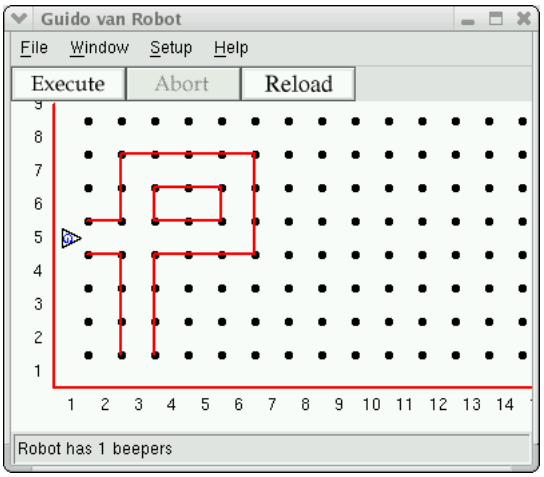
TD 1 – Une introduction en douceur à l'algorithmique avec Guido

1) Présentation de Guido

"Guido Van Robot" est un environnement de programmation d'un petit robot utilisant un micro-langage, en fait un sous-ensemble minimum de Python. Bien que ce langage soit extrêmement rudimentaire, il est très pédagogique.

Guido est un robot évoluant dans un monde en deux dimensions, constitué de rues et d'avenues. Voici un exemple de situation de départ :




```
define untour:
    do 3:
        move
    turnleft
    do 2:
        move
    turnleft
    do 3:
        move
    turnleft
    do 2:
        move

do 2:
    move
putbeeper
untour
pickbeeper
do 3:
    move
turnoff
```

Guido est matérialisé par une triangle dont l'orientation indique dans quelle direction il va avancer. Il se déplace de case en case, les coins des cases étant matérialisés par les points noirs. Les traits rouges sont des murs, que le Robot ne peut traverser. La barre de statut indique que le robot transporte un "beeper", une sonnette en français. Il peut à loisir prendre des sonnettes présentes sur le terrain ou en déposer s'il en a dans son *sac à sonnettes*.

Pour diriger Guido, on écrit un *code*, en utilisant les commandes connues par Guido, ou celles que l'on définit soi-même. Guido peut ainsi avancer, tourner sur lui-même, tester son environnement (présence ou absence de murs, de sonnettes...). On peut lire ci-dessus un exemple d'un tel code. Le plus simple est que vous constatiez de vous-même ce que ce programme fait effectuer au robot.

Les 12 premières lignes ne sont pas exécutées, elles constituent une *définition de fonction* permettant de découper la tâche à effectuer en sous-tâches plus simples à appréhender. La première commande exécutée par le robot est une *boucle* `do 2`, qui lui demande de répéter deux fois les instructions suivant la ligne `do` qui sont *indentées*. Ensuite, il dépose sa sonnette (`putbeeper`), exécute l'ensemble des instructions définissant la fonction `untour`, ramasse sa sonnette (`pickbeeper`), avance encore de trois pas, et s'éteint.

Pour bien comprendre ce qui se passe, le plus simple est que vous essayiez de modifier une ligne ou deux, pour voir ce qui se passe. Et s'il se passe quelque chose qui n'est pas prévu, tentez de comprendre en suivant pas-à-pas le déroulement des instructions.

2) Les commandes de Guido

Les commandes permettant de contrôler Guido sont de 3 types :

- les commandes *primitives* :
  - `move` : avancer d'une case vers l'avant,
  - `turnleft` : tourner de 90° vers la gauche,
  - `pickbeeper` : ramasser une sonnette,
  - `putbeeper` : déposer une sonnette,
  - `turnoff` : s'éteindre ;
- les *tests* :
  - `front_is_clear` : vrai si il n'y a pas de mur devant,
  - `front_is_blocked` : vrai si il y a un mur devant,
  - `left_is_clear` : vrai si il n'y a pas de mur à gauche,
  - `left_is_blocked` : vrai si il y a un mur à gauche,
  - `right_is_clear` : vrai si il n'y a pas de mur à droite,
  - `right_is_blocked` : vrai si il y a un mur à droite,
  - `next_to_a_beeper` : vrai si la case comporte une sonnette,
  - `not_next_to_a_beeper` : vrai si la case ne comporte pas de sonnette,
  - `any_beepers_in_beeper_bag` : vrai si Guido transporte au moins une sonnette,
  - `no_beepers_in_beeper_bag` : vrai si Guido ne transporte aucune sonnette,
  - `facing_north` : vrai si Guido regarde vers le nord,
  - `not_facing_north` : vrai si Guido ne regarde pas vers le nord,
  - `facing_south` : vrai si Guido regarde vers le sud,
  - `not_facing_south` : vrai si Guido ne regarde pas vers le sud,
  - `facing_east` : vrai si Guido regarde vers l'est,
  - `not_facing_east` : vrai si Guido ne regarde pas vers l'est,
  - `facing_west` : vrai si Guido regarde vers l'ouest,
  - `not_facing_west` : vrai si Guido ne regarde pas vers l'ouest ;
- les *structure* :
  - la *conditionnelle* : "`if <condition>: <actions>`" qui exécute les actions si la condition est vraie, et rien sinon,
  - la *boucle itérative* : "`do <nombre>: <actions>`" qui exécute les actions le nombre de fois indiqué,
  - la *boucle conditionnelle* : "`while <condition>: <actions>`" qui exécute les actions *tant que* la condition reste vraie,
  - la *définition de fonction* : "`define <fonction>: <instructions>`" qui définit `fonction` comme un raccourci pour la séquence des `instructions`.

Un tutoriel est disponible dans le répertoire de la classe, il vous est demandé d'en suivre au moins le début, et de faire les exercices, afin de vous familiariser avec l'ensemble des commandes.

3) Des exercices

Exercice n°1 : Guido à Rolland-Garros

31) Guido veut travailleur à Rolland-Garros comme ramasseur de balles. Le responsable lui fait passer un test : partant de la situation de la figure 1, Guido doit ramasser toutes les balles (les sonnettes) disséminées sur le terrain et les remettre dans la corbeille dans le coin inférieur gauche, pour arriver à la situation de la figure 2.

Écrivez un code permettant à Guido de réaliser cette opération. Il devra comporter une boucle `do` répétant 6 fois le ramassage d'une rangée de balles, cette opération étant elle-même constituée d'une boucle *do*.

32) Ça y est, Guido est engagé ! Lors de son premier travail, il doit ramasser les balles laissées sur le terrain après l'entraînement d'un champion. Mais il se rend compte que la situation (celle de la figure 3) est un peu différente de celle de son test d'embauche : il y a parfois plusieurs balles au même endroit, et d'autres endroits ne comportent pas de balle.

Modifiez le code de votre premier programme de manière à ce que Guido puisse accomplir cette nouvelle tâche.

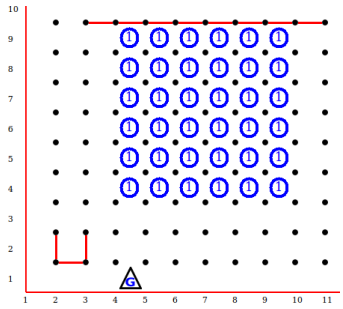


Figure 1

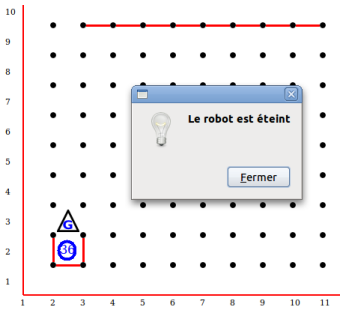


Figure 2

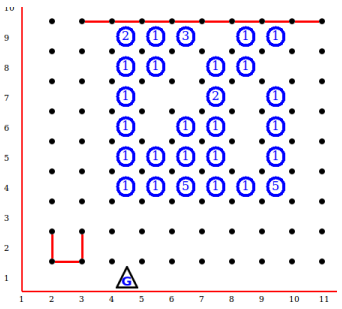


Figure 3

Exercice n°2 : Guido fait la course

31) Guido a décidé de faire du sport : il veut se mettre à la course de haie. Une course se passe de la manière suivante : Guido part du coin inférieur gauche, comme indiqué à la figure 4. Il doit courir vers la droite et contourner les haies représentées par des murs, et s'arrêter à la 17ème intersection (figure 5).

i. Écrivez une fonction `saute_haie` indiquant à Guido comment "sauter une haie" : partant de la gauche de la haie, regardant vers l'est, il doit se retrouver derrière la haie, regardant encore vers l'est.

ii. Utilisez cette fonction pour écrire un programme réalisant l'intégralité de la course.

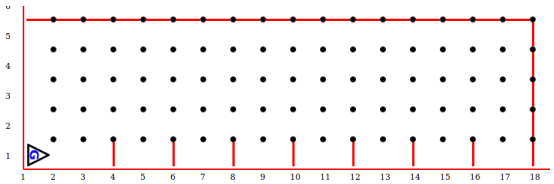


Figure 4

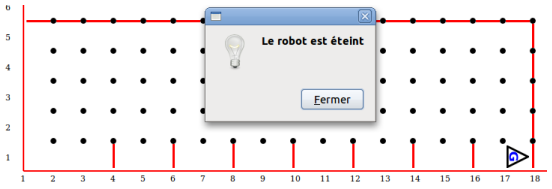


Figure 5

32) Guido trouve la course de haies trop facile, il veut se mettre à la "course de murs" : les obstacles sont disposés aux mêmes endroits sur la piste, mais il sont de hauteur variable (voir figure 6).

Modifiez votre code de manière à permettre à Guido d'exceller dans ce nouvel exercice.

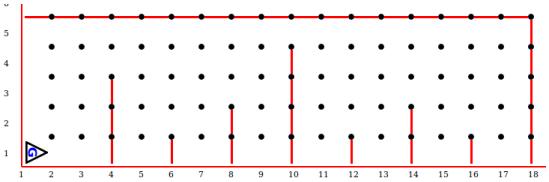


Figure 6

4) Travail personnel

Voici trois exercices plus complexes, qui ne sont pas simplement des exercices de programmation, mais pour lesquels il faudra *concevoir* un véritable algorithme. Il vous faudra réfléchir à une méthode plus ou moins astucieuse pour parvenir aux objectifs à atteindre, en tenant compte des contraintes.

De l'autre coté du miroir

Guido se trouve d'un coté d'un mur horizontal, et voudrait bien passer de l'autre coté. Le malheur, c'est qu'il ne sait pas quelle est la largeur du mur. La figure 7 montre la situation initiale, et la figure 8 la situation finale.

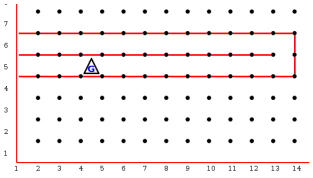


Figure 7

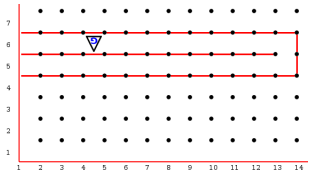


Figure 8

41) Dans un premier temps, on va supposer que :

- Guido n'a pas de sonnettes dans son sac à sonnettes,
- le parcours jusqu'au bord du mur est semé de sonnettes,
- par contre, il n'y a pas de mur vertical pour indiquer le bord du mur.

La figure 9 montre la situation de départ :

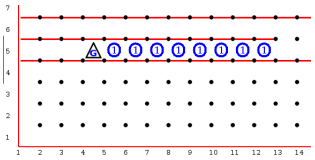


Figure 9

Concevez un algorithme permettant à Guido de se retrouver de l'autre coté du miroir. Vous avez le droit de ramasser les sonnettes, de les déposer autre part, seule la position finale de Guido sera prise en compte pour évaluer votre travail. Bien entendu, votre algorithme doit s'adapter à la taille du mur. Vous devrez le tester avec un mur de longueur différente pour vérifier ce fait.

42) Dans cette deuxième partie, il n'y a pas de sonnettes sur le terrain. La situation initiale est donc celle de la figure 7. Par contre, Guido a un sac rempli de sonnettes, dont il peut disposer à sa guise. Vous pouvez compter sur la présence du mur à droite, mais vous n'êtes pas obligé de l'utiliser !

Concevez un algorithme permettant à Guido de passer de l'autre coté du miroir. Encore une fois, seule la position finale du robot sera évaluée.

L'échiquier

Le père de Guido lui a demandé de peindre un échiquier. Il part d'un terrain rectangulaire délimité par des murs, comme dans la figure 10 ci-dessous, et doit peindre une case sur deux en y déposant une sonnette. Le résultat à atteindre est montré à la figure 11.

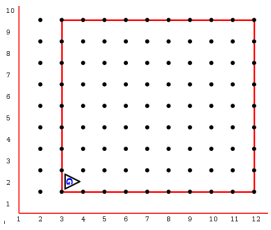


Figure 9

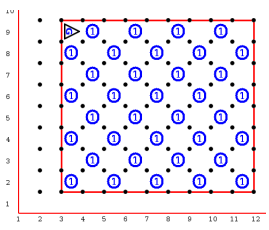


Figure 10

Bien entendu, Guido ne connaît pas initialement la taille du terrain, votre programme doit donc être adaptable. En particulier, ne pas oublier le cas des terrains filiformes, donc une (ou les deux !) dimension est égale à 1.

Pour les plus forts : trouver le milieu

Guido se trouve dans une pièce rectangulaire dont il ne connaît pas la largeur. Son père lui a demandé de positionner une sonnette au milieu de manière à percer un trou dans le mur pour accrocher la télévision.

Tout ce dont Guido dispose, c'est d'un sac rempli de sonnettes. Il sait qu'il se trouve à gauche de la pièce. Si la pièce est de largeur impaire, la sonnette devra être au milieu. Par contre, si la pièce est de largeur paire, l'une ou l'autre des deux positions centrales sera satisfaisante. Ainsi les deux figures ci-dessous sont acceptées comme positions finales :

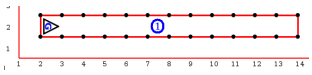


Figure 11

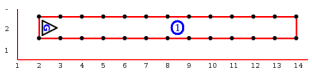


Figure 12

Concevez un algorithme permettant à Guido de positionner sa sonnette.