|  |
| --- |
| BERCY - JOULIN |
| Jeu de la grenouille |
| TP de programmation |
|  |
| **INFO1** |
| **2012** |

|  |
| --- |
|  |

Table des matières

[2 Présentation 4](#_Toc355371027)

[2.1 Présentation du jeu 4](#_Toc355371028)

[3 Composantes du jeu 4](#_Toc355371029)

[3.1 Personnages 4](#_Toc355371030)

[3.2 Zone de jeu 5](#_Toc355371031)

[3.3 Joueurs 6](#_Toc355371032)

[4 Architecture du jeu 7](#_Toc355371033)

[5 Intelligence artificielle 7](#_Toc355371034)

[5.1 Prévision des déplacements 7](#_Toc355371035)

[5.1.1 Recherche des meilleurs mouvements (pendant la partie) 7](#_Toc355371036)

[5.1.2 Recherche des meilleures mouvements (avant la partie) 8](#_Toc355371037)

[5.1.3 Chances de gagner 8](#_Toc355371038)

[5.2 Solution implémentée 9](#_Toc355371039)

[5.3 Stratégies 9](#_Toc355371040)

# Présentation

## Présentation du jeu

Le jeu oppose deux joueurs et consiste à déplacer des grouilles et des crapauds sur un plateau quadrillé de N lignes et M colonnes. En début de partie, les grenouilles sont attribuées à l’un des joueurs, les crapauds à l’autre. Le détenteur des grenouilles commence, puis les joueurs jouent à tour de rôle. Le premier qui ne peut plus avancer ses batraciens a perdu.  
Les grenouilles avancent horizontalement de la gauche vers la droite, et les crapauds de la droite vers la gauche.  
Si la case adjacente dans le sens de déplacement est libre, la grenouille (ou le crapaud) déplacé(e) vient occuper cette case. Si, par contre, elle est occupée par un crapaud (ou une grenouille) et que la case suivante est libre, le batracien vient occuper cette case libre. Dans tous les autres cas, le batracien ne peut être déplacé.

# Composantes du jeu

## Personnages

Deux solutions permettent de représenter les personnages : la première consiste à créer une classe pour chacun d’entre eux, la deuxième consiste à utiliser une seule classe comportant une propriété qui indique le type de personnage.

|  |  |
| --- | --- |
| Image 3.1 - Schéma de la première solution  Personnage  Grenouille  Crapaud | Dans le schéma ci-contre, Personnage est une classe abstraite, héritée par les classes Grenouille et Crapaud.  Avantages :   * L’architecture reflète la réalité. La classe Personnage a été nommée ainsi pour son application dans le contexte du jeu, mais aurait pu se nommer « Anoures » pour faire référence à la famille des grenouilles et des crapauds dans la nature.   Inconvénients :   * La mise en pratique d’une telle architecture de code conduit à écrire la totalité du code dans la classe abstraite, ou à écrire du code similaire dans les deux classes dérivées. * L’utilisation de cette architecture complique les interactions avec les classes. Par exemple, pour connaître le type d’un personnage donné dans la zone de jeu, il est nécessaire d’avoir recours à instanceOf dans une partie du code où il pourrait être possible de travailler à un plus haut niveau (et par conséquent de s’abstraire de ces constructions de bas niveau). Une alternative serait d’implémenter dans les classes la possibilité de les interroger sur leur type, au quel cas l’intérêt de les séparer se verrait réduit. |

|  |  |
| --- | --- |
| Image 3.2 - Schéma de la seconde solution  Personnage  type  *C*  *G* | Dans cet autre schéma, la classe Personnage est à la fois utilisée pour représenter les grenouilles et les crapauds. Le type de personnage peut par exemple être définit lors de la construction d’un objet, et récupéré via un accesseur.  Avantages :   * Architecture plus simple * Transparence complète pour le code du jeu au plus haut niveau * Création plus simple des personnages   Inconvénients :   * Il est plus difficile de rajouter de nouveaux personnages. Dans le cas du jeu de la grenouille, il est certain que le nombre de personnages sera toujours de deux, le problème ne se posera donc jamais. |

## Zone de jeu

D’un point de vue abstrait, la zone de jeu est une rivière comportant plusieurs rangées de feuilles sur lesquelles les grenouilles peuvent se déplacer.

Concrètement, une rangée de feuilles (ou « couloir ») est un graphe représentant les déplacements possibles des personnages.

Image 3.3 - Représentation d'un couloir par un graphe

La zone de jeu (rivière) est donc un ensemble de graphes tels que celui représenté dans l’Image 2.3. Il s’agit concrètement d’un tableau.

Le comportement attendu du graphe est celui d’une liste doublement lié. Une telle structure permet d’effectuer très simplement des déplacements et des recherches sur les feuilles alentours, comme dans le Code 2.1 - Exemple de recherche d'un déplacement possible.

1. // Examiner la feuille à droite d’une grenouille
2. Personnage voisin = grenouille.feuille().suivante().personnage() ;
4. if (voisin == null) {
5. // La feuille suivante est vide, on peut s’y déplacer
6. } elseif (voisin.type() == Personnage.TYPE\_GRENOUILLE) {
7. // Le personnage voisin est une grenouille – Déplacement impossible
8. } elseif (voisin.type() == Personnage.TYPE\_CRAPAUD) {
9. // Le personnage voisin est un crapaud, on peut examiner la feuille suivante
10. if (voisin.feuille().suivante().personnage() == null) {
11. // On peut sauter par-dessus le crapaud
12. } else {
13. // Déplacement impossible
14. }
15. }

Code 2.1 - Exemple de recherche d'un déplacement possible pour une grenouille

## Joueurs

Pour chacun des deux types de personnage (grenouilles et crapauds), deux types de joueurs sont possibles (humain ou virtuel). Afin de garantir le maximum de flexibilité, les deux types de joueurs doivent être interchangeables de façon totalement transparente au reste du jeu. Le polymorphisme permet d’arriver à cet idéal.

Image . - Gestion des joueurs

Humain

IA

Joueur grenouilles

Humain

IA

Joueur crapauds

Ce que l’Image 2.4 vise à montrer, est que seuls les systèmes « joueur grenouilles » et « joueur crapauds » sont exposés au reste du jeu. Aucun des autres systèmes n’a besoin de s’avoir si un personnage donné est dirigé par un humain ou par une intelligence artificielle.

# Intelligence artificielle

## Prévision des déplacements

La prévision de l’ensemble des déroulements possibles d’une partie permet d’implémenter un joueur « parfait », c'est-à-dire capable de gagner chaque fois que cela est possible. Au cours de la partie, l’ordinateur peut maintenir un graphe comportant les cheminements possibles du jeu et leurs résultats.

Image 5.1 - Graphe généré pour une partie dans une zone 1x6

|  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- |
| G | G |  |  | C | C |

|  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- |
| G |  | G |  | C | C |

|  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- |
| G |  | G | C |  | C |

|  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- |
|  | G | G | C |  | C |

|  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- |
|  | G | G | C | C |  |

|  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- |
| G |  |  | C | G | C |

|  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- |
| G |  | C |  | G | C |

|  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- |
|  | G | C |  | G | C |

|  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- |
| C | G |  |  | G | C |

|  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- |
|  | G | C | C | G |  |

**…**

**…**

*Victoire des crapauds*

*Situation de départ*

Une fois un tel graphe généré, le programme « sait » quels mouvements effectuer pour parvenir à la victoire. Le graphe devra être maintenu au cours de la partie, en supprimant les branches qui ne sont plus pertinentes, et en ajoutant de nouvelles branches si la totalité du graphe n’a pas pu être généré en début de partie.

### Recherche des meilleurs mouvements (pendant la partie)

La recherche du meilleur déplacement à effectuer dans une situation donnée peut s’effectuer en parcourant le graphe jusqu’à chacune des extrémités.

L’algorithme suivant permet une telle recherche :

Image . – Algorithme Depth-first search

Marquer le sommet actuel comme « visité »

Sélectionner un arc menant à un sommet non visité

Un tel arc   
existe-t-il ?

Suivre cet arc jusqu’au prochain sommet

Commencer au sommet correspondant à la situation de départ   
du jeu

Oui

Le sommet  
correspond-t-il à une  
victoire du joueur ?

Fin – Un chemin a été trouvé

Oui

Non

Le sommet  
correspond-t-il à la  
situation de  
départ ?

Non

Oui

Fin – Victoire impossible

Revenir au sommet précédent

Une pile est utilisée pour trouver le chemin jusqu’au résultat voulu.

### Recherche des meilleurs mouvements (avant la partie)

Il est plus efficace de stocker les chemins gagnants dans une liste au fur et à mesure que le graphe est généré, afin qu’ils puissent être retrouvés avec une complexité constante.

### Chances de gagner

Afin de permettre au programme de faire de meilleurs choix, on peut assigner un poids aux arcs du graphe, correspondant au pourcentage de chance de remporter la partie en effectuant un mouvement donné.

Cette technique permet aussi au programme de prendre des décisions si la totalité du graphe n’a pas été générée : on peut quantifier « l’intérêt » de chaque situation. Par exemple, une situation où la plupart des personnages du joueur sont au bout du couloir est préférable à une situation dans laquelle certains d’entre eux sont bloqués par l’adversaire.

## Solution implémentée

(A faire : expliquer la solution implémentée dans le jeu)

## Stratégies

Il est possible d’implémenter plusieurs intelligences artificielles différentes, suivant chacune une « stratégie » afin de mener des expériences sur l’efficacité d’une stratégie donnée sur les autres.

(A faire : décrire les stratégies expérimentées)

# Architecture du jeu

# Utilisation

(A faire : guide d’utilisation du jeu)

(Suite : version 2.0 avec interface graphique ?)