SNAKE PYTHON

DESCRIPTION DE PROJET :

Le projet snake est un projet d'étude qui à été réalisé à par un groupe de 4 élèves du 3/5/23 au 22/6/23.

Afin de renouveler ce vieux jeu connu de tous, le projet SNAKE en Python a été attribué au élèves avec comme consignes d'être « le plus innovant possible », de là en ont découlé diverses versions de snake, dont celle-ci présenté dans ce document possédant divers mode de jeu et mode de jouabilité.

Le snake possède :

- Un plateau de jeu d'au moins 10x10 pixels
- Un serpent d'une longueur minimale de 3 cases
- Une multitude d'objectifs à atteindre sous la forme de pommes
- Un affichage du temps de la partie ainsi que des points obtenus
- Une augmentation de la taille du serpent lors de l'acquisition de points
- Une augmentation du nombre de pommes présentes sur le terrain

De là, le snake en question suit également les règles primaires de ce jeu bien connu :

- L'impossibilité de toucher les murs sous peine de défaite
- L'impossibilité de passer à travers son propre corps sous peine de défaite

En plus de tout cela, ce projet aborde l'innovation et la révolution du jeu snake grâce à :

- La possibilité d'avoir une multitude de modes de jeu variés
- Le changement de couleur du serpent à chaque obtention de points
- La possibilité de jouer avec une manette

• La possibilité d'augmenter la vitesse du serpent à volonté

• La possibilité d'augmenter la difficulté à souhait

• L'ajout d'un court « dash » que le serpent peut effectuer pour esquiver divers obstacles

• L'ajout constant et aléatoire (RNG) d'obstacles venant perturber la progression du joueur

CONTRAINTES TECHNIQUES

Afin d’élaborer ce snake, ce dernier doit être développé en Python et, remplir les crtières cités ci-dessus.

COMPETENCES UTILES

Travailler en mode projet

« ▸ Analyser les objectifs et les modalités d'organisation d'un projet

▸ Planifier les activités

▸ Évaluer les indicateurs de suivi d'un projet et analyser les écarts »

Dans le cadre de notre projet de groupe, nous avons dû nous organiser et établir un cahier des charges détaillé pour répartir les tâches de manière ordonnée. Cela impliquait d'analyser les objectifs et les stratégies d'organisation du projet. Nous avons ensuite planifié les activités, en fixant des échéances claires au sein de notre groupe de discussion. Par ailleurs, nous avons évalué les indicateurs de suivi du projet et analysé les écarts pour assurer la finalisation du site dans les délais impartis. Des réunions régulières étaient organisées pour vérifier l'avancement de chacun et gérer les retards éventuels, afin de garantir la livraison du projet à temps.

Mettre à disposition des utilisateurs un service informatique

« ▸ Réaliser les tests d'intégration et d'acceptation d'un service

▸ Déployer un service

▸ Accompagner les utilisateurs dans la mise en place d'un service »

Bien entendu, nous avons dû tester notre projet pour garantir sa conformité et vérifier s'il répondait à toutes les exigences spécifiées dans le cahier des charges, une étape essentielle pour réaliser les tests d'intégration et d'acceptation du service. De plus, lors de la dernière semaine avant sa finalisation, le jeu était accessible sur un ordinateur local de l'école, permettant ainsi de jouer entre amis. Nous avons aussi pris le temps d'expliquer le fonctionnement de notre jeu à travers une démonstration et un dialogue avec notre professeur, ce qui nous a permis d'accompagner les utilisateurs, notamment dans la gestion de l'interface d'administration.

Organiser son développement professionnel

« ▸ Mettre en place son environnement d’apprentissage personnel

▸ Mettre en œuvre des outils et stratégies de veille informationnelle

▸ Gérer son identité professionnelle ▸ Développer son projet professionnel »

Pour développer notre jeu en Python, nous avons d'abord dû nous auto-former en consultant diverses documentations et tutoriels en ligne, y compris des vidéos disponibles sur YouTube. Cette étape nous a permis de « Mettre en place notre environnement d'apprentissage personnel ». Nous nous sommes également informés sur les dernières bonnes pratiques du langage Python et sur les fondamentaux du jeu « snake » pour créer une version moderne et à jour. Cette démarche a également contribué à « Développer notre projet professionnel », car elle s'intègre directement dans notre parcours académique et prépare au métier que nous envisageons de poursuivre à l'avenir.

DESCRIPTION DE CHAQUE PAGE

```python
class Pommes():
    def __init__(self, largeur, hauteur, taille_case, nb_pommes = 1):
        self.largeur = largeur
        self.hauteur = hauteur
        self.taille_case = taille_case
        self.x = round(random.randrange(0, self.largeur - self.taille_case)/self.taille_case)*self.taille_
        self.y = round(random.randrange(0, self.hauteur - self.taille_case)/self.taille_case)*self.taille_
        self.nb_pommes = nb_pommes
        self.pommes_list = []

    def spawn_pommes(self):
        for i in range(self.nb_pommes):
            self.pommes_list.append(Pommes(self.largeur, self.hauteur, self.taille_case))

    def remove_pomme(self, pomme):
        self.pommes_list.remove(pomme)

    def add_pomme(self):
        self.pommes_list.append(Pommes(self.largeur, self.hauteur, self.taille_case))
```

Dans le segment de code mentionné, la génération d'une pomme est mise en place sur la "map" du jeu, ce qui consiste à générer aléatoirement un pixel qui sert de "point" pour augmenter le score du joueur. Le code spécifie les dimensions du pixel (largeur et longueur) et programme le comportement selon lequel, une fois que le serpent passe sur la case contenant la pomme, celle-ci disparaît. Ce mécanisme est crucial pour le gameplay, car il ajoute un élément de challenge et de récompense à mesure que le serpent grandit à chaque pomme consommée.

```python
class Pieges():
    def __init__(self, largeur, hauteur, taille_case):
        self.largeur = largeur
        self.hauteur = hauteur
        self.taille_case = taille_case
        self.pieges_list = []
        self.x = round(random.randrange(0, self.largeur - self.taille_case)/self.taille_case)*self.taille_
        self.y = round(random.randrange(0, self.hauteur - self.taille_case)/self.taille_case)*self.taille_

    def add_piege(self):
        self.pieges_list.append(Pieges(self.largeur, self.hauteur, self.taille_case))

    def supprimer_piege(self,piege):
        for i in range(len(self.pieges_list)-1):
            if self.pieges_list[i].x == piege.x and self.pieges_list[i].y == piege.y:
                del(self.pieges_list[i])
```

Dans cette portion de code, nous avons implémenté la génération de pièges sur la "map" du jeu. Un piège est généré aléatoirement sur la carte sous la forme d'un pixel spécifique. Lorsque le serpent entre en collision avec ce pixel, il perd la partie. Le code gère ainsi les dimensions de ce pixel et programme le comportement de jeu où le contact avec le piège entraîne une défaite immédiate.

```
def murcollision(self):
    if self.snake.x < 0 or self.snake.y < 0 or self.snake.x > self.largeur - self.taille_case or self.
        return True
    if self.gamemode == 'pieges':
        for piege in self.pieges.pieges_list:
            if self.snake.x == piege.x and self.snake.y == piege.y:
                return True
    return False

def pommecollision(self):
    for pomme in self.pommes.pommes_list:
        if self.snake.x == pomme.x and self.snake.y == pomme.y:
            return pomme
    return False
```

Dans cette partie du code, nous traitons deux types de collisions : celles avec les murs et celles avec les pommes. Pour les collisions avec les murs, le code détermine si le serpent touche le bord de la "map" du jeu. Si le serpent entre en contact avec un mur, il perd la partie, ce qui ajoute un élément de contrainte spatial au jeu. En ce qui concerne les collisions avec les pommes, lorsque le serpent atteint un pixel où une pomme est présente, il la consomme, ce qui entraîne l'augmentation du score et de la taille du serpent.

ANNEXE

```
class jouer():
    black = (0,0,0)
    red = (200, 0, 0)
    green = (0, 200, 0)
    white = (240, 240, 240)
    grey = (20, 20, 20)

    colors = [(255, 110, 0), (255, 165, 0), (255, 195, 0), (255, 225, 0),
(255, 255, 0), (170, 213, 0), (85, 170, 0), (0, 128, 0), (0, 85, 85), (0, 43,
170), (0, 0, 255), (25, 0, 213), (50, 0, 172), (75, 0, 130), (129, 43, 166),
(184, 87, 202), (208, 58, 135), (231, 29, 67), (255, 0, 0), (255, 55, 0),
(255, 110, 0), (255, 165, 0), (255, 195, 0), (255, 225, 0), (255, 255, 0),
(170, 213, 0), (85, 170, 0), (0, 128, 0), (0, 85, 85), (0, 43, 170), (0, 0,
255), (25, 0, 213), (50, 0, 172), (75, 0, 130), (129, 43, 166), (184, 87,
202), (208, 58, 135), (231, 29, 67), (255, 0, 0), (255, 55, 0), (255, 110, 0),
(255, 165, 0), (255, 195, 0), (255, 225, 0), (255, 255, 0), (170, 213, 0),
(85, 170, 0), (0, 128, 0), (0, 85, 85), (0, 43, 170), (0, 0, 255), (25, 0,
213), (50, 0, 172), (75, 0, 130), (129, 43, 166), (184, 87, 202), (208, 58,
135), (231, 29, 67), (255, 0, 0), (255, 55, 0), (255, 110, 0), (255, 165, 0),
(255, 195, 0), (255, 225, 0), (255, 255, 0), (170, 213, 0), (85, 170, 0), (0,
128, 0), (0, 85, 85), (0, 43, 170), (0, 0, 255), (25, 0, 213), (50, 0, 172),
(75, 0, 130), (129, 43, 166), (184, 87, 202), (208, 58, 135)]

    def __init__(self, largeur, hauteur, taille_case):
        pygame.init()
        self.running = True
        self.endgame = False
        self.pregame = True
        self.largeur = largeur
        self.hauteur = hauteur
        self.taille_case = taille_case
        self.display = pygame.display.set_mode((self.largeur, self.hauteur))
        pygame.display.set_caption('Snake python')

        self.clock = pygame.time.Clock()
        self.snakespeed = 8
        self.time = 1

        self.snake = snake(self.display, self.taille_case)
        self.pommes = Pommes(self.largeur, self.hauteur, self.taille_case)

        self.gamemode = 'normal'

    def game_loop(self):
        while self.running == True:
            if self.pregame == True:
```

```
                while self.pregame == True:
                    self.display.fill(self.black)
                    self.message('Bienvenue au Snake ! Choisir la difficulté
:', self.largeur/2, 80)
                    self.message('A- Normal', self.largeur/2, (self.hauteur/2
- 64))
                    self.message('B- Snake plus rapide', self.largeur/2,
(self.hauteur/2))
                    self.message('C- Une seule pomme', self.largeur/2,
(self.hauteur/2 + 64))
                    self.message('D- Pieges aléatoires', self.largeur/2,
(self.hauteur/2 + 128))
                    pygame.display.update()
                    for event in pygame.event.get():
                        if event.type == pygame.KEYDOWN:
                            if event.key == pygame.K_a:
                                self.pregame = False
                            if event.key == pygame.K_b:
                                self.pregame = False
                                self.gamemode = 'rapide'
                            if event.key == pygame.K_c:
                                self.pregame = False
                                self.gamemode = 'une pomme'
                            if event.key == pygame.K_d:
                                self.pregame = False
                                self.gamemode = 'pieges'
                                self.pieges = Pieges(self.largeur,
self.hauteur, self.taille_case)
                if self.gamemode == 'une pomme':
                    self.pommes.spawn_pommes()
                else:
                    for i in range(3):
                        self.pommes.spawn_pommes()
            if self.gamemode == 'rapide':
                self.snakespeed = 20
            elif self.gamemode == 'pieges':
                nb = random.randint(0, 30)
                if nb == 3:
                    self.pieges.add_piege()
                for piege in self.pieges.pieges_list:
                    for pomme in self.pommes.pommes_list:
                        if pomme.x == piege.x and pomme.y == piege.y:
                            self.pieges.supprimer_piege(piege)


            if self.endgame == True:
                while self.endgame == True:
                    self.display.fill(self.red)
```

```
                    self.message('Tu as PERDU haha !', self.largeur/2,
(self.hauteur/2 - 64))
                    self.message('R pour une renvenche, Q pour ragequit',
self.largeur/2, self.hauteur/2)
                    self.scoremessage = ('Ton score : ') + self.printscore
                    self.message(self.scoremessage, self.largeur/2,
(self.hauteur/2 + 64))
                    pygame.display.update()
                    for event in pygame.event.get():
                        if event.type == pygame.KEYDOWN:
                            if event.key == pygame.K_r:
                                main()
                            if event.key == pygame.K_q:
                                self.running = False
                                pygame.quit()


            for event in pygame.event.get():
                if event.type == pygame.QUIT:
                    self.running = False
                    pygame.quit()
                if event.type == pygame.KEYDOWN:
                    if event.key == pygame.K_RIGHT:
                        if self.snake.direction != 'gauche':
                            self.snake.droite()
                    if event.key == pygame.K_LEFT:
                        if self.snake.direction != 'droite':
                            self.snake.gauche()
                    if event.key == pygame.K_UP:
                        if self.snake.direction != 'bas':
                            self.snake.haut()
                    if event.key == pygame.K_DOWN:
                        if self.snake.direction != 'haut':
                            self.snake.bas()
                    if event.key == pygame.K_q:
                        self.running = False
                        pygame.quit()

            if self.snake.bouger() == False:
                self.endgame = True

            if self.murcollision() == True:
                self.endgame = True

            pomme = self.pommecollision()
            if pomme != False:
                self.snake.grossir()
                self.pommes.add_pomme()
                self.pommes.remove_pomme(pomme)
```

```
            self.display.fill(self.black)
            self.drawfond()

            self.printscore = str(self.score())
            self.message('Score : ' + self.printscore, 100, 32)
            self.message('Temps : ' + str(round(self.time/self.snakespeed)),
700, 32)

            if self.gamemode == 'pieges':
                self.drawpiege(self.pieges)
            self.drawpomme(self.pommes)
            self.drawsnake(self.snake)

            pygame.display.flip()
            self.clock.tick(self.snakespeed)
            self.time += 1

    def drawsnake(self, snake):
        i=0
        for segment in self.snake.snakelist:
            pygame.draw.rect(self.display, self.colors[i], (segment[0],
segment[1], snake.taille_case, snake.taille_case))
            i+=1

    def drawpomme(self, pomme):
        for pomme in self.pommes.pommes_list:
            pygame.draw.circle(self.display, self.red,
(pomme.x+pomme.taille_case/2, pomme.y+pomme.taille_case/2),
pomme.taille_case/2)

    def drawpiege(self, piege):
        for piege in self.pieges.pieges_list:
            pygame.draw.rect(self.display, self.white, (piege.x, piege.y,
piege.taille_case, piege.taille_case))

    def drawfond(self):
        for i in range(0, self.largeur, self.taille_case):
            for j in range(0, self.hauteur, self.taille_case):
                if (i+j) %(self.taille_case*2) == 0:
                    pygame.draw.rect(self.display, self.grey, (i, j,
self.taille_case, self.taille_case))


    def murcollision(self):
        if self.snake.x < 0 or self.snake.y < 0 or self.snake.x > self.largeur
- self.taille_case or self.snake.y > self.hauteur - self.taille_case:
            return True
        if self.gamemode == 'pieges':
```

```
            for piege in self.pieges.pieges_list:
                if self.snake.x == piege.x and self.snake.y == piege.y:
                    return True
        return False

    def pommecollision(self):
        for pomme in self.pommes.pommes_list:
            if self.snake.x == pomme.x and self.snake.y == pomme.y:
                return pomme
        return False

    def message(self, message, x, y):
        self.font = pygame.font.SysFont('bahnschrift', 32)
        self.text = self.font.render(message, True, self.green)
        self.textRect = self.text.get_rect()
        self.textRect.center = (x,y)
        self.display.blit(self.text, self.textRect)

    def score(self):
        counter = -3
        for i in range(self.snake.taille_serpent):
            counter += 1
        return counter


def main():
    playgame = jouer(800, 800, 40)
    playgame.game_loop()


main()
```

```
class snake():
    def __init__(self, display, taille_case, x = 80, y = 80):
        self.display = display
        self.taille_case = taille_case
        self.x = x
        self.y = y
        self.x_change = 0
        self.y_change = 0
        self.snakelist = [(-x, -y),(0,0)]
        self.start = (x,y)
        self.snakelist.append(self.start)
        self.taille_serpent = 3
        self.direction = ''

    def gauche(self):
        self.x_change -= self.taille_case
```

```
        self.y_change = 0
        self.direction = 'gauche'

    def droite(self):
        self.x_change += self.taille_case
        self.y_change = 0
        self.direction = 'droite'

    def haut(self):
        self.x_change = 0
        self.y_change -= self.taille_case
        self.direction = 'haut'

    def bas(self):
        self.x_change = 0
        self.y_change += self.taille_case
        self.direction = 'bas'

    def grossir(self):
        self.taille_serpent += 1

    def bouger(self):
        self.x += self.x_change
        self.y += self.y_change
        self.snakehead = (self.x, self.y)
        if self.taille_serpent > 3:
            if self.snakehead in self.snakelist:
                return False
        self.snakelist.append(self.snakehead)
        if len(self.snakelist) > self.taille_serpent:
            del self.snakelist[0]
```

En annexe un, la classe « jouer » est présentée ; elle offre la possibilité de basculer entre différents modes de jeu et de lancer une partie. Cette classe permet également de configurer la partie selon les préférences de l'utilisateur, en ajustant la difficulté, le nombre de pommes et les pièges.

La deuxième annexe présente la classe « Snake », qui définit le serpent contrôlé par le joueur. Cette classe spécifie la taille du serpent, son positionnement et les commandes pour le diriger : haut, bas, gauche, droite.