PRO3600: Mars Lander

Augustin Bresset | Zacharie March

January 2024

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.

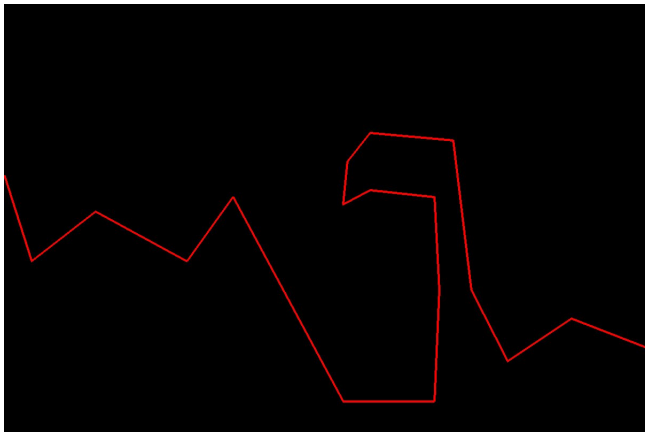

FIGURE 1 – Exemple de carte du problème

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 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
- Création de trajectoire répondant au problème notamment à l'aide d'algorithme génétique

1.2.1 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

1.2.2 Algorithme Génétique

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 et de génétique.

Dans un premier temps on génère une population de solution aléatoire, puis on évalue la qualité de chaque solution. On sélectionne ensuite les meilleurs solutions afin de les faire se reproduire et créer une nouvelle génération de solution. 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.

L'évolution d'une population de solution se fait en plusieurs étapes :
- **Sélection** : On sélectionne les meilleurs solutions de la population
- **Reproduction** : On fait se reproduire des solutions choisies aléatoirement
- **Mutation** : On applique une mutation à la nouvelle génération afin d'explorer de nouveaux espaces

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

2 Développement

2.1 Architecture du projet

2.1.1 Environnement

L'environnement est composé de plusieurs classes :

- **Environment** : Classe principale de l'environnement, elle permet 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** : Classe permettant de gérer les constantes et les fonctions utilitaires
- **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

Références

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