「プログラミング演習 2 (Python) レポート 1」

クラス　：　C

担当教員名　：　雪田修一

学籍番号　：　17k1004

氏名　：　伊神　亮

1．課題：

選んだ基本課題 → C の 2
拡張した項目 →　D , H

赤：ハンター
オレンジ：盗賊
紫：盗賊
ピンク：盗賊　　である。

作成した D の内容(「盗賊」を導入することを考える。(最大 30pts))

●「盗賊」はある確率でフィールドの適当な位置に出現し、貯蔵庫からキノコを奪おうとする。

●「盗賊」はハンターに複数回にわたって衝突されると、フィールドから消えるようにする。

作成した H の内容(上記以外の自由な拡張(各自希望のポイント数を書くこと))

●キノコを設置する際に任意の位置で複数のキノコが重ならないようにする。(10pts)

基本課題 C の問題文：

複数のハンターが協力して、一つの貯蔵庫にキノコを集めるようにせよ。なお、各ハンターが保持できるキノコの上限値をキノコ N 個とする。例えば N=15 など。

1. 貯蔵庫を一つに限定し、複数のハンターがその貯蔵庫にキノコを運ぶ。(50pts)
2. 貯蔵庫を複数配置し、各貯蔵庫に対して複数のハンターを所属させ、1 と同様に各ハンターが協力して自分が所属する貯蔵庫にキノコを運ぶ(65pts)

拡張課題 D の問題文：

キノコの貯蔵庫を奪い取る「盗賊」を導入することを考える。盗賊の特徴の例を書きに示す。

●「盗賊」はある確率でフィールドの適当な位置に出現し、貯蔵庫からキノコを奪おうとする。

●「盗賊」はハンターに複数回にわたって衝突されると、フィールドから消えるようにする。

●「「盗賊」はハンターに見つかると追われる。

拡張課題 H の問題文：

上記(拡張課題 D~G)以外の独自の拡張を施しても良い。

2. 課題の目的：

この授業を通して学んだこと(特にオブジェクト指向)を利用して、課題とされているプログラムを組む。
課題とされているプログラムを組むことによって、これまで学んできたことの復習を促しこれまで学んだことの理解を深める。

3. 方法：

プログラムの設計方針：
この課題はハンターの動きやキノコの有無を画像として表示させることによって現在の状況が分かりやすい。そのため tkinter を取り入れる。
プログラム内容を整理するため class を使う。今回は World, Mushroom, Thief, Box, AbstractHunter, HybridHunterA, RandomHunter, ForwardHunter の 8 つクラスを作成する。
ハンター、キノコ、貯蔵庫、盗賊などの生成また全体の実行を World クラスで関数を用いて定義する。
他のクラスでは主にクラスの名前に対する物に関係する情報、実行などを関数で定義する。
world = World()で world を World クラスとする。world.start()でプログラムを実行する。

4. 実行結果：

基本課題 C2:内容→

貯蔵庫を複数配置し、各貯蔵庫に対して複数のハンターを所属させ、1 と同様に各ハンターが協力して自分が所属する貯蔵庫にキノコを運ぶ(65pts)

結果→

貯蔵庫を 3 つ配置し、各貯蔵庫に 2 体のハンターを所属させた。各々のハンターは 15 個のキノコを集めると自分が所属する貯蔵庫にキノコを運んだ。

print(world.boxes[self.number].box)　(self.number=0,1,2)

とすると、15 の倍数の個数分キノコの座標が表示された。

拡張課題 D1:内容→

「盗賊」はある確率でフィールドの適当な位置に出現し、貯蔵庫からキノコを奪おうとする。

結果→　r = random.random()、r<0.1 のとき(確率)のみフィールドに出現するようにした。

print(r)を実行し、r=0.020821627858352176 の時に初めて盗賊が出現した。

盗賊一回目出現→(17,38)　二回目→(4,20)　三回目→(27,40)

これらより盗賊が現れる座標は適当であることが分かる。

#self.p とは盗賊が貯蔵庫から奪ったキノコを入れるリスト

print(self.p) → [(7, 3), (16, 6), (12, 1),…,(10,2)]

盗賊が貯蔵庫からキノコを奪えたことが分かる。

よって、課題の制作は成功した。

拡張課題 D2:内容→

「盗賊」はハンターに複数回にわたって衝突されると、フィールドから消えるようにする。

結果→　self.N = 3 の時、print(self.state)で"off"と表示された。self.state ="off"の時、盗賊は画像に表示されないプログラムになっている。また r=0.832172…となり、self.state が"on"にならなかったので self.render()が実行されないため画像に表示されなかった。よって課題の制作は成功した。

```
def state_check(self):
        if self.state == "off":
            r = random.random()
            if r < 0.1:
                self.x = int(50*random.random())
                self.y = int(50*random.random())
                self.state ="on"
        if self.state == "on":
            self.move()
```

```
                self.steal()
        self.check()


    def render_check(self):
            if self.state == "on":
        self.render()
```

拡張課題 H 内容→

キノコを設置する際に任意の位置で複数のキノコが重ならないようにする。(10pts)

結果→

```
            for i in range(4):
                n = int(30*random.random())
                m = int(30*random.random())
                u = self.mushroom.make_colony(n, m, 20, 60)
                for x in u:
                    if x in self.mushrooms:
                        None
                    else:
                    (self.mushrooms).add(x)
```

world.mushrooms に x を加える前に x という値が world.mushrooms の中にあるかを判定した。

5. 考察

今回の L07_問題 C 拡張課題 D をやってみて、考察されるのは、プログラムの内容がもっと分かりやすく、かつより良いプログラムが作れるのかということである。例えば

```
self.boxes[0].render(“orange”)
self.boxes[1].render(“purple”)
self.boxes[2].render(“pink”)
```

これは指定した貯蔵庫を指定した色で画像として表す操作である。見るとわかるが、[]内の数字と render のあとの色以外プログラムとしては同じである。この 3 行の文章を for 文を使って表すと簡潔である。

```
color = [“orange”, “purple”, “pink”]
for x in range(3):
    self.boxes[x].render(color[x])
```

同じ意味のプログラムでも今こうして２通りできた。この２通りはそれぞれ特徴を持っている。最初のプログラムは組むのが大変であるが、一目みるだけでプログラムの内容が分かりやすい。反対に２つ目のプログラムは組むのが１つ目に比べて楽ではあるが、一目見ただけではわかりづらい。このことよりプログラムを組むときは自分もしくは他人の目的、用途に合わせてつくるべきである。

Thief、Box、AbstractHunter、HybridHunterA、RandomHunter、ForwardHunter クラスで render 関数が定義されている。内容も色以外は同じである。今回プログラムではほとんど同じことを 6 回も書いている。これは大きな時間の無駄だ。したがって World クラスで render 関数を定義しようと思う。下記が新たな render 関数である。

```
def render(self,color):
        canvas.create_rectangle(self.x*8, self.y*8,
                                self.x*8+8, self.y*8+8, fill =color, outline =color)
```

これを World クラスに定義すると色(color)を指定するだけで同じ実装ができる。

ソースプログラム(プログラム 1)

```
from tkinter import*
import time, random
FIELD_X, FIELD_Y = 50,50
tk = Tk()
canvas = Canvas(tk, width = FIELD_X*8, height = FIELD_Y * 8)
canvas.pack()

class World:
    def __init__(self):
        self.hunters = []
        self.mushrooms = set()
        self.mushroom = Mushroom()
        self.boxes = []
```

```
        self.hp = []



def add_hunter(self, klass, x, y, number):
        self.hunters.append(klass(x, y, 0, [], number))

def add_box(self,x,y):
        self.boxes.append(Box(x,y))

def step(self):
        for x in self.hunters:
                x.move()
                x.hunt()
        self.mushroom.render("lawn green")
        self.boxes[0].render("orange")
        self.boxes[1].render("purple")
        self.boxes[2].render("pink")
        for x in self.hunters:
                x.render()
        tk.update()
        tk.update_idletasks()
        canvas.delete("all")
        time.sleep(0.04)



def start(self, n_steps):
        for i in range(4):
                n = int(30*random.random())
                m = int(30*random.random())
                u = self.mushroom.make_colony(n, m, 20, 60)
                for x in u:
                        (self.mushrooms).add(x)
        for x in range(3):
                a1 = int(50*random.random())
                a2 = int(50*random.random())
                self.add_box(a1,a2)
```

```
        for x in range(2):
            self.add_hunter(ForwardHunter, self.boxes[0].x, self.boxes[0].y, 0)
        for x in range(2):
            self.add_hunter(RandomHunter, self.boxes[1].x, self.boxes[1].y, 1)
        for x in range(2):
            self.add_hunter(HybridHunterA, self.boxes[2].x, self.boxes[2].y,2)
        for x in range(n_steps):
            self.step()




    def check_mushroom(self, x, y):
        if (x,y) in self.mushrooms:
            return True
        else:
            return False


class Mushroom:


    def render(self,color):
        for x in world.mushrooms:
            canvas.create_rectangle(8*(x[0]), 8*(x[1]),
                                    8*(x[0])+8, 8*(x[1])+8, outline= color ,fill= color)


    def make_colony(self, x, y, c, p):
        list =[]
        for u in range(0,c):
            for v in range(0,c):
                list.append((x+u, y+v))
        n = len(list)
        sel = random.sample(list, round(n * 0.01*p))
        return sel
```

```
class Box:

    def __init__(self,x,y):
        self.box = set()
        self.x, self.y = x, y

    def render(self,color):
        canvas.create_rectangle(8*self.x, 8*self.y,
                                8*self.x+8, 8*self.y+8, outline= color ,fill= color)


class AbstractHunter:
    def __init__(self, x, y, N, L, number):
        self.x, self.y = x, y
        self.location = (self.x, self.y)
        self.vx, self.vy = 1, 1
        self.N = N
        self.p = L
        self.number = number


    def move(self):
        if self.N == 15:
            self.change_dir2()
        else:
            self.change_dir()

        self.x = (self.x + self.vx) % FIELD_X
        self.y = (self.y + self.vy) % FIELD_Y
        self.location = (self.x, self.y)
        world.hp.append(self.location)



    def change_dir(self):
```

```
        raise NotImplementedError("subclass responsibility")


def change_dir2(self):
        if self.x > world.boxes[self.number].x:
                self.vx = -1
                if self.y < world.boxes[self.number].y:
                        self.vy = 1
                elif self.y > world.boxes[self.number].y:
                        self.vy = -1
                elif self.y == world.boxes[self.number].y:
                        self.vy = 0


        elif self.x < world.boxes[self.number].x:
                self.vx = 1
                if self.y < world.boxes[self.number].y:
                        self.vy = 1
                elif self.y > world.boxes[self.number].y:
                        self.vy = -1
                elif self.y == world.boxes[self.number].y:
                        self.vy = 0
        elif self.x == world.boxes[self.number].x:
                self.vx = 0
                if self.y < world.boxes[self.number].y:
                        self.vy = 1
                elif self.y > world.boxes[self.number].y:
                        self.vy = -1
                elif self.y == world.boxes[self.number].y:
                        self.vy = 0


def render(self):
        canvas.create_rectangle(self.x*8, self.y*8,
                                        self.x*8+8, self.y*8+8, fill = "black", outline = "black")


def __str__(self):
        return "{}:pos = ({}, {})".format(self.__class__.__name__,
                                                self.x, self.y)
```

```
    def hunt(self):
        if self.N == 15:
            if world.boxes[self.number].x - self.x == 0 and world.boxes[self.number].y - self.y == 0:
                for x in range(self.N):
                    (world.boxes[self.number].box).add((self.p)[0])
                    (self.p).remove((self.p)[0])
                self.N = 0
        else:
            if (self.x,self.y) in world.mushrooms:
                self.N = self.N + 1
                (self.p).append((self.x, self.y))
                (world.mushrooms).remove((self.x,self.y))

            if self.x == world.boxes[self.number].x and self.y == world.boxes[self.number].y:
                for x in range(self.N):
                    (world.boxes[self.number].box).add((self.p)[0])
                    (self.p).remove((self.p)[0])
                self.N = 0

    def search_mushroom(self):
        to_check = [(1, 1), (0, 1), (-1, 1), (-1, 0), (-1,-1), (0, -1), (1, -1), (1, 0)]
        for x in to_check:
            check_x = (self.x + x[0]) % FIELD_X
            check_y = (self.y + x[1]) % FIELD_Y
            if world.check_mushroom(check_x, check_y):
                return x
        return -1
    #self.__class__.__name__でクラス名が取得できる。


class HybridHunterA(AbstractHunter):

    def __init__(self, x, y, N, L,number):
        super().__init__(x, y, N, L, number)
        self.state = "exploring"
```

```
def change_dir(self):
    mushroom_dir = self.search_mushroom()
    if mushroom_dir != -1:
        self.vx, self.vy = mushroom_dir
    else:

        self.set_state()

        dirs = [(1, 1), (0, 1), (-1, 1), (-1, 0), (-1,-1), (0, -1), (1, -1), (1, 0)]
        ind = dirs.index((self.vx, self.vy))
        u = random.random()
        if self.state == "exploring":
            if u<0.1:
                newInd = (ind+1)%8
            elif u<0.2:
                newInd = (ind-1)%8
            else:
                newInd = ind

        elif self.state == "searching":
            if u<0.5:
                newInd = (ind+1)%8
            else:
                newInd = ind

        self.vx, self.vy = dirs[newInd]




def set_state(self):#状態を(state)を変更する
    if random.random() < 0.05:
        if self.state == "searching":
            self.state = "exploring"
```

```
            elif self.state == "exploring":
                self.state = "searching"

    def render(self):
        if self.state == "exploring":
            canvas.create_rectangle(self.x*8, self.y*8,
                                    self.x*8+8, self.y*8+8, fill ="pink", outline ="pink")
        elif self.state == "searching":
            canvas.create_rectangle(self.x*8, self.y*8,
                                    self.x*8+8, self.y*8+8, fill ="green", outline ="green")

class RandomHunter (AbstractHunter):
    def change_dir(self):
        mushroom_dir = self.search_mushroom()
        if mushroom_dir != -1:
            self.vx, self.vy = mushroom_dir
        else:
            dirs = [(1, 1), (0, 1), (-1, 1), (-1, 0), (-1,-1), (0, -1), (1, -1), (1, 0)]
            self.vx, self.vy = random.choice(dirs)




    def render(self):
        canvas.create_rectangle(self.x*8, self.y*8,
                                self.x*8+8, self.y*8+8, fill ="purple", outline ="purple")

class ForwardHunter(AbstractHunter):
    def change_dir(self):
        mushroom_dir = self.search_mushroom()
        if mushroom_dir != -1:
            self.vx, self.vy = mushroom_dir
        else:
            dirs = [(1, 1), (0, 1), (-1, 1), (-1, 0), (-1,-1), (0, -1), (1, -1), (1, 0)]
            ind = dirs.index((self.vx, self.vy))
            u = random.random()
            if u<0.05:
```

```
                newInd = (ind+1)%8
            elif u<0.1:
                newInd = (ind-1)%8
            else:
                newInd = ind
            self.vx, self.vy = dirs[newInd]


    def render(self):
        canvas.create_rectangle(self.x*8, self.y*8,
                                self.x*8+8, self.y*8+8, fill ="orange", outline ="orange")

world = World()

world.start(1000)
```

ソースプログラム(プログラム 2)

```
from tkinter import*
import time, random
FIELD_X, FIELD_Y = 50,50
tk = Tk()
canvas = Canvas(tk, width = FIELD_X*8, height = FIELD_Y * 8)
canvas.pack()

class World:
    def __init__(self):
        self.hunters = []
        self.mushrooms = set()
        self.mushroom = Mushroom()
        self.boxes = []
```

```
        self.thiefs2 = set()
        self.thiefs = []
        self.hp = []


def add_hunter(self, klass, x, y, number):
        self.hunters.append(klass(x, y, 0, [], number))

def add_box(self,x,y):
        self.boxes.append(Box(x,y))

def add_thief(self,x,y):
        self.thiefs.append(Thief( x,y, 0, [], "off"))

def step(self):
        for x in self.hunters:
                x.move()
                x.hunt()
        for thief in self.thiefs:
                thief.state_check()
        self.mushroom.render("lawn green")
        self.boxes[0].render("orange")
        self.boxes[1].render("purple")
        self.boxes[2].render("pink")
        for x in self.hunters:
                x.render()
        for thief in self.thiefs:
                thief.render_check()
        tk.update()
        tk.update_idletasks()
        canvas.delete("all")
        time.sleep(0.04)


def start(self, n_steps):
        for i in range(4):
```

```
            n = int(30*random.random())
            m = int(30*random.random())
            u = self.mushroom.make_colony(n, m, 20, 60)
            for x in u:
                if x in self.mushrooms:
                    None
                else:
                    (self.mushrooms).add(x)
        for x in range(3):
            a1 = int(50*random.random())
            a2 = int(50*random.random())
            self.add_box(a1,a2)
        for x in range(2):
            self.add_hunter(ForwardHunter, self.boxes[0].x, self.boxes[0].y, 0)
        for x in range(2):
            self.add_hunter(RandomHunter, self.boxes[1].x, self.boxes[1].y, 1)
        for x in range(2):
            self.add_hunter(HybridHunterA, self.boxes[2].x, self.boxes[2].y,2)
        for x in range(1):
            b1 = int(50*random.random())
            b2 = int(50*random.random())
            self.add_thief(b1, b2)
        for x in range(n_steps):
            self.step()




def check_mushroom(self, x, y):
        if (x,y) in self.mushrooms:
            return True
        else:
            return False


def check_thief(self, x, y):
        if (x,y) in self.thiefs2:
```

```
                return True
            else:
                return False
            self.thiefs2.clear()


class Mushroom:


    def render(self,color):
        for x in world.mushrooms:
            canvas.create_rectangle(8*(x[0]), 8*(x[1]),
                                    8*(x[0])+8, 8*(x[1])+8, outline= color ,fill= color)


    def make_colony(self, x, y, c, p):
        list =[]
        for u in range(0,c):
            for v in range(0,c):
                list.append((x+u, y+v))
        n = len(list)
        sel = random.sample(list, round(n * 0.01*p))
        return sel


class Thief:
    def __init__(self, x,y, N, L, state):
        self.x, self.y = x, y
        self.locate = (self.x, self.y)
        self.N = N
        self.vx, self.vy = 1, 1
        self.p = L
        self.state = state



    def state_check(self):
        if self.state == "off":
            r = random.random()
            if r < 0.1:
                self.x = int(50*random.random())
```

```
                self.y = int(50*random.random())
                self.state ="on"
        if self.state == "on":
            self.move()
            self.steal()
        self.check()


def render_check(self):
        if self.state == "on":
            self.render()


def move(self):
        self.change_dir()
        self.x = (self.x + self.vx)%FIELD_X
        self.y = (self.y + self.vy)%FIELD_Y
        world.thiefs2.add((self.x,self.y))


def change_dir(self):
        dirs = [(1, 1), (0, 1), (-1, 1), (-1, 0), (-1,-1), (0, -1), (1, -1), (1, 0)]
        ind = dirs.index((self.vx, self.vy))
        r = random.random()
        if r < 0.1:
            newInd = (ind + 1) % 8
        else:
            newInd = ind
        self.vx, self.vy = dirs[newInd]


def steal(self):
        for box in world.boxes:
            if self.x == box.x and self.y == box.y:
                for x in box.box:
                    (self.p).append(x)
```

```
                box.box.clear()


    def render(self):
        canvas.create_rectangle(self.x*8, self.y*8,
                                self.x*8+8, self.y*8+8, fill="red",
                                outline="red")


    def check(self):
        self.locate = (self.x, self.y)
        if self.locate in world.hp:
            self.N = self.N + 1
        if self.N == 3:
            self.state = "off"
            self.N = 0
        world.hp.clear()


    def check_thief(self, x, y):
        if x == self.x and y == self.y :
            return True
        else:
            return False




class Box:

    def __init__(self,x,y):
        self.box = set()
        self.x, self.y = x, y

    def render(self,color):
        canvas.create_rectangle(8*self.x, 8*self.y,
```

```
                                        8*self.x+8, 8*self.y+8, outline= color ,fill= color)


class AbstractHunter:
    def __init__(self, x, y, N, L, number):
        self.x, self.y = x, y
        self.location = (self.x, self.y)
        self.vx, self.vy = 1, 1
        self.N = N
        self.p = L
        self.number = number



    def move(self):
        if self.N == 15:
            self.change_dir2()
        else:
            self.change_dir()

        self.x = (self.x + self.vx) % FIELD_X
        self.y = (self.y + self.vy) % FIELD_Y
        self.location = (self.x, self.y)
        world.hp.append(self.location)




    def change_dir(self):
        raise NotImplementedError("subclass responsibility")


    def change_dir2(self):
        if self.x > world.boxes[self.number].x:
            self.vx = -1
            if self.y < world.boxes[self.number].y:
                self.vy = 1
            elif self.y > world.boxes[self.number].y:
                self.vy = -1
```

```
            elif self.y == world.boxes[self.number].y:
                self.vy = 0


        elif self.x < world.boxes[self.number].x:
            self.vx = 1
            if self.y < world.boxes[self.number].y:
                self.vy = 1
            elif self.y > world.boxes[self.number].y:
                self.vy = -1
            elif self.y == world.boxes[self.number].y:
                self.vy = 0
        elif self.x == world.boxes[self.number].x:
            self.vx = 0
            if self.y < world.boxes[self.number].y:
                self.vy = 1
            elif self.y > world.boxes[self.number].y:
                self.vy = -1
            elif self.y == world.boxes[self.number].y:
                self.vy = 0


def render(self):
    canvas.create_rectangle(self.x*8, self.y*8,
                                    self.x*8+8, self.y*8+8, fill = "black", outline = "black")


def __str__(self):
    return "{}:pos = ({}, {})".format(self.__class__.__name__,
                                            self.x, self.y)


def hunt(self):
    if self.N == 15:
        if world.boxes[self.number].x - self.x == 0 and world.boxes[self.number].y - self.y == 0:
            for x in range(self.N):
                (world.boxes[self.number].box).add((self.p)[0])
                (self.p).remove((self.p)[0])


            self.N = 0
```

```
            else:
                if (self.x,self.y) in world.mushrooms:
                    self.N = self.N + 1
                    (self.p).append((self.x, self.y))
                    (world.mushrooms).remove((self.x,self.y))

                if self.x == world.boxes[self.number].x and self.y == world.boxes[self.number].y:
                    for x in range(self.N):
                        (world.boxes[self.number].box).add((self.p)[0])
                        (self.p).remove((self.p)[0])
                    self.N = 0

        def search_mushroom(self):
            to_check = [(1, 1), (0, 1), (-1, 1), (-1, 0), (-1,-1), (0, -1), (1, -1), (1, 0)]
            for x in to_check:
                check_x = (self.x + x[0]) % FIELD_X
                check_y = (self.y + x[1]) % FIELD_Y
                if world.check_mushroom(check_x, check_y):
                    return x
            return -1
        #self.__class__.__name__でクラス名が取得できる。



class HybridHunterA(AbstractHunter):

        def __init__(self, x, y, N, L,number):
            super().__init__(x, y, N, L, number)
            self.state = "exploring"

        def change_dir(self):
            mushroom_dir = self.search_mushroom()
            if mushroom_dir != -1:
                self.vx, self.vy = mushroom_dir
            else:

                self.set_state()
```

```
            dirs = [(1, 1), (0, 1), (-1, 1), (-1, 0), (-1,-1), (0, -1), (1, -1), (1, 0)]
            ind = dirs.index((self.vx, self.vy))
            u = random.random()
            if self.state == "exploring":
                if u<0.1:
                    newInd = (ind+1)%8
                elif u<0.2:
                    newInd = (ind-1)%8
                else:
                    newInd = ind

            elif self.state == "searching":
                if u<0.5:
                    newInd = (ind+1)%8
                else:
                    newInd = ind

            self.vx, self.vy = dirs[newInd]




def set_state(self):#状態を(state)を変更する
    if random.random() < 0.05:
        if self.state == "searching":
            self.state = "exploring"
        elif self.state == "exploring":
            self.state = "searching"

def render(self):
    if self.state == "exploring":
        canvas.create_rectangle(self.x*8, self.y*8,
                                self.x*8+8, self.y*8+8, fill ="pink", outline ="pink")
    elif self.state == "searching":
```

```
            canvas.create_rectangle(self.x*8, self.y*8,
                                    self.x*8+8, self.y*8+8, fill ="green", outline ="green")


class RandomHunter (AbstractHunter):
    def change_dir(self):
        mushroom_dir = self.search_mushroom()
        if mushroom_dir != -1:
            self.vx, self.vy = mushroom_dir
        else:
            dirs = [(1, 1), (0, 1), (-1, 1), (-1, 0), (-1,-1), (0, -1), (1, -1), (1, 0)]
            self.vx, self.vy = random.choice(dirs)




    def render(self):
        canvas.create_rectangle(self.x*8, self.y*8,
                                    self.x*8+8, self.y*8+8, fill ="purple", outline ="purple")


class ForwardHunter(AbstractHunter):
    def change_dir(self):
        mushroom_dir = self.search_mushroom()
        if mushroom_dir != -1:
            self.vx, self.vy = mushroom_dir
        else:
            dirs = [(1, 1), (0, 1), (-1, 1), (-1, 0), (-1,-1), (0, -1), (1, -1), (1, 0)]
            ind = dirs.index((self.vx, self.vy))
            u = random.random()
            if u<0.05:
                newInd = (ind+1)%8
            elif u<0.1:
                newInd = (ind-1)%8
            else:
                newInd = ind
            self.vx, self.vy = dirs[newInd]
```

```
    def render(self):
        canvas.create_rectangle(self.x*8, self.y*8,
                                self.x*8+8, self.y*8+8, fill ="orange", outline ="orange")

world = World()

world.start(1000)
```