

INFO-F-106 : PROJET D'INFORMATIQUE

PROJET 1 – ULBLOQUÉ

Anthony Cnudde Gwenaël Joret Tom Lenaerts Robin Petit
Loan Sens Cédric Simar

version du 5 novembre 2024

Présentation générale

L'objectif du premier projet est de réaliser une implémentation en Python 3 du jeu décrit plus loin. C'est un jeu à 1 joueur sur une grille donnée. Le but du jeu est de permettre à une voiture de sortir du parking en déplaçant les voitures qui lui bloquent la route.



Le projet est décrit en long et en large plus loin dans ce document, mais avant d'entrer dans le cœur du sujet, voici d'abord quelques consignes générales pour la réalisation de ce projet. Elles seront également d'application pour le second projet au second semestre.

Consignes générales

- L'ensemble du projet est à réaliser en **Python 3**.
- Le module INFO-F106 est constitué de deux projets de programmation distincts, un par quadrimestre.
- En plus de ces projets à remettre, chaque étudiant(e) devra remettre un rapport final au second quadrimestre (Q2).

- La répartition des points est la suivante :
 - Projet 1 (Q1, ULBloqué) : 40 points.
 - Projet 2 (Q2) : 40 points.
 - Rapport (Q2) : 20 points.
 - **Total : 100 points.**
- Les deux projets et le rapport devront être remis sur l'UV.
- **Il n'y aura pas de seconde session pour ces projets !**

Veillez noter également que le projet vaudra **zéro** sans exception si :

- les noms de fonctions sont différents de ceux décrits dans cet énoncé, ou ont des paramètres différents ;
- à l'aide d'outils automatiques spécialisés, nous avons détecté un plagiat manifeste (entre les projets de plusieurs étudiant(e)s, ou avec des éléments trouvés sur Internet). Nous insistons sur ce dernier point car l'expérience montre que chaque année une poignée d'étudiant(e)s pensent qu'un petit copier-coller d'une fonction, suivi d'une réorganisation du code et de quelques renommages de variables passera inaperçu... Ceci sera sanctionné d'une note nulle pour l'entièreté du projet pour toutes les personnes impliquées, ainsi que d'éventuelles autres sanctions. Afin d'éviter cette situation, veillez en particulier à ne pas partager de bouts de codes sur des forums, Facebook, Discord, etc.

Soumission de fichiers

La soumission des fichiers se fait via la section ouverte sur l'UV.

Tests automatiques

Pour évaluer votre code, nous mettrons à votre disposition un fichier de test. Ce fichier contient des tests qui valideront (ou non) les fonctions que vous avez implémentées. Ces tests vous permettront de savoir si votre code réagit comme nous l'attendons ou si vous devez le retravailler. Cependant, cela implique que vous devrez respecter à la lettre les consignes qui vous sont données, sans quoi ces tests échoueront.

Librairies permises

Hormis celles mentionnées dans les consignes, aucune librairie n'est autorisée pour ce projet.

Communication avec l'équipe des enseignants

Remarque préliminaire : il y a 608 étudiant(e)s inscrits à ce cours (chiffres d'octobre 2024), il nous est donc simplement impossible de répondre individuellement aux questions par email. Ceci nous amène à la règle suivante :

Règle d'or : ne jamais envoyer d'email !

Vous aurez la possibilité de poser vos questions concernant le projet 1 sur un forum sur la page UV du cours. Vos assistants regarderont périodiquement le forum et répondront aux questions pertinentes. Ils ne répondront par contre pas aux questions dont la réponse se trouve dans l'énoncé, ou qui ont déjà été posées auparavant. Veuillez donc à bien lire l'énoncé dans son entièreté, et à chercher dans le forum avant de poser vos questions.

Toute question concernant la note reçue au projet devra être posée lors de la visite des copies, et uniquement à cette occasion. À nouveau, n'écrivez pas d'email (ou message Teams, etc) aux assistants / titulaires à ce sujet.

Notez que nous ne répondrons simplement pas aux emails concernant le projet d'année si jamais vous nous en envoyez. La seule exception à cette règle concerne les emails d'ordre administratif adressés aux titulaires (certificat maladie, réorientation, EBS, etc.)

Objectifs pédagogiques

Ce projet *transdisciplinaire* permettra de solliciter diverses compétences.

- *Des connaissances vues aux cours de programmation, langages, algorithmique ou mathématiques.* L'ampleur des deux projets requerra une analyse plus stricte et poussée que celle nécessaire à l'écriture d'un projet d'une page, ainsi qu'une utilisation rigoureuse des différents concepts vus aux cours.
- *Des connaissances non vues aux cours.* Les étudiant(e)s seront invité(e)s à les étudier par eux-mêmes, aiguillé(e)s par les *tuyaux* fournis par l'équipe encadrant le cours.
- *Des compétences de communication.* Les étudiant(e)s devront rédiger un rapport scientifique en \LaTeX . Ce sera l'occasion pour les étudiant(e)s de se familiariser avec ce langage, utilisé pour la rédaction de documents scientifiques. Une orthographe correcte sera exigée.

En résumé, vous devrez démontrer que vous êtes capables d'appliquer des concepts vus aux cours, de découvrir par vous-même des nouvelles matières, et enfin de communiquer de façon scientifique le résultat de votre travail.

Bon travail !

Contexte

Comme tous les matins, M. Joret prend sa voiture pour se garer dans le parking de l'ULB. Cependant, en ce jour très particulier, le gang des voleurs de volants de voitures s'empare de tous les volants des véhicules présents ce jour là.

En fin de journée, M. Joret ainsi que les autres conducteurs se rendent compte du funeste évènement. Heureusement pour lui, la sortie se trouve juste en face de son véhicule! En revanche, d'autres voitures sont dans son chemin.

Arrivera-t-il à déplacer les véhicules afin de trouver la voie vers la liberté?

Le jeu

Chaque voiture est représentée par une lettre, et le parking est une grille où chaque case peut être vide ou occupée par une partie d'une voiture.

.	.	B	C	C	C
.	.	B	.	.	.
A	A	B	.	.	→
D	.	.	E	E	F
D	.	.	.	G	F
.	H	H	H	G	F

FIGURE 1 – Exemple de parking.


Cette configuration de parking servira d'exemple de référence dans l'énoncé ainsi que dans les tests automatiques du code


Le but du jeu est de déplacer les voitures pour libérer le passage et permettre à la voiture A de sortir du parking. Vous avez un nombre limité de mouvements pour y parvenir.

1 Implémentation Technique

1.1 Structure globale

1. **Initialisation** : Le parking est chargé à partir d'un fichier texte qui décrit la disposition initiale des voitures et le nombre maximum de mouvements autorisés.
2. **Affichage** : Le jeu affiche le parking avec les voitures à leur position initiale.
3. **Sélection de la voiture** : Le joueur choisit une voiture à déplacer en entrant la lettre correspondante.
4. **Choix de la direction ou changement de voiture** : Le joueur choisit ensuite la direction dans laquelle déplacer la voiture (haut, bas, gauche, droite) OU de changer de voiture à déplacer.

Exemple :  déplacerait la voiture B vers le bas de deux cases et C à gauche de une case. Note : le fait qu'on ait appuyé sur A n'a servi à rien car on a directement enchaîné avec la sélection d'une autre voiture (ici C).

D'autres exemples complets de résolution du parking d'exemple son repris dans les sections 2.1 et 2.2. Si la touche  est pressée, alors la partie est immédiatement abandonnée.

5. **Vérification** : Le jeu vérifie si le mouvement est valide (la voiture reste dans les limites du parking et ne heurte pas d'autres voitures).
6. **Déplacement** : Si le mouvement est valide, la voiture est déplacée. Sinon, le mouvement est annulé (et ne compte pas dans le nombre de mouvements effectués).
7. **Vérification de la victoire** : Le jeu vérifie si la voiture cible a atteint la sortie du parking.
8. **Fin du jeu** : Le jeu se termine lorsque le joueur a réussi à sortir la voiture cible, épuisé le nombre de mouvements autorisés ou abandonné.

1.2 Système de coordonnées

Lors de la mention de coordonnées dans ce projet, soyez vigilants que le système de coordonnées utilisé est le cartésien à deux dimensions avec l'axe *y* **inversé** (orienté vers le bas au lieu de vers le haut).

Exemple : la voiture E est composée de deux morceaux : celui de gauche en (3, 3) et celui de droite en (4, 3)

1.3 Éléments

1.3.1 Voiture car

Dans le contexte de ce jeu, une voiture est un élément mobile représenté par une lettre alphabétique (par exemple, A, B, C, etc.) sur le plateau de jeu. Chaque voiture a une certaine taille (> 1) et orientation (horizontale ou verticale). Les voitures se déplacent **uniquement** dans la direction de leur orientation (gauche ou droite pour horizontale et haut ou bas pour verticale).

La voiture A sera **toujours** horizontale, sur le coté gauche et sera celle qui doit s'échapper par le coté droit pour que la partie soit un succès.

Techniquement, une voiture dans le code sera toujours représentée avec la structure suivante :

```
car = [position: tuple(x: int, y: int), orientation: str, taille: int]
```

- **position/origine** : tuple contenant les coordonnées x et y du véhicule. Ce point est celui d'origine du véhicule par rapport à l'origine de la grille, donc il sera toujours "le point le plus en haut à gauche" qui compose la voiture.
- **orientation** : soit 'h' (pour horizontale) ou 'v' (pour verticale).
- **taille** : longueur ou hauteur du véhicule (dépendamment de son orientation).

Listing 1 – Exemple de voiture dans le code

```
car_A = [(0, 2), 'h', 2]
car_B = [(2, 0), 'v', 3]
car_E = [(3, 3), 'h', 2]
```

Dans le cadre de ce projet, seuls **x** et **y** seront amenés à être modifiés. Autrement dit, l'orientation et la taille seront constantes, seule la position de la voiture change.

1.3.2 Groupe de voiture cars

La liste **cars** contient toutes les voitures présentes sur le plateau de jeu. C'est une liste de **car** (voir 1.3.1).

L'indice dans la liste permet de déterminer, pour chaque voiture, des valeurs comme :

- **Nom** : Correspond à la **index+1**-ème lettre de l'alphabet (on suppose qu'il n'y aura pas plus de 26 voitures). Autrement dit, la première voiture sera **A**, la deuxième **B**, etc.

Attention : La voiture avec l'indice 0 (représentée par la lettre **A**) est la voiture principale que le joueur doit déplacer vers le bord droit du plateau pour gagner la partie.

- **Couleur** : Chaque voiture utilisera une couleur qui servira lors de l'affichage comme couleur "de fond".

La voiture **A** sera la seule à être blanche (\u001b[47m).

Les autres voitures seront colorées en utilisant les six couleurs suivantes comme couleur de fond dans l'ordre, et de manière cyclique :

- rouge : \u001b[41m
- vert : \u001b[42m
- jaune : \u001b[43m
- bleu : \u001b[44m
- magenta : \u001b[45m
- cyan : \u001b[46m

Les voitures **B** et **H** seront donc colorées de la même couleur (rouge), idem pour **C** et **I** (vert), etc. Dans le cas de notre exemple où il y a 8 voitures, voici la répartition des couleurs :

1. **A** : blanche
2. **B** : rouge
3. **C** : vert
4. **D** : jaune
5. **E** : bleu
6. **F** : magenta
7. **G** : cyan
8. **H** : rouge

Remarque concernant l'affichage des couleurs : l'affichage d'une chaîne de caractères avec une certaine couleur c en fond demande une syntaxe particulière. Pour afficher **Hello World !** avec la couleur rouge en fond, vous devez exécuter :

```
print("\u001b[41mHello World!\u001b[0m") → Hello World!
```

Vos chaînes de caractères doivent donc être précédées de `"\u001b[c m"` et suivies de `"\u001b[0m"` où c est le code de la couleur.

Remarque : certaines versions de Windows n'interprètent pas correctement les balises couleurs telles que décrites ci-dessus (utilisez Linux si c'est le cas).

1.3.3 Fichier de jeu `game_file`

Le fichier de jeu est un fichier texte qui décrit l'état initial du plateau de jeu. Il contient les informations suivantes :

- **Bordures du plateau** : Le plateau est entouré de bordures représentées par les caractères `+`, `-` et `|`. La première et la dernière ligne du fichier contiennent des `+` pour indiquer les coins et des `-` pour indiquer les bordures horizontales. Les lignes intermédiaires commencent et se terminent par `|` pour indiquer les bordures verticales.
- **Voitures et espaces vides** : À l'intérieur des bordures, chaque ligne représente une rangée du plateau de jeu. Les voitures sont représentées par des lettres alphabétiques (`A`, `B`, `C`, etc.) et les espaces vides sont représentés par des points (`.`). Chaque lettre représente une voiture unique, et les voitures peuvent être de différentes tailles et orientations (horizontale ou verticale).
- **Nombre de mouvements** : La dernière ligne du fichier contient un nombre entier qui représente le nombre maximum de mouvements autorisés pour résoudre le puzzle.

Listing 2 – Exemple de fichier de jeu

```
+-----+
|. .BCCC|
|. .B. .|
|AAB. . .
|D. .EEF|
|D. .GF|
|.HHHGF|
+-----+
40
```

Explication de l'exemple

- Les lignes `+----+` et `+----+` représentent les bordures supérieure et inférieure du plateau.

- Les lignes `|..BCCC|`, `|..B...|`, etc. représentent les rangées du plateau avec des voitures (A, B, C, D, E, F, G, H) et des espaces vides (`.`).
- La dernière ligne 40 indique que le joueur a un maximum de 40 mouvements pour résoudre le puzzle.

À noter que l'on supposera que les cartes données sont correctes. En particulier :

- au moins une voiture est toujours présente ;
- il y a un passage en face de la voiture A pour lui permettre de s'échapper (pour gagner le jeu) ;
- le jeu a bien une solution pour le nombre de mouvements donnés ;
- les voitures sont réparties soit horizontalement soit verticalement.

1.3.4 Partie de jeu game

L'objet `game` est un dictionnaire Python qui contient toutes les informations nécessaires pour représenter l'état actuel du jeu. Il est construit à partir du fichier de jeu et est utilisé pour gérer et manipuler le plateau de jeu et les voitures. Voici les principaux éléments contenus dans l'objet `game` :

- **Dimensions du Plateau :**
 - `width` : La largeur du plateau de jeu (nombre de colonnes).
 - `height` : La hauteur du plateau de jeu (nombre de rangées).
- **Voitures :**
 - `cars` (voir 1.3.2).
- **Nombre Maximum de Mouvements :**
 - `max_moves` : Un entier représentant le nombre maximum de mouvements autorisés pour résoudre le puzzle.

Exemple de Dictionnaire game Voici un exemple de ce à quoi pourrait ressembler l'objet `game` pour le fichier de jeu donné dans l'exemple de fichier précédent :

Listing 3 – Exemple de game suite à l'initialisation de la carte d'exemple

```
game = {
    'width': 6,
    'height': 6,
    'cars': [
        [(0, 2), 'h', 2], # Voiture A
        [(2, 0), 'v', 3], # Voiture B
        [(3, 0), 'h', 3], # Voiture C
        [(0, 3), 'v', 2], # Voiture D
        [(3, 3), 'h', 2], # Voiture E
        [(5, 3), 'v', 3], # Voiture F
        [(4, 4), 'v', 2], # Voiture G
        [(1, 5), 'h', 3] # Voiture H
    ],
    'max_moves': 40
}
```

}

Utilisation de `game` L'objet `game` est utilisé tout au long du programme pour :

- afficher le plateau de jeu avec les positions des voitures ;
- vérifier la validité des mouvements des voitures ;
- déplacer les voitures sur le plateau ;
- vérifier si le joueur a gagné ou perdu la partie.

En résumé, l'objet `game` centralise toutes les informations nécessaires pour gérer l'état du jeu et permet de manipuler facilement les éléments du jeu en fonction des actions du joueur.

1.4 Bibliothèques demandées

1.4.1 `sys`

```
from sys import argv


...
argv[1] # -> str
...
```

Description La variable `argv` est utilisée pour récupérer les arguments de la ligne de commande passés au script Python. Elle fait partie du module `sys` et contient la liste des arguments, où le premier élément est toujours le nom du script. Dans le contexte du projet, `argv` peut être utilisé pour spécifier le chemin du fichier de jeu à analyser.

1.4.2 `getkey`

```
from getkey import getkey

...
getkey() # -> str
...
```

Description La fonction `getkey` est utilisée pour capturer les entrées clavier de l'utilisateur de manière interactive. Son but est d'éviter d'utiliser `input()` qui force à attendre d'appuyer sur la touche  (enter) pour valider. Cela permet une interaction plus fluide et réactive dans les applications en ligne de commande, en particulier pour les jeux ou les interfaces interactives.








Dans le contexte du jeu de ce projet, `getkey` est utilisée pour capturer les choix de l'utilisateur concernant la sélection des voitures et des directions de déplacement.

Cette bibliothèque n'est pas standard dans Python, le fichier `getkey.py` vous sera fourni sur l'UV.

Paramètres (aucun)

Valeur de Retour `str` : chaîne de caractère de la touche pressée.

Voici quelques exemples et cas particulier :

	'a'
	'b'
	'UP'
	'RIGHT'
	'DOWN'
	'LEFT'
	'ESCAPE'

1.5 Fonctions demandées

Nous vous demandons d'implémenter les fonctions suivantes. **Cette consigne doit être obligatoirement respectée, sans quoi les tests automatiques ne passeront pas, et votre code ne sera pas corrigé.** Bien entendu, tant que toutes ces fonctions sont implémentées, vous pouvez tout à fait en ajouter d'autres qui vous permettraient d'écrire un code plus propre et lisible ; ceci est fortement encouragé.

1.5.1 Parsing du fichier

Fonction 1 : `parse_game(game_file_path: str) -> dict`

Description :

La fonction `parse_game` analyse un fichier de jeu et retourne un dictionnaire `game` (voir 1.3.4) représentant l'état initial du jeu de puzzle de voitures. Le chemin du fichier sera lu depuis la ligne de commande (voir 1.4.1).

— `game_file_path` : Chemin vers le fichier texte contenant la description du jeu.

Return :

dict : dictionnaire `game` (voir 1.3.4).

1.5.2 Affichage du jeu

Fonction 2 : `get_game_str(game: dict, current_move_number: int) -> str`

Description :

La fonction `get_game_str` renvoie le texte correspondant à l'affichage du plateau de jeu en utilisant les informations contenues dans le dictionnaire `game` (voir 1.3.4). Elle montre les positions actuelles des voitures sur le plateau et le nombre de mouvements effectués.

Note : Il ne faut **PAS** que la fonction affiche elle-même le jeu, mais renvoie un string qui lui sera imprimé. Exemple :

```
print(get_game_str(game, 25))
```

N'hésitez pas à prendre des libertés sur l'affichage, cette partie ne sera pas testée mais jugée si cohérente et originale (voir 3.3). N'oubliez cependant pas les critères essentiels (couleur des voitures, nombre de mouvements effectué, nombre de mouvements restant, etc.)

Paramètres

- `game` : Dictionnaire contenant l'état actuel du jeu de puzzle de voitures (voir 1.3.4).
- `current_move_number` : Nombre actuel de mouvements effectués.

Return :

str : Texte correspondant à l'affichage du plateau de jeu

1.5.3 Déplacer une voiture

Fonction 3 : `move_car(game: dict, car_index: int, direction: str) -> bool`

Description :

La fonction `move_car` permet de déplacer une voiture sur le plateau de jeu dans une direction spécifiée. Elle vérifie si le déplacement est valide et met à jour la position de la voiture en conséquence.

Paramètres

- `game` : Dictionnaire contenant l'état actuel du jeu de puzzle de voitures (voir 1.3.4).
- `car_index` : Indice de la voiture à déplacer dans la liste des voitures du dictionnaire `game`.
- `direction` : Chaîne de caractères représentant la direction du déplacement. Les valeurs attendues sont 'UP', 'DOWN', 'LEFT', 'RIGHT'.

Return :

bool : Un booléen indiquant si le déplacement a été effectué avec succès :

- **True** : Le déplacement a été effectué.
- **False** : Le déplacement n'a pas pu être effectué (par exemple en raison d'un obstacle, d'un mouvement qui ferait sortir de la grille, d'une direction reçue inattendue ou incohérente avec l'orientation de la voiture).

1.5.4 Vérification de la victoire

Fonction 4 : `is_win(game: dict) -> bool`

Description :

La fonction `is_win` vérifie si le joueur a gagné le jeu de puzzle de voitures. Elle détermine si la voiture principale (la première voiture de la liste) a atteint la position de sortie.

Paramètres

- `game` : Dictionnaire contenant l'état actuel du jeu de puzzle de voitures (voir 1.3.4).

Return :

bool : Un booléen indiquant si le joueur a gagné :

- **True** : Le joueur a gagné.
- **False** : Le joueur n'a pas encore gagné.

1.5.5 Démarrage/Boucle de la partie

Fonction 5 : `play_game(game: dict) -> int`

Description :

La fonction `play_game` permet de jouer au jeu de puzzle de voitures en utilisant les informations contenues dans le dictionnaire `game` (voir 1.3.4). Cette fonction doit essentiellement implémenter la structure globale du jeu telle que décrite dans la section 1.1 (sauf pour l'étape d'initialisation). Le choix de la voiture et de la direction doivent se faire avec la fonction `getkey` (voir 1.4.2).

Note 1 : Même si la touche `A` renvoie `'a'`, on s'attend que la voiture `A` soit sélectionnée. Autrement dit, il ne doit PAS être nécessaire d'appuyer sur `↑` (shift) pour avoir un `'A'` au lieu d'un `'a'`.

Note 2 : L'affichage de la partie doit se faire au moins une fois par (tentative de) mouvement ainsi qu'en fin de partie.

Paramètres

- `game` : Dictionnaire contenant l'état initial du jeu de puzzle de voitures (voir 1.3.4).

Return :

int : Un entier représentant le résultat du jeu :

- 0 : Le joueur a gagné.
- 1 : Le joueur a perdu (nombre de mouvements maximum dépassé).
- 2 : Le joueur a abandonné (touche `esc` appuyée pendant une phase de sélection).

1.5.6 `__main__`

Il est nécessaire que le jeu ne débute pas si votre code est importé (comme dans le cas des tests par exemple). Pour cela, il vous sera nécessaire d'utiliser :

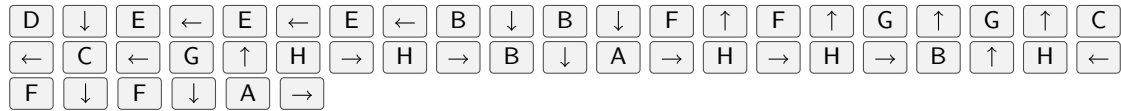
```
if __name__ == '__main__':  
    ...
```

Pour plus d'information : https://docs.python.org/3/library/__main__.html

2 Exemple de partie complète

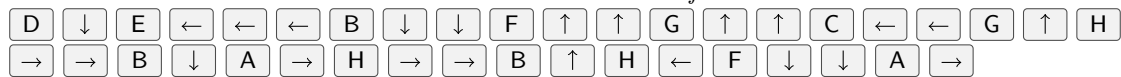
2.1 Séquence mode "voiture-mouvement"

Choix de la voiture suivi de son mouvement



2.2 Séquence mode "continu"

Choix de la voiture suivi de mouvements consécutifs



2.3 Schéma des mouvements



3 Critères d'évaluation

3.1 Tests automatiques

3.1.1 Fichiers de tests

Pour vérifier si vos projets correspondent aux exigences de l'énoncé, deux fichiers de tests sont mis à votre disposition sur l'UV :

- `test_ulbloque.py` : Tests basiques pour différentes fonctions obligatoires (voir 1.5). Tous les tests de ce fichier **DOIVENT** être réussis pour que votre projet soit corrigé.

- `test_ulblique_extra.py` : Tests plus avancés **facultatifs** qui testeront la réussite d'une partie pour une série de touches de claviers donnée. Ils ne sont pas obligatoires mais vous permettront de plus facilement vous assurer que votre projet fonctionne comme attendu.

Note : Les assistants en charge de la correction utiliseront ces mêmes tests sur vos projets ainsi que des tests supplémentaires similaires non-fournis. Autrement dit, nous utiliserons des valeurs différentes pour tester vos fonctions afin de vérifier si leur logique est cohérente (et que les valeurs de retours ne sont pas hard-codés dans votre solution).

3.1.2 Exécution des tests

Vous pouvez les exécuter comme des scripts python classiques :

```
$ python3 test_ulblique.py
```

Si les tests sont un succès, vous aurez un résultat similaire au suivant :

```
...
-----
Ran 6 tests in 0.010s

OK
```

Si ça n'est pas le cas, vous aurez les détails sur les problèmes rencontrés et le nombre de tests qui ont échoué.

pytest Si vous le souhaitez, vous pouvez utiliser **pytest** pour lancer les fichiers de tests. L'interface est plus visuelle et facile à comprendre qu'en exécutant les scripts directement.

Une fois la librairie installée, vous pouvez lancer un terminal dans le dossier avec les tests puis exécuter :

```
$ pytest
```

3.2 Respect des consignes

Il va sans dire qu'un non-respect des consignes fera immédiatement perdre un nombre significatif de points.

3.3 Affichage

L'affichage du jeu se fera en terminal avec un fond sombre (pour que la voiture A soit visible facilement). Vous n'avez pas d'obligation par rapport au dessin des contours ou à comment/où placer le nombre de mouvements restants. Ce projet est un jeu-vidéo, laissez libre cours à votre imagination tant que les informations nécessaires sont présentes, cohérente et "bien-pensées".

3.4 Code non-autorisé

- `input()` : Même pour des raisons d'affichage/présentation/introduction. Nous vous demandons de ne PAS l'utiliser pour éviter que les tests automatiques échouent
- *import de librairies* : Dans le cadre de ce projet, seul `getkey` (voir 1.4.2) et `sys` (voir 1.4.1) auront besoin d'être importés

3.5 Résilience

Comme tout bon jeu-vidéo qui se respecte, il est important de prendre en compte les différentes erreurs potentielles et d'en informer efficacement le joueur. Vous devez donc penser à gérer les différents crashes que le joueur pourrait causer. Par exemple : mouvements incohérents, sélection de voiture non-existantes, etc.

Certaines erreurs peuvent toutefois causer un "crash" qui ne peut pas être géré facilement comme avec le parsing du fichier (voir 1.5.1) si le chemin est incorrect ou que l'utilisateur oublie de préciser le chemin en paramètre par exemple. Il est dans ce cas préférable d'informer le joueur de son erreur tout en permettant au jeu de crasher avec :

```
exit(1)
```

3.6 Altération

Malgré le fait que `game` soit un dictionnaire, on vous demande de ne PAS l'altérer en dehors de ce qui est indiqué dans les consignes. Par exemple, il est interdit d'y ajouter un nouvel élément en cours de partie.

3.7 Propreté du code

Une part non-négligeable des points seront attribués pour la propreté du code. Nous évaluerons à la fois la structure globale (découpe en fonctions supplémentaires pertinentes, etc.) mais aussi l'esthétique (choix du nom des variables/fonctions, commentaires pertinents, "Don't repeat yourself", respect des conventions PEP8)

4 Consignes de remise et autres consignes

Compatibilité : Votre projet doit pouvoir fonctionner avec Python 3.12 sous Linux (ex : Ubuntu). Même si les incompatibilités de code python entre systèmes d'exploitation sont de plus en plus rares, il est de votre responsabilité de vérifier que votre code fonctionne bien sous Linux si vous utilisez Windows ou Mac OS. Vous pouvez par exemple utiliser les machines du NO (ou encore mieux : installer Linux;))

Réussite des tests automatiques : Votre code doit réussir les 6 tests fournis, sans quoi nous ne corrigerons pas votre projet, et votre note finale sera de 0.

En-tête : Au tout début de votre fichier `ulblique.py`, vous devez mettre en commentaire vos nom, prénom, et matricule.

Remise : Votre fichier `ulblique.py` (et uniquement ce fichier, il ne faut pas joindre `getkey.py` ni les fichiers `game`) est à remettre sur l'UV dans la section correspondante, avant le **dimanche 15/12 à 22h**. Attention à bien respecter le nom du fichier : `ulblique.py`, tout en minuscule. (Le fichier doit être soumis tel quel, ne faites pas de `.zip` ni de dossier).

Retards : Une remise en retard est techniquement possible sur l'UV (maximum 24h) mais fortement découragée : celle-ci sera sanctionnée d'une pénalité de 10 points sur 40. La pénalité est d'application même si remise 2s en retard, n'attendez donc pas le dernier moment pour soumettre votre projet sur l'UV! (Inutile de nous envoyer des emails expliquant les raisons d'une remise en retard).

Non-respect des consignes de remise : Une pénalité de 10 points sur 40 sera appliquée si le nom de fichier est incorrect, en-tête manquante, ou si une autre consigne de remise n'a pas été respectée.

Non-respect des signatures des fonctions : Il est très important de respecter les noms de fonctions et leurs paramètres, sinon les tests automatiques ne pourront s'exécuter. Une erreur à ce niveau-ci sera sanctionnée d'une note nulle pour le projet 1.