Université Catholique de Louvain

Computer Language Concepts

LINGI1131

Rapport de projet : PacmOz

Auteurs:
Noémie Bidoul
Baptiste Lapière

Professeur:
Peter Van Roy
Assistants:
Hélène Verhaeghe
Guillaume Maudoux

27 avril 2018

1 Introduction

Dans le cadre de ce projet, il nous a été demandé de programmer notre propre version du jeu Pacman, en s'intéressant particulièrement à la progammation concurrente. Nous avons donc implémenté nos propres message passing agents pour les Pacmans et les Ghosts.

2 Architecture

Notre Main repose sur l'utilisation de **port-objects**, permettant d'interagir avec différentes entités. Ils sont utilisés à deux fins distinctes: d'une part, pour la **gestion de l'ordre** entre les tours des players et les évènements qu'ils entraînent; d'autre part, pour la **gestion des accès concurrents** aux variables dynamiques de notre code. Les lectures de chaque Stream sont faite de manière concurrente, chaque fonction de lecture étant lancée dans un thread séparé au début du jeu.

2.1 Implémentation de la Main

2.1.1 Variables statiques

L'accès aux données des joueurs se fait *via* différentes listes: `Order`, l'ordre aléatoire de l'ensemble des joueurs; et `Pacmans` et `Ghosts`, des ordres aléatoires respectivement pour les Pacmans et les Ghosts. Leurs élements sont des records de la forme `<Player> ::= player(port:<Port> id:<ID>)`, avec `<ID> ::= <pacman>` | `<ghost>`. Les positions de Spawn des Pacmans et des Ghosts sont stockées respectivement dans les tuples `SpawnsP` et `SpawnsG`.

2.1.2 Variables dynamiques

Dans un souci de performance, notre Main contient quelques variables dynamiques. En effet, un code entièrement déclaratif aurait nécessité de transmettre (et parcourir, en cas de modification) la Map récursive à chaque itération, ainsi que d'ajouter une série de paramètres supplémentaires, alourdissant ainsi les fonctions.

- `Board`: Une map locale dynamique, construite sous forme d'un tuple de tuples de taille $M * N$. Chaque élément est une `Stack`, implémentée dans un functor séparé. Les éléments d'une Stack sont de type `<Item>`, avec `<Item> ::= <Player>` | `walkable` | `point` | `bonus` | `wall` Les différents éléments présents à un instant donné dans la Map sont ainsi insérés et supprimés à tour de rôle au sein du `Board`.
- `Alive`: Une cellule contenant le nombre de Pacmans en vie à un instant donné, permettant de vérifier l'état du jeu (terminé ou en cours).
- `HuntMode`: Une cellule permettant de connaître le mode du jeu à un instant donné.

2.1.3 Gestion de l'ordre

Afin de pouvoir contrôler l'ordre des tours des joueurs et d'ordonner les différentes actions qui en découlent, notre code dispose d'un port-object associé à la Main (`StreamMain`, `PortMain` et `ReadStreamMain`). Les messages acceptés par celui-ci sont:

- `start(Player)`: Lancement du premier tour du `<Player>` *Player*.
- `idle(Player N)`: Signale la mort du `<Player>` *Player*, qui devra être spawn à nouveau dans N tours (resp N. milisecondes en mode simulatenous).
- `replay(Player CurPos)`: Lancement du tour du `<Player>` *Player* se situant actuellement à la position $CurPos$.
- `spawnPoint(Pos N)`: Signale la disparition d'un point, qui devra être spawn à nouveau dans N tours (resp N. milisecondes en mode simulatenous).
- `spawnBonus(Pos N)`: Signale la disparition d'un bonus, qui devra être spawn à nouveau dans N tours (resp N. milisecondes en mode simulatenous).
- `huntTime(N)`: Signale le déclenchement du mode *hunt*, qui devra se terminer dans N tours (resp N. milisecondes en mode simulatenous).

Une première volée de message `start(Player)` est envoyée en suivant l'ordre aléatoire `Order` défini précédemment; par la suite, la gestion du Stream diffère en fonction du mode de jeu (turn by turn ou simultaneous).

En turn by turn, la fonction `TreatStreatMain` traîte les messages du `StreamMain` de manière séquentielle, garantissant ainsi l'ordre des tours des joueurs et des évèments associés. Lors de la réception d'un message possédant un argument N, celui-ci est progressivement décrémenté, et l'action finale n'est éxécutée que lors de la réception du message où $N == 1$.

En simultaneous, les messages du StreamMain sont traîtés simultanément, introduisant un facteur non déterministe dans la décision de l'ordre de jeu. Lors de la réception d'un message possédant un argument N, un délai correspondant est appliqué avant d'effectuer l'action associée.

Par ailleurs, de nombreux évènements du jeu doivent être anoncés à l'ensemble des Ghosts et des Pacmans. L'envoi d'un tel nombre de messages prenant un certains temps, deux nouveaux ports `PortPacmans` et `PortGhosts` ont été créés à cet effet, permettant à la gestion d'un message de continuer sans attendre que tous les joueurs concernés aient reçu l'information. Les fonctions de lectures des streams `StreamPacmans` et `StreamGhosts` transmettent donc simplement les messages reçus à l'ensemble des Pacmans/Ghosts.

2.1.4 Gestion des accès concurrents

En mode simultaneous, les variables dynamiques présentées ci-dessus sont souvent sujettes à des modifications et accès simultanés issus de plusieurs threads. Pour gérer cette concurrence, deux port-objects ont été créés: `PortBoard` / `StreamBoard`, associés à la map dynamique `Board`, et `PortObjects` / `StreamObjects`, associés aux deux autres variables dynamiques. Tout accès/modification d'une variable dynamique est donc envoyée sous forme de demande au port-object correspondant, qui les traite séquentiellement.

2.2 Implémentation des Players

2.2.1 Navigation sur la map

Les players Ghosts et Pacmans ont en commun leur manière de se repérer sur la map. Ce repérage s'effectue à l'aide de de cinq fonctions: `Find(P)`, qui permet d'identifier le type correspondant à la position P (donc 0 pour un spawn de point, 1 pour un mur, etc); et `North(P)`, `South(P)`, `East(P)` et `West(P)`, qui permettent de retourner les `<position>` voisine de la position P.

Ces fonctions sont également utilisée pour créer un graphe qui permettra aux players de déterminer des chemins efficaces. Une fois leur chemin déterminer, les players tiennent également compte de la dangerosité des positions alentours, suivant un certain **niveau de menace** défini ci-dessous:

```
<threat> ::=  pos(p:<position> l:<level>)
<level> ::=   -1 | 0 | 1 | 2
```

Un `level` de -1 correspond à un mur, 0 correspond à une absence de menace, 1 signale que la case visée est adjacente à un danger, et 2 signale que la case visée est occupée par un danger.

2.2.2 Graphe et Breadth-First Search

Les Pacmans comme les Ghosts utilisent un graphe généré à partir de la map fournie dans l'`Input`, pour déterminer leur stratégie. Le graphe est défini comme suit :

```
<graph> ::=    <graph>
               | <node>
               | nil
<node> ::=     node(p:<position> adj:<adjList>)
<adjList> ::=  <adjList>
               | <position>
               | nil
<edge> ::=     edge(v:<position> w:<position>)
<map> ::=      <map>
               | <edge>
               | nil
```

Un graphe est donc une liste de noeuds connaissant chacun leur propre position, ainsi que les positions de leurs voisins. Différentes fonctions permettent de le manipuler:

- `{GenerateGraph Map}`: Parcourt la map donnée dans l'`Input`, et crée un noeud pour chaque point n'étant pas un mur. Sa liste d'adjacence ne comprendra également que des points n'étant pas des murs.
- `{Bfs Point}`: Génère, à partir d'un point du graphe, la liste d'arêtes nécessaires pour trouver un chemin entre `Point` et n'importe quel autre noeud du graphe. Les chemins trouvé dans cette liste seront toujours les plus courts. La liste est gardée en mémoire par le player sous format `<map>`.
- `{Path To From Edges}`: Retourne le plus court chemin du point `From` au point `To` parmi la liste d'arêtes `Edges`.

2.2.3 Pacman091smart.oz

En plus des types définis plus haut, notre Pacman utilise les types suivants :

```
<status> ::=    status(p:<position> life:<lives> score:<score> m:<mode> spawn:<position>)
<score> ::=     0 | 1 | 2 | ...
<strategy> ::=  strat(t:<target> g:<ghosts> b:<bonuses> m:<map>)
<target> ::=    t(p:<position> r:<rank>)
<rank> ::=      -2 | -1 | 0 | <idNumG>
<ghosts> ::=    <ghosts>
                | <ghost>
                | nil
<ghost> ::=     g(id:<idNumG> p:<position>)
<bonuses> ::=   <bonuses>
                | <bonus>
                | nil
<bonus> ::=     b(p:<position> v:<bool>)
```

Le `<status>` du Pacman permet donc de stocker les informations nécessaires au respect de règles du jeu tels que le nombre de vies restantes, la position ou le score du Pacman. Il permet également de retenir la position de spawn, et de mettre à jour le mode de jeu. Les changements du status sont effectués grâce à la fonction `UpdateStatus`, qui retourne un nouveau record `<status>`.

La `<strategy>` permet au pacman de trouver le plus court chemin depuis sa `<position>` contenue dans le `<status>` vers la `<position>` de sa `<target>`.

Le `<rank>` de la cible permet d'identifier le type de cible: -1 pour un point, 0 pour un bonus, `<idNumG>` pour un ghost et -2 si le Pacman n'a pas de target. Cela permet au Pacman de définir l'importance d'une `<target>` : un `<rank>` plus élevé est choisi en priorité (sauf en cas d'`<idNumG>`). En cas d'égalité de rangs, c'est la cible la plus proche qui prime. Ces informations permettent à la fonction `UpdateTarget` de mettre à jour la stratégie du Pacman.

Les listes `<ghosts>` et `<bonuses>` servent à stocker les ID et les `<position>` des Ghosts, et les `<position>` et états (disponible ou indisponible) des bonus. Elles sont mises à jour quand le Pacman reçoit un `<bonusSpawn(P)>`, un `<bonusRemoved(P)>`, un `<ghostPos(ID P)>`, un `<deathGhost(ID)>` ou un `<killGhost(IDg IDp NewScore)>`. Ces listes permettent de d'utiliser les fonctions `<FindGhost>` et `<FindBonus>`, qui retournent respectivement le Ghost et bonus le plus proche, afin de mettre à jour la cible dans la stratégie.

A chaque déplacement, le Pacman va analyser les cases adjacentes, et y réagir en fonction du mode de jeu:

- En mode classic:
  - Si menace détectée: évitement de la menace en priorité
  - Si pas de menace: déplacement vers la position suivante du chemin vers la cible
- En mode hunt: Déplacement ininterrompu vers le Ghost le plus proche

2.2.4 Ghost091smart.oz

Le player Ghost est semblable au Pacman, avec une stratégie plus simple. En plus des types communs aux Pacmans et Ghosts, il utilise les types suivants:

```
<status> ::=    status(p:<position> m:<mode> spawn:<position>)
<strategy> ::=  strat(t:<position> p:<pacmans> m:<map>)
<pacmans> ::=   <pacmans>
                | <pacman>
                | nil
<pacman> ::=    p(id:<idNumG> p:<position>)
```

Le `<status>` et la `<strategy>` jouent les mêmes rôles que les types correspondant du Pacman, mais contiennent moins d'informations. Le Ghost utilise une stratégie très similaire à celle du Pacman, excepté qu'il n'a besoin de comparer les cibles possible qu'en fonction de la distance, sans autre ordre de priorité (n'utilisant pas les bonus et point).

3 Interopérabilité

Nous avons testé notre programme avec les players des groupes 37, 47, 55 et 123. La liste des players testé est la suivante :

- `Ghost037Angry.oz`
- `Pacman037Hungry.oz`
- `Ghost047basic.oz`
- `Pacman047basic.oz`
- `Ghost055other.oz`
- `Pacman055superSmart.oz`
- `Ghost123name.oz`
- `Pacman123name.oz`

Ghost037Angry et Pacman037Hungry Ces deux players fonctionnent sans problèmes en Turn By Turn; en Simultaneous, les procédures de leur Pacman et de leur Ghost permettant de gérer le Thinking renvoie une erreur de type `illegal argument`.

Pacman047basic et Pacman123name Ces players fonctionnent en Simultaneous et en Turn By Turn. Cependant, si on place plusieurs Pacman 123 en Input, ceux-ci renvoient une erreur (leur Player tente de réassigner une variable globale à chaque nouveau Pacman généré). Le Pacman 47 ne fonctionne pas lorsuqu'on on en met qu'un seul en Input: le jeu se freeze.

Les autres players fonctionnent sans que l'on ait repéré d'erreurs. Par ailleurs, le groupe 20, en testant notre `Pacman091smart`, nous a permis de nous rendre compte d'une mauvaise gestion des messages `getId(ID).TreatStream`.

4 GUI

L'interface graphique a été modifiée en s'inspirant de l'univers des premiers jeux *Mario* des années 1980.