Projet LINGI1131 : PacmOz

Florence Blondiaux 06521500, Noémie Verstraete 25021500

Avril 2018

1 Introduction

Dans ce rapport, nous expliquerons brièvement notre implémentation du projet PacmOz dans le cadre du cours LINGI1131. Nous décrirons brièvement notre structure, les choix que nous avons faits, et la modification du fichier *GUI.oz* que nous avons effectuée.

2 Comportement du `Main.oz`

Notre *Main* se divise en deux parties. Dans la première, nous initialisons la *window* et les ports. Ensuite nous lisons entièrement la *map* et initialisons toutes les variables utiles pour lancer le jeu soit en mode *turn-by-turn*, soit en mode *simultané*. Plus particulièrement, nous stockons :

- Une liste contenant tous les ports des *pacmans*. Chaque élément de la liste est un port et les ports sont triés dans la liste par ordre chronologique des ID des *pacmans*.
- Une liste contenant tous les ports des *ghosts*. Chaque élément de la liste est un port et les ports sont triés dans la liste par ordre chronologique des ID des *ghosts*.
- Une liste contenant tous les IDs des pacmans. Chaque élément de la liste est un ID de type $< pacman >$.
- Une liste contenant tous les IDs des ghosts. Chaque élément de la liste est un ID de type $< ghost >$.
- Une liste contenant tous les IDs des pacmans et des ghosts. Chaque élément est un ID de type $< pacman >$ ou de type $< ghost >$. L'ordre des IDs dans la liste a été choisi aléatoirement (grâce à *OS.rand*) et c'est cet ordre qui sera utilisé pour déterminer l'ordre des joueurs pour le mode *turn-by-turn*. Cette liste ne sera pas utilisée pour le mode *simultané*.
- Une liste contenant tous les emplacements des points de la *map*. Chaque élément est de type $< position >$.
- Une liste contenant tous les emplacements de bonus de la *map*. Chaque élément est de type $< position >$.
- Une liste contenant tous les emplacements "murs" de la *map*. Chaque élément est de type $< position >$.
- Une liste de records pour stocker les emplacements "spawn" assignés aléatoirement à chaque pacman. Chaque élément de la liste est de type $< pacman >\#< position >$.
- Une liste de records pour stocker les emplacements "spawn" assignés aléatoirement à chaque ghost. Chaque élément de la liste est de type $< ghost >\#< position >$.

C'est également dans la première partie de la *Main* que nous envoyons tous les messages nécessaires aux pacmans et aux ghosts. Par exemple, nous informons les pacmans de leurs emplacements "spawns" respectifs, de l'emplacement des points, bonus, ghosts, etc.

Dans la deuxième partie de la main, nous lançons le jeu soit en mode *turn-by-turn*, soit en mode *simultané* en fonction de ce qui est demandé dans le fichier *input.oz*.

2.1 Mode Turn-by-Turn

Dans le mode turn-by turn, nous créons un "port object" `Server` qui appelle une procédure récursive `ServerProc`. Cette dernière prend en argument le Stream et l'état (State) du jeu. Pour chaque message reçu sur le Stream, `ServerProc` se rappellera récursivement avec comme argument le Stream et l'état (State) du jeu modifié adéquatement.

L'état du jeu (State) est un record de type `state(posPac:PosPac posG:PosG posB:PosB posP:PosP m:Mode hT:HuntTime pacT:PacTime gT:GTime bT:BTime pT:PTime)` où :

- PosPac est une liste de records. Chaque record est de type $< pacman > \# < position >$. PosPac représente tous les pacmans vivants ("on board").
- PosG est une liste de records. Chaque record est de type $< ghost > \# < position >$. PosG représente tous les ghosts vivants ("on board").
- PosB est une liste de $< position >$. Chaque $< position >$ représente un bonus affiché sur la map.
- PosP est une liste de $< position >$. Chaque $< position >$ représente un point affiché sur la map.
- Mode est de type $< mode >$. On stocke dans ce champs le mode actuel du jeu (classic ou hunt).
- HuntTime est un nombre. Il représente le nombre de joueurs qui doivent encore jouer avant de repasser en mode classic. Tous les temps de l'input sont multipliés par le nombre de ghost et de pacman, en effet on décrémente cette valeur à chaque fois qu'un joueur joue.
- PacTime est une liste de tuples $< id >\#< Temps >$. Temps représente le nombre de joueurs qui doivent jouer avant que le pacman respawn.
- GTime est une liste de tuples $< id >\#< Temps >$. Temps représente le nombre de joueurs qui doivent jouer avant que le ghost respawn.
- BTime est une liste de tuples $< position >\#< Temps >$. Temps représente le nombre de joueurs qui doivent jouer avant que le bonus respawn.
- PTime est une liste de tuples $< position >\#< Temps >$. Temps représente le nombre de joueurs qui doivent jouer avant que le point respawn.

Nous créons également une procédure récursive terminale `ClientFonc`. Cette procédure prend en argument un message (nombre valant 0 ou 1) et le port du Server. Le message sert à gérer la fin du jeu. S'il est égal à 1, cela signifie que tous les pacmans sont définitivement morts. A ce moment, la procédure affiche le vainqueur puis se termine. Si par contre, le message est égal 0, alors le jeu doit continuer, et dans ce cas l'appel de la procédure servira à faire le déplacement d'un joueur et tout ce que cela implique. Nous commençons par décrémenter le temps des pacmans, ghosts, bonus et points qui attendent pour être à nouveau affichés sur la map. Si le jeu est en mode hunt, nous décrémentons aussi le temps restant avant de repasser en mode classique.

Ensuite, si le joueur est un pacman, la procédure récursive agit de la sorte :

- On demande au pacman de jouer
- Si mode == classic
  - Regarde si des ghosts sont sur sa case si oui, il meurt.
- Si mode == hunt
  - Regarde si des ghosts sont sur sa case si oui, les ghosts meurent.

Si par contre le joueur est un ghost, la procédure récursive agira plutôt comme ceci :

- On demande au ghost de jouer
- Si mode == classic
  - Regarde si des pacamns sont sur sa case si oui, les pacmans meurent.

- Si mode == hunt
  - Regarde si des pacmans sont sur sa case si oui, le ghost meurt.

Après ceci, `ClientFonc` va demander au server si c'est la fin du jeu, c'est-à-dire si tous les pacmans sont définitivement morts ou non. Si ils ne reste que des ghosts en jeu, l'appel récursif se fera avec le message égal à 0, et dans l'autre cas, avec le message égal à 1.

2.2 Mode simultané

Le mode simultané est un peu différent du mode turn-by-turn. Il y a toujours un "port object" `Server` qui appelle une procédure récursive `ServerProc2` qui agit exactement de la même manière que pour le mode simultané. Il y a cependant quelques différences : dans les messages que peut recevoir le Stream et dans l'état du jeu passé en argument. Certains messages que le Stream peut recevoir ont pu être repris tels quels des messages du mode turn-by turn tandis que d'autres sont plus spécifiques au mode simultané. En ce qui concerne l'état du jeu (le "State" passé en argument dans la procédure récursive), il est assez semblable à celui du mode simultané, mis à part qu'il contient moins de champs. En effet, les champs pacT, gT, bT, pT et hT ne sont plus nécessaires.

Gestion du délai. Une des particularité du mode simultané est le temps que doit attendre un joueur avant de pouvoir faire un déplacement sur la map et le temps que doit attendre un pacman, un ghost, un bonus ou un point avant d'être à nouveau affiché sur la map. Pour gérer ces délais, nous avons choisi de créer un "port object" `PortTimer`. Nous nous sommes inspirées du protocole Timer expliqué à la page 368 du manuel de référence.[1] Ceci permet au serveur de ne pas être bloqué pendant l'attente des délais et de continuer à recevoir d'autres messages pendant ce temps. Ainsi, par exemple, dans le cas où un point a été mangé par un pacman et doit attendre un certain temps avant de pouvoir à nouveau être affiché sur la map, le Stream du `PortTimer` recevoir un message `starttimerPoint(...)`, qui attendra le délai demandé dans `Input.oz` puis renverra au Server le message `stoptimerPoint(...)`.

3 Modification du `GUI.oz`

Un ajout d'images à été fait, au moyen d'un label rajouté lors de l'initiation des ghosts, pacmans et lors de la lecture de la map. Les immages sont importées au début. A présent, les ghosts sont représentés par les cartes d'Alice au pays des merveilles. Les pacmans sont représentés soit par alice, par le chat ou le lapin. Le bonus est représenté par une potion, les points par des champignons, les spawns des ghosts par le dos d'une carte à jouer et enfin les spawns des pacamns par un trou de serrure.

4 Comportement du pacman et du ghost

Le principe de base est le même pour les deux joueurs, il va choisir la meilleure case parmis les 4 cases qu'il à autour de lui. Que ça soit pour le ghost ou pour le pacman, les positions correspondant à un mur sont éliminées. Pour le pacman, lorsqu'il est en mode classic, il choisira sa prochaine position au hasard parmis les positions valides c'est à dire les cases voisine qui ne sont pas des murs et qui ne contiennent pas de ghost. Par contre lorsqu'il entre en mode hunt, les positions avoisinantes contenant un ghost sont privilégiées. Pour le ghost le principe est le même, en mode classic, les postions contenant des pacmans sont favorisées et en mode hunt elles sont éliminées au même titre que les cases contenant un mur.

5 Interopérabilité

Nous avons échangés nos pacamns et ghost avec 3 autres groupes, les groupes 18, 47 et 60. Certaines petites erreurs on pu être découverte grâce à cet échange. Nous avons pu découvrir que le groupe 47 envoyait un message ne respactant pas la synthaxe. que lorsque les pacmans apparaissaient pour la première fois sur la map, les ghosts n'étaient pas prévenus de leurs postion. Ce qui posait problème si le pacman mourrait pour la première fois.

[1] Peter Van Roy, Seif Haridi, "Concepts, Techniques, and Models of Computer Programming"

5.1 Groupe 18

Le groupe 18 à pu se rendre compte qu'ils envoyait une valeur qui était déjà liée alors qu'elle devait être liée suite à l'action du message. Ils ont également découvert que lorsque le pacman spawn pour le première fois, ils ne prévenaient pas les ghosts de cette nouvelle position de pacamn. Celà posait donc un problème si le pacman mourrait avant de jouer pour la première fois. Le ghost ne savait pas le supprimer de sa liste vu qu'il ne le connaissait pas. L'utilisation de leurs pacamns et leurs ghost dans notre map ne causait pas d'erreurs mais au vu de la stratégie appliquée par ceux ci, chaque joueur effectuait ses mouvements sur un nombre limité de cases, ils ne se croisaient donc jamais et le jeu boucle indéfiniment.

5.2 Groupe 47

Chez nous tout fonctionnait pour leurs pacmans et ghosts. Un des messages envoyé par l'autre groupe au pacman ne respectait pas la spécification, ce problème fut résolu rapidement. Cependant, lorsque notre ghost et notre pacman sont utilisés avec leur main, nous arrivons à un moment ou le ghost doit supprimer de son état un pacman qui est mort. Mais le ghost ne connait pas ce pacaman. Par manque de temps, le problème n'a pas été exploré par l'autre groupe mais il se pourrait que celà soit la même chose qu'avec le groupe 18.

5.3 Groupe 60

Un problème à été rencontré. En utilisant nos pacmans dans leur main, à un moment nous reçevions un killGhost avec comme ID un port. Ce problème n'à pas été réglé. Lorque l'on exécute leurs codes avec notre main, tout se déroule comme prévu.