Sujet 5

Jeu Isola

M3105 BREHON Antoine && GRAFFAN Jérémy

SOMMAIRE

INTRODUCTION

Présentation du sujet :

L'objectif du sujet était de réaliser en java un jeu appelé Isola.

Ce dernier consiste en un affrontement entre deux joueurs sur un terrain de 6x8 cases. Ils doivent réussir à isoler leur adversaire avant de l'être soi-même. Pour cela, ils doivent chacun leur tour se déplacer sur une case adjacente à celle où ils se trouvent, puis détruire n'importe quelle case sur le plateau. Un joueur a perdu si à la fin du tour il est entouré de cases détruites ou occupées par l'autre joueur.

ANALYSE

Liste des classes :

Utils
ImageUtils MathUtils Matrix

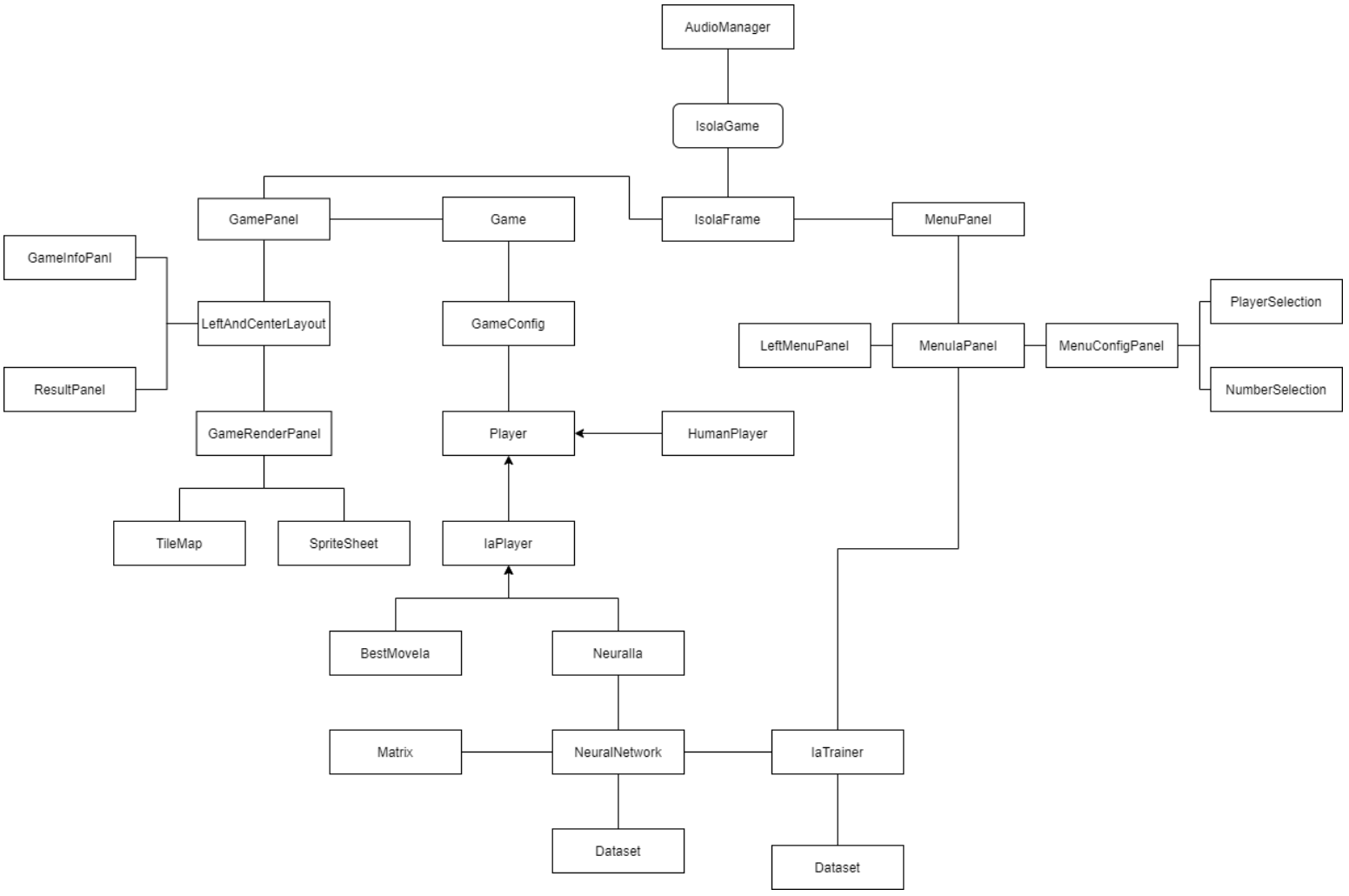
Audio
AudioManager

DeepL
NeuralNetwork Dataset IaTrainer

Game
Game GameConfig BestMoveIa HumanPlayer IaPlayer Player Point NeuralIa

UI
IsolaFrame TileMap SpriteSheet MenuButton NumberSelection PlayerSelection LeftAndCenterLayout GameInfoPanel GamePanel GameRenderPanel LeftMenuPanel MenuConfigPanel MenuIaPanel MenuPanel ResultPanel

Diagramme de classe :



Fonctionnement global :

L'application se sépare en deux parties distinctes : Le jeu et l'affichage.

Pour le jeu :
Une partie est définie par ces 2 joueurs (humain ou intelligence artificielle) et les dimensions du terrain.
Une fois la partie créée on l'initialise et on positionne de manière aléatoire les joueurs sur la carte. Afin de garantir une certaine équité, le joueur 1 sera placé de manière aléatoire sur la moitié gauche du plateau, et le joueur 2 sur la moitié droite.

Une fois que cela est fait, on démarre un tours qui se déroule de la manière suivante :

- ➔ Déplacement du joueur : Si le joueur est humain, on attend son choix, sinon on demande à l'IA un choix
- ➔ Si le choix proposé est invalide, on recommence la première étape
- ➔ On déplace le joueur
- ➔ Destruction d'une case : Si le joueur est humain on attend son choix, sinon on demande à l'IA un choix
- ➔ Si le choix proposé est invalide on recommence l'étape précédente
- ➔ On détruit la case
- ➔ On vérifie si un joueur a gagné
- ➔ Si non, on change le joueur actif et on recommence un tour. Si oui, on affiche le résultat de la partie.

Pour l'affichage :
Il se décompose en deux parties : le menu et le rendu du jeu.
Le menu permet de créer une partie, d'entraîner une IA ou de quitter l'application.
Le rendu du jeu affiche les différentes "tiles" et les joueurs sur l'écran, et attend un changement d'état du jeu afin de modifier l'affichage.

Réalisation

Choix techniques :

Architecture de l'application :

Nous avons décidé d'essayer de séparer au maximum les différents composants comme l'affichage du jeu et la gestion du jeu afin d'avoir un code plus maintenable et clair.
Par conséquent, ce n'est pas le jeu qui demande l'affichage d'un élément mais l'affichage qui affiche des éléments en fonction de l'état du jeu. Aucune fonction d'affichage ou d'entrée utilisateur n'est présente dans la partie de gestion du jeu.

L'utilisation de librairies externes :

Dans un premier temps, nous souhaitions utiliser des libraires externes afin de faciliter la conception de l'application tout en améliorant sa qualité. Cependant, les libraires que nous souhaitions utiliser possèdent des contraintes, ce qui nous a finalement poussé à ne pas en utiliser. Nous avons donc recréé nous-même une partie des modules concernés.

Voici quelques exemples de librairies et de leurs contraintes :

- ➔ DeepLearning4J (Librairies de deep learning) : Librairie trop volumineuse (environ 0.5Go), ce qui posait des problèmes pour le rendu du projet, par conséquence, nous avons implémenté une version ultra simplifiée de Deep Learning correspondant à notre besoin.
- ➔ LWJGL (Librairie OpenGL) : La librairie nécessite des natives propres à chaque plateforme de production, ce qui posait aussi des contraintes pour le rendu final.

Actualisation du rendu :

Étant donné que l'application comporte un certain nombre d'élément graphiques qui nécessite une actualisation régulière de leur affichage, nous avons decidé de ne pas l'actualiser séparément pour chaque élément en fonction des actions, mais plutôt d'actualiser de manière générale l'affichage de la fenêtre à un rythme de 30 images par seconde afin de faciliter la conception, et d'éviter des conflits.

Ia de Base (BestMoveIa) :

Cette intelligence artificielle est celle par défaut de l'application, elle fonctionne sur le principe de prédiction de la sûreté d'une case.

Pour le déplacement, elle regarde les 8 cases qui l'entourent. Pour chacune de ces cases, on calcul un score de sûreté qui est égal à la somme des cases libres (c'est à dire sur lesquelles on peut se déplacer) qui entoure cette case.
Par la suite, l'IA décide de se déplacer sur la case qui a le plus grand score de sûreté.

Pour la destruction d'une case, le principe est le même que pour le déplacement mais avec le joueur adverse afin de lui supprimer son meilleur choix de déplacement.
Pour cela, elle regarde les 8 cases qui entourent l'adversaire. Pour chacune de ces cases, on calcul un score de sûreté qui est égale à la somme des cases libres (c'est-à-dire sur lesquelles on peut se déplacer) qui entoure cette case.
Par la suite, l'IA décide de détruire la case qui a le plus grand score de sûreté.

Pour améliorer l'IA, on aurait pu effectuer une prédiction sur plusieurs tours en testant à chaque fois chaque déplacement possible pour le joueur adverse, cependant, après quelques tests, nous nous sommes rendu compte que la version actuelle de l'algorithme est relativement performante et suffisante pour nos besoins.

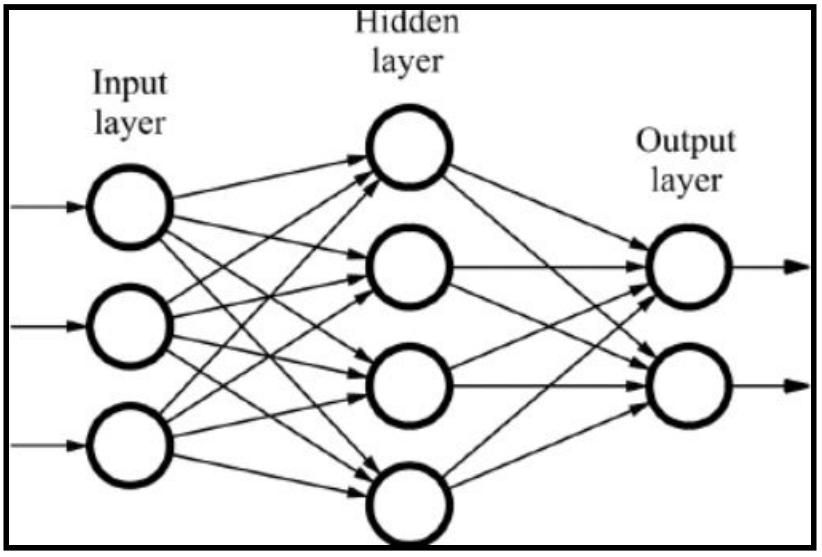
Neural Network :

Nous avons décidé d'expérimenter le DeepLearning en faisant une IA présente dans l'application. Pour les raisons expliquées dans la partie précédente sur l'utilisation de librairies externes, nous avons implémenté une version très simplifiée et basique d'un réseau de neurones qui pourrait plus s'apparenter à un joueur qu'à une vraie librairie. Mais cela nous suffit par rapport à nos besoins et au temps dont nous disposions pour mettre tout cela en place.

L'implémentation du réseau a été inspiré de la série de vidéo de The Coding Train qui a tenté de réaliser un réseau neuronal en JavaScript :
https://www.youtube.com/playlist?list=PLRqwX-V7Uu6aCibgK1PTWWu9by6XFdCfh

De plus, l'implémentation nécessitant des calculs matriciels, nous avons implémenté une version simplifiée de la gestion de matrices avec les fonctions de calculs que nous nécessitons.

Le réseau de neurone se décompose en seulement 3 couches (pas de système de multi-layer implémenté par manque de temps), une couche d'entrée dans laquelle on passe les données à traiter, une couche intermédiaire (hidden layer) et une couche de sortie qui tente de prédire les résultats.



Notre implémentation ne permettant pas de faire des modèles RNN (recurrent neural network) et LSTM (long short term memory), nos modèles ont un nombre de neurone dans chaque couche non défini et non modifiable, par conséquent, nous avons décidé de faire des modèles qui fonctionnent que pour des terrain de jeu de taille 8x6.

L'intelligence artificielle qui utilise le DeepLearning, utilise 2 réseaux de neurones, un pour le déplacement, et un pour la destruction des cases.

- ➔ Celui pour le déplacement comporte 52 neurones d'entrées (2 pour la position de l'IA, 2 pour la position de l'adversaire, et 48 pour l'état de chaque case du terrain), 48*2=96 neurones pour la couche intermédiaire (hidden layer) définit de manière arbitraire, et 8 neurones de sorties qui représentent les 8 cases autours de l'IA. On sélectionne ensuite la case qui a la plus grande valeur. Si la case sélectionnée est invalide, on utilise la prédiction de l'IA de base.

- ➔ Celui pour la destruction des cases comporte 52 neurones d'entrées (2 pour la position de l'IA, 2 pour la position de l'adversaire, et 48 pour l'état de chaque case du terrain), 48*2=96 neurones pour la couche intermédiaire (hidden layer) définit de manière arbitraire, et 48 neurones de sorties qui représentent les 48 cases du terrain. On sélectionne ensuite la case qui a la plus grande valeur. Si la case sélectionnée est invalide, on utilise la prédiction de l'IA de base.

L'entrainement des réseaux se base des DataSets (ensembles de données) extraites des différentes parties réalisées. L'entraînement actuel est réalisé sur une moyenne de 1000 données pour chaque modèle avec 100 000 itérations d'entraînement.

Ils sont ensuite sauvegardés dans un fichier en ".model" qui est un simple fichier texte qui comporte la valeur des poids entre chaque couche et des bias correspondants à chaque couche.

Utilisation

Mode d’emploi :

Vous pouvez compiler et exécuter les sources manuellement ou avec l’IDE Intellij Idea. Cependant un fichier “jar” est fourni permettant de lancer directement le programme en cliquant dessus.

Configuration requise :

Logiciel : Une JVM avec au minimum Java 8 (version utilisée pour le développement).
Processeur : Un mono-coeur pas trop vieux
Ram minimum : 100 Mo
Ram recommandée : 200 Mo

Conclusion

Bilan :

Le programme que nous avons créé permet donc de jouer au jeu Isola, que ce soit en joueur contre joueur ou en joueur contre IA. Mais si le jeu fonctionne parfaitement tel qu'il est actuellement, des optimisations sont possibles et des extensions sont envisageables.

Optimisation possibles :

Nous avons déterminé plusieurs optimisations possibles :

- ➔ Amélioration de la gestion de la TileMap du rendu du terrain.
- ➔ Améliorations du module de Neural Network en optimisant les calculs matricielles, et en implémentant un système de multi-layering.
- ➔ Une meilleure gestion de l'audio (modifier le volume via un menu...)
- ➔ Amélioration du lien affichage-jeu avec un système d'événements globaux.

Extensions possibles :

Il est encore possible d'améliorer le projet en ajoutant plusieurs extensions comme :

- ➔ Un mode multijoueur en ligne avec pourquoi pas du multijoueur compétitif avec un système de points et de classement en fonction du nombre de victoires en PvP.
- ➔ D'autres IA (algorithme génétique, QLearning, DeepQLearning ...).
- ➔ Un mode jeu personnalisé avec par exemple des cases à effets.
- ➔ Un mode de jeu à plus de 2 joueurs sur une carte plus grande.
- ➔ Ajouter des effets visuels comme des particules pour améliorer le retour utilisateur.
- ➔ Si l'on exagère, on pourrait imaginer un mode histoire ou entraînement avec des puzzles (choisir les meilleurs action dans un cas prédéfinis et scripté) à résoudre.
- ➔ Rajouter la possibilité au joueur de choisir/rajouter son propre avatar personnalisé.