|  |  |
| --- | --- |
| **Auteurs :** | Armand Delessert  Simon Baehler  Benoit Zuckschwerdt  Ngueukam Djeuda Wilfried Karel |
| **Destinataires :** | Eric Lefrançois  Sandy Vibert |

|  |
| --- |
| WarTanks |
| Rapport de projet de GEN |

Contenu

[Introduction 3](#_Toc422346739)

[Déroulement d’une partie 4](#_Toc422346740)

[Conception technique 5](#_Toc422346741)

[Architecture générale 5](#_Toc422346742)

[Technologies utilisées 5](#_Toc422346743)

[Serveur 6](#_Toc422346744)

[Architecture 6](#_Toc422346745)

[Liste des classes 6](#_Toc422346746)

[Diagramme UML 8](#_Toc422346747)

[Communication réseau 9](#_Toc422346748)

[Protocole 9](#_Toc422346749)

[Client 12](#_Toc422346750)

[Architecture 12](#_Toc422346751)

[List des classes 12](#_Toc422346752)

[Diagramme UML 14](#_Toc422346753)

[Début de jeu 15](#_Toc422346754)

[En jeux 15](#_Toc422346755)

[Fin de la partie 17](#_Toc422346756)

[Bonus 18](#_Toc422346757)

[Bonus speed 18](#_Toc422346758)

[Mine 18](#_Toc422346759)

[Ultra Rapide Fire (non implémenté) 18](#_Toc422346760)

[Laser 18](#_Toc422346761)

[Foudroiement (non implémenté) 18](#_Toc422346762)

[Volée de lame 18](#_Toc422346763)

[Bélier (non implémenté) 18](#_Toc422346764)

[Alpha Strick 19](#_Toc422346765)

[Soin 19](#_Toc422346766)

[La Mort 19](#_Toc422346767)

[Fenêtre 19](#_Toc422346768)

[Implémentation du projet 19](#_Toc422346769)

[Technologies utilisées 19](#_Toc422346770)

[Tiled 19](#_Toc422346771)

[Slick2d 19](#_Toc422346772)

[Gestion du projet 20](#_Toc422346773)

[Rôles des participants 20](#_Toc422346774)

[Simon Baehler 20](#_Toc422346775)

[Suivi du projet 21](#_Toc422346776)

[Bilans des itérations 21](#_Toc422346777)

[Stratégie de test 22](#_Toc422346778)

[Client 22](#_Toc422346779)

[Stratégie d’intégration du code de chaque participant 24](#_Toc422346780)

[Etat des lieux 24](#_Toc422346781)

[Côté serveur 24](#_Toc422346782)

[Côté client 24](#_Toc422346783)

[Travail restant 25](#_Toc422346784)

[Côté client 25](#_Toc422346785)

[Autocritiques 26](#_Toc422346786)

[Simon Baehler 26](#_Toc422346787)

[Armand Delessert 26](#_Toc422346788)

[Conclusions personnelles 26](#_Toc422346789)

[Simon Baehler 26](#_Toc422346790)

[Armand Delessert 26](#_Toc422346791)

[Annexes 27](#_Toc422346792)

# Introduction

Le but de ce projet de génie logiciel (GEN) est de réaliser un jeu de tank en deux dimensions en vue de dessus. Le jeu est décomposé en deux parties : un client et un serveur.

* Le client sert d’interface graphique pour le joueur. Il se connecte au serveur pour rejoindre une partie. Le joueur n’a pas besoin de compte pour se connecter au serveur. Il suffit juste de connaître l’IP de ce dernier.
* Le serveur gère la partie et la synchronisation des clients. Il permet à 2 clients de se connecter pour une partie.

Ce projet a été réalisé dans le cadre du cours GEN dispensé en 2ème année de l’HEIG-VD et d’une durée d’un semestre à raison recommandée/estimée de 6 heures de travail par personne et par semaine, par les étudiants suivants :

* Armand Delessert (chef du groupe)
* Benoit Zuckschwerdt
* Baehler Simon
* Ngueukam Djeuda Wilfried Karel

# Déroulement d’une partie

#### Démarrage de la partie :

L’admin démarre une partie depuis le serveur. Une fois la partie créée, les joueurs rejoignent le serveur en saisissant l’IP de celui-ci. Lorsque tout le monde est prêt, l’admin lance la partie.

#### Déroulement de la partie :

Au début de la partie, chaque joueur apparaît sur la carte. Tous les joueurs possèdent au départ un tank avec une barre de vie. Les joueurs peuvent se déplacer sur les parties libres de la carte (là où il n’y a pas d’obstacle) et tirer. Les déplacements sont limités aux 4 directions nord, sud, ouest, est. Les tirs sont limités à un seul obus à la fois, ce qui veut dire que lorsqu’un joueur tir un coup il ne peut tirer un second coup seulement lorsque le premier tir aura atteint un adversaire ou un obstacle. Chaque tir reçu inflige des dégâts au tank. Lorsque le tank est détruit, le joueur a perdu. Le dernier joueur en vie gagne la partie. S’il reste plus d’un joueur en vie et que le temps de jeu est écoulé, la partie se termine.

#### Les bonus :

Pendant la partie, des bonus apparaissent sur la carte et les joueurs peuvent les ramasser. Un bonus donne certains avantages au joueur tels qu’un tir plus rapide ou une arme différente.

#### Fin de la partie :

La partie prend fin dans les 3 cas suivants :

* Lorsqu’il ne reste plus qu’un joueur en jeu.
* Lorsque le temps de la partie est écoulé.
* Lorsque l’admin met fin à la partie depuis le serveur.

Dans les 2 premiers cas, la fenêtre de jeu se ferme et le tableau des scores s’affiche à l’écran de chaque joueur. Dans le troisième cas, la partie s’arrête et l’admin peut choisir de relancer une partie ou de fermer le serveur.

# Conception technique

## Architecture générale

L’application se compose d’un serveur et de 2 clients (un seul est représenté dans l’illustration qui suit).

Le diagramme bloc présentant les principaux composants utilisés dans le cadre de ce projet est le suivant :

FAIRE UN AUTRE DIAGRAMME



## Technologies utilisées

* Langage de programmation : Java
* Interface graphique des fenêtres : Swing
* Interface graphique du jeu : Slick2D
* Base de données : fichiers XML.

# Serveur

## Architecture

Le serveur est composé de 2 parties qui répondent chacune à une tâche. La première partie est le « GameManager » qui s’occupe de la gestion de la partie et la seconde partie est le couple « ClientListener » et « ClientHandler » qui s’occupe de la communication réseau avec les clients.

## Liste des classes

* Package wartanks
  + WarTanks  
    Classe générale qui démarre le serveur de jeu. Cette classe démarre également 2 clients pour les tests lors du débug du serveur.
* Package gamemanager
  + GameManager  
    Serveur de jeu. Cette classe gère le déroulement de la partie, c’est-à-dire la vérification et la synchronisation des actions des clients.
    - Package map
      * Map
      * Cell
      * GlobalCell
      * Direction
      * Movable
    - Package player
      * Player
    - Package projectile
      * Projectile
      * Bullet
    - Package stategame
      * StateGameManager
    - Package vehicle
      * Vehicle
      * Tank
* Package network  
  Ce package fait partie des dépendances externes du projet « WarTanks » (partie serveur). Il se situe dans le projet « ClientServer ».
  + Package server
    - ClientListener  
      Le « ClientListener » est le serveur qui reste à l’écoute de nouveaux clients. Lorsqu’un client se connecte, il créer un « ClientHandler » qui prendra en charge le client.
    - ClientHandler  
      Le « ClientHandler » est un serveur dédié à un client. Il s’occupe de recevoir les commandes envoyées par ce client et lui retourne les mises à jour de l’état du jeu.
  + Package client
    - Client  
      Ce client sert de test pour la communication. Il implémente le protocole de communication et simule des commandes envoyées au serveur.
  + Package protocol
    - CommunicationProtocol  
      La classe « CommunicationProtocol » regroupe toutes les méthodes nécessaires à la communication entre le client et le serveur par sérialisation des classes du package « protocol.messages ».
    - Package messages  
      Toutes ces classes seront transmises par le réseau et sont donc sérialisables.
      * InfoClient  
        Cette classe comporte plusieurs informations concernant le client telles que son adresse IP.
      * InfoPlayer  
        Cette classe comporte plusieurs informations concernant le joueur telles que son ID, son nom ou encore sa couleur.
      * Command  
        Cette classe abstraite permet de regrouper toutes les classes véhiculant une commande du client au serveur sous une même classe par l’héritage. Les classes héritant de la classe « Command » sont listées ci-dessous :
        + Movement  
          Cette classe contient la commande de déplacement du joueur (déplacement vers le haut, à droite, etc.).
        + Position  
          Cette classe contient la position du joueur.
        + Shoot  
          Cette classe informe le serveur que le joueur vient de tirer.
        + UseBonus  
          Cette classe informe le serveur que le joueur utilise un bonus.
      * TiledMapMessage  
        Cette classe contient la carte Tiled avec le design de la carte. Elle est transmise aux clients lors de l’initialisation de la partie.
      * StateGame  
        Cette classe contient tout ce qui est nécessaire pour connaître l’état de la partie. Elle contient entre-autre l’état et la position de chaque joueur, la position et la direction des tirs, la position des bonus, etc. Elle contient également le statut de la partie (en cours, terminée, temps restant, etc.).
      * Map  
        Cette classe contient la position des divers éléments de la carte. Elle est incluse dans la classe « StateGame ».
      * Message  
        Cette classe était utilisée pour le débug de la sérialisation des classes.

## Diagramme UML

## Communication réseau

### Protocole

Pour la conception du protocole réseau, nous sommes partis de l’objectif de créer un protocole très simple pour la communication client-serveur en cours de partie. Nous avions prévu d’utiliser la sérialisation de classes avec que 2 classes pour la communication en cours de partie. La première classe servira au client pour lui permettre d’envoyer les commandes au serveur. La seconde classe permettra au serveur d’envoyer l’état du jeu mis à jour aux clients. Ainsi, les clients peuvent à tout moment envoyer une commande que le joueur aurait entré et le serveur s’occupe de maintenir les clients à jour en leur envoyant régulièrement le nouvel état du jeu qui inclut les dernières actions des joueurs. Cependant, l’initialisation de la partie et des clients nécessite aussi l’envoie de plusieurs messages. Il a donc fallu créer plusieurs classes sérialisables pour permettre cette phase d’initialisation.

#### Liste des classes sérialisables

Phase d’initialisation :

* InfoClient : Demande de connexion de la part du client.
* InfoPlayer : Transmission des infos concernant le joueur.
* TiledMapMessage : Transmission du fichier de la carte aux clients.

En cours de partie :

* Command : Transmission des commandes du client au serveur.
  + Mouvement : Commandes de mouvement du joueur.
  + Position : Transmission de la position du joueur.
  + Shoot : Commande de tir.
  + UseBonus : Utilisation d’un bonus.
* StateGame : Transmission du nouvel état du jeu.

Le protocole, étape par étape :

#### Initialisation :

* Le serveur général (« ClientListener ») reste à l’écoute de nouveaux clients.
* Lorsqu’un client se connecte au serveur, ce dernier créer un serveur spécifique au client (« ClientHandler ») qui prendra en charge le client.
* Le client envoie ensuite l’objet « InfoClient » au serveur « ClientHandler ».
* Le serveur répond « OK » en cas d’acceptation du client ou « Refused » en cas de refus du client (s’il y a déjà 2 joueurs et que la partie est pleine par exemple).
* Si le serveur « ClientHandler » accepte le client, il lui envoie l’objet « InfoPlayer » contenant l’ID du joueur, son nom (imposé par le serveur, du type « Player 1 »), sa couleur, etc.

À ce moment, l’initialisation de la partie est terminée. Les clients attendent le signal « Start » de la part du serveur.

#### Début de la partie :

C’est le serveur principal (« ClientListener ») qui s’occupe de synchroniser les « ClientHandler » pour que ceux-ci envoient le signal « Start » simultanément à tous les clients.

* Le « ClientListener » attend sur les « ClientHandler » que ceux-ci aient fini la partie d’initialisation avec leur client.
* Lorsqu’un « ClientHandler » a terminé la partie d’initialisation avec son client, il le signal au « ClientListener » et se bloque.
* Lorsque tous les « ClientHandler » sont prêts, le « ClientListener » libère les « ClientHandler ».
* Une fois libérés, les « ClientHandler » envoient le signal « Start » à leur client et la partie démarre.

#### En cours de partie :

En cours de partie, la communication client-serveur se limite à l’envoie des commandes entrées par le client.

* En cours de partie, le client envoie les commandes entrées par le joueur au serveur (« ClientHandler »).
* Chaque « ClientHandler » reste à l’écoute des commandes envoyées par leur client pour les stocker dans une queue.
* Le « GameManager » (serveur gérant la partie) traite les commandes de chacun des « ClientHandler » en vérifiant leur validité et en les exécutant sur l’état du jeu puis retourne l’état du jeu mis à jour aux « ClientHandler ».
* Les « ClientHandler » envoient la mise à jour de l’état du jeu à leur client dès que celle-ci est disponible.

#### Schéma de la communication client-serveur :

Résumé de la communication client-serveur :



Communication client-serveur (en cours de partie seulement) :



# Client

Notre implémentation possède deux facettes : la partie du client et la partie du serveur. Pour le client nous avons utilisé le moteur graphique 2D nommée Skick2D ainsi qu’une foison d’images, de tuiles et de sprites car la charge de travail pour la création d’image par notre main était trop conséquente. Les différentes images ainsi que leurs sources et décrite plus bas dans la documentation.

## Architecture

Voici la liste des classes du client, le diagramme UML du client est présenté plus bas. Il n’est représenté ici que la partie sans l’infrastructure réseau.

### List des classes

**Game**

Il s’agit de la classe principale qui va lier tous les éléments du jeu. Sa tâche est principalement de créer les objets et de les initialiser. Elle gère aussi toute la partie collision, la pression de touche, et le mouvement de la caméra.

Elle s’occupe de créer et d’initialiser les autres classes comme

* Au travers de la factory
  + Le joueur
  + Les ennemies du joueur
  + Les bonus
* Les scores
* La map
* La HUD du joueur
* L’affichage de fin
  + Victoire
  + Défaite

**End**

Cette classe est la super-classe des deux sous classe qui sont appelé en fin de partie. Il s’agit d’une classe relativement simple ayant pour but d’afficher l’encan de défait ou de victoire. Et si le temps l’avait permis l’affichage des score. Ses sous classe sont

* Vitctory
* Defeat

**HUD**

Cette classe est l’observateur de la classe Player, cette dernière est notifiée quand le joueur passe sur un bonus afin de mettre à jour la barre d’action du joueur. Elle aurait aussi du afficher les point de vie du joueurs, mais faute de temps cette possibilité a été abandonnée au profit d’autre tache

**Factory**

Cette classe est utilisée par la classe Game, elle est chargée de créer les objets de jeux comme les joueurs, les ennemies et les bonus.

**Bonus**

Cette classe permet la création de bonus sur la map. Une instance de cette objet et crée quand on l’ajout dans la map mais également quand un joueur roule sur le bonus une instance de cette objet est créé dans la classe Player.

**IFightable**

L’interface Ifightable implémente nos protagonistes joueurs et ennemy, ils utilisent tous les deux une classe Score qui contient leurs score respectif. Globalement ces deux classe sont pareil à la différence près que Player est observé par la HUD crée par Game et que le joueur et représenté en bleu et Ennemy en rouge. Ce deux classe créent un objet Explosion a leurs mort.

**Bullet**

La classe bullet est la super-classe des trois différents types de munition/projectile qui sont :

* AlphaStrick
* Mine
* Laser

Ses projectiles sont décrits plus loin dans la section dédiée au bonus.

**Explosion**

Il s’agit d’une classe relativement simple, elle n’a pour tâche que de gérer l’animation des explosions qui sont appelée quand un joueur meurt ou quand le bonus AlphaStrick est lancé

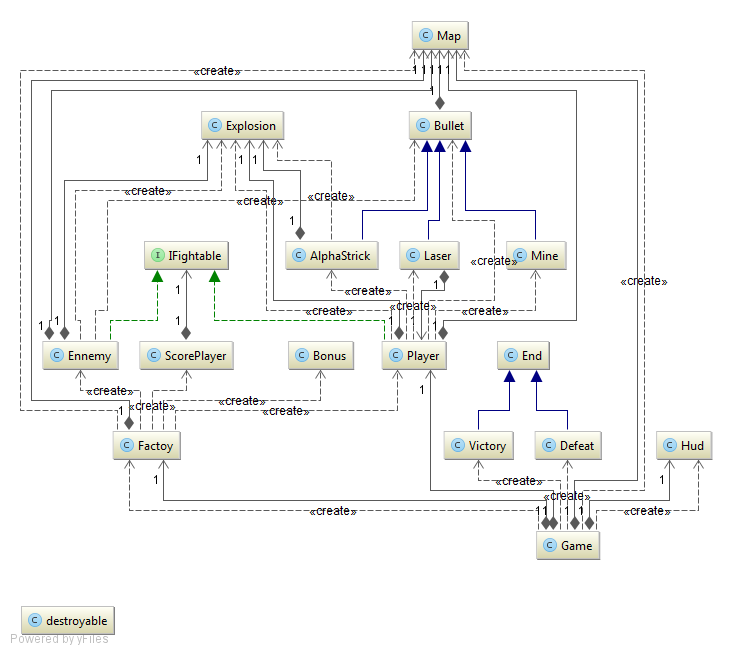
**Map**

Cette classe gère la génération de la map avec les collisions entre les bords de maps et les object-obstacles

**Destroyable**

Cette classe devait implémenter les objets destructibles, mais pour cause de manque de temps et de compétences cette classe a été inachevée.

### Diagramme UML



# Début de jeu

Idéalement avant le début de la partie le joueur aurait dû avoir la possibilité de choisir son tank. L’interface a été réalisé sur Photoshop mais n’a pas été intégré faut de temps. Un bloc de couleur par tank, dans le cas présent nous aurions eu le tank bleu.



# En jeux

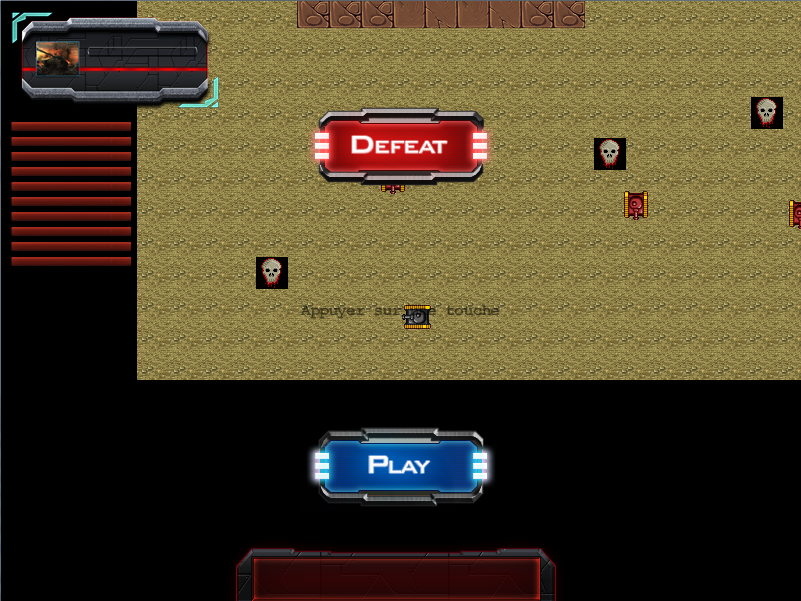
Une fois un serveur rejoint et le nombre de joueurs demandés pour commencer une partie attient, la partie peut commencer. La fenêtre de jeux dispose des éléments suivants :

* Le tank du joueur en bleu
* Le/les tanks ennemi en rouge
  + En noir si détruit
* La HUD
  + Bloc en haut à gauche
  + Barre de bonus
* Les bonus disponible sur le sol
* Les projectiles (décrit plus loin)
  + Projectile simple
  + Mine
  + Laser
  + Attaque alpha



# Fin de la partie

Lors ce que la partie se termine un affichage de fin apparait et nous ne pouvons plus rien faire. Idéalement nous aurions dû afficher le score des joueurs et avoir la possibilité de relancé la partie en cliquant sur « play ».





Les trois images ont été réalisées par les soins de Simon sur le logiciel Photoshop CS6.

# Bonus

Dans notre jeux le joueur peux ramasser des objets, dans le cas où ils ces bonus sont disponible (munition supérieur à 0) il peut les utiliser. Voici un listing des différents bonus.

## Bonus speed

* Image :C:\Users\Simon\Documents\2em\GEN\projet\WarTanks\WarTanks\WarTanksGUI\src\ressources\UI\bonus\1.bmp
* Quantité : 3
* Capacité offerte : Augmente la vitesse de déplacement du tank pour une durée de 10 secondes

## Mine

* Image : C:\Users\Simon\Documents\2em\GEN\projet\WarTanks\WarTanks\WarTanksGUI\src\ressources\UI\bonus\2.bmp
* Quantité : 3
* Capacité offert : dépose une mine sur la position courante représentée par l’image suivante C:\Users\Simon\Documents\2em\GEN\projet\WarTanks\WarTanks\WarTanksGUI\src\ressources\bullet\2.png

## Ultra Rapide Fire (non implémenté)

* Image : C:\Users\Simon\Documents\2em\GEN\projet\WarTanks\WarTanks\WarTanksGUI\src\ressources\UI\bonus\3.bmp
* Quantité 3
* Capacité offert : Permet de tirer plus rapidement. De base, le tire simple ne doit pas être enchainée

## Laser

* Image :C:\Users\Simon\Documents\2em\GEN\projet\WarTanks\WarTanks\WarTanksGUI\src\ressources\UI\bonus\4.bmp
* Quantité : 3
* Capacité offerte : Créer un laser d’une taille de 32x200 devant le joueur blessant tous les joueurs pris dans le laser. Le laser est représenté par l’image suivante :C:\Users\Simon\Documents\2em\GEN\projet\WarTanks\WarTanks\WarTanksGUI\src\ressources\bullet\laser.png

## Foudroiement (non implémenté)

* Image : C:\Users\Simon\Documents\2em\GEN\projet\WarTanks\WarTanks\WarTanksGUI\src\ressources\UI\bonus\5.bmp
* Quantité : 3
* Capacité offerte : foudroie une zone d’un rayon de 32px autour du joueur toutes les secondes durant 3 secondes, blessant les joueurs pris dans la zone

## Volée de lame

* Image :C:\Users\Simon\Documents\2em\GEN\projet\WarTanks\WarTanks\WarTanksGUI\src\ressources\UI\bonus\6.bmp
* Quantité : 3
* Capacité offerte : tire un bullet dans les quatre directions

## Bélier (non implémenté)

* Image :C:\Users\Simon\Documents\2em\GEN\projet\WarTanks\WarTanks\WarTanksGUI\src\ressources\UI\bonus\7.bmp
* Quantité : 3
* Capacité offerte : Fixe de piques sur l’avant du tank, si le joueur entre en collision avec l’avant de son tank dans les 10 secondes qui suivent l’activation du bonus, le tank adverse subit un point de dégât.

## Alpha Strick

* Image : C:\Users\Simon\Documents\2em\GEN\projet\WarTanks\WarTanks\WarTanksGUI\src\ressources\UI\bonus\8.bmp
* Quantité : 3
* Capacité offerte : Effectue une attaque sur une zone de 64x64 devant le tank, les joueurs dans la zone prennent un point de dégât. Une animation est déclenchée à l’impact.

C:\Users\Simon\Documents\2em\GEN\projet\WarTanks\WarTanks\WarTanksGUI\src\ressources\bullet\explosion1.png

## Soin

* Image : C:\Users\Simon\Documents\2em\GEN\projet\WarTanks\WarTanks\WarTanksGUI\src\ressources\UI\bonus\9.bmp
* Quantité : 3
* Capacité offerte : ajoute un point de vie au joueur

## La Mort

* Image : C:\Users\Simon\Documents\2em\GEN\projet\WarTanks\WarTanks\WarTanksGUI\src\ressources\UI\bonus\10.bmp
* Quantité : 3
* Capacité offerte : inflige un point de dégât à un joueur au hasard (possible que ça tombe sur nous).

# Fenêtre

Décrire ici la différente fenêtre et ce qu’elles permettent de faire.

# Implémentation du projet

## Technologies utilisées

### Tiled

Bien que ça ne soit pas une technologie en soie il est tout de même important de citer ce logiciel qui nous a permis de réaliser nos maps de jeux facilement et rapidement.

### Slick2d

Pour la réalisation de ce projet nous avons utilisé le moteur graphique slick2D qui permet facilement d’intégrer des maps fait par Tiled. La principale difficulté avec Slick2D fût le fait que nous ne connaissions pas et que pour un problème donné il y avait une foison de solutions différentes, nous avons dû donc naviguer entre ces différente solutions trouvées. La principale erreur avec cette technologie fut que je (Simon) suis partie sur une solution d’un tutoriel que j’ai suivie et j’ai donc patché le jeu par la suite ce qui a produit du code mal organisé et difficilement compréhensible.

# Gestion du projet

## Rôles des participants

### Simon Baehler

Graphisme : je me suis occupé de réaliser la HUD de notre jeu comme les barres de vie de joueurs, la barre d’action et les affichages de fin de jeu.

Game design : Je me suis occupé de chercher des images pour les jeux qui m’auraient pris trop de temps à faire moi-même. J’ai également imaginé les bonus des jeux et fait quelque maps.

Programmation : Uniquement la partie client, j’ai beaucoup cherché de information sur slick2D (toute la partie client).

## Suivi du projet

### Bilans des itérations

Les bilans des itérations ainsi que les bilans personnels sont disponibles en annexe.

# Stratégie de test

## Client

Durant le long du développement de la partie client des tests ont été réalisés une fois une fonctionnalité implantée. (Testé uniquement sut le client hors-réseau).

|  |  |  |
| --- | --- | --- |
| Test | Ce qui doit ce passé | Resultat /Remarques |
| Chargement d’une map | La map doit apparaitre dans la fenêtre de jeux | Ok |
| Spawn du personnage | Le personnage doit apparaitre sur la map, dans un premier temps avec des coordée fix puis aléatoire | Ok |
| Deplacement du peronnage | Le personnage doit pouvoir se déplacer vers le haut, le bas, la gauche, la droite. Sans prendre en compte les collisions avec les éléments | Ok |
| Caméra | La caméra suit le personnage s’il s’approche trop du bord de la fenêtre | Ok |
| Collinson avec obstacle | Le joueur peut entrer en collision avec un élément | La collision se faisant dans un premier temps trop tard, puis trop tôt. Désormais elle s’effectue au bon moment mais il arrive qu’on reste bloqué |
| Collision avec le bord de la map | Le joueur ne peut pas sortir de la map | Ok |
| Musique | Une musique se lance au début du jeu et boucle indéfiniment | OK |
| Soud Effect | Quand nous tirons un effet sonore se lance qui fait « boom » | OK |
| Affichage des HUD | Les hud s’affichent correctement à savoir les pv et tank ennemis, la barre d’action et l’avatar du joueur (bloc en haut à gauche) | Pas de point de vie pour le joueur -> manque de temps. |
| Spawn de bonus | Les bonus apparaissent aléatoirement sur la map | OK |
| Ramassages de bonus | Quand un joueur passe sur un bonus il le ramasse et l’ajoute dans sa barre d’action si elle n’est pas plaine | OK, Collinson pas de plus précise (2 ou 3 px de décalage) |
| Tire | Quand un joueur appuie sur « espace » il tire un projectile | OK |
| Utilisation bonus vitesse | Quand un joueur utilise le bonus « vitesse » sa vitesse augmente correctement, puis au bout de 10sec il perd sa vitesse | OK |
| Utilisation bonus Mine | Le joueur peut bien déposer une mine derrière lui | OK |
| Utilisation bonus URF | Le joueur tire plus rapidement. (3 bullet par seconde au lieu de une). Puis au bout de 10 secs le perd | NON IMPLEMENTE |
| Utilisation bonus laser | Créer un laser d’une taille de 32x200 devant le joueur blessant tous les joueurs pris dans le laser | Il arrive que le laser touche de joueur ne devant pas être pris dans le laser |
| Utilisation bonus foudroiement | Foudroie une zone d’un rayon de 32px autour du joueur toutes les secondes durant 3 secondes, blessant les joueurs pris dans la zone | NON IMPLEMENTE |
| Utilisation bonus Volée de lame | tire un bullet dans les quatre directions | OK |
| Utilisation bonus bélier | Fixe de piques sur l’avant du tank, si le joueur entre en collision avec l’avant de son tank dans les 10 secondes qui suivent l’activation du bonus, le tank adverse subit un point de dégât. | NON IMPLEMENTE |
| Utilisation bonus alphaStick | Effectue une attaque sur une zone de 64x64 devant le tank, les joueurs dans la zone prennent un point de dégât. Une animation est déclenchée à l’impact. | OK, peut-être un peu trop puissant |
| Utilisation bonus soin | ajoute un point de vie au joueur | OK |
| Utilisation bonus mort | inflige un point de dégât à un joueur au hasard (possible que ça tombe sur nous) | OK |
| Décrémentation du nombre de munition | Quand nous utilisions X fois un bonus il n’est plus disponible | OK, par contre il n’y a aucune information sur le nombre de munition, il aurait aussi été bien de pouvoir régler le nombre de munition reçus lors du ramassage. |
| Animation | Quand un joueur meurt ou que l’Alpha Strick est lancé une animation d’explosion se lance | OK |
| Fin de la partie | Quand le joueur meurs ou que tous les ennemis sont mort un écran de fin de partie apparait | OK |

# Stratégie d’intégration du code de chaque participant

Nous avons utilisé Git pour le partage de code et un groupe WhatsApp pour une communication plus rapide.

# Etat des lieux

## Côté serveur

* Initialement
  + Choix du nombre de joueurs
  + Choix de la carte
  + Choix du temps de la partie
* Finalement
  + Le nombre de joueur est fixe
    - Il est néanmoins possible de changer ça directement dans le code
  + La carte est fixe
    - Il est néanmoins possible de changer ça directement dans le code
  + Il n’y a pas de temps limite de jeux
    - Rien de mis en place à ce niveau la

## Côté client

* Initialement
  + Choix le serveur sur lequel se connecter
    - En tant que joueur
    - En tant qu’observateur de la partie
  + Si joueur
    - Avant le début de partie
      * Choisir son tank ayant de caractéristiques différentes
        + Point de vie
        + Vitesse
        + Couleur
    - Durant la partie
      * Tirer
      * Se déplacer
      * Ramasser des bonus
      * Utiliser les bonus
    - A la fin de la partie
      * Consulter les scores (automatique)
      * Relancé la partie
* Finalement
  + Choix le serveur sur lequel se connecter
    - En tant que joueur
  + Si joueur
    - Avant le début de partie
      * -
    - Durant la partie
      * Tirer
      * Se déplacer
      * Ramasser des bonus
      * Utiliser les bonus
    - A la fin de la partie
      * Consulter les scores (automatique)
      * Relancé la partie

# Travail restant

## Côté client

* Début de la partie : ajoute la possibilité de choisir son tank, soit par la pression de touche soit par clique. Ayant trouvé un tutoriel sur la réalisation d’un écran de démarrage cela devrait prendre environ 3h de travail.
* En cours de partie : réaliser les bonus non réalisé comme le bélier et le foudroiement et le URF. Le foudroiement et le bélier semble assez complexe à implémenté ce qui n’est pas le cas du URF. Un temps de travail de 5h pour l’ensemble de bonus devrait jouer.
* Fin de partie : Affichage des scores en fin de partie devrais être fait rapidement car une structure est déjà existant, il ne reste plus qu’a réalisé qui a touché qui et l’afficher. 3h devrait être suffisant. Pour ce qui en est du relancement d’une partie il s’agit ici de la grande inconnue, en effet des recherches d’implémentation de zone cliquable ont été effectuées, malheureusement sans succès.

# Autocritiques

## Simon Baehler

Comme dit plus loin dans la conclusion pour moi ce projet un fiasco. Mon erreur principale fut de partir coder la partie client sans que la partie serveur ne suive, il y a eu un décalage entre les deux parties et le merge trop dure. Mon erreur fut également d’avoir « sur couché » le code réalisé durant le tutoriel au lieu de, une foi l’outil slick2D connu, refaire un projet depuis 0 avec une implémentation réfléchie et basé sur des cas d’utilisation, diagramme d’activité etc…

## Armand Delessert

# Conclusions personnelles

## Simon Baehler

Ce projet se résume à un mot : fiasco. Durant tout le long du projet il n’y a eu aucune communication, dès les 1ères itérations la partie réseau a pris du retard. Le rendu des bilans d’itération personnelle toujours en retard des fois même de plusieurs semaines, et qui plus pour la plupart du temps minimaliste. Un investissement faible de membre du groupe et un partage totalement inégal du travail et un laxisme de plus énorme.

## Armand Delessert

Il s’agit de la première fois que je m’attaque à une communication client-serveur complète. Bien que j’aie apprécié développer cette partie, ce n’était pas une tâche facile tant en terme de conception et développement qu’en terme de débug. Malgré tout j’ai trouvé ce chalenge très intéressant et très enrichissant. Je suis certes déçu de ne pas avoir pu continuer cette partie jusqu’au bout mais je suis tout de même fière du travail réalisé.

# Annexes

Bilans