Gaps - Projet P3

Owen Gombas

27.01.2023

Table des matieres

1 Introduction **2**
- 1.1 Description du projet . . . 2
- 1.2 Objectifs . . . 2

2 Environment **3**
- 2.1 Swift . . . 3
- 2.2 SwiftUI . . . 3
- 2.3 Playground . . . 3

3 Code **4**
- 3.1 UI . . . 4
- 3.2 GameState.swift . . . 4
- 3.3 Heuristic . . . 5
- 3.4 Statistics . . . 6
- 3.5 Heuristics.playground . . . 6

4 Heuristics **7**
- 4.1 Méthodologie . . . 7

Chapitre 1
Introduction

1.1 Description du projet

"Gaps Solitaire" est un jeu de cartes, également connue sous le nom de "Montana Solitaire" et des "Quatre jeudis" à cause de sa difficulté. L'objectif du jeu est d'organiser les cartes séquentiellement et par couleur, de l'as à la dame, en déplaçant les cartes dans quatre espace initialement choisis au hasard. Une carte peut être déplacée vers un espace vide si et seulement si elle est de la même couleur et directement supérieure à la carte située à gauche du dit espace.

Il existe plusieurs sites pour jouer en ligne, comme par exemple `https://www.jeuxsolitaire.fr/jeu/Gaps+Solitaire`.

"Gaps Solitaire" est un jeu très difficile à terminer, et contrairement à la majorité des jeux de patience requiert beaucoup de stratégie.

1.2 Objectifs

L'objectif du projet P3 est de développer un moteur permettant de placer le plus de cartes possible à partir de toute configuration initiale des cartes. Ce projet vous permettra de vous familiariser et d'approfondir les concepts suivants:

- Structures de données pour modéliser le jeu et les coups possibles
- Algorithmes élémentaires de parcours de graphes (largeur, profondeur)
- Algorithmes d'intelligence artificielle (A*, ...).

Chapitre 2
Environment

2.1 Swift

Le choix s'est porté sur le langage d'Apple, Swift, pour la réalisation de ce projet. C'est un langage de programmation moderne, puissant et facile à apprendre. Il est conçu pour être sûr, performant et expressif. Swift est un langage de programmation multi-paradigme, qui supporte le paradigme impératif, le paradigme fonctionnel et le paradigme orienté objet.

Ce langage s'est donc imposé comme le choix idéal pour ce projet, car il permet de développer des applications performante et facilement portable sur les différents systèmes d'exploitation d'Apple.

2.2 SwiftUI

SwiftUI est un framework de développement d'interface graphique pour les applications iOS, macOS, watchOS et tvOS. Il permet de créer des interfaces graphiques de manière déclarative, et de les mettre à jour automatiquement lorsqu'un changement est détecté. Il est possible de créer des interfaces graphiques en utilisant des composants prédéfinis, ou de créer ses propres composants.

2.3 Playground

Apple fournit un outil de développement nommé Playground, qui permet de développer et d'exécuter du code Swift. Il est possible de créer des pages de code, qui peuvent être exécutées séparément. De la manière qu'un notebook Jupyter permet de créer des cellules de code, un Playground permet de créer des pages de code. Ces pages peuvent être exécutées séparément, et les résultats sont affichés dans la console. Cela permet de tester des idées, de débugger du code, et de créer des exemples.

Cette fonctionnalité a été très utile afin d'effectuer les calculs nécessaires afin de trouver les meilleures heuristiques pour le projet.

Chapitre 3
Code

3.1 UI

Le code principal de l'interface graphique se trouve dans le fichier `ContentView.swift`. Elle permet d'afficher une configuration de jeu et d'intéragir avec. On peut déplacer les cartes en cliquant dessus, et en les déplacer vers un espace vide. Lorsqu'on execute un des algorithmes, on peut consulter les logs de l'exécution dans un espace dédié, et voir le déroulement de l'algorithme dans l'interface graphique une fois que celui-ci est terminé.

L'interface est composée des composants "custom" suivants:

- `StateUI`
  Affiche le plateau de jeu et permet d'interagir avec celui-ci.
- `ChartUI` et `Measure`
  Affiche un graphique représentant divers statistiques.

3.2 GameState.swift

Cette classe représente l'état du jeu. Elle nous permet de stocker les cartes. Elle permet aussi de déplacer les cartes, de vérifier si le jeu est terminé, de générer les coups possibles, et d'effectuer toutes les opérations permettant l'intéraction avec le jeu.

La structures de données utilisées pour représenter le jeu est une matrice de `Card` (voir `Card.swift`). Les cartes sont stockées dans un tableau à deux dimensions, et sont indexées par leur position dans le jeu. Les cartes sont stockées dans l'ordre dans lequel elles sont affichées dans l'interface graphique.

Les enfants (`state.getMoves()`) du jeu sont les configurations de jeu obtenues en déplaçant une carte. Pour obtenir les enfants, on parcourt la matrice de cartes, et on déplace les cartes en respectant les règles du jeu. On obtient ainsi une liste de configurations de jeu, qui sont les enfants de la configuration actuelle. On peut alors utiliser les algorithme de parcours de graphe ou de recherche par heuristique pour trouver la solution.

3.3 Heuristic

Cette classe représente une heuristique. Elle permet de calculer la valeur d'une configuration de jeu. Elle est utilisée par les algorithmes de recherche par heuristique pour trouver la solution.

Une heuristique est une fonction qui prend en paramètre une configuration de jeu (`GameState`), et qui retourne un nombre (`Int`). Plus ce nombre est petit, plus la configuration est proche de la solution. Plus ce nombre est grand, plus la configuration est éloignée de la solution.

La classe permet de composer des heuristiques en les combinant de la manière suivante avec des poids:

```
Heuristic.combine(
    heuristics: [
        Heuristic.countMisplacedCards,
        Heuristic.wrongColumnPlacement
    ],
    weights: [
        1,
        2
    ]
)
// or
Heuristic.combine([
    (1, Heuristic.countMisplacedCards),
    (2, Heuristic.wrongColumnPlacement)
])
```

Heuristic.combine, prend en paramètre une liste de heuristiques (fonctions prenant une configuration de jeu en paramètre et retournant un nombre) et une liste de poids. Elle retourne une heuristique (une fonction prenant une configuration de jeu en paramètre et retournant un nombre) qui est la somme des heuristiques pondérées.

3.4 Statistics

Cette classe permet de comparer les algorithmes de recherche par heuristique et les algorithmes de parcours de graphe. Elle permet de:

- `generateGames`
  Générer X configurations de jeu aléatoires de taille MxN.
- `getArrangements`
  Générer tous les arrangements N de nombres dans un intervalle donné, ce qui permet plus tard de tester les poids des heuristiques afin de trouver les plus optimaux.
- `findBestWeights`
  Met en competition les differentes heuristique générées avec les poids passées en paramètre en utilisant A*, ce qui nous permet de determiner les poids générant l'heuristique la plus performante.
- `getPeformances`
  On donne N jeux et différents algorithme et nous renvoit un rapport sur l'execution des algorithme sur ces jeux afin de connaître le plus performant.
- `executeAlgorithms`
  Execute plusieurs algorithme sur un jeu donné et revoit un rapport de l'execution.

3.5 Heuristics.playground

Chapitre 4
Heuristics

4.1 Méthodologie