Rapport : Tchu Extension

Table des matières

Description

Menu

Le menu est composé de 4 boutons : Solo, Host, Join a Game, Exit. Le mode Solo amène l'utilisateur à jouer contre un bot. Host permet à l'utilisateur d'héberger une partie, lorsque l'utilisateur clique sur ce bouton, un dialogue se crée ou il peut ainsi choisir le nombre de joueurs d'une partie ainsi que le nom des joueurs de la partie. Enfin, il reçoit le port et l'IP qu'il transmettra à ses adversaires. Si le joueur clique sur le bouton « Join a Game », alors il a le rôle inverse, c'est-à-dire qu'il doit se connecter à l'host de la partie. En effet, un dialogue se crée ou il doit entrer l'IP et le port de l'host. Une fois tous les joueurs connectés, la partie se lance.
Si l'utilisateur clique sur le bouton Exit, alors le menu se ferme.
Ces fonctionnalités sont implémentées dans la classe MenuViewCreator du package extension. En effet, cette classe est une application javafx qui appelle des méthodes selon le bouton choisis.

Couleur des wagons + new Info

Dans la version de l'étape 11, on ne possédait que 2 couleurs de wagons. Pour cela on a modifié les fichiers map.css et colors.css afin que l'on puisse avoir jusqu'à 5 couleurs. De plus, le nombre de wagons que possède chaque joueur a été modifié, puisque sinon la partie ne peut se terminer avec si peu de routes. Ainsi, on a écrit une formule en fonction du nombre de joueurs dans Constants pour le nombre de wagons. Enfin, le message won a été adapté pour plus 2 joueurs. Pour résoudre cela, on a écrit dans StringsFr une nouvelle constante WINS_MULTI, et une nouvelle méthode dans Info qui se nomme wonMulti pour afficher un message de victoire dans une partie avec plus de 2 joueurs.

Mode multijoueur (plus de 2 joueurs)

On a fait en sorte de pouvoir jouer avec 5 joueurs maximum. On a premièrement modifié PlayerId afin de satisfaire le nombre de joueurs que souhaite l'host, on a donc rendu les attributs COUNT et ALL privés et non finals. Ils possèdent maintenant des getters ainsi qu'un setter, qui contrôle que la valeur passé en argument est correcte. On a d'ailleurs mis en place que le setter ne peut être appelé seulement 1 fois à l'aide de l'attribut nrChange. Ensuite, on a adapté la méthode initPlayers de RemotePlayerProxy, ainsi que le cas de l'initialisation des joueurs dans la méthode run de RemotePlayerClient : en effet, le client doit être au courant du nombre de joueurs de la partie, c'est pour cela que l'on appelle la méthode setNbrPlayer de PlayerId. De plus, l'attribut publicGameStateSerde de la classe Serdes a aussi dû être adapté afin de convenir à n joueurs. La fin de partie pour Game a dû être implémentés pour un nombre de joueurs supérieurs à 2. On peut ainsi préciser le cas où plusieurs joueurs sont à égalité (mais pas tous) et le cas où il n'y a qu'un seul vainqueur.

ServerMain a aussi dû être adapté afin de créer un proxy pour chaque client. On a transféré le code qui était dans start pour le mettre dans une méthode publique server afin de l'appeler dans MenuViewCreator, on a fait de même avec ClientMain ou on a déclaré la méthode client. Pour ServerMain, On a aussi ajouté à cela les messages dans le terminal lorsqu'un joueur rejoint une partie, ainsi que l'alerte (qui appelle showMyIpAdress) pour que l'host informe son IP et son port à ses adversaires.

BotPlayer

Description

Pour permettre de mieux tester notre jeu et de faire des parties contre l'ordinateur, nous avons choisi de concevoir un bot. Au début de la partie la sélection des billets par le bot se fait aléatoirement.

Puis à chaque tour :

1. Le bot vérifie s'il a besoin d'une nouveau chemin cible -> s'il n'y a pas encore de chemin cible ou si il n'est plus capturable (le joueur adverse a pris une des routes lui appartenant...)
2. S'il a besoin d'un nouveau chemin cible, un nouveau entre 2 stations composant l'un des tickets du bot sera régénéré en utilisant la nouvelle méthode Trail. shortest ()
3. -> si aucun chemin n'est créable pour chacun des tickets du bot ou si tous les tickets ont été complétés, les tickets sont considérés comme fait et le bot tire de nouveaux tickets du paquet de tickets.
4. S'il a un chemin cible valide, le bot essaie de la compléter en prenant l'une des routes qui lui appartiennent.
5. Dans le cas où le bot a un chemin cible mais ne peut prendre aucune des routes qui y appartiennent, il regarde dans les cartes faces visibles, si une des cartes qui s'y trouvent peut être utilisée pour prendre une route de son chemin cible ; si c'est le cas, il la prend, sinon il tire une carte de la pioche.

Implémentation

- Méthode : Trail.shortestTrail() : Sur le même principe que longest, on teste toutes les possibilités pour garder uniquement la plus courte
- Class : Botplayer implements Player : agit de manière similaire à graphicalPlayer excepté que à la place d'ouvrir une fenêtre pour proposer les choix à l'utilisateur, les choix sont directement effectués par le bot sur la logique cité plus haut (Description)
- Méthode : Botplayer. generateTargetTrail () : parcours les tickets et pour ceux qui non pas été capturé on appelle la méthode shortestTrail () avec toutes les routes possédées ou encore capturable (sans prendre en compte les cartes(wagon/locomotive)) et la station de début et fin du ticket.

Affichage des tickets

Description

Pour permettre au joueur de plus rapidement identifier quel ticket, il a déjà complété et l'impact de ces tickets sur son score final, nous avons modifier l'affichage de la liste des tickets pour y inclure leur valeur, sous forme de points positif si complété et négatif si encore non complété. La couleur du texte dépend du même de son statut vert si complété sinon noir.

Implémentation

- Ajout d'une redéfinition de la classe ListCell gérant l'affichage des tickets dont ont redéfini la méthode updateItem pour adapter l'affichage du texte et la couleur du texte.
- Pour déterminer si un ticket a été complété, une propriété ticketsPlayerPoints avec son getter correspondant a été ajouter à ObservebaleGameState contenant le nombre de point de chaque ticket. Ce nombre est actualiser dans setState en fonction du nouvel état du joueur (On y ajoute tous les billets du joueur(foreach) en récupérant leur nombre de point en utilisant la méthode points du ticket avec la méthode points de Ticket et comme argument la valeur retourner par la méthode stationConnectivityPlayer () appliqué sur le nouveau PlayerState.