2019-2020

Lunar Lander

Mission alunissage

GAUDET Clément, VIALLARD Thibault

UPEM – L1 Math info

Groupe : TD D / tp 7

Table des matières

[1 Présentation du projet 2](#_Toc29156454)

[2 Manuel utilisateur 2](#_Toc29156455)

[3 Structure générale du programme, principales fonctions et variables 3](#_Toc29156456)

[3.1 Corps du programme 3](#_Toc29156457)

[3.1.1 Principales variables 3](#_Toc29156458)

[3.1.2 Structure du corps du programme 4](#_Toc29156459)

[3.2 Principales fonctions 4](#_Toc29156460)

[4 Organisation et déroulement du projet 6](#_Toc29156461)

# Présentation du projet

**Lunar Lander** est un jeu de borne d'arcade développé par Atari et sorti en 1979. Le jeu, qui se joue à un seul joueur, consiste à poser un atterrisseur sur la Lune, le joueur pouvant contrôler l'inclinaison du module lunaire ainsi que la propulsion du moteur. La difficulté du jeu étant la gestion du carburant, le jeu continuant tant que le module avait du carburant. Le joueur était assisté par des informations affichées à l'écran comme le pourcentage de carburant restant, la vitesse verticale, la vitesse horizontale ainsi que l'altitude du module. Le jeu fut très bien accueilli par les joueurs, notamment grâce à ses graphismes vectoriels et ses différents points de vue selon l'altitude du module.

L'objectif de notre projet de programmation du S1 2019-2020 de L1 Math Info était de recréer ce jeu grâce au langage Python et le module graphique upemtk, en incluant autant de fonctionnalités du jeu original que possible : physique de la fusée, génération aléatoire du terrain, affichage des informations à l'écran, gestion du carburant, etc.

Ce rapport décrit le fonctionnement du jeu, les principaux points du code, et ce qui a entouré son développement.

# Manuel utilisateur

Le jeu se lance en chargeant le fichier lunarlander.py dans l’interpréteur python.

L’écran titre présente deux boutons pouvant être cliqués :

* **« Jouer » :** lance le jeu
* **« Options »** : permet d’accéder à un menu pour changer les paramètres de la partie

Le menu des options permet de modifier un certain nombre de paramètres :

* **Mode de jeu :**
  + **Mode A** : les réacteurs latéraux de la fusée font pivoter celle-ci
  + **Mode B** : les réacteurs latéraux de la fusée bougent la fusée latéralement sans rotation
* **Vitesse maximale d’alunissage** : plus elle est élevée, plus la fusée peut atterrir vite
* **Consommation de carburant**
* **Gravité**
* **Puissance des propulseurs**

Pour ces quatre derniers paramètres, cliquer sur les boutons fléchés permet de les modifier.  
Le bouton **« Sortir »** permet de retourner à l’écran titre.

Lors du démarrage d’une partie, la fusée démarre vers le haut de l’écran et commence immédiatement à chuter sous l’effet de la gravité.

La fusée se contrôle au clavier, les contrôles sont les suivants :

* **Touche** **G**: active le propulseur principal
* **Touche fléchées gauche et droite** : activent les réacteurs latéraux, pivotent la fusée ou la font bouger latéralement selon le mode de jeu choisi
* **Touche P**: met le jeu en pause, appuyer de nouveau sur P pour en sortir

L’utilisation des propulseurs, principal ou latéraux, consomme du carburant. Si le stock est épuisé, les propulseurs deviennent inutilisables, et la fusée incontrôlable.

Le sol, en bas de l’écran, est généré aléatoirement à chaque partie. Le but du jeu est de faire atterrir la fusée sans crash. Plusieurs conditions sont requises pour cela :

* Le terrain doit être plat et horizontal
* La fusée doit être verticale
* La vitesse doit être suffisamment faible au moment de l’atterrissage

Si toutes ces conditions sont respectées, la partie sera gagnée. Sinon, il y aura un crash.

Une barre horizontale en haut de la fenêtre renseigne sur différents paramètres, de gauche à droite on trouve :

* **Vitesse :** la flèche rouge représente le vecteur vitesse de la fusée, le cercle vert au centre représente la zone permettant un alunissage réussi. A droite de cette représentation, la valeur numérique de la vitesse est indiquée
* **Altimètre**: donne l’altitude de la fusée par rapport au sol situé directement sous celle-ci
* **Jauge de carburant :** la barre verticale rouge indique la quantité de carburant restant. Quand la barre se vide, il n’y a plus de carburant
* **Temps :** affiche le temps passé depuis le début de la partie
* **Score :** nombre d’alunissage réussis d’affilé

A la fin d’une partie, deux boutons vous permettent de choisir ce que vous voulez faire :

* **« Rejouer » :** relance une partie avec les mêmes paramètres
* **« Quitter »** : ferme le jeu

# Structure générale du programme, principales fonctions et variables

Le fichier lunarlander.py est schématiquement scindé en trois parties :

* Imports et définition de certaines constantes
* Définition des fonctions
* Corps du programme

## Corps du programme

### Principales variables

* ***parametres***: liste stockant plusieurs paramètres du jeu pouvant être modifiés par le joueur dans le menu des options, dans l’ordre, vitesse maximale d’alunissage, ratio de consommation du carburant, force de la gravité et ratio de puissance des propulseurs
* ***mode***: string, mode de jeu choisi (‘A’ ou ‘B’)
* ***fusee\_pos :*** tuple stockant les coordonnées x et y de la fusée sous formes d’entiers
* ***fusee\_angle :*** flottant stockant l’angle de la fusée en degrés par rapport à l’horizontale
* ***fusee\_vit :*** tuple stockant le vecteur vitesse de la fusée sous forme cartésienne (x, y)
* **Vecteurs accélérations :** tuples stockant des vecteurs accélération de diverses origines sous forme cartésienne (x, y)
  + ***gravite*** : vecteur pointé vers le bas, représentant la gravité
  + ***propulsion :*** représente l’accélération du réacteur principal de la fusée
  + ***prop\_laterale :*** représente l’accélération des réacteurs latéraux, utilisé dans le mode B
* ***fusee\_vit\_angulaire :*** flottant stockant la vitesse de rotation actuelle de la fusée
* ***fusee\_accel\_angulaire :*** flottant stockant l’accélération de la rotation de la fusée
* ***carburant\_max :*** entier stockant la quantité de carburant initiale de la fusée
* ***carburant :*** entier stockant la quantité de carburant restante au temps t
* ***terrain :*** liste de tuples, chaque tuple représente les coordonnées (x, y) d’un point du sol, terrain étant donc la liste de tous les points formant la topographie du sol
* ***timer :*** entier stockant le temps passé depuis le début de la partie

### Structure du corps du programme

Une **courte partie initiale** permet de créer la fenêtre de jeu, de lancer l’écran-titre et d’initialiser les paramètres de la partie choisis par le joueur.

Puis **deux boucles imbriquées** forment le corps du programme :

* Une boucle permettant de recommencer une partie tant que le joueur le désire
* Une boucle permettant de simuler chaque 1/30 secondes du jeu

La **structure de la première boucle** est comme suit :

* **Initialisation** des diverses variables nécessaires au fonctionnement du jeu, la plupart ayant été décrites ci-dessus
* **Affichage** des éléments restant de façon permanent à l’écran (notamment, le fond et le sol, qui n’ont nul besoin d’être réaffichés à chaque trentième de seconde)
* **Boucle** simulant le jeu, qui s’arrête lorsque la fusée touche le sol
* **Vérification des conditions de victoire**, affichage du résultat et demande au joueur de faire un choix de rejouer ou non

La structure de la seconde boucle est comme suit :

* **Affichages** de la fusée et de la barre d’information au haut de la fenêtre
* **Gestion des évènements** :
  + Fermeture de la fenêtre en appuyant sur la croix en haut à droite
  + Pause en appuyant sur P
  + Propulsion principale en appuyant sur G
  + Rotation/propulsion latérale en appuyant sur les touches fléchées
* **Gestion de la physique et du mouvement** de la fusée
* **Vérification d’une collision** avec le sol

Le corps principal du programme ne contient que peu de parties techniques, la majorité du code se trouvant dans les fonctions définies avant.

## Principales fonctions

Nous allons ici parler des fonctions des plus importantes du programme.

* ***update\_vitesse :*** cette fonction est celle qui met à jour le vecteur vitesse de la fusée en fonction des différents vecteurs d’accélérations s’appliquant sur la fusée.   
  Son fonctionnement est relativement simple, additionnant les composantes horizontales et verticales du vecteur vitesse précédent et de tous les vecteurs d’accélération.  
  La partie plus technique est de limiter la vitesse maximale. Si la norme du vecteur vitesse dépasse la vitesse maximale prévue, un passage par les coordonnées polaires du vecteur permet de réduire la norme tout en maintenant l’angle.
* ***update\_vitesse\_angulaire :*** de façon similaire à la fonction précédente, celle-ci met à jour la vitesse de rotation de la fusée. En cas d’accélération, elle augmente jusqu’au maximum prévu. En l’absence d’accélération en cours, la fonction décélère automatiquement le mouvement de rotation jusqu’à l’arrêt, permettant de rendre la rotation de la fusée plus contrôlable
* ***move\_fusee :*** il s’agit de la fonction qui bouge la fusée. Elle est simple, se contentant d’additionner le vecteur vitesse à la position de la fusée. Elle comporte en plus du code empêchant la fusée de sortir de la fenêtre
* ***check\_gnd\_collision :*** cette fonction est très importante puisque c’est elle qui permet de déterminer quand la fusée rentre en collision avec le sol. Elle cherche le segment de terrain directement sous la fusée, et recherche si l’un des 4 côtés du rectangle représentant la fusée croise ce segment de sol.  
  La fonction vérifie également une collision avec le segment de sol à gauche et celui à droite du segment directement sous la fusée.
* ***check\_victoire :*** cette fonction, une fois la partie finie, vérifie si l’atterrissage était réussi ou non, en se basant sur un certain nombre de paramètres. Elle vérifie que la fusée est en effet sur le sol, si la pente du segment d’atterrissage est inférieure à 5° par rapport à l’horizontale, si la fusée est verticale (à plus ou moins 5°) et enfin si la vitesse de la fusée est inférieure au seuil choisi par le joueur
* ***cree\_terrain :*** cette fonction est responsable de la génération aléatoire du terrain  
  Pour la modélisation du terrain nous avons choisi de réaliser un terrain aléatoire sans « surplombs ». Nous avons choisi cette méthode après avoir réalisé successive un terrain plat puis un terrain avec simples ligne brisée, nous décidant alors à réaliser un terrain aléatoire avec relief variable. Le terrain est composé de plusieurs types de surface : des surfaces planes, montantes, descendantes et des collines. Chacune de ces surfaces à une hauteur ou pente aléatoire. Le sol est divisé en 12 segments de 5 points, et un type de terrain est choisi aléatoirement pour chacun de ces segments, en garantissant au moins un terrain plat, et les points de chaque segments sont générés aléatoirement selon le type de terrain.
* **Fonctions affichant et gérant des écrans :**Ces trois fonctions s’occupent d’afficher des écrans et menus, et gèrent les évènements, notamment les clics de souris, permettant de la naviguer
  + ***ecran\_titre***
  + ***menu\_options***
  + ***game\_over***

# Organisation et déroulement du projet

Une fois le sujet du projet choisi, une étape d’analyse s’est imposée, notamment une lecture attentive des différents fichiers fournis. En raison des circonstances pendant le développement du projet, quasiment aucune rencontre n’a pu se faire, et toute la communication a été faite via internet, avec cependant une consultation fréquente entre nous deux.

Rapidement, il s’est imposé d’établir un cahier des charges, en reprenant toutes les fonctionnalités demandées dans la liste des sujets, permettant d’être sûr de ne pas en oublier une et d’avoir un œil sur l’avancement du projet.

Après cela, une première phase de développement, réalisée à deux, a été de coder un premier prototype de la fusée et des mécaniques physiques régissant son mouvement. Un mix de programmation et de recherche s’est opéré pour réaliser des mécaniques cohérentes.

Suivant l’obtention d’un prototype satisfaisant, deux parties complexes du programme ont été réparties entre nous deux. L’un s’est occupé de la détection de la collision de la fusée avec le sol, l’autre de la génération aléatoire du terrain. Avant de nous lancer chacun dans notre partie correspondante, nous avons convenu ensemble de la façon dont le terrain serait stocké et représenté en jeu, c’est-à-dire sous forme d’une liste de points tous séparés de 20 pixels horizontalement, le sol formé en reliant tous ces points entre eux.

Ces deux fonctionnalités étaient sans doute les plus compliquées à mettre en place, et ont toutes les deux nécessitées des recherches sur la manière de procéder, ainsi qu’une importante quantité d’expérimentation. Une recherche notamment sur les algorithmes de détection des collisions a été faite, avec au final la décision de simplement vérifier le croisement de segments au lieu de se diriger vers un algorithme complet de détection de la collision entre deux polygones convexes. Pour la génération aléatoire du terrain, l’exploration de certaines fonctions de Python non vues auparavant ont été faites, telle que la fonction *map*.

Une fois un résultat satisfaisant obtenu, et le fonctionnement du jeu en place, la troisième phase a été de mettre en place l’interface du jeu et les menus, la partie la plus simple, mais coûteuse en temps et en lignes de code.

Une fois le jeu terminé avec toutes les fonctionnalités demandées, il est resté majoritairement un travail de raffinage du code et de l’ajout quelques fonctionnalités supplémentaires.