

CAHIER DES CHARGES

Projet TowerForce

CHARLOTTE CORAZZA ; GAULTHIER MARTIN ; GUILLAUME TREME

08/03/2018



Table des matières

I.	Contexte du projet.....	2
II.	Présentation du jeu.....	2
A.	Règles générales du jeu.....	2
B.	Au début du jeu.....	2
C.	Détails des niveaux et interfaces.....	2
D.	Description des fonctionnalités.....	4
III.	Contraintes.....	4
IV.	Diagramme des modules.....	5
A.	Diagramme des modules simplifié.....	5
B.	Les modules et leur contenu.....	6
V.	Agenda de développement.....	7
A.	Diagramme de Gantt.....	7
B.	Description détaillée des tâches.....	7
1.	Cahier des charges.....	7
2.	Création des classes.....	8
3.	Affichage et Interface.....	9
4.	Création des niveaux.....	10
5.	Améliorations.....	10
6.	Compilation.....	10

I. CONTEXTE DU PROJET

Cette application est développée dans le cadre d'un projet commun en Conception et Développement d'Applications (LIFAP4) en seconde année de licence Informatique. L'équipe est constituée de trois étudiants de l'Université Claude Bernard Lyon 1 en association avec les enseignants de cette matière.

Ayant tous un goût prononcé pour les jeux vidéo, notre choix de projet s'est naturellement tourné vers l'un d'eux. Nous avons décidé de développer sur ordinateur un classique des jeux mobiles : un Tower Défense. Celui-ci portera le nom de « TowerForce ».

Ce cahier des charges comporte :

- ✚ Une présentation générale du jeu « TowerForce » avec une explication des règles et des fonctionnalités
- ✚ Une exposition des contraintes de développement
- ✚ La présentation du diagramme des modules du jeu
- ✚ Un agenda de développement grâce à un diagramme de Gantt et une description détaillée des tâches à accomplir

II. PRESENTATION DU JEU

A. Règles générales du jeu

Le principe d'un Tower Défense est simple. Une carte prédéfinie et comportant un chemin précis est affichée à l'écran puis des ennemis apparaissent et se déplacent dessus par vague. Le joueur doit placer des Tours le long de la route, à l'emplacement qu'il désire, de sorte à pouvoir tirer sur ces ennemis pour les anéantir.

Si 20 ennemis réussissent à arriver au bout du chemin pendant un niveau, le joueur a perdu. Au contraire, s'il tue tous les ennemis avant que ceux-ci n'atteignent la fin du chemin, il gagne. Le but est de réussir à éliminer tous les ennemis au fil des niveaux. La vie du joueur se réinitialise à chaque nouveau niveau atteint.

Au fil des niveaux : les cartes changeront, le nombre de vagues évoluera et la quantité d'ennemis par vague augmentera. Il y aura plusieurs types d'ennemis et de Tours avec des statistiques particulières.

Enfin, le joueur gagne des Pièces d'Or à chaque ennemi tué grâce à une Tour. Cet argent lui permettra d'acheter d'autres Tours entre chaque vague pour augmenter sa défense et ses chances de victoires.

B. Au début du jeu

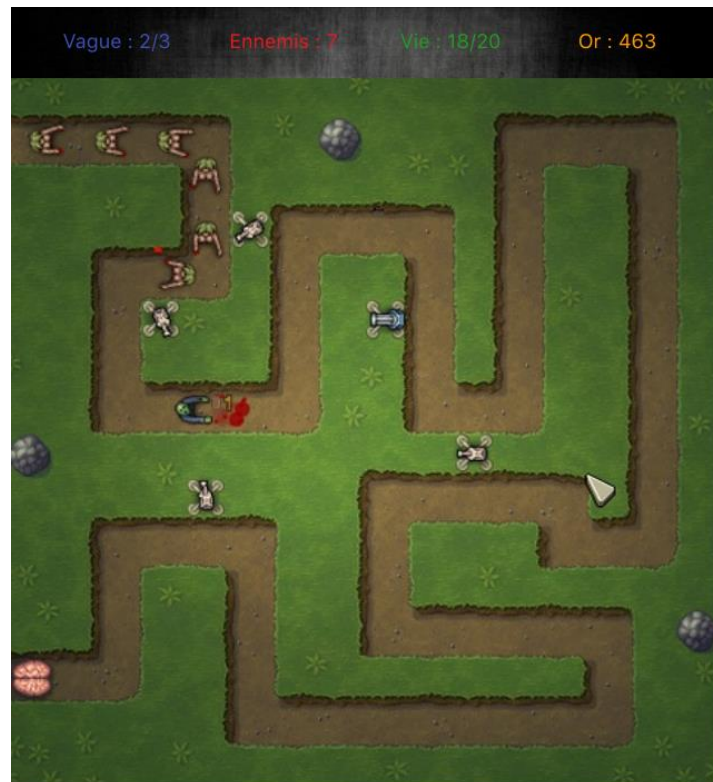
Au début d'une nouvelle partie, la carte du niveau 1 s'affiche sur l'écran du joueur. Il peut alors placer 2 Tours Standards où il le souhaite le long de la route. Un bouton « Démarrer Vague » permet de commencer la partie.

C. Détails des niveaux et interfaces

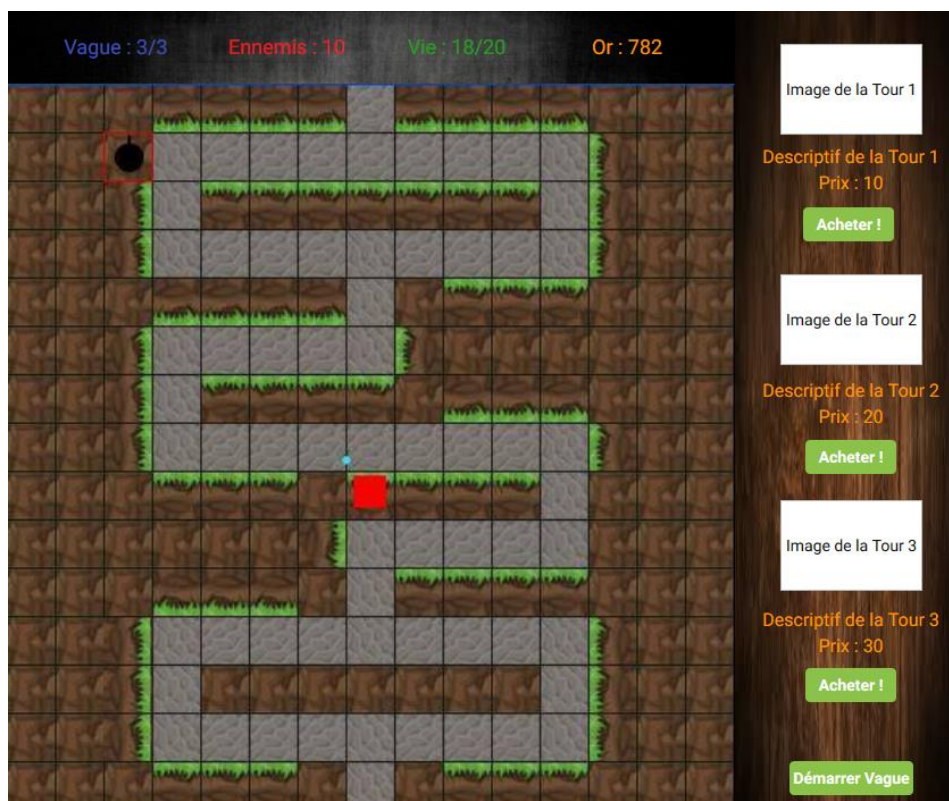
Chaque partie s'articule sur plusieurs niveaux. Un niveau est défini par :

- ✚ Une carte prédéfinie
- ✚ Un nombre de vagues précis
- ✚ Un nombre d'ennemis par vague
- ✚ Un chemin prédéfini pour les ennemis

L'interface affiche la carte du niveau au centre de la fenêtre. Un espace dédié en haut indique la vague en cours et le nombre d'ennemis à tuer ou qui reste à abattre. La vie du joueur et la quantité de Pièces d'Or acquises sont également affichées. Voici un exemple d'interface lors du jeu :



Lors de la phase inter-vague, le joueur peut acheter de nouvelles Tours grâce à un bandeau d'achat qui s'affiche sur la partie de gauche de l'écran. Voici un exemple d'interface lors de la phase inter-vague :



D. Description des fonctionnalités

Voici les fonctionnalités de « TowerForce » :

- ✚ Implémentation de plusieurs niveaux différents pour faire évoluer la difficulté au cours de la partie.
- ✚ Mise en place d'une évolution possible des Tours et de leurs statistiques à partir d'un certain nombre d'ennemis tués par la Tour.
- ✚ Plusieurs ennemis différents avec des statistiques particulières.
- ✚ Possibilité pour le joueur d'acheter des Tours entre chaque vague ennemie grâce aux Pièces d'Or amassées.
- ✚ Placement des Tours libre le long du chemin pour le joueur.

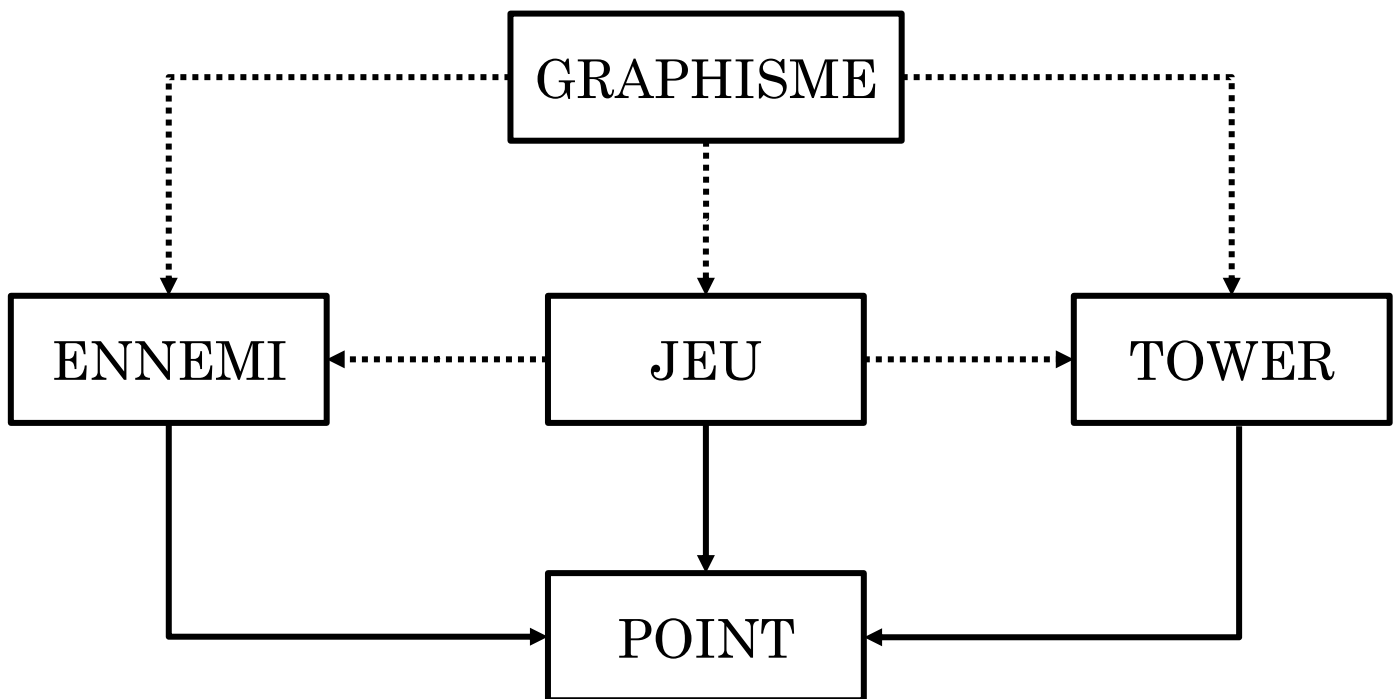
III. CONTRAINTES

Nous développerons notre projet avec cet ensemble de contraintes :

- ✚ Le jeu sera codé en C/C++ et sous le système d'exploitation Linux (GCC) et Windows grâce à un projet CodeBlocks ou Microsoft Visual Studio 2015.
- ✚ La bibliothèque graphique utilisée sera la SFML ainsi qu'une librairie texte pour une version démo.
- ✚ Nous utiliserons le débogueur gdb ainsi que l'outil de profiling Valgrind.
- ✚ Une documentation Doxygen sera générée automatiquement.
- ✚ Le code sera correctement indenté et commenté.
- ✚ Les variables porteront des noms qui ont du sens.
- ✚ Une démo du jeu en version texte sera présentée le 27 mars 2018.
- ✚ Le jeu complet sera livré le 23 avril 2018 avant 18h.

IV. DIAGRAMME DES MODULES

A. Diagramme des modules simplifié



B. Les modules et leur contenu

Point
-x: unsigned int
-y: unsigned int
+Point()
+~Point()
+setX(nx:unsigned int): void
+getX(): unsigned int
+getY(): unsigned int
+setY(ny:unsigned int): void

Tower
-PA: unsigned int
-Zone: unsigned int
-Coord: Point
-NbTués: unsigned int
-VA: unsigned int
-level: unsigned int
-tTower: sf::Texture
-sprite_tower: sf::Sprite
+Tower()
+~Tower()
+getPA(): unsigned int
+getLevel(): unsigned int
+getNbTues(): unsigned int
+getVA(): unsigned int
+getZone(): unsigned int
+getSprite(): sf::Sprite
+setPA(nPA:unsigned int): void
+setZone(nZone:unsigned int): void
+setNbTues(nNbTués:unsigned int): void
+setVA(nVA:unsigned int): void
+setLevel(nLvl:unsigned int): void
+levelUp(nbTués:unsigned int): void

Graphisme
+fenetre: sf::RenderWindow
+affiche(): void
+eventClavier(): void
+eventSouris(): void
+chargerBackground(map:int [20][20]): void
+chargerEnnemi(<Ennemi *>:vector): void

Ennemi
-PV: unsigned int
-Coord: Point
-Loot: unsigned int
-ID: unsigned int
-anim: sf::Vector2i
-Type: uint
-tEnnemi: sf::Texture
-sprite_ennemi: sf::Sprite
+Ennemi()
+~Ennemi()
+getPV(): unsigned int
+setPV(nPV:unsigned int): void
+bouge(): void
+recoitDegats(Degats:const unsigned int): unsigned int <i>renvoie le loot si tué sinon 0</i>
+getTexture(): sf::Texture
+setTexture(NomFic:const string): void
+getSprite(): sf::Sprite
+setSprite(Texture:const sf::Texture): void
+getLoot(): unsigned int
+getAnim(): sf::Vector2i
+setAnim(a:unsigned int,b:unsigned int): void

Jeu
-Matrice: int [20][20]
-Niveau: int = 1
-NumVague: int = 1
-NbEnnemis: int
-PO: int = 0
-Vie: int = 20
-NbVague: int
-PointDebParcours: const Point
-lEnnemis: vector <Ennemi *>
-lTowers: vector <Tower *>
+afficheMatrice(): void
+boucleEvent(): void
+chargerMatrice(Niveau:const int): void
+creerVague(Niveau:const int,Vague:const int): void <i>-> alimente lEnnemis</i>
+afficheVague(): void
+poserTower(Coord:Point): void <i>-> alimente lTowers</i>
+tirTowers(): void
+bougerEnnemis(): void
+testDroite(Ennemi:sf::Sprite &,ID:unsigned int): void
+testGauche(ID:unsigned int,ennemi:sf::Sprite &): void
+testHaut(ID:unsigned int,ennemi:sf::Sprite &): void
+testBas(ID:unsigned int,ennemi:sf::Sprite &): void
+getPO(): int
+getNbEnnemis(): int
+getNbVague(): int
+getVie(): int
+setNumVague(NvNum:int): void
+setVie(NvVie:int): void
+setPO(NvPO:int): void


V. AGENDA DE DEVELOPPEMENT

A. Diagramme de Gantt


	Tache	Description	Février	Mars					Avril		
			S5	S1	S2	S3	S4	S5	S1	S2	S3
1- Cahier des charges	1.1	Diagramme des classes									
	1.2	Diagramme de Gantt									
	1.3	Rédaction du Cahier des charges									
2- Création des classes	2.1	Création base et boucle du jeu									
	2.2	Création et implémentation des graphismes									
	2.3	Création de la classe Point									
	2.4	Création de la classe Ennemis									
	2.5	Création de la classe Tower									
3- Affichage et Interface	3.1	Fonction de tir et de mouvement									
	3.2	Gestion des vagues									
	3.3	IHM et menu d'achat									
4- Création de niveaux	4.1	Initialisation de niveau									
	4.2	Gestion de difficulté									
5- Améliorations	5.1	Ajout Ennemis									
	5.2	Ajout Tours									
6- Compilation	6.1	Makefile et Codeblocks									
	6.2	Test de Régression									

B. Description détaillée des tâches


1. Cahier des charges

 **Tâche 1.1 : Création du diagramme des modules**

Durée : 1 semaine

 **Tâche 1.2 : Création du diagramme de Gantt**

Durée : 1 semaine

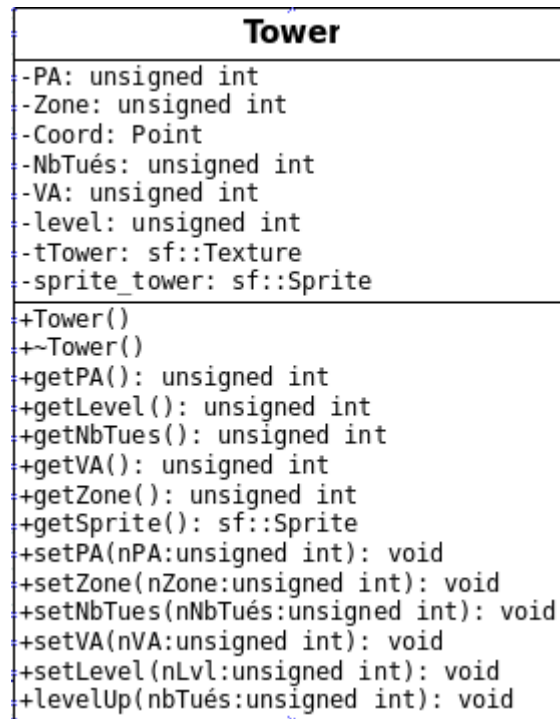
 **Tâche 1.3 : Rédaction du cahier des charges**

Durée : 1 semaine

Tâche 2.5 : Création de la classe Tower

Durée : 2 semaines

Description : Développement de la classe basique Tower. Une Tour a des Points d'Attaque (PA), une Vitesse d'Attaque (VA), une position (Coord), une zone d'attaque (Zone), un nombre d'ennemis tués (NbTués) et un niveau (level). D'un point de vue graphique, il a également une texture (tTower) associée ainsi qu'un Sprite permettant de le faire bouger (sprite_tower).



3. Affichage et Interface

Tâche 3.1 : Fonction de tir et mouvement

Durée : 2 semaines

Description : Implémentation des mouvements de marche automatique des ennemis le long d'un chemin puis gestion des tirs des Tours contre ceux-ci. Les mouvements de marche se feront à l'aide des Sprites et seront dans la boucle d'événements comme élément automatique. La fonction de Tir doit toucher un ennemi dans la zone d'attaque de la Tour et déduire les points d'attaque de cette dernière aux points de vie de l'ennemi. Si les points de vie d'un ennemi sont égaux ou inférieurs à 0, alors l'ennemi doit être supprimé de l'écran.

Tâche 3.2 : Gestion de vagues

Durée : 2 semaines

Description : Gestion des différentes vagues le long du chemin et mise en place de la phase inter-vague avec pause. Plusieurs ennemis doivent pouvoir emprunter le chemin au fur et à mesure et se faire canarder par les Tours. Entre les vagues, les ennemis ne doivent plus apparaître. Affichage d'un bouton « Démarrer vague » pour relancer la suivante. Quand l'utilisateur clique, la nouvelle vague doit s'initialiser et se lancer automatiquement.

Tâche 3.3 : IHM et menu d'achat

Durée : 2 semaines

Description : Création de l'interface finale avec tous les éléments à afficher + création d'une seconde interface lors de l'inter-vague pour acheter des Tours et les placer. Doit apparaître sur l'interface : la vague en cours et le nombre d'ennemis à tuer ou qui reste à abattre, le nombre de vie restant et la quantité de Pièce d'Or acquises. Mise en place de l'économie du joueur avec augmentation du nombre de Pièces d'Or à chaque ennemi tué.

4. *Création des niveaux*

Tâche 4.1 : Initialisation de niveau

Durée : 2 semaines

Description : Création de divers niveaux avec des cartes différentes. Implémentation des fichiers de niveau en suivant un prototype fixé à l'avance et tests avec l'affichage et les vagues ennemies.

Tâche 4.2 : Gestion de difficulté

Durée : 3 semaines

Description : Mise en place d'un nombre d'ennemis différent par vague selon le niveau et équilibrage des forces Ennemis/Tours.

5. *Améliorations*

Tâche 5.1 : Ajout d'ennemis

Durée : 2 semaines

Description : Création de différents types d'ennemis avec évolution de type. Les ennemis n'auront plus le même « type » et seront dotés d'attributs différents au niveau des Points de vie, de la vitesse de déplacement, de leur apparence, etc... Tests des vagues avec divers ennemis.

Tâche 5.2 : Ajout de Tours

Durée : 2 semaines

Description : Création des évolutions de Tours par niveau. Les Tours n'auront plus le même niveau : ce dernier sera décidé en fonction du nombre d'ennemis tués par cette Tour. Toutes auront des attributs différents au niveau des Points d'attaque, de la vitesse d'attaque, de leur apparence, etc...

6. *Compilation*

Tâche 6.1 : Makefile & CodeBlocks

Durée : 8 semaines

Description : Implémentation du Makefile et du projet CodeBlocks avec une actualisation perpétuelle tout au long de la création du jeu.

Tâche 6.2 : Test de Régression

Durée : 8 semaines

Description : Tests et débogage de l'ensemble des parties du code tout au long du développement du projet suivant la Méthode AGILE. Nous utiliserons le débogueur gdb ainsi que Valgrind pour les fuites mémoires et l'optimisation de code.