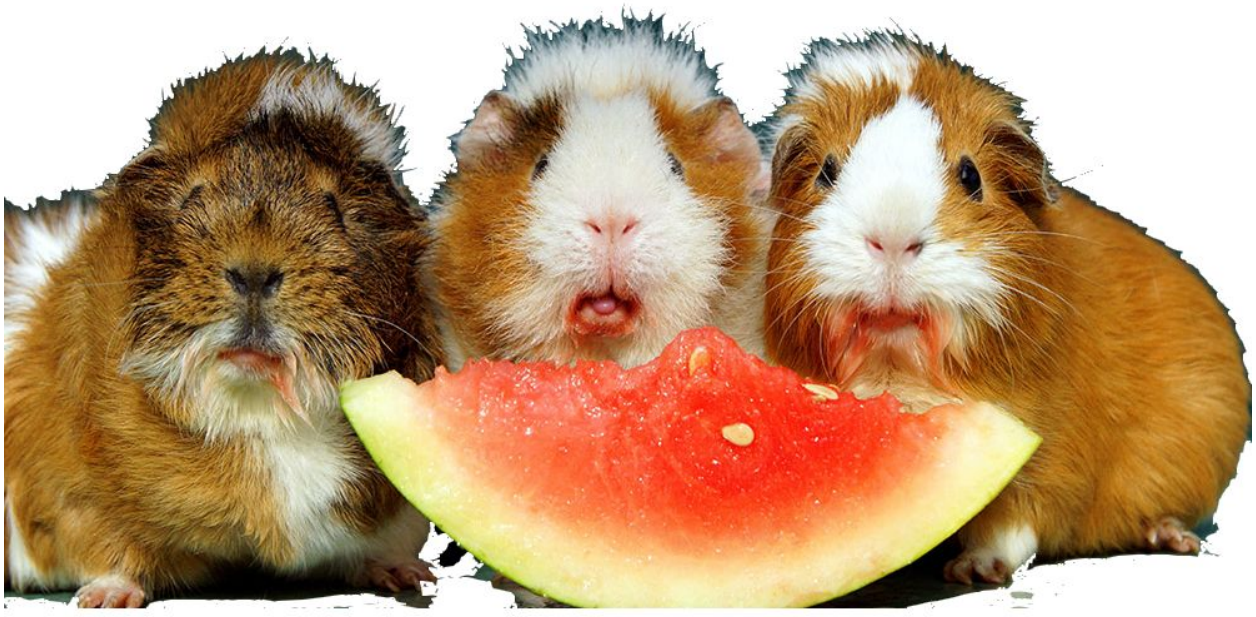
Projet Programmation-Algorithmie - Synthèse de l'Image

Guinea Pig Attack - Tower Defense

VALLET Gaëlle



Sommaire :

Introduction :

Les cochons d'inde de sont échappés ! Place des légumes sur le chemin pour les ralentir et pouvoir les remettre dans leur cage avant qu'ils ne sortent !

Le but de ce projet de programmation consiste à réaliser à jeu de type Tower Defense.
Le but de ce jeu est de placer des tours pour empêcher les monstres qui apparaissent à un point d'entrée d'atteindre le point de sortie. Les vagues de monstres sont de plus en plus difficile à vaincre pour amener le joueur à mettre en place un stratégie pour la placement des tours.

Ce projet devait à l'origine être réalisé en binôme, mais suite au départ de mon camarade, j'ai dû trouver un nouveau binôme. Ce dernier ne désirant pas participer au projet j'ai finalement décidé de continuer le projet seule. Par conséquent, par manque de temps, certains points demandés ne seront pas implémenté et une seule map est jouable.
Je n'ai pas implémenté :

- La vérification du fichier IDT.
- La constructibilité des zones.
- Les sprites de déplacement des monstres et du placement des tours.
- Le menu, la page d'aide, la possibilité de mettre le jeu en pause, le relancement du jeu en fin de partie.
- Les bâtiments.

Présentation de l'application :

Le programme a été codé en langage C, à partir des bibliothèques SDL_image (pour l'affichage des texture), et SDL_ttf pour l'affichage des textes.
Au lancement de l'application itd, le joueur doit entrer dans le terminal le nom du fichier correspondant à la carte du jeu. Le programme affiche alors la carte souhaitée en fonction et récupérer les coordonnées des noeuds pour le déplacement des monstres.

Comment jouer ?

Ouvrir le dossier dans un terminal.
(Optionnel : pour supprimer les fichiers présents dans le dossier bin, utiliser la commande "make clean")
Lancer la compilation avec la commande "make".
Après compilation, le joueur doit lancer le programme du fichier bin/itd.
Le programme lui demandera d'entrer le nom du fichier de la carte qu'il souhaite jouer (Ex : map1.itd). Si le nom de fichier n'existe pas, le programme s'arrête. Si le fichier existe, la fenêtre SDL s'ouvre, les variables utiles au jeu sont implémentées, la carte est affichée. Le jeu commence directement avec l'envoie de la première vague.
Le joueur doit cliquer sur les zones constructibles pour ajouter les tours. Par défaut, ce sont les tours de type 1 (les aubergines) qui seront ajouté au clique. Il est possible de changer le type de tour construite à l'aide des touches suivantes :

- C : Tour de type 1 (aubergine)
- V : Tout de type 2 (poivron)
- B : Tour de Type 3 (pomme)
- N : Tour de type 4 (tomate)

Il faut disposer de suffisamment d'argent pour construire les tours :

- Tour de type 1 : 15 coins
- Tour de type 2 : 20 coins
- Tour de type 3 : 30 coins
- Tour de type 4 : 10 coins

Si un monstre est dans le rayon d'action d'une tour, la tour tir, le monstre perd de la vie.
Lorsque toutes les vagues de monstres ont été vaincues, l'écran de victoire d'affiche : le joueur a gagné. Si des monstres atteignent le point de sortie : le joueur a perdu, l'écran e défaite s'affiche.
Il est possible de quitter le jeu à tout moment en appuyant sur la touche Echap ou Q.

Architecture :

Arborescence :

TowerDefense
- Makefile
- [+] bin
- [+] data
- [+] doc
- [-] images
  - [+]monstre
  - [+]tours
  - [+]scénario
  - [+]police
- [-] include
  - chemin.h
  - main.h
  - map.h
  - monstre.h
  - node.h
  - affichage.h
  - texte.h
  - texture.h
  - tour.h
  - vague.h
  - map.h
- [+] lib
- [+] obj
- [-] src
  - chemin.c
  - main.c
  - map.c
  - monstre.c
  - node.c
  - affichage.c
  - texte.c
  - texture.c
  - tour.c
  - vague.c
  - main.c
  - map.c

Déroulement du jeu :

DEBUT :

Initialisation :

Initialisation de la SDL
initialisation de SDL_ttf (pour le texte)
On déclare les tableaux qui seront utilisés pour chacun des paramètres (construction , noeud...)
On demande le fichier itd
On déclare les listes utilisées
Puis on déclare toutes les variables liées au jeu

BOUCLE AFFICHAGE :

Affichage de la carte en fond.
Affichage du chemin.
Si le jeu est en cours (ni perdu ni gagné)
Affichage des tours.
Gestion des vagues
Gestion des monstres éliminés.
Gestion des déplacements des monstres et des tirs des tours
Affichage des monstres.
Si la partie est perdue ou gagnée (un monstre est parvenu au bout du chemin ou le joueur survit à 20 vagues).
Suppression des monstres et des tours.
Affichage du curseur personnalisé.
Boucle de gestion des événements souris
Boucle de gestion des événements clavier

FIN BOUCLE AFFICHAGE (le joueur a quitté le jeu)

Libération de l'espace mémoire

FIN

Structure des données :

Les montres, tours et noeuds sont implémentés sous forme de liste chainées.

Structure de monstre

```
typedef struct s_Monstre{
    int x;
    int y;
    node * dest;
    node * oldDest;
    int vitesse;
    int vie;
    int direction;
    int idmonstre;
    int typemonstre;
    float resistTour1;
    float resistTour2;
    float resistTour3;
    float resistTour4;
    int go;
    struct s_Monstre* next;
} Monstre;
typedef Monstre* l_Monstre;
```

Structure de Tour

```
typedef struct s_Tour{
    int x; //position
    int y;
    int ray;
    int cost;
    int type;
    int puissance;
    float cadence;
    int active;
    float timeCrea;
    float timeConst;
    float timeDernierTir;
    struct s_Tour* next;
} Tour;
typedef Tour* l_Tour;
```

Structure de Noeud

```
typedef struct s_node{
    int x;
    int y;
    struct s_node* next;
} node;
typedef node* l_node;
```

Fonctionnalités :

Les éléments du jeu :

Les tours :

Tours de type 1 : Aubergine
Rayon d'action = 200 px
Puissance = 700
Cadence = 0.9
Coût = 15
Temps de construction = 2.f

Tour de type 2 : Poivron
Rayon d'action = 200
Puissance = 100
Cadence = 0.1
Coût = 20
Temps de construction = 10.f

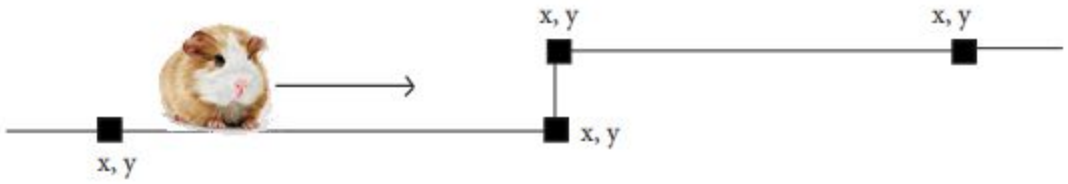
Tour de type 3 : Pomme
Rayon d'action = 200
Puissance = 70
cadence = 0.2
Coût = 30
Temps de construction = 1.f

Tour de type 4 : Tomate
Rayon d'action = 200
Puissance = 200
Cadence = 0.5
Coût = 10
Temps de construction = 2.f

Gestion du jeu

Les déplacements :

Les monstres se déplacent selon un chemin défini par une suite de noeuds selon l'algorithme de plus court chemin. Chaque noeud possède ses coordonnées (x et y) ainsi qu'un pointeur sur son prochain. Les monstres possèdent chacun leurs coordonnées en x et en y, le noeud de provenance et le noeud de destination. Chaque monstre possède un pointeur sur son prochain. Les déplacements d'un monstre consistent à le mener d'un point A (noeud d'origine) à un point B (noeud de destination). Lorsqu'il attend sa destination il prend pour noeud d'origine celui-ci et pour noeud de destination le prochain dans la liste de noeud (chemin). Le jeu recalcule les évènements de jeu à chaque passage dans la boucle d'affichage.
On calcule dans un premier temps la nouvelle position comme si le monstre se déplaçait seulement selon les x ou y, s'il y a un angle de déplacement on calcule la pente du segment [noeud d'origine ; noeud de destination] (équation de la droite), puis on y applique le déplacement en x ou y précèdent afin de connaître ses nouvelles coordonnées. On réitère cette opération avec tous les monstres de la liste.



Construire une Tour :

Lorsque l'utilisateur clique sur la carte, le programme vérifie qu'il n'y a pas déjà un tour et que le joueur a assez d'argent. Si c'est le cas, on ajoute un Tour à la liste avec les coordonnées du cliques et le type de tour sélectionné avant le clique.

Gestion des tours :

Pour la création d'une tour, on vérifie si elle se situe sur une zone constructible (terrain et superposition de tours) , ont créé une tour et on lui attribue sa position et ses caractéristiques en fonction de son type. On cherche le monstre le plus près de la tour dans son rayon d'action, pour lui ôter de la vie à hauteur de la puissance de la tour on applique la résistance du monstre à ce type de tour. On trace une ligne entre le monstre et la tour pour représenter l'attaque. On réitère ces étapes pour les tours suivantes.

Les vagues :

La difficulté du jeu change avec son avancé.
Chaque vague voit sa vie augmenter de 1000, à chaque nouvelle vague (vie = vague x 1000).
La résistance à chaque tour est par défaut de 0.6 pour les types 1, 1 pour les types 2, 1 pour les types 3, et 1 pour les type 4.La résistance des monstres aux différents types de tour change en fonction de l'avancée du jeu :

Vague 5 et supérieurs : la résistance est de 50% pour les tours de type 1.

Vagues 10 et supérieur : la résistance est de 50% pour les tours de type 2.

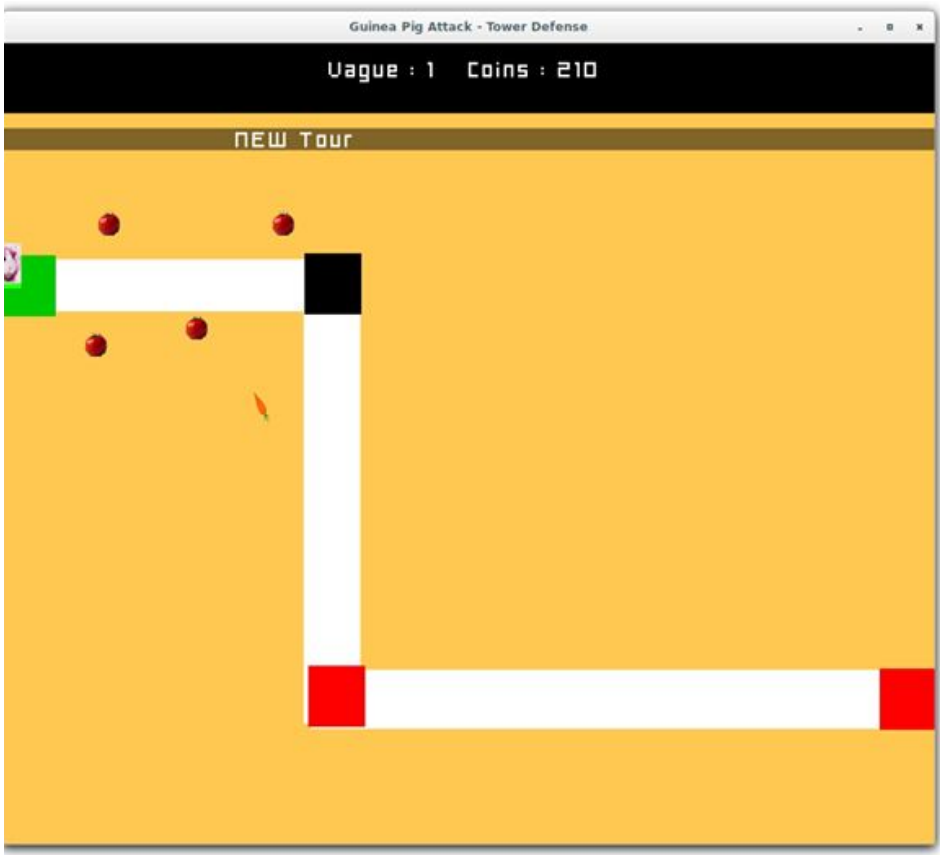
Vague 15 et supérieurs : la résistance est 70% sur les tours de type 2, et 50% pour les tours type 3.

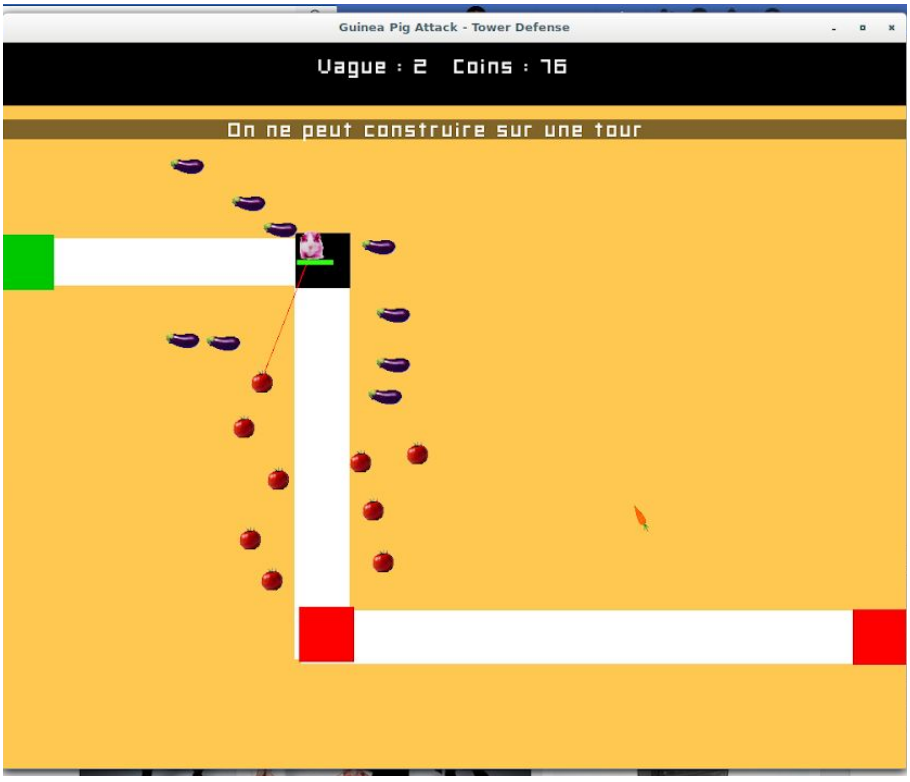
Vague 20 et supérieur : la résistance est 30% sur les tours de type 2, et 50% pour les types 3, et 50% pour les types 4.

Raccourcis clavier :

ESC/Q : quitter le jeu
C : Tour de type 1
V : Tour de type 2
B : Tour de type 3
N : Tour de type 4

Résultats obtenus





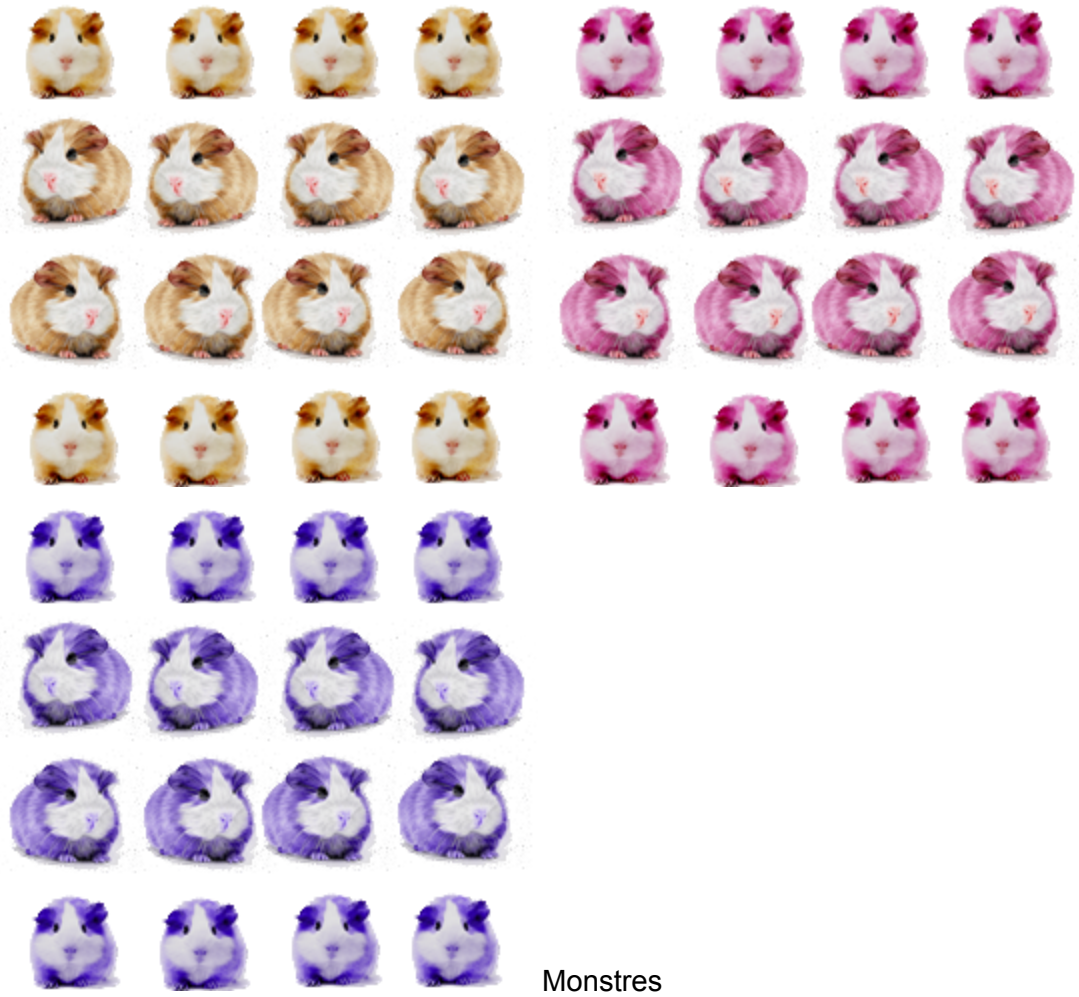
Guinea Pig Attack - Tower Defense

Design

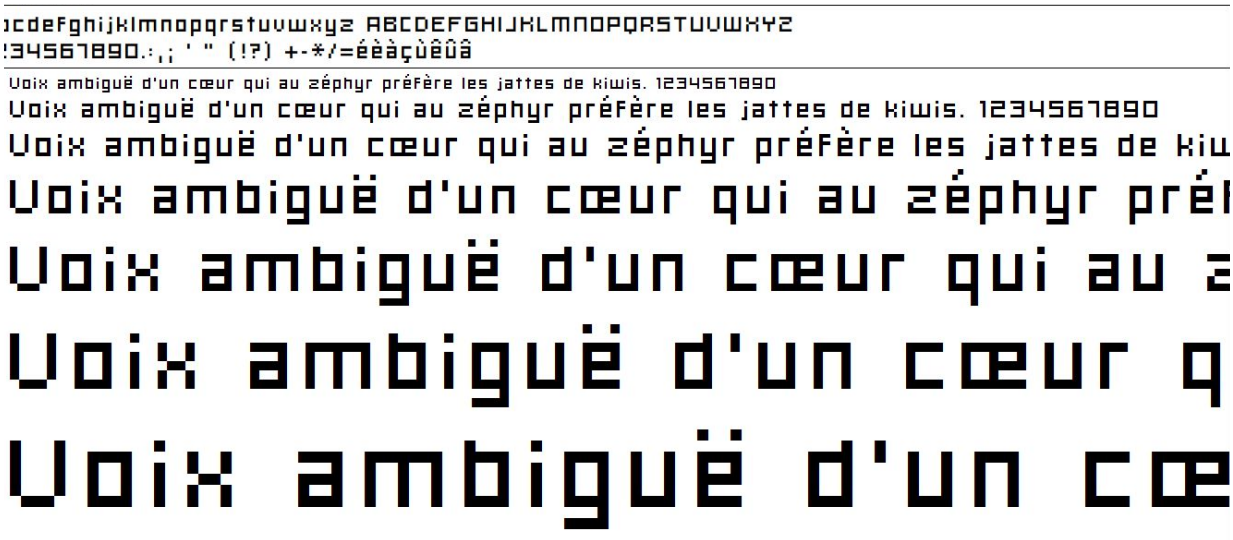
Ecrans de Défaite/Victoire

Image du curseur de la sourie

Tours



Monstres



Police hooge 06_56 utilisée pour les textes du jeu

Toutes les images utilisées sont libres de droit.

Conclusion

Ce projet a été l'occasion de mettre en oeuvre les notions vues en cours, mais aussi d'aller au delà, aussi bien dans l'utilisation de nouvelles bibliothèques qu'en recherche algorithmique.
Ce projet étant mon premier jeu, j'ai dû adapter ma façon de travailler et mon organisation pour optimiser ma productivité. Les dimension ludique et créatives ont été très motivante. Ce projet m'a permis d'approfondir mes connaissances en langage C et découvrir de nouvelles fonctionnalités, comme la librairie SDL_ttf pour l'affichage du texte, et d'acquérir de nouvelles méthodes de travail. J'ai été confronté à des problématiques nouvelles qui m'ont fait revoir ma façon de travailler et de penser le projet plusieurs fois. Mon but était de créer une application jouable, je suis satisfaite de cet aspect. J'aurai néanmoins aimé livré un projet avec toute les fonctionnalités demandées et un meilleur rendu visuel, et travailler le rendu sonor.

Bien qu'ayant pris du retard suite au départ de mes binômes successifs, j'ai pu compter sur l'aide et le soutien de mes camarades et de Lug ESLAN (étudiant en troisième année de Bachelor Informatique au CS2I de Nevers) pour rendre un programme utilisable, bien que incomplet ; je tiens à les remercier pour leur aide.