UNIVERSITÉ
CAEN
NORMANDIE

ADVANCED TETRIS

CAJEAT Romain - 21808338

DEMOLINIS Rémy - 21702827

GOURNAY Alexandre - 21806749

L1 Informatique - Groupe 1B

Rapport Conception Logiciel

2018-2019

Table des matières

1 Objectifs du projet

1.1 Description du concept

Le concept était simple, tout d'abord il fallait réaliser un tetris fonctionnel reprenant le principe de base du célèbre jeu, en ajoutant des fonctionnalités comme la pièce fantôme et la visualisation de la pièce suivante, pour ensuite faire un mode multijoueur local contre joueur(s) et/ou IA, en finissant si possible par ajouter une couche réseau pour jouer en ligne.

1.2 Ce que nous avions à faire

Faire un tetris, à l'aide de pygame en ajoutant quelques fonctionnalités personnelles, comme le fais de pouvoir choisir un thème de jeu ou encore de pouvoir changer ses touches, mais aussi quelques fonctionnalités demandées, ainsi qu'un multijoueurs local (pas d'IA car nous trouvions cela inutile de jouer contre, autant jouer en solo), et nous avons décidé de tenter la couche réseau pour pouvoir jouer en ligne.

1.3 Ce qui existe déjà

Le principe du tetris existe déjà dans le jeu sortie en 1984, ainsi que plein d'autres versions sorties depuis. Il existe aussi des codes de tetris déjà fait sur internet, mais l'intérêt de les reprendre est assez minime, le but étant de le faire nous mêmes pour savoir comment cela fonctionne, mais aussi savoir quoi modifier et comment lorsque nous rencontrons un problème.

Logo du jeu original.

2 Fonctionnalités implémentées

2.1 Description des fonctionnalités

Nous avons décidé d'implémenter les fonctionnalités suivantes :

➠ **La visualisation de la pièce suivante :** dans le cadre des informations du joueur (avec le nom du joueur et le score) la pièce suivante est affichée à chaque nouvelle pièce.

➠ **La pièce fantôme :** un "fantôme" de la pièce actuelle est affiché à l'endroit ou celle-ci est censé atterrir en fonction des mouvements et rotations de la pièce actuelle.

➠ **Vitesse progressive :** la vitesse du jeu augmente en fonction du temps de la partie.

➠ **Le mode LOCAL :** nous avons choisi de faire un mode local de 1 à 4 joueurs sur le même écran et clavier, sans IA car nous avons jugé cela inutile étant donné que l'on peut jouer en solo.

➠ **X-line :** quand un joueur fini une ligne, une ligne de "X" apparaît chez le (ou les) joueur(s) adverse(s), sur la ligne la plus basse.

➠ **Le mode ONLINE :** nous avons aussi choisi de faire le mode online de 2 à 4 joueurs sur écrans différents (c'est le principe du online...), qui fonctionne sur le réseau de l'université, mais comme les ports sont bloqués, nous pouvons jouer uniquement sur les ordinateurs à notre disposition en salle de TP, même si cela fonctionne sur des machines extérieurs lorsque l'on héberge sur un serveur extérieur.

➠ **Chat en réseau :** avec le mode ONLINE nous avons aussi fait un chat pour permettre aux joueurs de discuter entre eux, pendant une partie.

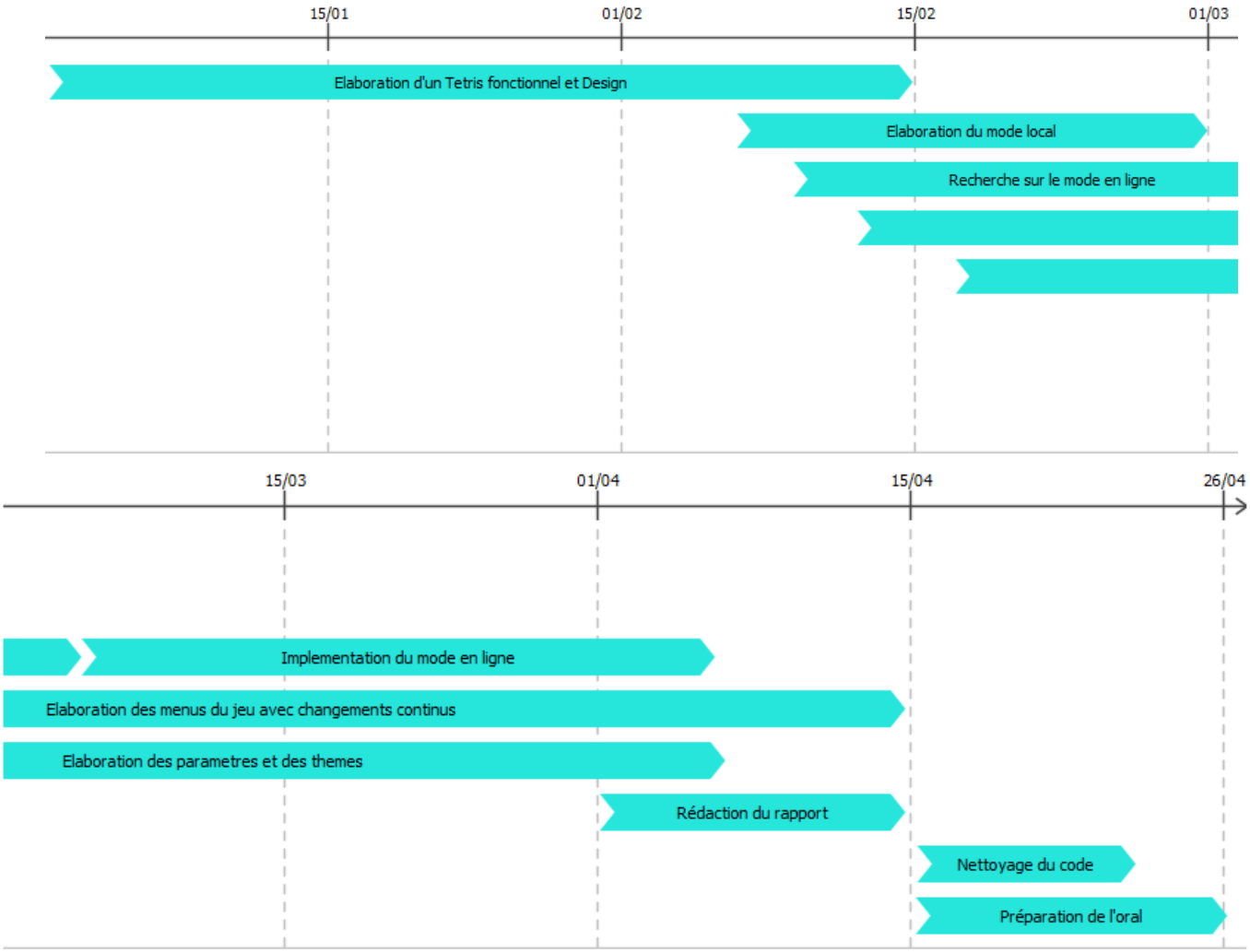
➠ **Un menu fonctionnel et simple d'utilisation :** Nous avons fait un menu agréable à utiliser qui guide simplement les utilisateurs, et qui permet de choisir son mode de jeu (local ou online), ainsi en choisissant le mode local on peut choisir de jouer entre 1 et 4 joueurs, et en choisissant le mode online on peut choisir de jouer entre 2 et 4 joueurs en ligne. Si au départ dans le menu l'on choisi l'onglet paramètres (le rouage), nous avons alors le choix d'aller dans les paramètres des touches ou des thèmes.

➠ **Changement de touches de jeu :** en allant dans les paramètres de touches, nous pouvons à notre guise changer les touches de chaque joueurs en appuyant sur "ENTRÉE" puis sur la touche souhaitée (celles-ci prendront effet en partie local, alors que celles du joueur1 fonctionneront aussi pour le online). Pour quitter ce menu il suffit d'appuyer sur la touche "BACKSPACE" / "RETOUR ARR.".

➠ **Changement de thème de jeu :** en allant dans les paramètres de thèmes, nous pouvons changer le thème de jeu, effectivement quand nous arrivons dans le menu thèmes, une preview du thèmes classique (son + image des bloques en jeu) s'active, ainsi nous pouvons naviguer dans les différents thèmes en regardant et écoutant les previews de chacun pour choisir son thème (parmi ceux disponible). Pour choisir un thème, il suffit d'appuyer sur la touche "ENTRÉE" sur le thème souhaité.

2.2 Organisation du projet

2.2.1 Diagramme de Gantt



2.2.2 Répartition des tâches

Pour ne pas perdre de temps, nous avons beaucoup travaillé en parallèle, en nous répartissant le travail à faire, en fonction de nos compétences personnelles.
➪ Élaboration du tetris de base : Romain, Alexandre & Rémy
➪ Sur la partie local/ONLINE : Romain & Alexandre
➪ Élaboration d'un chat réseau (partie graphique) : Alexandre
➪ Sur la menu/paramètres (partie graphique et fonctionnelle) : Rémy

3 Éléments techniques

3.1 Algorithmes

Voici la liste des fichiers ainsi qu'une description de leur rôle :

main.py : Ce fichier contient l'entrée du programme. Celui-ci crée les menus et le jeu puis l'affiche.

grid.py : Ce fichier contient les classes Board et BoardParameters.

- Board : Le jeu Tetris de base. Voici les fonctions principales :
  - handleDisplay(self) : Affiche la grille de jeu.
  - handleDropAndLateral(self) : Se charge de faire tomber la pièce et de faire vérifier si le joueur à complété une ligne.
  - handlePlayerInput(self, type, key) : S'occupe des entrées du joueur.
  - displayUserInfo(self, x, y) : Affiche le panneau d'information du joueur.
  - can place(self, grid, block, y, x) : Vérifie si une pièce peut être placée.
  - canMove(self, direction) : Vérifie si l'utilisateur peut effectuer son coup.
- BoardParameters : Classe permettant de configurer la classe Board. Les paramètres du constructeur ont des valeurs par défaut pour simplifier l'utilisateur. Voici la liste des options disponibles :
  - game block width : Largeur en pixels d'une unité/case de la grille.
  - game size x : Nombre de colonnes.
  - game size y : Nombre de lignes.
  - game x offset : Position en x du plateau de jeu.
  - game y offset : Position en y du plateau de jeu.
  - game grid thickness : Épaisseur en pixels des lignes de la grille.
  - game delay ms : Délai en millisecondes de décente du bloc.
  - game delay ms fast : Délai en millisecondes de décente rapide du bloc.
  - game userinfo x : Position en x du panneau d'informations utilisateur.
  - game userinfo y : Position en y du panneau d'informations utilisateur.
  - game bckgd color : Couleur du fond du jeu.

chat interface.py : Ce fichier contient le code nécessaire au chat textuel du jeu.

- Chat : Classe permettant d'afficher la fenêtre du chat. Comprenant les paramètres suivants :
  - name : Nom du joueur local.
  - color : Couleur du joueur local.
  - x : Position en x de la fenêtre du chat.
  - y : Position en y de le fenêtre du chat.
  - w : Largeur de la fenêtre du chat.
  - h : Hauteur de la fenêtre du chat.
- run chat(username : str, input queue : queue, output queue : queue, set chat) : Thread qui crée la fenêtre de chat et qui lui envoie les données pour les afficher.

network.py : Ce fichier contient le code permettant de communiquer avec le serveur.

- DataBlock : Classe contenant les informations de jeu transmises au serveur, tel que le nom du joueur, son score, sa grille de jeu et cetera.
- comm thread recv(ip : str, port : int, username : str, input queue : queue, conn status : list) : Thread qui reçoit les informations provenant du serveur.
- comm thread send(ip : str, port : int, username : str, output queue : queue, conn status : list) : Thread qui envoie les informations au serveur.

parameters.py : Ce fichier contient les menus du jeu.
— Touches : Classe qui se charge d'afficher le menu de configuration des touches.
— Themes : Classe qui se charge d'afficher le menu de sélection du thème.

player.py : Ce fichier contient le code du joueur.
— Player : Classe qui représente un joueur et contenant son pseudo, score, lignes bloc actuel et bloc suivant.
— RemotePlayer : Classe qui hérite de Player et qui est utilisé pour le multijoueur en ligne.

3.2 Structures de données

Voici la liste des classes permettant de grouper des données afin de permettre une utilisation du code plus propre et facile.

— Pour la création du plateau de jeu Board, la classe BoardParameters est disponible. celle-ci permet de changer les paramètres du plateau de jeu tel que le nombre de cases en largeur, hauteur, la taille des carreaux, l'épaisseur des lignes de la grille et cetera. Les paramètres du constructeur de BoardParameters ayant tous des valeurs par défaut, l'utilisation est simple.
— Pour la communication des données de jeu, le jeu utilise une Class DataBlock qui est sérialisée avec du JSON puis transmise. JSON est un format d'échange de données qui est facilement lisible par un humain. La classe DataBlock est donc constituées des attributs suivants : le nom du joueur, son score, son nombre de lignes effectuées et son plateau de jeu.

Voici la structure pour le dossier ressources contenant les données d'affichage et les thèmes.

3.3 Bibliothèques

Afin de mener à bien notre projet et pour ne pas réinventer la roue, nous avons utilisé des bibliothèques existantes. On distinguera dans notre cas deux types de bibliothèques, les bibliothèques principales avec lesquelles nous avons développé une grande partie de notre projet tel que `pygame` et les bibliothèques spécifiques que nous n'avons utilisé que dans certaines parties de notre projet tel que `socket`, `queue` et bien d'autres.

3.3.1 Les Bibliothèques principales

La première bibliothèque que nous avons utilisé et qui nous a permit de développer une grande partie de notre programme est `pygame`. Il s'agit d'une bibliothèque qui permet de faciliter le développement de jeux vidéo avec le langage de programmation Python. Elle nous a permit de créer notre fenêtre principale et de manipuler les nombreux sprites qui composent notre jeu. Nous avons notamment importé des constantes utilisées par Pygame grâce à l'instruction `from pygame.locals import *`.

Une grande partie de notre jeu est le multijoueur en réseau, qui a été réalisé avec succès grâce aux bibliothèques `threading`, `socket` et `queue` (voir partie sur le réseau).

La bibliothèque `threading` permet d'exécuter simultanément plusieurs threads (tâche, appel de fonction). Dans notre cas ceci nous a permit de séparer lors du multijoueur en ligne, l'affichage du plateau de jeu, le chat, l'envoie et la réception des données avec le serveur. Le serveur aussi utilise cette bibliothèque afin de séparer les différents clients qui se connecteront au serveur.

La bibliothèque `socket`, qui permet d'ouvrir une connexion avec une machine locale ou distante et d'échanger avec elle. Elle est donc utilisé côté client et côté serveur.

La bibliothèque `queue`, qui permet d'implémenter des files multi-productrices et multi-consommatrices. Utile en programmation multi-thread, lorsque les informations doivent être échangées sans risques entre plusieurs threads notamment lors du multijoueur en ligne entre les threads chargés de recevoir et d'envoyer les données et les threads du jeu et du chat.

3.3.2 Les Bibliothèques spécifiques

Parmi les bibliothèques spécifiques que nous avons utilisées, il y a la bibliothèque `tkinter` qui permet la réalisation d'interfaces graphiques, cette bibliothèque a permis de réaliser l'interface graphique du chat.

La bibliothèque `json`, a été utilisée afin de sérialiser et désérialiser des objets qui sont envoyés et reçus par le client au serveur.

La base de données qui est utilisé pour sauvegarder les clients, utilise des bibliothèques tel que `os` qui permet la lecture et l'écriture de fichier, `sqlite3` qui permet la communication entre le programme et la base de données et `hashlib` qui permet de sécuriser les informations sensibles des utilisateurs.

Enfin d'autres bibliothèques plus courantes ont été utilisées comme `string` pour les opérations sur les chaînes de caractères, `time` qui permet d'utiliser des fonctions liées au temps, `copy` pour réaliser des copies d'objet et de les modifier sans modifier les originaux, et `random` pour tout ce qui est aléatoire dans le programme.

3.4 Le réseau

Le réseau permet de jouer à Advanced-Tetris de deux à quatre joueurs sur Internet ou sur un réseau local. Le protocole utilisé pour les communications est le TCP. Une amélioration pour utiliser du UDP pourrait être effectuée à l'avenir.

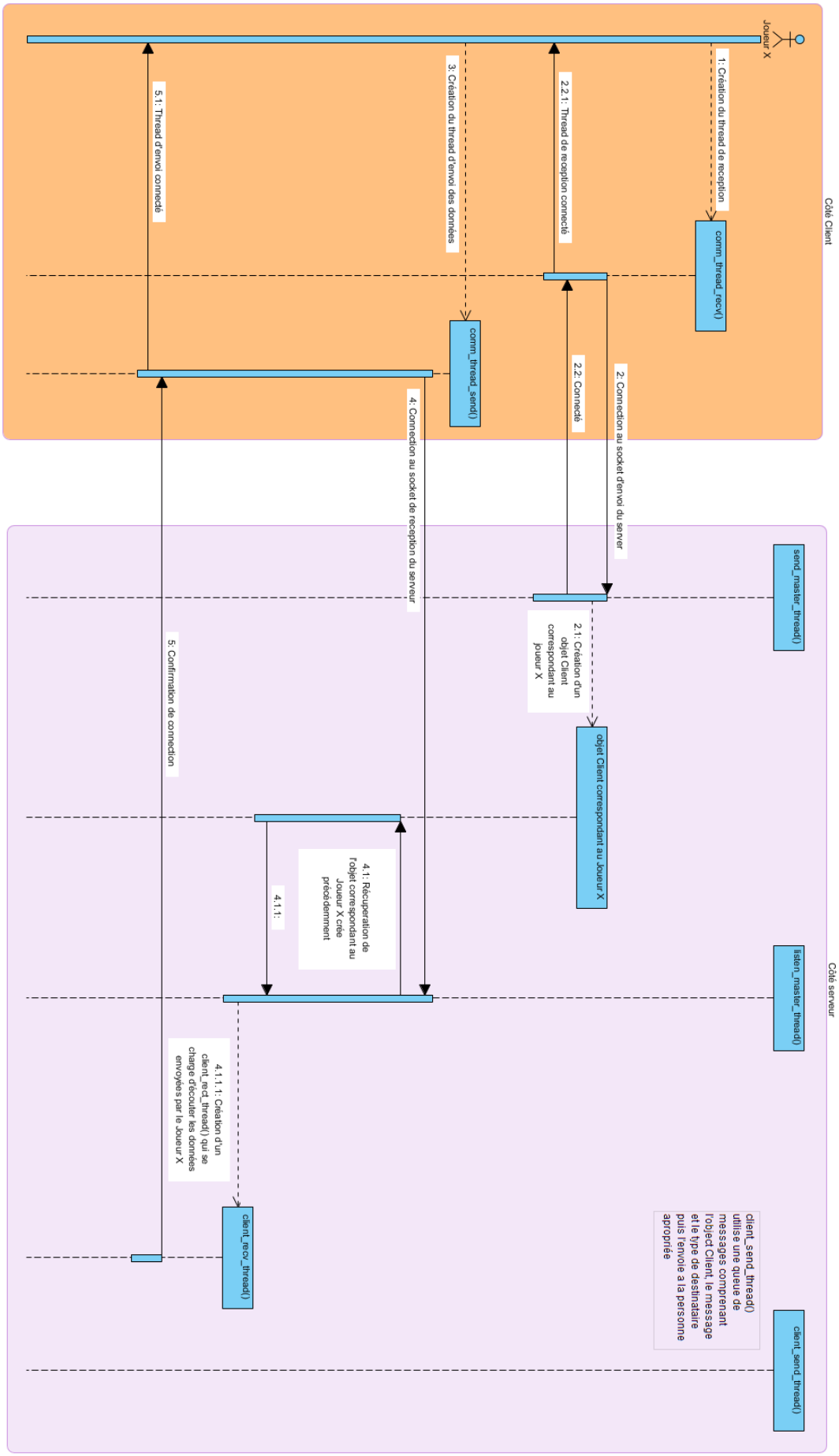
Du côté serveur, les messages sont stockés dans une queue de type LIFO prenant un tuple comme message. Le tuple contient :

— La `classe Client` qui dispose du socket d'envoi et de réception, du nom d'utilisateur et de la room dans laquelle il se situe.
— Le `message`, soit un message de chat ou les données de jeu a transmettre.
— Le `destinataire` du message, a soi-même ou aux autres joueurs dans la même room.

Les communications utilisent deux ports :

— Port 50000 : Transmission Serveur => Client
— Port 51000 : Transmission Client => Serveur

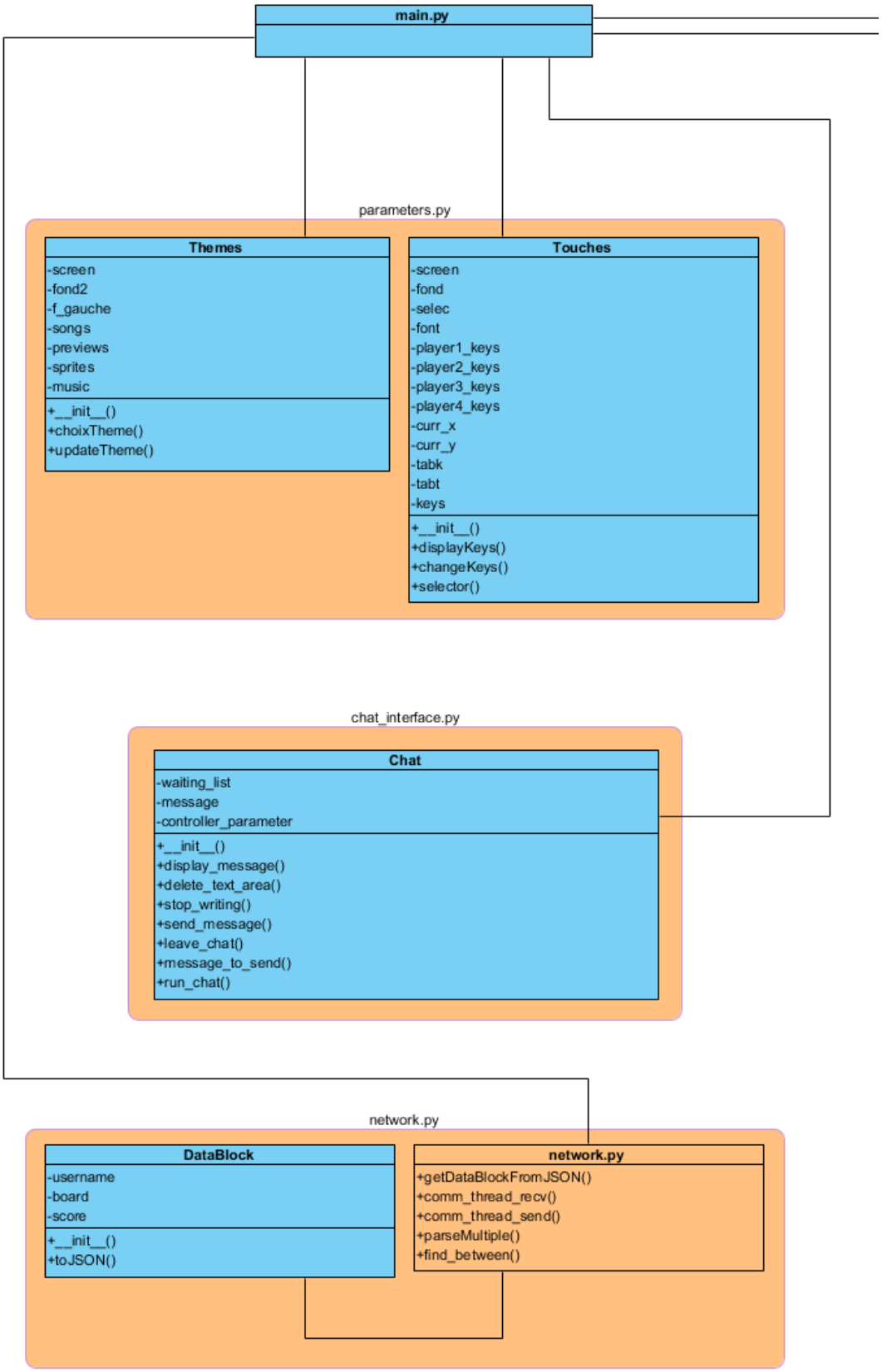
Ce diagramme de séquence décrit comment la partie réseau fonctionne.

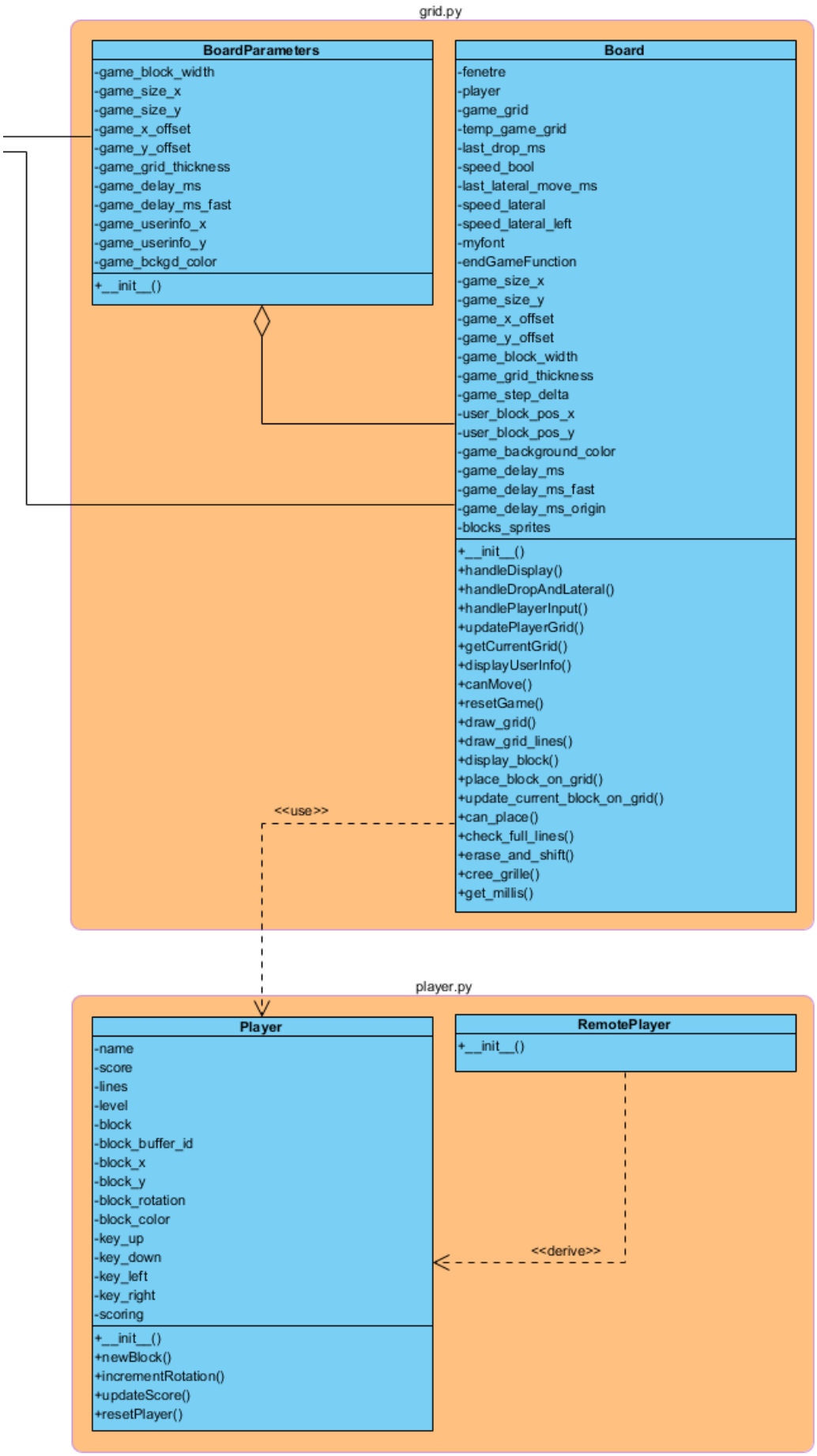


4 Architecture du projet

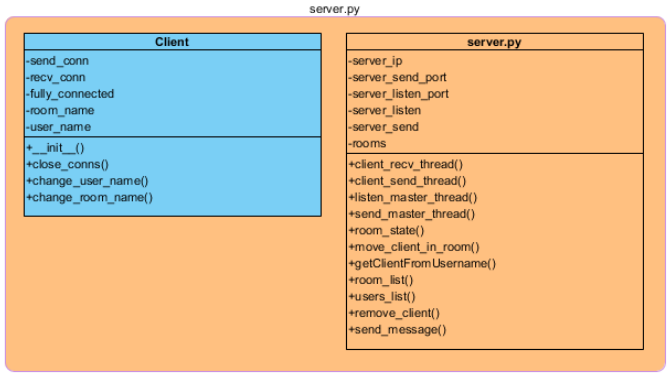
4.1 Diagrammes de classes

Ce diagramme de classe représente l'ensemble du code exécuté sur la machine client. Celui-ci comprends donc le code permettant de jouer en solo et en multijoueur local.



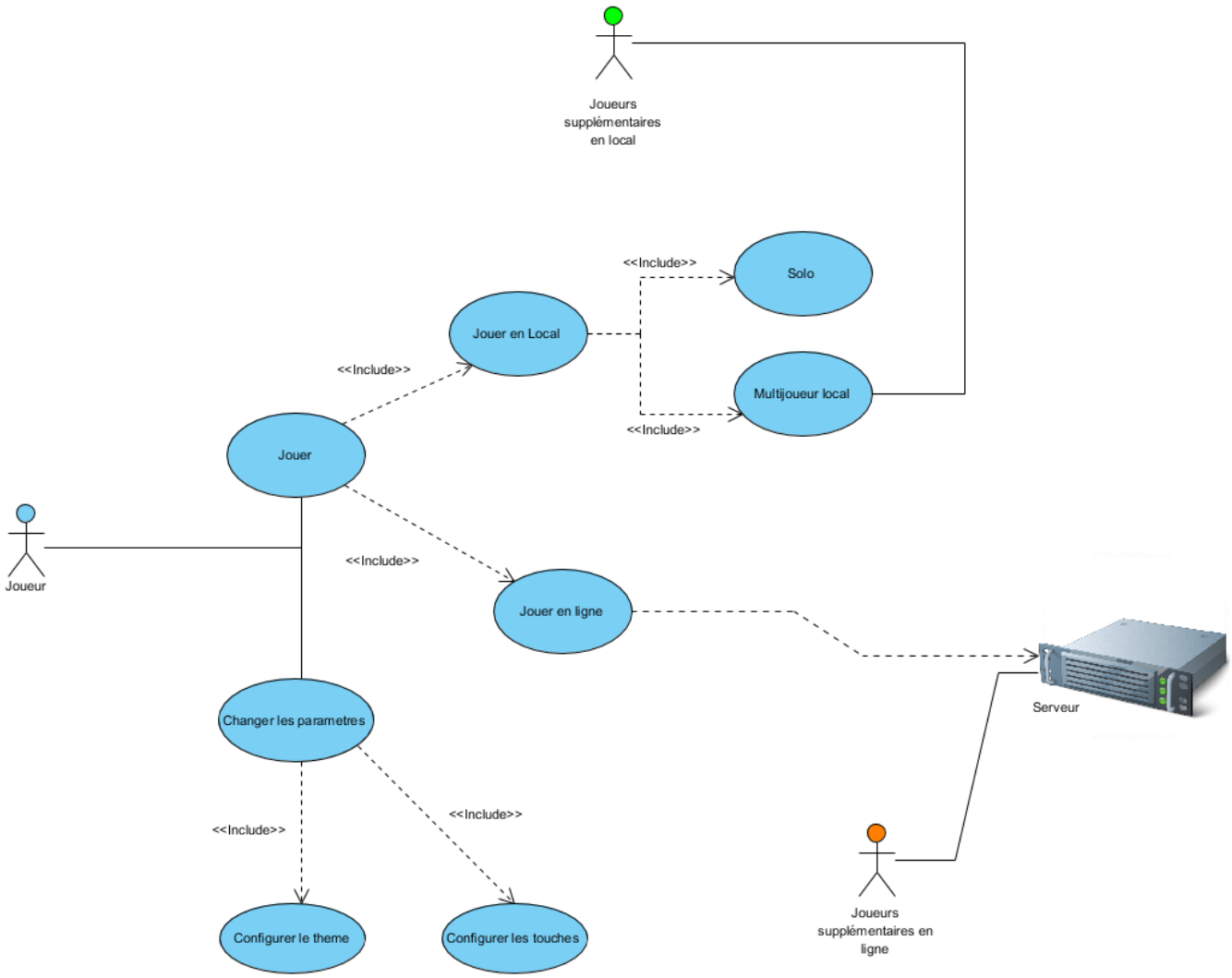

grid.py
BoardParameters
-game_block_width
-game_size_x
-game_size_y
-game_x_offset
-game_y_offset
-game_grid_thickness
-game_delay_ms
-game_delay_ms_fast
-game_userinfo_x
-game_userinfo_y
-game_bckgd_color
+__init__()
Board
-fenetre
-player
-game_grid
-temp_game_grid
-last_drop_ms
-speed_bool
-last_lateral_move_ms
-speed_lateral
-speed_lateral_left
-myfont
-endGameFunction
-game_size_x
-game_size_y
-game_x_offset
-game_y_offset
-game_block_width
-game_grid_thickness
-game_step_delta
-user_block_pos_x
-user_block_pos_y
-game_background_color
-game_delay_ms
-game_delay_ms_fast
-game_delay_ms_origin
-blocks_sprites
+__init__()
+handleDisplay()
+handleDropAndLateral()
+handlePlayerInput()
+updatePlayerGrid()
+getCurrentGrid()
+displayUserInfo()
+canMove()
+resetGame()
+draw_grid()
+draw_grid_lines()
+display_block()
+place_block_on_grid()
+update_current_block_on_grid()
+can_place()
+check_full_lines()
+erase_and_shift()
+cree_grille()
+get_millis()
<<use>>
player.py
Player
-name
-score
-lines
-level
-block
-block_buffer_id
-block_x
-block_y
-block_rotation
-block_color
-key_up
-key_down
-key_left
-key_right
-scoring
+__init__()
+newBlock()
+incrementRotation()
+updateScore()
+resetPlayer()
RemotePlayer
+__init__()
<<derive>>

Ce diagramme représente le code du serveur. Ce code est nécessaire pour que plusieurs joueurs jouent sur des ordinateurs différents.



4.2 Cas d'utilisation

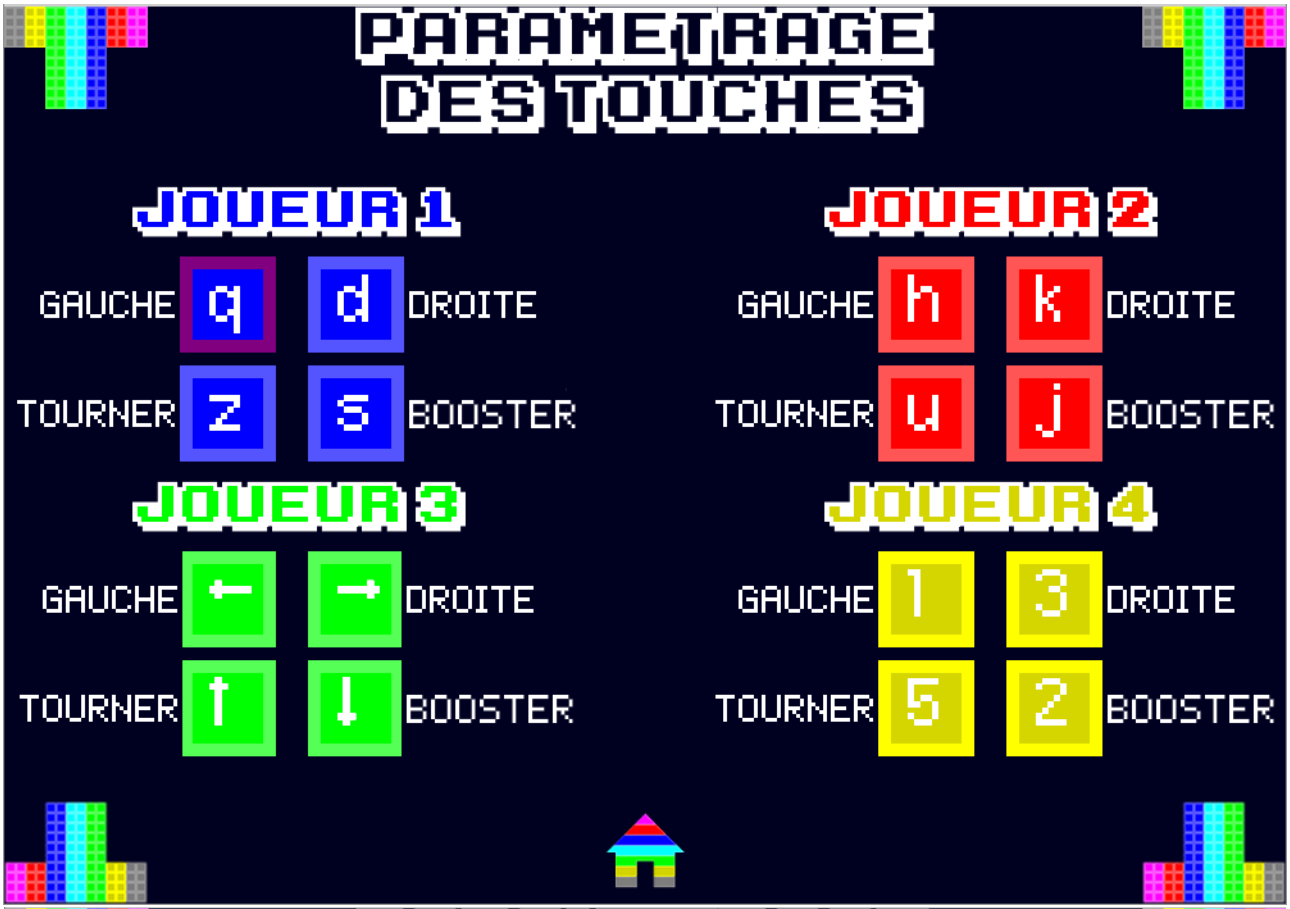
Voici le diagramme d'utilisation de notre projet Advanced-Tetris.

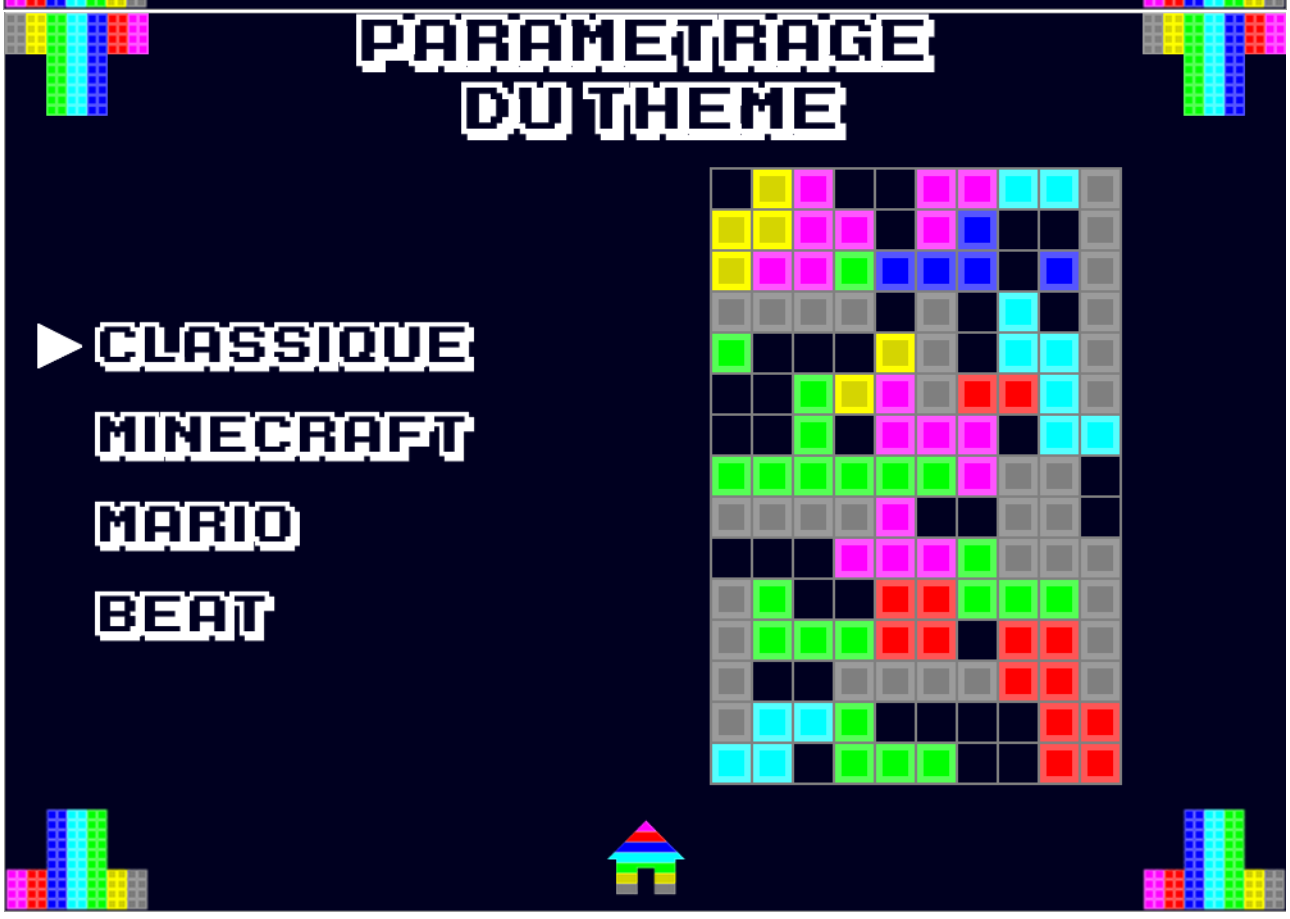


5 Expérimentations et usages

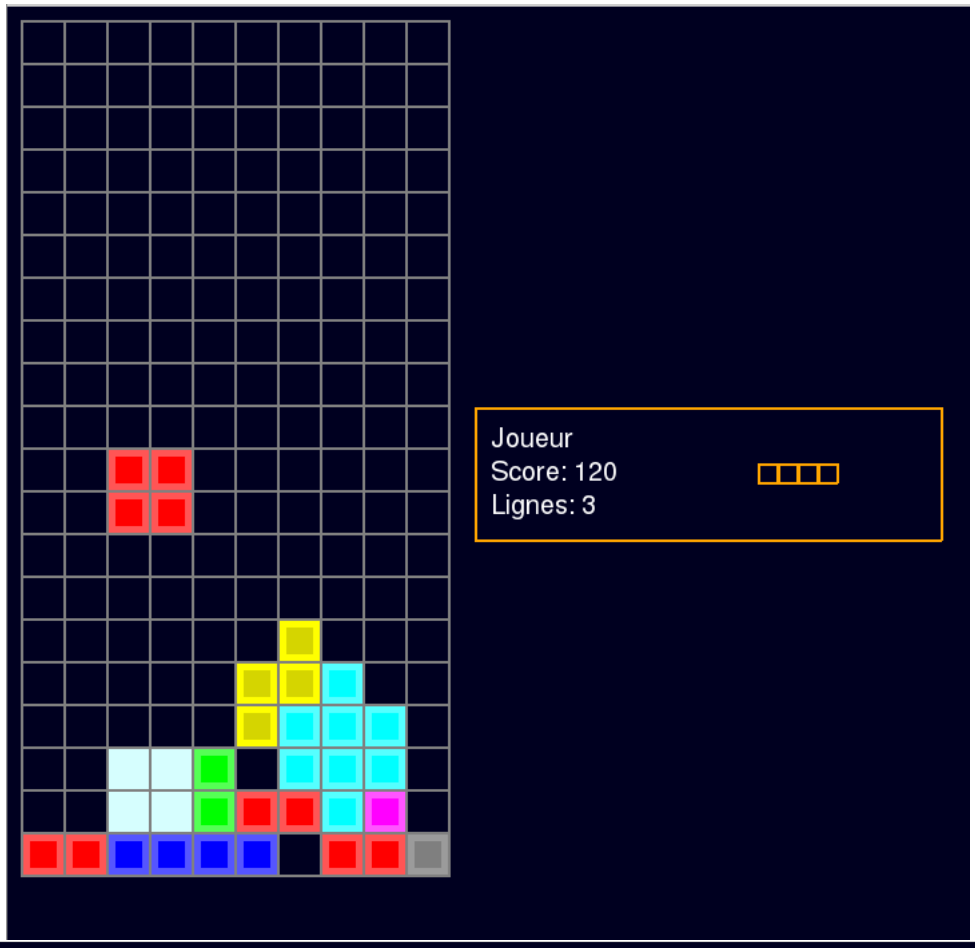
5.1 Captures d'écrans

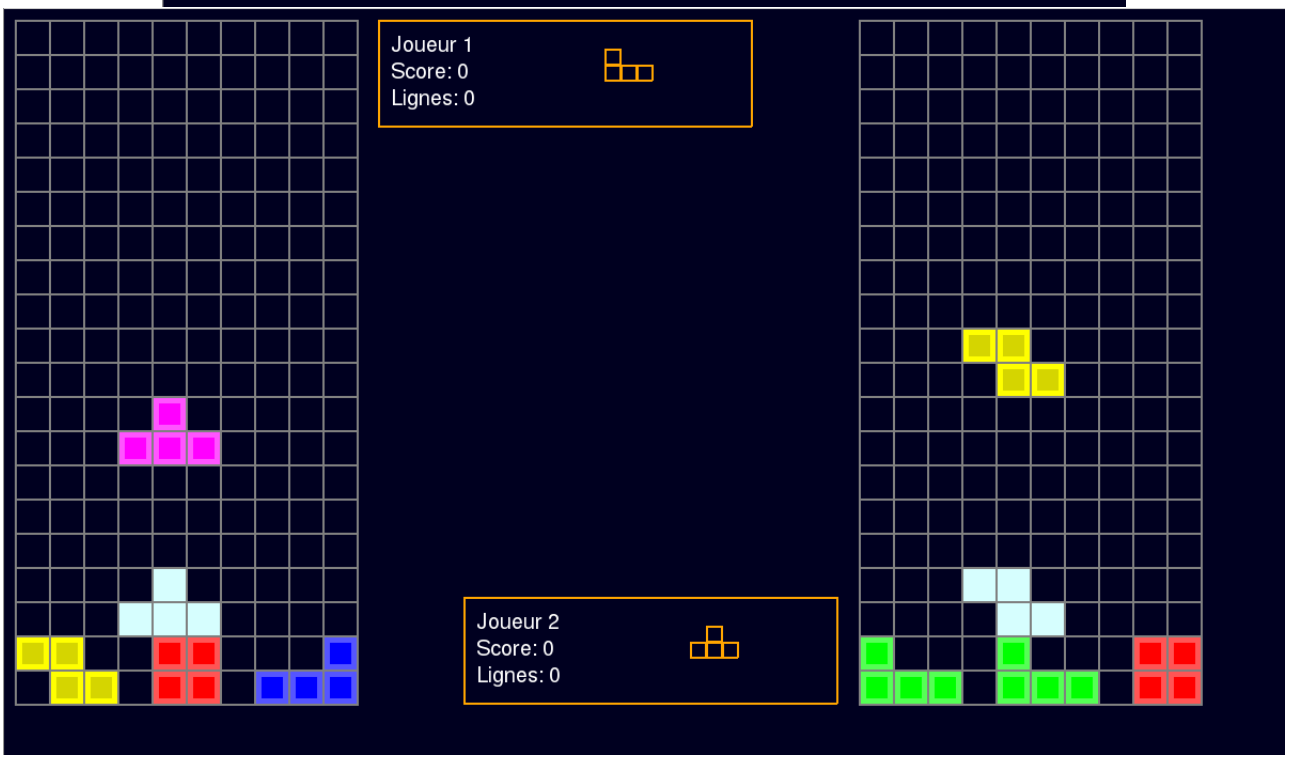
5.1.1 Interface graphique du menu de jeu

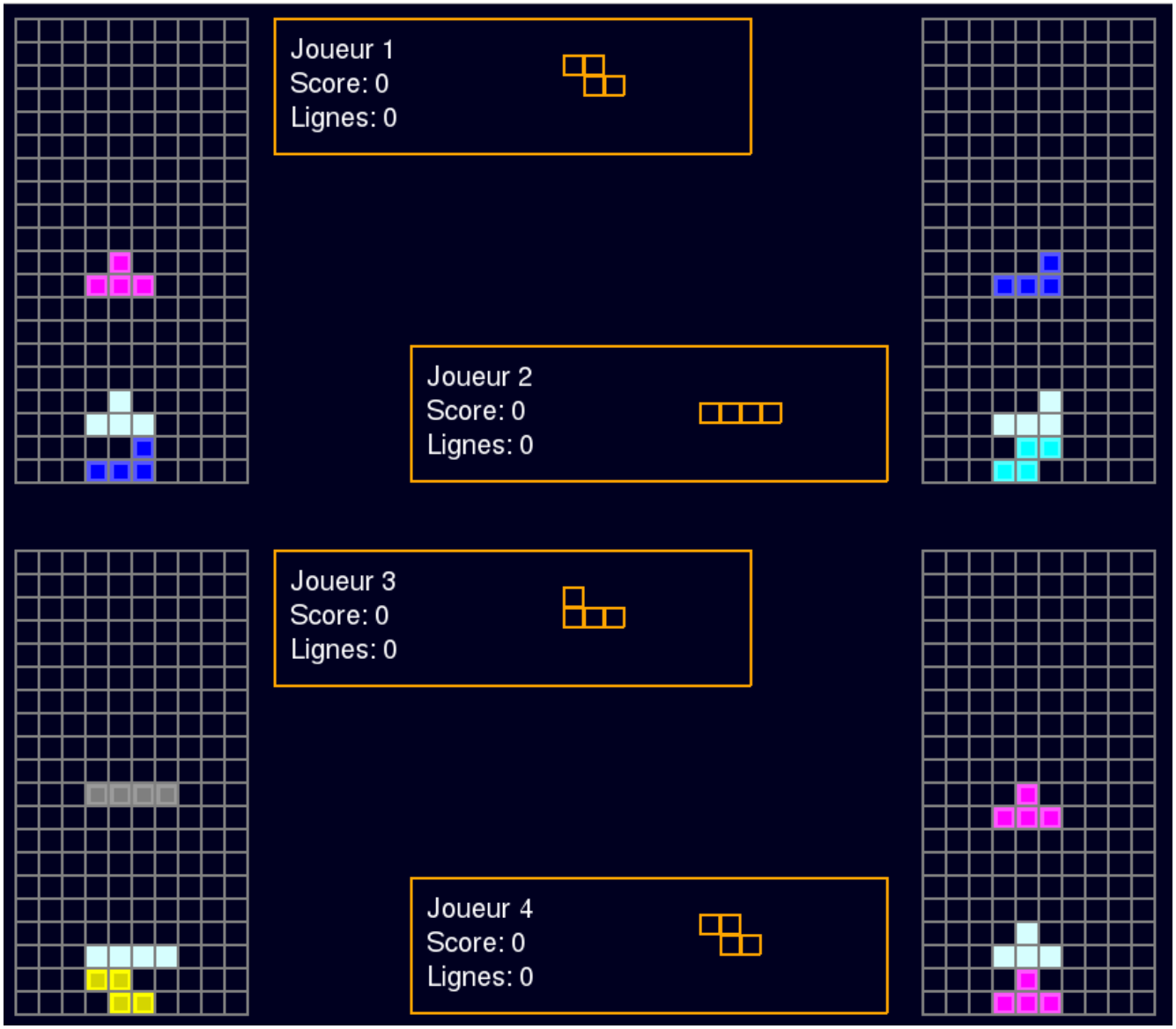

PARAMETRAGE
DES TOUCHES
JOUEUR 1
GAUCHE q d DROITE
TOURNER z s BOOSTER
JOUEUR 2
GAUCHE h k DROITE
TOURNER u j BOOSTER
JOUEUR 3
GAUCHE ← → DROITE
TOURNER ↑ ↓ BOOSTER
JOUEUR 4
GAUCHE 1 3 DROITE
TOURNER 5 2 BOOSTER


PARAMETRAGE
DU THEME
▶ CLASSIQUE
MINECRAFT
MARIO
BEAT

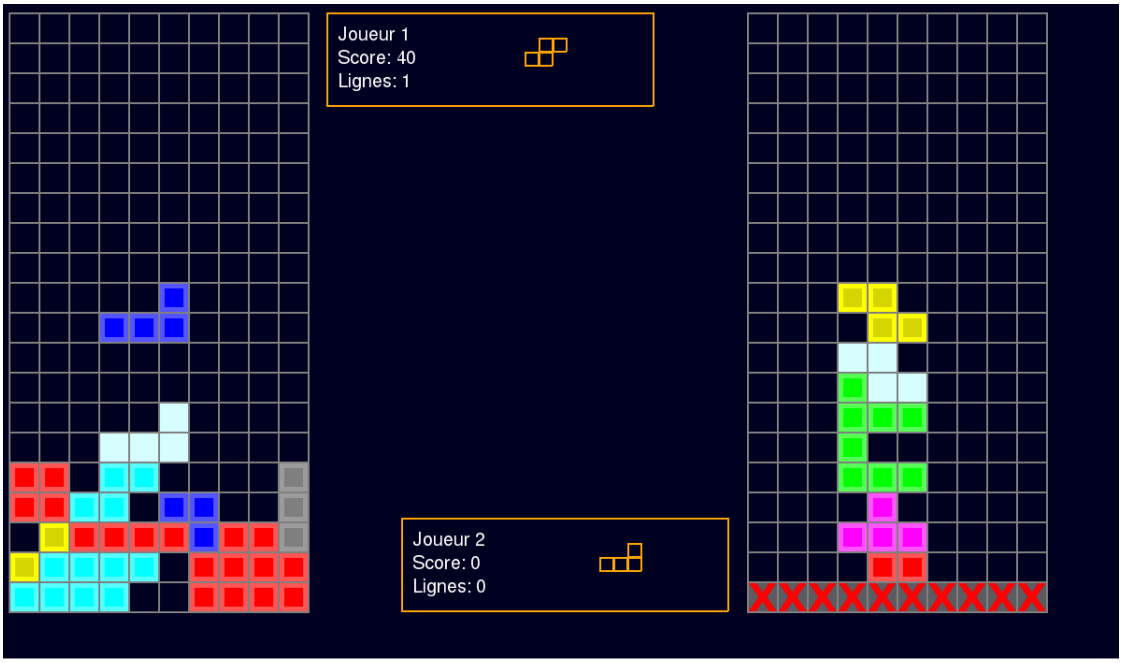
5.1.2 Interface de jeu : (1 joueur, 2 joueurs & 4 joueurs)



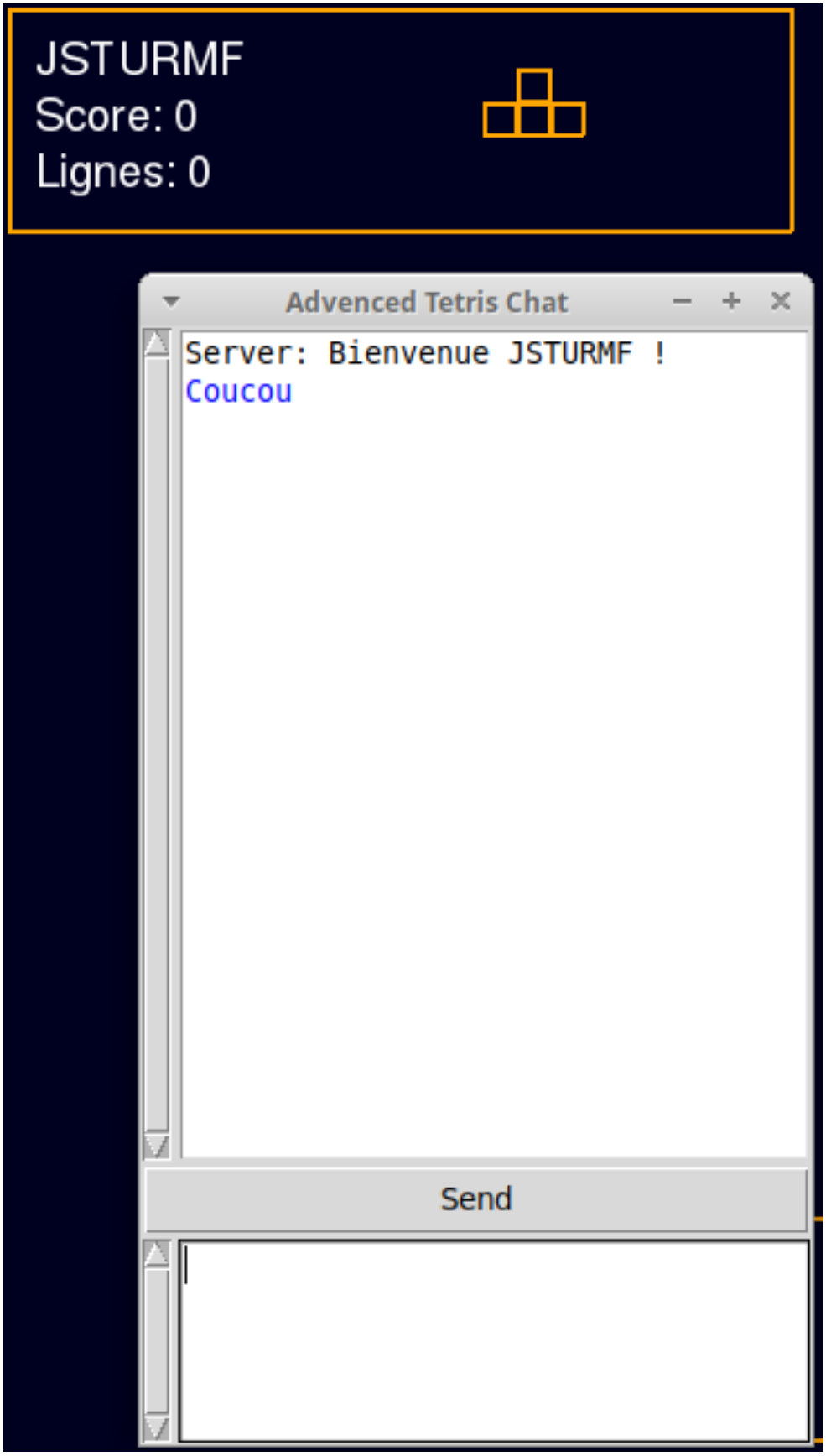




5.1.3 X-line activé : (exemple pour 2 joueurs)



5.1.4 Chat réseau



6 Conclusion

6.1 Récapitulatif des fonctionnalités principales

Pour la description de chaque fonctionnalités cf. partie 2.1

➥ **Menu complet et fonctionnel**

➥ **Mode LOCAL**

➥ **Mode ONLINE**

➥ **Chat dans le mode ONLINE**

➥ **Visualisation de la pièce suivante**

➥ **La pièce fantôme**

➥ **Vitesse progressive**

➥ **X-line**

➥ **Personnalisation des touches**

➥ **Configuration du thème de jeu**

6.2 Propositions d'améliorations

Voici une liste des améliorations possibles que nous pourrions faire dans le futur :

➻ Permettre au joueur d'importer son propre thème.

➻ Bloc explosif à envoyer chez le (ou les) adversaire(s).

➻ Tempo de la musique qui augmente en fonction du temps.

➻ Ajout d'une ligne incomplète chez un ou tous les adversaires lorsque l'on complète une ligne.

➻ Remplacement du JSON de DataBlock par une sérialisation avec Pickle or Protobuf pour de meilleurs performances.

➻ Utilisation du protocole UDP pour les communications réseau afin d'augmenter les performances en ligne.

➻ Plateforme intégrée permettant d'organiser des tournois de Tetris.

➻ Sauvegarde de la partie en cours pour la continuer plus tard.

➻ Possibilité de jouer avec plus de 4 joueurs en ligne comme sur Tetris 99 par exemple.

➻ Sauvegarde des paramètres choisis lorsque l'on quitte l'application.

6.3 Apport personnel

Ce projet nous a beaucoup apporté au niveau connaissance, du fait qu'il a fallu découvrir une nouvelle bibliothèque python, et se familiarisé un peu avec la programmation orienté objet. De ces faits nous avons acquis une assez bonne connaissance de python sous beaucoup de formes, et nous sommes préparés pour la programmation orienté objet en java, de la 2e année de licence informatique. Nous avons aussi acquis quelques connaissances en réseau, car il a fallu pas mal se documenter et se renseigner pour pouvoir effectuer la partie réseau du jeu, puisque nous n'avons encore rien vu en cours à ce sujet.
Donc ce fut un apport très riche en connaissances, et cela nous a aussi permis d'apprendre à travailler en équipe sur un projet, ce qui est important pour travailler correctement au sein d'une entreprise.