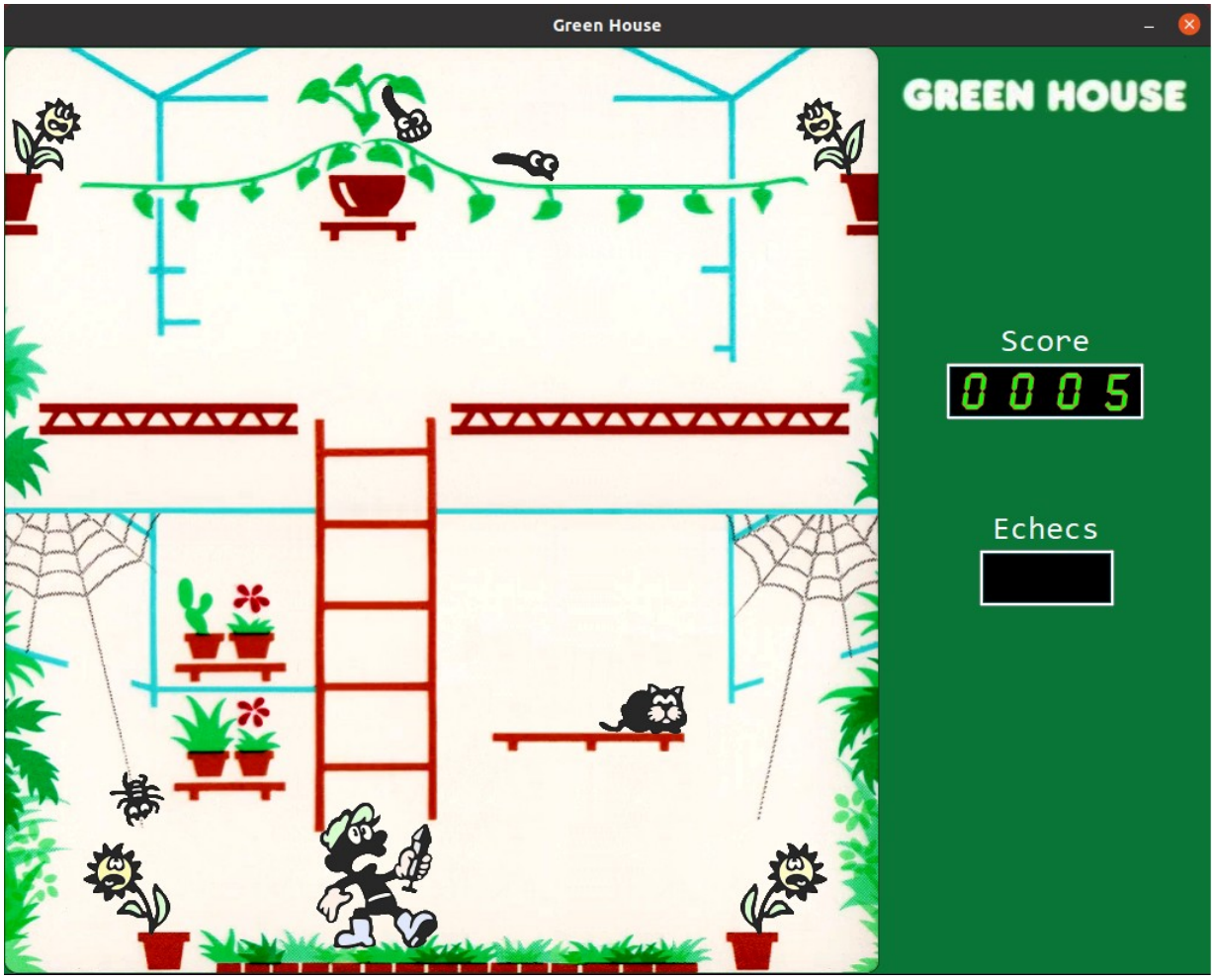
GREEN HOUSE

Origine de l'idée : le jeu électronique Green House (Game & Watch) de Nintendo.
Référence : https://www.youtube.com/watch?v=CS-JGLZpSIU

1. Aperçu général du jeu



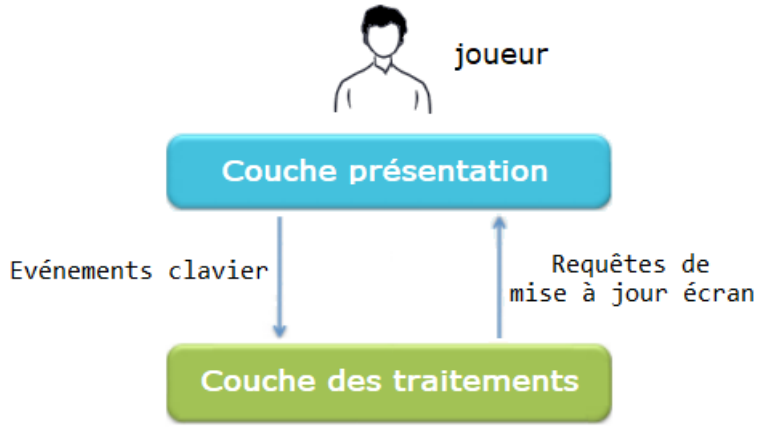
Le joueur incarne le jardinier Stanley dont la fonction est de défendre sa serre contre divers ennemis. Il doit éliminer avec son insecticide les nombreuses araignées et chenilles qui s'approchent de ses fleurs ainsi les guêpes qui s'approchent de son chat. Chaque ennemi éliminé rapporte 1 point. Un échec survient quand une des fleurs est mangée par une chenille ou par une araignée ou quand le chat est piqué par une guêpe. La partie se termine dès l'accumulation de 3 échecs.

2. Architecture générale du jeu

Le jeu est construit sur l'architecture en deux couches suivante :

- La **couche présentation**, pour le compte de la couche des traitements, crée la fenêtre du jeu, intercepte les événements produits par le joueur (frappe sur une touche du curseur, clic sur la croix de la fenêtre) et réalise les affichages des personnages.

- La **couche des traitements** gère tous les traitements relatifs au jeu : gestion de l'état de chaque personnage dans le jeu, prise en compte des collisions entre les personnages, gestion du score et gestion des échecs.



La couche présentation est donnée aux étudiants. Le travail va consister à implémenter complètement la couche des traitements.

3. Projet initial donné aux étudiants

Une version de départ du projet est donnée aux étudiants (archive *gh.rar* sur l'EV). On y trouve les images ainsi que les fichiers suivants : *main.py, jeu.py, contantes.py, presentation.py* et *stantley.py*.

A – Fichier main.py

C'est dans le fichier *main.py* que se trouve le code de démarrage du jeu.

```
from jeu import *

jeu = Jeu()
jeu.demarrer()
```

B – Fichier jeu.py

Dans le fichier *jeu.py* se trouve une version de départ de la classe **Jeu** qui est la classe principale du jeu.

```
import time
from presentation import *
from stanley import *

class Jeu:
    def __init__(self):
        self.presentation = Presentation()   # attribut pour la couche présentation
        self.stanley = Stanley()             # attribut pour Stanley
        # ...                                # attributs pour les ennemis, le score, ...

    # -----------------------------------------------------------------------
    # méthode qui contient la boucle principale du jeu

    def demarrer(self):
        while True:
            # le code de gestion du déplacement des ennemis et des collisions va venir ici ...

            # le code de génération des ennemis va venir ici ...

            # récupérer l'événement du joueur et changer l'état de Stanley

            self.stanley.actualiserEtat(self.presentation.lireEvenement())

            # mettre à jour l'image à l'écran

            self.actualiserEcran()

            # attendre 100 millisecondes (délai de référence)

            time.sleep(0.1)

    # -----------------------------------------------------------------------
    # méthode qui met à jour l'image du jeu à l'écran

    def actualiserEcran(self):
        self.presentation.effacerImageInterne()

        self.presentation.afficherStanley(self.stanley.etat, self.stanley.position,
                                          self.stanley.action)

        self.presentation.actualiserFenetreGraphique()
```

Description générale :

- Dans le **constructeur** de la classe Jeu, seront créés les attributs relatifs à la couche présentation, à Stanley, à la guêpe, à chaque liste (vide au départ) stockant les ennemis d'un type donné, à la liste des amis, au score et au nombre d'échecs.

- La méthode **actualiserEcran()** restaure l'image interne initiale du jeu en mémoire en invoquant **effacerImageInterne()** de la couche présentation, puis elle copie l'image souhaitée de chaque personnage dans l'image interne du jeu (dans le cas présent, uniquement Stanley est présent) et enfin, elle rend visible à l'écran la nouvelle image du jeu en invoquant **actualiserFenetreGraphique()** de la couche présentation.

- La méthode **demarrer()** contient la boucle principale du jeu dans laquelle on restera pendant toute la partie du jeu. Voici comment sera organisé le code interne à cette boucle :

  1. Tout le code relatif à la gestion des ennemis et des collisions sera placé au début de la boucle.

  2. Ensuite, viendra le code relatif à la génération de nouveaux ennemis.

  3. Puis, l'événement produit par le joueur (appui sur la touche vers la gauche, etc.) sera récupéré et transmis à la méthode actualiserEtat() de Stanley.

  4. Après, sera invoquée la méthode actualiserEcran() qui mettra à jour l'image du jeu à l'écran.

  5. Enfin, il y aura une attente de 0,1 seconde qui est le délai de référence à partir duquel tous les autres délais dans le jeu seront définis.

C – Fichier constantes.py

Le fichier *constantes.py* contient la classe **Constantes** qui regroupe toutes les constantes symboliques qu'on va utiliser à différentes endroits dans le code du jeu.

```
class Constantes:
    # différents états de Stanley

    HAUT = 0
    ECHELLE = 1
    BAS = 2

    # quand aucune touche du curseur/espace n'a été pressée

    AUCUN_EVENEMENT = -1

    # action de Stanley quand on a appuyé sur la touche espace

    SPRAY = 1

    # état normal des fleurs, du chat, de la guêpe, de chaque chenille, de Stanley, etc.

    NORMAL = 0

    # état d'un ennemi quand il a terminé son parcours
```

```
    TERMINE = 1

    # état d'un ami quand il est touché par un ennemi (ex: le chat est touché par la guêpe)

    TOUCHE = 2

    # types d'ami

    FLEUR_HG = 0
    FLEUR_HD = 1
    FLEUR_BG = 2
    FLEUR_BD = 3
    CHAT = 4

    # types d'ennemi

    GUEPE = 0
    CHENILLE_G = 1
    CHENILLE_D = 2
    ARAIGNEE_G = 3
    ARAIGNEE_D = 4

    # valeur quand aucun ennemi ne doit être générer pour l'instant

    AUCUN_ENNEMI = 5
```

D – Fichier presentation.py

Dans le fichier *presentation.py* se trouve la classe **Presentation** contenant les méthodes permettant d'utiliser l'écran, le clavier et la souris.

```
import pygame
import time
from constantes import *

class Presentation:
    def __init__(self):
        pygame.init()

        # charger toutes les images

        self.imgFondEcran = pygame.image.load("images/greenhouse.png")

        self.imgStanleyBasNormal0 = pygame.image.load("images/stanley/bas_normal_0.png")
        self.imgStanleyBasNormal1 = pygame.image.load("images/stanley/bas_normal_1.png")
        self.imgStanleyBasNormal2 = pygame.image.load("images/stanley/bas_normal_2.png")
        self.imgStanleyBasNormal3 = pygame.image.load("images/stanley/bas_normal_3.png")
        self.imgStanleyEchelle0 = pygame.image.load("images/stanley/echelle_0.png")
        self.imgStanleyEchelle1 = pygame.image.load("images/stanley/echelle_1.png")
        self.imgStanleyHautNormal0 = pygame.image.load("images/stanley/haut_normal_0.png")
        self.imgStanleyHautNormal1 = pygame.image.load("images/stanley/haut_normal_1.png")
        self.imgStanleyHautNormal2 = pygame.image.load("images/stanley/haut_normal_2.png")
        self.imgStanleyHautNormal3 = pygame.image.load("images/stanley/haut_normal_3.png")
        self.imgStanleyHautNormal4 = pygame.image.load("images/stanley/haut_normal_4.png")
        self.imgStanleyHautNormal5 = pygame.image.load("images/stanley/haut_normal_5.png")

        self.imgStanleyBasSpray0 = pygame.image.load("images/stanley/bas_spray_0.png")
        self.imgStanleyBasSpray2 = pygame.image.load("images/stanley/bas_spray_2.png")
        self.imgStanleyBasSpray3 = pygame.image.load("images/stanley/bas_spray_3.png")
        self.imgStanleyHautSpray0 = pygame.image.load("images/stanley/haut_spray_0.png")
        self.imgStanleyHautSpray1 = pygame.image.load("images/stanley/haut_spray_1.png")
        self.imgStanleyHautSpray3 = pygame.image.load("images/stanley/haut_spray_3.png")
        self.imgStanleyHautSpray4 = pygame.image.load("images/stanley/haut_spray_4.png")
        self.imgStanleyHautSpray5 = pygame.image.load("images/stanley/haut_spray_5.png")
```

```
        self.imgInsecticideHaut0 = pygame.image.load("images/insecticide/haut_gauche_0.png")
        self.imgInsecticideHaut1 = pygame.image.load("images/insecticide/haut_gauche_1.png")
        self.imgInsecticideHaut3 = pygame.image.load("images/insecticide/haut_droite_3.png")
        self.imgInsecticideHaut4 = pygame.image.load("images/insecticide/haut_droite_4.png")
        self.imgInsecticideHaut5 = pygame.image.load("images/insecticide/haut_droite_5.png")

        self.imgInsecticideBas0 = pygame.image.load("images/insecticide/bas_0.png")
        self.imgInsecticideBas2 = pygame.image.load("images/insecticide/bas_2.png")
        self.imgInsecticideBas3 = pygame.image.load("images/insecticide/bas_3.png")
        self.imgInsecticideMvtG = pygame.image.load("images/insecticide/bas_gauche_mvt.png")
        self.imgInsecticideMvtD = pygame.image.load("images/insecticide/bas_droite_mvt.png")

        self.imgEchec = pygame.image.load("images/stanley/echec.png")

        self.imgChat0 = pygame.image.load("images/amis/chat_0.png")
        self.imgChat1 = pygame.image.load("images/amis/chat_1.png")

        self.imgFleurHG0 = pygame.image.load("images/amis/haut_gauche_0.png")
        self.imgFleurHG1 = pygame.image.load("images/amis/haut_gauche_1.png")
        self.imgFleurHD0 = pygame.image.load("images/amis/haut_droite_0.png")
        self.imgFleurHD1 = pygame.image.load("images/amis/haut_droite_1.png")
        self.imgFleurBG0 = pygame.image.load("images/amis/bas_gauche_0.png")
        self.imgFleurBG1 = pygame.image.load("images/amis/bas_gauche_1.png")
        self.imgFleurBD0 = pygame.image.load("images/amis/bas_droite_0.png")
        self.imgFleurBD1 = pygame.image.load("images/amis/bas_droite_1.png")

        self.imgChenilleG0 = pygame.image.load("images/ennemis/chenille_gauche_0.png")
        self.imgChenilleG1 = pygame.image.load("images/ennemis/chenille_gauche_1.png")
        self.imgChenilleG2 = pygame.image.load("images/ennemis/chenille_gauche_2.png")
        self.imgChenilleG3 = pygame.image.load("images/ennemis/chenille_gauche_3.png")
        self.imgChenilleD0 = pygame.image.load("images/ennemis/chenille_droite_0.png")
        self.imgChenilleD1 = pygame.image.load("images/ennemis/chenille_droite_1.png")
        self.imgChenilleD2 = pygame.image.load("images/ennemis/chenille_droite_2.png")
        self.imgChenilleD3 = pygame.image.load("images/ennemis/chenille_droite_3.png")

        self.imgGuepe0 = pygame.image.load("images/ennemis/guepe_0.png")
        self.imgGuepe1 = pygame.image.load("images/ennemis/guepe_1.png")

        self.imgAraigneeG0 = pygame.image.load("images/ennemis/araignee_gauche_0.png")
        self.imgAraigneeG1 = pygame.image.load("images/ennemis/araignee_gauche_1.png")
        self.imgAraigneeD0 = pygame.image.load("images/ennemis/araignee_droite_0.png")
        self.imgAraigneeD1 = pygame.image.load("images/ennemis/araignee_droite_1.png")

        self.imgChiffre0 = pygame.image.load("images/chiffres/Zero.png")
        self.imgChiffre1 = pygame.image.load("images/chiffres/Un.png")
        self.imgChiffre2 = pygame.image.load("images/chiffres/Deux.png")
        self.imgChiffre3 = pygame.image.load("images/chiffres/Trois.png")
        self.imgChiffre4 = pygame.image.load("images/chiffres/Quatre.png")
        self.imgChiffre5 = pygame.image.load("images/chiffres/Cinq.png")
        self.imgChiffre6 = pygame.image.load("images/chiffres/Six.png")
        self.imgChiffre7 = pygame.image.load("images/chiffres/Sept.png")
        self.imgChiffre8 = pygame.image.load("images/chiffres/Huit.png")
        self.imgChiffre9 = pygame.image.load("images/chiffres/Neuf.png")

        # créer la fenêtre avec l'image du fond et le titre

        pygame.display.set_caption("Green House")
        pygame.display.set_icon(pygame.image.load("images/iconeFenetre.png"))
        self.ecran = pygame.display.set_mode((1016, 780))
        self.ecran.blit(self.imgFondEcran, (0, 0))
        pygame.display.update()

    # ------------------------------------------------------------------------
    # retourner la touche sur laquelle a appuyé le joueur ou fermer la fenêtre
    # si clic sur la croix

    def lireEvenement(self):
        for evenement in pygame.event.get():
```

```
            if evenement.type == pygame.QUIT:
                pygame.quit()
                exit()
            elif evenement.type == pygame.KEYDOWN:
                if evenement.key in [pygame.K_SPACE, pygame.K_UP, pygame.K_RIGHT, pygame.K_DOWN,
                                     pygame.K_LEFT]:
                    return evenement.key
        return Constantes.AUCUN_EVENEMENT

    # ------------------------------------------------------------------------
    # Fermer la fenêtre si clic sur la croix

    def attendreFermetureFenetre(self):
        while True:
            for evenement in pygame.event.get():
                if evenement.type == pygame.QUIT:
                    pygame.quit()
                    exit()
            time.sleep(0.2)

    # ------------------------------------------------------------------------
    # afficher Stanley avec ou sans le spray

    def afficherStanley(self, etat, position, action = Constantes.NORMAL):
        if action == Constantes.NORMAL:
            if etat == Constantes.HAUT:
                if position == 0:
                    self.afficherImage(18, 149, self.imgStanleyHautNormal0)
                elif position == 1:
                    self.afficherImage(128, 149, self.imgStanleyHautNormal1)
                elif position == 2:
                    self.afficherImage(249, 189, self.imgStanleyHautNormal2)
                elif position == 3:
                    self.afficherImage(360, 149, self.imgStanleyHautNormal3)
                elif position == 4:
                    self.afficherImage(470, 149, self.imgStanleyHautNormal4)
                elif position == 5:
                    self.afficherImage(585, 149, self.imgStanleyHautNormal5)
            elif etat == Constantes.ECHELLE:
                if position == 0:
                    self.afficherImage(249, 348, self.imgStanleyEchelle0)
                elif position == 1:
                    self.afficherImage(267, 490, self.imgStanleyEchelle1)
            elif etat == Constantes.BAS:
                if position == 0:
                    self.afficherImage(138, 608, self.imgStanleyBasNormal0)
                elif position == 1:
                    self.afficherImage(252, 608, self.imgStanleyBasNormal1)
                elif position == 2:
                    self.afficherImage(373, 608, self.imgStanleyBasNormal2)
                elif position == 3:
                    self.afficherImage(494, 608, self.imgStanleyBasNormal3)
        elif action == Constantes.SPRAY:
            if etat == Constantes.HAUT:
                if position == 0:
                    self.afficherImage(18, 149, self.imgStanleyHautSpray0)
                    self.afficherImage(57, 22, self.imgInsecticideHaut0)
                elif position == 1:
                    self.afficherImage(128, 149, self.imgStanleyHautSpray1)
                    self.afficherImage(144, 22, self.imgInsecticideHaut1)
                elif position == 3:
                    self.afficherImage(360, 149, self.imgStanleyHautSpray3)
                    self.afficherImage(352, 22, self.imgInsecticideHaut3)
                elif position == 4:
                    self.afficherImage(470, 149, self.imgStanleyHautSpray4)
                    self.afficherImage(459, 22, self.imgInsecticideHaut4)
                elif position == 5:
                    self.afficherImage(585, 149, self.imgStanleyHautSpray5)
```

```
                self.afficherImage(560, 22, self.imgInsecticideHaut5)
        elif etat == Constantes.BAS:
            if position == 0:
                self.afficherImage(138, 608, self.imgStanleyBasSpray0)
                self.afficherImage(70, 596, self.imgInsecticideBas0)
            elif position == 2:
                self.afficherImage(373, 608, self.imgStanleyBasSpray2)
                self.afficherImage(392, 481, self.imgInsecticideBas2)
            elif position == 3:
                self.afficherImage(494, 608, self.imgStanleyBasSpray3)
                self.afficherImage(595, 595, self.imgInsecticideBas3)

    # ------------------------------------------------------------------------
    # afficher l'insecticide à gauche

    def afficherInsecticideG(self, position):
        if position == 0:
            self.afficherImage(24, 413, self.imgInsecticideMvtG)
        elif position == 1:
            self.afficherImage(36, 461, self.imgInsecticideMvtG)
        elif position == 2:
            self.afficherImage(48, 509, self.imgInsecticideMvtG)
        elif position == 3:
            self.afficherImage(60, 555, self.imgInsecticideMvtG)

    # ------------------------------------------------------------------------
    # afficher l'insecticide à droite

    def afficherInsecticideD(self, position):
        if position == 1:
            self.afficherImage(610, 555, self.imgInsecticideMvtD)
        elif position == 2:
            self.afficherImage(620, 509, self.imgInsecticideMvtD)
        elif position == 3:
            self.afficherImage(632, 461, self.imgInsecticideMvtD)
        elif position == 4:
            self.afficherImage(644, 413, self.imgInsecticideMvtD)

    # ------------------------------------------------------------------------
    # afficher un ami (fleur, chat...)

    def afficherAmi(self, typeAmi, etat):
        if typeAmi == Constantes.FLEUR_HG:
            if etat == Constantes.NORMAL:
                self.afficherImage(5, 41, self.imgFleurHG0)
            elif etat == Constantes.TOUCHE:
                self.afficherImage(15, 85, self.imgFleurHG1)
        elif typeAmi == Constantes.FLEUR_HD:
            if etat == Constantes.NORMAL:
                self.afficherImage(665, 41, self.imgFleurHD0)
            elif etat == Constantes.TOUCHE:
                self.afficherImage(659, 85, self.imgFleurHD1)
        elif typeAmi == Constantes.FLEUR_BG:
            if etat == Constantes.NORMAL:
                self.afficherImage(64, 668, self.imgFleurBG0)
            elif etat == Constantes.TOUCHE:
                self.afficherImage(43, 693, self.imgFleurBG1)
        elif typeAmi == Constantes.FLEUR_BD:
            if etat == Constantes.NORMAL:
                self.afficherImage(616, 668, self.imgFleurBD0)
            elif etat == Constantes.TOUCHE:
                self.afficherImage(638, 693, self.imgFleurBD1)
        elif typeAmi == Constantes.CHAT:
            if etat == Constantes.NORMAL:
                self.afficherImage(495, 498, self.imgChat0)
            elif etat == Constantes.TOUCHE:
                self.afficherImage(495, 425, self.imgChat1)
```

```
    # ------------------------------------------------------------------------
    # afficher une chenille à gauche

    def afficherChenilleG(self, position):
        if position == 0:
            self.afficherImage(45, 46, self.imgChenilleG0)
        elif position == 1:
            self.afficherImage(85, 48, self.imgChenilleG1)
        elif position == 2:
            self.afficherImage(140, 48, self.imgChenilleG2)
        elif position == 3:
            self.afficherImage(194, 40, self.imgChenilleG1)
        elif position == 4:
            self.afficherImage(234, 16, self.imgChenilleG3)

    # ------------------------------------------------------------------------
    # afficher une chenille à droite

    def afficherChenilleD(self, position):
        if position == 0:
            self.afficherImage(308, 16, self.imgChenilleD0)
        elif position == 1:
            self.afficherImage(353, 40, self.imgChenilleD1)
        elif position == 2:
            self.afficherImage(409, 48, self.imgChenilleD2)
        elif position == 3:
            self.afficherImage(466, 48, self.imgChenilleD1)
        elif position == 4:
            self.afficherImage(522, 48, self.imgChenilleD2)
        elif position == 5:
            self.afficherImage(578, 48, self.imgChenilleD1)
        elif position == 6:
            self.afficherImage(624, 45, self.imgChenilleD3)

    # ------------------------------------------------------------------------
    # afficher une araignee à gauche

    def afficherAraigneeG(self, position):
        if position == 0:
            self.afficherImage(39, 412, self.imgAraigneeG0)
        elif position == 1:
            self.afficherImage(51, 461, self.imgAraigneeG1)
        elif position == 2:
            self.afficherImage(63, 510, self.imgAraigneeG0)
        elif position == 3:
            self.afficherImage(75, 555, self.imgAraigneeG1)
        elif position == 4:
            self.afficherImage(87, 608, self.imgAraigneeG0)

    # ------------------------------------------------------------------------
    # afficher une araignee à droite

    def afficherAraigneeD(self, position):
        if position == 0:
            self.afficherImage(613, 608, self.imgAraigneeD0)
        elif position == 1:
            self.afficherImage(624, 555, self.imgAraigneeD1)
        elif position == 2:
            self.afficherImage(634, 510, self.imgAraigneeD0)
        elif position == 3:
            self.afficherImage(645, 461, self.imgAraigneeD1)
        elif position == 4:
            self.afficherImage(656, 412, self.imgAraigneeD0)

    # ------------------------------------------------------------------------
    # afficher la guêpe

    def afficherGuepe(self, position):
```

```
        if position == 0:
            self.afficherImage(410, 511, self.imgGuepe0)
        elif position == 1:
            self.afficherImage(468, 528, self.imgGuepe1)

    # ------------------------------------------------------------------------
    # afficher la tête de Stanley en cas d'échec
    # nbEchecs = (nombre d'échecs) 1, 2 ou 3

    def afficherEchecs(self, nbEchecs):
        for i in range(nbEchecs):
            self.afficherImage(824 + (i * 36), 427, self.imgEchec)

    # ------------------------------------------------------------------------
    # afficher le score

    def afficherScore(self, score):
        self.afficherChiffre(797, 270, int(score / 1000))
        self.afficherChiffre(837, 270, int(score /100) % 10)
        self.afficherChiffre(877, 270, int(score / 10) % 10)
        self.afficherChiffre(917, 270, score % 10)

    # ------------------------------------------------------------------------
    # afficher un chiffre

    def afficherChiffre(self, x, y, chiffre):
        if chiffre == 0:
            self.afficherImage(x, y, self.imgChiffre0)
        elif chiffre == 1:
            self.afficherImage(x, y, self.imgChiffre1)
        elif chiffre == 2:
            self.afficherImage(x, y, self.imgChiffre2)
        elif chiffre == 3:
            self.afficherImage(x, y, self.imgChiffre3)
        elif chiffre == 4:
            self.afficherImage(x, y, self.imgChiffre4)
        elif chiffre == 5:
            self.afficherImage(x, y, self.imgChiffre5)
        elif chiffre == 6:
            self.afficherImage(x, y, self.imgChiffre6)
        elif chiffre == 7:
            self.afficherImage(x, y, self.imgChiffre7)
        elif chiffre == 8:
            self.afficherImage(x, y, self.imgChiffre8)
        elif chiffre == 9:
            self.afficherImage(x, y, self.imgChiffre9)

    # ------------------------------------------------------------------------
    # afficher une image sur l'image de fond d'écran initiale

    def afficherImage(self, x, y, image):
        rect = image.get_rect()
        rect.x = x
        rect.y = y
        self.ecran.blit(image, rect)

    # ------------------------------------------------------------------------
    # restaurer l'image interne d'origine de l'écran (ceci provoque
    # l'effacement de tous les personnages)

    def effacerImageInterne(self):
        self.ecran.blit(self.imgFondEcran, (0, 0, 1016, 780), (0, 0, 1016, 780))

    # ------------------------------------------------------------------------
    # mettre à jour l'image visible l'écran

    def actualiserFenetreGraphique(self):
        pygame.display.update()
```

Description générale :

- Ces méthodes utilisent la bibliothèque **pygame** qui offre des facilités pour réaliser des affichages graphiques.

- Le **constructeur** de la classe Presentation charge en mémoire toutes les images du jeu et crée la fenêtre graphique dans laquelle est affichée l'image principale du jeu.

- La méthode **effacerImageInterne()** restaure l'image interne du jeu sans les personnages, le score et les échecs. Cette méthode ne change pas ce qui est visible pour l'instant dans la fenêtre du jeu à l'écran.

- Les différentes méthodes d'affichage des personnages (Stanley, la chenille vers la gauche, etc.) copient les images des personnages dans l'image principale du jeu en mémoire. Ces méthodes ne changent pas ce qui est visible pour l'instant dans la fenêtre du jeu à l'écran.

- La méthode **actualiserFenetreGraphique()** rend visible à l'écran les modifications faites dans l'image principale du jeu en mémoire.

- La méthode **lireEvenement()** teste si le joueur a appuyé sur une touche du curseur ou a cliqué sur la croix. Elle retourne l'identifiant de la touche (pygame.K_UP, pygame.K_DOWN, pygame.K_LEFT, pygame.K_RIGHT ou pygame.K_SPACE) sur laquelle a appuyé le joueur ou Constantes.AUCUN_EVENEMENT quand le joueur n'a produit aucun événement. Elle termine l'exécution de l'application si le joueur a cliqué sur la croix.

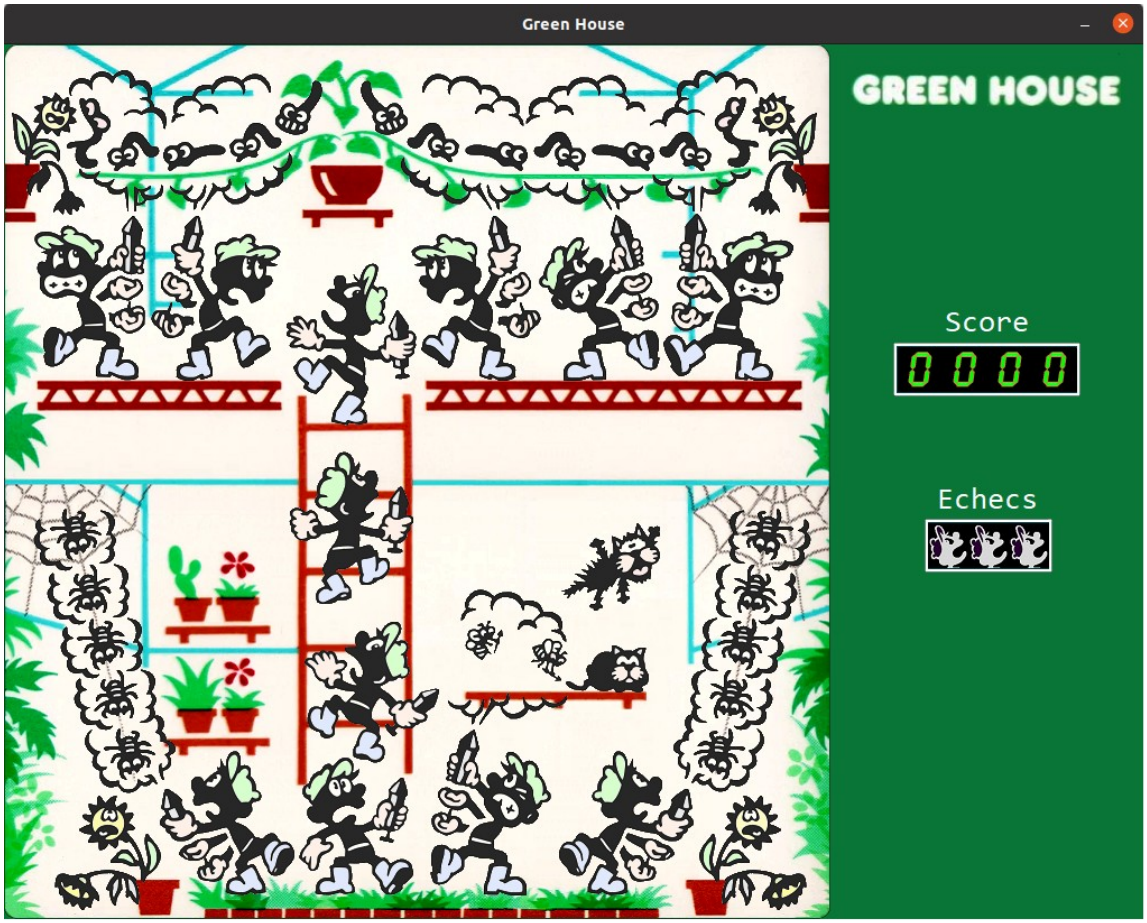
- La méthode **attendreFermetureFenetre()** attend que le joueur clique sur la croix, puis elle ferme la fenêtre du jeu. Cette méthode est bloquante.

Voici un exemple de code montrant comment utiliser les méthodes de la couche présentation :

```
from constantes import *
from presentation import *

pres = Presentation()

for i in range(0, 6):
    pres.afficherStanley(Constantes.HAUT, i, Constantes.NORMAL)
    pres.afficherStanley(Constantes.HAUT, i, Constantes.SPRAY)

pres.afficherStanley(Constantes.ECHELLE, 0)
pres.afficherStanley(Constantes.ECHELLE, 1)

for i in range(0, 4):
    pres.afficherStanley(Constantes.BAS, i, Constantes.NORMAL)
    pres.afficherStanley(Constantes.BAS, i, Constantes.SPRAY)

for i in range(0, 5):
    pres.afficherAmi(i, Constantes.NORMAL)
    pres.afficherAmi(i, Constantes.TOUCHE)

for i in range(0, 5):
    pres.afficherChenilleG(i)

for i in range(0, 7):
```

```
    pres.afficherChenilleD(i)

for i in range(0, 5):
    pres.afficherAraigneeG(i)
    pres.afficherAraigneeD(i)

for i in range(0, 4):
    pres.afficherInsecticideG(i)
    pres.afficherInsecticideD(i + 1)

pres.afficherGuepe(0)
pres.afficherGuepe(1)

pres.afficherEchecs(3)

pres.afficherScore(0)

pres.actualiserFenetreGraphique()

pres.attendreFermetureFenetre()
```

L'exécution de ce code produit la fenêtre du jeu suivante dans laquelle figurent toutes les images des personnages, les 3 échecs et la valeur 0 pour le score :



Comme on le voit, les méthodes d'affichage implémentées dans la classe Presentation ne sont pas difficiles à utiliser. Par exemple, pour afficher Stanley inactif dans l'état BAS à la position 0, il faut utiliser la méthode afficherStanley(Constantes.BAS, 0, Constantes.NORMAL). Les coordonnées X et Y de chaque image à l'écran sont enregistrées dans les méthodes d'affichage.

4. Etapes de réalisation du jeu

Remarques générales

- Il vous est demandé de suivre les étapes décrites à la suite dans l'ordre et de respecter les contraintes d'implémentation citées, même si elles ne vous paraissent pas les plus appropriées (le but étant d'apprendre des techniques de programmation en Python et non d'apprendre à faire un jeu).

- Pour chaque étape à réaliser, assurez-vous de bien comprendre ce qui est demandé, passez à l'implémentation, puis testez. Ne passez à l'étape suivante que lorsque l'étape courante est complètement terminée et fonctionnelle.

- Ce travail devra être réalisé individuellement. Il ne peut être fait en groupe. Toute copie entraînera une note de 0/20.

Fichiers dans le projet

Au fur et à mesure du développement du jeu, il faudra mettre à jour les classes suivantes :

- La classe **Jeu**
- La classe **Stanley**

Il faudra aussi créer les classes suivantes :

- La classe **Ennemis**
- La classe **Guepe**
- La classe **ChenilleG**
- La classe **ChenilleD**
- La classe **AraigneeG**
- La classe **AraigneeD**
- La classe **InsecticideG**
- La classe **InsecticideD**

Dans le projet, chaque classe devra se trouver dans un fichier source qui lui est propre. Par exemple, la classe Guepe devra se trouver dans le fichier *guepe.py*.

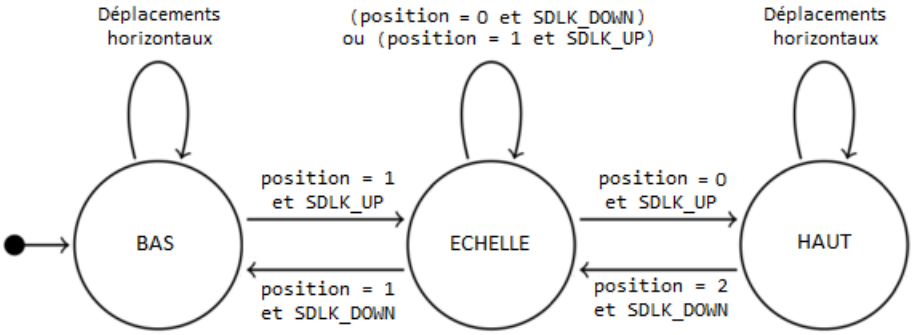
Nous allons commencer le développement du jeu en permettant à Stanley, notre personnage, de se déplacer partout dans la serre et de propager de l'insecticide ...

Étape 1 : gestion de Stanley

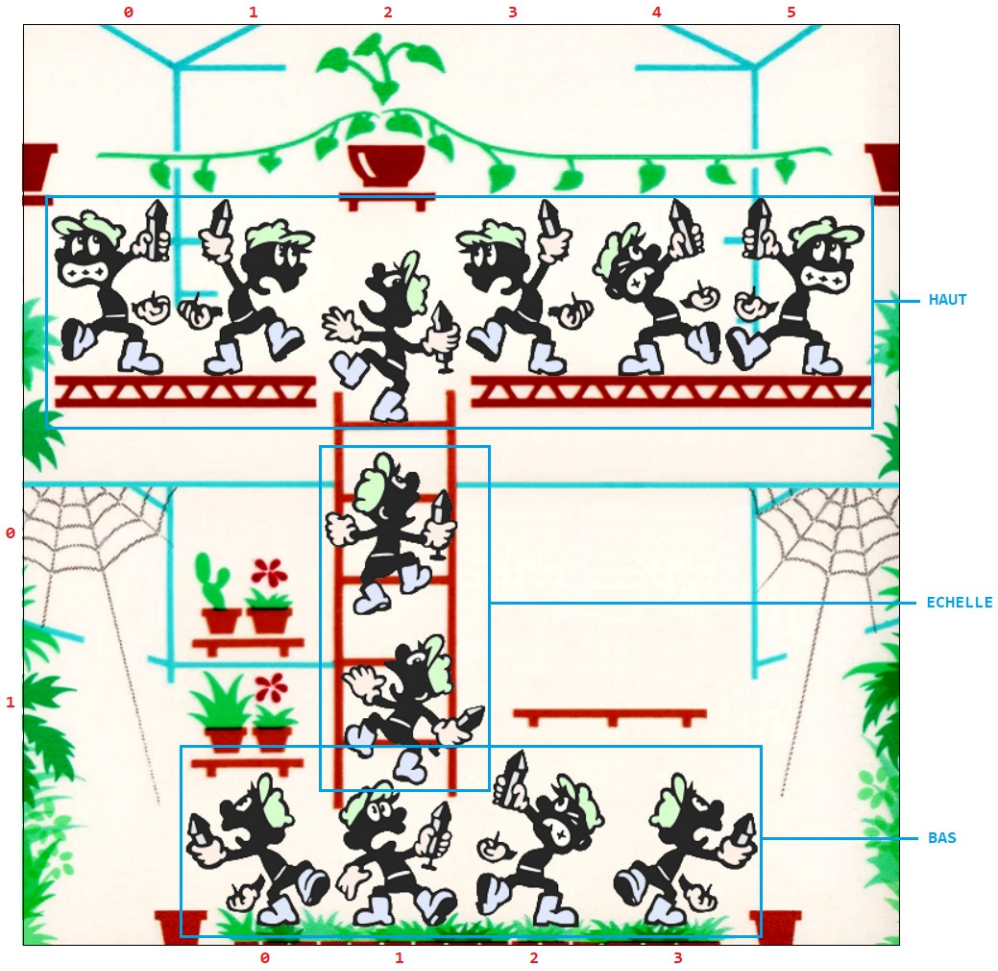
A - Déplacements de Stanley

La classe **Stanley** contient le code qui assure les déplacements et l'action de Stanley qui est notre personnage dans le jeu. L'objet issu de cette classe sera créé dans le constructeur de la classe Jeu.

La classe Stanley peut être vue comme implémentant une machine à états finis dans laquelle figurent les 3 états par lesquels Stanley peut passer à différentes reprises durant une partie du jeu, ainsi que les transitions possibles entre ces états :

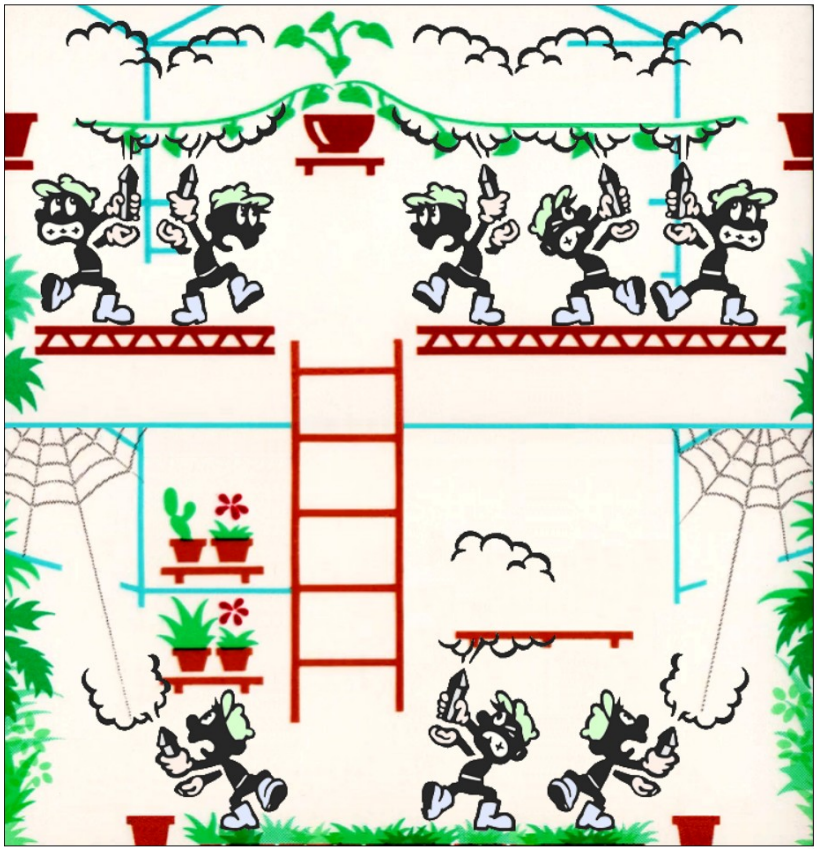


Dans l'état HAUT, Stanley peut se trouver à une position allant de 0 à 5 (les positions sont en rouge dans l'image ci-dessous). Dans l'état ECHELLE, Stanley peut se trouver à la position 0 ou 1. Dans l'état BAS, Stanley peut se trouver à une position allant de 0 à 3. Le déplacement de Stanley prend 0,1 s.



B – Action de Stanley

Le joueur peut appuyer sur la barre d'espace pour permettre à Stanley de répandre de l'insecticide (action Constantes.SPRAY). Voici les endroits où Stanley peut répandre de l'insecticide :



Cette action dure pendant 0,1 s, puis Stanley revient dans la situation par défaut (action Constantes.NORMAL) où il ne fait rien.

Voicici une version basique de la classe Stanley :

```
import pygame
from constantes import *

class Stanley:
    def __init__(self):
        self.etat = Constantes.BAS
        self.position = 1
        self.action = Constantes.NORMAL

    def actualiserEtat(self, evenement):
        if evenement == pygame.K_SPACE:
            print("à implémenter ...")
        else:
            if self.etat == Constantes.BAS:
                if evenement == pygame.K_RIGHT:
                    if self.position < 3:
                        self.position += 1
                elif evenement == pygame.K_LEFT:
                    if self.position > 0:
                        self.position -= 1
                elif evenement == pygame.K_UP:
                    print("à implémenter ...")
```

```
            elif self.etat == Constantes.ECHELLE:
                print("à implémenter ...")
            elif self.etat == Constantes.HAUT:
                print("à implémenter ...")
```

Quelques commentaires sur ce code :

- L'état de Stanley est stocké dans l'attribut **etat.** Plutôt que d'utiliser des numéros pour les etats, on peut utiliser les noms symboliques déclarés dans la classe Constantes : Constantes.BAS, Constantes.HAUT et Constantes.ECHELLE pour augmenter la clarté du code.

- Dans un état donné, la position de Stanley est stockée dans l'attribut **position**.

- En fonction de l'événement produit par le joueur, la méthode **actualiserEtat()** détermine la position suivante ou l'état suivant de Stanley. Pour savoir exactement où se trouve Stanley, il faut tenir compte conjointement des attributs etat et position. Dans le code partiellement donné ci-dessus de la méthode actualiserEtat(), on voit explicitement ce qui se passe quand Stanley est dans l'état BAS et que le joueur appuie sur la flèche droite ou la flèche gauche du curseur.

- Lorsque Stanley ne fait rien, la valeur Constantes.NORMAL doit se trouver dans l'attribut **action**, tandis que lorsque Stanley répand de l'insecticide, la valeur Constantes.SPRAY doit se trouver dans l'attribut action.

Étape 2 : affichage des "amis"

Cette étape est brève. Il s'agit de créer, dans le constructeur de la classe Jeu, une liste dans laquelle on place les 5 "amis" : la fleur en haut à gauche, la fleur en haut à droite, la fleur en bas à gauche, la fleur en bas à droite et le chat, en tenant compte, pour la position de chaque ami dans la liste, des constantes symboliques suivantes figurant dans la classe Constantes :

```
FLEUR_HG = 0
FLEUR_HD = 1
FLEUR_BG = 2
FLEUR_BD = 3
CHAT = 4
```

Un ami dans la liste est simplement représenté par une valeur entière. Au départ, sa valeur est Constantes.NORMAL indiquant qu'il est dans son état normal. Quand un ami est touché par un ennemi, son état doit passer à Constantes.TOUCHE.

Dans la méthode actualiserEcran(), il faut veiller à appeler la méthode **afficherAmi**() de la couche présentation pour afficher les amis.

Étape 3 : génération des ennemis

La classe **Ennemis** contient le code qui décide quand il faut créer un nouvel ennemi et de quel type est cet ennemi. L'objet issu de cette classe sera créé dans le constructeur de la classe Jeu.

À chaque itération, la boucle principale du jeu va appeler la méthode **actualiserEtat()** de l'objet issu de la classe Ennemis. Cette méthode réalise les 2 choses suivantes :

1. Elle décrémente le **délai d'attente** avant de créer un ennemi. Ce délai est stocké dans un attribut spécifique créé dans le constructeur de la classe Ennemis. Tant que ce délai d'attente n'est pas 0, la méthode actualiserEtat() retourne Constantes.AUCUN_ENNEMI. Quand ce délai passe à 0, actualiserEtat() retourne à la boucle principale le type d'ennemis à créer (choix aléatoire entre guêpe (Constantes.GUEPE), chenille à gauche (Constantes.CHENILLE_G), chenille à droite (Constantes.CHENILLE_D), araignée à gauche (Constantes.ARAIGNEE_G) et araignée à droite (Constantes.ARAIGNEE_D)). Ensuite, elle recharge un délai d'attente pour réaliser une nouvelle attente avant la création de l'ennemi suivant.

2. Le délai d'attente avant la génération d'un nouvel ennemi est fixé initialement à 1,5 s. Il faut donc au départ charger la valeur 16 dans l'attribut stockant ce délai. Dans le but de faire varier la difficulté du jeu pendant la partie, la méthode actualiserEtat() va faire varier ce délai toutes les 10 secondes (choix aléatoire du délai entre 1 s et 1,5 s).

La boucle principale du jeu récupère le type d'ennemi à créer et crée un objet approprié (par exemple, elle crée un objet à partir de la classe Guepe si l'ennemi à créer est du type Constantes.GUEPE.

Maintenant qu'on a la possibilité de générer un nouvel ennemi, le premier type d'ennemis qu'on va intégrer dans le jeu sera la guêpe ...

Étape 4 : les guêpes

A – Déplacements d'une guêpe

La classe **Guepe** contient le code qui contrôle les déplacements d'une guêpe.

Il ne peut y avoir qu'une seule guêpe à un moment donné dans le jeu. Un attribut **guepe** est créé dans le constructeur de la classe Jeu. On lui assigne soit **None** quand il n'y a aucune guêpe dans le jeu, soit un objet issu de la classe Guepe lorsqu'une guêpe est présente. Cet objet est créé lorsque la méthode actualiserEtat() de l'objet issu de la classe Ennemis indique à la boucle principale du jeu qu'une guêpe doit être créée.

Une guêpe commence à la **position** 0, ensuite elle passe à la position 1, puis elle pique le chat. Un attribut spécifique stocke la position courante de la guêpe.

Chaque déplacement de la guêpe se fait après une attente d'une seconde. Un attribut est utilisé pour stocker le **délai d'attente** restant avant le déplacement de la guêpe. La valeur de cet attribut est décrémentée par la méthode **actualiserEtat**() de l'objet guepe. Cette méthode est appelée à chaque itération de la boucle principale du jeu. Lorsque le délai d'attente est 0, la position ou l'état de la guêpe change, puis un nouveau délai d'attente d'1 s démarre. Tant que la guêpe est en cours de déplacement, son **état**, stocké dans l'attribut, est NORMAL. Quand la guêpe termine son parcours, son état passe à TERMINE.

B – La guêpe pique le chat

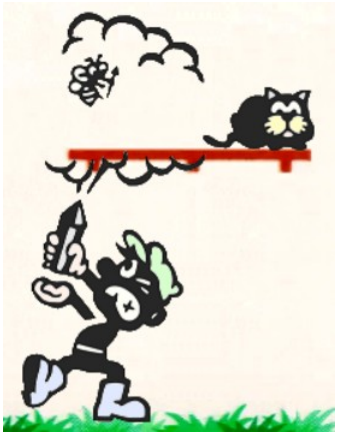
Quand la guêpe est dans l'état TERMINE, cela veut dire qu'elle pique le chat et qu'on est dans une situation d'échec.

Quand cette situation survient, elle est gérée dans la boucle principale du jeu ainsi : la valeur de l'attribut **echec** est incrémentée (la valeur 0 est assignée initialement à l'attribut echec dans le constructeur de la classe Jeu), le chat passe dans l'état TOUCHE, l'image à l'écran est mise à jour, le jeu se met en pause pendant 1,5 s, ensuite, on met la valeur None dans l'attribut guepe pour que la guêpe disparaisse et, finalement, le chat revient dans l'état NORMAL.

Quand 3 échecs ont été accumulés, la partie se termine. À ce moment, il faut invoquer la méthode attendreFermetureFenetre() qui attend que le joueur clique sur la croix pour fermer la fenêtre.

C – De l'insecticide tue une guêpe

Cette situation est prise en compte dans la boucle principale du jeu. Pour tuer une guêpe, Stanley doit être dans l'état BAS, à la position 2, et doit effectuer l'action SPRAY.



Cela prend 0,1 s à Stanley pour répandre de l'insecticide. Une guêpe n'est tuée qu'après cette attente de 0,1 s, ceci pour laisser le temps à l'insecticide de faire son effet. Quand la guêpe est tuée, on gagne 1 point et la guêpe disparaît.

D – Affichages

Dans la méthode actualiserEcran(), il faut veiller à appeler les méthodes **afficherGuepe**(), **afficherEchecs**() et **afficherScore**() de la couche présentation pour rendre visible, respectivement, la guêpe, les échecs et le score.

Remarque sur la boucle principale du jeu

Pour éviter d'avoir trop de lignes de code dans la boucle principale du jeu, la gestion de la guêpe (appeler la méthode actualiserEtat() de l'objet guepe pour changer l'état de la guêpe, tuer la guêpe quand Stanley répand de l'insecticide, ainsi que gérer l'échec quand la guêpe pique le chat) peut être implémentée dans une méthode spécifique de la classe Jeu, par exemple appelée gererGuepe(), qui est invoquée à partir de la boucle principale. Voici un exemple :

```
import time
from presentation import *
from stanley import *
from guepe import *

class Jeu:
    def __init__(self):
        self.presentation = Presentation()   # pour la couche présentation
        self.stanley = Stanley()             # pour Stanley
        self.ennemis = Ennemis()             # pour la génération des ennemis
        self.guepe = None                    # pour la guêpe
        # ...                                # attributs pour le score, les échecs, ...

    # ------------------------------------------------------------------------
    # méthode qui contient la boucle principale du jeu

    def demarrer(self):
        while True:
            self.gererGuepe()

            # le code de gestion des autres ennemis va venir ici ...

            # le code de génération des ennemis va venir ici ...

            # récupérer l'événement du joueur et changer l'état de Stanley

            self.stanley.actualiserEtat(self.presentation.lireEvenement())

            # mettre à jour l'image à l'écran

            self.actualiserEcran()

            # attendre 100 millisecondes (délai de référence)

            time.sleep(0.1)

    # ------------------------------------------------------------------------
    # méthode qui gère une guêpe

    def gererGuepe(self):
        # ...

    # ------------------------------------------------------------------------
    # méthode qui met à jour l'image du jeu à l'écran

    def actualiserEcran(self):
        self.presentation.effacerImageInterne()

        self.presentation.afficherStanley(self.stanley.etat, self.stanley.position,
                                          self.stanley.action)

        # ... code d'affichage de la guêpe

        self.presentation.actualiserFenetreGraphique()
```

La même chose pourra être faite pour les chenilles à gauche, puis pour les chenilles à droite, etc.

Nous allons à présent ajouter les chenilles dans le jeu ...

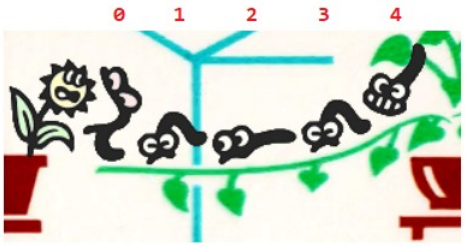
Étape 5 : les chenilles en haut à gauche

A – Déplacements d'une chenille

La classe **ChenilleG** contient le code qui contrôle le déplacement d'une chenille en haut à gauche.

Il peut y avoir plusieurs chenilles en même temps. Les chenilles sont stockées dans une **liste spécifique aux chenilles à gauche** qui est déclarée dans le constructeur de la classe Jeu. Un nouvel objet issu de la classe ChenilleG est créé et ajouté à la fin de la liste des chenilles lorsque la méthode actualiserEtat() de l'objet issu de la classe Ennemis indique à la boucle principale du jeu qu'une chenille à gauche doit être créée.

Voici ce qui se passe pour chaque chenille de la liste des chenilles à gauche. Une chenille commence à la **position** 4, va jusqu'à la position 0, puis elle mange la fleur en haut à gauche. Un attribut spécifique stocke la position courante de la chenille.



Chaque déplacement d'une chenille se fait après une attente de 0,6 s. Un attribut est utilisé pour stocker le **délai d'attente** restant avant le déplacement de la chenille. La valeur de cet attribut est décrémentée par la méthode **actualiserEtat**() de la chenille. Cette méthode est appelée à chaque itération de la boucle principale du jeu. Lorsque le délai d'attente est 0, la position ou l'état de la chenille change, puis un nouveau délai d'attente de 0,6 s démarre. Tant qu'une chenille est en cours de déplacement, son **état** est NORMAL. Quand une chenille termine son parcours, son état passe à TERMINE.

B – Une chenille mange la fleur en haut à gauche

Quand une chenille est dans l'état TERMINE, cela veut dire qu'elle mange la fleur en haut à gauche et qu'on est dans une situation d'échec.

Quand cette situation survient, elle est gérée dans la boucle principale du jeu comme quand une guêpe pique le chat, à ceci près que la chenille, à la fin de la pause de 1,5 s, est supprimée de la liste des chenilles à gauche.

C – De l'insecticide tue 1 ou 2 chenilles

Cette collision est prise en compte dans la boucle principale du jeu. Pour tuer une ou deux chenilles en même temps, Stanley doit être dans l'état HAUT, à la position 0 ou 1, et doit effectuer l'action SPRAY. À la position 0, Stanley peut tuer les chenilles aux positions 0 et 1. À la position 1, Stanley peut tuer les chenilles aux positions 2 et 3. Une chenille à la position 4 ne peut pas être tuée par de l'insecticide.

Cela prend 0,1 s à Stanley pour répandre de l'insecticide. Une chenille n'est tuée qu'après cette attente de 0,1 s, ceci pour laisser le temps à l'insecticide de faire son effet. Quand une chenille est tuée, on gagne 1 point et la chenille est supprimée de la liste des chenilles à gauche.

D – Affichages

Dans la méthode actualiserEcran(), il faut veiller à appeler la méthode **afficherChenilleG**() de la couche présentation pour rendre visible chaque chenille figurant dans la liste des chenilles à gauche.

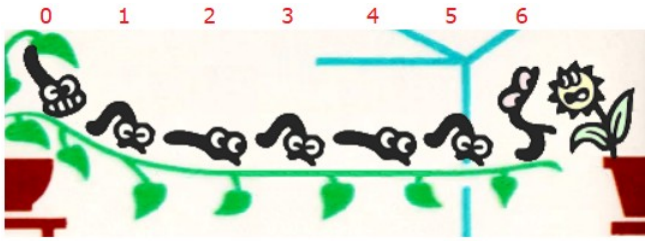
Étape 6 : les chenilles en haut à droite

A – Déplacements d'une chenille

La classe **ChenilleD** contient le code qui contrôle le déplacement d'une chenille en haut à droite.

Il peut y avoir plusieurs chenilles en même temps. Les chenilles sont stockées dans une **liste spécifique aux chenilles à droite** qui est déclarée dans le constructeur de la classe Jeu. Un nouvel objet issu de la classe ChenilleD est créé et ajouté à la fin de la liste des chenilles lorsque la méthode actualiserEtat() de l'objet issu de la classe Ennemis indique à la boucle principale du jeu qu'une chenille à droite doit être créée.

Voici ce qui se passe pour chaque chenille de la liste des chenilles à droite. Une chenille commence à la **position** 0, va jusqu'à la position 6, puis elle mange la fleur en haut à droite. Un attribut spécifique stocke la position courante de la chenille.



Chaque déplacement d'une chenille se fait après une attente de 0,6 s. Un attribut est utilisé pour stocker le **délai d'attente** restant avant le déplacement de la chenille. La valeur de cet attribut est décrémentée par la méthode **actualiserEtat**() de la chenille. Cette méthode est appelée à chaque itération de la boucle principale du jeu. Lorsque le délai d'attente est 0, la position ou l'état de la chenille change, puis un nouveau délai d'attente de 0,6 s démarre. Tant qu'une chenille est en cours de déplacement, son **état** est NORMAL. Quand une chenille termine son parcours, son état passe à TERMINE.

B – Une chenille mange la fleur en haut à droite

Quand une chenille est dans l'état TERMINE, cela veut dire qu'elle mange la fleur en haut à droite et qu'on est dans une situation d'échec.

Quand cette situation survient, elle est gérée dans la boucle principale du jeu comme quand une guêpe pique le chat, à ceci près que la chenille, à la fin de la pause de 1,5 s, est supprimée de la liste des chenilles à droite.

C - De l'insecticide tue 1 ou 2 chenilles

Cette collision est prise en compte dans la boucle principale du jeu. Pour tuer une ou deux chenilles en même temps, Stanley doit être dans l'état HAUT, à la position 3, 4 ou 5, et doit effectuer l'action SPRAY. À la position 3, Stanley peut tuer les chenilles aux positions 1 et 2. À la position 4, Stanley peut tuer les chenilles aux positions 3 et 4. À la position 5, Stanley peut tuer les chenilles aux positions 5 et 6. Une chenille à la position 0 ne peut pas être tuée par de l'insecticide.

Cela prend 0,1 s à Stanley pour répandre de l'insecticide. Une chenille n'est tuée qu'après cette attente de 0,1 s, ceci pour laisser le temps à l'insecticide de faire son effet. Quand une chenille est tuée, on gagne 1 point et la chenille est supprimée de la liste des chenilles à droite.

D – Affichages

Dans la méthode actualiserEcran(), il faut veiller à appeler la méthode **afficherChenilleD**() de la couche présentation pour rendre visible chaque chenille figurant dans la liste des chenilles à droite.

Après avoir testé que tout se passait bien dans le jeu avec les guêpes et les chenilles, nous pouvons à présent ajouter les araignées ...

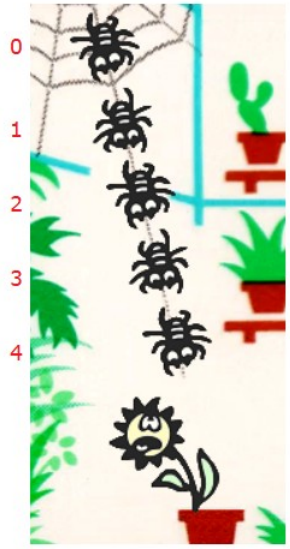
Étape 7 : les araignées en bas à gauche

A – Déplacements d'une araignée

La classe **AraigneeG** contient le code qui contrôle le déplacement d'une araignée en bas à gauche.

Il peut y avoir plusieurs araignées en même temps. Les araignées sont stockées dans une **liste spécifique aux araignées à gauche** qui est déclarée dans le constructeur de la classe Jeu. Un nouvel objet issu de la classe AraigneeG est créé et ajouté à la fin de la liste des araignées lorsque la méthode actualiserEtat() de l'objet issu de la classe Ennemis indique à la boucle principale du jeu qu'une araignée à gauche doit être créée.

Voici ce qui se passe pour chaque araignée de la liste des araignées à gauche. Une araignée commence à la **position** 0, va jusqu'à la position 4, puis elle mange la fleur en bas à gauche. Un attribut spécifique stocke la position courante de l'araignée.



Chaque déplacement d'une araignée se fait après une attente de 0,4 s. Un attribut est utilisé pour stocker le **délai d'attente** restant avant le déplacement de l'araignée. La valeur de cet attribut est décrémentée par la méthode **actualiserEtat**() de l'araignée. Cette méthode est appelée à chaque itération de la boucle principale du jeu. Lorsque le délai d'attente est 0, la position ou l'état de l'araignée change, puis un nouveau délai d'attente de 0,4 s démarre. Tant qu'une araignée est en cours de déplacement, son **état** est NORMAL. Quand une araignée termine son parcours, son état passe à TERMINE.

B – Une araignée mange la fleur en bas à gauche

Quand une araignée est dans l'état TERMINE, cela veut dire qu'elle mange la fleur en bas à gauche et qu'on est dans une situation d'échec.

Quand cette situation survient, elle est gérée dans la boucle principale du jeu comme quand une guêpe pique le chat, à ceci près que l'araignée, à la fin de la pause de 1,5 s, est supprimée de la liste des araignées à gauche.

C - De l'insecticide tue une araignée

Cette collision est prise en compte dans la boucle principale du jeu. Pour l'instant, la seule araignée qui peut être tuée est celle à la position 4. Pour tuer celle-ci, Stanley doit être dans l'état BAS, à la position 0, et doit effectuer l'action SPRAY.

Cela prend 0,1 s à Stanley pour répandre de l'insecticide. Une araignée n'est tuée qu'après cette attente de 0,1 s, ceci pour laisser le temps à l'insecticide de faire son effet. Quand une araignée est tuée, on gagne 1 point et l'araignée est supprimée de la liste des araignées à gauche.

D – Affichages

Dans la méthode actualiserEcran(), il faut veiller à appeler la méthode **afficherAraigneeG**() de la couche présentation pour rendre visible chaque araignée figurant dans la liste des araignées à gauche.

Étape 8 : les araignées en bas à droite

A – Déplacements d'une araignée

La classe **AraigneeD** contient le code qui contrôle le déplacement d'une araignée en bas à droite.

Il peut y avoir plusieurs araignées en même temps. Les araignées sont stockées dans une **liste spécifique aux araignées à droite** qui est déclarée dans le constructeur de la classe Jeu. Un nouvel objet issu de la classe AraigneeD est créé et ajouté à la fin de la liste des araignées lorsque la méthode actualiserEtat() de l'objet issu de la classe Ennemis indique à la boucle principale du jeu qu'une araignée à gauche doit être créée.

Voici ce qui se passe pour chaque araignée de la liste des araignées à droite. Une araignée commence à la **position** 4, va jusqu'à la position 0, puis elle mange la fleur en bas à droite. Un attribut spécifique stocke la position courante de l'araignée.

Chaque déplacement d'une araignée se fait après une attente de 0,4 s. Un attribut est utilisé pour stocker le **délai d'attente** restant avant le déplacement de l'araignée. La valeur de cet attribut est décrémentée par la méthode **actualiserEtat**() de l'araignée. Cette méthode est appelée à chaque itération de la boucle principale du jeu. Lorsque le délai d'attente est 0, la position ou l'état de l'araignée change, puis un nouveau délai d'attente de 0,4 s démarre. Tant qu'une araignée est en cours de déplacement, son **état** est NORMAL. Quand une araignée termine son parcours, son état passe à TERMINE.

B – Une araignée mange la fleur en bas à droite

Quand une araignée est dans l'état TERMINE, cela veut dire qu'elle mange la fleur en bas à droite et qu'on est dans une situation d'échec.

Quand cette situation survient, elle est gérée dans la boucle principale du jeu comme quand une guêpe pique le chat, à ceci près que l'araignée, à la fin de la pause de 1,5 s, est supprimée de la liste des araignées à droite.

C - De l'insecticide tue une araignée

Cette collision est prise en compte dans la boucle principale du jeu. Pour l'instant, la seule araignée qui peut être tuée est celle à la position 0. Pour tuer celle-ci, Stanley doit être dans l'état BAS, à la position 3, et doit effectuer l'action SPRAY.

Cela prend 0,1 s à Stanley pour répandre de l'insecticide. Une araignée n'est tuée qu'après cette attente de 0,1 s, ceci pour laisser le temps à l'insecticide de faire son effet. Quand une araignée est tuée, on gagne 1 point et l'araignée est supprimée de la liste des araignées à droite.

D – Affichages

Dans la méthode actualiserEcran(), il faut veiller à appeler la méthode **afficherAraigneeD**() de la couche présentation pour rendre visible chaque araignée figurant dans la liste des araignées à droite.

Il nous reste à présent à implémenter le déplacement de nuages d'insecticide en bas à gauche et à droite afin de tuer les araignées plus éloignées de Stanley...

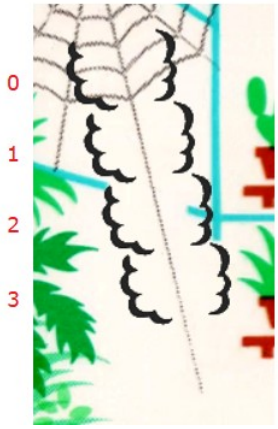
Bonus 1 : les nuages d'insecticide en bas à gauche

A – Déplacement d'un nuage d'insecticide

La classe **InsecticideG** contient le code qui contrôle le déplacement d'un nuage d'insecticide en bas à gauche.

Il peut y avoir plusieurs nuages d'insecticide en même temps. Les nuages d'insecticide sont stockés dans une **liste spécifique aux nuages d'insecticide à gauche** qui est déclarée dans le constructeur de la classe Jeu. Un nouvel objet issu de la classe InsecticideG est créé et ajouté à la fin de la liste des nuages d'insecticide lorsque Stanley est en bas à la position 0, que son action est SPRAY et qu'aucune araignée n'est présente à la position 4.

Voici ce qui se passe pour chaque nuage d'insecticide de la liste des nuages d'insecticide à gauche. Un nuage d'insecticide commence à la **position** 3, va jusqu'à la position 0, puis il disparaît. Un attribut spécifique stocke la position courante du nuage d'insecticide.



Chaque déplacement d'un nuage d'insecticide se fait après une attente de 0,1 s. La méthode **actualiserEtat**() du nuage d'insecticide est appelée à chaque itération de la boucle principale du jeu. Elle change la position du nuage d'insecticide. Tant qu'un nuage d'insecticide est en cours de déplacement, son **état** est NORMAL. Quand un nuage d'insecticide termine son parcours, son état passe à TERMINE.

B – Un nuage d'insecticide tue une araignée

Cette collision est prise en compte dans la boucle principale du jeu. Si une araignée est présente à la même position que le nuage d'insecticide, alors elle n'est pas immédiatement tuée, mais après une attente de 0,1 s, afin de laisser un peu de temps à l'insecticide pour faire son effet.

Quand cela survient, on gagne 1 point, l'araignée est supprimée de la liste des araignées à gauche et le nuage d'insecticide est supprimé de la liste des nuages d'insecticide à gauche.

C – Affichages

Dans la méthode actualiserEcran(), il faut veiller à appeler la méthode **afficherInsecticideG**() de la couche présentation pour rendre visible chaque nuage d'insecticide figurant dans la liste des nuages d'insecticide à gauche.

Bonus 2 : les nuages d'insecticide en bas à droite

A – Déplacement d'un nuage d'insecticide

La classe **InsecticideD** contient le code qui contrôle le déplacement de tout nuage d'insecticide en bas à droite.

Il peut y avoir plusieurs nuages d'insecticide en même temps. Les nuages d'insecticide sont stockés dans une **liste spécifique aux nuages d'insecticide à droite** qui est déclarée dans le constructeur de la classe Jeu. Un nouvel objet issu de la classe InsecticideD est créé et ajouté à la fin de la liste des nuages d'insecticide lorsque Stanley est en bas à la position 3, que son action est SPRAY et qu'aucune araignée n'est présente à la position 0.

Voici ce qui se passe pour chaque nuage d'insecticide de la liste des nuages d'insecticide à droite. Un nuage d'insecticide commence à la **position** 1, va jusqu'à la position 4, puis il disparaît. Un attribut spécifique stocke la position courante du nuage d'insecticide.

Chaque déplacement d'un nuage d'insecticide se fait après une attente de 0,1 s. La méthode **actualiserEtat**() du nuage d'insecticide est appelée à chaque itération de la boucle principale du jeu. Elle change la position du nuage d'insecticide. Tant qu'un nuage d'insecticide est en cours de déplacement, son **état** est NORMAL. Quand un nuage d'insecticide termine son parcours, son état passe à TERMINE.

B – Un nuage d'insecticide tue une araignée

Cette collision est prise en compte dans la boucle principale du jeu. Si une araignée est présente à la même position que le nuage d'insecticide, alors elle n'est pas immédiatement tuée, mais après une attente de 0,1 s, afin de laisser un peu de temps à l'insecticide pour faire son effet.

Quand cela survient, on gagne 1 point, l'araignée est supprimée de la liste des araignées à droite et le nuage d'insecticide est supprimé de la liste des nuages d'insecticide à droite.

C – Affichages

Dans la méthode actualiserEcran(), il faut veiller à appeler la méthode **afficherInsecticideD**() de la couche présentation pour rendre visible chaque nuage d'insecticide figurant dans la liste des nuages d'insecticide à droite.

L'implémentation du jeu est à présent terminée ! Il vous reste à apprécier le résultat ...