ENSSAT

LANNION

# Compte rendu de projet : Algorithme 2

Minesweeper

**Julien BOURDET**

**Florian ABADIE**

ENSSAT

1ère année - Informatique

Lannion, January 2025

# Table des matières

**1 Introduction** **2**

**2 Stratégie** **3**

2.1 Preliminaires . . . . . 3

2.2 Algorithme naïf . . . . . 3

2.2.1 Dénombrement . . . . . 3

2.2.2 Pseudo-code . . . . . 4

2.2.3 Complexité . . . . . 5

2.3 Algorithme optimisé . . . . . 5

2.3.1 Optimizations . . . . . 6

# 1

# Introduction

Dans le cadre du module Algorithme 1, nous avons pu appliquer les concepts étudiés en cours en réalisant un projet. Ce projet consiste à réaliser une intelligence artificielle (IA) capable de jouer au jeu Lode Runner. C'est un jeu d'arcade qui se déroule sur une carte en deux dimensions où le joueur doit récupérer des bonus tout en évitant des ennemis.

L'objectif du projet est de développer une IA capable de pouvoir terminer n'importe quel niveau du jeu. Cependant, des limitations ont été imposées : l'IA ne connaît pas les actions futures des ennemis et ne peut pas mémoriser des informations d'un tour à l'autre. Ce cadre strict force l'IA à développer des stratégies approfondies permettant de se projeter dans le futur pour ne pas réaliser des actions inutiles ou contre productives

Au cours du développement, j'ai rencontré plusieurs défis. L'absence de mémorisation empêche d'utiliser une stratégie trop coûteuse en temps de calcul ou de planifier une stratégie sur plusieurs tours, tandis que le fait de ne pas connaître les prochains coups des ennemis empêche d'explorer un "arbre des positions". Par ailleurs, l'implémentation en C, avec ses contraintes de gestion manuelle de la mémoire, a ajouté une dimension technique non négligeable.

Malgré ces contraintes, j'ai pu élaborer une IA fonctionnelle et performante (au moins sur les niveaux disponibles).

Ce compte rendu présente ma démarche de développement, la stratégie utilisée et les résultats obtenus.

# 2

# Stratégie

Même si le démineur est un jeu simple, une IA parfaite n'aurait pas 100% de victoire. En effet, c'est un jeu de probabilités, et il existe donc des situations dans lesquelles la possibilité de trouver un bombe dans chaque case non-révélée est équiprobable. Dans ce cas, l'IA doit faire un choix aléatoire.
Notre stratégie sera donc de calculer la probabilité de chaque case non-révélée de contenir une bombe, et de choisir une des cases avec la plus faible probabilité.

## 2.1 Preliminaires

Dans toute la suite de ce chapitre, on considère une grille de démineur de taille $n \times m$ avec $b$ bombes, et on note $G$ la grille, $G_{i,j}$ la case à la ligne $i$ et la colonne $j$.
$G_{i,j} = -1$ si la case n'est pas révélée, $G_{i,j} = k$ si la case est révélée et contient $k$ bombes autour d'elle.

## 2.2 Algorithme naïf

Calculer la probabilité de chaque case non-révélée de contenir une bombe est une tâche complexe : une information à un bout de la grille peut influencer une case à l'autre bout.
Intuitivement, on commence par essayer de faire du dénombrement : on calcule toutes les combinaisons possibles de bombes, et ne garde que les combinaisons valides, et on compte le nombre de fois où chaque case est une bombe.

### 2.2.1 Dénombrement

On construit un algorithme qui, pour une grille $G$ donnée calcule toutes les combinaisons possibles de bombes, et ne garde que les combinaisons valides.
Il y a deux difficultés majeures à résoudre :

- Comment valider une combinaison ?
- Comment éviter de tester des combinaisons qui sont évidemment invalides ?

Pour répondre à ces problèmes, on introduit une idée simple, mais qui nous sera utile dans la suite : on va grouper des cases ensemble.
On définit un groupe comme un ensemble de cases non-révélées.
Dans cet algorithme, pour chaque case révélée, on crée un groupe avec les cases adjacentes.
On peut alors, pour chaque combinaison de 2 groupes, trouver l'intersection de ces derniers, et poser une condition sur les cases de cette intersection.

### 2.2.2 Pseudo-code

```
1   Fonction Probabilite
2   Parametres :
3       n en entier, m en entier
4       G en matrice d'entiers de taille n x m
5   Declarations :
6       groupes en liste de listes de couples d'entiers
7       intersections en liste de listes de couples d'entiers
8       P en matrice d'entiers
9       combinaisons en liste de listes de couples d'entiers
10      combinaisons_valides en entier
11  Sortie :
12      P en matrice de reels de taille n x m
13  Debut
14      P = matrice de 0 de taille n x m
15
16      groupes = liste des groupes des cases révélées
17
18      intersections = liste des intersections de chaque combinaison de 2 groupes
19
20      combinaisons = []
21      Pour k allant de 1 a |groupes| faire
22          Ajouter a combinaisons une liste de toutes les combinaisons possibles
            ↪ de G[i][j] bombes dans les cases de groupes[k]
23      FinPour
24
25      combinaisons_valides = 0
26      Pour chaque element du produit cartesien de combinaisons faire
27          Pour chaque intersection dans intersections faire
28              Si l'intersection contient plus de bombes que le nombre de bombes
                ↪ autour de chaque case dans l'intersection alors
29                  Continuer
30              FinSi
31          FinPour
32
33          Pour chaque case dans l'element faire
34              P[case] += 1
35          FinPour
36          combinaisons_valides += 1
37      FinPour
```

```
38
39              Pour i allant de 1 a n faire
40                  Pour j allant de 1 a m faire
41                      Si G[i,j] = -1 alors
42                          P[i,j] /= combinaisons_valides
43                      FinSi
44                  FinPour
45              FinPour
46
47              Retourner P
48          Fin
```

**Listing 2.1:** *Pseudo-code de l'algorithme naif de calcul de probabilité*

### 2.2.3 Complexité

La complexité de cet algorithme est très élevée, car il faut calculer le produit cartésien de toutes les combinaisons possibles de bombes : la complexité est exponentielle en le nombre de cases non-révélées. Cet algorithme est donc inutilisable pour des grilles de taille moyenne.
Si on veut calculer les probabilités de chaque case, il va falloir radicalement optimiser cette stratégie.

## 2.3 Algorithme optimisé

Pour optimiser cet algorithme, on va représenter notre problème sous une forme différente.
On commence par définir un graphe $G'$, où chaque sommet est une case, et où il y a une arête entre deux cases si elles sont adjacentes. On construit alors la matrice A de taille $n * m \times n * m$ telle que $A_{(i,j),(i',j')} = 1$ si $(i, j)$ et $(i', j')$ sont adjacents, et $0$ sinon. C'est la matrice d'adjacence de $G'$.
On remarque alors que l'ont peut très facilement retrouver la matrice $G$ complète (c'est à dire avec toutes les cases révélées) à partir de $A$ et de la position des bombes.

Notons $B$ la colonne de taille $n * m$ telle que $B_i = 1$ si la case $i$ contient une bombe, et $0$ sinon. De même, soit $v$ la colonne de taille $n * m$ telle que $v_i = G_{i,j}$ si la case $i$ est révélée, et $-1$ sinon.

On a alors la relation suivante :

$$A \cdot B = v$$

Cette relation est un système linéaire, que l'on peut résoudre pour trouver toutes les colonnes $B$ qui satisfassent $v$ (c'est à dire toutes les positions possible de bombes qui respectent les cases révélées). Nous pouvons alors calculer la probabilité de chaque

case non-révélée de contenir une bombe.

### 2.3.1 Optimizations

Représenter le problème sous forme de système linéaire ne change pas la complexité de la résolution du problème, résoudre le système entier revient à la même complexité que l'algorithme naïf. Mais cette représentation nous permet de faire des optimisations qui vont nous permettre de réduire la complexité de l'algorithme.

Prenons un exemple simple :

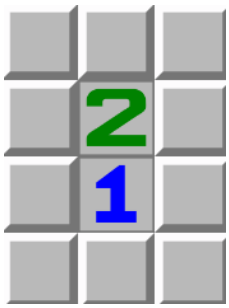


**Figure 2.1:** *Exemple de grille.*

On obtient les matrices suivantes : (A est symétrique, on ne montre que la moitié de la matrice)

$$A = \begin{pmatrix} 0 & 1 & 0 & 1 & 1 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\ & 0 & 1 & 1 & 1 & 1 & 0 & 0 & 0 & 0 & 0 & 0 \\ & & 0 & 0 & 1 & 1 & 0 & 0 & 0 & 0 & 0 & 0 \\ & & & 0 & 1 & 0 & 1 & 1 & 0 & 0 & 0 & 0 \\ & & & & 0 & 1 & 1 & 1 & 1 & 0 & 0 & 0 \\ & & & & & 0 & 0 & 1 & 1 & 0 & 0 & 0 \\ & & & & & & 0 & 1 & 0 & 1 & 1 & 0 \\ & & & & & & & 0 & 1 & 1 & 1 & 1 \\ & & & & & & & & 0 & 0 & 1 & 1 \\ & & & & & & & & & 0 & 1 & 0 \\ & & & & & & & & & & 0 & 1 \\ & & & & & & & & & & & 0 \end{pmatrix} \qquad v = \begin{bmatrix} -1 \\ -1 \\ -1 \\ -1 \\ 2 \\ -1 \\ -1 \\ 1 \\ -1 \\ -1 \\ -1 \\ -1 \end{bmatrix}$$

On commence par ne garder que les lignes de $A$ et de $v$ qui correspondent à des cases révélées. On peut enlever ces lignes car on ne connait pas la valeur de ces cases dans $v$, les utiliser pour résoudre le système ne nous apporterait aucune information.

On enleve aussi les colonnes de $A$ qui correspondent à des cases révélées, car il ne peut y avoir de bombes dans ces cases.

On obtient alors :

$$A = \begin{pmatrix} 1 & 1 & 1 & 1 & 1 & 1 & 1 & 0 & 0 & 0 \\ 0 & 0 & 0 & 1 & 1 & 1 & 1 & 1 & 1 & 1 \end{pmatrix} \qquad v = \begin{bmatrix} 2 \\ 1 \end{bmatrix}$$

On remarque alors que plusieurs colonnes de $A$ sont identiques, on va donc les regrouper. Ce regroupement n'est pas intuitif, on est en fait en train de grouper les cases qui sont influencées par les mêmes cases révélées. Ce dernier est possible car on sait que chaque case d'un groupe à la même probabilité de contenir une bombe.
Dans notre exemple, on obtient ces groupes :

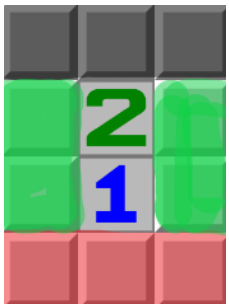


**Figure 2.2:** *Grille avec les groupes en couleur.*

On groupe alors ces colonnes :

$$A = \begin{pmatrix} 1 & 1 & 0 \\ 0 & 1 & 1 \end{pmatrix} \qquad v = \begin{bmatrix} 2 \\ 1 \end{bmatrix}$$

Trouver un solution à ce système revient à distribuer des bombes dans les groupes. On connait le nombre de cases dans chaque groupe, on peut alors facilement calculer la probabilité de chaque case de contenir une bombe.

On utilise l'algorithme de Gauss-Jordan pour réduire la matrice $A$ en forme échelonnée réduite :

$$A = \begin{pmatrix} 1 & 0 & -1 \\ 0 & 1 & 1 \end{pmatrix} \qquad v = \begin{bmatrix} 1 \\ 1 \end{bmatrix}$$

Dans un cas classique, ce système aurait une infinité de solutions, mais dans notre cas, on peut mettre des bornes sur les variables (le nombres de bombes par groupe). Initialement, ces bornes sont $[0, |groupe|]$, mais on peut les réduire grâce au système $A|v$.
On cherche alors les variables libres, c'est à dire les colonnes de $A$ qui ne sont pas des pivots. Les solutions sont alors les combinaisons de ces variables qui respectent les bornes.

Dans notre exemple, on a une seule variable libre, la dernière colonne de $A$. Les solutions sont alors :

$$\begin{bmatrix} 2 \\ 0 \\ 1 \end{bmatrix} \qquad \begin{bmatrix} 1 \\ 1 \\ 0 \end{bmatrix}$$

Retrouver la probabilité de chaque case revient à trouver la probabilité de chaque solution. On obtient alors :

$$\mathbb{P}(\text{Solutions}_k) = \prod_{i=1}^{|\text{Groupes}|} \binom{|\text{Groupe}_i|}{\text{Solution}_i}$$

$$\mathbb{P}(\text{Groupes}_k) = \sum_{i=1}^{|\text{Solutions}|} \frac{\mathbb{P}(\text{Solutions}_i) \times \text{Solution}_k}{|\text{Groupes}_k|}$$

Finallement, toute les cases d'un groupe ont la probabilité de ce groupe de contenir une bombe.