Dans son répertoire personnel (partage réseau `U`, cliquer sur Ordinateur pour le faire apparaître), créer un répertoire `MPS`. Dans ce répertoire créer un répertoire `Rurple` et dans ce répertoire créer deux répertoires `scenes` et `scripts`.

1 Présentation de l'environnement

Lancer le logiciel Rurple de puis le partage réseau `T`. Ce logiciel permet de déplacer un robot nommée Rurple sur une scène en utilisant le langage de programmation PYTHON.
Le robot Rurple « obéit » aux ordres suivants :

- « `avance( )` » (ou appui sur touche A) : le robot avance d'une case devant lui.
- « `gauche( )` » (ou appui sur touche G) : le robot effectue un quart de tour vers la gauche.
- « `depose( )` » (ou appui sur touche D) : le robot dépose une bille sur la case où il se trouve.
- « `prends( )` » (ou appui sur touche P) : le robot ramasse une bille sur la case où il se trouve.

Il comprend également le langage PYTHON dont on découvrira quelques instructions.

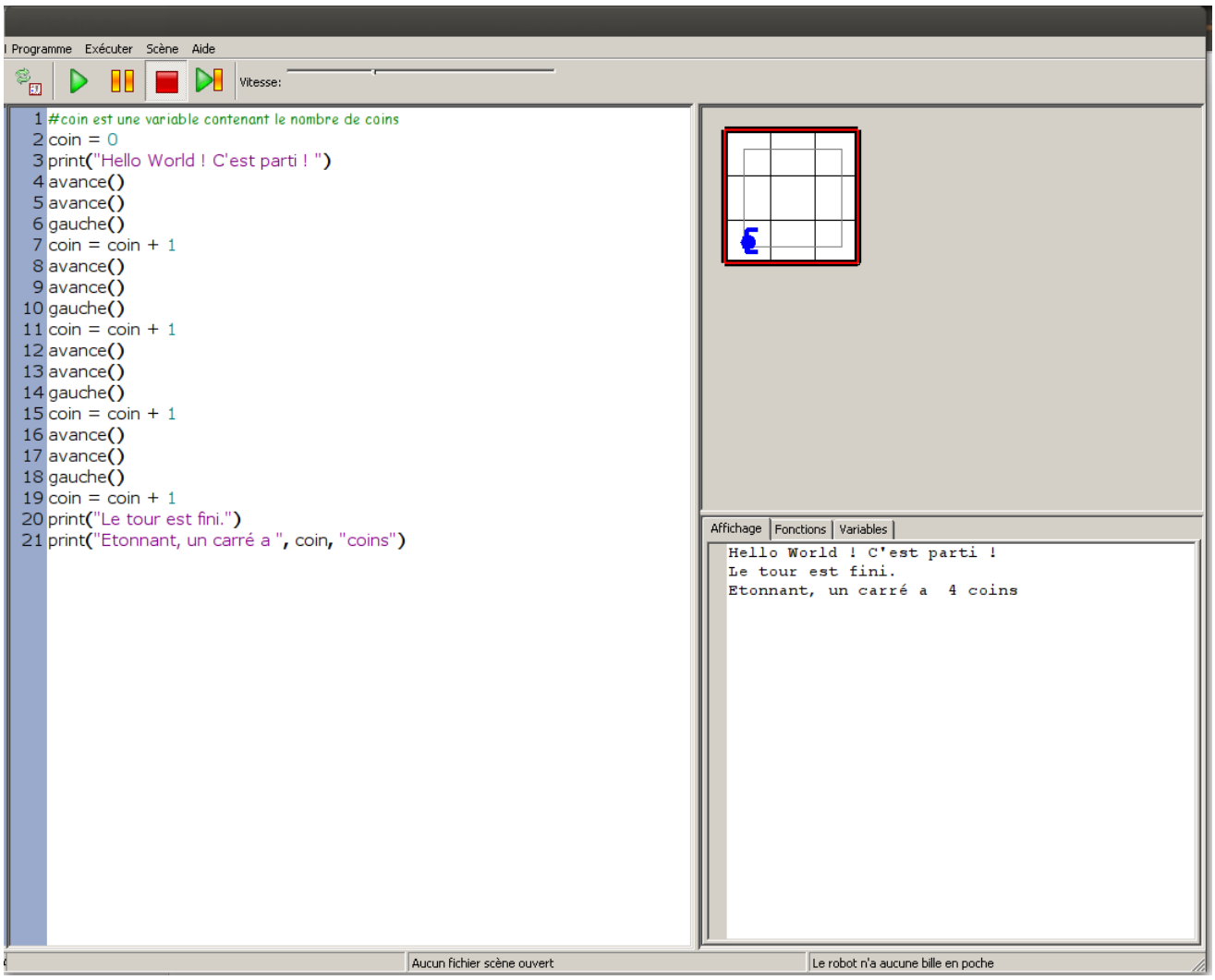
1. Faire une recherche documentaire : citer au moins quatre langages de programmation. Quelle est la différence entre un langage compilé et un langage interprété ?

2. Quel est le site internet officiel de Python. C'est un logiciel libre, qu'est-ce que celà signifie ?

L'interface graphique est composé d'une barre de menu classique en haut et de trois fenêtres :

- La fenêtre de gauche accueille le programme (ou script) des *instructions* que l'on souhaite faire exécuter au robot. Celui-ci lit le programme de haut en base et exécute les instruc*tions de façon séquentielle. Certaines instructions comme les conditions ou les boucles* permettent de faire des sauts en avant ou en arrière dans le *code source* du programme. On peut enregistrer/ouvrir/créer un nouveau un programme/script en sélectionnant le menu Programme. Il faut enregistrer un nouveau programme avant de l'exécuter.
- La fenêtre en haut à droite contient la grille d'évolution du robot, on peut régler sa taille à partir du menu Scène et il est possible d'insérer des murs ou des billes dans la grille en cliquant dessus avec le bouton gauche.
- La fenêtre en bas à droite contient trois volets Affichage/Fonctions/Variable. Dans Affichage seront écrites les messages du robot (c'est le programmeur qui le fait parler ...). Dans Fonctions se trouvent la liste des fonctions spécifiques à Rurple en plus des fonctions classiques de Python. Dans Variables on pourra observer l'évolution du contenu des variables.

En dessous de la barre de menu se trouve un barre d'outils avec trois icônes cliquables pour contrôler l'exécution d'un programme (lecture puis interprétation par le robot) selon trois fonctionnalités : Exécuter, Suspendre, Stopper, Exécuter pas à pas (fait une pause après chaque ligne/instruction).



2 Premier programme

1. Créer une nouvelle scène de dimensions 3 × 3 et l'enregistrer sous le nom `sceneinitiale.wld` dans le répertoire `scenes` créé précédemment.

2. On veut que le robot Rurple fasse un tour de la scène. On peut faire un premier tour en le dirigeant avec les touches du clavier A ou G mais on veut désormais créer une suit d'instructions, un programme qui dicte au robot les instructions qui lui permettront de faire un tour complet.

   Créer un nouveau programme puis l'enregistrer dans le répertoire `scripts` créé précédemment sous le nom `hellowolrd.py`.

3. Saisir dans la fenêtre d'édition le programme de la capture d'écran précédente. Exécutez ensuite deux fois le programme en mode pas à pas en sélectionnant le volet Affichage puis le volet Variables.

4. Quel reproche peut-on faire au code source de ce programme ?

Pour réinitialiser la scène, on peut cliquer sur le bouton correspondant de la barre d'outils ou effectuer la combinaison de touches sur `CTRL + R`.

- On commence chaque instruction en début de ligne puis on va à la ligne pour passer à la suivante.

- Une ligne commençant par un # n'est pas une instruction mais un commentaire, elle n'est pas lue par le robot mais elle sert à l'auteur du programme pour expliquer le sens d'une instruction (un programme peut être réutilisé par d'autres)

- `coin` est une variable c'est-à-dire une zone de la mémoire du robot [1] repéré par un nom dans laquelle on va ranger une valeur qui peut changer au cours du programme. On lui *affecte* la valeur 0 avec l'instruction `coin = 0`. L'effet de l'instruction `coin = coin + 1` est encore plus évident lorsqu'on exécute en mode pas à pas avec le volet Variables.

- Les instructions qui se terminent par des parenthèses comme `avance()` ou `print()` sont des fonctions du langage : elles modifient l'état du robot ou de son environnement (scène, fenêtre d'affichage). `avance()` est une `procédure` car elle ne prend pas d'argument. `print("Hello Wolrd")` prend comme argument une chaîen de caractères (entre double guillemets) et la retourne vers la fenêtre d'affichage.

3 Structure conditionnelle

1. Créer une nouvelle scène de dimensions 2 × 1 avec deux billes sur la deuxième case , créer un nouveau programme (`ifelse.py`) et saisir le code ci-dessous. Remarquer les deux points qui terminent la ligne du `if` et le décalage vers la droite de l'instruction après le `if`, on parle d'indentation. On l'obtient avec la touche tabulation.

```
avance()
if bille_au_sol():
    print("Chouette mes billes !")
else:
    print("Où sont passées mes billes ?")
```

2. Exécuter le programme avec des billes ou pas de billes sur le deuxième case. Que se passe-t-il ?

3. Ecrire un programme similaire qui demande au robot de prendre une bille s'il y a au moins une bille sur la deuxième case ou de ne pas en prendre s'il n'y en a pas.

4. Créer un nouveau programme, saisir le code ci-dessous, exécuter, que se passe-t-il ?

```
if lance_le_de() == 6:
    print("Gagné")
else:
    print("perdu")
```

Modifier le programme pour que le robot affiche "Gagné" si le résultat du lancer de dé est supérieur à 3 puis si le résultat est pair. En PYTHON le reste de la division euclidienne de l'entier `a` par `b` est obtenu par l'opérateur % avec `a%b`.

5. Créer un programme avec trois variables `a`, `b` et `c` contenant des entiers au hasard entre 1 et 6 et qui permette au robot d'afficher les trois entiers dans l'ordre décroissant.
On pourra s'aider de l'écriture de toutes les configurations possibles à l'aide d'un arbre.
Tester le programme une dizaine de fois pour voir s'il fonctionne.

```
a = lance_le_de()
b = lance_le_de()
c = lance_le_de()
```

1. Robot ou ordinateur ou machine ou golem appelez-le comme vous voulez

4 Boucles

4.1 Boucle Pour, boucle définie

1. Créer une une nouvelle scène de dimensions 8×1, créer un nouveau programme (`bouclepour.py`) et saisir le code ci-contre.

   Comme pour le `if`, la ligne du `for` se termine par deux points et le bloc d'instruction qu'il commande (appelé corps de la boucle) est indenté.

   Comment aurait-on pu obtenir le même comportement du robot en utilisant seulement l'instruction `avance()` ? Quel est l'avantage de la structure de boucle `for` (ou Pour en français) ?

```
for k in range(1, 8):
    avance()
```

2. Observer l'évolution de la variable de boucle `k` en exécutant le programme pas à pas.

3. Remplacer `for k in range(1, 8):` par `for k in range(0, 7):` puis par `for k in range(7):` . Est-ce que le comportement change ? Et si on remplace par `for k in range(1, 7):` ?

   Que se passe-t-il si on change la largeur de la scène ? Tester pour une largeur inférieur puis supérieure à 8

.

4.2 Boucle Tant Que, boucle indéfinie

1. Créer un nouveau programme (`boucletantque.py`), saisir le code 1 ci-contre avec une boucle `while` pour laquelle on retrouve les deux points et l'indentation. Exécuter le programme avec une scène rectangulaire 8×1 comme pour la boucle `for` ci-dessus.

2. Observer l'évolution de la variable de boucle `k` en exécutant le programme pas à pas.

3. Que se passe-t-il si on change la largeur de la scène ? Tester pour une largeur inférieure puis supérieure à 8.

4. Créer un nouveau programme (`boucletantque2.py`) avec le code 2 ci-contre. Tester ce programme. Quel avantage présente-t-il ?

5. Si on remplace l'instruction `avance()` par `print('Bonjour')` que se passe-t-il ?

```
#code tant que 1
k = 0
while k < 7 :
    k = k + 1
    avance()
```

```
#code tant que 2
while not mur_devant():
    avance()
```

.

5 Exercices

Exercice 1 *Première cueillette*

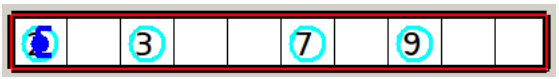
1. Créer un nouveau programme `cueillettepour.py`, saisir le code ci-dessous et l'exécuter sur une scène comme celle ci-dessous de dimensions 9×1.
2. Modifier le programme pour que le robot affiche à la fin le nombre de billes/fraises qu'il a cueillies. Interdit de l'aider en lui soufflant la réponse, il faut qu'il compte!!!
3. Modifier encore le programme pour que le robot puisse ramasser une fraise sur sa case de départ.
4. Proposer plusieurs scènes de dimensions 8×1 pour laquelle ce programme ne permettra pas au robot de ramasser toutes les fraises.

```
for k in range(8):
    avance()
    if bille_au_sol():
        prends()
```

Exercice 2 *Seconde cueillette*

1. En utilisant une boucle Tant Que, créer un nouveau programme `cueillettetantque.py` qui permet au robot de cueillir toutes les fraises déposées sur une scène rectangulaire de dimensions $n \times 1$ avec au plus 1 fraise par case.
2. En utilisant une boucle Tant Que, créer un nouveau programme `cueillettetantque1.py` qui permet au robot d'avancer puis de cueillir toutes les fraises déposées sur la deuxième case comme ci-dessous où il y a 3 fraises.

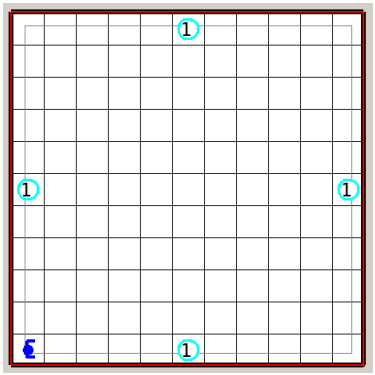
3. En utilisant une boucle Tant Que, créer un nouveau programme `cueillettetantque2.py` qui permet au robot d'avancer puis de cueillir toutes les fraises déposées sur les cases d'une scène rectangulaire de dimensions $n \times 1$ avec éventuellement plus d'une fraise par case.



Exercice 3

Tour de scène

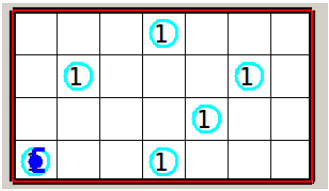
1. En utilisant deux boucles `for` imbriquées, écrire un programme de quatre lignes qui permet au robot de faire le tour d'une scène de dimensions 4×4 et de revenir à sa position de départ comme dans l'exemple initial.
2. Comment modifier le programme précédent pour que le robot puisse faire le tour de n'importe quelle scène rectangulaire de dimensions $n \times m$ avec $n > 1$ et $m > 1$?
3. Dans le menu scène, sélectionner Nombre de billes et placer quatre billes dans la poche du robot.
   Modifier le programme précédent pour que le robot fasse un tour de la scène et dépose une bille à chaque coin de n'importe quelle scène rectangulaire.
4. Choisir une scène rectangulaire de dimensions 11×11. Modifier le programme précédent pour que le robot fasse un tour de la scène et dépose une bille au milieu de chaque côté.



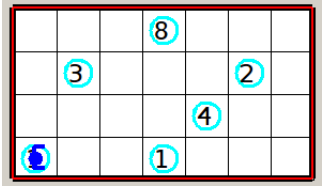
Exercice 4

Troisième cueillette

1. Quelle suite d'instructions permet au robot de faire demi-tour ? de tourner à droite ?
2. Créer un nouveau programme qui permette au robot de cueillir toutes les billes/fraises déposées sur une grille rectangulaire de dimensions 7×4 avec au plus une fraise sur chaque case (y compris sur la case de départ). Le programme doit fonctionner quelle que soit la disposition des fraises.
   A la fin de la cueillette, le robot doit afficher le nombre de fraises cueillies.



3. Modifier le programme précédent pour qu'il permette au robot de cueillir toutes les billes/fraises déposées sur n'importe quelle grille rectangulaire de dimensions 7×4 avec éventuellement plus d'une fraise sur chaque case (y compris sur la case de départ).

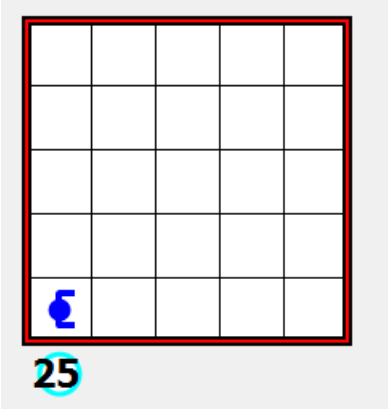


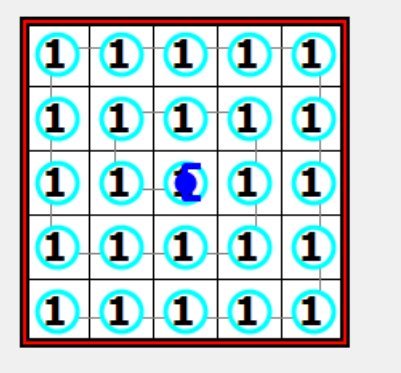
Exercice 5

Ecrire un programme qui permet à `Rurple` d'effectuer n tours (valeur fixée par l'utilisateur) d'une scène carré de dimensions $m \times m$ où m est entier positif.

Exercice 6

Ecrire un programme qui permet à `Rurple` de déposer une bille sur chacune des cases d'une scène de dimensions 5×5, en parcourant la scène en spirale depuis le coin inférieur gauche (Voir figure 2 ci-dessous).
On commencera par placer 25 billes dans la poche de `Rurple` depuis le menu `Scène/Nombre de billes`.





Exercice 7

Ecrire un programme qui permet à `Rurple` de faire le tour d'une pièce (avec des angles de 90° en déposant d'abord une bille sur la case de départ jusqu'au retour à la case de départ où la bille a été déposée.