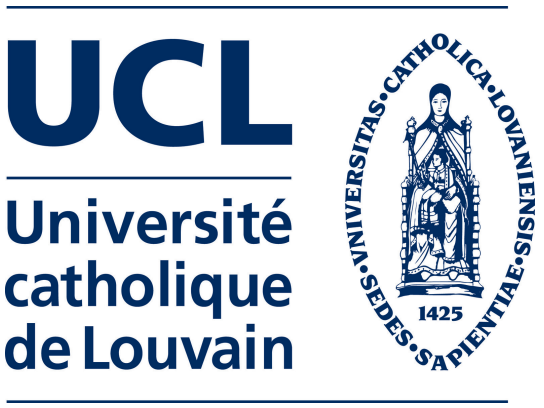

LINGI1131

Computer Language Concept

Project: PacmOz

Professor : Peter Van Roy

Authors : Jacques Yakoub, Sophie Madessis

2018

1 Introduction

Dans le cadre du cours de **Computer Language Concept** donné par le Professeur Van Roy nous avons implémenté un jeu inspiré de PacMan se basant sur les techniques de *message passing* développées dans ce cours. Les sections suivantes décrivent l'implémentation du jeu, les difficultés rencontrées, les séances d'interopérabilité avec les autres étudiants ainsi que les améliorations que nous avons apportées à l'énoncé de base.

2 Implémentation

Nous commencerons par expliquer chacun des nouveaux fichiers que nous avons jugé utile d'ajouter

2.1 Fichiers additionnels

WarningFunctions.oz :

Contient des fonctions utilitaires pour prévenir le GUI et les joueurs des updates de la situation (nouveau point, bonus, etc)

CommonUtils.oz :

Contient des fonctions utilitaires pour diverses taches un peu partout dans le code : génération de nombre random, validateur de mouvements , etc...

StateWatcher.oz :

Un agent indépendant entièrement dédié à la gestion de l'état courant de la partie. Ses principaux messages qu'il reçoit sont les suivants :

- checkTimers(?F) : Vérifie chaque respawnTime (respawnTimePoint/respawnTimeBonus/respawnTimePacman/respawnTimeGhost) et effectue le respawn de leurs éléments respectifs si la condition de temps correspondant à leur règle définie selon le type de jeu doit s'activer. A la fin, F est assignée à true si le jeu est terminé (0 pacman avec de la vie) , false sinon.

- move(CurrentPlayer Position) : Effectue la décision du joueur de bouger à la position demandée. Par la suite, celle-ci détermine et gère plusieurs situations :
  - Si le message doit être traité (c'est à dire si le joueur est toujours en vie au moment de la réception de ce message)
  - Si le joueur tue un ou plusieurs adversaires (selon le mode actuel de la partie)
  - Si le joueur se fait éliminer par un adversaire (selon le mode actuel de la partie)
  - Si le joueur (étant un pacman) est toujours en vie et qu'il peut récupérer un point ou un bonus.

  Le nouvel état du jeu est ensuite enregistré.

  Précision : Position correspond au type <position> , CurrentPlayer correspond à un record ayant un field id (soit de type <pacman> soit de type <ghost>) et d'un field port (correspondant au port du joueur)

- increaseTurn : Incrémente le numéro du tour / temps actuel (selon le type de jeu)

- displayWinner : Calcule le Pacman vainqueur et prévient le GUI

L'état conservé par StateWatcher est donc représenté sous la forme d'un record avec divers champs représentatifs de la situation du jeu.

2.2 Contrôleur Principal

Grâce au StateWatcher, le Main n'a qu'un nombre réduit de tâches à réaliser :
— Exploration de la map pour en extraire les GhostSpawn , PacmanSpawn , BonusSpawn et PointsSpawn
— Initialisation du jeu (création des joueurs, assignSpawn et init/spawn des points/bonus sur le tableau)
— Déclenchement du StateWatcher avec un état initial
— Lancement de la partie : soit le mode turnByTurn (LaunchTurn) soit le mode simultané (LaunchSimultaneous)

2.2.1 Implémentation des deux types de jeu

Pour illustrer, voici les actions réalisées lors d'un appel de leur procédure respective :

turnByTurn
- Vérification des respawnTime (et de la fin du mode hunt)
- Si la partie est terminée, demander l'affichage du gagnant
- Prendre le premier joueur de la liste (NDLR : Il s'agit d'une liste récursive consistant en une répétition infinie de l'ordre de passage des joueurs)
- Demander au joueur sa position et envoyer au StateWatcher
- Si on a effectué un tour complet, on prévient StateWatcher d'incrémenter le numéro du tour
- On rappelle cette fonction avec le reste de la liste

simultaneous
- Vérification des respawnTime (et de la fin du mode hunt)
- Si la partie est terminé, demander l'affichage du gagnant
- Interrogation de tous les joueurs de manière simultané et collecte des réponses (grâce au WaitAgent , un agent défini dans le main.oz)
- Attendre jusqu'à que tous les joueurs répondent grâce à une interrogation jusqu'à obtenir la confirmation par le WaitAgent
- Si c'est le cas, le WaitAgent prévient le StateWatcher avec les messages collectés et réappelle la fonction principale en prenant soin de l'avertir d'updater son temps actuel.

2.3 Images

Afin d'afficher des images pour les Pacmans, Ghost, Murs, Bonus et Points nous avons utilisé le module GTK pour importer les images et les placer ensuite dans les records Label. Les images ont été trouvées sur différents sites et on été modifiées pour correspondre au format 32x32 pixels GIF.

http ://www.softicons.com
https ://icons8.com

3 Difficultés Rencontrées

Les plus grandes difficultés étaient :

La réalisation de la méthode principale StateWatcher qui gérait le message move : handleMove. Comme expliqué précédemment, cela dépendait de deux grands facteurs : l'ordre d'arrivée du message et la gestion des diverses situations . De ce fait, il fallait s'assurer que cette méthode fonctionne bien en fonction de la situation.

En ce qui concerne l'intelligence artificielle derrière le Pacman Super Smart, il est difficile d'éviter des cas exceptionnels où l'on a l'heuristique qui nous fait osciller entre deux cases indéfiniment. Nous aurions pu garder les états précédents en mémoire et lui faire choisir un mouvement aléatoire après une certaine série de mouvements similaires mais l'implémentation nous semblait assez compliquée.

4 Interopérabilité

Une première version du Pacman Super Smart a été envoyée au groupe 61. Leur choix de Map a mis notre Pacman dans un état de boucle infinie, ne pouvant pas se rapprocher des bonus car leurs ghosts bloquaient le chemin. Nous n'avons pas pu corriger le problème facilement comme expliqué précédemment dans la section difficultés rencontrées.

Une deuxième erreur a été détectée grâce à un Input d'un autre groupe différent qui nous a fait remarquer que notre heuristique était influencée par le nombre de Ghosts et de Bonus sur la map. En effet selon nos calculs plus il y a de Ghosts plus leur présence aura de l'influence sur le comportement du Pacman si on ne divise pas par le nombre de Ghost.

Nous avons testé avec le groupe 61, le 85 et le 80. Le premier groupe avait un joueur humain mais étant donné qu'il fallait changer notre GUI notre Main nous nous sommes concentrés sur les joueurs qui fonctionnaient normalement. Nous avons pu voir que notre Pacman était généralement plus efficace que ceux des autres groupes, mais que certains de leurs Ghosts faisaient tomber nos Pacmans dans des boucles infinies et vice-versa.

5 Améliorations

Nous avons implémenté un Pacman amélioré en se basant sur le système d'heurstique tel qu'appris au cours d'Intelligence Artificielle donné par le Professeur Deville au premier quadrimestre.

5.1 Pacman Super Smart

Le Pacman Super Smart a été implémenté de façon à se diriger vers les bonus présents, à éviter les Ghosts ainsi qu'à favoriser une case contenant un point plutôt qu'une case vide. Lorsque le mode 'Hunt' s'active, il maintient son comportement excepté qu'il se dirige vers les Ghosts.

Pour choisir son prochain mouvement, le Pacman va d'abord consulter les quatre nouvelles positions possibles : haut, bas, gauche, droite. La fonction *WrappingMoves* se charge de gerer les cas de dépassement de la map et la fonction **ValidMoves** renvoie une liste avec les mouvements qui sont autorisés en tenant compte des murs.

Afin de choisir parmi cette liste, le Pacman va se baser sur l'heuristique décrite précédemment. Pour ça il utilise la fonction *BestMove* qui en fonction du Mode (classic|hunt) donné en argument, défini la meilleure position à choisir. La fonction calcule et attribue une valeur à chaque position : celle avec la plus haute valeur se verra être la position choisie.

Pour attribuer une valeur à une position nous allons utiliser une combinaison linéaire basée sur trois paramètres différents : la position des **Ghosts**, celle des **Bonus** présents ainsi que celle des **Points** présents.

Pour calculer la valeur attribuée en fonction des **Ghosts** présents, ces derniers sont passés en attributs et la distance de Mannathan est calculée entre cette position et celle de chaque Ghost.
Se trouver loin des Ghost est avantageux par dans le cas du Mode classic et désavantageux dans le cas du Mode hunt.

Pour ce qui concerne les bonus, s'en rapprocher est avantageux dans tous les cas. La formule qui suit permet de favoriser le rapprochement vers un Bonus en particulier plutôt que d'être à distance égales de deux Bonus présents sur la Map. La formule est : $(Bonus1 + Bonus2)^2 - (Bonus1^2 + Bonus2^2)$

Pour définir la valeur qu'aurait une case avec Point par rapport à une case sans, nous avons arbitrairement défini la valeur 10 car nous préférons effectivement prendre un bonus plutôt qu'un Point, mais nous préférons un point à une case vide.

5.2 Pacman Super Shy

Le Pacman suit la même structure que le précédent excepté qu'il fuit les ghosts sans se soucier d'attraper les bonus qui se présentent devant lui.

5.3 Pacman Other

Ce Pacman a un comportement local, il regarde autour de lui et s'il voit un point va vers celui-ci, et s'il voit un ghost va éviter de se jeter dessus.

5.4 Ghost Other

Le player Ghost Other prend en argument la liste des positions des Pacmans, choisit le Pacman le plus proche et décide de le suivre.

6 Conclusion

Nous avons eu par la réalisation de ce projet réaliser de manière concrète les différents concepts appris lors de ce cours. Le jeu est fonctionnel dans son ensemble, tant en mode tour par tour que simultané. Notre interface graphique suit le jeu efficacement.
Si nous avions eu plus de temps, nous aurions aimé améliorer le comportement des Pacmans et Ghost et en créer davantage de nouveaux.