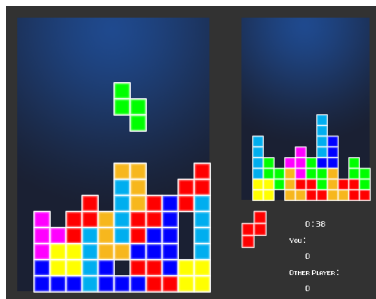


Le Vouitris



Cynthia MAILLARD, Félix ROYER, Alexandre DILLON

13 mars 2019

Tuteur de projet : Julien BERNARD

① Adaptation du jeu d'origine

- Le jeu originel

- Notre adaptation

② Modélisation du jeu

- Architecture réseau

- Protocole d'échange de messages

③ La communication

- Le serveur

- La sérialisation

④ La jouabilité

- Les structures de données

- Le client graphique

① Adaptation du jeu d'origine

- Le jeu originel

- Notre adaptation

② Modélisation du jeu

- Architecture réseau

- Protocole d'échange de messages

③ La communication

- Le serveur

- La sérialisation

④ La jouabilité

- Les structures de données

- Le client graphique

Adaptation du jeu d'origine

Le jeu originel

Le premier Tétris (1984) - Alekseï Pajitnov

- jeu de puzzle
- succès mondial dans les années 1990
- adapté sur pratiquement toutes les consoles
- certaines versions multijoueurs

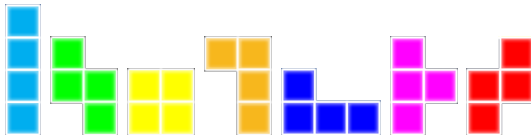


Adaptation du jeu d'origine

Le jeu originel

La jouabilité

- déplacement latéral et rotation des tétramino, chute accélérée des tétramino
- agencer les tétramino en ligne
- les lignes disparaissent, rapportant du score
- la partie s'arrête quand les tétramino touche le plafond



Adaptation du jeu d'origine

Notre adaptation

Le cahier des charges

- architecture client-serveur
- un joueur, un client, un ordinateur
- développement en C++ / Gamedev Framework / Boost::Asio
- développement de notre propre bibliothèque de sérialisation

Les objectifs

- approfondir la programmation répartie
- établir un protocole réseau
- découvrir la sérialisation
- se familiariser aux bibliothèque graphiques

Adaptation du jeu d'origine

Notre adaptation

Les choix d'implémentation

- le serveur fait loi
- le client envoie les actions du joueur au serveur
- le serveur lui renvoie l'état du jeu mis à jour
- détruire des lignes inflige des malus à l'adversaire
 - 1 ligne : pas de malus
 - 2 lignes : empêche la rotation
 - 3 lignes : accélération de la chute
 - 4 lignes : suppression de block en bas du tableau

Adaptation du jeu d'origine

Notre adaptation

Les contrôles

- ← : gauche
- → : droite
- ↓ : chute rapide
- *SPACE* : rotation

① Adaptation du jeu d'origine

Le jeu originel

Notre adaptation

② Modélisation du jeu

Architecture réseau

Protocole d'échange de messages

③ La communication

Le serveur

La sérialisation

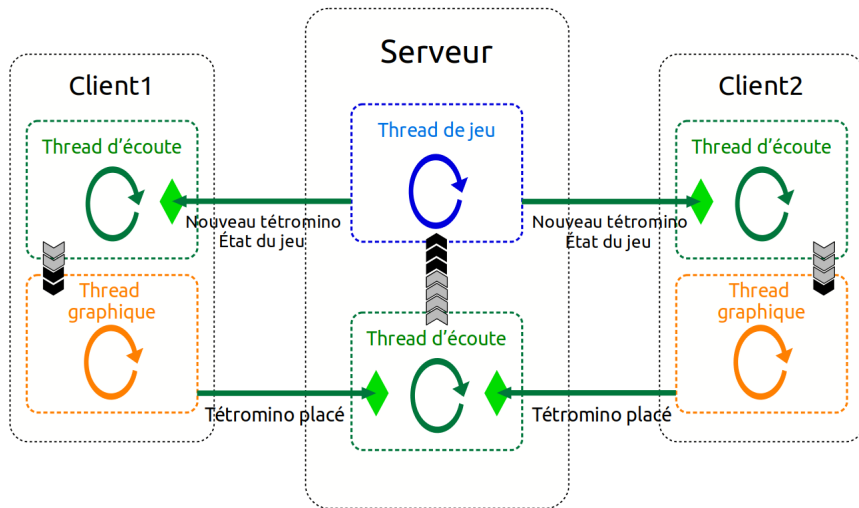
④ La jouabilité

Les structures de données

Le client graphique

Modélisation du jeu

Architecture réseau



Modélisation du jeu

Protocole d'échange de messages

Les messages

- 1 $S \rightarrow C$: StartGame : début de la partie
- 2 $S \leftarrow C$: TétrominoPlaced : position et rotation du tétromino placé
- 3 $S \rightarrow C$: MalusStart : signal le début d'un malus
- 4 $S \rightarrow C$: UpdateGrid : nouvelle version des grilles
- 5 $S \rightarrow C$: NewTetromino : nouveau tétromino utilisé par le joueur
- 6 $S \rightarrow C$: MalusEnd : signal la fin d'un malus
- 7 $S \rightarrow C$: GameOver : signal la fin de la partie et donne le gagnant

① Adaptation du jeu d'origine

Le jeu originel

Notre adaptation

② Modélisation du jeu

Architecture réseau

Protocole d'échange de messages

③ La communication

Le serveur

La sérialisation

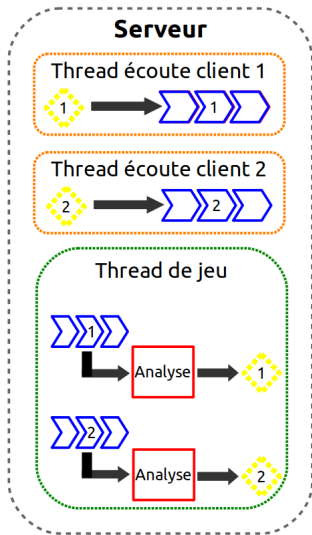
④ La jouabilité

Les structures de données

Le client graphique

Mise en oeuvre


Le serveur



Les données contenues dans le serveur pour chacun des joueurs

- grille de jeu
- scores
- un vérificateur de triche
- un gestionnaire de malus

 : thread

 : socket

 : file

L'exploitation des messages

- 1 Réception message TetrominoPlaced
- 2 Vérification triche
- 3 Mise à jour de la grille
- 4 Mise à jour du score
- 5 Envoi de malus à l'adversaire
- 6 Envoi grille mise à jour aux joueurs

Le vérificateur de triche

- vérifie que les données renvoyé par le client sont cohérentes
 - le temps pour jouer la pièce n'est pas trop long
 - que les malus sont bien appliqués
- trois fautes autorisée

Objectif

Permettre les échanges de structures complexes entre clients et serveur via des messages standardisés.

Modèles

- GamedevFramework
- SFML/Packet

Structures de sérialisation

Deux classes symétriques communes aux clients et au serveur :

- Serializer
- Deserializer

Mise en oeuvre

Sérialisation

Sérialisation templâtée

Utilisation d'une méthode templâtée pour sérialiser tout type simple.

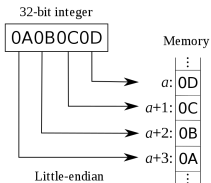
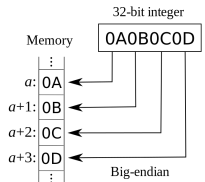
Endianess

Ordre sequentiel dans lequel sont ranger nos données sérialisées.

Endianess de format Big-Endian : par défaut sur les structures réseaux.

Sérialisation d'objet

Sérialisation des attributs de l'objet que l'on souhaite communiquer.



Les messages

Deux types de messages :

- Du client au serveur : structure Request_CTS
- Du serveur au client : structure Request_STC

Contenu des messages

- Un type enum correspondant au type du message
- Une union des différentes structures de message

Sérialisation de message

- Sérialisation du type de message.
- Sérialisation des éléments de la structure du message.

① Adaptation du jeu d'origine

Le jeu originel

Notre adaptation

② Modélisation du jeu

Architecture réseau

Protocole d'échange de messages

③ La communication

Le serveur

La sérialisation

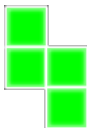
④ La jouabilité

Les structures de données

Le client graphique

La représentation d'un tetromino

- type
- position abscisse/ordonnée
- rotation
- matrices $\langle 2,4 \rangle$ des différents types de tétramino



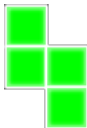
1	0
2	1
0	1
0	0

Les structures de données communes

Tétromino

Récupérer les cases à partir de l'ancre

- 1 parcours de la matrice
- 2 détermination de l'ancre
- 3 renvoi la liste de cases occupées par le tétramino
- 4 la rotation du tétramino change le sens de parcours



1	0
2	1
0	1
0	0

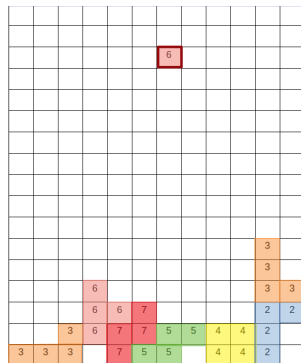
Grid

Vérifier les actions

- rotation possible
- déplacement possible
- descente possible

Vérifier l'état du jeu

- ligne complète
- grille complète
- nouveau tétramino en jeu

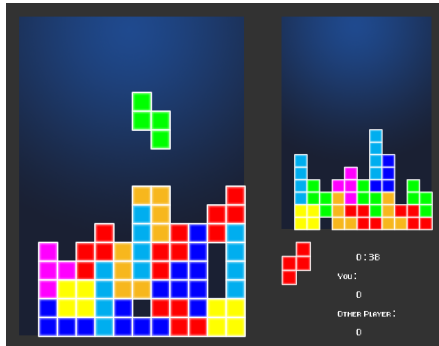


Mise en oeuvre

Le client graphique

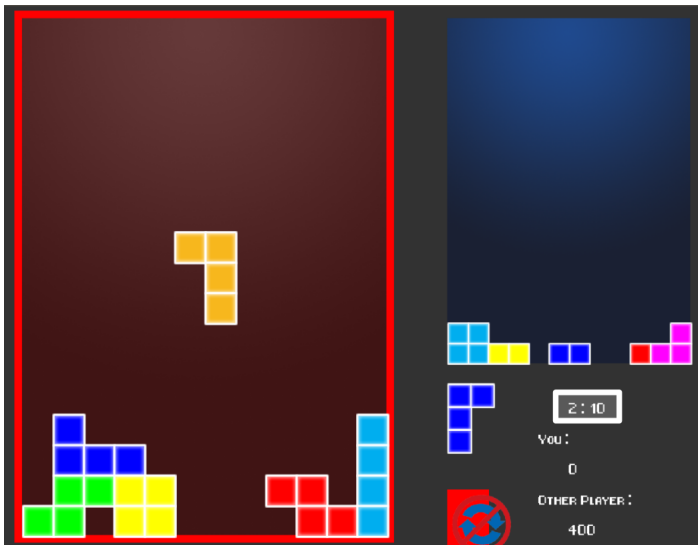
Le rôle du client

- afficher le jeu
- recevoir les messages du serveur
- gérer les interactions du joueur



Mise en oeuvre

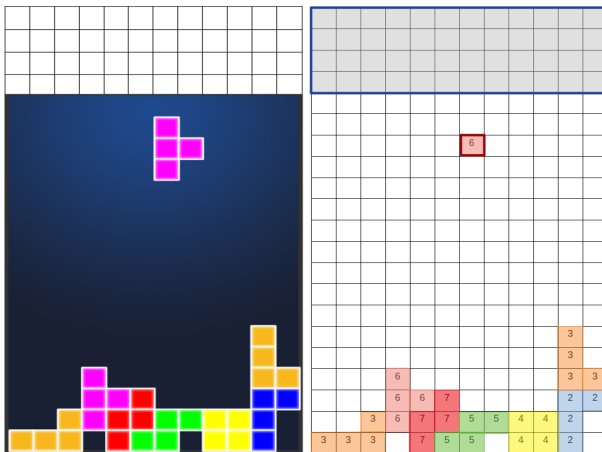
Le client graphique



Mise en oeuvre

Le client graphique

La zone de jeu



La boucle de jeu

- ① vérifier si un message est présent dans le file
 - interpreter le message
- ② recevoir les actions du joueur
 - vérifier si elles sont possibles
 - executer l'action
- ③ verifier si un tétramino est placé
 - le signaler au serveur
- ④ mettre à jour l'affichage

L'interprétation des messages

- ① GameStart
 - lancer l'horloge
- ② NewTetromino
 - place le prochain tétramino en jeu
- ③ UpdateGrid
 - met à jour les grilles des joueurs
- ④ Malus
 - applique le malus
- ⑤ GameOver
 - affiche le message de fin

Un projet complet...

- jeu fonctionnel
- cahier des charges respecté

... avec des améliorations possibles

- mode de jeu avec plus de joueurs
- système anti-triche plus performant

Avez-vous des questions ?