Clovis Rollet
Joshua Lozano
TD 4

Rapport projet C

1)Répartition du travail

Les parties dont le titre est de couleur rouge ont été traité par Joshua et celle en bleu par Clovis. J'ai (Clovis) commencé par écrire la fonction permettant de récupérer les parties et d'initialiser la structure Jeu. Pendant ce temps, Joshua s'est occupé de la simulation des tours et de l'interaction avec l'utilisateur pour placer les tourelles.
Par la suite, j'ai codé les fonctions pour sauvegarder les scores, puis pour charger et sauvegarder les parties. Finalement, Joshua et moi avons créé les autres types de tourelles et de zombies, et le scénario.

2) Scénario

Au départ, nous avions pensé à un scénario sur le thème du football. Nous avons changé en cours de route en s'inspirant de jeux de science-fiction auxquels nous avons joué notamment "Fallout 4".

3) Récupération des parties

- Choix:
  Un de mes premiers choix a été d'ouvrir directement les parties dans mon code sans utiliser la commande "./a.out partie.txt". J'ai trouvé que cela nuit grandement au gameplay de l'utilisateur. J'ai préféré questionner l'utilisateur sur quelles parties il souhaitait jouer.
  Un second choix a été de considérer que dans les parties, le tour maximum auquel un zombie peut apparaître est le tour 15. En effet, dans le cas contraire la visualisation des vagues est incompréhensible car il n'y a que 15 positions sur le jeu.
  Un troisième choix a été d'imposer que les zombies soient chaînés avec les champs next_line et prev_line de droite à gauche. Je m'explique : s'il y a deux zombies sur la même ligne, celui le plus à gauche est le next_line de celui à droite et ce dernier est le prev_line de celui à gauche. Cela ne m'est pas venu à l'esprit de prévenir tout de suite Joshua de mon choix. Alors que j'avais achevé cette fonction, je lui ai annoncé. De manière opposée, il a chaîné de gauche à droite. Heureusement, il a été simple de modifier cela car nous n'étions qu'au début de notre code. C'est à ce moment-là que nous avons réalisé l'importance de communiquer ce qui nous paraît logique car ce n'est pas forcément le cas pour l'autre. Ce projet a été le premier grand travail de groupe que j'ai fait. Il m'a fait réaliser la difficulté de travailler à plusieurs

(surtout sur un code) mais il m'a aussi permis de comprendre que l'union fait la force car je n'aurai jamais eu les capacités d'achever ce projet seul. En somme, ce fut une très bonne expérience.
Mon dernier choix a été de considérer que dans les fichiers représentant les parties, les tours auxquels les zombies apparaissent soient triés dans l'ordre croissant, que la cagnotte soit minimum de 50 (prix de la tourelle la moins cher) et enfin qu'il n'y ait pas de partie sans zombie. Sinon le code indique que la partie lue n'est pas valide.

- DIfficultés/bugs:
  Une première difficulté a été de chaîner avec les champs next_line et prev_line les zombies et d'afficher les vagues de zombies. Ma solution (assez logique) a été d'utiliser un double tableau de pointeurs sur la structure Etudiant. Un tel tableau était difficile à initialiser car j'ai du déclarer un pointeur TAB de type Etudiant***, ce que je n'avais jamais fait auparavant.
  Une difficulté découlant de la création de ce tableau a été la libération de la mémoire allouée à celui-ci. Je l'avais omise au départ mais grâce à valgrind, j'ai ensuite réalisé mon erreur.

- Fonction:
  creation_jeu:
  Cette fonction demande à l'utilisateur quelles parties il souhaite jouer ou s'il souhaite charger une partie sauvegardée. Si l'utilisateur fait ce dernier choix, un appel à la fonction charger (expliquée en détail plus loin en-dessous) permet de renvoyer le bon pointeur sur la structure Jeu.
  Si on est dans un des autres cas, la fonction lit le fichier texte représentant la partie associée.
  Elle récupère la cagnotte, la liste des zombies et la chaîne avec les champs next, next_line et prev_line. Elle ajoute aux bonnes coordonnées le zombie de la ligne du fichier lue dans le tableau. Et cela grâce à une variable retient_tour qui permet de repérer lorsqu'on lit un zombie qui va apparaître à un tour différent de tous les précédents. On décale donc en changeant la position de tous ces zombies grâce au tableau.
  Cette méthode de faire n'est clairement pas la meilleure, j'en ai pris conscience que plus tard. Par soucis de temps, j'ai préféré développer d'autres aspects du jeu plutôt que d'améliorer cette partie.
  A la fin de la boucle qui lit le fichier texte, le tableau est rempli avec tous les zombies à leur correcte ligne et position. Je me suis donc servi de ce dernier pour les chaîner avec next_line et prev_line.
  Pour finir d'initialiser le jeu, la fonction s'occupe du champ utilisateur (nous verrons les raisons de sa création plus tard) en lisant le pseudo de l'utilisateur.

On affiche à l'intention de ce dernier sa cagnotte disponible et les vagues de zombie à venir.

La complexité de cette fonction est en theta de n, où n est le nombre de lignes dans le fichier partie. En effet, il y a une boucle qui permet de lire chaque ligne du fichier et elle s'arrête lorsqu'on atteint la fin.

4) Simulation des tours

4.1) Fonctions annexes

Pour pouvoir initialiser les tourelles (les zombies ont été initialisés par Clovis) j'ai utilisé une fonction annexe "placer_tourelles(Jeu *J, int **tab)" qui permet de créer des maillons et de les lier pour la liste chaînée J->tourelles. La partie la plus compliquée de ces fonctions a été de déterminer pourquoi je pouvais les utiliser. Évidemment pour placer les tourelles mais pour quoi d'autre? Je me suis rendu compte plus tard que je devais aussi l'utiliser pour afficher les emplacements des tourelles lors des tours en remplissant progressivement le tableau qui sera utilisé pour l'affichage. J'ai dû modifier ma fonction afin d'y intégrer un tableau à deux dimensions d'entiers pour pouvoir procéder à l'affichage.

J'ai choisi de le faire comme ça car cela m'évitait de faire une autre fonction pour remplir le tableau et que cela marchait bien mis à part le fait que la fonction soit très redondante et longue.

L'affichage ne fut pas facile au premier abord, j'avais d'abord opté pour une méthode peu othodoxe en listant tous les types de tourelles et si tab[i][j] était différent de 0, je regardais ensuite à quoi correspondait la tourelle avec une série de tests. Après de nombreuses exécutions j'ai pu me rendre compte que l'efficacité de cette fonction se révélait être similaire à la maîtrise de la mémoire par python. Je l'ai modifiée et utilisée avec (char) J->..... pour afficher les bons types ainsi que les points de vie associés à la tourelle/zombie.

Cependant, un problème demeurait, l'alignement des lignes n'était pas bon. En parlant avec différentes personnes de ce problème, j'ai pu réadapter ma fonction en laissant uniquement deux espaces aux zombies ainsi qu'aux tourelles à la place de trois car leur vie est aussi affichée. J'ai ensuite rempli mon tableau de "." à la place de "0" pour faciliter la création de la fonction.

Un autre problème est apparu lorsque les tourelles mouraient, elles ne disparaissaient pas, pour y remédier j'ai réinitialisé le tableau à chaque tour et reremplit avec les tourelles actualisées.

L'actualisation des positions des zombies a aussi été problématique au premier abord. Dans mon programme, les zombies étant en face de tourelles

avancent forcément à moins d'être bloqué par une tourelle ou par un autre zombie. J'ai rajouté use dans la structure des zombies afin de faire la distinction entre les zombies déjà avancés et le reste. La fonction "reinit_use" pour pouvoir remettre les use de chaque zombie à 0 à la fin de chaque tour.

J'ai fait ce choix pour avancer les zombies car j'avais procédé naturellement comme ça dès le début, ça me paraissait logique, mais en écrivant ce rapport je me rend compte que j'aurai pu simplement faire avancer tous les zombies en même temps avec la fonction avance et que ce premier choix n'était pas le plus judicieux. De nombreux bugs ont eu lieu avec l'athlète : il ne détectait pas les tourelles devant lui et allait sur les positions et ne sautait pas au-dessus des zombies. Nous avons donc listé une par une toutes les possibilités et modifié la ligne `current->use=1;` dans "tour_gen" qui empêchait l'athlète de sauter au-dessus des tourelles. Nous avons mis à la place une condition qui laisse le use de l'athlète à 0.

Deux autres fonctions annexes ont été créées pour pouvoir gérer la suppression des zombies "suppr_tourelles" et "suppr_etudiant". Elles parcourent la liste chaînée des etudiants/tourelles et supprime tous les étudiants/tourelles dont la vie est inférieur ou égale à 0.

La fonction "suppr_etudiant" a été plus compliquée que "suppr_tourelles" car le chaînage de prev_line et next_line n'était pas facile à assimiler et à chaîner quand un zombie mourait. Je l'avais mal fait, cela a créé plusieurs bugs (je ne me rappelle pas exactement la nature) et Clovis m'a aidé à les identifier ainsi qu'à les régler.

J'ai choisi de faire ces fonctions comme ça car le fait qu'elle supprime toutes les occurrences des fonctions qui sont mortes est très utile.

Enfin, "Win" (gagner) et "Loose" (perdre) les conditions d'arrêt de ma fonction "tour_gen" si la liste chaînées des étudiants est vide on gagne et si étudiant atteint une position inférieure ou égale à 0 on perd.

4.2) Fonction principale

Pour aborder la simulation des tours, je l'avais initialement divisé en deux parties : le premier tour qui devait se charger de l'initialisation du placement des tourelles ainsi que de la position des zombies qui était différente de celle que j'avais imaginé car nous avions dû les modifier pour l'affichage des vagues. Je pensais aussi que le déroulement du premier tour serait différent des autres. Puis une fois la fonction finie et celle des autres tours commencée je me suis rendu compte que le déroulement était le même et j'ai donc regroupé les fonctions en une seule. Avant de commencer les tours, on procède à toutes les initialisations nécessaires.

Elle s'occupe de l'initialisation, création de J->tourelles. Ainsi que de simuler les tours et de déterminer si la partie est gagnée ou perdue. La fonction a été modifiée et améliorée de multiples fois. Les tours fonctionnent avec des boucles while. La première boucle s'arrête si "win" ou "loose" est vrai, la deuxième s'arrête si

on arrive à la fin de la liste chaînée des tourelles et enfin la dernière si on arrive à la fin de liste chaînée des étudiants. J'ai commencé par distinguer deux manières de contact entre les zombies et les tourelles. Lorsque la tour tire sur le zombie et le zombie ne lui inflige pas de dégâts car il est trop loin (exemple: T . . Z) et le cas où la tourelle et le zombie s'infligent mutuellement des dégâts (exemple: T Z). Cela m'a paru logique et surtout bien plus facile à utiliser. Cela permet d'identifier clairement où sont les bugs (si un zombie ne met pas de dégâts ou ne met trop, pareil pour les tourelles). Pour la création de la mine cela a été d'une grande aide car il a suffit d'une condition pour qu'elle soit opérationnelle. J'ai aussi utilisé des listes chaînées pour parcourir J->etudiants et J->tourelles pour être sûr de parcourir tous les cas possibles et pour respecter la consigne "les tourelles doivent tirer par ordre de chaînage". A chaque tourelle on parcourt la liste chaînée des zombies jusqu'à trouver un zombie, si on ne trouve pas de zombie, on passe à la tourelle suivante (s'il y en a une) et la liste chaînée des zombies est remise au début…

Le premier problème survenu a été une erreur d'inattention ainsi que de maîtrise de liste chaînée current (resp. T) qui parcourt la liste chaînée en modifiant (si modification il y a) la liste chaînée principale. En voulant supprimer les zombies et les tourelles mortes, je mettais la liste current (resp. T) en argument de ma fonction "suppr_etudiant/tourelle", ce qui ne supprimait pas vraiment les zombies et donc créait des problèmes d'allocation mémoire ainsi que certaines fois de déplacement des zombies. Un autre bug auquel nous avons fait face est d'avoir plusieurs tourelles sur la même ligne. Deux problèmes ont émergé de ce placement de tourelles :

- le double avancement des zombies.
- prendre en compte la tourelle la plus proche des zombies pour empêcher un conflit de position et de déplacement. En effet, si l'on plaçait la première tourelle en position 1 et la deuxième sur la position 14, le zombie avançait une première fois sans prendre en compte la tourelle en position 14 et une deuxième fois ensuite dépassant la tourelle.

Pour pallier ces deux problèmes, j'ai défini une nouvelle fonction prenant en argument la liste chaînée, et la ligne de la tourelle qui est en action et vérifie s'il n'y a pas une autre tourelle devant qui empêche le déplacement du zombie (fonction : "pos_max_T").
Un autre problème assez similaire est survenu avec le ralentisseur, si on plaçait une tourelle de base puis un ralentisseur sur la même ligne, la tourelle de base mettait des dégâts au zombie, le faisait avancer sans prendre en compte le ralentisseur, (si le ralentisseur est actif pendant le tour, le zombie n'avance pas). J'ai créé une fonction "ralentir_sur_la_ligne" qui permet d'éviter cela en renvoyant si le ralentisseur est actif ou non.

On peut remarquer la présence dans les if et else imbriqués dans les while de conditions telles :

```
if (T->next) {
```

```
   T=T->next;//si il y a une autre tourelle on continue sinon
on arrete ici la boucle while
}
else {
   break;
}
```

Car je pense que la condition d'arrêt nécessaire pour la boucle while aurait été while(T->next){ … }, or cela m'était moins instinctif j'ai préféré le faire comme ça. D'autres paramètres sont rentrés en compte, je ne pouvais pas tester le programme sur mon ordinateur, je ne sais pas pourquoi, le programme ne trouvait pas le fichier partie.txt sur mon ordinateur. Cela nous a fait perdre énormément de temps à moi ainsi qu'à Clovis car je lui demandais souvent de me prêter son ordinateur pour tester mes fonctions, corriger des bugs et quand clovis n'était pas là ou que je travaillais de chez moi, je ne pouvais pas tester mes fonctions. J'ai créé une liste chaînée de zombie manuellement qui simulait le premier tour, mais cela ne représentait qu'une infime partie des problèmes que nous pouvions rencontrer lors d'une partie complète. Quand une fonction marchait bien, je n'avais pas le réflexe de "l'optimiser", car cela aurait pu être une source de bugs et comme je ne pouvais pas tester le programme sur partie entière, j'avais peur d'avancer sur une fonction faussée.

5) Système des scores

- Choix:
  J'ai choisi de mettre en place un système d'étoiles que l'utilisateur reçoit à la fin de sa partie en m'inspirant de jeux comme "Angry Birds". Le nombre d'étoiles est calculé en fonction de la proportion de la cagnotte initiale qui a été dépensée, et non en fonction du temps mis pour éliminer les zombies, après tout dans l'Apocalypse, on en a du temps. Plus sérieusement, je trouve que c'est beaucoup plus intéressant que l'utilisateur cherche à dépenser le moins possible car cela l'oblige à réfléchir à des stratégies et à visualiser sa défense. Un désavantage de ce choix a été de trouver les proportions justes. Un deuxième choix a été de créer la structure Utilisateur afin de stocker son pseudo, son score et son nombre d'étoiles et de pouvoir les inscrire simplement dans le fichier meilleurs_scores.txt.

- Difficultés/bugs:
  J'ai eu pas mal de soucis avec la fonction meilleurs_scores. En premier, je n'avais pas réfléchi au cas où l'utilisateur a 0 étoile. Puisqu'il n'y avait rien écrit après le score sur la ligne de cet utilisateur, la fonction fscanf ne marchait pas bien et créait un fichier meilleurs_scores.txt avec 600 lignes, un

bug assez troublant. J'ai donc décidé d'inscire le caractère "_" pour indiquer un nombre d'étoiles égale à 0.
Un deuxième gros problème a été que je lisais au départ le pseudo de l'utilisateur avec la fonction scanf, autorisant ce dernier à inscrire un chiffre en première place de son pseudo. Cependant, cela entraînait une mauvaise lecture de la fonction fscanf (encore elle …) et un bug de mon programme. J'ai donc opté pour la fonction fgets, plus difficile à comprendre mais résolvant mon problème.

- Fonction:
  meilleurs_scores:
  Cette fonction sert à inscrire dans le fichier meilleurs_scores.txt les 10 meilleurs scores triés par ordre décroissant. Elle commence donc par ouvrir ce fichier. Si le fichier est vide, elle inscrit le pseudo, le score et le nombre d'étoiles de l'utilisateur courant dans le fichier puis le ferme.
  S'il y a déjà des scores inscrits, la fonction lit ces scores au moyen d'un tableau de pointeur sur la structure utilisateur. Elle regarde ensuite si le score de l'utilisateur courant est parmi les 10 meilleurs en prenant compte de ces scores sauvegardés. Si ce n'est pas le cas, on ne touche pas au fichier assez logiquement. Sinon, on insère le nouveau score au bon emplacement en décalant toutes les lignes si nécessaire.

  La complexité de meilleurs_scores est en theta de 1. En effet, on lit un fichier dont on sait qu'il comporte au maximum 10 lignes.

6) Système de sauvegarde

- Choix:
  Le premier choix a été de décider pour pouvoir laisser l'utilisateur sauvegarder de lui demander entre chaque tour s' il le souhaitait.
  Un gros désavantage que je pressentais était une dégradation du gameplay. Cependant, je ne trouvais pas d'autres solutions. J'ai donc laissé cette implémentation. Néanmoins, je la trouve désormais assez utile car elle permet à l'utilisateur de prendre le temps de voir le tout en cours, les actions des zombies et des tourelles et de visualiser ce qu'il va se passer ensuite.
  Un deuxième choix a été de sauvegarder pour la liste chaînée des zombies, toutes leurs caractéristiques comme par exemple leur dégâts. On pourrait penser que c'est inutile puisqu'en connaissant le type du zombie, on a ses dégâts. Cependant, la création du zombie orateur déjoue ce plan. Ce choix est donc loin d'être inutile.

- Difficultés/bugs:
- Son implémentation m'a paru assez simple par rapport à d'autres fonctions.

Je n'ai pas vraiment eu de bugs avec la fonction sauvegarde mais plutôt avec sa fonction paire charger (cf ci-dessous).

- Fonction:
  sauvegarde:
  Cette fonction inscrit dans un fichier texte l'état d'une partie en cours.
  En première ligne, on trouve la cagnotte initiale (que l'on sauvegarde car si on charge la partie on a besoin de cette valeur pour calculer la proportion de l'argent dépensé par l'utilisateur). En deuxième ligne, le numéro du tour en cours et la cagnotte. Cette dernière valeur ne sert à rien mais je pensais au départ que nous allions implémenter une version du jeu où l'on peut rajouter des tourelles pas seulement au début. En troisième ligne, il y a le pseudo et le score de l'utilisateur courant. En quatrième ligne, la liste chaînée des zombies. En cinquième ligne, "---" ,nous allons voir ci-dessous pourquoi. Enfin en dernière ligne, la liste chaînée des tourelles.

  Sa complexité est en theta de n + m où n est le nombre de zombies et m le nombres de tourelles car on parcourt toute la liste chaînée des zombies et aussi celle des tourelles

7)Système de charge

- Choix:
  Les choix sont reliés à ceux pour le système de sauvegarde.

- Difficultés/bugs:
  Sans la cinquième ligne "---", mon programme buguait car la fonction fscanf ne s'arrêtait pas entre la ligne avec la liste chaînée des étudiants et la ligne avec la liste chaînée des tourelles. J'ai du rajouté cette ligne pour contrer ce problème.
  Difficulté à chaîner avec prev_line et next_line mais grâce à l'information du tour d'apparition du zombie et de sa ligne, j'ai réussi.

- Fonctions:
  charger:
  Cette fonction est la fonction paire de la fonction sauvegarde. Elle lit les lignes du fichier sauvegarde.txt pour créer le jeu. On peut presque dire que c'est la fonction contraire à sauvegarde. Pour

  Sa complexité est aussi en theta de n+m.

8) Entente du groupe/informations diverses

Le nom de la partie porte à confusion, mais nous tenions à la faire pour parler des quelques quiproquos, différences de compréhension du sujet, qui ont pu amener à certains choix dans les positions des zombies/tourelles, des chaînages next_line, prev_line… Pour moi (joshua), il m'était plus logique que les zombies commencent en position 15 et pour Clovis c'était l'inverse, mais comme la manipulation des placements était plus abordée dans la simulation des tours, Clovis a fait le placement en fonction de comme je l'imaginais pour me faciliter la tâche. Mis à part ça, l'entente était au top. Nous n'avons pas fait d'interface graphique car nous manquions de temps et que nous avions de nombreux bugs dans le programme. Nous avons préféré se concentrer sur la correction de ces derniers plutôt que de faire une interface graphique et de risquer de rajouter d'autres problèmes. De plus, nous avons décidé d'un commun accord de ne pas faire de headers. Bien que la majorité des personnes en fassent, nous avons préféré faire un seul code avec tout le code et des gros titres pour séparer les parties et les différents (cf. code) était mieux, plus facile pour se l'envoyer et plus lisible.