Aix*Marseille université
Socialement engagée

Année 2023-24

Licence informatique

Applications réseaux : Projet

Godfinder

1 Goldfinder

Le but de ce projet est d'implémenter un jeu multi-joueur avec une architecture client-serveur.

Un joueur incarne un chercheur d'or dans une mine abandonnée dont la carte a été perdue depuis longtemps. Le but du jeu est de collecter le plus d'or. Un joueur est représenté par un carré coloré, et la mine par un labyrinthe créé à partir d'une grille rectangulaire.

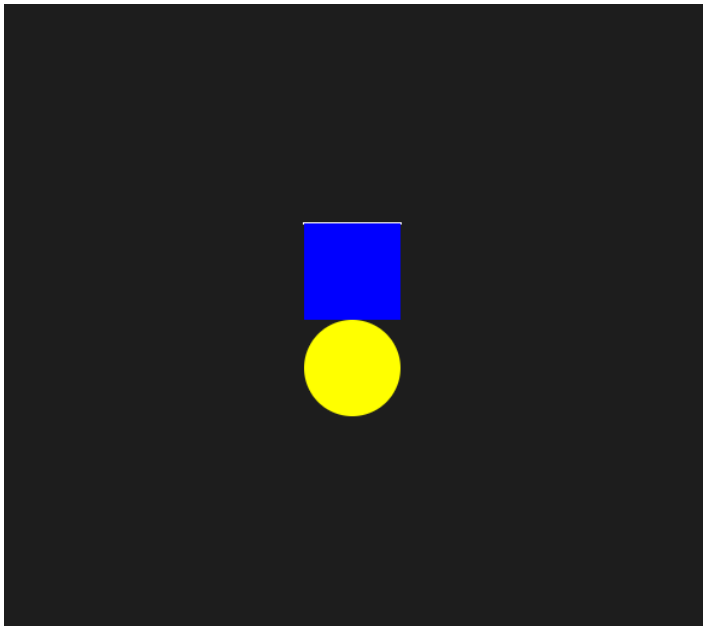

Figure 1 – Chercheur d'or (carré bleu) à coté d'une pépite (rond jaune)

Chaque joueur démarre d'une case aléatoire de la grille et se déplace à l'aide des touches du clavier (`q`,`s`,`d`,`z`). Le but est de collecter plus de pièces que les adversaires. Les joueurs découvrent le labyrinthe au fur et à mesure de leurs déplacements. Au début, chaque joueur ne connaît que sa case départ et ses alentours (`SURROUNDING`). En fonction des déplacements (`UP`, `DOWN`, `LEFT` ou `RIGHT`) entrés au clavier, le joueur se déplace et découvre de nouvelles parties du labyrinthe.

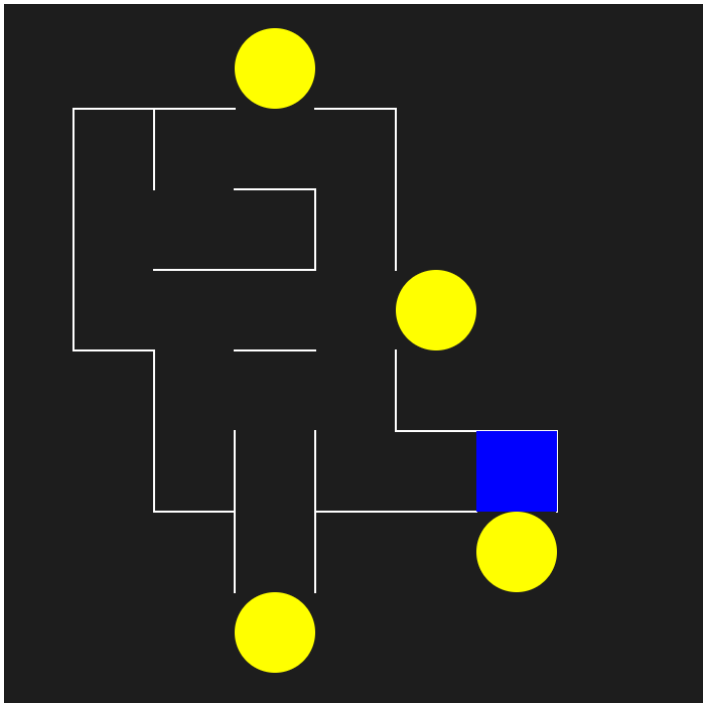

FIGURE 2 – Après quelques pas dans la mine

Concrètement, au démarrage d'une partie, un nouveau labyrinthe est généré par le serveur. Les joueurs sont positionnés aléatoirement sur celui-ci. La position des joueurs, ainsi que le labyrinthe ne sont connus que par le serveur. Chaque joueur communique avec les serveur au moyen de requêtes, qui permettent soit de connaître son voisinage, soit de se déplacer. En cas de déplacement sur une case contenant une pièces d'or, elle est récupérée par le joueur. Une requête de voisinage entraîne coté serveur l'envoi de messages décrivant l'environnement immédiat du joueur dans le labyrinthe : pour chaque direction, présence ou absence d'un mur, d'une pièce, d'une case libre ou occupée par un autre joueur.

Une requête de déplacement change la position du joueur dans la grille, à condition que cela soit possible. Le déplacement est relatif à la position courante du joueur. Par exemple, un déplacement `UP` place le joueur sur la case immédiatement au-dessus à condition qu'elle ne soit pas séparée par un mur de la case courante.

Le jeu s'arrête lorsque l'ensemble des cases du labyrinthe ont été visitées et tout l'or récupéré. Le vainqueur est le joueur ayant récupéré le plus de pièces.

1.1 Protocole goldfinder

Les échanges entre le client et le serveur sont régis par le protocole suivant. Le port du serveur est **1234**. Vos clients et serveur devront communiquer **en respectant ce protocole**.

Début/fin d'une partie Le démarrage d'une partie se fait en deux phases. Le client demande à rejoindre une partie au moyen d'un message

— `GAME_JOIN:player_name` où *player_name* est le nom du joueur

Lorsque la partie est prête à démarrer, le serveur répond par le message

— `GAME_START` suivi par la liste des joueurs participants. Les participants sont transmis via une séquence d'éléments de la forme `player_name:index` séparés par des espaces et terminée par `END`. *index* est un entier entre 0 et le nombre de participants -1. Par exemple, pour une partie à 4 joueurs, le serveur pourrait envoyer `GAME_START alice:0 bob:1 carole:2 denis:3 END`

La fin d'une partie est déterminée par le serveur (toutes les cases ont été explorées et toutes les pièces d'or ont été récupérées). Lorsque les conditions de fin de partie sont réunies, le serveur envoie successivement :

— `GAME_END` suivi d'une séquence de la forme `player_name:score` où *player_ name* est le nom du joueur et *score* son score. Les éléments `player_name:score` sont séparés par des espaces. Le serveur termine la séquence par `END`. Par exemple, le message suivant annonce la fin d'une partie gagnée par carole : `GAME_END alice:10 bob:8 carole:12 denis:1 END`

Déplacement et voisinage Le client envoie des requêtes au serveur pour obtenir le voisinage du joueur dans le labyrinthe, et pour déplacer le joueur dans le labyrinthe.

— Le message `SURROUNDING` permet au client de demander au serveur ce qui entoure le joueur par rapport à sa position courante. Le serveur répond par une séquence d'éléments de la forme `dir:item` séparés par un ou plusieurs espaces et terminé par `END`.
  — `dir` est direction de `item` par rapport à la position courante du joueur. `dir` est `UP`, `DOWN`, `LEFT` ou `RIGHT`.
  — `item` décrit ce qui se trouve dans la direction `dir` par rapport à la position courante du joueur. `item` est soit `WALL` (mur), soit `PLAYERi` où i est l'index du joueur, soit `GOLD` pour une pièce d'or, soit `EMPTY` (case vide).

  Par exemple, un message `SURROUNDING` pourrait entraîner la réponse suivante du serveur `UP:WALL RIGHT:EMPTY DOWN:GOLD LEFT:PLAYER2 END`
— Pour déplacer le joueur, le client envoie un message `dir` où *dir* est la direction souhaitée (`UP`, `DOWN`, `LEFT` ou `RIGHT`). Le serveur si le déplacement est possible met à jour la position du joueur. Il met aussi à jour le labyrinthe et incrémente le score du jouer si une pièce d'or est présente à la nouvelle position. Il répond avec un message de la forme `INVALIDMOVE` si le déplacement est impossible ou `VALIDMOVE:item` si le déplacement est possible. Dans ce dernier cas, `item` est soit `EMPTY`, soit `GOLD`. La valeur de *item* représente le contenu de la nouvelle case sur laquelle se déplace le joueur.

Par exemple, un déplacement vers la case immédiatement au dessus de la case courante, en admettant qu'une pièce d'or s'y trouve, générera l'échange suivant :

— message `UP` du client au serveur
— réponse `VALIDMOVE:GOLD` du serveur vers le client.

1.2 code de démarrage

Un dépôt etulab qui contient le code à partir duquel démarrer votre projet est accessible ici : `https://etulab.univ-amu.fr/travers.c/goldfinder-template`

Grille et génération de labyrinthes Des classes permettant de représenter la grille et de générer des labyrinthes aléatoires sont fournies.

— Classe `Grid`. Un labyrinthe est représenté par une grille rectangulaire.
  — Pour chacune des directions *up, down, left, right*, chaque case de la grille est séparée ou pas de sa voisine par un mur. Les méthodes `boolean leftWall(int i, int j)`, `boolean rightWall(int i, int j)`, `boolean upWall(int i, int j)`, `boolean downWall(int i, int j)` permettent de savoir si un mur sépare dans la case d'indices *(i,j)* de sa voisine dans la direction *left,*, *right*, *up* et *down* respectivement.
  — L'attribut `boolean[][] gold` gère la présence d'or dans les cases de la grille. `gold[i][j]` est vrai si une pièce d'or est présente dans la case d'indices *(i,j)*.
  — Le constructeur `Grid(int columnCount, int rowCount, Random random)` génère de manière aléatoire un labyrinthe avec une grille de *columnCount* colonnes et *rowCount* lignes. Le labyrinthe est parfait, dans le sens où chaque case est accessible. Le constructeur s'occupe aussi de répartir aléatoirement des pièces d'or sur les cases de la grille.
— Classe `RandomMaze`. Cette classe est responsable de la génération aléatoire de labyrinthes. Elle est appelée par le constructeur de `Grid`.

Interface graphique Coté client, l'interface graphique montre la partie connue du labyrinthe. La classe `GridView` permet d'afficher la grille à partir de la connaissance courante des positions des murs et des pièces d'or (stockée dans les tableaux `hWall`, `vWallet` `GoldAt` respectivement.). Un squelette de contrôleur est également fourni (classe `Controller`). Il s'occupera en particulier de récupérer les touches pressées par l'utilisateur (méthode `handleMove`.). L'agencement de l'interface est décrite dans le fichier `gridView.fxml`.

2 Tâche 0 : Mise en place

1. Choisissez votre binôme et inscrivez-vous tous les deux dans la même équipe dans ametice. Il est obligatoire de faire partie d'une équipe pour être évalué.
2. Créez un dépôt etulab qui contiendra le code de votre équipe. Ajouter dans les membres votre chargé de TP avec les droits d'au moins *reporter*. Vous pouvez forker le dépôt contenant le code de démarrage.
3. Indiquez ensuite l'adresse de votre dépôt dans ametice.

3 Tâche 1 : Parties mono-joueur

Écrire le code client et serveur pour permettre de jouer en mode mono-joueur :
— Le serveur n'accepte qu'une seule connexion à la fois
— Lorsqu'un client se connecte , le jeu se termine une fois que le joueur a exploré tout le labyrinthe et a collecté l'ensemble des pièces d'or.
Vous devez respecter le protocole décrit plus haut. Vous utiliserez au choix TCP ou UDP.

4 Tâche 2 : Parties multi-joueur

Le serveur doit être maintenant capable de gérer des parties multi-joueurs. Le serveur devra avertir un joueur lorsqu'un autre joueur est situé dans ses alentours : c'est-à-dire dans une case adjacente non séparé par un mur. Le serveur est donc maintenant susceptible d'envoyer des message de la forme `dir:playerN END` ou `dir:empty END` pour signifier la présence d'un autre joueur dans une case adjacente ou le départ de celui-ci.

Le serveur devra faire en sorte qu'à tout moment, au plus un joueur est positionné sur une case donnée. Autrement dit, deux joueurs ne peuvent se trouver simultanément sur la même case.

5 Tâche 3 : Bot

Pour des test de performances (comme dans les TPs 4 et 5), écrire un client automatique (par exemple qui se déplace de façon aléatoire). Il servira pour des tests de performances et par exemple pour compléter une partie multi-joueurs .

6 Tâche 4 : Leader-board

Le serveur maintient un tableau des meilleurs scores. Le score d'un joueur est la somme des pièces collectées au cours de l'ensemble de ses parties. Les scores pourront être stockés dans un fichier `score.txt` dont chaque ligne a le format *player :score. player* est le nom d'un joueur et *score* son score total.

On ajoute au protocole la requête `LEADER:n` qui permet d'obtenir la liste des au plus `n` meilleurs scores. En réponse à ce message, le serveur renvoie une séquence d'au plus `n` éléments de la forme `SCORE:name:xx` où `name` est le nom d'un joueur et `xx` son score actuel. La séquence est terminée par `END`.

Pour tester ce service, écrire un client qui envoie périodiquement des requête au serveur et affiche la liste des n meilleurs scores, n étant un paramètre du client.

7 Tâche 5 : Gestion de parties multiples

Le serveur doit être capable de gérer simultanément plusieurs parties multi-joueur. Le nombre de joueurs par partie est configuré au démarrage du serveur (par exemple 4). Au fur et à mesure que des joueurs se connectent, de nouvelles parties sont créées.

8 Tâche 6 : Multi-serveurs

Le jeu gagne en popularité et il n'est plus possible à un seul serveur de supporter la charge. On souhaite donc avoir plusieurs serveurs, chacun capable de gérer simultanément plusieurs parties. Plus précisément, il s'agit de mettre en oeuvre l'architecture suivante :

— Un serveur *dispatch* qui régule la charge. Son rôle est de recevoir les connections des joueurs, et les diriger vers l'un des serveurs de jeux disponibles. On ajoute au protocole goldfinder le message `REDIRECT:ip` où *ip* désigne une adresses IPv4 ou IPv6. Lorsqu'un client se connecte au serveur *dispatch*, il reçoit un message `REDIRECT:ip` où *ip* est l'adresse du serveur de jeu auquel il doit se connecter.

— Plusieurs serveurs de jeu (*game server*). Chacun d'entre eux gère plusieurs parties multi-joueurs.

Le nombre de serveurs de jeux est configuré au démarrage. Le serveur *dispatch* devra maintenir à jour le score de chaque joueur. Il devra aussi tenir à jour la charge de chaque serveur de jeu (le nombre de parties actives pour chaque serveur.). Lorsqu'un client se connecte, il est redirigé vers le serveur de jeu le moins chargé.

9 Tâche 7 : Montée en charge

Évaluer la charge supportable par votre implémentation multi-serveur. Utiliser pour cela plusieurs clients automatiques (tâche 3 : bot). Faire un graphique montrant la charge maximale supportable en fonction du nombre de serveurs de jeux.

10 Tâche 8 : Cops vs. robbers

Implémenter une variante *cops vs. robbers*. Les joueurs sont répartis en deux équipes (voleurs et policiers). Le but des policiers est de capturer les voleurs (un voleur est capturé lorsqu'il se trouve sur la même case qu'un policier). Le but est des voleurs est de récupérer toutes les pièces d'or.

11 Tâche 9 : TCP et UPD

Faire en sorte que vos serveurs puisse gérer indifféremment des clients utilisant UDP ou TCP.

12 Tâches 10 : améliorations

Implémenter une ou plusieurs des améliorations suivantes :

1. *Bonus/malus* : en plus des pièces d'or, on peut trouver d'autres objets qui apportent des malus ou bonus au joueur qui les récupèrent. Par exemple :
   — `B` (break wall) : une fois ce bonus récupéré, le joueur peut casser les murs qui l'entoure (une seule fois) en appuyant sur la touche `B`.

   — `S`(slow) : une fois ce malus récupéré, le joueur est ralenti pour une durée fixée (par exemple 30 secondes)
   — `T` (teleport) : ce bonus téléporte le joueur sur une case aléatoire

— `D` (dragon) : s'approcher de cette case réveille le dragon. Il parcourt le labyrinthe en mangeant les joueurs qui croisent son chemin.

On pourra en imaginer d'autres.

2. *Choix des paramètres d'une partie.* Lorsqu'un joueur se connecte, il indique quel type de partie il souhaite disputer : standard ou cop vs. robber, avec ou sans bonus, solo ou multi-jouer. Dans le cas d'une partie solo, les adversaires et/ou équipiers sont des bots. Le serveur *dispatch* devra faire en sorte que des joueurs souhaitant disputer le même type de partie se retrouve sur le même serveur de jeu.
3. *Visibilité.* Jusqu'à maintenant, un autre jouer n'est visible que s'il est dans le voisinage immédiat d'un joueur (c'est-à-dire sur une case adjacente non séparée par un mur.). Ajouter la possibilité de créer des parties dans lesquelles la visibilité est un paramètre choisi au démarrage de la partie.
4. *Mode massivement multi-joueurs.* On souhaite pouvoir jouer avec un très grand nombre de joueurs sur des labyrinthes de très grande taille.
5. *Architecture pair-à-pair.* Le coût de maintenance des serveurs de jeux devient trop important. Les parties sont maintenant hébergées par les joueurs eux-mêmes. On pourra quand même garder des serveurs de types *dispatch* qui maintiennent à jour les adresses des joueurs hébergeant des parties, ainsi que le leader-board.