Projet Java A3P

Guillaume Legrain
Florian Martin
Groupe 3J E3S

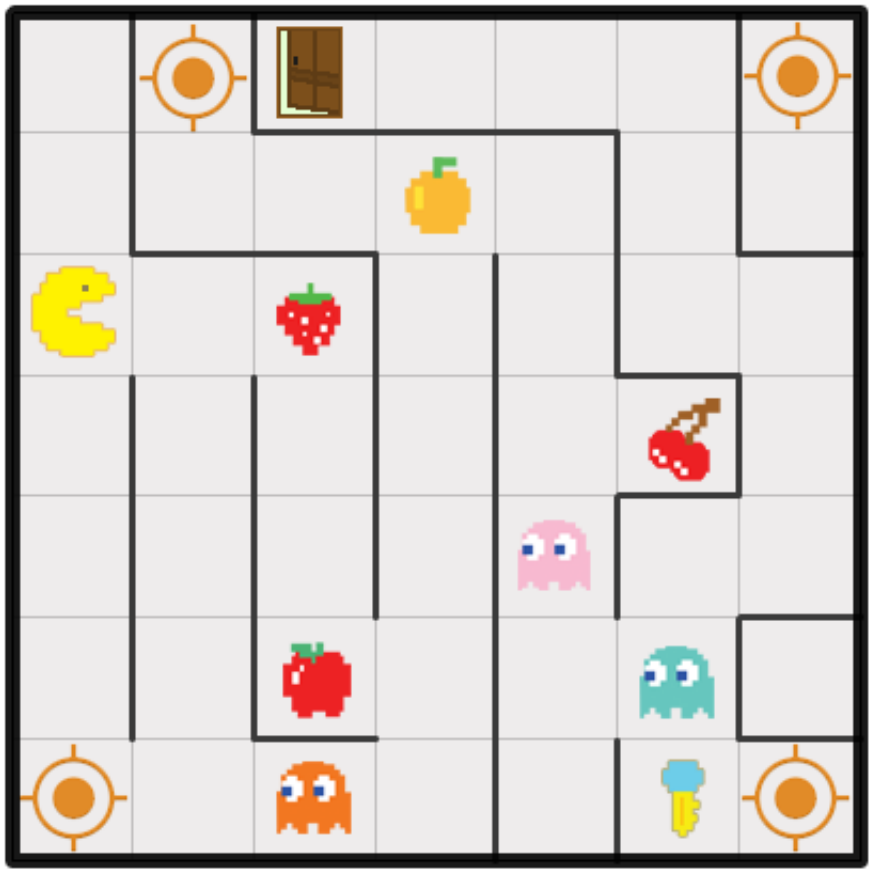
19 décembre 2013

Table des matières

Chapitre 1

Présentation du projet

Vous êtes Pac-Man, perdu dans un étrange labyrinthe peuplé de fantômes. Vous voulez vous échapper. Mais il faut trouver la clef de la sortie ainsi que des fruits pour affronter les fantômes qui en bloquent l'accès.



1.1 Buts

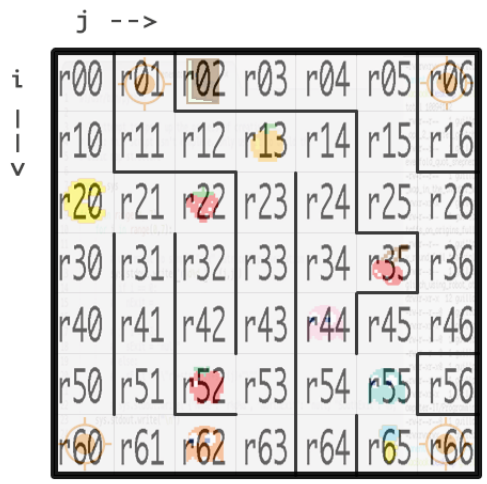
— But principal : trouver la clef pour pouvoir ouvrir la porte et sortir du labyrinthe.
— Les fantômes sont dangereux ! Il faut donc trouver et manger assez de fruits pour pouvoir les affronter.
— Pour aller plus vite, Pac-Man peut utiliser des téléporteurs. Mais attention, ils ne sont pas toujours très fiable et peuvent parfois conduire à des pièces sans issues.

1.2 Idées ...

Note : Tous ces points ne seront pas forcément implémentés vu que nous ne pouvons pas encore vraiment savoir ce qui est envisageable à notre niveau.

— Chaque case du labyrinthe sera en fait une "pièce" avec des sorties (Nord, Sud, Est, West)
— Le joueur pourra choisir parmi plusieurs labyrinthes (peut être même en créer lui-même, dans un fichier que le jeu va "parser" avant de démarrer).
— EN PLUS de la console (avec le "prompt"), le joueur pourra se déplacer avec les touches du clavier (Up, down, left, right).
— Le Pac-Man se déplace aussi, en temps réel sur l'écran.
— Dessiner le plan du labyrinthe en Java (ou autre si mieux ...) suivant sa description plutôt que d'afficher une ou des images statiques.
— Créer une carte évolutive qui se met a jour lors du passage dans une piece.
— Imposer un certain nombre de fruit pour vaincre un type de fantôme.
— Insérer une musique dans le jeu.
— Insérer des sons lors de la récupération d'objet (clé, fruit) ou l'appartion d'un fantôme

1.3 Organisation d'un plan



Chaque "Room" est représentée par rij indiquant sa position sur le plan. Chaque "Room" possède des attributs indiquant la position de la case suivante ou un mur (avec un `null`)

Chapitre 2

Avancement des exercices

Exercice 7.0 :

Création de la page web

Exercice 7.1 :

Découverte de zuul-bad

La classe Room

Création de la première salle avec pour attributs la description de la salle et quatre sorties (Nord, Sud, Est, West)

La classe Command

Création de la commande « go ». On doit déterminer dans un premier si un ou deux mots ont été tapés. Ensuite on doit vérifier si les commandes tapés sont valides d'où l'introduction d'une fonction booléenne « isUnknown »

La classe Game

- Création d'une méthode « createRooms » ayant pour but d'initialiser le jeux (les salles, les sorties, le lieu courant)
- La procédure « goRoom » doit s'assurer de la présence d'un second mot (direction) et doit vérifier l'égalité de ce second mot avec une direction avec la fonction « equals ». Si les conditions sont réunis « goRoom » change la current room
- On ajoute de nouvelles méthodes : 2 procédures renvoyant un message de bienvenue et un message d'aide. La première doit pouvoir faire appel à la current room et à ses différentes sorties. La seconde doit pouvoir renvoyer le nom de chaque commande disponible. On crée donc des procédure auxquelles on fera appel pour renvoyer ces informations et cela indépendamment du message écrit.
- On crée une procédure « processCommand » renvoyant un booléen, capable d'appeler la bonne méthode en fonction de la commande passée en paramètre

Les classes CommandWords et Parser

La classe CommandWords contient les mots de commande acceptés par le jeu
La classe Parser lie les commandes tapée au clavier, vérifie si la commande est valide et construit l'objet « Command » correspondant auquel on fera appel
V Jeu fonctionnel
On ajoute dans la classe Game un attribut aParser et une procédure « play» qui doit lire répétitivement les commandes tapé au clavier et les exécuter jusqu'à ce que l'on tape « quit ». On introduit une boucle

while qui s'exécutera jusqu'à ce qu'une variable booléenne soit égal à vrai. Ce changement d'état aura lieu lorsque l'on tapera la commande « quit ».

Exercice 7.1.1 : Thème

Exercice 7.2.1 : La classe Scanner

On crée un objet Scanner possédant le clavier en tant que paramètre. Les lignes de caractères tapés au clavier seront retranscrites dans une String.

Exercice 7.4 : Room

On intègre dans le jeu les différentes salles qui compose notre scénario

Exercice 7.5 : printLocationInfo

On ajoute une méthode affichant les informations de la current Room (nom pour l'instant)

Exercice 7.6 : getExit

On ajoute une méthode renvoyant les sorties de la current Room

Exercice 7.7 : getExitString

On modifie la méthode « printLocationInfo » pour qu'elle affiche aussi les sorties de la current Room. On fait appel pour cela à la méthode « getExit »

Exercice 7.8 : HashMap, setExit

On modifie la classe Room pour qu'elle possède un attribut aExit sous la forme d'une HashMap regroupant ainsi toutes les sorties.

Exercice 7.9 : keySet

Exercice 7.10 : getExitString CCM ?

ToDo

Exercice 7.11 : getLongDescription

ToDo

Exercice 7.14 : look

Exercise 7.40 :

We have to add LOOK("look") to CommandWord and implement a action in Game.processCommand

Exercise 7.41 :

If we change the word associated with the help command in CommandWord, this change automatically reflected in the welcome text when you start the game.

Exercise 7.42 :

TODO : implement mvc
+ Added KeyListener to change room.
You can now go to another room using the keyboard's arrow keys.

Exercise 7.43 : (Trap door)

trap door player in room r06
TODO : empty previous room stack if trapdoor (the back command shouln't work)
TODO : show message

Exercise 7.44 : (beamer)

seems ok

Exercise 7.45 : (locked door) (opt.)

TODO : the player needs to find a key to open a locked door
NOTE :use heritage

Exercise 7.45.1 : (update test files)

TODO

Exercise 7.45.2 : (update javadoc)

TODO

Exercise 7.46 : (TransporterRoom)

+ Created a new TransporterRoom subclass from Room overriding getExit
Change r60, .. instantiation to TransporterRoom instead of Room
+ Added a getRooms method to the GameModel to get a Room HasMap
Converter the HashMap into a ArryList to select a value from a Random integer instead of a string description.

Exercise 7.46.1 : (alea)

TODO

Exercise 7.46.2 :

Beamer was already a item subclass
+ Created TrapRoom subclass (TODO : clear history when entering the room)

Exercise 7.53 : (main)

+ Added public static void main(String[] args) to instantiate a new game

Exercise 7.54 : (without BlueJ)

Launch the game using j̈ava Gameïn the game's directory