

# **DASS Assignment-1**

Developer : Dolton Fernandes  
Roll No : 2018111007

## **1. Rules of the game**

- 1) Game runs for a fixed amount of time which when exceeded make you lose the game
- 2) Many coins come on the way which when collected fetches you 5pts
- 3) Kill enemies or obstacles to gain 2pts
- 4) You get 3 lives
- 5) Many Power(ups/downs) like shield , Magnets , Boost come on the way which can be collected
- 6) In the end is the boss level. The boss shoots snow balls which need to be dodged
- 7) Ultimate Dragon powerup can be used between the game
- 8) Boss follows the mandalorian along the vertical axis
- 9) To kill Boss, it needs to be hit by 10 bullets

## **2. Description of Main Classes**

- 1) Board : Contains the main 2d numpy grid, timer, description, rules of the game.
- 2) Game : This is where the main loop of the game runs and all other classes are called.

## **3. Instructions to play**

Run the following command in the directory :

- 1) python3 main.py
- 2) Use the following controls:
  - a) w,a,d to move up, move left, move right respectively.
  - b) 'space' to use shield
  - c) l to shoot bullets
  - d) Press Q to quit the game
  - e) Press O to activate Ultimate Dragon which can be used only once in the entire game

## **4. Requirements**

- 1) Python3  
colorama==0.4.3  
numpy==1.18.1

## How the OOPS concepts are used :

### 1) Inheritance:

```
class Person:

    def __init__(self):
        self._r = ROWS
        self._c = COLUMNS
        self._offset = 0
        self._x = 0
        self._y = 0
        self._name = ""

class Parent_Func:

    def get_priority(self):
        return self._priority

    def get_x(self):
        return self._x

    def get_y(self):
        return self._y

    def get_rows(self):
        return self._rows

    def get_columns(self):
        return self._columns

    def get_image(self):
        return self._image

    def get_name(self):
        return self._name

    def get_r(self):
        return self._r

    def get_c(self):
        return self._c

    def get_delete(self):
        return self._delete

    def delt(self):
        self._delete = 1

class Enemy(Person,Parent_Func):

    def enemy_init(self,x,y):
        self._x = x
        self._y = y
        self._rows = len(ascii_enemy)
        self._columns = len(ascii_enemy[0])
        self._image = ascii_enemy
        self._vsp = 0
        self._hsp = ENEMY_SPEED
        self._name = "enemy"
        self._delete = 0
        self._priority = priorities["enemy"]

    def move(self):
        self._y -= self._hsp
        if self._y <= -10:
            self._delete = 1

    def chk(self,obj):
        arr = []
        for i in range(obj.get_r()):
            arr.append([0]*obj.get_c())

        for j in range(obj.get_rows()):
            for k in range(obj.get_columns()):
                if obj.get_y()+k>=0 and obj.get_y()+k<obj.get_c() and obj.get_image()[j][k]!=' ':
                    arr[obj.get_x()+j][obj.get_y()+k] = 1

        for j in range(self._rows):
            for k in range(self._columns):
                if self._x+j>=0 and self._x+j<obj.get_r() and self._y+k>=0 and self._y+k<obj.get_c() and self._image[j][k]!=' ':
                    if arr[self._x+j][self._y+k]==1:
                        return 1

        return 0
```

As you can see in this example the class Enemy is inherited from two classes Person and Parent\_Func.

Some variables are kept protected because private variables can't be accessed in an inherited class.

## 2) Polymorphism :

```
class Bars(Parent,Parent_Func):

    def __init__(self,x,y,p):
        self._rows = len(ascii_bars[p])
        self._columns = len(ascii_bars[p][0])
        self._image = ascii_bars[p]
        self._x = x
        self._y = y
        self._delete = 0
        self._name = "bar"
        self._priority = priorities["bar"]

    def move(self):
        self._y -= 1
        if self._y == -20:
            self._delete = 1
```

This the class for the bars (obstacles)

According to the value of p given during initialisation of the object of this class ,one of the 4 types of bars ( Vertical , Horizontal , Upper left to Lower right Diagonal , Upper right to Lower left Diagonal ) is made.

## 3) Encapsulation :

Classes and object are being used and made so encapsulation is done.

#### 4) Abstraction :

```
def move_right(self):
    if self._y+self._hsp<self._c:
        self._y+=self._hsp

def move_left(self):
    if self._y-self._hsp>=0:
        self._y-=self._hsp

def move_down(self,t):
    if self._uplast - t <= 1:
        self._interval2 = 0
        return 0
    if self._interval2 < 10:
        self._interval2 += 0.3
    if self._x+self._vsp+int(self._interval2)+9<self._r:
        self._x+=self._vsp+int(self._interval2)
    else:
        self._x = 40 - self._par
    return 1

def move_up(self):
    if self._x-self._vsp-self._interval1>4:
        self._x-=self._vsp + int(self._interval1)
    else:
        self._x = 6

def shoot(self,arr):
    if self._can_shoot == 0:
        arr.append(ball(self._x+1,self._y+5,0))
        self._can_shoot += (self._fr*self._shoot_time)

def add_shield(self,arr,t):
    if arr[0]._shield_time >= 60:
        arr.append(Shield(t))
```

This is the Mandalorian class.

We can see that functions like move left, move right etc are made to stow away inner details from the end user.