourOff(0, 1, 100, 400, 40, 0, 0);
83      TourOff to2 = new TourOff(1, 1, 200, 800, 80, 0, 0);
84      TourOff to3 = new TourOff(2, 2, 100, 400, 60, 0, 0);
85      TourRal tr = new TourRal(3, 1, 100, 2000, 200, 0, 0);
86      TourOff to4 = new TourOff(4, 1, 200, 400, 60, 0, 0);
87      TourOff to5 = new TourOff(5, 1, 400, 800, 100, 0, 0);
88      TourOff to6 = new TourOff(6, 2, 600, 400, 80, 0, 0);
89      TourRal tr2 = new TourRal(7, 1, 400, 1200, 200, 0, 0);
90
91      listeTours.add(to1);
92      listeTours.add(to2);
93      listeTours.add(to3);
94      listeTours.add(tr);
95      listeTours.add(to4);
96      listeTours.add(to5);
97      listeTours.add(to6);
98      listeTours.add(tr2);
```

```
 99    }
100
101    @Override
102    public String toString() {
103
104        String ch = "";
105        for (int i = 0; i < terrain.length; i++) {
106            for (int j = 0; j < terrain[i].length; j++) {
107                ch = ch + terrain[i][j] + " ";
108            }
109            ch = ch + System.lineSeparator();
110        }
111        return ch;
112    }
113
114    public void chargerChemin(int chem) {
115
116        FileReader fich = null;
117        try {
118            fich = new FileReader("Chemin" + chem + ".txt");
119            BufferedReader br = new BufferedReader(fich);
120            String ligne; // IOException
121            String tab[] = new String[CASES];
122            Chemin Lechemin;
123            int l, c;
124            int i = 0;
125            do {
126                // On lit chaque ligne du fichier texte :
127                ligne = br.readLine();
128                i++;
129                if (ligne != null) {
130                    // On entre les coordonnees dans un tableau :
131                    tab = ligne.split("/");
132                    l = Integer.valueOf(tab[0]);
133                    c = Integer.valueOf(tab[1]);
134                    Lechemin = new Chemin(i, l, c);
135                    // On ajoute le chemin aux coordonnees donnees :
136                    terrain[l][c].addFirst(Lechemin);
137                }
138            } while (ligne != null);
139            fich.close(); // IOException
140        } catch (IOException ex) {
141        }
142    }
143
144    public void chargerObstacle(int chem) {
145
146        FileReader fich = null;
147        try {
148            fich = new FileReader("Foret" + chem + ".txt");
149            BufferedReader br = new BufferedReader(fich);
150            String ligne;// IOException
151            String tab[] = new String[CASES];
152            Obstacle Obs;
153            int l, c;
154
155            do {
156                // On lit chaque ligne du fichier texte :
157                ligne = br.readLine();
158                if (ligne != null) {
159                    // On entre les coordonnees dans un tableau :
160                    tab = ligne.split("/");
```

```
161                 l = Integer.valueOf(tab[0]);
162                 c = Integer.valueOf(tab[1]);
163                 Obs = new Obstacle(l, c);
164                 // On ajoute le chemin aux coordonnees donnees :
165                 terrain[l][c].addFirst(Obs);
166             }
167         } while (ligne != null);
168         fich.close();//IOException
169     } catch (IOException ex) {
170     }
171 }
172
173 public void choixNiveau(int niv) {
174     joueur.choixNiveau(niv);
175 }
176
177 public boolean PossibleOuPas(int type) {
178     Tour nomTour = listeTours.get(type);
179     if (joueur.getArgent() - nomTour.getCout() < 0) {
180         return false;
181     }
182     return true;
183 }
184
185 public boolean acheterTours(int type, int ligne, int colonne) {
186     // type : numero de la tour dans la liste
187     Tour nomTour = listeTours.get(type);
188     Tour newTour = nomTour.copierTour();
189
190     // Si le joueur peut acheter la tour :
191     if (joueur.acheterTours(newTour) == true) {
192         // Si la case est vide (pas de tour)
193         if (terrain[ligne][colonne].isEmpty() == false) {
194             NonJoueur PNJ = terrain[ligne][colonne].getFirst();
195             if (PNJ.isTour() == true) {
196                 Tour tour = (Tour) terrain[ligne][colonne].getFirst();
197                 if (tour.getType() == type - 4) {
198                     supprimerTours(tour);
199                     boolean placer = placerTour(newTour, ligne, colonne);
200                     if (placer == true) {
201                         joueur.PassageALaCaisse(newTour);
202                     }
203                     return placer;
204                 }
205             }
206
207         } else {
208             boolean placer = placerTour(newTour, ligne, colonne);
209             if (placer == true) {
210                 joueur.PassageALaCaisse(newTour);
211             }
212             return placer;
213         }
214     }
215     return false;
216 }
217
218 public boolean placerTour(Tour tour, int ligne, int colonne) {
219
220     if (isTourOff(tour) == true) {
221         TourOff tourOff = new TourOff(tour.getType(), tour.getPortee(), tour.getCout(), tour.getVitesse(),
tour.getAttaque(), tour.getLigne(), tour.getColonne());
```

```
222             tourOff.setTemps(timer);// On fait correspondre le temps de la tour avec le timer
223             // Lorsque l'objet a placer est une tour, on verifie que la case est vide :
224             if (terrain[ligne][colonne].isEmpty() == true) {
225                 tourOff.initCoord(ligne, colonne);
226                 listeToursTerrain.add(tourOff);  // On ajoute la tour dans la liste des tours du même type sur le terrain
227                 terrain[ligne][colonne].add(tourOff);
228                 return true;
229             }
230             return false;
231         } else {
232             TourRal tourRal = new TourRal(tour.getType(), tour.getPortee(), tour.getCout(), tour.getVitesse(),
tour.getAttaque(), tour.getLigne(), tour.getColonne());
233             tourRal.setTemps(timer);// On fait correspondre le temps de la tour avec le timer
234             if (terrain[ligne][colonne].isEmpty() == true) {
235                 tourRal.initCoord(ligne, colonne);
236                 listeToursTerrain2.add(tourRal);  // On ajoute la tour dans la liste des tours du même type sur le terrain
237                 terrain[ligne][colonne].add(tourRal);
238                 return true;
239             }
240             return false;
241         }
242     }
243
244     public void creerZombie(int type) {
245
246         Zombie nomZombie = listeZombies.get(type);
247         Zombie newZombie = nomZombie.copierZombie(timer);
248         placerZombie(newZombie, 3, 0);
249     }
250
251     public void placerZombie(Zombie z, int ligne, int colonne) {
252
253         z.initCoord(ligne, colonne);
254         listeZombTerrain.add(z);
255         // Si l'objet a placer est un zombie on verifie qu'il est bien place sur un chemin:
256         NonJoueur NJ = terrain[ligne][colonne].getFirst();
257
258         if (terrain[ligne][colonne].isEmpty() != true && NJ.isChemin() == true) {
259             terrain[ligne][colonne].add(z);
260         }
261     }
262
263     public boolean testCase(int ligne, int colonne, NonJoueur ch, Zombie zombie) {
264         // On verifie si la case possede un chemin :
265         if ((ligne >= 0 && ligne < LIGNES && colonne >= 0 && colonne < COLONNES) &&
(terrain[ligne][colonne].isEmpty() == false)) {
266             NonJoueur ch2 = terrain[ligne][colonne].getFirst();
267
268             // Si oui, on fait avancer le zombie :
269             if ((ch2.isChemin() == true) && (ch2.getOrdre() > ch.getOrdre())) {
270                 terrain[ligne][colonne].add(zombie);// On ajoute le zombie sur le prochaine case
271                 supprimerZombie(zombie);// On le supprime sur l'ancienne
272                 zombie.initCoord(ligne, colonne);// On reinitialise ces coordonnees
273                 return true;
274             }
275         }
276         return false;
277     }
278
279     public void avancerZombie() {
280
281         Iterator<Zombie> it = listeZombTerrain.iterator();
```

```
282
283         // S'il y a un zombie sur le terrain
284         if (listeZombTerrain.isEmpty() == false) {
285             while (it.hasNext()) {
286                 Zombie zomb = it.next();
287 
288                 // Si la vitesse + le temps du zombie = timer
289                 if ((zomb.getTemps() + zomb.getVitesse()) == timer) {
290                     int ligne = zomb.getLigne();
291                     int colonne = zomb.getColonne();
292 
293                     //Si le zombie est arrive au bout du chemin
294                     if (ligne == 3 && colonne == 14) {
295                         joueur.perdVie(zomb);
296                         listeZombSuppr.add(zomb);
297                     }
298 
299                     if (timer % 500 == 0) { // Toutes les 0.5 secondes
300                         int type = zomb.getType();
301                         Zombie ZombieInit = listeZombies.get(type);
302                         // On reinitalise la vitesse du zombie après etre ralenti
303                         zomb.reinitVitesse(ZombieInit.getVitesse());
304                     }
305 
306                     NonJoueur ch = terrain[ligne][colonne].getFirst();
307 
308                     try {
309                         // On verifie si les cases aux alentours sont des chemins :
310                         if (testCase(ligne + 1, colonne, ch, zomb) == true) {
311 
312                         } else if (testCase(ligne, colonne + 1, ch, zomb) == true) {
313 
314                         } else if (testCase(ligne, colonne - 1, ch, zomb) == true) {
315 
316                         } else if (testCase(ligne - 1, colonne, ch, zomb) == true) {
317 
318                         }
319                     } catch (NoSuchElementException ex) {
320                         System.out.println("Déplacement impossible\n");
321                     }
322                     zomb.modiftemps(timer);
323                 }
324             }
325         }
326     }
327 
328     public void supprimerZombie(Zombie z) {
329 
330         int ligne = z.getLigne();
331         int colonne = z.getColonne();
332 
333         Iterator<NonJoueur> it = terrain[ligne][colonne].iterator();
334         while (it.hasNext()) {
335             NonJoueur PNJ2 = it.next();
336             if (PNJ2 == z) {// Si le personnage correspond
337                 // On le supprime de la case:
338                 it.remove();
339             }
340         }
341 
342         Iterator<Zombie> it2 = listeZombTerrain.iterator();
343         while (it2.hasNext()) {
```

```
344             Zombie z2 = it2.next();
345             if ((z2 == z && z2.getVie() <= 0) || (z2 == z && z2.getLigne() == 3 && z2.getColonne() == 14)) {// Si le
zombie est mort ou a atteint la fin du terrain
346                 // On le supprime du terrain:
347                 it2.remove();
348             }
349         }
350     }
351
352     public boolean verifTour(int ligne, int colonne) {
353         if (terrain[ligne][colonne].isEmpty() == false) {
354             return true;
355         } else {
356             return false;
357         }
358     }
359
360     public void supprimerTour(int ligne, int colonne) {
361         //Vu qu'il n'y a qu'une tour au plus dans la linked list on peut la vider
362         terrain[ligne][colonne].clear();
363
364         Iterator<TourOff> it = listeToursTerrain.iterator();
365         Iterator<TourRal> it2 = listeToursTerrain2.iterator();
366
367         while (it.hasNext()) {
368             TourOff t = it.next();
369             int i = t.getLigne();
370             int j = t.getColonne();
371             if ((i == ligne) && (j == colonne)) {// Si le personnage correspond
372                 // On le supprime :
373                 it.remove();
374             }
375         }
376
377         while (it2.hasNext()) {
378             TourRal t2 = it2.next();
379             int i = t2.getLigne();
380             int j = t2.getColonne();
381             if ((i == ligne) && (j == colonne)) {// Si le personnage correspond
382                 // On le supprime :
383                 it2.remove();
384             }
385         }
386     }
387
388     public void supprimerTours(Tour t) {
389
390         int ligne = t.getLigne();
391         int colonne = t.getColonne();
392
393         Iterator<NonJoueur> it = terrain[ligne][colonne].iterator();
394         while (it.hasNext()) {
395             NonJoueur PNJ2 = it.next();
396             if (PNJ2 == t) {// Si le personnage correspond
397                 // On le supprime :
398                 it.remove();
399             }
400         }
401
402         if (isTourOff(t) == true) {
403             Iterator<TourOff> it2 = listeToursTerrain.iterator();
404             while (it2.hasNext()) {
```

```
405                 Tour t2 = it2.next();
406                 if (t2 == t) {// Si le personnage correspond
407                     // On le supprime :
408                     it2.remove();
409                 }
410             }
411         } else {
412             Iterator<TourRal> it2 = listeToursTerrain2.iterator();
413             while (it2.hasNext()) {
414                 Tour t2 = it2.next();
415                 if (t2 == t) {// Si le personnage correspond
416                     // On le supprime :
417                     it2.remove();
418                 }
419             }
420         }
421     }
422
423     // On verifie s'il y a un zombie dans le champ de la tour :
424     public Zombie verifZombie(Tour tour) {
425         int ligne = 0;
426         int colonne = 0;
427
428         int i = tour.getLigne();
429         int j = tour.getColonne();
430
431         if (terrain[i][j].contains(tour)) {
432             ligne = i;
433             colonne = j;
434
435             int inc = tour.getPortee();// On recupere la portee de la tour
436
437             // On verifie si la portee de la tour n'est pas hors terrain :
438             int minL = ligne - inc;
439             if (minL < 0) {
440                 minL = 0;
441             }
442
443             int maxL = ligne + inc;
444             if (maxL > LIGNES - 1) {
445                 maxL = LIGNES - 1;
446             }
447
448             int minC = colonne - inc;
449             if (minC < 0) {
450                 minC = 0;
451             }
452
453             int maxC = colonne + inc;
454             if (maxC > COLONNES - 1) {
455                 maxC = COLONNES - 1;
456             }
457
458             int ordre1 = 0;
459             int l = 0;
460             int c = 0;
461             int ordre2;
462
463             // On parcourt le champ d'attaque de la tour :
464             for (int k = minL; k <= maxL; k++) {
465                 for (int w = minC; w <= maxC; w++) {
466                     if (terrain[k][w].isEmpty() == false) {
```

```
467
468                     NonJoueur NJ = terrain[k][w].getFirst();
469                     NonJoueur NJ2 = terrain[k][w].getLast();
470
471                     if (NJ.isChemin() == true) {
472                         ordre2 = NJ.getOrdre();
473                         // Si le zombie est le plus avance
474                         if (NJ2.isZombie() == true && ordre2 > ordre1) {
475                             ordre1 = ordre2;
476                             // On sauvegarde sa position
477                             l = k;
478                             c = w;
479                         }
480                     }
481                 }
482             }
483         }
484
485         // On attaque le zombie le plus avance sur le chemin
486         if (terrain[l][c].isEmpty() == false) {
487             NonJoueur NJ3 = terrain[l][c].getLast();
488             if (NJ3.isZombie() == true) {
489                 return (Zombie) NJ3;
490             }
491         }
492     }
493     return null;
494 }
495
496 public void attaque() {
497
498     if (listeToursTerrain.isEmpty() == false) {
499         Iterator<TourOff> it = listeToursTerrain.iterator();
500         while (it.hasNext()) {
501
502             TourOff tourOff = it.next();
503
504             if ((tourOff.getTemps() + tourOff.getVitesse()) == timer) {
505                 Zombie zombie = verifZombie(tourOff);
506
507                 if (zombie != null) {
508                     // Si le zombie n'a plus de vie :
509                     if (tourOff.attaque(zombie) != true) {
510                         // On le supprime :
511                         supprimerZombie(zombie);
512                         joueur.incremArgent(zombie.getValeur());
513                         joueur.incremScore(zombie.getScore());
514                     }
515                 }
516                 tourOff.modiftemps(timer);
517             }
518         }
519     }
520
521     if (listeToursTerrain2.isEmpty() == false) {
522         Iterator<TourRal> it2 = listeToursTerrain2.iterator();
523         while (it2.hasNext()) {
524             TourRal tourRal = it2.next();
525
526             if ((tourRal.getTemps() + tourRal.getVitesse()) == timer) {
527                 Zombie zombie = verifZombie(tourRal);
528
```

```
529                 if (zombie != null) {
530                     // Si le zombie n'a plus de vie :
531                     if (tourRal.attaque(zombie) != true) {
532                         joueur.incremArgent(zombie.getValeur());
533                         joueur.incremScore(zombie.getScore());
534                     }
535                 }
536                 tourRal.modiftemps(timer);
537             }
538         }
539     }
540 }
541
542 // On sauvegarde les cases contenants une tour :
543 public void sauvegarderTerrain() {
544     try {
545         FileWriter fich = new FileWriter(NOM_FICHIER);
546
547         for (int i = 0; i < LIGNES; i++) {
548             for (int j = 0; j < COLONNES; j++) {
549                 if (terrain[i][j].isEmpty() == false) { // si la case n'est pas vide
550                     NonJoueur PNJ = terrain[i][j].getFirst();
551
552                     // Si ce n'est pas un chemin et un obstacle
553                     if (PNJ.isChemin() == false && PNJ.isObstacle() == false) {
554                         fich.write(i + "," + j);// On ecrit les coordonnees de la case
555                         Iterator it = terrain[i][j].iterator();
556                         // Tant que l'iterateur n'est pas arrivé à la fin de la liste :
557                         while (it.hasNext() == true) {
558                             // On ecrit le type de la tour :
559                             fich.write("," + it.next());
560                         }
561                         fich.write(System.lineSeparator());
562                     }
563                 }
564             }
565         }
566         fich.close();
567     } catch (IOException ex) {
568         System.out.println("La partie n'a pas pu etre sauvegardee ");
569     }
570 }
571
572 public void chargerTerrain() throws FileNotFoundException, IOException {
573
574     FileReader fich = new FileReader(NOM_FICHIER);// FileNotFoundException
575     BufferedReader br = new BufferedReader(fich);
576     String ligne = "";// IOException
577
578     String[] tab = new String[8];
579
580     while (ligne != null) {
581         ligne = br.readLine();
582         if (ligne != null) {// si la ligne n'est pas vide
583             // On ajoute les elements de la ligne dans un tableau :
584             tab = ligne.split(",");
585             // On recupere les coordonnees :
586             int l = Integer.valueOf(tab[0]);
587             int c = Integer.valueOf(tab[1]);
588             // Et le type :
589             int type = Integer.valueOf(tab[2]);
590             // Puis on recreer la tour correspondante :
```

```
591             Tour nomTour = listeTours.get(type);
592
593             Tour newTour = nomTour.copierTour();
594             if (isTourOff(newTour) == true) {
595                 TourOff tourOff = new TourOff(newTour.getType(), newTour.getPortee(), newTour.getCout(),
newTour.getVitesse(), newTour.getAttaque(), newTour.getLigne(), newTour.getColonne());
596                 listeToursTerrain.add(tourOff);
597             } else {
598                 TourRal tourRal = new TourRal(newTour.getType(), newTour.getPortee(), newTour.getCout(),
newTour.getVitesse(), newTour.getAttaque(), newTour.getLigne(), newTour.getColonne());
599                 listeToursTerrain2.add(tourRal);
600             }
601             // Et la replace sur le terrain aux coordonnees indiquees :
602             placerTour(newTour, l, c);
603         }
604     }
605     fich.close();//IOException
606 }
607
608 public void afficherJoueur() {
609     joueur.afficher();
610 }
611
612 public void sauvegarderJoueur() {
613     joueur.sauvegarderJoueur();
614 }
615
616 public void chargerJoueur() {
617
618     try {
619         joueur.chargerJoueur();
620     } catch (IOException ex) {
621     }
622 }
623
624 public void sauvegarderScore() {
625     try {
626         System.out.println("Sauvegarde Score");
627         joueur.lireJoueur();
628         joueur.sauvegarderScore();
629     } catch (IOException ex) {
630         Logger.getLogger(Jeu.class.getName()).log(Level.SEVERE, null, ex);
631     }
632 }
633
634 public void afficherTerrain(MonLabel tabLabel[][]) {
635
636     for (int i = 0; i < terrain.length; i++) {
637         for (int j = 0; j < terrain[i].length; j++) {
638             if (terrain[i][j].isEmpty() == false) {
639
640                 // Si la case est un chemin :
641                 if (terrain[i][j].getFirst().isChemin() == true && terrain[i][j].getLast().isChemin() == true) {
642                     terrain[i][j].getFirst().affichageImage(tabLabel[i][j]);
643                 }
644                 // Si la case est un obstacle :
645                 if (terrain[i][j].getFirst().isObstacle() == true && terrain[i][j].getLast().isObstacle() == true) {
646                     terrain[i][j].getFirst().affichageImage(tabLabel[i][j]);
647                 }
648
649                 NonJoueur NJ = terrain[i][j].getLast();
650
```

```
651                 // S'il y a un zombie sur la case et qu'il vient de se déplacer sur la case :
652                 if ((terrain[i][j].getLast().isChemin() == false && NJ.getTemps() == timer) && (NJ.isZombie() == true) ||
(terrain[i][j].getLast().isChemin() == false && terrainAv[i][j] != null && terrainAv[i][j].getTemps() == timer)) {
653
654                     NJ.affichageImage(tabLabel[i][j]);
655                     terrain[i][j].getLast().affichageImage(tabLabel[i][j]);
656                     terrainAv[i][j] = (Zombie) NJ;
657
658                 } else if (NJ.isTour() == true) { // Si c'est une tour
659                     NJ.affichageImage(tabLabel[i][j]);
660                     NJ.affichageText(tabLabel[i][j]);
661                 } else if (NJ.isZombie() == true) { // Si c'est un Zombie
662                     // On affiche le texte correspondant à l'image :
663                     terrain[i][j].getLast().affichageText(tabLabel[i][j]);
664                 }
665             } else {
666                 tabLabel[i][j].setIcon(null);
667                 tabLabel[i][j].setText(null);
668             }
669         }
670     }
671 }
672
673 public void trierScore(JLabel tabLabel[][]) {
674     joueur.trierScores(tabLabel);
675 }
676
677 public int getVie() {
678     return joueur.getVie();
679 }
680
681 public int getArgent() {
682     return joueur.getArgent();
683 }
684
685 public int getNbrVagues() {
686     return joueur.getNbrVagues();
687 }
688
689 public int getScore() {
690     return joueur.getScore();
691 }
692
693 public int getNiveau() {
694     return joueur.getNiveau();
695 }
696
697 public String getNom() {
698     return joueur.getNom();
699 }
700
701 public void lancerVague() {
702
703     Random rand = new Random();
704     int i = joueur.getNbrVagues();
705
706     if (i < FIN - 1) {
707         nbreAlZ = rand.nextInt((8 + i) - (5 + i) + 1) + (5 + i); // nextInt(max-min+1)+min
708         nbreAjoutes = 0;
709         System.out.println("\nnbrealZ: " + nbreAlZ);
710         ajouterZombie();
711     } else {
```

```
712             creerZombie(3);
713         }
714     }
715
716     // Ajout aleatoire de Zombie :
717     public void ajouterZombie() {
718
719         if (nbreAjoutes < nbreAlZ) {
720             System.out.println("nbrajoutes: " + nbreAjoutes);
721             Random rand = new Random();
722             // Tire un nombre aleatoire entre 0 et 2 (compris) => type du zombie creer :
723             int nbreAlT = rand.nextInt(3);
724             System.out.println("nbrealt: " + nbreAlT);
725             this.creerZombie(nbreAlT);
726             nbreAjoutes++;
727         }
728     }
729
730     public void incremNbrVagues() {
731         joueur.incremNbrVagues();
732     }
733
734     // On verifie s'il n'y a plus de zombies sur le terrain :
735     public boolean terrainVide() {
736         if (listeZombTerrain.isEmpty() == true) {
737             return true;
738         }
739         return false;
740     }
741
742     // On verifie si une des case du terrain est vide :
743     public boolean caseVide(int l, int c) {
744         if (terrain[l][c].isEmpty() == true) {
745             return true;
746         }
747         return false;
748     }
749
750     // On reinitialise le timer à 0 :
751     public void reinitTimer() {
752         timer = 0;
753         Iterator<TourOff> it = listeToursTerrain.iterator();
754         // On reinitialise le timer de toutes les tours
755         while (it.hasNext()) {
756             TourOff tourOff = it.next();
757             tourOff.modiftemps(0);
758         }
759
760         Iterator<TourRal> it2 = listeToursTerrain2.iterator();
761         // On reinitialise le timer de toutes les tours
762         while (it2.hasNext()) {
763             TourRal tourRal = it2.next();
764             tourRal.modiftemps(0);
765         }
766     }
767
768     // Boucle de jeu utilise dans le run :
769     public void Jeu() {
770
771         timer = timer + 200;
772
773         if (timer % 2000 == 0 && joueur.getNbrVagues() < FIN - 2) { // Toutes les 2 secondes
```

```
774             ajouterZombie();
775         }
776
777         avancerZombie();
778         attaque();
779
780         Iterator<Zombie> it = listeZombSuppr.iterator();
781         while (it.hasNext()) {
782             Zombie z = it.next();
783             supprimerZombie(z);
784             it.remove();
785         }
786     }
787
788     // Description des tours de départ :
789     public String DescriptionTours(int type) {
790
791         Tour tour = listeTours.get(type);
792
793         String txt = "";
794         String Desc;
795
796         if (type != 3 && type != 7) {
797             txt = "- Offensive -";
798         } else {
799             txt = "- Ralentisseur -";
800         }
801
802         String t = " Tour " + type;
803         String c = " Coût : " + tour.getCout();
804         String a = " Attaque : " + tour.getAttaque();
805         String p = " Portée : " + tour.getPortee();
806         String v = " Vitesse : " + tour.getVitesse();
807
808         Desc = "<html>" + t + "<br>" + txt + "<br>" + c + "<br>" + p + "<br>" + a + "<br>" + v + "</html>";
809
810         return Desc;
811
812     }
813
814     // Description des ameliorations des tours sur le terrain :
815     public String DescriptionTours(int l, int c) {
816         int type = 0;
817
818         if (terrain[l][c].isEmpty() == false) {
819
820             Tour tour0 = (Tour) terrain[l][c].getFirst();
821             type = tour0.getType();
822             type = type + 4;
823             Tour tour = listeTours.get(type);
824
825             String txt = "";
826             String Desc;
827
828             if (type != 3 && type != 7) {
829                 txt = "- Offensive -";
830             } else {
831                 txt = "- Ralentisseur -";
832             }
833
834             String co = " Coût : " + tour.getCout();
835             String a = " Attaque : " + tour.getAttaque();
```

```
836             String p = " Portée : " + tour.getPortee();
837             String v = " Vitesse : " + tour.getVitesse();
838 
839             Desc = "<html>" + "Amélioration" + "<br>" + txt + "<br>" + co + "<br>" + p + "<br>" + a + "<br>" + v +
"</html>";
840 
841             return Desc;
842         }
843         return "";
844     }
845 
846     public boolean ameliorerTour(int ligne, int colonne) {
847 
848         if (terrain[ligne][colonne].getFirst().isTour()) {
849             Tour tour = (Tour) terrain[ligne][colonne].getFirst();
850             int type = tour.getType();
851 
852             if (type < 4) { // Si la tour est ameliorable (4 premiere tours de la liste)
853                 type = type + 4;
854                 // On achete l'amelioration si possible :
855                 if (acheterTours(type, ligne, colonne) == true) {
856                     return true;
857                 }
858             }
859         }
860         return false;
861     }
862 
863     // On verifie si la case du tableau est une tour :
864     public boolean tour(int l, int c) {
865 
866         if (terrain[l][c].isEmpty() == false) {
867             NonJoueur nj = terrain[l][c].getFirst();
868             if (nj.isTour() == true) {
869                 return true;
870             }
871         }
872         return false;
873     }
874 
875     public boolean isTourOff(Tour tour) {
876         if ((tour.getType() == 3) || (tour.getType() == 7)) {
877             return false;
878         } else {
879             return true;
880         }
881     }
882 
883     public boolean FinDePartie() {
884         if (joueur.getVie() <= 0 || (joueur.getNbrVagues() == FIN && terrainVide() == true)) {
885             return true;
886         } else {
887             return false;
888         }
889     }
890 }
891 
```

5.2. Classe Non Joueur

```
1 /*
2  EPF 2A
3  Auteurs : Nicolas Breton et Cecile Tran
4  Zombie shooter
5  Groupe A
6  Date : 02/02/15
7  Edit : 15/05/15
8  */
9 package bretontran_projetcpo;
10
11 import pfiches.MonLabel;
12
13 public class NonJoueur {
14
15     private int temps;
16     private int ligne;
17     private int colonne;
18
19     public NonJoueur(int temps, int ligne, int colonne) {
20
21         this.temps = temps;
22     }
23
24     public void initCoord(int l, int c) {
25         ligne = l;
26         colonne = c;
27     }
28
29     public void setTemps(int temps) {
30         this.temps = temps;
31     }
32
33     public int getOrdre() {
34         return 0;
35     }
36
37     public int getLigne() {
38         return ligne;
39     }
40
41     public int getColonne() {
42         return colonne;
43     }
44
45     public boolean isZombie() {
46         if (this instanceof Zombie) {
47             return true;
48         } else {
49             return false;
50         }
51     }
52
53     public boolean isChemin() {
54         if (this instanceof Chemin) {
55             return true;
56         } else {
57             return false;
58         }
59     }
```

```
60
61      public boolean isObstacle() {
62          if (this instanceof Obstacle) {
63              return true;
64          } else {
65              return false;
66          }
67      }
68
69      public boolean isTour() {
70          if (this instanceof Tour) {
71              return true;
72          } else {
73              return false;
74          }
75      }
76
77      public void modiftemps(int t) {
78          temps = t;
79      }
80
81      public int getTemps() {
82          return temps;
83      }
84
85      public int getVitesse() {
86          return 0;
87      }
88
89      public void affichageImage(MonLabel monLabel) {
90      }
91
92      public void affichageText(MonLabel monLabel) {
93      }
94  }
95
```

5.3. Classe Obstacle

```
1  /*
2   EPF 2A
3   Auteurs : Nicolas Breton et Cecile Tran
4   Zombie shooter
5   Groupe A
6   Date : 02/02/15
7   Edit : 15/05/15
8   */
9  package bretontran_projetcpo;
10
11 import static bretontran_projetcpo.Jeu.DOSS_IMAGES;
12 import java.awt.Image;
13 import java.awt.Toolkit;
14 import java.util.Random;
15 import javax.swing.ImageIcon;
16 import pfiches.MonLabel;
17
18 public class Obstacle extends NonJoueur {
19
20      private static Image img;
21      private boolean deja;
```

```
22
23    Toolkit t = Toolkit.getDefaultToolkit();
24
25    public Obstacle(int l, int c) {
26        super(0, l, c);
27        deja = false;
28    }
29
30    public void affichageImage(MonLabel monLabel) {
31
32        if (deja == false) {
33            Random rand = new Random();
34            //Tire un nombre aléatoire entre 0 et 1 (compris)
35            int nbreAlT = rand.nextInt(2);
36            img = t.getImage(DOSS_IMAGES + "Obstacle" + nbreAlT + ".png");
37            img = img.getScaledInstance(monLabel.getWidth(), monLabel.getHeight(), Image.SCALE_DEFAULT);
38            monLabel.setIcon(new ImageIcon(img));
39        }
40        deja = true;
41    }
42 }
43
```

5.4. Classe Chemin

```
1  /*
2   EPF 2A
3   Auteurs : Nicolas Breton et Cecile Tran
4   Zombie shooter
5   Groupe A
6   Date : 02/02/15
7   Edit : 15/05/15
8   */
9  package bretontran_projetcpo;
10
11 import java.awt.Color;
12 import pfiches.MonLabel;
13
14 public class Chemin extends NonJoueur {
15
16    private int ordre; //Valeur qui augmente de 1 de case en case
17
18    public Chemin(int ordre, int l, int c) {
19        super(0, l, c);
20        this.ordre = ordre;
21    }
22
23    @Override
24    public String toString() {
25        return "C" + ordre + "";
26    }
27
28    public int getOrdre() {
29        return ordre;
30    }
31
32    public void affichageImage(MonLabel monLabel) {
33        monLabel.setBackground(new Color(20, 114, 127));
34        monLabel.setOpaque(true);
35        monLabel.setIcon(null);
```

```
36         monLabel.setText(null);
37     }
38
39 }
40
```

5.5. Classe Zombie

```
1 /*
 2  EPF 2A
 3  Auteurs : Nicolas Breton et Cecile Tran
 4  Zombie shooter
 5  Groupe A
 6  Date : 02/02/15
 7  Edit : 15/05/15
 8  */
 9 package bretontran_projetcpo;
10
11 import static bretontran_projetcpo.Jeu.DOSS_IMAGES;
12 import java.awt.Color;
13 import java.awt.Image;
14 import java.awt.Toolkit;
15 import javax.swing.ImageIcon;
16 import javax.swing.JLabel;
17 import pfiches.MonLabel;
18
19 public class Zombie extends NonJoueur {
20
21     private int vitesse;
22     private int vie;
23     private int attaque;
24     private int valeur;
25     private int type;
26
27     private static Image imgz0, imgz1, imgz2, imgzb;
28
29     public Zombie(int type, int vitesse, int vie, int attaque, int valeur) {
30         super(0, 3, 0);
31         this.vitesse = vitesse;
32         this.vie = vie;
33         this.attaque = attaque;
34         this.valeur = valeur;
35         this.type = type;
36     }
37
38     public Zombie(int type, int vitesse, int vie, int attaque, int valeur, int temps) {
39         super(temps, 3, 0);
40         this.vitesse = vitesse;
41         this.vie = vie;
42         this.attaque = attaque;
43         this.valeur = valeur;
44         this.type = type;
45     }
46
47     static {
48         Toolkit t = Toolkit.getDefaultToolkit();
49         //acces a l'image
50         imgz0 = t.getImage(DOSS_IMAGES + "Zombie0.gif");
51         imgz1 = t.getImage(DOSS_IMAGES + "Zombie1.gif");
52         imgz2 = t.getImage(DOSS_IMAGES + "Zombie2.gif");
```

```
53      imgzb = t.getImage(DOSS_IMAGES + "ZombieB.gif");
54    }
55
56    @Override
57    public String toString() {
58      return "Z{" + vie + '}';
59    }
60
61    public int getAttaque() {
62      return attaque;
63    }
64
65    public int getVitesse() {
66      return vitesse;
67    }
68
69    public int getVie() {
70      return vie;
71    }
72
73    public int getType() {
74      return type;
75    }
76
77    public int getValeur() {
78      return valeur;
79    }
80
81    public boolean baisseVie(int attaque) {
82      // Permet de baisser la vie du zombie lorsqu'attaque :
83      vie = vie - attaque;
84      if (vie <= 0) {
85        return false;
86      } else {
87        return true;
88      }
89    }
90
91    public Zombie copierZombie(int temps) {
92      return new Zombie(type, vitesse, vie, attaque, valeur, temps);
93    }
94
95    public boolean ralentit(int attaque) {
96      vitesse = vitesse + attaque;
97      return true;
98    }
99
100   public void reinitVitesse(int vitInit) {
101     vitesse = vitInit;
102   }
103
104   public int getScore() {
105     return valeur;
106   }
107
108   public void affichageImage(MonLabel monLabel) {
109
110     if (type == 0) {
111       imgz0 = imgz0.getScaledInstance(monLabel.getWidth(), monLabel.getHeight(), Image.SCALE_DEFAULT);
112       monLabel.setIcon(new ImageIcon(imgz0));
113
114     } else if (type == 1) {
```

```
115             imgz1 = imgz1.getScaledInstance(monLabel.getWidth(), monLabel.getHeight(), Image.SCALE_DEFAULT);
116             monLabel.setIcon(new ImageIcon(imgz1));
117
118         } else if (type == 2) {
119             imgz2 = imgz2.getScaledInstance(monLabel.getWidth(), monLabel.getHeight(), Image.SCALE_DEFAULT);
120             monLabel.setIcon(new ImageIcon(imgz2));
121
122         } else if (type == 3) {
123             imgzb = imgzb.getScaledInstance(monLabel.getWidth(), monLabel.getHeight(), Image.SCALE_DEFAULT);
124             monLabel.setIcon(new ImageIcon(imgzb));
125         }
126     }
127
128     public void affichageText(MonLabel monLabel) {
129         int v = getVie();
130         monLabel.setBackground(new Color(20, 114, 127));
131         monLabel.setText("" + v);
132         monLabel.setForeground(new Color(255, 255, 255));
133
134         // Si le zombie est ralentit on change la couleur du texte
135         if (type == 0) {
136             if (vitesse > 400) {
137                 monLabel.setBackground(new Color(1, 111, 166));
138                 monLabel.setForeground(new Color(204, 255, 255));
139             }
140         } else if (type == 1) {
141             if (vitesse > 800) {
142                 monLabel.setBackground(new Color(1, 111, 166));
143                 monLabel.setForeground(new Color(204, 255, 255));
144             }
145         } else if (type == 2) {
146             if (vitesse > 1000) {
147                 monLabel.setBackground(new Color(1, 111, 166));
148                 monLabel.setForeground(new Color(204, 255, 255));
149             }
150         } else if (type == 3) {
151             if (vitesse > 1200) {
152                 monLabel.setBackground(new Color(1, 111, 166));
153                 monLabel.setForeground(new Color(204, 255, 255));
154             }
155         }
156         monLabel.setVerticalTextPosition(JLabel.BOTTOM);
157         monLabel.setHorizontalTextPosition(JLabel.RIGHT);
158         monLabel.setIconTextGap(-35);
159     }
160 }
161
```

5.6. Classe Tour

```
1 /*
2  EPF 2A
3  Auteurs : Nicolas Breton et Cecile Tran
4  Zombie shooter
5  Groupe A
6  Date : 02/02/15
7  Edit : 15/05/15
8  */
9 package bretontran_projetcpo;
10
```

```
11 import static bretontran_projetcpo.Jeu.DOSS_IMAGES;
12 import java.awt.Color;
13 import java.awt.Image;
14 import java.awt.Toolkit;
15 import javax.swing.ImageIcon;
16 import javax.swing.JLabel;
17 import pfiches.MonLabel;
18
19 public class Tour extends NonJoueur {
20
21     private int portee;
22     private int cout; // Lors de l'achat
23     private int vitesse; // Vitesse de tir (degats par seconde)
24     private int type;
25     private int attaque;
26
27     private static Image imgt0, imgt1, imgt2, imgt3, imgt4, imgt5, imgt6, imgt7;
28
29     public Tour(int type, int portee, int cout, int vitesse, int attaque, int ligne, int colonne) {
30         super(0, ligne, colonne);
31         this.type = type;
32         this.portee = portee;
33         this.cout = cout;
34         this.vitesse = vitesse;
35         this.attaque = attaque;
36     }
37
38     static {
39         // acces au toolkit
40         Toolkit t = Toolkit.getDefaultToolkit();
41         // acces a l'image
42         imgt0 = t.getImage(DOSS_IMAGES + "Tour0.png");
43         imgt1 = t.getImage(DOSS_IMAGES + "tour1.png");
44         imgt2 = t.getImage(DOSS_IMAGES + "tour2.png");
45         imgt3 = t.getImage(DOSS_IMAGES + "TourRal0.png");
46         imgt4 = t.getImage(DOSS_IMAGES + "tourEv0.png");
47         imgt5 = t.getImage(DOSS_IMAGES + "TourEv1.png");
48         imgt6 = t.getImage(DOSS_IMAGES + "TourEv2.png");
49         imgt7 = t.getImage(DOSS_IMAGES + "TourRalEv0.png");
50     }
51
52     public Tour copierTour() {
53         return new Tour(type, portee, cout, vitesse, attaque, 0, 0);
54     }
55
56     public int getType() {
57         return type;
58     }
59
60     public int getAttaque() {
61         return attaque;
62     }
63
64     public int getVitesse() {
65         return vitesse;
66     }
67
68     public String toString() {
69         return type + "";
70     }
71
72     public int getCout() {
```

```
73      return cout;
74  }
75
76  public int getPortee() {
77      return portee;
78  }
79
80  public boolean isTourOff() {
81      if (this instanceof TourOff) {
82          return true;
83      } else {
84          return false;
85      }
86  }
87
88  public boolean isTourRal() {
89      if (this instanceof TourRal) {
90          return true;
91      } else {
92          return false;
93      }
94  }
95
96  public void affichageImage(MonLabel monLabel) {
97
98      if (type == 0) {
99          imgt0 = imgt0.getScaledInstance(monLabel.getWidth(), monLabel.getHeight(), Image.SCALE_DEFAULT);
100         monLabel.setIcon(new ImageIcon(imgt0));
101
102     } else if (type == 1) {
103         imgt1 = imgt1.getScaledInstance(monLabel.getWidth(), monLabel.getHeight(), Image.SCALE_DEFAULT);
104         monLabel.setIcon(new ImageIcon(imgt1));
105
106     } else if (type == 2) {
107         imgt2 = imgt2.getScaledInstance(monLabel.getWidth(), monLabel.getHeight(), Image.SCALE_DEFAULT);
108         monLabel.setIcon(new ImageIcon(imgt2));
109
110     }
111     if (type == 3) {
112         imgt3 = imgt3.getScaledInstance(monLabel.getWidth(), monLabel.getHeight(), Image.SCALE_DEFAULT);
113         monLabel.setIcon(new ImageIcon(imgt3));
114
115     } else if (type == 4) {
116         imgt4 = imgt4.getScaledInstance(monLabel.getWidth(), monLabel.getHeight(), Image.SCALE_DEFAULT);
117         monLabel.setIcon(new ImageIcon(imgt4));
118
119     } else if (type == 5) {
120         imgt5 = imgt5.getScaledInstance(monLabel.getWidth(), monLabel.getHeight(), Image.SCALE_DEFAULT);
121         monLabel.setIcon(new ImageIcon(imgt5));
122
123     } else if (type == 6) {
124         imgt6 = imgt6.getScaledInstance(monLabel.getWidth(), monLabel.getHeight(), Image.SCALE_DEFAULT);
125         monLabel.setIcon(new ImageIcon(imgt6));
126
127     } else if (type == 7) {
128         imgt7 = imgt7.getScaledInstance(monLabel.getWidth(), monLabel.getHeight(), Image.SCALE_DEFAULT);
129         monLabel.setIcon(new ImageIcon(imgt7));
130     }
131 }
132
133 public void affichageText(MonLabel monLabel) {
134     int v = getAttaque();
```

```
135         monLabel.setText("" + v);
136         monLabel.setForeground(new Color(255, 255, 255));
137         monLabel.setVerticalTextPosition(JLabel.BOTTOM);
138         monLabel.setHorizontalTextPosition(JLabel.RIGHT);
139         monLabel.setIconTextGap(-26);
140     }
141 }
142
```

5.7. Classe TourOff

```
1 /*
2  EPF 2A
3  Auteurs : Nicolas Breton et Cecile Tran
4  Zombie shooter
5  Groupe A
6  Date : 02/02/15
7  Edit : 15/05/15
8  */
9 package bretontran_projetcpo;
10
11 public class TourOff extends Tour {
12
13     public TourOff(int type, int portee, int cout, int vitesse, int attaque, int ligne, int colonne) {
14         super(type, portee, cout, vitesse, attaque, ligne, colonne);
15     }
16
17     public boolean attaque(Zombie z) {
18         // On utilise l'attaque de la tour pour affaiblir le zombie :
19         return z.baisseVie(this.getAttaque());
20     }
21 }
22
```

5.8. Classe TourRal

```
1 /*
2  EPF 2A
3  Auteurs : Nicolas Breton et Cecile Tran
4  Zombie shooter
5  Groupe A
6  Date : 02/02/15
7  Edit : 15/05/15
8  */
9 package bretontran_projetcpo;
10
11 public class TourRal extends Tour {
12
13     public TourRal(int type, int portee, int cout, int vitesse, int attaque, int ligne, int colonne) {
14         super(type, portee, cout, vitesse, attaque, ligne, colonne);
15     }
16
17     public boolean attaque(Zombie z) {
18         // On ralentit le zombie avec l'attaque de la tour
19         return z.ralentit(this.getAttaque());
20     }
21 }
22
```

5.9. Classe Joueur

```
1 /*
2  EPF 2A
3  Auteurs : Nicolas Breton et Cecile Tran
4  Zombie shooter
5  Groupe A
6  Date : 02/02/15
7  Edit : 15/05/15
8  */
9 package bretontran_projetcpo;
10
11 import java.io.BufferedReader;
12 import java.io.FileNotFoundException;
13 import java.io.FileReader;
14 import java.io.FileWriter;
15 import java.io.IOException;
16 import java.util.Collections;
17 import java.util.Iterator;
18 import java.util.LinkedList;
19 import javax.swing.JLabel;
20
21 public class Joueur {
22
23    final String NOM_FICHIER = "Joueur.txt"; // Sauvegarde des donnees joueur
24    final String NOM_FICHIER2 = "Scores.txt"; // Fichier Liste de scores
25    final int ARGENT_DEPART = 600;
26
27    private String nom;
28    private int score;
29    private static int argent;
30    private int vie; // Depend de la difficulte
31    private int nbrVagues;
32    private int niveau; // Facile, moyen ou dur (pour le chemin choisi)
33
34    private LinkedList<Joueur> listeScore;// Liste des scores
35
36    public Joueur(String nom) {
37       this.nom = nom;
38       score = 0;
39       this.argent = ARGENT_DEPART;
40       nbrVagues = 0;
41       this.vie = 20;
42       this.listeScore = new LinkedList<>();
43    }
44
45    public Joueur(String nom, int score) {
46       this.nom = nom;
47       this.score = score;
48       this.listeScore = new LinkedList<>();
49    }
50
51    // La methode renvoie vrai si le joueur peut acheter la tour
52    public boolean acheterTours(Tour nomTour) {
53       if ((argent - nomTour.getCout()) >= 0) {
54          return true;
55       } else {
56          return false;
57       }
```

```
58    }
59
60    public void choixNiveau(int niv) {
61        niveau = niv;
62    }
63
64    public int getNiveau() {
65        return niveau;
66    }
67
68    @Override
69    public String toString() { // Pour le fichier de sauvegarde
70        return nom + "/" + score + "/" + argent + "/" + vie + "/" + nbrVagues + "/" + niveau;
71    }
72
73    public void afficher() { // Pour les tests
74        System.out.println("Nom = " + nom + "\nScore = " + score + "\nArgent = " + argent + "\nVie = " + vie +
"\nNombre de vagues = " + nbrVagues + "\nNiveau = " + niveau);
75    }
76
77    public String getNom() {
78        return nom;
79    }
80
81    public int getScore() {
82        return score;
83    }
84
85    public int getVie() {
86        return vie;
87    }
88
89    public int getNbrVagues() {
90        return nbrVagues;
91    }
92
93    public static int getArgent() {
94        return argent;
95    }
96
97    public void PassageALaCaisse(Tour nomTour) {
98        argent = argent - nomTour.getCout();
99    }
100
101   public void sauvegarderJoueur() {
102       try {
103           FileWriter fich = new FileWriter(NOM_FICHIER);
104           fich.write(this.toString());
105           fich.close();
106       } catch (IOException ex) {
107           System.out.println("La partie n'a pas pu etre sauvegardee ");
108       }
109
110   }
111
112   public void chargerJoueur() throws IOException { // Throws : pour propager l'exception
113
114       FileReader fich = new FileReader(NOM_FICHIER); // FileNotFoundException
115       BufferedReader br = new BufferedReader(fich);
116       String ligne;// IOException
117
118       String[] tab = new String[10];
```

```
119
120         ligne = br.readLine();
121
122         // On place les elements de la ligne dans un tableau
123         tab = ligne.split("/");
124
125         // On definit les attributs du joueur avec les elements du fichier
126         nom = tab[0];
127         score = Integer.valueOf(tab[1]);
128         argent = Integer.valueOf(tab[2]);
129         vie = Integer.valueOf(tab[3]);
130         nbrVagues = Integer.valueOf(tab[4]);
131         niveau = Integer.valueOf(tab[5]);
132
133         fich.close(); // IOException
134     }
135
136     public void perdVie(Zombie z) {
137         vie = vie - z.getAttaque();
138     }
139
140     public void incremScore(int s) {
141         score = score + s;
142     }
143
144     public void incremArgent(int a) {
145         argent = argent + a;
146     }
147
148     public void lireJoueur() throws FileNotFoundException, IOException {
149
150         FileReader fich = new FileReader(NOM_FICHIER2);// FileNotFoundException
151         BufferedReader br = new BufferedReader(fich);
152         String ligne;// IOException
153
154         String[] tab = new String[20];
155
156         ligne = br.readLine();
157         while (ligne != null) {
158
159             // On place les elements de la ligne dans un tableau
160             tab = ligne.split("/");
161
162             // On definit les attributs du joueur avec les elements du fichier
163             String fnom = tab[0];
164             int fscore = Integer.valueOf(tab[1]);
165
166             Joueur j = new Joueur(fnom, fscore);
167             listeScore.add(j);
168             ligne = br.readLine();
169         }
170         fich.close(); // IOException
171     }
172
173     public void sauvegarderScore() throws IOException {
174
175         FileWriter fich = new FileWriter(NOM_FICHIER2);// FileNotFoundException
176         Joueur j = new Joueur(this.nom, this.score);
177         listeScore.add(j);
178
179         Iterator<Joueur> it = listeScore.iterator();
180         // Tant que l'iterateur n'est pas arrive à la fin de la liste :
```

```
181         while (it.hasNext() == true) {
182             Joueur j2 = it.next();
183             fich.write(j2.getNom() + "/" + j2.getScore());
184             fich.write(System.lineSeparator());
185         }
186         fich.close();
187     }
188
189     public void trierScores(JLabel tabLabel[][]) {
190         int l = 0;
191         try {
192             this.lireJoueur();
193         } catch (FileNotFoundException ex) {
194         } catch (IOException ex) {
195         }
196
197         Collections.sort(listeScore, new JoueurComparator());
198         Iterator<Joueur> it = listeScore.iterator();
199         while (it.hasNext() == true && l < 10) {
200             Joueur j2 = it.next();
201             tabLabel[l + 1][1].setText(j2.getNom());
202             tabLabel[l + 1][2].setText("" + j2.getScore());
203             l++;
204         }
205     }
206
207     public void incremNbrVagues() {
208         nbrVagues++;
209     }
210 }
211
```

5.10. Classe JoueurComparator

```
1  /*
2   EPF 2A
3   Auteurs : Nicolas Breton et Cecile Tran
4   Zombie shooter
5   Groupe A
6   Date : 02/02/15
7   Edit : 15/05/15
8   */
9  package bretontran_projetcpo;
10
11 import java.util.Comparator;
12
13 public class JoueurComparator implements Comparator<Joueur> {
14
15     @Override
16     public int compare(Joueur a, Joueur b) {
17         return -(a.getScore() - b.getScore());
18     }
19 }
20
```

6. Conclusion

Ce projet qui avait pour objectif d'appréhender la programmation en Java avec une interface graphique nous a confrontés à de nombreuses problématiques liées à ce travail. En effet, nous avons dû faire face à de multiples erreurs issues de la manipulation d'images, de boucles qui amènent l'itérateur hors des listes … Néanmoins, cela nous a permis d'apprendre à débuguer un programme, de réfléchir à la manière dont nous voulions coder par exemple le déplacement des zombies ou l'attaque des tours. Bien entendu, notre programme aurait pu voir de nombreuses possibilités supplémentaires d'actions de jeu ajoutées telles que l'apparition spontanée d'obstacles, l'évolution du chemin à chaque vague … Ce jeu offre une diversité de développement infinie et à ce stade du travail, notre imagination nous a amené au programme écrit précédemment dans le rapport.

De plus, ce projet se réalisait en binôme, ce qui amène à faire des choix dans les règles du jeu, dans la logique de programmation et à s'entraider face à des obstacles liés au code. Cette expérience nous a donc apporté bien plus que la découverte de la programmation orientée objet ; elle nous a permis de travailler à plusieurs sur ce que nous pourrions appeler un exercice de logique avec son propre langage.