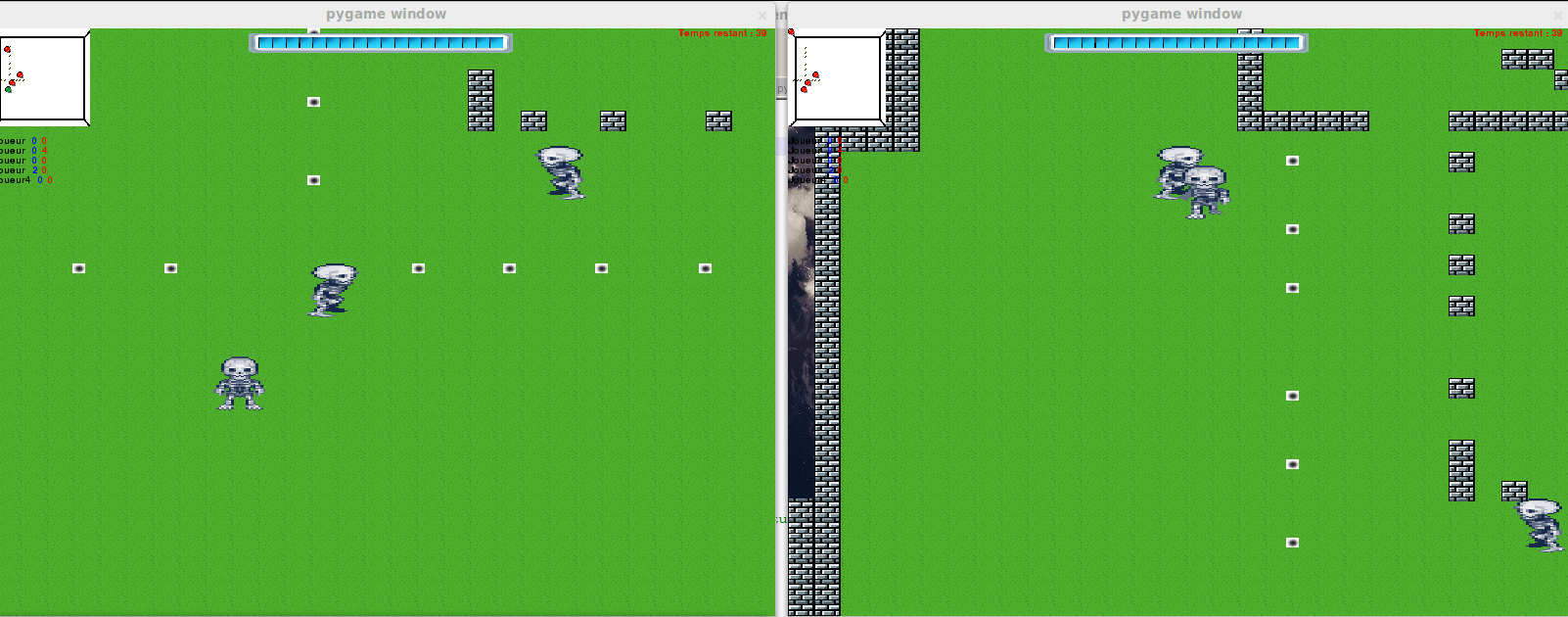
Projet d’Informatique et des Sciences du Numérique:

**Super Mortal Combat Fight: Extreme gun edition**



Equipe composée de :

MOUREY Thibaud

PHAN Tony

WESTPHAL Raphaël

TS3

Année 2012/2013

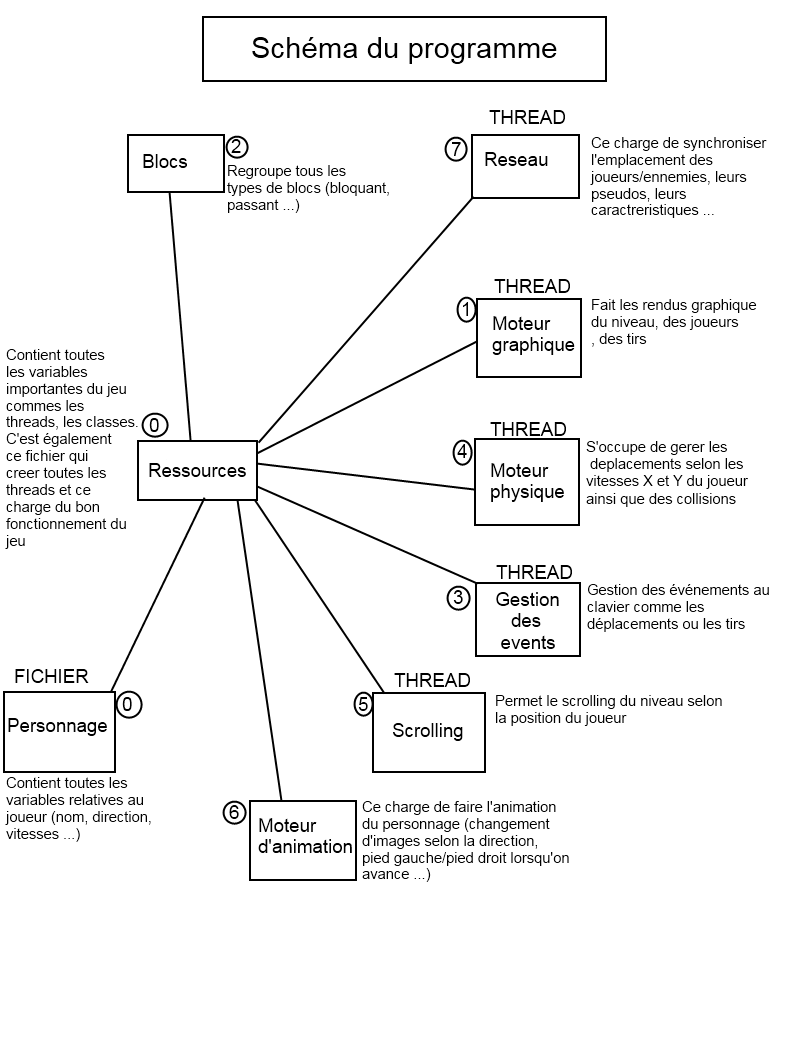
Lycée Marcel Rudloff

Ce dossier se décompose en 2 parties, la première est l’explication de la partie client, la seconde est l’explication de la partie serveur.

But du jeu: L’objectif du jeu est de toucher (et donc de tuer) un maximum de fois les autres joueurs avant la fin de la manche tout en évitant de se faire tirer dessus. On peut classer ce jeu dans la catégorie des beat’em all multi-joueurs.

**Partie client:**

La partie client se décompose en plusieurs fichiers et plusieurs threads, cependant elles se rattachent toutes au même fichier, le fichier “res.py” (diminutif de ressources)



Ce schéma se lit d’une certaine manière: Tout d’abord, chaque case est un fichier différent. Ensuite il y est lié sa description et son ordre de programmation (c’est à dire quand est-ce qu’il a été programmé par rapport aux autres fichiers.)

**Ressources:**

Le fichier ressources contient toutes les variables “importantes” qui ont besoin d’être accessibles par les différents threads et fonctions. Il était donc évident de le créer en premier. Comme dit précédemment, il contient toutes les threads (chaque thread doit être stockée dans une variable pour pouvoir la lancer). Nous avons choisis de créer ce “super fichier” pour que les autres fichiers ou thread soient un peu comme des “modules” que l’on peut programmer à part et inclure facilement dans le projet. Ceci est certainement la meilleure architecture pour la programmation d’un projet à plusieurs, car chacun peut modifier le code à sa guise et ajouter ses nouvelles fonctionnalités sans bloquer la programmation des autres personnes.

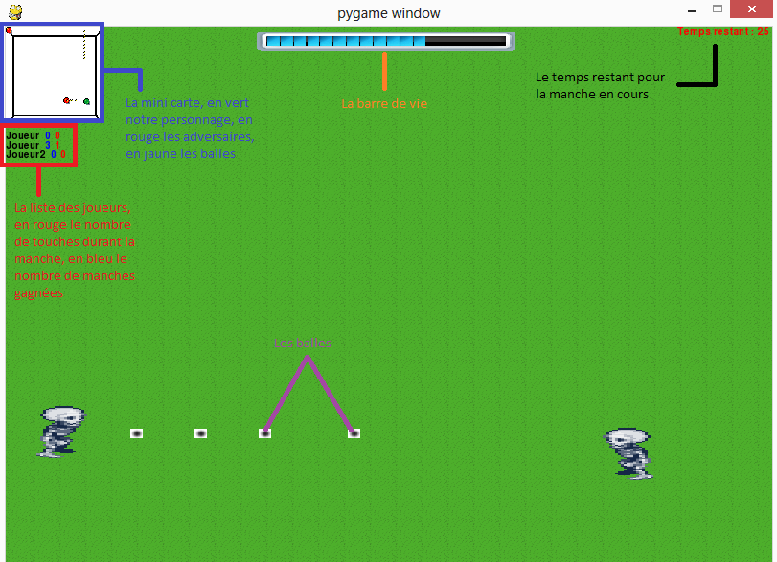
**Personnage:**

Ce fichier contient une classe personnage. C’est à l’intérieure de celle-ci qu’on trouve les variables de positions X et Y, de position dans le monde (expliquée dans la partie Moteur Graphique), ainsi que les vitesses de déplacement (car oui, on dispose d’un “vrai” petit moteur physique), le pseudo par défaut du joueur, son score et pour terminer les points de vie restant. Toutes ces variables ne sont pas utilisées directement dans cette classe, mais principalement dans les autres. Seule la fonction de gestion des touches de déplacement du personnage utilise ces variables. Justement, cette fonction attribue la bonne vitesse sur le bon axe selon la touche appuyée

**Moteur graphique:**

Le moteur graphique est l’une des parties les plus importantes du programme. C’est lui qui permet l’affichage de la carte, du personnage, de ses tirs ainsi que de l’interface. C’est un thread s’exécutant indépendamment du reste du code, grâce à cela même si une autre partie du code venait à ralentir (par exemple le moteur physique ou la partie réseau) le jeu ne se bloque pas entièrement et la partie peut continuer malgré les ralentissements. Cependant le moteur graphique ne comporte pas que la fonction principale d’affichage, mais également des fonctions de calcul “pur” car dans notre choix de faire un niveau plus grand que l’écran, et aussi de pouvoir adapter notre programme à toutes les résolutions (ce qui je pense a été le principal facteur de difficulté) nous avons dû créer plusieurs référentiel. A la base censé être trois, nous nous sommes réduis à deux. Le premier est le référentiel du personnage à l’écran (les variables de position du personnage sont res.character.x et res.character.y), c’est à cette position que notre personnage est affiché à l’écran. Le second référentiel est celui du personnage sur la carte (donc y compris sur ce qui n’est pas affiché à l’écran). Il fallait donc pouvoir convertir la position de notre personnage dans le monde par rapport à la position de la camera pour l’afficher à l’écran. C’est ce que font les fonctions “CalculerPositionJoueur”, “CalculerPositionJoueurMulti”, et “CalculerConstante”.

**Interface :**



L'interface (le HUD ou affichage tête haute en français) est composée de plusieurs images mises dans des variables puis redimensionnées et enfin affichées sur l'écran.

**La mini carte :**

Dans la mini carte est affichée; elle même, la position de son personnage, la position des ennemis et la position des balles. Ces sprites sont affichés par rapport à leurs positions réelles sur la carte, la taille du niveau et la taille de la mini carte.

La barre de vie quant à elle, se modifie en fonction de la vie restante du joueur. La vie du joueur est calculée du côté du serveur puis envoyée vers le client pour directement actualiser la vie du joueur avec la variable “vie” (ici “res.character.vie” car la variable ne se trouve pas dans le même fichier)

Pour les scores, le chrono et les pseudos, il a fallu modifier le contenu des variables en sprites pour ensuite les afficher à l’écran. Ces trois éléments sont tous gérer du côté serveur, les pseudos sont récupérés lorsque le client se connecte, les scores sont modifiés lorsqu’une balle touche un autre joueur et le chrono est initialisé lorsque les joueurs sont connectés entre eux.

**Le scrolling :**

Nous avons décidé de pouvoir jouer sur de grandes cartes, mais pour des questions de jouabilité, seulement une petite partie du niveau est affichée. Pour se déplacer à l’intérieur, un scrolling (un déplacement de l’écran sur la carte) était nécessaire.

Pour faire le scrolling de l’écran, on regarde où se trouve le personnage dans le niveau et on vérifie s’il ne s’approche pas trop près des bordures de l’écran. Si c’est le cas, on déplace l’écran pour afficher une nouvelle zone du niveau.

**Le moteur physique :**

**Les déplacements :**

Les déplacements du personnage sont directement reliés au fichier personnage. D’abord l’utilisateur appuie sur une flèche de direction, qui va modifier la vitesse de déplacement sur l’axe X et/ou Y dans le fichier personnage. Dans le thread du moteur physique, on va vérifier que la vitesse de déplacement est supérieure à 0 (donc on vérifie si une touche de direction est enfoncée) Si c’est le cas, on déplace le personnage du nombre de case correspondant à sa vitesse. Il est donc possible de se déplacer en diagonale si nous le souhaitons, contrairement à une gestion basique ou l’on ne fait que déplacer le personnage lorsque nous appuyons sur la touche. Le second intérêt ici est de pouvoir, dans le futur, rajouter d’autres déplacements sur de nouveaux objets (comme par exemple, un nuage qui avancerait tout seul dans le ciel comme dans le jeu Super Mario)

**Les collisions :**

Un niveau est composé d’un certain nombre de blocs d’herbes et de murs. Dans le fichier collisions, nous avons défini qu’une case comportant un mur est bloquant et que s’il n’y a pas de mur, le bloc est non bloquant.

Après cela, nous avons défini à quel moment le joueur touche un bloc bloquant ou non, pour cela, nous avons créé des points tout autour du sprite du personnage avec lesquels nous vérifions que les bords du personnage ne se trouvent pas sur une case définie comme bloquante. Si le bloc est bien bloquant, on fait reculer le personnage, cependant, le personnage recule instantanément juste après avoir avancé, ce qui fait qu’on ne peut pas voir ce mouvement et le personnage aura l’air bloqué.

**La partie réseau :**

Le jeu étant multi-joueurs, il possède une partie réseau. Cette partie est décomposée en 2 parties, celle de l’envoi et de la réception. Une thread indépendante est créée pour la gestion des envois. Cependant le rôle de cette thread ce résume à envoyer continuellement les positions (dans le monde) du personnage au serveur. Il ne faut pas croire que les envois du programme au serveur ne se résument qu’à cela, mais tout au long du programme et dans les différents fichiers sont implémentés des envois répondant aux besoins immédiats de la fonction. Ils sont détaillés dans la partie réception de la partie serveur. La seconde partie, celle de la réception traite plusieurs cas:

-La réception d’une commande de déplacement d’un des autres personnages sur la carte.

-La réception d’une nouvelle position de balle

-Un changement dans notre vie

-Un changement de score (le nombre de fois où il a touché un autre joueur)

-Un changement de pseudo

-Un changement de chrono (le temps est géré par le serveur, après chaque seconde le temps est actualisé)

-Un changement de score (le nombre de parties gagnées)

**Le moteur d’animation :**

Le moteur d’animation se charge de faire l’animation des personnages à l’écran. Par animation, il est entendu

le changement de “pied” lorsque le personnage est en mouvement et cela dans le but de donner l’impression que le personnage marche. Nous avons implémenté un système de nommage spécial des sprites de façon à ce que le chargement des sprites du personnage soit simplifié et qu’ainsi nous puissions mettre à disposition du joueurs plusieurs skin de personnages. Le nom des sprites est décomposé de la manière suivante :

Premier chiffre : Le numéro du personnage

Deuxième chiffre: L’orientation du personnage: 0 Haut, 1 Bas, 2 Gauche, 3 Droite

Troisième chiffre: Le numéro d’animation du personnage: 0 A l’arret ,1 Pied gauche, 2 Pied droite

Par exemple, pour un personnage numéro 1, regardant vers le haut et avec le pied droit en avant, nous avons: 1-0-2.png

**Le Menu :**



Au lancement du jeu un menu s’affiche, c’est ici que tous les paramètres du jeu sont réglés.

Sur ce menu principal, on a donc accès à 2 sous-menus (Se connecter, Options) ainsi qu’une page d’aide et de crédits.

Cette partie du programme se concentre essentiellement sur l’affichage d’image, de texte, et la gestion d’événements au clavier et à la souris.

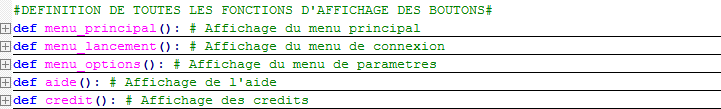
La programmation s’est faite sur 3 sous-parties différentes :

- La Boucle Principale appelant toutes les fonctions permettant l’affichage complet

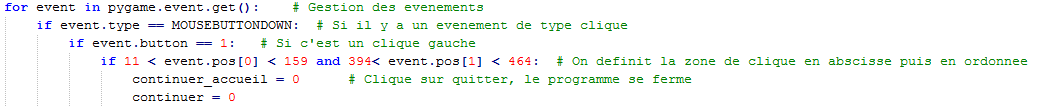
- Les fonctions d’affichage des boutons

- Les fonctions prenant en charge la gestion d’événements du clavier dans les boites de dialogues

Tout d’abord il a fallu faire afficher les boutons de chaque menu, pour cela nous avons créé plusieurs fonctions d’affichage pour chaque menu que nous appellerons par la suite.



Ensuite il a fallu mettre en place la gestion des événements de la souris, ainsi lors d’un clique sur une zone prédéfinie soi-même on peut effectuer une certaine action, par exemple :



Puis, il a fallu faire appel à un module intégré de Pygame pour ajouter de la musique en arrière-plan, c’est le module Mixer. Il suffit d’ajouter un fichier audio à une playlist puis de contrôler la lecture de celle-ci.

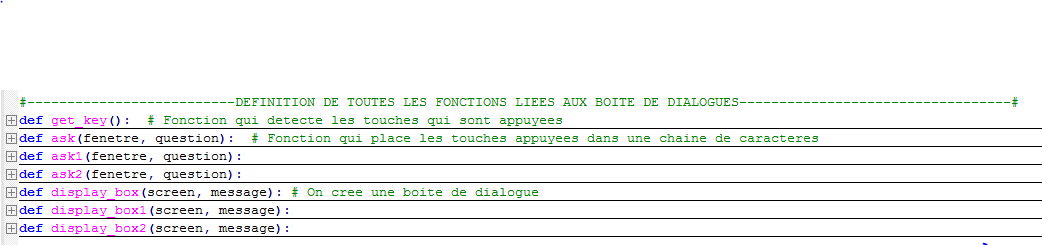


De même, pour pouvoir personnaliser son pseudo, rentrer l’adresse ip du serveur, le port utilisé, il nous fallait une boite de dialogue directement dans l’interface graphique. N’ayant pas de module intégré, il a fallu créer 3 fonctions :

- une fonction pour la gestion des événements au clavier

- une fonction pour l’écriture du texte

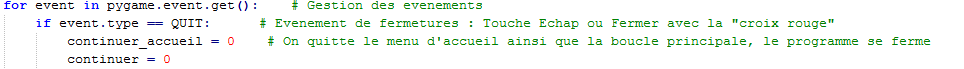
- une fonction pour dessiner les boîtes de dialogues



Nous avons également ajouté d’autres fonctionnalités à ce menu, tel que le passage du mode plein-écran au mode fenêtré ainsi que l’inverse, la mise en pause ou non de la musique d’arrière-plan, un menu pour changer le skin du personnage joué.

Certaines fonctionnalités ont été testées mais non finalisés tel que le changement de résolution de la fenêtre, le personnage qui peut tourner sur lui-même dans le menu de choix de skin.

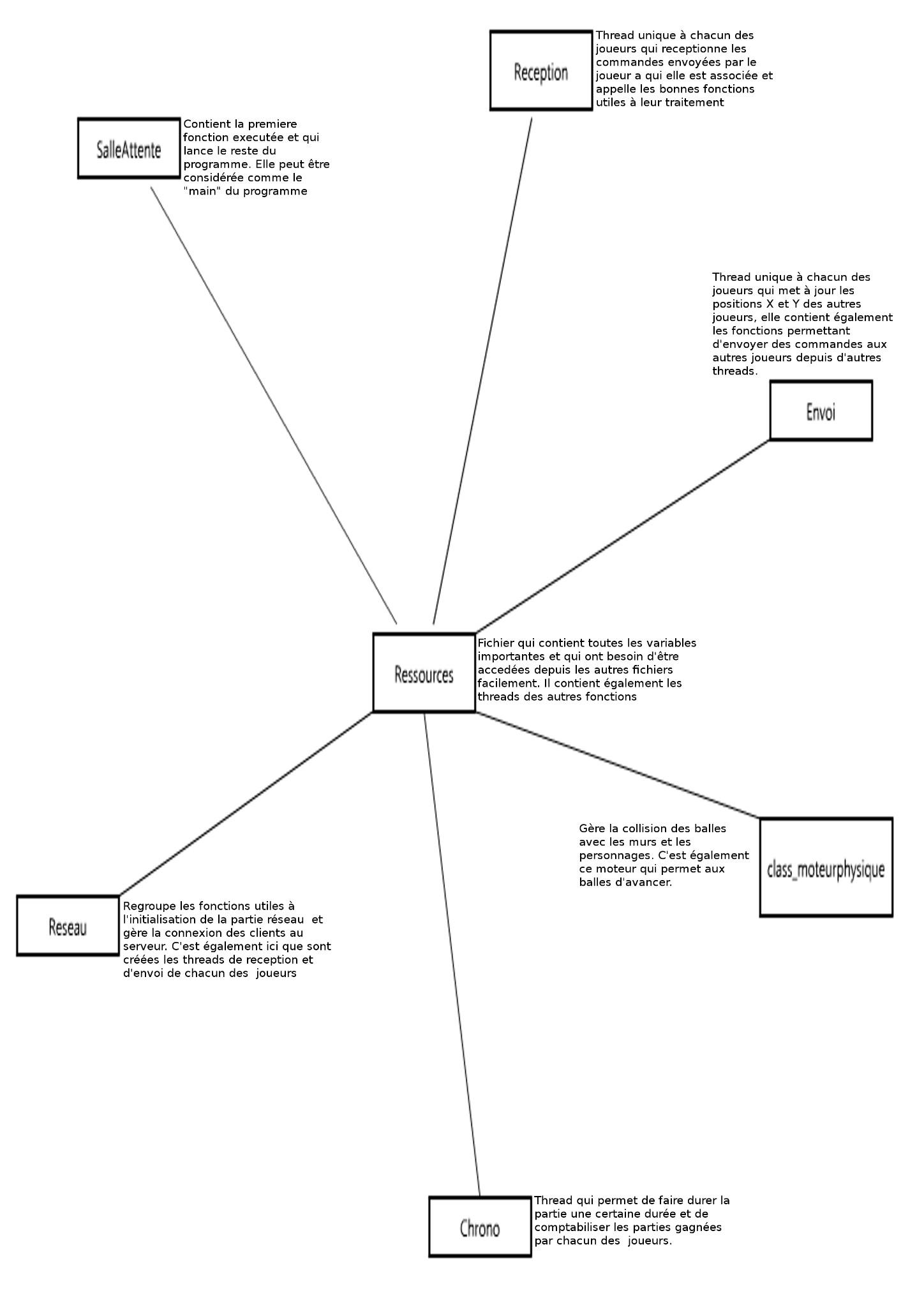
Et le dernier élément du menu, la gestion d’événements de fermeture du programme a dû également être ajoutée. Il permet de sortir de la boucle principale du menu et donc de fermer le programme de manière propre.



**Partie serveur:**

La partie serveur a été faite seulement dans un second temps, après avoir programmé une grosse partie du client. L’architecture du serveur est basée sur celle utilisée par le client

Le serveur ce décompose en 3 parties majeurs, la réception, l’envoi, et le moteur physique.



**La salle d’attente:**

Cette fonction est l’équivalent de la fonction “main”. C’est elle qui lance les “grandes fonctions” du programme, qui sont d’initialiser la partie réseau, de permettre la connexion des joueurs au serveur, de lancer le moteur physique et le chrono.

**Réception:**

La fonction Reception est une thread créée pour chaque joueur. Sa fonction est de réceptionner les commandes envoyées par le client et de les traiter. Les commandes recevables sont :

-Une mise à jour de la position du joueur dans le monde

-Création d’une nouvelle balle

La création d’une nouvelle balle est le fait de mettre en mémoire la position du joueur ainsi que la direction du tir. Le traitement (faire avancer la balle ainsi que les collisions) est traité automatiquement par le moteur physique.

**Envoi:**

Tout comme la fonction de Réception, une thread d’envoi est créée par joueur connecté. La seule fonction de la fonction principale est de mettre à jour les positions des autres joueurs chez le client connecté. Cependant ceci n’est pas la seule donnée envoyée au client, bien au contraire. Le reste est envoyé par les autres threads de façon automatique. Par exemple, lorsque le moteur physique a fini de calculer la position et les collisions des balles c’est lui qui va envoyer les coordonnées aux joueurs et cela grâce à la fonction BroadcastMessage et EnvoyerMessage. Justement, ces fonctions sont celles qui envoient le message au joueur concerné, ou justement à tous les joueurs (pour la fonction BroadcastMessage)

**Moteur physique:**

Le moteur physique est une thread unique qui gère le déplacement des balles et les collisions balles/murs et balles/personnages. C’est également lui qui gère la vie des personnages.

**Chrono:**

Le chrono est une thread unique qui a pour unique compte de faire un décompte du temps restant de la

manche, et de rajouter un point au gagnant de la manche.

**Réseau:**

Ce dernier fichier créait la connexion des clients au serveur en créant les threads d’envoi et de réception de chaque joueur. C’est également lui qui initialise la partie réseau du serveur.

**Conclusion:**

Pour conclure ce dossier, nous sommes plutôt contents et fier du résultat final même si nous n’avons pas pu atteindre notre objectif initial qui était de réaliser un standalone 2D d’un mode de jeu Kodotag du jeu Warcraft 3. Cependant le jeu est jouable et est surtout multijoueurs ce qui lui permet de le rendre attrayant et lui offre un potentiel de rejouabilité important. De plus nous comptons continuer la programmation de ce jeu et cela même après notre Baccalauréat.

Pour ce qui est du langage de programmation, nous avons trouvé que python est un langage très “productif” dans le sens où nous avons besoin de faire “peu” pour produire “beaucoup” contrairement t à d’autres langages de bas-niveau. S’ajoute à cela que c’est un langage “agréable” à écrire car il offre une syntaxe très lisible.

Sources :

Images libre de droits prisent sur <http://browserquest.mozilla.org/>

Musique tirée de <http://www.youtube.com/watch?v=ETzYUgT4qbg>, sous licence Creative Commons