PRO3600: Mars Lander

Augustin Bresset | Zacharie March

January 2024

FIGURE 1 – Logo TSP

Table des matières

1 Introduction

1.1 Présentation du problème : Mars Lander

Mars Lander est un problème d'optimisation proposé sur la plateforme *CodinGame* s. d. Ce jeu consiste en le contrôle d'un vaisseau spatial et de ses caractéristiques (position, vitesse, puissance des moteurs, angles de rotation...) afin de le faire atterrir en toute sécurité et en douceur sur Mars sur une surface plane, quelle que soit sa position de départ. L'idée est donc de trouver une trajectoire fonctionnelle.

1.1.1 Environnement

La surface de Mars est représenté localement par une suite continue de segments. Parmis ces segments un et un seul est rigouresement vertical, celui-ci définit la zone d'atterissage que la navette doit viser.

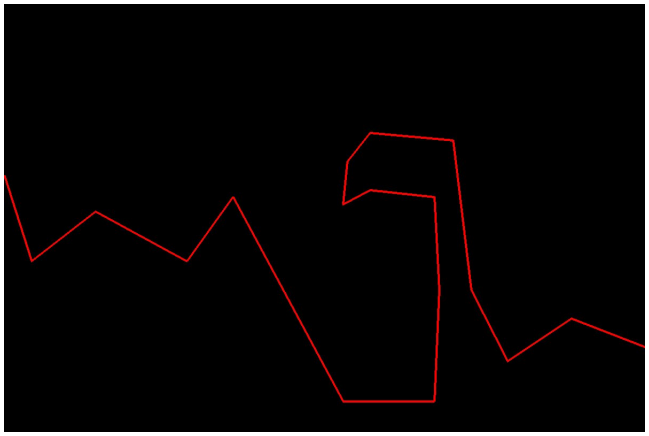

FIGURE 2 – Exemple de carte du problème

1.1.2 Espace des Etats

L'état du vaisseau est représenté par un vecteur de 7 valeurs :

- x : position horizontale
- y : position verticale
- $hSpeed$: vitesse horizontale
- $vSpeed$: vitesse verticale
- $fuel$: quantité de carburant restante
- $rotate$: angle de rotation
- $power$: puissance des moteurs

Certains de ces paramètres est borné par une valeur minimale et maximale que voici :

- x : $[0, 7000]$
- y : $[0, 3000]$
- $fuel$: $[0, \infty[$
- $rotate$: $[-90, 90]$
- $power$: $[0, 4]$

1.1.3 Espace des Actions

Toutes les secondes, en fonction des paramètres d'entrée (position, vitesse, fuel, etc.), le programme doit fournir le nouvel angle de rotation souhaité ainsi que la nouvelle puissance des fusées de Mars Lander. Mais les commandes de puissance des fusées et de l'angle de rotation sont bornés par les valeurs suivantes qui forme l'espace des actions :

- rotation : $[-15, 15]$
- puissance : $[-1, 1]$

1.1.4 Conditions d'atterrissage

Les conditions d'atterissage doivent être représentées :

- atterrir sur un terrain plat
- atterrir en position verticale (angle d'inclinaison = 0°)
- la vitesse verticale doit être limitée ($\leq 40m.s^{-1}$ en valeur absolue)
- la vitesse horizontale doit être limitée ($\leq 20m.s^{-1}$ en valeur absolue)

1.2 Algorithme Génétique

Afin de résoudre ce problème, nous avons choisi d'utiliser une approche heuristique, les algorithmes génétiques.

Les algorithmes génétiques sont des algorithmes d'optimisation stochastique inspirés de la théorie de l'évolution naturelle. Ils sont basés sur le principe de **sélection naturelle**.

Dans un premier temps on génère une **population** de solution aléatoire, puis on évalue la qualité de chaque solution. On sélectionne fait ensuite évoluer cette population à travers des phénomènes semblables à la sélection naturelle tel que la **sélection**, la **reproduction** et la **mutation**. On répète ce processus jusqu'à ce qu'une solution satisfaisante soit trouvée. Dans notre problème, une solution est définie par une trajectoire, c'est à dire une suite de commande à effectuer par le vaisseau.

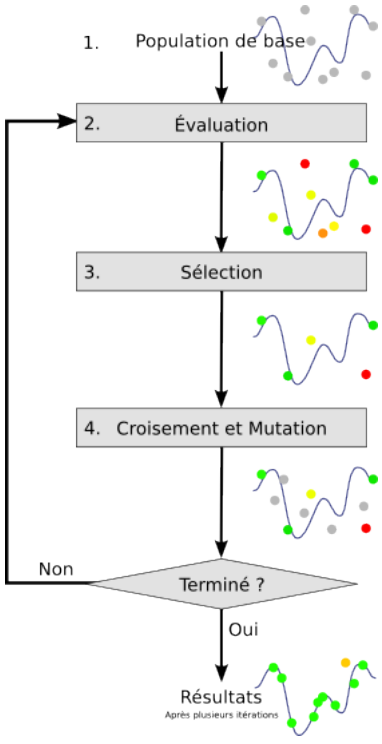

FIGURE 3 – *Schéma d'un algorithme génétique* s. d.

Le calcul du score peut dépendre de nombreux paramètres, dans notre cas on a choisi de prendre en compte les paramètres suivants :

- La **distance** au sol qui sépare la zone de collision et d'atterissage
- Les **vitesses** verticale et horizontale
- L'**angle** d'inclinaison du vaisseau à l'atterissage

1.3 Objectif

Notre objectif est de résoudre ce problème à l'aide d'une approche heuristique, les algorithmes génétiques. Pour cela on à définit plusieurs points afin de mener à bien notre projet :

- Création d'une **interface graphique** sur pygame
- Mise en place d'une **interface client** permettant de piloter la navette
- Calcul de **solution** notamment à l'aide d'algorithme génétique

1.3.1 Cahier des charges

Développer une applicattion avec interface graphique. Les fonctionnalités attendues sur l'application sont les suivantes :

- Choix de la solution à utiliser
- Choix de la carte à utiliser
- Exécution de la simulation
- Adapter l'affichage en fonction de la solution

1.4 Organisation du projet

1.4.1 Répartition des tâches

En claire la répartition des tâches est la suivante :

- **Augustin Bresset**
  Développement de l'environnement et des solutions
- **Zacharie March**
  Développement de l'interface graphique et de l'interface client

Dans les deux cas, un apprentissage de la librairie pygame est nécessaire. Que ce soit pour la création de la solution "manuelle" (contrôle par l'utilisateur) ou pour la création de l'interface graphique.

1.4.2 Outils de travail

Afin de communiquer, nous nous reposons pour la communication active sur plusieurs outils, étant deux, nous n'étions pas restreint par le besoin de créer un groupe sur un outil de communication. Pour la commun passive nous avons créée une organization sur Github dans laquelle on peut trouver deux repertoire :

- **backend** : Contient le code source du projet
- **meta** : Contient la documentation du projet (et ce livrable)

1.4.3 Organisation du code

Afin de faciliter la lecture du code ainsi que le développement, nous avons séparer en plusieurs modules le code source :

- **environment** : Contient le code source de l'environnement
- **gui** : Contient le code source de l'interface graphique
- **solution** : Contient le code source des solutions
- **utils** : Contient le code source des fonctions utilitaires

2 Développement

2.1 Architecture du projet

2.1.1 Environnement

L'environnement est composé de plusieurs classes :

- **Environment** : Objet permettant de gérer les interactions entre les entités et l'environnement
- **Surface** : Classe permettant de gérer la surface de Mars
- **Lander** : Classe permettant de gérer le vaisseau
- **Action** : Classe permettant de gérer les actions du vaisseau
- **Utils** : Ensemble de fonctions utilitaires et constantes
- **Entity** : Classe abstraite permettant de gérer les entités
- **Constants** : Classe permettant de gérer les constantes

2.1.2 Gui

2.1.3 Solution

Une solution est définie par une classe abstraite **AbstractSolution** qui permet de définir les méthodes communes à toutes les solutions. On a ensuite deux types de solutions :

- **ManualSolution** : Solution manuelle permettant de piloter le vaisseau à l'aide du clavier
- **GeneticSolution** : Solution génétique permettant de piloter le vaisseau à l'aide d'un algorithme génétique

2.2 Problèmes rencontrés

Tout au long du projet, on a essayé de garder une architecture de code permettant une meilleure répartition des processus tout en gardant une cohérence dans le code.

Nous voulions diviser les repertoires github entre le frontend et le backend, mais nous avons finalement pas rencontrés ce besoins étant donné que l'on travaillait sur le même langage. C'est pourquoi le nom du repertoire est **backend**.

Références

CodinGame (s. d.). URL : https://www.codingame.com/multiplayer/optimization/mars-lander.

Schéma d'un algorithme génétique (s. d.). URL : https://fr.wikipedia.org/wiki/Algorithme_g%C3%A9n%C3%A9tique.