Conduite de projet
Exemple - Course_puces

C'est un exemple simple qui peut aide lors des premiers projets en première.
Le document ne constitue par un modèle, c'est un exemple qui illustre les différentes étapes de conception.

Table des matières

1 Spécification....1
1.1 Expression du besoin....1
1.2 Le cahier des charges....1
1.3 Fiche de validation....2
2 Représentation graphique du projet....2
2.1 Structure des données....2
3 Algorigramme (facultatif)....4
4 L'algorithme général (obligatoire)....5
5 Définition des interfaces (signatures)....6
6 Conception détaillée....7
7 Réalisation....7
8 Test unitaire....7
9 Intégration....8
10 Validation....8
11 Annexe 1 : le planning....9
12 Annexe 2 : ressources....9
13 Annexe 3 : documentations....9

1 Spécification

1.1 Expression du besoin

Le projet permet de répondre au besoin de simuler et d'observer des sauts de puces.
On ne cherche pas à simuler de vrais sauts, mais simplement de visualiser des déplacements dans des cases.

1.2 Le cahier des charges

Les puces se déplacent de cases en cases :

- de manière aléatoire ;
- avec des sauts compris entre 1 et 3.

Chaque puce se déplace de manière linéaire.
Les déplacements des puces sont parallèles.
Les différentes puces sont placées sur une grille de largeur 10 et de hauteur, le nombre de puces.
Au début de la simulation, toutes les puces sont placées sur les cases numéro 0 à gauche
La simulation prend fin lorsqu'une puce dépasse la position 10.

L'interface IHM :

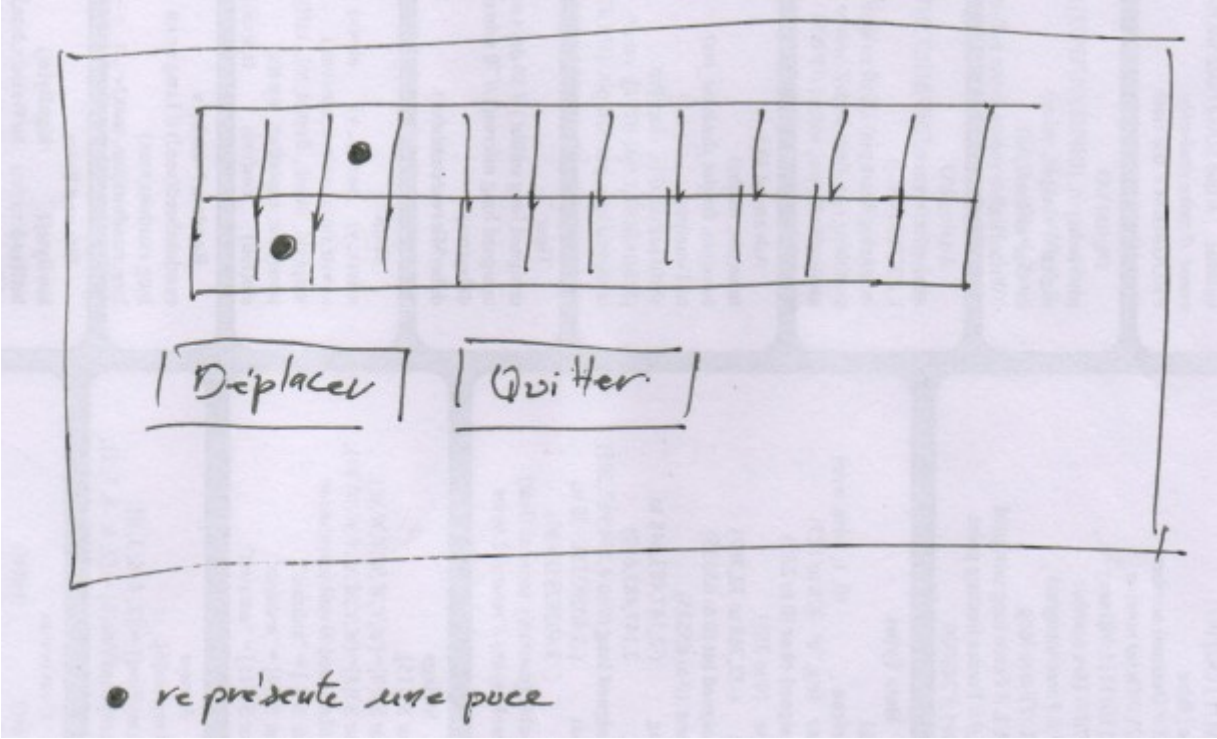
- dans un premier temps en mode console ;
- puis éventuellement avec une interface graphique.

1.3 Fiche de validation

Nous allons vérifier 3 points pour valider le fonctionnement.

- Au début de la simulation les puces sont en position initiale à gauche.
- Les déplacements sont compris entre 1 et 3.
- La simulation se termine dès qu'une puce dépasse la position 10.

2 Représentation graphique du projet

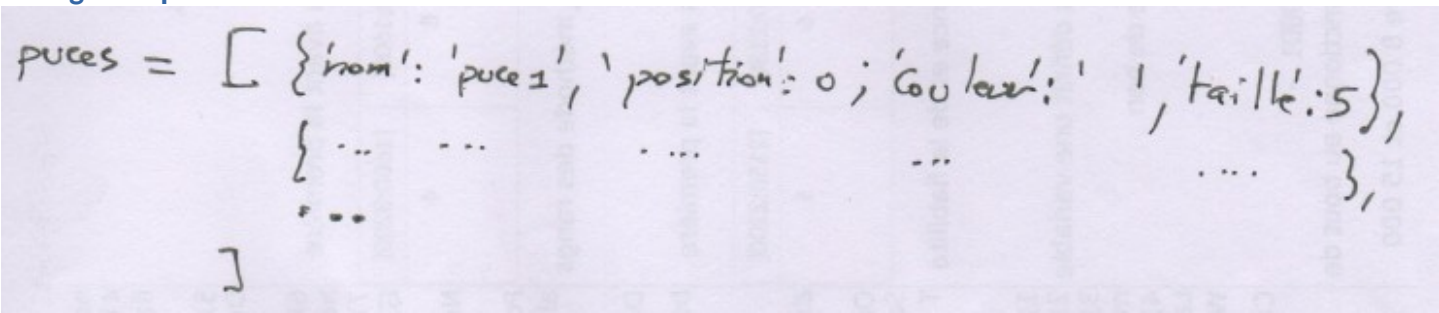


Dans la console :

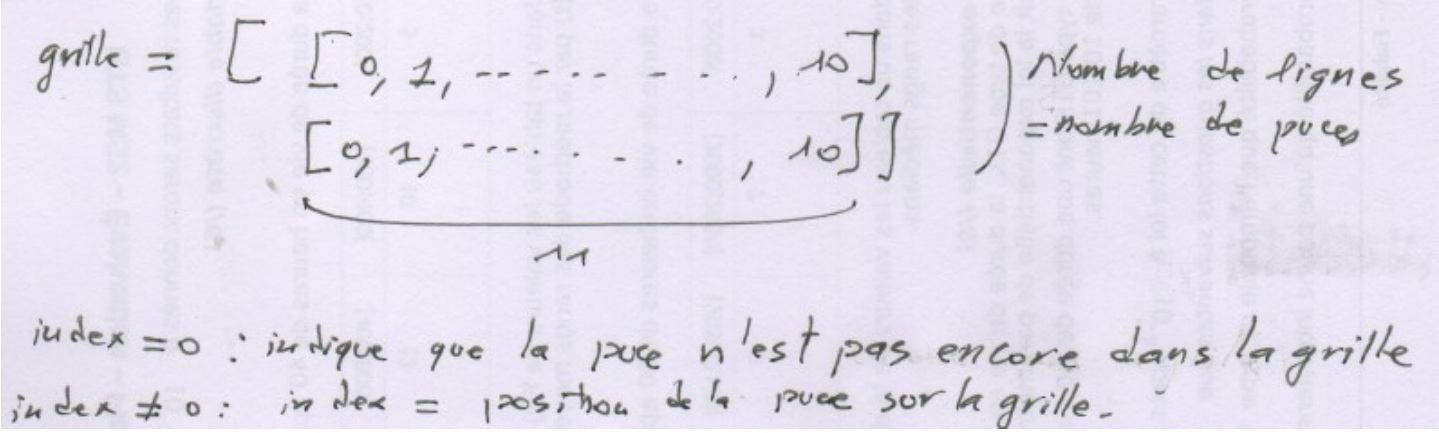
- Le **bouton Déplacer** sera remplacé par l'action sur la **touche D**.
- Le **bouton Quitter** sera remplacé par l'action sur la **touche Q**.

2.1 Structure des données

Codage des puces :



Codage de la grille:



$index = 0$ correspond à la position initiale

$index \neq 0$, index correspond à la position de la puce sur la grille

Finalement la structure initiale a évolué comme suit :
- 5 puces => 5 lignes
- 10 cases de déplacements => 13 colonnes
    - 1 cases init
    - 10 cases de déplacement
    - si puce en case 9
        - saut de 1 arrivée en 10
        - saut de 2 arrivée en 11
        - saut de 3 arrivée en 12

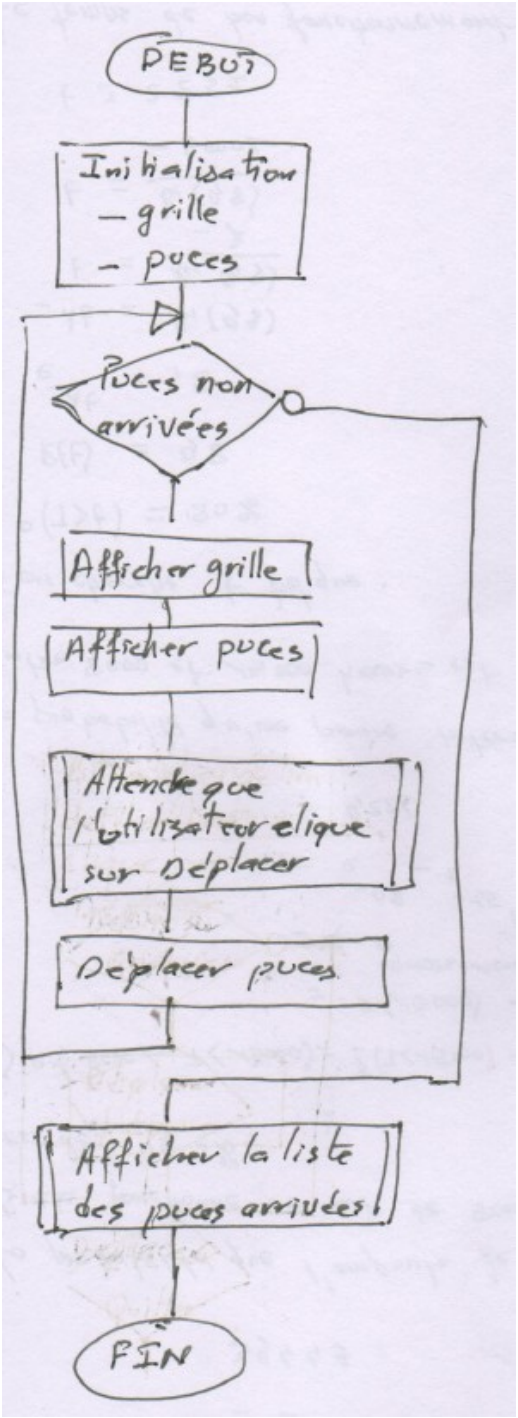
```
grille =    [[0, 0, 0, 1, 0, 0, 0, 0, 0, 0, 0, 0],
             [0, 0, 0, 0, 2, 0, 0, 0, 0, 0, 0, 0],
             [0, 0, 3, 0, 0, 0, 0, 0, 0, 0, 0, 0],
             [0, 0, 0, 0, 0, 4, 0, 0, 0, 0, 0, 0],
             [0, 0, 0, 0, 0, 0, 0, 5, 0, 0, 0, 0]]
```

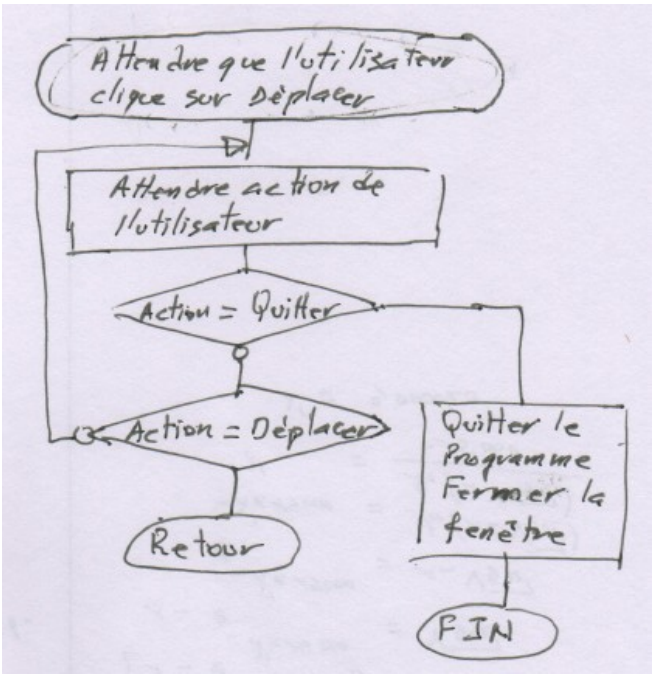
Exemple de représentation dans la console :

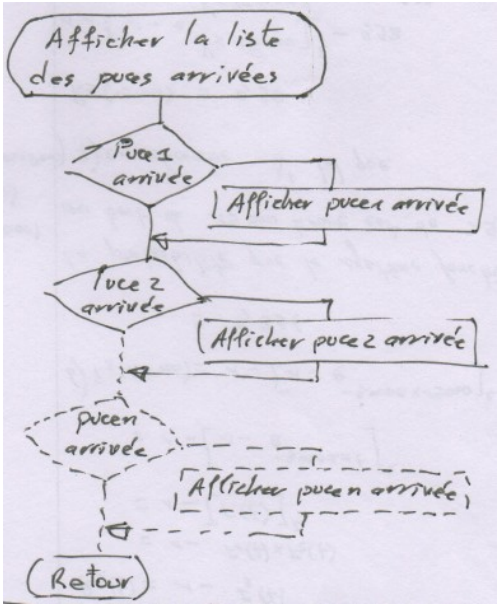
```
-------------------------------------------------
|   |   |   | 1 |   |   |   |   |   | 1 |   |   |
-------------------------------------------------
|   |   |   |   | 2 |   |   |   |   |   |   |   |
-------------------------------------------------
|   |   | 3 |   |   |   |   |   |   |   |   |   |
-------------------------------------------------
|   |   |   |   |   | 4 |   |   |   |   |   |   |
-------------------------------------------------
|   |   |   |   |   |   |   | 5 |   |   |   |   |
-------------------------------------------------
```

3 Algorigramme (facultatif)

Avant de produire l'algorithme général, on peut décrire le fonctionnement en fonction du temps à l'aide d'un algorigramme. Néanmoins, on s'aperçoit avec l'expérience que cette étape n'est pas utile.

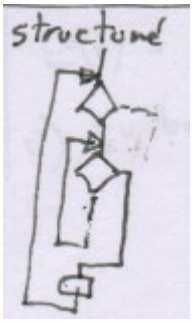


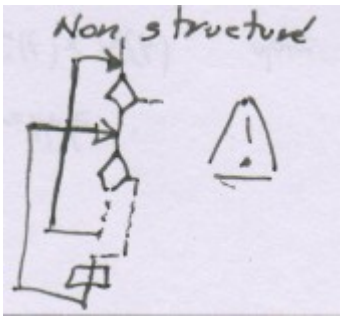




Remarque :

- Les algorithmes produisent des programmes bien structurés :
  - Bonne imbrication des boucles …
- Ce n'est pas toujours le cas pour les algorigrammes
  - Sur la figure de droite on observe une mauvaise imbrication des boucles …



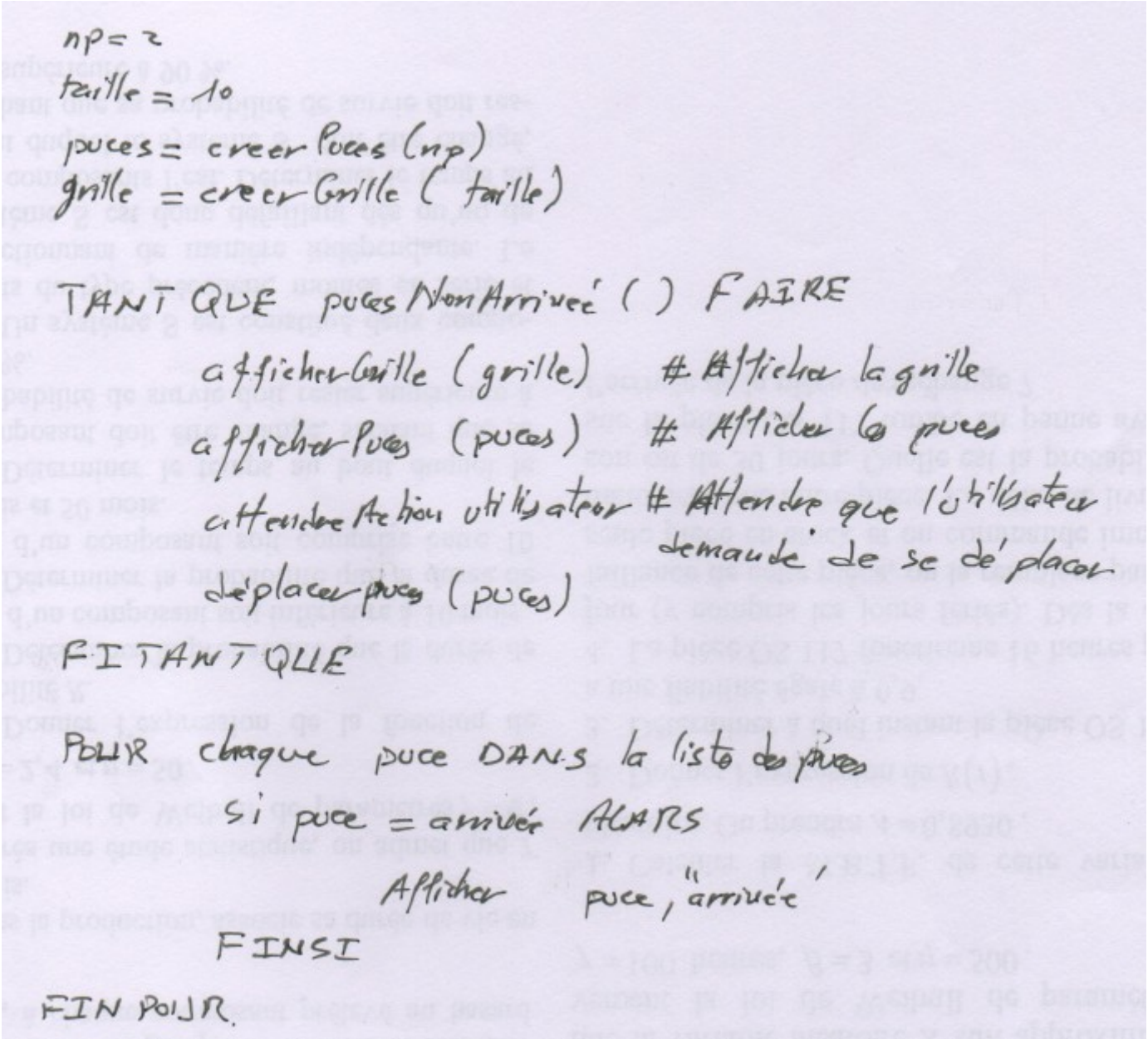


4 L'algorithme général (obligatoire)

Contrairement à l'algorigramme, l'algorithme général est obligatoire.
Il permet de :

- décrire le fonctionnement général ;
- répartir les fonctions à réaliser par chaque membre du groupe.

L'algorithme général est aussi la base pour construire le script principal du projet.



5 Définition des interfaces (signatures)

```
def creer_puces(np:int) -> list:
    """
    Créer un ensemble de puces sous la forme d'un tableau de dictionnaires
      np : (int) nombre de puces
    renvoi
      puces : (list) tableau de dictionnaires, un dictionnaire par puce
    """
```

```
def creer_grille(taille:int, puces:list) -> list:
    """
    Créer la grille sur laquelle sont placées les puces
        taille : (int) longueur du parcours de la course
        puces : (list) tableau de dictionnaires, un dictionnaire par puce
    renvoi :
      grille : (list) tableau 2 dimensions,
        - longueur correspond à la longueur du parcours ;
        - hauteur correspond au nombre de puces
    """
```

```
def afficher_grille_puces(grille:list):
    """
    Afficher dans la console, la grille avec les puces
      grille : (list) tableau 2 dimensions
    Sortie :
      Affichage de la grille dans la console
    """
```

```
def deplacer_puces(grille:list, puces:list) -> tuple:
    """
    Déplacer les puces aléatoirement.
    Chaque déplacement est une valeur aléatoire comprise entre 1 et 3
      grille : (list) tableau 2 dimensions
      puces : (list) tableau de dictionnaires, un dictionnaire par puce
    Renvoi :
      grille, puces : (tuples)  les tableaux grille et puces
    """
```

```
def puces_arrivees(grille:list, puces:list) -> bool:
    """
    Vérifier qu'une ou plusieurs puces sont arrivées
      puces : (list) tableau de dictionnaires, un dictionnaire par puce
    Renvoi :
      True/False : (bool) True si au moins une puce est arrivée, False
sinon
    """
```

```
def attendre_action_utilisateur():
    """
    Attendre une action de l'utilisateur sur le clavier
      Si touche Q appuyée alors quitter le programme
      Si touche D (ou autre touche) appuyée alors le déplacement des puces
se poursuit
    Renvoi :
      Pas de renvoi
    """
```

```
def afficher_puces_arrivees(puces:list):
    """
    Lorsqu'au moins une puce est arrivée
      Afficher les puces arrivées
      Afficher les puces non arrivées
    Renvoi :
      Pas de renvoi
    """
```

6 Conception détaillée

Le projet est organisé en 2 scripts :

- Le main : course_puces_console.py
- Un module : courses_puces.py

Dans le cas d'un travail en groupe, Il est souhaitable que chaque élève réalise son propre module.
Le script principal est à réaliser en groupe.

Dans la conception détaillée il faut :

- écrire des algorithmes avant le code (python ou autre langage) ;
- décomposer des fonctions en sous fonctions ;
- traduire un concept mathématique en algorithme.

7 Réalisation

Dans notre exemple :

- Les fonctions sont rassemblées dans le script : courses_puces.py
- Les appels de tests des fonctions sont dans le script : tests_fonctions.py
- Le programme principal correspond au script : course_puces_console.py

On peut donc consulter ces fichier pour observer la réalisation de chaque fonction.

8 Test unitaire

- Test des données
  - Dans le fichier course_puces.py on peut voir que chaque fonction commence par des assertions de vérification les arguments.
  - On peut aussi ajouter des assertions avant le renvoi pour vérifier le résultat de l'algorithme.
- Test du code
  - Le fichier tests_fonctions.py permet de lancer individuellement chaque fonction pour vérifier son fonctionnement.
  - Ce fichier peut aussi contenir des assertions pour vérifier ce qui est renvoyé par les fonctions.

9 Intégration

À présent toutes les fonctions ont été validées une à une.
On les intègre une à une dans le script principal.

Intégration dans notre cas :

- script principal : course_puces_console.py
- on commence par intégrer les modules de chaque élève
  - Ici un seul module : import courses_puces
- en suite le code est la traduction de l'algorithme général

Facultatif : pour lancer le script dans la console Linux

- placer le code de l'algorithme général dans une fonction main()
- ajouter le code ci-dessous à la fin du script

```
if __name__ == "__main__":
    main()
```

- écrire en première ligne du script

```
#!/usr/bin/env python3
# -*- coding: utf-8 -*-
```

10 Validation

On réalise quelques simulations pour vérifier :

- la validité de l'état initial ;
- la validité de l'évolution ;
- la validité de l'état final.

Facultatif : pour vérifier le script dans la console Linux

Exemple 1 de commande :

```
nsi@nsi-portable-01:~$ python3 course_puces_console.py
```

Exemple 2 de commande

```
nsi@nsi-portable-01:~$ chmod u+x course_puces_console.py
nsi@nsi-portable-01:~$ ./course_puces_console.py
```

11 Annexe 1 : le planning

Date	Travail / Recherche
??/ ??/ ??	• Analyse de l'existant
??/ ??/ ??	• Spécifications du projet • Rédaction du cahier des charges
...	
??/ ??/ ??	• Conception générale • Rédaction d'un document commun
...	
??/ ??/ ??	• Conception détaillée de ... • Test de ...
...	
??/ ??/ ??	• Intégration • phase ... • validation ...
...	
??/ ??/ ??	• Finalisation des documents à rendre
??/ ??/ ??	• Préparer la présentation orale

12 Annexe 2 : ressources

Liens vers mes ressources :

- Liens vers des sites avec des algorithmes et des codes ;
- Lien vers des outils en ligne.

13 Annexe 3 : documentations

Dans un projet, on peut utiliser des outils particuliers : du matériel, un module ou site ...

On place alors dans cet annexe une documentation succincte pour :

- justifier les choix algorithmiques ;
- comprendre les scripts du projet.

Exemple de documentation possible à fournir dans ce projet :

- Interface Graphique PySimpleGUI ;
- Carte Micro:Bit ;
- Site vittascience ;
- ...