Clones War

CHAPITRE 1 - Présentation du projet :

Nous sommes un groupe de trois, de DDB corporation:

- Adrien DUPERRIER p1508879
- Thibault DUMENIL p1306146
- Vincent BORNE p1508333

Dans le cadre de la matière LIFAP4, nous sommes amenés à réaliser le développement d'une application.

Clones War est un jeu de stratégie en temps réel en 2 dimensions (joueur vs IA) dans l'univers de Star Wars. Les deux camps possèdent une base avec des points de vie, positionnée aux deux extrémités de la carte.

Le joueur a une barre d'expérience (xp) , un indicateur d'argent (gold) et un catalogue de soldats et de tourelles pouvant être achetés. Le soldat ainsi acheté progresse jusqu'à la base adversaire ou jusqu'à l'ennemi le plus proche.

Les tourelles, quant à elle, sont fixes au niveau de la base et la défendent contre les ennemis. Le joueur possède aussi un pouvoir spécial qui occasionne des dégâts aux unités de l'adversaire. Ce pouvoir a un cooldown, c'est-à-dire un temps de rechargement, entre deux utilisation.

Chaque ennemi détruit rapporte un certain nombre de gold, différent selon la robustesse de l'ennemi en question, et donne également de l'expérience. Une fois assez d'xp accumulé, le joueur peut changer de niveau, faisant ainsi progresser sa base, ses soldats, ses tourelles ainsi que son pouvoir.

La partie se finit lorsque qu'une des deux bases est détruite.

CHAPITRE 2 – Description de la demande :

Résultats visés : Création d'un jeu solo pouvant être utilisé rapidement, fonctionnel et fluide.

Principales caractéristiques :

- Jeu évolutif : le joueur passe des niveaux et débloque de nouvelles unités.

- Il y a un catalogue d'unités corps à corps (CAC) ou distance qui font des dégâts différents ainsi que des tourelles pour défendre sa base. Ce catalogue s'étoffe après chaque gain de niveau .

- Pas de minimum ni de maximum de temps de jeu (la partie prend fin lorsqu'un joueur élimine la défense adverse). Malgré cela, les parties restent courtes, une dizaine de minutes au maximum.

- Le joueur possède une attaque spéciale qui lui permet de faire des dégâts importants aux unités ennemies. Les dégâts de ce pouvoir augmente avec les niveaux et celui-ci possède un temps de recharge.

- Le jeu s'ouvre avec un menu permettant de choisir son camp pour la partie qui suit.

CHAPITRE 3 - Contraintes :

- Durée de développement : un semestre (4h par semaine encadrées) de 9 semaines.

- Première expérience dans un projet de développement.

- Le jeu sera développé en C++ sous Linux avec un éditeur de code.

- Le code doit être géré avec Mercurial.

- Produire une documentation du code via Doxygen.

- Créer un diagramme des modules complets et cohérents.

- Utilisation d'un debugeur : gdb, et d'un profiler : valgrind ou gprof.

CHAPITRE 4 - Déroulement du Projet :

Liste des tâches :

Tâche 0 : Création du cahier des charges
- Rédaction de la présentation du jeu, des différentes tâches et du diagramme de Gant.
- Durée : 2 semaines
- Réalisé quand le cahier des charges se trouve sur TOMUSS

Tache 1 : Création du diagramme de module
- Durée : 2 semaines

Tache 2 : Mise en place des unités avec la création des classes Unit et Bullet

Tache 3 : Faire les tests de régression de ce module unité pour vérifier l'attaque entre unité, les dégâts infligés, les points de vie dynamique, les projectiles avec Bullet, le prix de l'unité

Tache 4 : Mise en place des tourelles avec la création de la classe Turret

Tache 5 : Faire les tests de régression de ce module tourelle pour vérifier l'attaque, le prix, actif/passif

Tache 6 : Mise en place des attaques spéciales avec la création de la classe SpecialWeapon

Tache 7 : Faire les tests de régression de ce module attaque spéciale pour vérifier l'attaque et le temps d'attente

Tache 8 : Mise en place des niveaux avec la création de la classe Level. Cela regroupe tous les autres modules précédent, c'est une classe centrale et donc très importante.

Tache 9 : Faire les tests de régression de ce module niveau en testant la mise en place d'unité, la récupération d'unité par id, la mise en place de tourelle, d'attaque spécial.

Tache 10 : Mise en place d'une autre classe qui regroupe les niveaux nommée Init. Elle va servir de conteneur des niveaux avec des fonctions membres privée du type LevelOne() et des publics getLevelOne()

Tache 11 : Faire les tests de régression de cette classe pour vérifier que le conteneur marche bien, qu'on puisse appelé les niveaux correctement.

Tache 12 : Mise en place du module Player et Bot qui permettent d'avoir toutes les caractéristiques nécessaire pour pouvoir jouer, point de vie, expérience, niveau (pour ensuite invoquer les unités), argent.

Tache 13 : Faire les tests de régression de Player et Bot sachant que Bot sera une sous classe de Player avec des fonctions qui permettront de jouer tout seul du type autoBuyUnit(), autoLevelUp()

Tache 14 : Enfin mise en place de la classe Map qui va permettre de détecter ou les unités sont et les projectiles aussi

Tache 15 : Faire les tests de régression de Map en vérifiant que chaque fonction membre fonctionne correctement

Tache 16 : Enfin mise en place de la logique du jeu avec la classe Game qui regroupe la logique du jeu

Tache 17 : Faire les tests de régression de Game, c’est des tests très importants

Tache 18 : Maintenant tous les modules fonctionnent correctement ensemble. Mise en place de la version texte du jeu. Cette version ne sera certainement pas très aboutie dans le sens où le jeu va être pas mal visuel. Mais on aura une première vue des fonctions du jeux

Tache 19 : Une fois la version texte faite, on pourra ajouter des fonctionnalités qu’on aurait pu oublier

Tache 20 : Enfin mise en place de la version SDL. Première partie, recherche des images ou créations des images. On se met d’accord sur le design de chaque unité et de chaque projectile

Tache 21 : Ensuite implémenter une classe sdlGame qui générera les images et le jeu de façon beaucoup plus visuel que la version texte

Tache 22 : Ajouter du fond sonore au jeu, telle qu’une musique de fond et des bruits pendant les combats

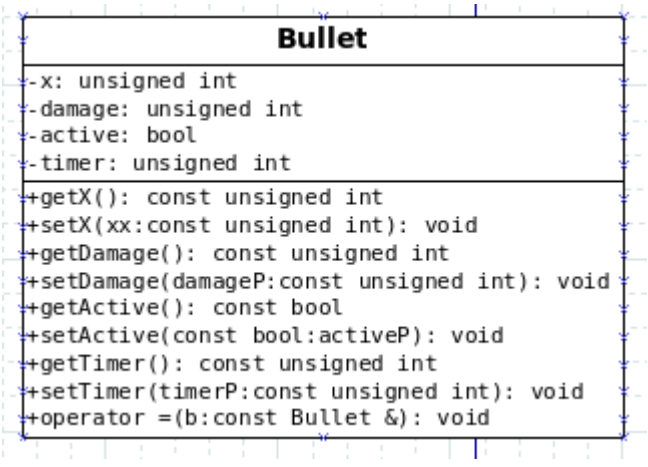
Tache 23 : Ajouter un menu au jeu, une fonction pour recommencer.

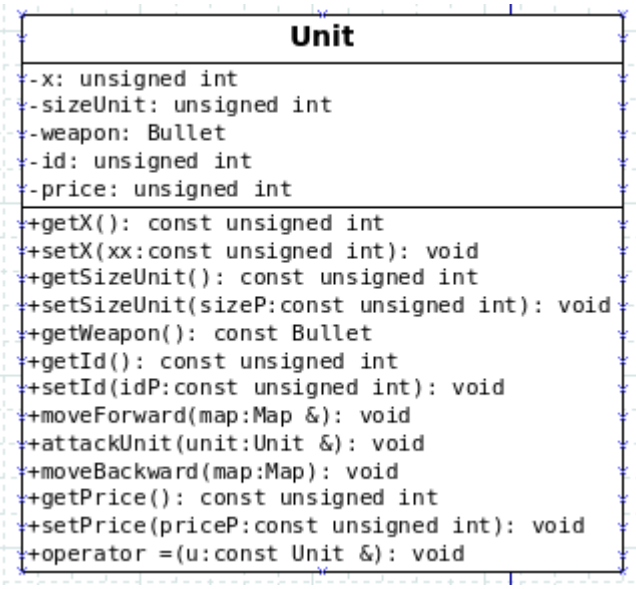
Tache 24 : Optimiser le code, vérifier que tout fonctionne correctement.

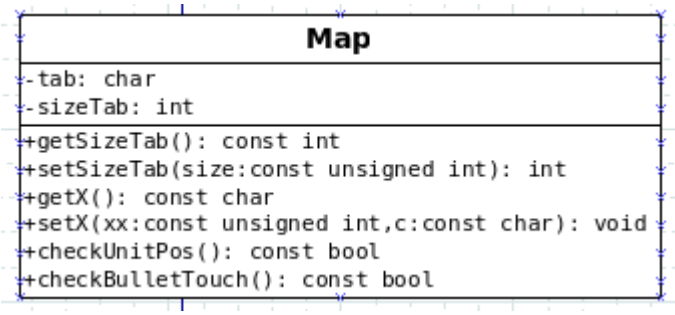
CHAPITRE 5 - Diagramme de Gant :

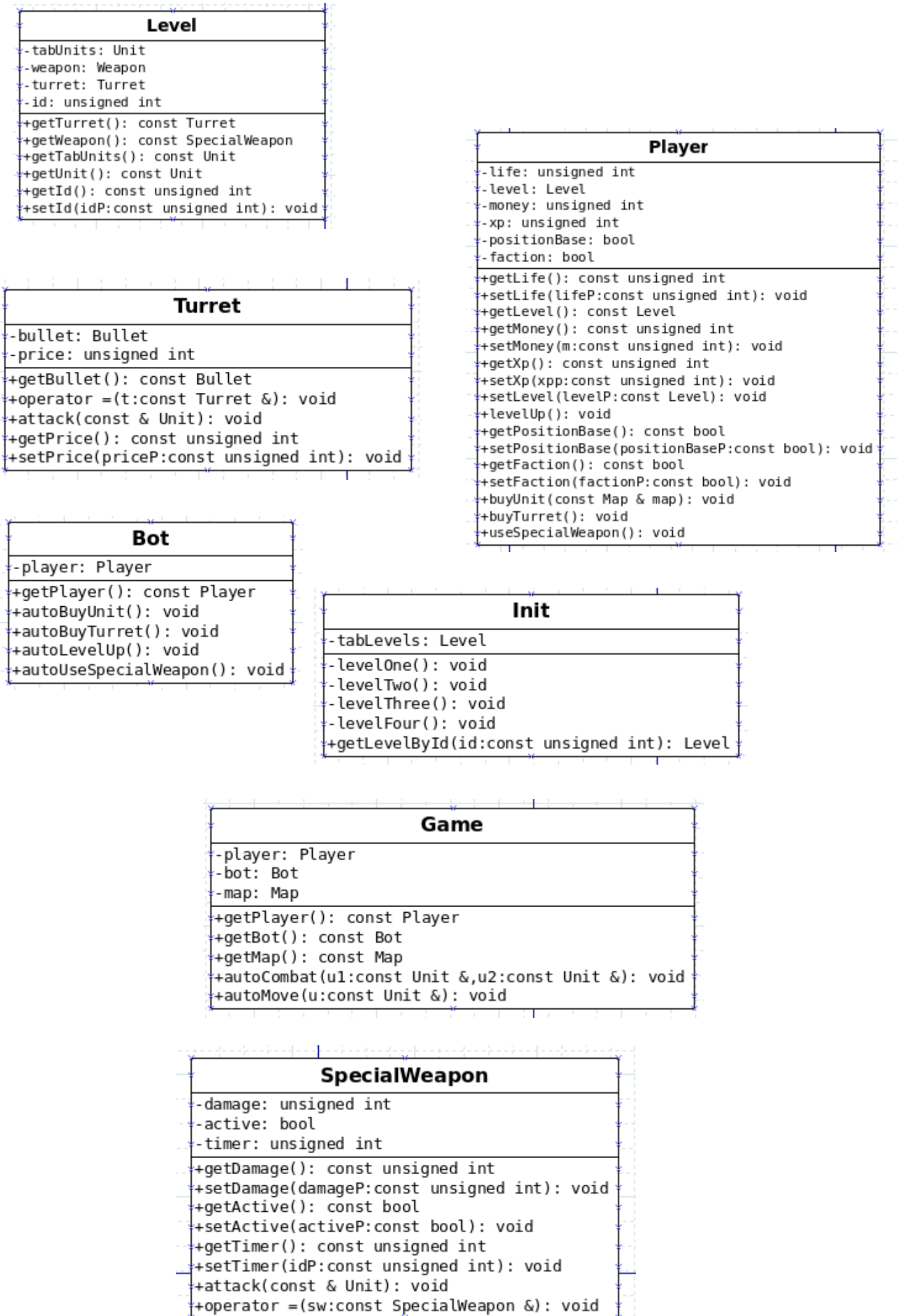
Tache n°	Semaine 1	Semaine 2	Semaine 3	Semaine 4	Semaine 5	Semaine 6	Semaine 7	Semaine 8	Semaine 9
0									
1									
2									
3									
4									
5									
6									
7									
8									
9									
10									
11									
12									
13									
14									
15									
16									
17									
18									
19									
20									
21									
22									
23									
24									

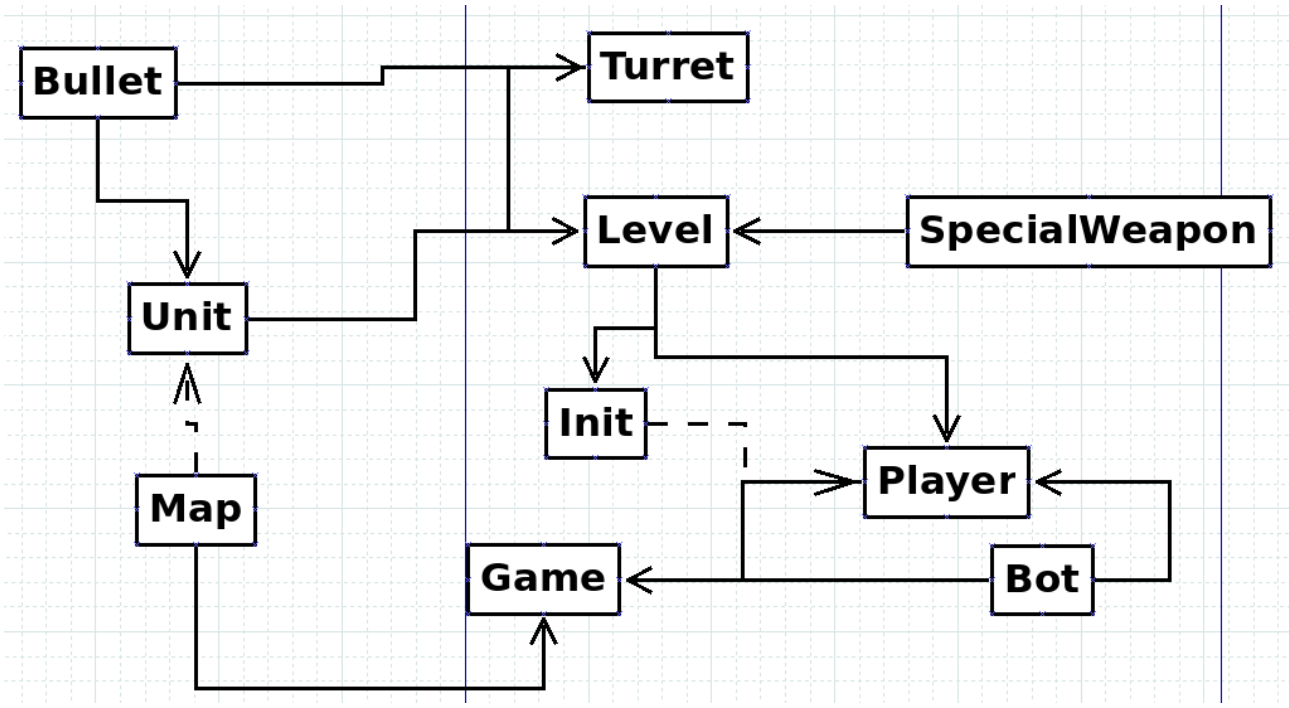
CHAPITRE 5 - Diagramme des classes:








Level
-tabUnits: Unit
-weapon: Weapon
-turret: Turret
-id: unsigned int
+getTurret(): const Turret
+getWeapon(): const SpecialWeapon
+getTabUnits(): const Unit
+getUnit(): const Unit
+getId(): const unsigned int
+setId(idP:const unsigned int): void
Player
-life: unsigned int
-level: Level
-money: unsigned int
-xp: unsigned int
-positionBase: bool
-faction: bool
+getLife(): const unsigned int
+setLife(lifeP:const unsigned int): void
+getLevel(): const Level
+getMoney(): const unsigned int
+setMoney(m:const unsigned int): void
+getXp(): const unsigned int
+setXp(xpp:const unsigned int): void
+setLevel(levelP:const Level): void
+levelUp(): void
+getPositionBase(): const bool
+setPositionBase(positionBaseP:const bool): void
+getFaction(): const bool
+setFaction(factionP:const bool): void
+buyUnit(const Map & map): void
+buyTurret(): void
+useSpecialWeapon(): void
Turret
-bullet: Bullet
-price: unsigned int
+getBullet(): const Bullet
+operator =(t:const Turret &): void
+attack(const & Unit): void
+getPrice(): const unsigned int
+setPrice(priceP:const unsigned int): void
Bot
-player: Player
+getPlayer(): const Player
+autoBuyUnit(): void
+autoBuyTurret(): void
+autoLevelUp(): void
+autoUseSpecialWeapon(): void
Init
-tabLevels: Level
-levelOne(): void
-levelTwo(): void
-levelThree(): void
-levelFour(): void
+getLevelById(id:const unsigned int): Level
Game
-player: Player
-bot: Bot
-map: Map
+getPlayer(): const Player
+getBot(): const Bot
+getMap(): const Map
+autoCombat(u1:const Unit &,u2:const Unit &): void
+autoMove(u:const Unit &): void
SpecialWeapon
-damage: unsigned int
-active: bool
-timer: unsigned int
+getDamage(): const unsigned int
+setDamage(damageP:const unsigned int): void
+getActive(): const bool
+setActive(activeP:const bool): void
+getTimer(): const unsigned int
+setTimer(idP:const unsigned int): void
+attack(const & Unit): void
+operator =(sw:const SpecialWeapon &): void


Bullet
Turret
Level
SpecialWeapon
Unit
Init
Player
Map
Game
Bot