

TIPE

Résolution algorithmique et génération de problèmes

Go-getter

Introduction

Le jeu en lui-même

TIPE

Introduction

Modélisation

Bruteforce

Optimisation

Génération de problèmes



Introduction

Le jeu en lui-même

TIPE

Introduction

Modélisation

Bruteforce

Optimisation

Génération de problèmes



Introduction

Le jeu en lui-même

TIPE

Introduction

Modélisation

Bruteforce

Optimisation

Génération de problèmes



Introduction

Le jeu en lui-même

TIPE

Introduction

Modélisation

Bruteforce

Optimisation

Génération de
problèmes

24 problèmes différents :



Introduction

Le jeu en lui-même

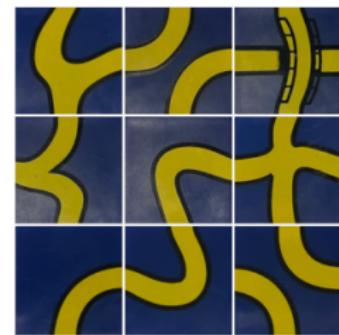
TIPE

Introduction

Modélisation

Bruteforce

Optimisation

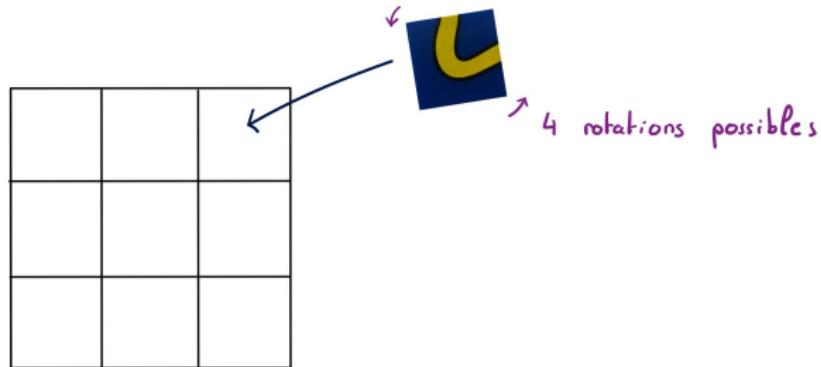
Génération de
problèmes

Introduction

Brève étude du problème

TIPE

Introduction
Modélisation
Bruteforce
Optimisation
Génération de problèmes



Soit n le nombre de pièces :

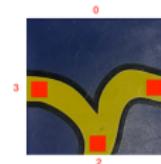
$$\underbrace{n!}_{n \text{ emplacements}} \times \underbrace{4^n}_{rotations} \text{ plateaux possibles}$$

Modélisation

Une pièce quelconque, du jeu au schéma (1)

TIEPE

Prenons une pièce du jeu, qu'on nommera P_1 :



Modélisation

Une pièce quelconque, du jeu au schéma (2)

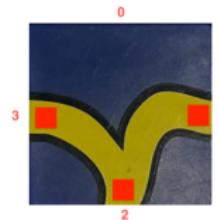
TIPE

Introduction

Modélisation

Bruteforce

Optimisation

Génération de
problèmes

On convertit alors cette pièce par une liste d'adjacence :

- 0
- 1 [2; 3]
- 2 [1; 3]
- 3 [1; 2]

Modélisation

Une pièce quelconque, du schéma au code

TIEPE

Introduction

Modélisation

Bruteforce

Optimisation

Génération de
problèmes

0

1 [2; 3]

2 [0; 3]

3 [0; 2]

et on convertit cette liste d'adjacence en C :

```
int case_quelconque[4][4] = {  
    {},  
    {2, 3},  
    {1, 3},  
    {1, 2}  
};
```

Modélisation

Moteur graphique

TIPE

Introduction
Modélisation
Bruteforce
Optimisation
Génération de problèmes



Modélisation

Vocabulaire

TIPE

Introduction

Modélisation

Bruteforce

Optimisation

Génération de
problèmes

On définit le vocabulaire suivant :

- *Deck* : ensemble de cartes
- *Slot* : case (remplie/non-remplie) du plateau disposant des côtés numérotés (de la même manière qu'une pièce) de 0 à 3.

Modélisation

Implémentation d'un deck

TIEPE

Introduction

Modélisation

Bruteforce

Optimisation

Génération de
problèmes

Le jeu nous met à disposition 9 cartes, on va les placer dans un tableau (que l'on nommera l'*index*), un peu à la manière de mettre à plat sur une table toutes les cartes :



Modélisation

Définition des types

TIPE

Introduction

Modélisation

Bruteforce

Optimisation

Génération de
problèmes

On définit alors les types suivants :

```
typedef int** cell;

struct plateau_s{
    int len;
    cell* components;
    bool completed;
};
typedef struct plateau_s plateau;
```

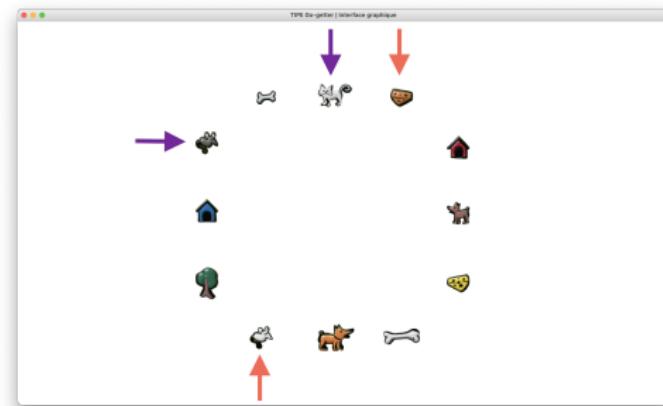
Bruteforce

Avec des schémas (0)

TIEPE

Posons en guise d'exemple deux conditions C_1 et C_2 avec :

- C_1 :  \rightarrow 
- C_2 :  \rightarrow 



Bruteforce

Avec des schémas (1)

TIE

Introduction

Modélisation

Bruteforce

Optimisation

Génération de
problèmes

Choisissons P_1 comme première pièce posée, sans rotation :



Bruteforce

Avec des schémas (2)

TIE

Introduction

Modélisation

Bruteforce

Optimisation

Génération de
problèmes

Choisissons ensuite P_2 comme deuxième pièce posée, sans rotation :

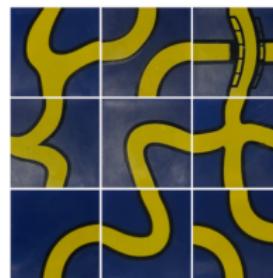


Bruteforce

Avec des schémas (3)

TIPE

Introduction
Modélisation
Bruteforce
Optimisation
Génération de problèmes



Remplissons le reste... On obtient un plateau qu'on note \mathbb{T}_1



Bruteforce

Avec des schémas (4)

TIPE

Introduction

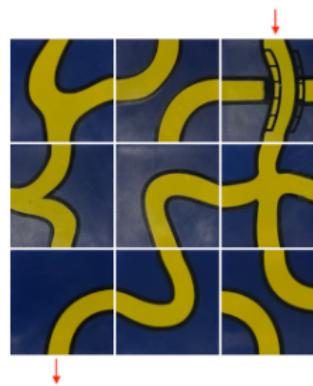
Modélisation

Bruteforce

Optimisation

Génération de
problèmes

Traduisons la condition C_1 sur le schéma.



Bruteforce

Avec des schémas (5)

TIEPE

Introduction

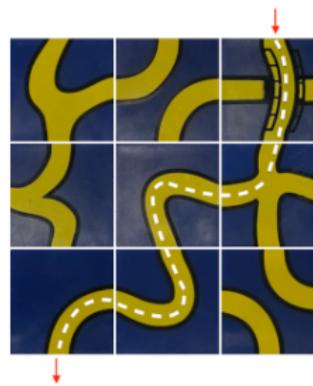
Modélisation

Bruteforce

Optimisation

Génération de
problèmes

On voit bien que \mathbb{T}_1 respecte bien C_1



Bruteforce

Avec des schémas (6)

TIEPE

Introduction

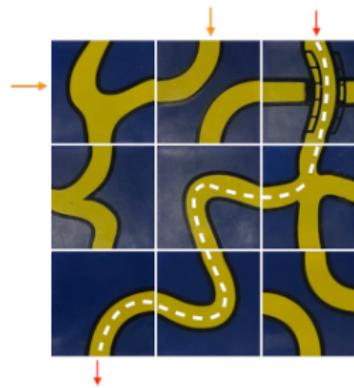
Modélisation

Bruteforce

Optimisation

Génération de
problèmes

Néanmoins C_2 est insatisfaite par \mathbb{T}_1



Bruteforce

Avec des schémas (7)

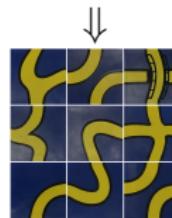
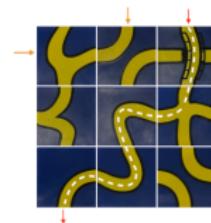
TIPE

Introduction

Modélisation

Bruteforce

Optimisation

Génération de
problèmes

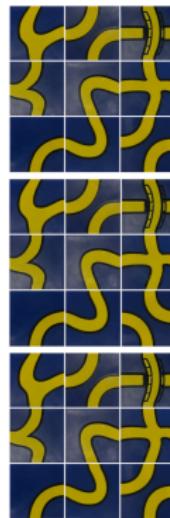
On modifie la dernière pièce posée **en faisant une rotation de 90 degrés** sur cette dernière.

Bruteforce

Avec des schémas (8)

TIEPE

Néanmoins ce nouveau plateau ne respecte pas C_2 . On réitère alors notre opération de rotation...



Bruteforce

Avec des schémas (9)

TIEPE

Aucune rotation ne correspond, on retourne à l'avant dernier slot :



Bruteforce

Avec des schémas (10)

TIEPE

Introduction

Modélisation

Bruteforce

Optimisation

Génération de
problèmes

Etc... On recommence en faisant tourner P_1 de 90 degrés :

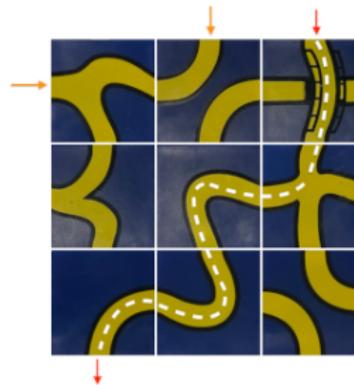


Bruteforce

Avec des schémas (11)

TIEPE

Introduction
Modélisation
Bruteforce
Optimisation
Génération de problèmes



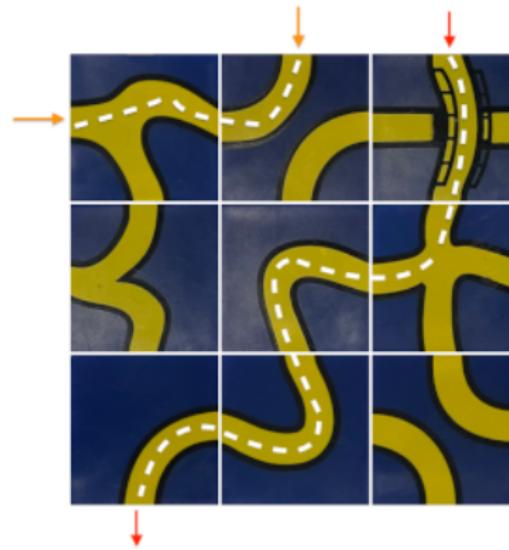
On obtient alors un plateau T_2

Bruteforce

Avec des schémas (12)

TIEPE

Introduction
Modélisation
Bruteforce
Optimisation
Génération de problèmes



Et on remarque que \mathbb{T}_2 respecte bien C_1 et C_2

Bruteforce

Étude de complexité

TIEPE

Introduction

Modélisation

Bruteforce

Optimisation

Génération de
problèmes

Complexité :

$$\mathcal{O}(n! \times 4^n)$$

il est d'ailleurs NP-Complet (\leq_p Couverture exacte). D'où le fait qu'on reste sur la stratégie backtrack par la suite, en l'optimisant.

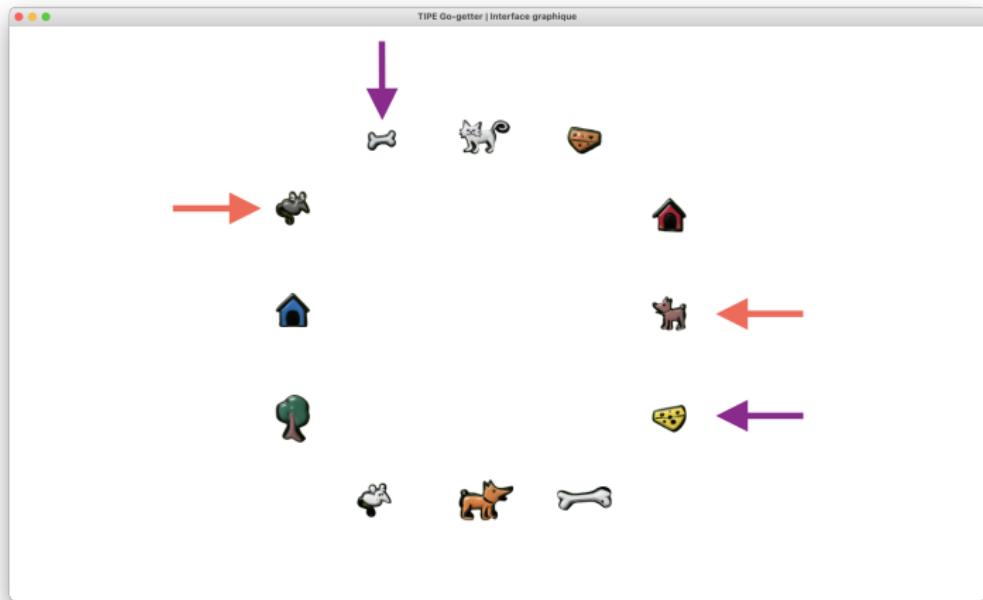
Optimisation

Premier problème : éviter de compléter linéairement

TIEPE

Introduction
Modélisation
Bruteforce
Optimisation
Génération de problèmes

Prenons le plateau avec les contraintes suivantes :



Optimisation

Premier problème : éviter de compléter linéairement

TIEPE

Introduction

Modélisation

Bruteforce

Optimisation

Génération de problèmes

Le joueur humain prendra le réflexe de d'abord compléter les cases avec les contraintes :



Optimisation

Premier problème : éviter de compléter linéairement

TIEPE

Introduction

Modélisation

Bruteforce

Optimisation

Génération de problèmes

On va donc créer une file de priorité pour trouver les slots prioritaires à traiter et les ordonner

- On identifie le nombre de contraintes qu'elle possède
- On lui attribue une valeur de priorité en fonction de ce nombre

Optimisation

Premier problème : éviter de compléter linéairement

TIE

Introduction

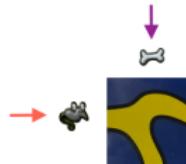
Modélisation

Bruteforce

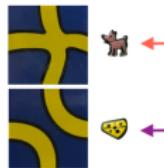
Optimisation

Génération de
problèmes

On associe la priorité 20 par exemple au slot à 2 contraintes :



et la priorité 10 aux deux autres slots :



Les autres slots possèdent la priorité 0. On sait donc quels sont les slots à traiter en premier.

Optimisation

Troisième problème : gestion du deck

TIEPE

Introduction

Modélisation

Bruteforce

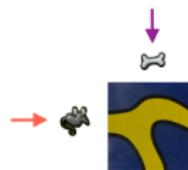
Optimisation

Génération de
problèmes

Concernant la file **par slot**,

- On sélectionne juste les cartes qui sont capables de répondre au nombre de contraintes

Par exemple, pour la contrainte suivante, il n'y a que les cartes à au moins 2 chemins qui peuvent convenir :



ce qui permet de réduire le nombre de cartes éligibles par slot, et donc réduire le nombre de coups.

Optimisation

Deuxième problème : vérification partielle des configurations

TIEPE

Introduction

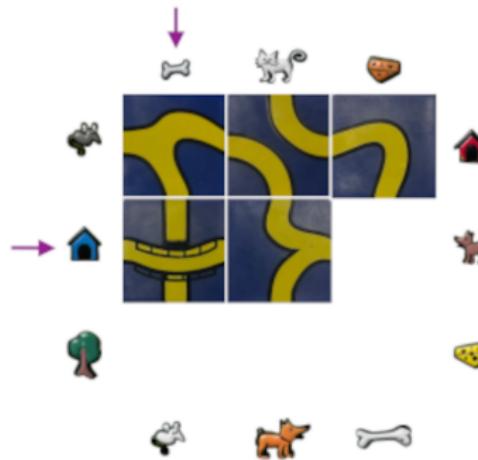
Modélisation

Bruteforce

Optimisation

Génération de problèmes

Il y a également un besoin de répondre directement si une configuration partielle correspond au critère ou non.
Exemple d'une configuration posant problème :



Optimisation

Deuxième problème : vérification partielle des configurations

TIEPE

Introduction

Modélisation

Bruteforce

Optimisation

Génération de problèmes

On va donc créer une heuristique vérifiant si :

- ➊ la construction respecte bien les contraintes imposées pour les cartes déjà posées (exemple précédent)
- ➋ il n'y a pas de cycles fermés
- ➌ il n'y pas de situations dites sans issues (profondeur de recherche limitée à 1)

2. Exemple de cycle fermé

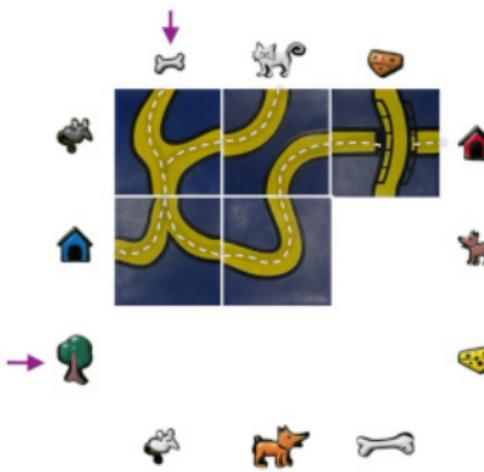
Introduction

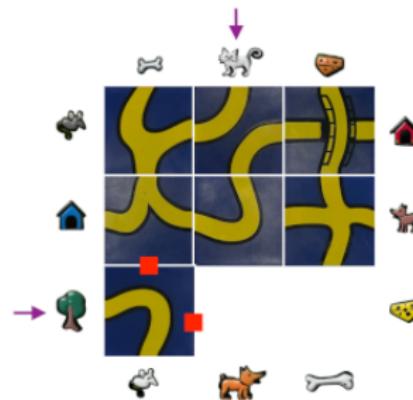
Modélisation

Bruteforce

Optimisation

Génération de problèmes





Problèmes :

- ➊ Complète linéairement le tableau
- ➋ Pas d'élimination immédiate des configurations absurdes, impossibles
- ➌ Dépend trop de l'ordre dans lequel les cartes sont disposées sur le deck

Statistiques :

ne peut résoudre que les niveaux faciles en moins d'1'

Optimisation

Bilan des optimisations

TIEPE

Introduction
 Modélisation
 Brute-force
Optimisation
 Génération de problèmes

Bilan de performance (arrondie à l'unité supérieure):

Niveaux	Sans opt, c.i	Sans opt, c.p
Facile	1'	30'
Moyen	15'	∞
Difficile	∞	∞
Très Difficile	∞	∞

Niveaux	Avec opt, c.i	Avec opt, c.p
Facile	5"	10"
Moyen	8"	15"
Difficile	1'	4'
Très Difficile	5'	30'

c.i = chemin imparfait ; c.p = chemin parfait ; opt = optimisation

Génération de problèmes

Quelques niveaux

TIEPE

Introduction

Modélisation

Bruteforce

Optimisation

Génération de
problèmes

Générer un problème à k contraintes consiste à la réalisation de 2 étapes :

- Générer aléatoirement $k \times (2$ emplacements distincts et leurs côtés)
- Vérifier avec notre algorithme si une solution au problème généré avec chemin parfait existe. On déduit le niveau de difficulté du temps requis à l'algorithme.

Génération de problèmes

Quelques niveaux

TIEPE

Un exemple de problème généré et résolu :

$$\bullet \text{ bone} \rightarrow \text{ bone}$$

$$\bullet \text{ dog} \rightarrow \text{ dog}$$

$$\bullet \text{ bone} \rightarrow \text{ bone}$$

$$\bullet \text{ tree} \rightarrow \text{ bone bone}$$

$$\bullet \text{ dog} \rightarrow \text{ cheese}$$

