Invasion MIDO*

Projet Programmation C, L2 MIDO 2024-2025

Résumé

Le but de ce projet est de développer un petit jeu de *Tower Defense* dont les combats seront discrétisés en tour par tour. C'est une version simplifiée et librement inspirée du jeu *Plants versus Zombies*. [1]

1 Dates importantes

- ▷ Deadline composition des groupes (Excel dans le Teams) : 22 décembre 2024 à 23h59.
- ▷ Rendu du projet : 4 février 2025 à 23h (sur Moodle).
- ▷ Soutenances : à définir.

2 Versions

Ce sujet est susceptible d'être modifié.

- ▷ 9 décembre 2024 : première version du projet.

3 Description

3.1 Déroulé et but

On cherche à créer un jeu à un seul joueur qui se joue contre la machine. Durant chaque partie, une somme de crédits (argent) est donnée au joueur qui devra acheter et mettre en place des tourelles de défense (mises en place par les profs) pour arrêter une vague de vilains étudiants ennemis souhaitant envahir l'université.

Sur l'exemple de plateau de jeu ci-dessous, on voit 7 lignes (chemins) menant à l'université. Les points verts sont les lieux du plateau de jeu sur lesquels le joueur pourrait placer des tourelles de défense mais aussi les points de passage des vilains ennemis sur leur chemin pour atteindre l'université.

Les vilains ennemis arrivent tour par tour en suivant une ligne depuis le bord opposé à l'université. Le but des vilains ennemis est de traverser le plateau de jeu pour atteindre et infester l'université, le but du joueur est de mettre en place des défenses qui arrêteront la vague d'invasion.

Par défaut, les tourelles sont statiques, implantées par le joueur en début de partie et bloquent les ennemis tant qu'elles ne sont pas détruites. Les vilains ennemis se déplacent avec une vitesse définie par leur type.

*http://fr.alphahistory.com/guerre-froide/invasion-usa-XNUMX/

1. http://fr.wikipedia.org/wiki/Plantes_contre_zombies.
Jouable en ligne à http://www.miniplay.com/game/plants-vs-zombies

3.2 Début de la partie

Une partie est en fait encodée par un fichier texte qui contient les informations de la vague d'invasion ainsi que l'argent disponible pour que le joueur puisse construire ses dispositifs de défense. Voici un exemple de fichier contenant uniquement des vilains ennemis de type Z (type de base) :

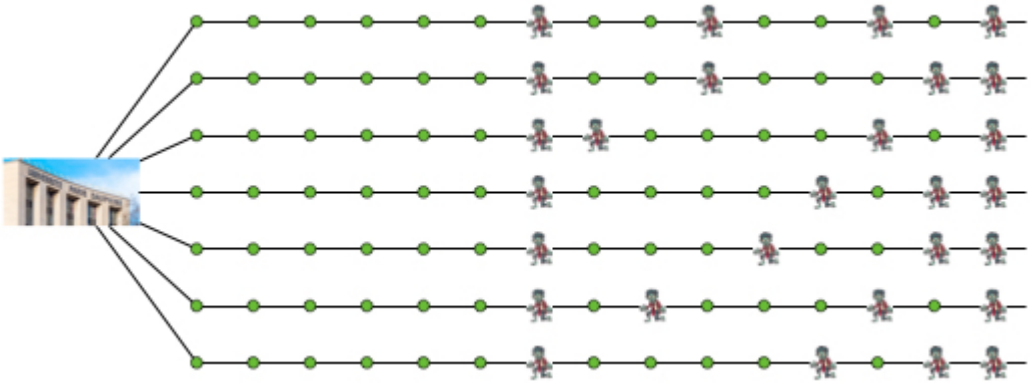
```
700
1 1 Z
1 2 Z
1 3 Z
1 4 Z
1 5 Z
1 6 Z
1 7 Z
2 3 Z
3 6 Z
4 1 Z
4 2 Z
5 5 Z
6 4 Z
6 7 Z
7 6 Z
7 1 Z
7 3 Z
8 2 Z
8 7 Z
8 5 Z
8 4 Z
9 1 Z
9 2 Z
9 3 Z
9 4 Z
9 5 Z
9 6 Z
9 7 Z
```

Ce fichier commence par un entier seul sur une ligne : c'est l'argent qui sera à disposition du joueur pour construire ses dispositifs de défense. Ensuite, toutes les lignes suivent le formatage suivant :

`[numéro du tour] [numéro de la ligne] [type de vilain ennemi]`

Par exemple, les sept lignes suivant l'argent à disposition pour cet exemple de partie montrent qu'au premier tour, il devra apparaître un vilain ennemi de type Z sur chacune des lignes. De même, les 7 dernières lignes montrent que la vague d'invasion se termine par un ennemi de type Z encore une fois sur chacune des lignes mais au tour numéro 9.

L'image ci-dessous visualise la vague d'invasion.



On peut aussi la visualiser en "ascii art" :

```
Vagues :
1|   Z   .   .   Z   .   .   Z   .   Z
2|   Z   .   .   Z   .   .   .   Z   Z
3|   Z   Z   .   .   .   .   Z   .   Z
4|   Z   .   .   .   .   Z   .   Z   Z
5|   Z   .   .   .   Z   .   .   Z   Z
6|   Z   .   Z   .   .   .   Z   .   Z
7|   Z   .   .   .   .   Z   .   Z   Z
```

Si au moment où un vilain ennemi devrait apparaître sur sa ligne, sa case d'arrivée est occupée, il doit patienter et n'apparaitra qu'au tour suivant.

Une fois que le joueur connaît son argent disponible et a pu visualiser les vagues d'ennemis, **avant le début des tours**, le joueur place les tourelles sur le plateau (il choisit le type et la position des tourelles) dans la limite de son argent disponible.

3.3 Déroulé d'un tour

Le premier tour sera le tour 1. Pour chaque tour les actions devront être résolues dans un ordre précis :

- ▷ entrée sur le plateau, en position opposée à l'université, des vilains ennemis apparaissant au tour courant ;
- ▷ résolution des actions des tourelles de défense selon leur chaînage `next` ;
- ▷ résolution des actions des ennemis arrivant au contact des tourelles selon leur chaînage `next` ;
- ▷ déplacement des ennemis selon leur chaînage `next`, leur ligne ainsi que leur vitesse ;
- ▷ si un ennemi atteint l'université, la partie est perdue ;
- ▷ s'il n'y a plus d'ennemi dans la liste chaînée, alors la partie est gagnée.

À la suite de ces actions, il faut incrémenter le compteur de tour pour amorcer le tour suivant.

Note : les ennemis ne peuvent pas se doubler sur les lignes. Quand le premier ennemi d'une ligne a une toute petite vitesse, il ralentit tous ceux qui sont derrière lui. Cela peut offrir des stratégies pour le joueur comme pour les méchants.

3.4 Éléments du jeu

On pourra considérer que dans la version basique du jeu il y a 7 lignes avec 15 positions chacune. Le but est aussi de laisser libre court à son imagination et essayer de développer un jeu jouable et intéressant. Pour que le jeu soit fun, il est bien d'avoir testé des jeux de *tower défense* durant son temps libre. Parmi les défenses, voici ce qui ressort souvent de tels jeux :

- ▷ une tourelle de base abordable, faisant de petits dégâts uniquement sur sa ligne ;
- ▷ une tourelle qui ralentit des vilains ennemis (elle fait passer à 1 la vitesse des cibles touchées) ;
- ▷ une tourelle de type mine, abordable, qui explose au contact en tuant ainsi seulement le premier vilain ennemi ;
- ▷ une tourelle extrêmement chère faisant de lourds dommages de zone sur la ligne et les deux lignes voisines ;
- ▷ une tourelle de type mur de défense, n'attaquant pas, mais avec un grand nombre de points de vie pour ralentir les ennemis...

Une telle diversité peut déjà favoriser des niveaux intéressants et des stratégies différentes.

De même, pour générer de l'enjeu, les vilains ennemis doivent aussi avoir des caractéristiques demandant au joueur défenseur de s'adapter. Par exemple :

- ▷ un vilain de base pas trop solide et pas trop rapide ;
- ▷ un vilain très solide mais très lent ;
- ▷ un vilain très rapide mais de résistance raisonnable ;
- ▷ un vilain docteur qui soigne devant et derrière lui ;
- ▷ un vilain qui change de ligne une fois dans la partie en sautant aléatoirement sur une des deux lignes adjacentes ;
- ▷ un vilain qui accélère le premier de sa ligne ainsi que les deux premiers ennemis des autre lignes voisines à la sienne...

Ce sont ici des suggestions. Ce jeu est transportable dans plein d'univers différents pourvu que le game play soit bien pensé et équilibré. La limite est votre imagination (vous avez le droit d'aller dans l'humour pour le nom des défenses et des ennemis).

3.5 Structures

Voici les structures de données initiales importantes et imposées du sujet (il est possible d'y ajouter des éléments).

```
typedef struct tourelle {
    int type;
    int pointsDeVie;
    int ligne;
    int position;
    int prix;
    struct tourelle* next;
```

```
} Tourelle;

typedef struct etudiant {
    int type;
    int pointsDeVie;
    int ligne;
    int position;
    int vitesse;
    int tour;
    struct etudiant* next;
    struct etudiant* next_line;
    struct etudiant* prev_line;
} Etudiant;

typedef struct {
    Tourelle* tourelles;
    Etudiant* etudiants;
    int cagnotte;
    int tour;
} Jeu;
```

La structure `Tourelle` modélise les tourelles de défense. Ces dernières ont un type, un certain nombre de points de vie, sont placées sur une ligne et à une position données. Elles coûtent une certaine valeur et sont chaînées les unes aux autres via le champ `next`. En fait le chaînage sera fait par ordre de construction des tourelles (qui seront insérées à la fin lors des constructions). Au début, les tourelles forment la liste vide (un pointeur `NULL`) puis l'utilisateur insère une première tourelle, puis une seconde qui se place à la suite et ainsi de suite. Ce chaînage est très important car les tourelles passeront à l'action par ordre de ce chaînage. Certaines des tourelles tirent et font des effets et des dégâts aux vilains ennemis. La première tourelle à tirer est en tête de liste puis la seconde et ainsi de suite (le gameplay est donc impacté par ce choix).

La structure `Etudiant` modélise les vilains ennemis. Ces derniers ont un type, un certain nombre de points de vie, sont placés sur une ligne et à une position données. Ils ont une vitesse de déplacement et apparaissent à un certain tour. Les vilains ennemis sont chaînés par ordre d'apparition dans le jeu via le champ `next` mais ils sont de plus doublement chaînés par ligne (champs `next_line` et `prev_line`). On peut ainsi rapidement connaître celui qui précède et celui qui suit sur la même ligne. Ces trois pointeurs devront toujours être à jour durant l'exécution du programme, à chaque insertion mais aussi à chaque suppression.

Enfin, la structure `Jeu` définit une partie : une liste chaînée d'étudiants, une liste chaînée de tourelles ainsi qu'un montant d'argent et une variable pour connaître le numéro du tour courant.

3.6 Interface

Une première version du jeu pourra être affichée dans le terminal grâce aux caractères d'échappement ASCII. [2] Par exemple, le code suivant efface tout le terminal :

```
    printf("\033[0;0H");  // on se place en haut à gauche
    printf("\033[2J");    // on efface tout
```

2. Le terminal Unix ou MacOS accepte nativement ces codes. Pour les utiliser sous Windows, vous pouvez ajouter quelques lignes de code.
Voir https://solarianprogrammer.com/2019/04/08/c-programming-ansi-escape-codes-windows-macos-linux-terminals/ ou https://superuser.com/questions/413073/windows-console-with-ansi-colors-handling

Une fois les emplacements des tourelles saisis par l'utilisateur avant le début des tours, on pourrait avoir ce genre d'affichage qui visualise les tourelles (T), les vilains ennemis ainsi que les points de vie de ces derniers.

```
Tour 1
1|   T   .   .   .   .   .   .   .   .   .   .   .   .   .   .   3Z
2|   T   .   .   .   .   .   .   .   .   .   .   .   .   .   .   3Z
3|   T   .   .   .   .   .   .   .   .   .   .   .   .   .   .   3Z
4|   .   .   .   .   .   .   .   .   .   .   .   .   .   .   .   3Z
5|   .   .   .   .   .   .   .   .   .   .   .   .   .   .   .   3Z
6|   .   .   .   .   .   .   .   .   .   .   .   .   .   .   .   3Z
7|   .   .   .   .   .   .   .   .   .   .   .   .   .   .   .   3Z
```

```
Tour 2
1|   T   .   .   .   .   .   .   .   .   .   .   .   .   2Z
2|   T   .   .   .   .   .   .   .   .   .   .   .   .   2Z
3|   T   .   .   .   .   .   .   .   .   .   .   .   .   2Z  3Z
4|   .   .   .   .   .   .   .   .   .   .   .   .   .   3Z
5|   .   .   .   .   .   .   .   .   .   .   .   .   .   3Z
6|   .   .   .   .   .   .   .   .   .   .   .   .   .   3Z
7|   .   .   .   .   .   .   .   .   .   .   .   .   .   3Z
```

```
Tour 3
1|   T   .   .   .   .   .   .   .   .   .   .   .   1Z
2|   T   .   .   .   .   .   .   .   .   .   .   .   1Z
3|   T   .   .   .   .   .   .   .   .   .   .   .   1Z  3Z
4|   .   .   .   .   .   .   .   .   .   .   .   .   3Z
5|   .   .   .   .   .   .   .   .   .   .   .   .   3Z
6|   .   .   .   .   .   .   .   .   .   .   .   .   3Z  .   3Z
7|   .   .   .   .   .   .   .   .   .   .   .   .   3Z
```

```
Tour 4
1|   T   .   .   .   .   .   .   .   .   .   .   .   .   .   3Z
2|   T   .   .   .   .   .   .   .   .   .   .   .   .   .   3Z
3|   T   .   .   .   .   .   .   .   .   .   .   .   3Z
4|   .   .   .   .   .   .   .   .   .   .   .   .   3Z
5|   .   .   .   .   .   .   .   .   .   .   .   .   3Z
6|   .   .   .   .   .   .   .   .   .   .   .   .   3Z  .   3Z
7|   .   .   .   .   .   .   .   .   .   .   .   .   3Z
```

```
Tour 5
1|   T   .   .   .   .   .   .   .   .   .   .   .   .   2Z
2|   T   .   .   .   .   .   .   .   .   .   .   .   .   2Z
3|   T   .   .   .   .   .   .   .   .   .   .   .   2Z
4|   .   .   .   .   .   .   .   .   .   .   .   3Z
5|   .   .   .   .   .   .   .   .   .   .   .   3Z  .   .   .   3Z
6|   .   .   .   .   .   .   .   .   .   .   .   3Z  .   3Z
7|   .   .   .   .   .   .   .   .   .   .   .   3Z
```

En mode console, il faudra arriver à bien gérer l'alignement vertical. Par exemple, pour les points de vie et le type du méchant/tourelle, on peut utiliser ce code :

```
printf(" %2d%c", ...);
```

qui force l'affichage de l'entier sur 2 caractères (le 2 entre le % et le d).

Pour l'affichage uniquement (des vagues de méchant ou d'un tour), vous avez le droit de créer un tableau temporaire à deux dimensions.

4 Niveaux de difficulté

4.1 Invasion 50's

- ▷ Affichage du jeu dans le terminal (ASCII art)
- ▷ Lecture des fichiers standardisés et chargement des niveaux
- ▷ Menu et interaction clavier pour laisser le joueur construire ses défenses
- ▷ Un ennemi de base (appelé type Z (comme l'étudiant qui dort zzzz en cours)) qui avance d'une case par tour de jeu avec 5 PV et fait 1 de dégât au contact d'une tourelle par tour.
- ▷ Une tourelle de base, avec 3 PV, qui tire et fait 1 de dégât par tour uniquement sur le premier ennemi de la ligne où elle est posée.
- ▷ Visualisation en ASCII art des vagues de méchants
- ▷ Visualisation de la partie à vitesse raisonnable
- ▷ Allocation et libération de la mémoire parfaite (vérifiée avec `valgrind`)

4.2 Invasion 70's

- ▷ Tout Invasion 50's
- ▷ Au moins 5 types de tourelles et d'ennemis avec des capacités « rigolotes »
- ▷ Au moins 5 niveaux de jeu en fichiers « rigolos » à jouer

4.3 Invasion 90's

- ▷ Tout Invasion 70's
- ▷ Système de score affiché (On gagne d'autant plus de points qu'on tue les vilains ennemis rapidement mais on perd des points à dépenser trop d'argent)
- ▷ Sauvegarde/chargement d'une partie (on stocke dans un fichier texte l'état du jeu)
- ▷ Système de meilleurs scores : on sauvegarde dans un fichier texte les 10 meilleurs scores réalisés avec le nom de l'auteur, stockés dans l'ordre décroissant

4.4 Invasion repoussée

- ▷ Tout Invasion 90's
- ▷ Visualisation graphique initiale de la vague d'ennemis prévue dans le fichier
- ▷ Interaction à la souris pour construire les défenses
- ▷ Une interface graphique simple pour jouer, par exemple à l'aide de la bibliothèque ncurses (un tuto) ou bien SDL2 (un tuto) installées au Crio Unix

4.5 Bonus

Pour les plus motivés, voici quelques pistes de bonus possibles.

- ▷ Visualisation des "tirs" des tourelles
- ▷ On démarre avec moins d'argent, mais tuer des ennemis permet d'en gagner et il est possible d'ajouter (ou améliorer !) des tourelles en cours de jeu
- ▷ Faire un niveau infini où les salves d'ennemis sont aléatoires mais vont grandissantes avec le temps. C'est une sorte de mode *survival* où il faut faire le plus de points possibles à partir d'une somme d'argent donnée fixe
- ▷ une « IA » proposant une stratégie de défense ou d'attaque
- ▷ ...

5 Conditions de rendu

Le projet est à effectuer en **binôme**. En cas de nombre impair d'étudiants, **un seul** groupe sera autorisé à effectuer le projet en **trinôme** (notation plus sévère). Pour former les groupes, utiliser le fichier Excel dans le canal Teams correspondant. **Les binômes doivent être différents de ceux utilisés pour le projet d'Archi !**

Toute question sur le projet devra être posée dans l'équipe Teams (et non par mail ou en individuel) afin de limiter les doublons de questions.

Le projet est à rendre avant la date et l'heure indiquées plus haut sur l'espace Moodle dédié. **Chaque heure de retard sera pénalisée d'un point sur la note finale** (une heure entamée étant due). Le format de rendu est une archive au format ZIP contenant :

- Le code-source de votre projet (éventuellement organisé en sous-dossiers).
- Un fichier `README` à la racine, indiquant comment compiler et exécuter votre projet.
- Un document `dev.pdf`, devant justifier les choix effectués, les avantages et inconvénients de vos choix, expliquer les algorithmes et leur complexité, indiquer quelles ont été les difficultés rencontrées au cours du projet ainsi que la répartition du travail entre les membres du binôme. Un programmeur averti devra être capable de faire évoluer facilement votre code grâce à sa lecture. En aucun cas on ne doit y trouver un copier/coller de votre code source. Ce rapport doit faire le point sur les fonctionnalités apportées, celles qui n'ont pas été faites (et expliquer pourquoi). Il ne doit pas paraphraser le code, mais doit rendre explicite ce que ne montre pas le code. Il doit montrer que le code produit a fait l'objet d'un travail réfléchi et minutieux (comment un bug a été résolu, comment la redondance dans le code a été évitée, comment telle difficulté technique a été contournée, quels ont été les choix, les pistes examinées ou abandonnées...). Ce rapport est le témoin de vos qualités scientifiques mais aussi littéraires (style, grammaire, orthographe, présentation).
- Optionnellement un `makefile`.

L'archive aura pour nom `Nom1Nom2.zip`, où `Nom1` et `Nom2` sont les noms des membres du binôme par ordre alphabétique. L'extraction de l'archive devra créer un dossier `Nom1Nom2` contenant les éléments précisés ci-dessus.

Il va sans dire que les différents points suivants doivent être pris en compte :

- Projet compilant sans erreur ni warning avec l'option `-Wall` de gcc.
- Uniformité de la langue utilisée dans le code et des conventions de nommage et de code.
- Les sources doivent être commentées, dans une unique langue, de manière pertinente (pas de commentaire « fait un test » avant un `if`).
- Le code doit être propre et correctement indenté.
- Bonne gestion de la libération de la mémoire (utilisez `valgrind` pour vérifier que le nombre de `free` est égal au nombre de `malloc` : il faut d'abord compiler avec l'option `-g`, puis lancer `valgrind` avec en argument le nom du programme. Par exemple :

  ```
  gcc -g main.c
  valgrind ./a.out
  ```

  `Valgrind` peut également vous indiquer la ligne où une *segmentation fault* a eu lieu, pratique ! – vous pouvez aussi compiler avec l'option `-fsanitize=address` pour savoir cela.
- Le projet doit évidemment être propre à chaque binôme. **Un détecteur automatique de plagiat sera utilisé**. Si du texte ou une petite portion de code a été emprunté (sur internet, chez un autre binôme,...), il faudra l'indiquer dans le rapport, ce qui n'empêchera pas l'application éventuelle d'une pénalité. Nous rappelons l'importance d'avoir du code original écrit par vous. L'utilisation de générateur automatique de code basé sur l'IA est à proscrire. Tout manque de sincérité sera lourdement sanctionné (conseil de discipline) – c'est déjà arrivé.

La documentation (rapport, commentaires...) compte pour un quart de la note finale.

Soutenance

Une soutenance d'une dizaine de minutes aura lieu pour chaque binôme, à la date et à l'heure indiquées plus haut, **sur les machines de l'université** (vérifier que votre code y fonctionne à l'avance). Elle doit être préparée et menée par le binôme (*i.e.* fonctionnant parfaitement du premier coup, avoir préparé des illustrations des qualités de votre projet, etc.).

Pendant la soutenance, ne perdez pas de temps à nous expliquer le sujet : nous le connaissons puisque nous l'avons écrit. Essayez de montrer ce qui fonctionne et de nous convaincre que vous avez fait du bon travail.

Bon courage !