Workshop 2024-2025

« Vis ma vie d'étudiant en informatique »

Document Technique

La mission

Contexte

« Allo ?! C'est la direction de l'innovation et de la pédagogie (DIP) de l'EPSI. On a un problème récurrent et on se demandait si vous, les étudiants, vous pourriez nous aider. Lors des salons, des JPO, lors de nos visites dans les collèges et lycées, nous essayons de décrire au mieux la vie quotidienne d'un étudiant « IT » en général, mais ça n'est pas toujours clair pour notre public. »

Demande

Développer un jeu qui mets une personne dans la peau d'un étudiant EPSI afin de lui expliquer d'une manière ludique « qu'est-ce que c'est que d'être un epsien ».

Architecture Technique

Type de jeu video

Visual novel qui est un type de jeu vidéo narratif, principalement basé sur du texte, des dialogues et des images statiques ou animées. Le joueur fait des choix qui influencent ou non le déroulement de l'histoire. Dans notre cas il n'y aura pas d'influence.

Logiciels, langages et outils


Nous avons utilisé le logiciel **Ren'py**. Ce dernier est un moteur de création de Visual Novels et de jeux narratifs interactifs. Il permet aux utilisateurs de créer des histoires visuelles en combinant des textes, des images, des musiques et des choix interactifs. Le logiciel utilise le langage de script **python**, qui offre des fonctionnalités plus avancées pour les développeurs expérimentés.

Pour les images, nous avons utilisé l'appareil photo Nikon du MyDil. Pour les personnages, on a généré depuis l'intelligence artificielle de Canva.


Le code s'est fait avec Visual Studio code et Atome et Pour communiquer notre travail, nous avons utilisé Trello et la suite Google avec Drive.

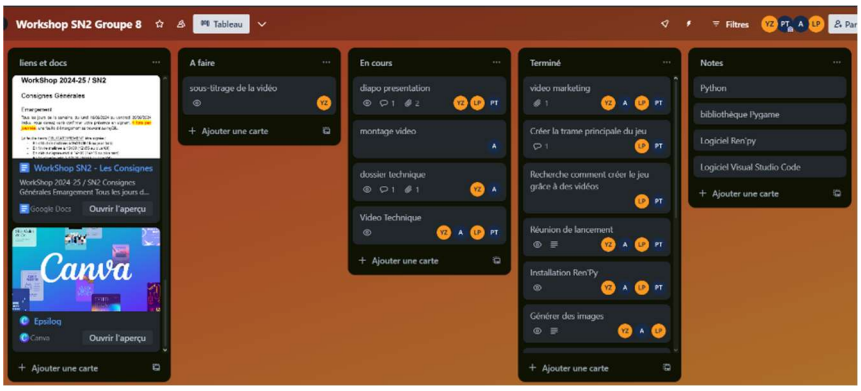



Figure 1:Screenshot Trello

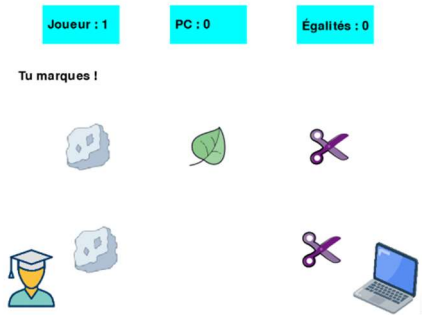
Le jeu

Epsilog suit l'histoire d'un(e) ancien(ne) étudiant(e)/collégien(ne)/lycéen(ne) qui intègre tout juste l'EPSI-WIS. Cette histoire se compose de plusieurs scènes séparées par des missions sans lesquels le jeu ne peut pas continuer.

Premier jour

-***La visite du campus*** : Une ambassadrice, Yousra, Présente les locaux à l'élève prospect (EP). En commençant de l'extérieur avec le local à vélo, en passant par la cafétéria, la pédagogie et enfin le MyDil qui sera présenté par Yvan.

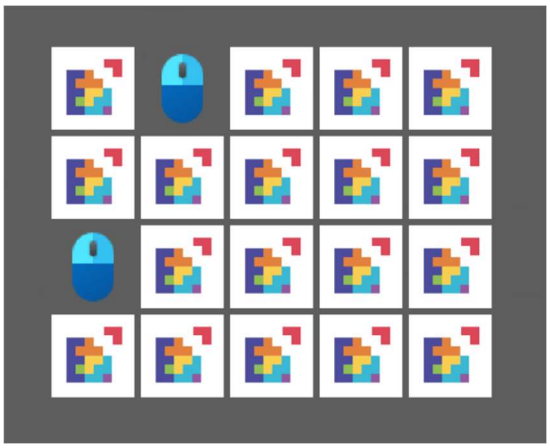
Mission 1 : Aller manger



Avec un pierre-feuille-ciseaux contre un ordinateur. Il faut choisir parmi la pierre la feuille et les ciseaux pour gagner 3 points. La logique est déterminée par Pierre>Ciseaux>Feuille>Pierre.

-***Le cours de HEP*** : Une professeure pose une question aux élèves et propose ensuite une activité : se présenter avec 3 adjectifs sans les énoncer.

Mission 2 : HEP

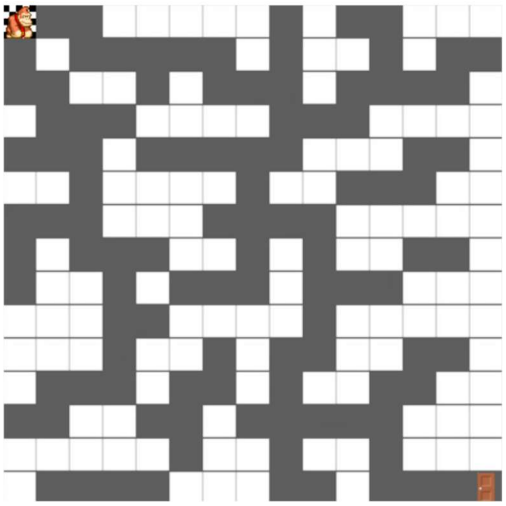


Memory : Sur les 20 cartes, 10 paires doivent être retournée. 2 par 2, on tourne les cartes pour vérifier si elles sont identiques, on les laisse tournées si elles sont identiques on les retourne faces cachées sinon. Le but est d'avoir à la fin toutes les cartes tournées.

Jour suivant

-***Journées portes ouvertes*** : Une ambassadrice, Saossane, prends sous son aile EP pour lui présenter la tâche des ambassadeurs lors des journées portes ouvertes. Elle lui demande de se mettre dans la peau d'un ambassadeur et tenter de trouver un élève à prospecter.

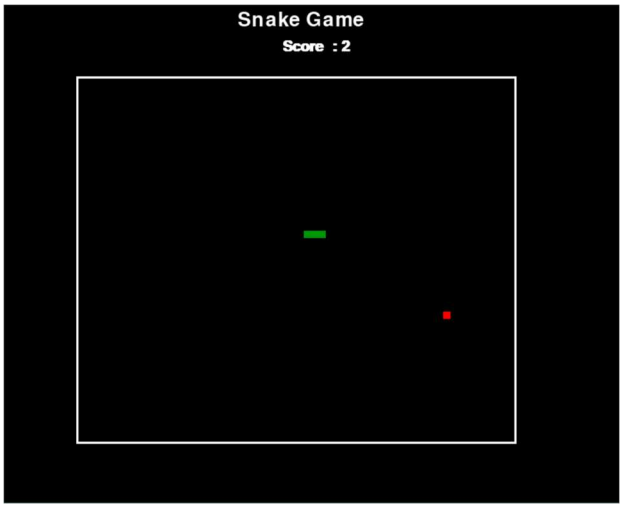
Mission 3 : Trouve un futur élève



Dans un labyrinthe, on incarne Donkey Kong. En se déplaçant en haut en bas à gauche et à droite, on doit trouver le chemin qui nous mène à la porte sans pouvoir traverser les obstacles.

-***Ateliers du soir*** : EP retourne auprès d'Yvan au MyDil pour sa première fois au club du MyDil. Yvan propose à EP de lui imprimer un porte clé en guise de souvenir pour ses débuts dans le campus. EP accepte et Yvan lui propose de jouer avec le casque VR le temps que son porte clé soit imprimé (par l'imprimante 3D).

Mission 4 : Snake



Pour Terminer ce jeu, EP s'amuse avec le jeu Snake. Le but étant de ne pas toucher ni le cadre, ni soi-même, de manger 5 pommes (parce que « 5 fruits et légumes par jours. »).

Extrait de code critique

Pour réaliser Epsilog, nous avons édité le fichier `sript.rpy` *tout au long de la semaine. Nous avons pu apporter les modifications nécessaires comme créer des personnages, les faire apparaitre et disparaitre, afficher des images de fond, et ajouter des dialogues... En parallèle, nous avons codé 4 mini jeux qui ont ensuite été intégrés à notre tram.*

Les boutons CTA

Au début du jeu ainsi qu'avant chaque mini-jeux, il y a un bouton qui débloque une action :

```
#création button jeu/ début de journée
screen redBoutton():
    imagebutton:
        # idle "images/boutton.png"
        # hover "images/boutton_select.png"
        align (0.51,0.65)
        auto "boutton_%s.png" action Jump ("scene1")
```

Dans cet exemple, on créer un « bouton rouge », c'est enfaite une image qui amène vers la scène 1 `jump (''scene1'')`.

Définition des personnages

Les personnages du jeu sont définis à l'aide de la classe **Character** de **Ren'Py.** Chaque personnage est associé à un nom et à une couleur pour leur texte :

```
define y = Character('Yousra', color="#0fab16")
define l = Character('Laura', color="#8020e1")
define a = Character('Amandine', color="#d2116b")
define s = Character('Saossane', color="#d21111")
```

Le joueur lui est identifié par le pseudonyme qu'iel aura choisi :

```
$ nom_du_perso = renpy.input("Entrez un nom.") #variable afin d'entrer son nom
y "Bienvenue [nom_du_perso] !"
```

Avec la fonction `renpy.input()` On stocke dans `$nom_du_perso` le pseudonyme choisit.

Étiquettes (label)

Les étiquettes dans **Ren'Py** définissent les différentes étapes de l'histoire. Chaque étiquette représente une scène ou un point de décision dans le jeu.

Exemple avec le label start :

C'est la description de la séquence start : Il y a une image de fond (`entrer.png`).

```
label start:
    scene entrer
    show yousra:
        xalign 0.1
        yalign 0.481
```

Menus interactifs

Le script inclut plusieurs menus qui permettent au joueur de faire des choix qui influencent le déroulement de l'histoire.

```
menu:
    y "Qu'est-ce qui t'a poussé à venir découvrir notre école ?"
    "Par passion pour l'informatique":
        jump informatique

    "Par curiosité":
        jump curiosité

    "Par recommandation":
        jump recommandation
```

La fonction `jump` permet de se déplacer vers le label saisit en argument.

Gestion des transitions et animations

- Apparition des personnages : Les personnages sont positionnés et alignés sur l'écran en utilisant les fonctions `xalign`, `yalign` et `align` pour définir leur positionnement exact. Exemple :

```
label start:
    scene entrer
    show yousra:
        xalign 0.1
        yalign 0.481
```

- Disparition en fondu : en utilisant la fonction `hide` avec `dissolve` pour que la transition soit moins brusque et donc plus agréable.

```
hide yousra
with dissolve
hide saossane
with dissolve
```

- Changements de scène : Les scènes sont chargées avec la commande `scene`, qui change l'arrière-plan du jeu.

```
label cafet:
    scene cafet1
```

- Pause et fin de jeu : Le jeu inclut également des pauses contrôlées avec `renpy.pause` pour des transitions fluides.

```
scene black
show amandine:
    align(0.85, 0.535)
a "Bienvenue dans le jeu"
$ renpy.pause(1.0)
```

Bandes sonores

La fonction qui nous permet de lancer une musique est `play music` qui prend en argument le nom du fichier son.

```
label start:
    play music "music2.mp3" # lance la musique
    scene accueil : #affiche la scène
        zoom 1.79 #zoom le fond de scène
    call screen redBoutton    #appel du bouton pour l'afficher
    stop music    # arrete la musique
```

Ici on lance la bande sonore `music2.mp3` lors de la séquence start.

Les mini-jeux

Avant de commencer à coder les mini jeux, nous avons cherché une bibliothèque python qui pourrait nous aider. Nous avons alors trouvé Pygame, spécialement conçu pour développer des jeux en python. Nous nous sommes alors documentés pour connaitre ses fonctions principales, comme « pygame.display.set_mode() » pour créer une fenêtre.

```
#Afficher l'écran du jeu
screen = pygame.display.set_mode((gameWidth, gameHeight), pygame.NOFRAME)
```

Les paramètres permettent de fixer la longueur, la largeur et le dernier élément « pygame.NOFRAME » indique que nous ne voulons pas de barre de titre. Ainsi, on ne peut pas quitter le jeu car il n'y a pas la croix.

Pour chaque mini jeu, nous nous sommes aidés d'internet pour comprendre le processus, ce que nous devions réaliser et comprendre certaines erreurs. Nous avons ensuite codé nos besoins avant de mettre au propre et d'organiser les fichiers.

Labyrinthe

Dans le labyrinthe, le but est de déplacer le personnage jusqu'à la sortie. Pour gérer les images du personnage, nous avons créé la classe Perso suivante.

```
class Perso:

    def __init__(self, droite, gauche, haut, bas, niveau):
        #sprites du personnage
        self.droite = pygame.image.load(droite).convert_alpha()
        self.gauche = pygame.image.load(gauche).convert_alpha()
        self.haut = pygame.image.load(haut).convert_alpha()
        self.bas = pygame.image.load(bas).convert_alpha()

        #position du personnage
        self.case_x = 0
        self.case_y = 0
        self.x = 0
        self.y = 0

        #direction par defaut
        self.direction = self.droite

        self.niveau = niveau
```

Elle charge les images correspondantes à chaque direction et positionne le personnage sur la case de départ, aux cordonnées (0,0).

Pour les conditions de direction, on a créé dans cette même classe une fonction déplacer permettant de modifier les coordonnées et l'image du personnage.

```
def deplacer(self, direction, son):
    if direction == 'droite':
        if self.case_x < (nombre_sprite_cote - 1):
            if self.niveau.structure[self.case_y][self.case_x+1] != 'm':
                son.play()
                self.case_x +=1
                self.x = self.case_x * taille_sprite
        self.direction = self.droite

    if direction == 'gauche':
        if self.case_x > 0:
            if self.niveau.structure[self.case_y][self.case_x-1] != 'm':
                son.play()
                self.case_x -=1
                self.x = self.case_x * taille_sprite
        self.direction = self.gauche

    if direction == 'haut':
        if self.case_y > 0:
            if self.niveau.structure[self.case_y-1][self.case_x] != 'm':
                son.play()
                self.case_y -= 1
                self.y = self.case_y * taille_sprite
        self.direction = self.haut

    if direction == 'bas':
        if self.case_y < (nombre_sprite_cote - 1):
            if self.niveau.structure[self.case_y+1][self.case_x] != 'm':
                son.play()
                self.case_y += 1
                self.y = self.case_y * taille_sprite
        self.direction = self.bas
```

En revanche, il faut prendre en compte que le personnage ne peut aller ni sur une case où se situe un mur, ni hors du jeu. Pour ça, on rajoute les conditions suivantes et c'est seulement si elles sont respectées que les modifications se font.

```
if self.case_x < (nombre_sprite_cote - 1):
    if self.niveau.structure[self.case_y][self.case_x+1] != 'm':
```

Pour le labyrinthe, on l'a d'abord écrit avec des lettres dans un fichier texte. d signifie « départ », a « arrivée », m « mur » et 0 vide.

```
d00mmmmm0m00mmm
0m00000m0mm0m00
00mm0m000m0000m
m000mmmm000mmmm
000m00000mmm00m
mm0mmmm0mm000mm
000mmm0000mmmmm
0m000mm0m0mm00m
0mm0m000m000mmm
mmm00mmmm0mmmmm
mmm0mm0m00mm00m
m000m00m0mm00mm
00mm00m00000mmm
mmmmm0mm0mm0mmm
m0000mmm00m000a
```

Les cases 0 ne sont pas remplacées par des images comme c'est le cas pour les autres lettres car ce sera le fond du jeu.

Ainsi, pour remplacer les lettres, on a créé 2 fonctions. Générer (ci-dessous) permet de parcourir le fichier ligne par ligne et lettre par lettre sans compter les sauts à la ligne avant de sauvegarder la structure.

```
def generer(self):
    #ouverture du fichier
    with open (self.fichier, "r") as fichier:
        structure_niveau = []

        #parcourir les lignes du fichier
        for ligne in fichier:
            ligne_niveau = []

            #parcourir les lettres de la ligne
            for lettre in ligne:

                #ignorer les saut de fin de ligne
                if lettre != '\n':

                    #ajouter la lettre à la liste de la ligne
                    ligne_niveau.append(lettre)

            #ajouter la ligne à la liste du niveau
            structure_niveau.append(ligne_niveau)

        #sauvergarder la structure
        self.structure = structure_niveau
```

```
def afficher(self, fenetre):
    #charger les images
    mur = pygame.image.load(image_mur).convert()
    depart = pygame.image.load(image_depart).convert()
    arrivee = pygame.image.load(image_arrivee).convert_alpha()

    #parcourir la liste du niveau
    num_ligne = 0
    for ligne in self.structure:
        num_case = 0
        for lettre in ligne:
            x = num_case*taille_sprite
            y = num_ligne*taille_sprite
            if lettre == 'm':
                fenetre.blit(mur, (x,y))
            elif lettre == 'd':
                fenetre.blit(depart, (x,y))
            elif lettre == 'a':
                fenetre.blit(arrivee, (x,y))
            num_case +=1
        num_ligne +=1
```

Afficher charge les images nécessaires et pour chaque lettre de la structure enregistrée précédemment, on affiche son image correspondante .

Ainsi, dans la boucle principale du jeu, on appelle la fonction déplacer avec la direction souhaitée selon la flèche choisie par l'utilisateur puis on met à jour la fenêtre.

```
        #touches de déplacement de donkey kong
        if event.key == K_RIGHT:
            dk.deplacer('droite', son)
        elif event.key == K_LEFT:
            dk.deplacer('gauche', son)
        elif event.key == K_UP:
            dk.deplacer('haut', son)
        elif event.key == K_DOWN:
            dk.deplacer('bas', son)

#affichage aux nouvelles positions
fenetre.blit(fond, (0,0))
niveau.afficher(fenetre)
fenetre.blit(dk.direction, (dk.x, dk.y))
pygame.display.flip()
```

Pour que le jeu se termine seulement lorsque le personnage atteint la sortie, on rajoute en fin de boucle la condition suivante, qui vérifie si le personnage se situe sur la même case que celle d'arrivée.

```
if niveau.structure[dk.case_y][dk.case_x] == 'a':
    running = 0
    pygame.mixer.music.stop()
    pygame.quit()
```

Memory

Pour créer le mini jeu memory, on initie dans un premier temps plusieurs variables : la taille des cartes, le nombre de colonnes et de lignes, les marges, etc.

On crée ensuite une liste d'images mémoire contenant toutes les images ainsi qu'une copie de chacune pour faire les paires, cela se fait grâce aux lignes suivantes.

```
memoryPictures = []
for item in os.listdir(f"{img_path2}/"):
    memoryPictures.append(item.split('.')[0])

memoryPicturesCopy = memoryPictures.copy()
memoryPictures.extend(memoryPicturesCopy)
memoryPicturesCopy.clear()
random.shuffle(memoryPictures)
```

Dans la boucle principale du jeu, on affiche alors les cartes, après avoir géré les évènements, notamment les interactions de l'utilisateur avec la fenêtre et les cartes.

```
if event.type == pygame.MOUSEBUTTONDOWN and event.button == 1:
    for item in memPicsRect:
        if item.collidepoint(event.pos):
            son.play()
            if hiddenImages[memPicsRect.index(item)] != True:
                if selection1 != None:
                    selection2 = memPicsRect.index(item)
                    hiddenImages[selection2] = True
                else:
                    selection1 = memPicsRect.index(item)
                    hiddenImages[selection1] = True
```

Ainsi, si l'utilisateur clique sur une carte avec le bouton gauche de la souris, on vérifie quelle carte a été cliquée, si la carte n'est pas déjà retournée, un son est joué et la carte se retourne. Si c'est la première carte sélectionnée, elle est mémorisée, si c'est la deuxième carte, elle est aussi mémorisée et marquée comme visible.

```
for i in range(len(memoryPictures)):
    if hiddenImages[i] == True:
        screen.blit(memPics[i], memPicsRect[i])
    else:
        rect = pygame.Rect(memPicsRect[i][0], memPicsRect[i][1], picSize, picSize)
        screen.blit(cartes, rect)
        cartes = pygame.transform.scale(cartes, (rect.width, rect.height))
```

A chaque fois que l'utilisateur a sélectionné deux cartes, on vérifie que ces dernières correspondent grâce aux lignes ci-dessous, et c'est le cas, on enlève l'image de carte et on laisse les images affichées. Sinon, on recache les images.

```
if selection1 != None and selection2 != None:
    if memoryPictures[selection1] == memoryPictures[selection2]:
        selection1, selection2 = None, None
    else:
        pygame.time.wait(1000)
        hiddenImages[selection1] = False
        hiddenImages[selection2] = False
        selection1, selection2 = None, None
```

Pour gagner et fermer la fenêtre, on s'assure que toutes les paires soient trouvées, c'est le cas lorsque win = 1, comme indiqué ci-dessous.

```
    for number in range(len(hiddenImages)):
        win *= hiddenImages[number]

    if win == 1:
        gameLoop = False
        pygame.mixer.music.stop()

    pygame.display.update()

pygame.quit()
```

Pierre Feuille Ciso

Le pierre feuille ciseaux regroupe les images et l'algorithme. En amont de la boucle principale du jeu, on initialise les valeurs et on charge toutes les images utiles (pierre, feuille, ciseaux, étudiant, ordinateur). On associe chaque image à choix avec ce dictionnaire.

```
img_choix = {
    "p" : img_pierre,
    "f" : img_feuille,
    "c" : img_ciseaux
}
```

Pour comparer les futurs choix de l'utilisateur et de l'ordinateur, on a créé la fonction comparaison suivante.

```
def comparaison(coup_joueur, coup_pc):
    global joueur_vic, pc_vic, egal

    if coup_joueur == coup_pc:
        egal +=1
        return "Egalité !"

    elif coup_joueur == "p":
        if coup_pc == "f":
            pc_vic +=1
            return "Le PC marque"
        else:
            joueur_vic +=1
            pierre.play()
            return "Tu marques !"

    elif coup_joueur == "f":
        if coup_pc == "c":
            pc_vic +=1
            return "Le PC marque"
        else:
            joueur_vic +=1
            feuille.play()
            return "Tu marques !"

    elif coup_joueur == "c":
        if coup_pc == "p":
            pc_vic +=1
            return "Le PC marque"
        else:
            joueur_vic +=1
            ciseau.play()
            return "Tu marques !"
```

Si les choix sont identiques, le score d'égalité est incrémenté de 1. Si l'étudiant l'emporte, c'est son score qui augmente, sinon, c'est celui de l'ordinateur.

Dans la boucle du jeu, on affiche toutes les images fixes ainsi que les cases contenant les scores, ces derniers sont créés ainsi :

```
score_joueur = font.render(f"Joueur : {joueur_vic}", True, NOIR)
screen.blit(score_joueur, (rect_joueur.x + 10, rect_joueur.y + 25))

score_pc = font.render(f"PC : {pc_vic}", True, NOIR)
screen.blit(score_pc, (rect_pc.x + 10, rect_pc.y + 25))

score_egal = font.render(f"Égalités : {egal}", True, NOIR)
screen.blit(score_egal, (rect_egal.x + 10, rect_egal.y + 25))
```

A chaque fois que le joueur a cliqué sur un objet, les coups de chacun s'affichent en bas de l'écran grâce à la fonction screen.blit suivante, avec en paramètre l'image correspondant au choix et la position à laquelle les afficher.

```
if coup_joueur:
    screen.blit(img_choix[coup_joueur], (150, 500))
if coup_pc:
    screen.blit(img_choix[coup_pc], (590, 500))
```

Pour savoir ce que le joueur a choisi, on se repère grâce aux coordonnées.

```
pos_x, pos_y = event.pos

# Détecter quel image est cliquée en fonction de la position
if 135 <= pos_x <= 135 + 150 and 250 <= pos_y <= 250 + 150:
    coup_joueur = "p"  # Pierre
elif 370 <= pos_x <= 370 + 150 and 250 <= pos_y <= 250 + 150:
    coup_joueur = "f"  # Feuille
elif 605 <= pos_x <= 605 + 150 and 250 <= pos_y <= 250 + 150:
    coup_joueur = "c"  # Ciseaux
else:
    coup_joueur = None
```

On définit des délimitations pour chaque objet selon leur position initiale et si le clic se situe dans un d'entre eux, on peut déterminer l'objet joué.

L'ordinateur n'est pas un autre joueur, on le fait choisir un objet au hasard avec la fonction random, et on compare son choix avec celui du joueur en appelant la fonction comparaison.

```
if coup_joueur:
    coup_pc = random.choice(choix)
    resultat = comparaison(coup_joueur, coup_pc)

    # Vérification de fin de partie
    if joueur_vic == 3:
        resultat = "Bravo, tu as gagné la partie !"
        running = False
        pygame.mixer.music.stop()
    elif pc_vic == 3:
        resultat = "Tu as perdu ! Recommences"
        joueur_vic = 0
        pc_vic = 0
        egal = 0
```

Si le joueur atteint 3 points, la partie est finie, sinon, elle recommence.

Snake

Enfin, le jeu Snake prend en compte les flèches du clavier et la direction du serpent tout en respectant certaines conditions. On indique premièrement les valeurs initiales, que l'on met dans une fonction pour pouvoir les appeler avant la boucle du jeu et plus tard.

```
def init():

    #Variables de position
    serpent_x = 300
    serpent_y = 300

    #Variables de direction
    serpent_direction_x = 0
    serpent_direction_y = 0

    serpent_corps = 10

    #Creation de la pomme
    pomme_x = random.randrange(110, 690, 10)
    pomme_y = random.randrange(110, 590, 10)
    pomme = 10

    #Creation de la liste permettant de garder la tete du serpent
    positions_serpent = []
    taille_du_serpent = 1

    #Debut du jeu
    score = 0

    return (serpent_x, serpent_y, serpent_direction_x, serpent_direction_y,
    serpent_corps, pomme_x, pomme_y, pomme, taille_du_serpent,
    positions_serpent, score)
```

Dans la boucle, on prend en compte toutes les directions, et on ajoute 10 au paramètre correspondant à chacune.

```
    #Déplacement du serpent
    if event.type == pygame.KEYDOWN:

        #Aller à droite
        if event.key == pygame.K_RIGHT:
            serpent_direction_x = 10
            serpent_direction_y = 0

        #Aller en gauche
        if event.key == pygame.K_LEFT:
            serpent_direction_x = -10
            serpent_direction_y = 0

        #Aller en bas
        if event.key == pygame.K_DOWN:
            serpent_direction_x = 0
            serpent_direction_y = 10

        #Aller en haut
        if event.key == pygame.K_UP:
            serpent_direction_x = 0
            serpent_direction_y = -10

serpent_x += serpent_direction_x
serpent_y += serpent_direction_y
```

On ajoute ensuite ce paramètre incrémenté à la position du serpent. Pour afficher la délimitation du jeu, le serpent et la pomme, on utilise les fonctions draw.rect ci-dessous avec en paramètres la fenêtre, la couleur et les coordonnées. Le dernier paramètre pour la délimitation permet de renseigner l'épaisseur, ici celle de la ligne du carré blanc. On ne le précise pas pour le serpent et la pomme car ce sont des carrés pleins.

```
#Couleur du fond et délimitation
pygame.draw.rect(ecran, (255, 255, 255), (100, 100, 600, 500), 3)

#Pomme
pygame.draw.rect(ecran, (255, 0, 0), (pomme_x, pomme_y, pomme, pomme))

#Serpent
pygame.draw.rect(ecran, (0, 150, 8), (serpent_x, serpent_y, serpent_corps, serpent_corps))

#Parties du serpent
for partie_serpent in positions_serpent:
    pygame.draw.rect(ecran, (0, 150, 8), (partie_serpent[0], partie_serpent[1], serpent_corps, serpent_corps))
```

Pour gagner le jeu, il faut atteindre 5 points, comme indiqué dans le code ci-dessous.

```
if score == 5:
    running = False
    pygame.mixer.music.stop()
    pygame.quit()
```

En revanche, nous devons prendre en compte 2 conditions : celle ou la tête du serpent touche un mur et celle où il se mord la queue. Si un de ces cas arrive au cours de la partie, le jeu recommence.

```
#Cas où le serpent touche un mur
if ((serpent_x <= 100) or (serpent_x >= 700) or (serpent_y <= 100) or (serpent_y >= 600)):
    (serpent_x, serpent_y, serpent_direction_x, serpent_direction_y, serpent_corps, pomme_x,
    pomme_y, pomme, taille_du_serpent, positions_serpent, score) = init()

#Cas où le serpent se mord
for partie_serpent in positions_serpent[:-1]:
    if (partie_serpent == tete_du_serpent):
        (serpent_x, serpent_y, serpent_direction_x, serpent_direction_y, serpent_corps,
        pomme_x, pomme_y, pomme, taille_du_serpent, positions_serpent, score) = init()
```

Pour ce faire, on vérifie les conditions en prenant en compte la position du serpent selon celle du mur et celle de son corps, et si les elles sont égales, ont réinitialise toutes les valeurs pour re commencer la partie du début. C'est donc ici qu'on appelle la fonction init() présentée plus tôt.

Organisation des scripts

Lorsque tous les mini jeux étaient fonctionnels, nous avons décidé de créer un fichier contenant tous les chemins nécessaires comme ceux d'image ou de musiques. Cela permet à ce que quiconque téléchargeant notre jeu n'ai à changer qu'une seule ligne. Nous avons donc classé tous les chemins par jeux et les avons mis sous forme de variables, pour pouvoir ensuite les appeler dans les fichiers des jeux grâce aux lignes suivantes.

```
4   # Obtenir le chemin du dossier parent
5   parent_dir = os.path.abspath(os.path.join(os.path.dirname(__file__), '..'))
6
7   # Ajouter le dossier parent au chemin de recherche
8   sys.path.append(parent_dir)
9
10  from constantes_all import *
```

L'intégration des mini-jeux dans Ren'py

Une fois tous les mini jeux ainsi que la trame principale crées, l'objectif était d'intégrer les mini jeux dans le jeu principal, ce qui a été assez complexe. Dans un premier temps, nous avons essayé d'exécuter le fichier python du mini en utilisant la commande « os.system("python chemin/vers/fichier.py") ». Cette dernière ne fonctionnant pas, nous avons testé d'autres alternatives avec la fonction « subprocess.Popen » mais rien ne changeait. Le problème n'était pas le code des mini jeux car ils se lançaient correctement lorsqu'on exécutait le script seul (dans le terminal python de VScode). Simplement, le logiciel Ren'Py n'autorise pas le lancement d'un fichier externe via un terminal car il n'en a pas. Pour surmonter ça, nous avons procéder en 2 temps. Nous avons d'abord créé un nouveau fichier python contenant 4 fonctions permettant d'exécuter chaque mini-jeu.

```python
import subprocess, os, argparse
from mini_jeu.constantes_all import *

def lancer_labyrinthe():
    mini_jeu1_path = os.path.join(mini_jeu_directory, mini_jeu1, "dklabyrinthe.py")
    subprocess.run(["python", mini_jeu1_path], cwd=project_directory, capture_output=True, text=True)

def lancer_memory():
    mini_jeu2_path = os.path.join(mini_jeu_directory, mini_jeu2, "main.py")
    subprocess.run(["python", mini_jeu2_path], cwd=project_directory, capture_output=True, text=True)

def lancer_pfc():
    mini_jeu3_path = os.path.join(mini_jeu_directory, mini_jeu3, "main.py")
    subprocess.run(["python", mini_jeu3_path], cwd=project_directory, capture_output=True, text=True)

def lancer_snake():
    mini_jeu4_path = os.path.join(mini_jeu_directory, mini_jeu4, "main.py")
    subprocess.run(["python", mini_jeu4_path], cwd=project_directory, capture_output=True, text=True)

```

Nous y avons ajouté quelques lignes permettant de choisir le mini jeu à exécuter dans une invite de commande.

```python
if __name__ == "__main__":
    parser = argparse.ArgumentParser(description="Lancer un mini-jeu.")
    parser.add_argument('--jeu', choices=['labyrinthe', 'memory', 'pfc', 'snake'], help="Choisir le mini-jeu à lancer.")
    args = parser.parse_args()

    if args.jeu == 'labyrinthe':
        lancer_labyrinthe()
    elif args.jeu == 'memory':
        lancer_memory()
    elif args.jeu == 'pfc':
        lancer_pfc()
    elif args.jeu == 'snake':
        lancer_snake()
```


```python
if __name__ == "__main__":
    parser = argparse.ArgumentParser(description="Lancer un mini-jeu.")
    parser.add_argument('--jeu', choices=['labyrinthe', 'memory', 'pfc', 'snake'], help="Choisir le mini-jeu à lancer.")
    args = parser.parse_args()

    if args.jeu == 'labyrinthe':
        lancer_labyrinthe()
    elif args.jeu == 'memory':
        lancer_memory()
    elif args.jeu == 'pfc':
        lancer_pfc()
    elif args.jeu == 'snake':
        lancer_snake()
```


Ainsi, si l'on veut lancer le mini jeu memory, il faut taper les deux commandes suivantes dans l'invite de commande.

```
PS C:\Users\duran> cd "P:/1-EPSI/Projet_Workshop/SN2/Epsilog/game"
PS P:\1-EPSI\Projet_Workshop\SN2\Epsilog\game>  python "P:/1-EPSI/Projet_Workshop/SN2/Epsilog/game/test.py" --jeu memory
```

Il a ensuite fallu ajouter les lignes ci-dessous dans le script de la trame Renpy pour appeler un terminal et y exécuter les commandes précédentes.

```python
python:
    import subprocess, os
    project_directory = "P:/1-EPSI/Projet_Workshop/SN2/Epsilog/game"
    test_path = os.path.join(project_directory, "test.py")
    command = ["cmd", "/C", f"cd /d {project_directory} & python {test_path} --jeu pfc", ]
    creation_flags = subprocess.CREATE_NO_WINDOW
    subprocess.call(command, creationflags=creation_flags)
```

Ces lignes permettent d'exécuter une commande via un terminal externe, dont la page ne s'affiche pas grâce à la variable creation_flags. On renseigne en premier les chemins du répertoire contenant le projet et celui du fichier à exécuter, on indique ensuite la commande avec le jeu à lancer (ici pfc). Enfin, on l'exécute avec la fonction « subprocess.call() », permettant de mettre en pause la trame RenPy tant que le mini jeu n'est pas fini et de fermer la fenêtre à la fin pour reprendre la trame grâce au paramètre « /C » de la commande.

Analyses & Réflexions sur Epsilog

Nous pensons que ce format est un excellent moyen, à la fois visuel et concret, de mettre en valeur les particularités de notre école. À travers un scénario bien conçu, des dialogues simples et un fil conducteur fluide, il est possible de transmettre plus d'informations qu'avec des mots seuls. Ce jeu offre une représentation d'une journée type, mais regroupe en réalité les moments clés qui peuvent se produire au cours de l'année, reflétant de manière fidèle l'esprit de notre établissement. En somme, il s'agit simplement d'une façon différente de communiquer.

Axes d'amélioration :

- Trouver une meilleure transition entre les phases narratives et les mini-jeux.

- Intégrer des séquences reliées à un fichier de collecte de données pour recueillir, par exemple, des avis anonymes.

- Introduire des variations dans l'histoire selon les choix du joueur, offrant ainsi plusieurs fins possibles ou des expériences personnalisées pour chaque utilisateur.

- Les mini-jeux pourraient permettre de cumuler un score visible en fin de partie ou de débloquer un niveau secret.

En résumé, cette version est une démonstration, une version Bêta, qui permet d'imaginer le potentiel du jeu dans sa forme finale.