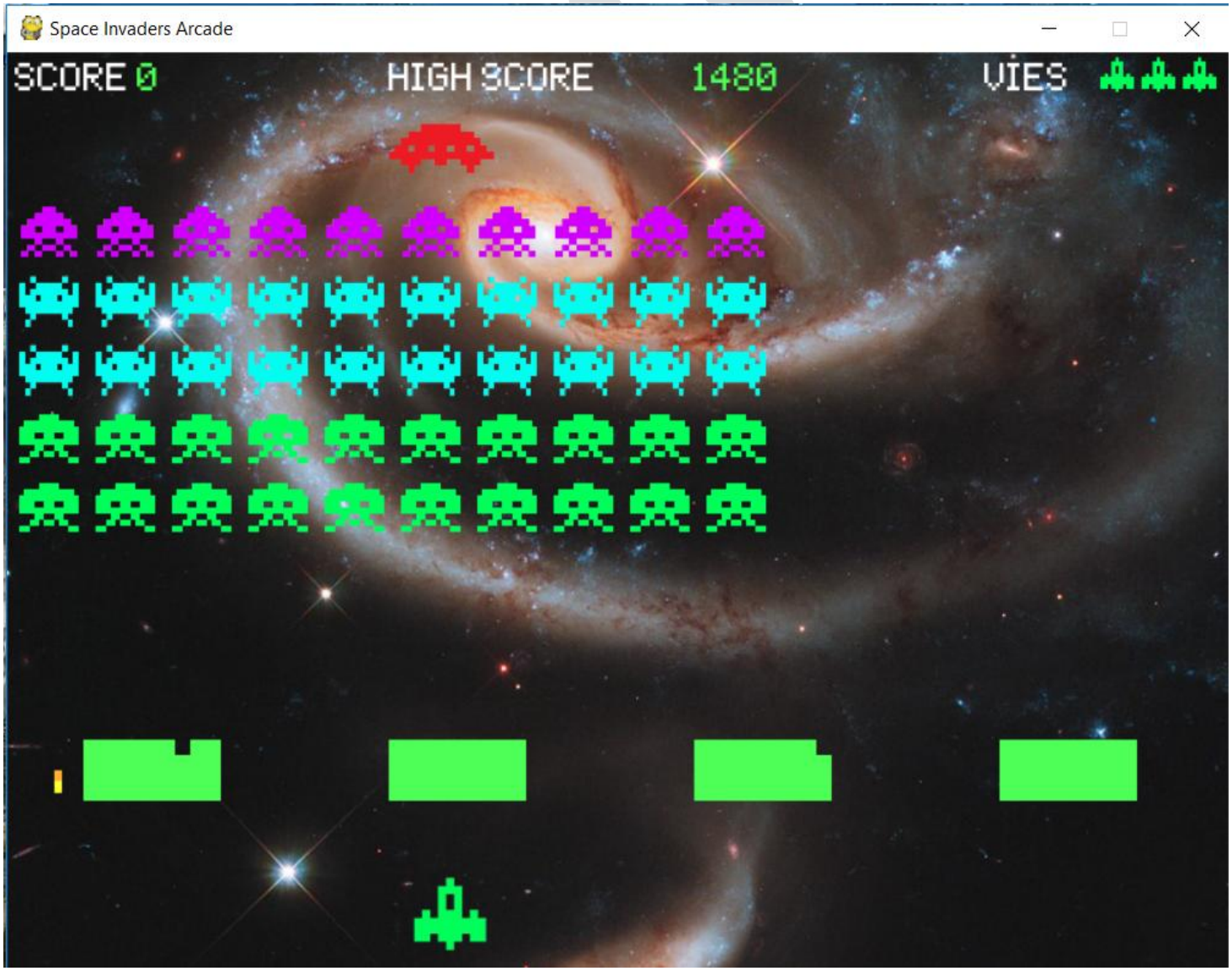
2018

BOURDON Antoine-Alexis
N° Etudiant : 20172105
L2MI

[SPACE INVADERS - DESCRIPTIF DES DIFFERENTES FONCTIONS UTILISEES]

Programme (Projet) en PYTHON



Afin de concrétiser le projet destiné à conclure le semestre 3 d'Algorithmique et Programmation 3, il a été décidé, par nos enseignants, de réaliser le jeu SPACE INVADERS.

Le rapport sur le descriptif des différentes fonctions utilisées dans le jeu « Space Invaders » va vous être présenté ainsi :

Sommaire

I) PROJET

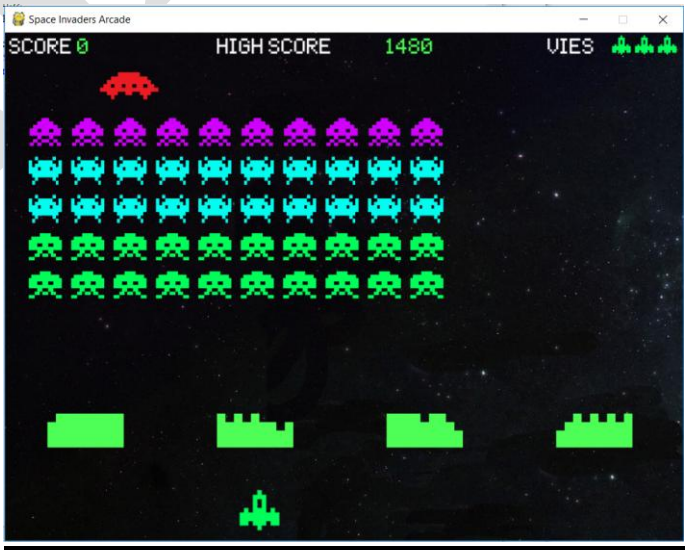
a) Qu'est que le Space Invaders ?[1]

Space Invaders スペースインベーダー (*Supēsu Inbēdā*[2]) est un jeu vidéo développé par la société japonaise Taito, sorti en 1978 sur borne d'arcade. Il s'agit d'un *shoot them up* fixe.
Le principe est de détruire des vagues d'aliens au moyen d'un canon laser en se déplaçant horizontalement sur l'écran.

b) A quoi ressemble t-il ?[1]

Space Invaders est un *shoot them up* fixe en deux dimensions. Le joueur contrôle un canon laser qu'il peut déplacer horizontalement, au bas de l'écran. Dans les airs, des rangées d'aliens se déplacent latéralement tout en se rapprochant progressivement du sol et en lançant des missiles. L'objectif est de détruire avec le canon laser une vague ennemie, qui se compose de cinq rangées de onze aliens chacune, avant qu'elle n'atteigne le bas de l'écran. Le joueur gagne des points à chaque fois qu'il détruit un envahisseur. Le jeu n'autorise qu'un tir à la fois et permet d'annuler ceux des ennemis en tirant dessus. La vitesse et la musique s'accélèrent au fur et à mesure que le nombre d'aliens diminue. L'élimination totale de ces derniers amène une nouvelle vague ennemie plus difficile, et ce indéfiniment. Le jeu ne se termine que lorsque le joueur perd, ce qui en fait le premier jeu sans fin.

Les aliens tentent de détruire le canon en tirant dessus pendant qu'ils s'approchent du bas de l'écran. S'ils l'atteignent ou arrivent jusqu'au sol, ils ont réussi leur invasion et le jeu est fini. De temps en temps, un vaisseau spatial apparait tout en haut de l'écran et fait gagner des points bonus s'il est détruit. Quatre bâtiments destructibles permettent au joueur de se protéger des tirs ennemis. Ces défenses se désintègrent progressivement sous l'effet des projectiles adverses et de ceux du joueur. Le nombre de bâtiments n'est pas le même d'une version à l'autre.



[1] Source wikipedia : https://fr.wikipedia.org/wiki/Space_Invaders

c) La division du travail

Comme nous pouvons le voir sur les images et aussi d'après le descriptif de Wikipedia sur le Space Invaders ; Il nous faudrait créer des classes :

La classe Vaisseau représente le canon du joueur.
La classe Laser représentant les lasers des ennemies et le laser du joueur.
La classe ennemie qui représente un seul ennemi.
La classe GroupEnnemies représente l'ensemble des ennemies.
La classe Bouclier pour protéger notre joueur.
La classe Vie indiquant au joueur le nombre de vies restant.
La classe Text qui affiche les éléments textuels du jeu.
Et la classe SpaceInvaders qui est le cœur du programme car gère les autres classes.

II) Les Constantes

Avant de s'attaquer au jeu. Il est important de définir des variables globales telles que les fichiers images, sons, police, les couleurs et la taille de l'écran pour le jeu.

```
###########################################

#CONSTANTES

##DOSSIERS
RACINE = abspath(dirname(__file__))
DOSSIER_POLICE = RACINE + '/fonts/'
DOSSIER_IMGS = RACINE + '/images/'
DOSSIER_SONS = RACINE + '/sons/'

#COULEURS
NOIR=(0,0,0)
VERT = (78, 255, 87)
BLEU = (80, 255, 239)
VIOLET = (203, 0, 255)
ROUGE = (237, 28, 36)
JAUNE = (241, 255, 0)
BLANC = (255, 255, 255)

surface = display.set_mode((800, 600))
FONT = DOSSIER_POLICE + 'space_invaders.ttf'
NOMS_IMGS = ['vaisseau_joueur', 'vaisseau_mystere', 'ennemi1_1', 'ennemi1_2', 'en
IMAGES = {name: image.load(DOSSIER_IMGS + '{}.png'.format(name)).convert_alpha()
          for name in NOMS_IMGS}

###########################################
```

III) La classe Vaisseau

La classe Vaisseau modélise le sprite du joueur sur l'écran.
Son constructeur, associe l'image du vaisseau joueur, place son vaisseau à la position x=375 et y=540 et donne une vitesse de 5 pour chaque déplacement de celui-ci.

```
class Vaisseau(sprite.Sprite):
    '''Classe qui modélise le sprite du joueur sur l'écran
    '''
    def __init__(self):
        '''Vaisseau -> Vaisseau(modif)
        Associe l'image du vaisseau du joueur, place le vaisseau à la position (375,540) et donne une vitesse de 5
        '''
        sprite.Sprite.__init__(self)
        self.image = IMAGES['vaisseau_joueur']
        self.rect = self.image.get_rect(topleft=(375, 540))
        self.vitesse = 5
```

La fonction update, met à jour la position du joueur quand celui-ci appuie soit sur la flèche droite ou sur la touche gauche de son clavier.

```
def update(self, touches, *args):
    '''Vaisseau(modif) , KEY, *args(groupsennemies) -> None
    Fonction mettant à jour la position du Vaisseau Joueur'''
    if touches[K_LEFT] and self.rect.x > 10:
        self.rect.x -= self.vitesse
    if touches[K_RIGHT] and self.rect.x < 740:
        self.rect.x += self.vitesse
    jeu.surface.blit(self.image, self.rect)
```

IV) La classe Laser

La classe Laser modélise le laser du vaisseau qui tire.

Son constructeur, à besoin d'information comme la position du vaisseau tirant, la direction pour le déplacement du laser dans l'écran, le nom du sprite à donner au laser et aussi savoir qu'elle est la nature du laser (centre ou gauche/droite quand il y a l'amélioration).

```
class Laser(sprite.Sprite):
    '''Class qui modelise le laser des vaisseaux en fonction de sa direction et
    des coordonnées du vaisseau émetteur du tire, et si le tire est de centre ou
    gauche/droite (pour l'amélioration)
    '''
    def __init__(self, xpos, ypos, direction, vitesse, nom_fichier, side):
        '''Initialise le laser avec le nom de l'image à afficher (laser pour le joueur et ennemilaser
        pour les ennemies), la direction du tir (1 pour les ennemies et -1 pour joueur), vitesse du tir
        (de base ennemies 5 et joueur 15), side pour savoir si c'est un laser de centre ou gauche/droite
        Laser, xpos(int), ypos(int), direction(int), vitesse(int), nom_fichier(str), side(str) --> Laser(modif)
        '''
        sprite.Sprite.__init__(self)
        self.image = IMAGES[nom_fichier]
        self.rect = self.image.get_rect(topleft=(xpos, ypos))
        self.vitesse = vitesse
        self.direction = direction
        self.side = side
        self.nom_fichier = nom_fichier
```

La fonction update met à jour dans l'écran la position du laser. Si celui-ci est dans une zone non visible, il supprime la classe laser rattaché au laser pour permettre au joueur de tirer de nouveau.

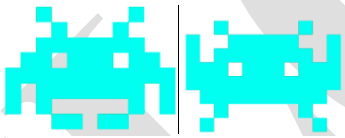
```
def update(self, touches, *args):
    '''Met a jour la modelisation du laser et supri
    Laser(kill), KEY, *argv(groupennemies) -> None
    '''
    jeu.surface.blit(self.image, self.rect)
    self.rect.y += self.vitesse * self.direction
    if self.rect.y < 15 or self.rect.y > 600:
        self.kill()
```

V) La classe Ennemi

La classe Ennemi modélise et initialise les éléments d'un ennemi.
Représenté par son image, sa position en ligne et colonne, la direction de celui-ci à droite et à gauche, le nombre de mouvements à effectuer avant collision et le temps entre chaque mouvement.

```
class Ennemi(sprite.Sprite):
    '''Classe qui modélise et initialise les é
    Représenté par son image, sa position en l
    direction de celui ci à droite et à gauche
    à effectuer avant collisionet le temps ent
    def __init__(self, ligne, colonne):
        '''Initialise l'ennemi.
        Ennemi, ligne(int), colonne(int) -> En
        '''
        sprite.Sprite.__init__(self)
        self.ligne = ligne
        self.colonne = colonne
        self.images = []
        self.charger_imgs()
        self.indice = 0
        self.image = self.images[self.indice]
        self.rect = self.image.get_rect()
        self.direction = 1
        self.mouv_droite = 30
        self.mouv_gauche = 30
        self.nbre_mouv = 15
        self.tmp_mouv = 600
        self.timer = time.get_ticks()
```

La fonction charger_imgs permet de charger les deux images correspondant à un ennemi.



```
def charger_imgs(self):
    '''Charge l'image de l'ennemi (un ennemi à deux images pour le mouvement)
    ennemi(modif) -> None
    '''
    images = {0: ['1_2', '1_1'],1: ['2_2', '2_1'],2: ['2_2', '2_1'],3: ['3_1', '3_2'],4: ['3_1', '3_2']}
    img1, img2 = (IMAGES['ennemi{}'.format(img_num)] for img_num in images[self.ligne])
    self.images.append(transform.scale(img1, (40, 35)))
    self.images.append(transform.scale(img2, (40, 35)))
```

La fonction update met à jour la position de l'ennemi et aussi ses images pour donner l'impression de mouvement.

```
def update(self, touches, tmp_save, enemies):
    '''Met a jour l'emplacement des enemies
    ennemi(modif), tmp_save(int), enemies(groupEnnemies) -> None
    '''
    if tmp_save - self.timer > self.tmp_mouv:
        if self.direction == 1:
            mouv_max = self.mouv_droite + enemies.add_mouv_drt
        else:
            mouv_max = self.mouv_gauche + enemies.add_mouv_ghe

        if self.nbre_mouv >= mouv_max:
            if self.direction == 1:
                self.mouv_gauche = 30 + enemies.add_mouv_drt
            elif self.direction == -1:
                self.mouv_droite = 30 + enemies.add_mouv_ghe
            self.direction *= -1
            self.nbre_mouv = 0
            self.rect.y += 35
        elif self.direction == 1:
            self.rect.x += 10
            self.nbre_mouv += 1
        elif self.direction == -1:
            self.rect.x -= 10
            self.nbre_mouv += 1

        self.indice += 1
        if self.indice >= len(self.images):
            self.indice = 0
        self.image = self.images[self.indice]

        self.timer += self.tmp_mouv

    jeu.surface.blit(self.image, self.rect)
```

VI) La classe GroupEnnemies

La classe GroupEnnemies modélise l'ensemble des ennemies d'un sprite.group. Les ennemies sont représentés dans une liste de liste chaque ennemie vivant vaut 0 et mort 1. Quand une colonne disparaît en extrémité un mouvement en plus est nécessaire aux ennemies pour être à l'extrémité de l'écran.

```
class GroupEnnemies(sprite.Group):
    '''Modelise le group de tout les ennemies
    '''
    def __init__(self, colonnes, lignes):
        '''Initialise le group en list [[0,0,0,0,0],[0,0,0,0,0]...]
        GroupEnnemies, colonnes (int), lignes(int) -> GroupEnnemies
        '''
        sprite.Group.__init__(self)
        self.enemies = [[0] * colonnes for _ in range(lignes)]
        self.colonnes = colonnes
        self.lignes = lignes
        self.add_mouv_ghe = 0
        self.add_mouv_drt = 0
        self._colonnes_vivantes = list(range(colonnes))
        self._colonnes_ghe_vivantes = 0
        self._colonnes_drt_vivantes = colonnes - 1
        self._colonnes_ghe_tuees = 0
        self._colonnes_ghe_tuees = 0
```

La fonction add permet de rajouter des sprites d'ennemis dans le sprite.Group et fait correspondre à la liste de liste des ennemies.

```
def add(self, *sprites):
    '''Ajoute un sprite ennemi dans le GroupEnnemies
    GroupEnnemies(modif), sprites(groups) -> None
    '''
    super(sprite.Group, self).add(*sprites)

    for s in sprites:
        self.enemies[s.ligne][s.colonne] = s
```

La fonction est_colonne_morte, vérifie si une colonne est morte ou non.

```
def est_colonne_morte(self, colonne):
    '''fonction qui dit si une colonne est morte ou non
    GroupEnnemies -> Bool
    '''
    for ligne in range(self.lignes):
        if self.enemies[ligne][colonne]:
            return False
    return True
```

La fonction ennemies_choisit, choisit l'ennemi qui utilisera le laser.

```
def enemies_choisit(self):
    '''fonction qui choisit un ennemi pour tirer avec son laser
    GroupEnnemies -> ennemi sprite
    '''
    random_indice = randint(0, len(self._colonnes_vivantes) - 1)
    col = self._colonnes_vivantes[random_indice]
    for ligne in range(self.lignes, 0, -1):
        ennemi = self.enemies[ligne - 1][col]
        if ennemi:
            return ennemi
    return None
```

La fonction kill, vérifie si un ennemi est mort.

```
def kill(self, ennemi):
    '''fonction qui vérifie si un ennemi est mort
    GroupEnnemies(modif) -> None'''
    # sur les appels à double coup pour le même ennemi, alors vérifiez avant
    if not self.enemies[ennemi.ligne][ennemi.colonne]:
        return  None

    self.enemies[ennemi.ligne][ennemi.colonne] = None
    estColonneMorte = self.est_colonne_morte(ennemi.colonne)
    if estColonneMorte:
        self._colonnes_vivantes.remove(ennemi.colonne)

    if ennemi.colonne == self._colonnes_drt_vivantes:
        while self._colonnes_drt_vivantes > 0 and estColonneMorte:
            self._colonnes_drt_vivantes -= 1
            self._colonnes_ghe_tuees += 1
            self.add_mouv_drt = self._colonnes_ghe_tuees * 5
            estColonneMorte = self.est_colonne_morte(self._colonnes_drt_vivantes)

    elif ennemi.colonne == self._colonnes_ghe_vivantes:
        while self._colonnes_ghe_vivantes < self.colonnes and estColonneMorte:
            self._colonnes_ghe_vivantes += 1
            self._colonnes_ghe_tuees += 1
            self.add_mouv_ghe = self._colonnes_ghe_tuees * 5
            estColonneMorte = self.est_colonne_morte(self._colonnes_ghe_vivantes)
```

VII) La classe Bouclier

La classe Bouclier modélise un bouclier. Représenté par une taille qui correspondra à un carrer du bouclier, ligne et colonne indiquent sur quelle ligne et quelle colonne dessinée les carrer du bouclier.

```
class Bouclier(sprite.Sprite):
    '''classe qui modélise un bouclier représenter par un rectangle
    '''
    def __init__(self, taille, couleur, ligne, colonne):
        '''Constructeur avec les valeur de la taille du bouclier, son emplacement et sa couleur
        Bouclier, taille(int), couleur(tuple(int,int,int)), ligne(int), colonne(int) -> Bouclier
        '''
        sprite.Sprite.__init__(self)
        self.hauteur = taille
        self.largeur = taille
        self.couleur = couleur
        self.image = Surface((self.largeur, self.hauteur))
        self.image.fill(self.couleur)
        self.rect = self.image.get_rect()
        self.ligne = ligne
        self.colonne = colonne
```

La fonction update dessine le bouclier sur l'écran.

```
def update(self, touches):
    '''fonction qui dessine les boucliers sur surface
    Bouclier, touches (pygame surface ) -> None
    '''
    jeu.surface.blit(self.image, self.rect)
```

VIII) La classe vaisseau_mys

La classe vaisseau_mys modélise le vaisseau rouge donnant un nombre aléatoire pour le score.

Son constructeur initialise le vaisseau mystère en lui donnant l'image associée à ce vaisseau, sa direction (de la gauche vers la droite), sa ligne (donc sa coordonnée en x), tous les combien de temps doit apparaitre ce vaisseau.

```
class vaisseau_mys(sprite.Sprite):
    '''classe qui modelise le vaisseau mystère
    '''
    def __init__(self):
        '''Constructeur qui initialise le vaisseau mystère en lui donnant l'image as
        vaisseau_mys -> vaisseau_mys
        '''
        sprite.Sprite.__init__(self)
        self.image = IMAGES['vaisseau_mystere']
        self.image = transform.scale(self.image, (75, 35))
        self.rect = self.image.get_rect(topleft=(-80, 45))
        self.ligne = 5
        self.tmp_mouv = 25000
        self.direction = 1
        self.timer = time.get_ticks()
        self.vaisseau_mystere = mixer.Sound(DOSSIER_SONS + 'vaisseau_mystere.wav')
        self.vaisseau_mystere.set_volume(0.3)
        self.jouerSon = True
```

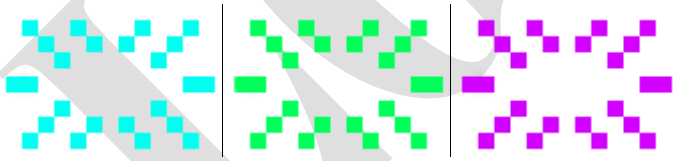
La fonction update met à jour la position du vaisseau mystère et met aussi à jour la musique, en diminuant le niveau sonore des autres sons quand celui-ci est sur l'écran.

```
def update(self, touches, tmp_save, *args):
    '''Met a jour la position du vaisseau mystère
    vaisseau_mys(modif), touches(list(liste des touches appuyées)), t
    '''
    timerReinit = False
    tmp_passe = tmp_save - self.timer
    if tmp_passe > self.tmp_mouv:
        if (self.rect.x < 0 or self.rect.x > 800) and self.jouerSon:
            self.vaisseau_mystere.play()
            self.jouerSon = False
        if self.rect.x < 840 and self.direction == 1:
            self.vaisseau_mystere.fadeout(4000)
            self.rect.x += 2
            jeu.surface.blit(self.image, self.rect)
        if self.rect.x > -100 and self.direction == -1:
            self.vaisseau_mystere.fadeout(4000)
            self.rect.x -= 2
            jeu.surface.blit(self.image, self.rect)

    if self.rect.x > 830:
        self.jouerSon = True
        self.direction = -1
        timerReinit = True
    if self.rect.x < -90:
        self.jouerSon = True
        self.direction = 1
        timerReinit = True
    if tmp_passe > self.tmp_mouv and timerReinit:
        self.timer = tmp_save
```

IX) La classe Explosions

La classe Explosions modélise les explosions que subissent les vaisseaux.



A chaque tir qui touche, la classe Explosions est appelée. Son constructeur possède alors les coordonnées x et y du vaisseau touché, la ligne du vaisseau touché pour savoir sa couleur, si le vaisseau touché est le joueur ou non, si le vaisseau est le vaisseau mystère ou non.

```
class Explosion(sprite.Sprite):
    '''Classe qui modélise les Explosions que subit les vaisseaux
    '''
    def __init__(self, xpos, ypos, ligne, vaisseau, vaisseau_Mystere, score):
        '''
        A chaque tir qui touche le constructeur ce mets
        Explosion, xpos(int), ypos(int), ligne(int), vaisseau (bool(pour savoir
        '''
        sprite.Sprite.__init__(self)
        self.estvaisseau_Mystere = vaisseau_Mystere
        self.estvaisseau = vaisseau
        if vaisseau_Mystere:
            self.text = Text(FONT, 20, str(score), BLANC, xpos + 20, ypos + 6)
        elif vaisseau:
            self.image = IMAGES['vaisseau_joueur']
            self.rect = self.image.get_rect(topleft=(xpos, ypos))
        else:
            self.ligne = ligne
            self.charger_image()
            self.image = transform.scale(self.image, (40, 35))
            self.rect = self.image.get_rect(topleft=(xpos, ypos))
            jeu.surface.blit(self.image, self.rect)

        self.timer = time.get_ticks()
```

La fonction charger_images permet de charger l'image de l'explosion en fonction de la ligne qui donne l'image de couleur à prendre pour l'explosion.

```
def charger_image(self):
    '''Explosion(modif)-> None
    fonction qui charge les images d'explosion en fonction de leur repr
    '''
    imgCouleurs = ['violet', 'bleu', 'bleu', 'vert','vert']
    self.image = IMAGES['explosion{}'.format(imgCouleurs[self.ligne])]
```

La fonction update affiche l'animation de l'explosion en fonction du vaisseau touché et supprime la classe explosion rattaché à l'explosion.

```
def update(self, touches, tmp_save):
    '''Explosion, touches(list), tmp_save(int) -> None
    affiche l'animation d'explosion
    '''
    tmp_passe = tmp_save - self.timer
    if self.estvaisseau_Mystere:
        if tmp_passe <= 200:
            self.text.draw(jeu.surface)
        elif 400 < tmp_passe <= 600:
            self.text.draw(jeu.surface)
        elif tmp_passe > 600:
            self.kill()
    elif self.estvaisseau:
        self.kill()
    else:
        if tmp_passe <= 100:
            jeu.surface.blit(self.image, self.rect)
        elif 100 < tmp_passe <= 200:
            self.image = transform.scale(self.image, (50, 45))
            jeu.surface.blit(self.image,(self.rect.x - 6, self.rect.y - 6))
        elif tmp_passe > 400:
            self.kill()
```

X) La classe Vie

La classe Vie modélise la vie du joueur sous forme d'image.
Son constructeur prend les coordonnées x et y de l'emplacement où seront les vies à afficher.

```
class Vie(sprite.Sprite):
    '''Classe qui modélise la vie du joueur sous forme d'image
    '''
    def __init__(self, xpos, ypos):
        '''Constructeur avec les valeur x et y de l'emplacement des images de la vie
        Vie, xpos(int), ypos(int) -> Vie
        '''
        sprite.Sprite.__init__(self)
        self.image = IMAGES['vaisseau_joueur']
        self.image = transform.scale(self.image, (23, 23))
        self.rect = self.image.get_rect(topleft=(xpos, ypos))
```

La fonction update affiche le nombre de vies sur l'écran.

```
def update(self, touches, *args):
    '''Fonction affichant le nombre de vie
    Vie, touches(list(liste des touches appuyé
    '''
    jeu.surface.blit(self.image, self.rect)
```

XI) La classe Text

La classe Text modélise le texte à afficher sur l'écran.
Son constructeur prend les valeurs de x et y pour l'emplacement du texte, la couleur de celui-ci, la police et le message à afficher.

```
class Text(object):
    '''Classe modélisant le texte à afficher
    '''
    def __init__(self, textFont, taille, message, couleur, xpos,
        '''Constructeur avec les valeur x et y pour le texte, la
        Text, textFont(str(emplacement de la police)), taille(int
        '''
        self.font = font.Font(textFont, taille)
        self.surface = self.font.render(message, True, couleur)
        self.rect = self.surface.get_rect(topleft=(xpos, ypos))
```

La fonction draw affiche le texte.

```
    def draw(self, surface):
        '''Fonction qui affiche le text
        Text -> None
        '''
        surface.blit(self.surface, self.rect)
```

XII) La classe SpaceInvaders

```
class SpaceInvaders(object):
    '''Classe Principale du jeu
    '''
    def __init__(self):
        '''Constructeur qui initialise les élèments principaux du jeu
        SpaceInvaders -> SpaceInvaders
        '''
        mixer.pre_init(44100, -16, 1, 4096)
        init()
        self.nom_fenetre = display.set_caption('Space Invaders Arcade')
        self.surface = surface
        self.background = image.load(DOSSIER_IMGS + 'background.jpg').convert()
        self.startjeu = False
        self.surface_ecran = 0
        self.gameOver = False
        # valeur initiale pour un nouveau jeu
        self.positionEnnemiDefaut = 65
        # Compteur pour la position de départ de l'ennemi (augmenté chaque nouveau tour)
        self.compteurPositionActive = self.positionEnnemiDefaut
        # Position de départ actuelle de l'ennemi
        self.positionEnnemi = self.compteurPositionActive
        self.hi_score = int(open("hi_score","r").read())
        self.coins = 0
```

La classe SpaceInvaders est la classe principale, gérant toutes les composantes nécessaires au bon fonctionnement du jeu. Cette classe s'occupe de remettre à zéro les composantes du jeu, de créer le group des boucliers (4 boucliers), affichage des vies restantes à l'écran, remise à zéro des vies, le stockage de tous les sons, sauf le son de « MIT_Concert_Choir_-_01_-_O_Fortuna », lire les sons des ennemies, le stockage des textes à afficher, la création des ennemies, le tire des ennemies, le score, les collisions et la création de notre vaisseau en cas de mort.

```
    def reset(self, score, vies, nouveauJeu=False):
        '''Fonction qui remet tout à zero sauf les bouclier, le score et les vies si le jeu a été gagné
        SpaceInvaders(modif), score (int), vies(int), nouveauJeu(bool) -> None
        '''
        self.joueur = Vaisseau()
        self.joueurGroup = sprite.Group(self.joueur)
        self.explosionsGroup = sprite.Group()
        self.lasers = sprite.Group()
        self.vaisseau_Mystere= vaisseau_mys()
        self.mysteryGroup = sprite.Group(self.vaisseau_Mystere)
        self.lasersEnnemi = sprite.Group()
        self.reset_vies(vies)
        self.positionEnnemi = self.compteurPositionActive
        self.creation_ennemies()
        self.hi_score = int(open("hi_score","r").read())
        # Créer uniquement des bloqueurs sur un nouveau jeu, pas un nouveau tour
        if nouveauJeu:
            self.ensembleBoucliers = sprite.Group(self.creation_boucliers(0), self.creation_boucliers(1), self.creation_boucliers(2), self.creation_boucliers(3))
        self.touches = key.get_pressed()
        self.clock = time.Clock()
        self.timer = time.get_ticks()
        self.noteTimer = time.get_ticks()
        self.vaisseauTimer = time.get_ticks()
        self.score = score
        self.vies = vies
        self.stockage_sons()
        self.stockage_text()
        self.creerNouveauVaisseau = False
        self.VaisseauVie = True
```

La fonction reset met tout à zéro sauf les boucliers, le score et les vies si le jeu a été gagné par le joueur.

```
def creation_boucliers(self, number):
    '''Fonction qui creer le group de bouclier
    SpaceInvaders -> Group (36 sprites ici)
    '''
    bouclierGroup = sprite.Group()
    for ligne in range(4):
        for colonne in range(9):
            bouclier = Bouclier(10, VERT, ligne, colonne)
            bouclier.rect.x = 50 + (200 * number) + (colonne * bouclier.largeur)
            bouclier.rect.y = 450 + (ligne * bouclier.hauteur)
            bouclierGroup.add(bouclier)
    return bouclierGroup
```

La fonction création_bouclier créé le group de bouclier (ici 4 boucliers).

```
def reset_vies_sprites(self):
    '''Fonction qui creer les Vies du joueur (sprite) avec leur positi
    SpaceInvaders(modif) -> None
    '''
    self.vie1 = Vie(715, 3)
    self.vie2 = Vie(742, 3)
    self.vie3 = Vie(769, 3)

    if self.vies == 3:
        self.viesGroup = sprite.Group(self.vie1, self.vie2, self.vie3)
    elif self.vies == 2:
        self.viesGroup = sprite.Group(self.vie1, self.vie2)
    elif self.vies == 1:
        self.viesGroup = sprite.Group(self.vie1)
```

La fonction reset_vies_sprites, créer les Vies du joueur (sprite) avec leur position sur la surface et définie combien de sprites dans le ViesGroup.

```
def reset_vies(self, vies):
    '''Fonction qui remet à zer
    SpaceInvaders(modif) -> Non
    '''
    self.vies = vies
    self.reset_vies_sprites()
```

La fonction reset_vies a pour but de mettre le nombre de vies du joueur à son maximum. C’est une fonction appelée au commencement d’une nouvelle partie.

```
def stockage_sons(self):
    '''Fonction qui stocke les fichiers sons et leur attribut du pygame.mixer dans le self.s
    SpaceInvaders(modif) -> None
    '''
    self.sons = {}
    for nom_son in ['tire', 'tire2', 'ennemimort', 'mystmort','vaisseau_joueur_explosion']:#
        self.sons[nom_son] = mixer.Sound(DOSSIER_SONS + '{}.wav'.format(nom_son))
        self.sons[nom_son].set_volume(0.2)

    self.musicNotes = [mixer.Sound(DOSSIER_SONS + '{}.wav'.format(i)) for i in range(4)]
    for son in self.musicNotes:
        son.set_volume(0.5)

    self.noteindice = 0
```

La fonction soctkage_sons ajoute dans le dictionnaire sons tous les sons joués. La variable noteindice permettra de jouer le bon son associé aux déplacements ennemis.

```
def lire_musique_ennemies(self, tmp_save):
    '''Fonction qui demarre le son des ennemies (soit
    SpaceInvaders(modif) -> None
    '''
    tmp_mouv = self.enemies.sprites()[0].tmp_mouv
    if tmp_save - self.noteTimer > tmp_mouv:
        self.note = self.musicNotes[self.noteindice]
        if self.noteindice < 3:
            self.noteindice += 1
        else:
            self.noteindice = 0

        self.note.play()
        self.noteTimer += tmp_mouv
```

La fonction lire_musique_ennemies, lance le son des ennemies à chacun de leurs déplacements, il y a donc 4 sons, comme sur la version arcade.

```
def stockage_text(self):
    '''Fonction qui stocke les différents texte à afficher avec leur cara
    SpaceInvaders(modif) -> None
    '''
    self.titleCoinText = Text(FONT, 50,'INSERT COIN', BLANC,201,225)
    self.titleCoin2Text = Text(FONT, 20,'Credits ', BLANC,640,580)
    self.titleCoin3Text = Text(FONT, 50,'PLAYS ', BLANC,300,225)
    self.titleText = Text(FONT, 50, 'Space Invaders', BLANC, 164, 155)
    self.gameOverText = Text(FONT, 50, 'GAME OVER', BLANC, 250, 270)
    self.ennemi1Text = Text(FONT, 25, '    =   10 pts', VERT, 368, 270)
    self.ennemi2Text = Text(FONT, 25, '    =  20 pts', BLEU, 368, 320)
    self.ennemi3Text = Text(FONT, 25, '    =  30 pts', VIOLET, 368, 370)
    self.ennemi4Text = Text(FONT, 25, '    =  ?????', ROUGE, 368, 420)
    self.scoreText = Text(FONT, 20, 'Score', BLANC, 5, 5)
    self.highscoreText = Text(FONT,20,'high score', BLANC, 250, 5)
    self.highscoreText2 = Text(FONT,20,str(self.hi_score), VERT, 450, 5)
    self.viesText = Text(FONT, 20, 'vies ', BLANC, 640, 5)
```

La fonction stockage_text, rassemble l'ensemble des textes qui seront affichés à l'écran avec la police utilisée, la taille, le label, la couleur, et les positions x et y.

```
def sortie(self,evt):
    '''Fonction vérifiant si l'utilisateur a demandé de partir
    SpaceInvaders, evt(list(liste des touches appuyées)) -> bool
    '''
    return evt.type == QUIT or (evt.type == KEYUP and evt.key == K_ESCAPE)
```

La fonction sortie, permet de fermer l'application dès que l'utilisateur à appuyer sur la croix, sur échappe ou sur Alt+F4.

```
def touche_appuyee(self):
    '''Fonction qui sauvegarde les touches appuyer et actionne l'action du laser si la touche espace et enfoncé
    SpaceInvaders(modif) -> None
    '''
    self.touches = key.get_pressed()
    for e in event.get():
        if self.sortie(e):
            sys.exit()
        if e.type == KEYDOWN:
            if e.key == K_SPACE:
                if len(self.lasers) == 0 and self.VaisseauVie:
                    if self.score < 1000:
                        laser = Laser(self.joueur.rect.x + 23, self.joueur.rect.y + 5, -1, 15, 'laser', 'center')
                        self.lasers.add(laser)
                        self.allSprites.add(self.lasers)
                        self.sons['tire'].play()
                    else: #BONUS SCORE
                        lasergauche = Laser(self.joueur.rect.x + 8,self.joueur.rect.y + 5, -1, 15, 'laser', 'gauche')
                        laserdroite = Laser(self.joueur.rect.x + 38,self.joueur.rect.y + 5, -1, 15, 'laser', 'droite')
                        self.lasers.add(lasergauche)
                        self.lasers.add(laserdroite)
                        self.allSprites.add(self.lasers)
                        self.sons['tire2'].play()
```

La fonction touche_appuyee, sauvegarde une liste des touches (appuyé = 1 et non appuyé = 0), et provoque l'action du laser avec la touche espace. (Un bonus de deux lasers apparaît au-dessus de 1000 de score).

```
def creation_ennemies(self):
    '''Fonction qui creer le group de ennemies
    SpaceInvaders -> Group (50 sprites ici)
    '''
    enemies = GroupEnnemies(10, 5)
    for ligne in range(5):
        for colonne in range(10):
            ennemi = Ennemi(ligne, colonne)
            ennemi.rect.x = 157 + (colonne * 50)
            ennemi.rect.y = self.positionEnnemi + (ligne * 45)
            enemies.add(ennemi)

    self.enemies = enemies
    self.allSprites = sprite.Group(self.joueur, self.enemies,self.viesGroup, self.vaisseau_Mystere)
```

La fonction création_ennemies, crée un group d'ennemies de 10 lignes et 5 colonnes. Ajoute chaque ennemi nouvellement créé dans le groupe.

```
def creation_ennemies_tire(self):
    '''Fonction qui fait tirer un ennemis à un interval de 0.7seconde
    SpaceInvaders(modif) -> None
    '''
    if (time.get_ticks() - self.timer) > 700:
        ennemi = self.enemies.enemies_choisit()
        if ennemi:
            self.lasersEnnemi.add(Laser(ennemi.rect.x + 14, ennemi.rect.y + 20, 1, 5,'ennemilaser', 'center'))
            self.allSprites.add(self.lasersEnnemi)
            self.timer = time.get_ticks()
```

La fonction creation_ennemies_tire, choisit à un intervalle de 0.7 seconde un ennemi pour tirer s'il reste au moins un ennemi en vie.

```
def calcul_score(self, ligne):
    '''Fonction qui calcul le score en fonction de la ligne touchée
    SpaceInvaders(modif) -> None
    '''
    scores = {0: 30, 1: 20,2: 20, 3: 10, 4: 10, 5: choice([50, 100, 150, 300])}
    score = scores[ligne]
    self.score += score
    return score
```

La fonction calcul_score, augmente le score actuel à chaque fois qu'un ennemi se fait toucher par un laser en fonction de la ligne où se situe l'ennemi.

```
def creation_menu_regle(self):
    '''Fonction qui creer le menu 2 donc celui des règles a
    SpaceInvaders -> None
    '''
    self.ennemi1 = IMAGES['ennemi3_1']
    self.ennemi1 = transform.scale(self.ennemi1, (40, 40))
    self.ennemi2 = IMAGES['ennemi2_2']
    self.ennemi2 = transform.scale(self.ennemi2, (40, 40))
    self.ennemi3 = IMAGES['ennemi1_2']
    self.ennemi3 = transform.scale(self.ennemi3, (40, 40))
    self.ennemi4 = IMAGES['vaisseau_mystere']
    self.ennemi4 = transform.scale(self.ennemi4, (80, 40))
    self.surface.blit(self.ennemi1, (318, 270))
    self.surface.blit(self.ennemi2, (318, 320))
    self.surface.blit(self.ennemi3, (318, 370))
    self.surface.blit(self.ennemi4, (299, 420))

    for e in event.get():
        if self.sortie(e):
            sys.exit()
        if e.type == KEYUP:
            self.surface_ecran = 3
            self.startjeu = True
```

La fonction creation_menu_regle, affiche le menu 2 correspondants au menu du nombre de points que nous fait gagner chaque ennemi.

```
def update_enemies_vitesse(self):
    '''fonction qui change la vitess
    SpaceInvaders(modif) -> None
    '''
    if 10 < len(self.enemies) <= 20:
        for ennemi in self.enemies:
            ennemi.tmp_mouv = 400
    if len(self.enemies) <= 10:
        for ennemi in self.enemies:
            ennemi.tmp_mouv = 250
    if len(self.enemies) == 1:
        for ennemi in self.enemies:
            ennemi.tmp_mouv = 100
```

La fonction update_ennemies_vitesse, change la vitesse des ennemies en fonction du nombre d'ennemies restantes.

```
def verifier_collisions(self):
    '''Fonction qui verifier la colision des lasers joueur et ennemies [laser_jou
    SpaceInvaders(modif) -> None
    '''
    collidedict = sprite.groupcollide(self.lasers, self.lasersEnnemi,True, False)
    if collidedict:
        for value in collidedict.values():
            for spriteCourant in value:
                self.lasersEnnemi.remove(spriteCourant)
                self.allSprites.remove(spriteCourant)
```

La fonction verifier_collisions, vérifie toutes les collisions des lasers et des vaisseaux.

L'image au-dessus représente la vérification entre le tir du joueur et les tires des ennemies.

```
enemiesdict = sprite.groupcollide(self.lasers, self.enemies,True, False)
if enemiesdict:
    for value in enemiesdict.values():
        for spriteCourant in value:
            self.enemies.kill(spriteCourant)
            self.sons['ennemimort'].play()
            score = self.calcul_score(spriteCourant.ligne)
            explosion = Explosion(spriteCourant.rect.x, spriteCourant.rect.y, spriteCourant.ligne, False, False,score)
            self.explosionsGroup.add(explosion)
            self.allSprites.remove(spriteCourant)
            self.enemies.remove(spriteCourant)
            self.jeuTimer = time.get_ticks()
            break
```

Ici nous vérifions si le tir du joueur touche un ennemi ou non.

```
mysterydict = sprite.groupcollide(self.lasers, self.mysteryGroup,True, True)
if mysterydict:
    for value in mysterydict.values():
        for spriteCourant in value:
            spriteCourant.vaisseau_mystere.stop()
            self.sons['mystmort'].play()
            score = self.calcul_score(spriteCourant.ligne)
            explosion = Explosion(spriteCourant.rect.x,spriteCourant.rect.y,spriteCourant.ligne, False, True,score)
            self.explosionsGroup.add(explosion)
            self.allSprites.remove(spriteCourant)
            self.mysteryGroup.remove(spriteCourant)
            newVaisseau = vaisseau_mys()
            self.allSprites.add(newVaisseau)
            self.mysteryGroup.add(newVaisseau)
            break
```

Ici nous vérifions si le tir touche le vaisseau mystère.

```
lasersdict = sprite.groupcollide(self.lasersEnnemi, self.joueurGroup,True, False)
if lasersdict:
    for value in lasersdict.values():
        for joueurVaisseau in value:
            if self.vies == 3:
                self.vies -= 1
                self.viesGroup.remove(self.vie3)
                self.allSprites.remove(self.vie3)
            elif self.vies == 2:
                self.vies -= 1
                self.viesGroup.remove(self.vie2)
                self.allSprites.remove(self.vie2)
            elif self.vies == 1:
                self.vies -= 1
                self.viesGroup.remove(self.vie1)
                self.allSprites.remove(self.vie1)
            elif self.vies == 0:
                self.gameOver = True
                self.startjeu = False
            self.sons['vaisseau_joueur_explosion'].play()
            explosion = Explosion(joueurVaisseau.rect.x, joueurVaisseau.rect.y,0, True, False, 0)
            self.explosionsGroup.add(explosion)
            self.allSprites.remove(joueurVaisseau)
            self.joueurGroup.remove(joueurVaisseau)
            self.creerNouveauVaisseau = True
            self.VaisseauTimer = time.get_ticks()
            self.VaisseauVie = False

if sprite.groupcollide(self.enemies, self.joueurGroup, True, True):
    self.gameOver = True
    self.startjeu = False
```

Ici nous vérifions si le joueur, c'est fait toucher par un tir ennemi et s'il faut mettre le jeu en perdu ou non.

```
def creer_nouveau_vaisseau(self, creerVaisseau):
    '''Fonction qui creer un vaisseau pour le joue
    SpaceInvaders (modif) -> None
    '''
    if creerVaisseau:
        self.joueur = Vaisseau()
        self.allSprites.add(self.joueur)
        self.joueurGroup.add(self.joueur)
        self.creerNouveauVaisseau = False
        self.VaisseauVie = True
```

La fonction créer_nouveau_vaisseau, crée le vaisseau du joueur si celui-ci se fait toucher par un tir et qui lui reste de la vie.

```
def menu_jeu_perdu(self, tmp_save):
    '''Fonction qui dessine le menu quand le joueur
    SpaceInvaders (modif) -> None.
    '''
    self.surface.blit(self.background, (0, 0))
    tmp_passe = tmp_save - self.timer
    if tmp_passe < 750:
        self.gameOverText.draw(self.surface)
    elif 750 < tmp_passe < 1500:
        self.surface.blit(self.background, (0, 0))
    elif 1500 < tmp_passe < 2250:
        self.gameOverText.draw(self.surface)
    elif 2250 < tmp_passe < 2750:
        self.surface.blit(self.background, (0, 0))
    elif tmp_passe > 3000:
        self.surface_ecran = 0

    for e in event.get():
        if self.sortie(e):
            sys.exit()
```

La fonction menu_jeu_perdu, affiche l'écran de perte du joueur en faisant clignoter le « GAME OVER ».

```
def main(self):
    '''
    Fonction principal
    SpaceInvaders (modif) -> None
    '''
    #Musique de fond
    mixer.Channel(0).play(mixer.Sound('sons/MIT_Concert_Choir_-_01_-_O_Fortuna.wav'), loops = -1)
```

La fonction main, est notre fonction principale. Elle est divisée en plusieurs parties : La première est la musique de fond sur le canal 0 qui est réservé juste pour cette musique. Par défaut pygame.mixer est sur 8 canaux mais si nous ne précisions pas le canal celui-ci ne laisse pas la musique de fond si d'autres sons arrivent au fur et à mesure.

```
while True:

    #Menu 1 (INSERT COIN ["espace" pour mettre des crédits, "entrer" pour
    if self.surface_ecran == 0:
        self.reset(0, 3, True)
        self.surface.blit(self.background, (0, 0))
        Text(FONT, 20,str(self.coins), BLANC,750,580).draw(self.surface)
        self.titleCoin2Text.draw(self.surface)
        if (self.timer % 3000)<1500 and self.coins < 1:
            self.titleCoinText.draw(self.surface)
        if (self.timer % 3000)<1500  and (self.coins > 0):
            self.titleCoin3Text.draw(self.surface)
        for e in event.get():
            if self.sortie(e):
                sys.exit()
            if e.type == KEYUP:
                if e.key == K_SPACE:
                    self.coins += 1
                if e.key == K_RETURN and self.coins > 0:
                    self.coins -=1
                    self.surface_ecran += 1
```

La deuxième partie est tant que le jeu n'est pas fermé nous devons afficher des Menus.

Le Menu 1 représente un menu comme sur Arcade où il faut ajouter des crédits pour jouer. Pour ajouter des crédits il faut appuyer sur la touche « espace » ou « select » sur une manette et pour aller sur le Menu 2 (le score de chaque ennemi) il faut appuyer sur la touche « entrée » ou « start » sur manette.

```
#Menu 2 (Regles du Jeu score)
elif self.surface_ecran == 1:
    self.reset(0, 3, True)
    self.surface.blit(self.background, (0, 0))
    self.titleText.draw(self.surface)
    self.ennemi1Text.draw(self.surface)
    self.ennemi2Text.draw(self.surface)
    self.ennemi3Text.draw(self.surface)
    self.ennemi4Text.draw(self.surface)
    self.creation_menu_regle()
```

La troisième partie est le menu du score que nous apporte chaque ennemi touché.

```
#Menu 3 Jeu
elif self.startjeu:
    if len(self.ennemies) == 0:
        self.surface.blit(self.background, (0, 0))
        self.scoreText2 = Text(FONT, 20, str(self.score),VERT, 85, 5)
        self.scoreText.draw(self.surface)
        self.scoreText2.draw(self.surface)
        self.viesText.draw(self.surface)
        self.viesGroup.update(self.touches)
        self.touche_appuyee()
        self.compteurPositionActive += 35
        self.reset(self.score, self.vies)
        self.jeuTimer += 3000
    else:
        tmp_save = time.get_ticks()
        self.lire_musique_ennemies(tmp_save)
        self.surface.blit(self.background, (0, 0))
        self.ensembleBoucliers.update(self.surface)
        self.scoreText2 = Text(FONT, 20, str(self.score), VERT,85, 5)
        self.scoreText.draw(self.surface)
        self.scoreText2.draw(self.surface)
        self.viesText.draw(self.surface)
        self.touche_appuyee()
        self.allSprites.update(self.touches, tmp_save,self.ennemies)
        self.explosionsGroup.update(self.touches, tmp_save)
        self.verifier_collisions()
        self.creer_nouveau_vaisseau(self.creerNouveauVaisseau)
        self.update_ennemies_vitesse()

        if len(self.ennemies) > 0:
            self.creation_ennemies_tire()
```

La quatrième partie est le jeu en lui-même représenté par le menu 3 (surface_ecran = 2)

```
#Menu 4 : Perdu
elif self.gameOver:
    tmp_save = time.get_ticks() '''Ennemi position de départ'''
    self.compteurPositionActive = self.positionEnnemiDefaut
    self.menu_jeu_perdu(tmp_save)
```

La cinquième partie du jeu est l'affichage du menu « GameOver » si le joueur perd sa partie.

```
#Vérification du score
if self.score > self.hi_score:
    open("hi_score","w").write(str(self.score))
    self.highscoreText2 = Text(FONT,20,str(self.score), VERT, 450, 5)
self.highscoreText.draw(self.surface)
self.highscoreText2.draw(self.surface)

display.update()
self.clock.tick(60)
```

La dernière partie correspond à la vérification du score. Si celui-ci est au-dessus du score maximal alors le « High score » augmente et chaque modification du score maximal et enregistrer dans un fichier annexe pour ne pas perdre le score le plus élevé.