Maillet Louis
Meunier Tifany
Ayan Murat
2C

2024-2025

SAE 1.02 - Projet python

SAE serpiuto

IUT d'Orléans
Département informatique
Rue d'Issoudun, 45067 Orléans cedex 02

Durant cette “SAE_Serpiuto”, nous devions compléter et implémenter des fonctions dans le but de créer une IA pour un jeu de serpent. Nous avions plusieurs pages à compléter. En effet, toute la page serpent.py, dernière fonction de partie.py, case.py, arene.py donc les fonctions copy qui étaient à faire pour mardi, et pour vendredi le fichier IA, directions_possibles et objets_voisinages et les fonctions de stratégies qui étaient à faire par nous-même.

Le problème en général était qu’on ne comprenait pas du tout les docstrings. Nous étions bien gênés par cela.

Au début, nous avions remarqué qu’il y avait un problème qu’après avoir commencé les fonctions IA. Bien sûr dans la fonction « serpent_2_str » un peu comme tout le monde. Nous n’avions pas compris où était notre erreur. Puis nous avons compris qu’elle était censée renvoyer un « \n » et donc, réussi.

```python
def serpent_2_str(serpent:dict, sep=";")->str:
    """Sérialise un serpent sous la forme d'une chaine de caractères
    contenant 2 lignes.
    nom_j;num_j;nb_point;tps_surpuissance;tps_mange_mur;tps_protection
    lig1;col1;lig2;col2;...
    La première ligne donne les informations autres que la liste des positions du serpent
    la deuxième ligne donné la liste des position du serpent en commençant par la tête
    Args:
        serpent (dict): le serpent considéré
        sep (str, optional): le caractère séparant les informations du serpent. Defaults to ";".

    Returns:
        str: la chaine de caractères contenant les toutes informations du serpent
    """
    premier_phrase = get_nom(serpent) + sep + str(get_num_joueur(serpent)) + sep + str(get_points(serpent)) + \
          sep + str(get_temps_surpuissance(serpent)) + sep +  str(get_temps_mange_mur(serpent)) + sep + str(get_temps_protection(serpent)) + sep + get_derniere_direction(serpent)
    deuxieme_phrase = ""
    for pos in get_liste_pos(serpent):
        deuxieme_phrase += str(pos[0]) + sep + str(pos[1]) + sep
    deuxieme_phrase = deuxieme_phrase[:-1]
    return premier_phrase + "\n" + deuxieme_phrase + "\n"
```

Aussi, nous avons utilisé « .copy » pour les fonctions copy, qui était interdite par les professeurs, donc nous avons dû les refaire.

```python
def copy_case(case:dict)->dict:
    """fait une copie de la case

    Args:
        case (dict): la case considérée

    Returns:
        dict: la copie de la case passée en paramètre
    """
    case_copie = {}
    case_copie["mur"] = case["mur"]
    case_copie["valeur"] = case["valeur"]
    case_copie["proprietaire"] = case["proprietaire"]
    case_copie["temps_restant"] = case["temps_restant"]
    return case_copie
```

Ce qui nous a pris le plus de temps était en premier la fonction « objets_voisinages » et deuxièmement, nos programmes de stratégies.

Voici une ancienne version de « objets_voisinages », ici nous avons fait le programme pour faire en sorte, que nous essayons de trouver les positions des objets qui se trouvent aux alentours du serpent, avec plusieurs conditions si, sinon, sinon si.

```
pos_objets = dict()
    case_depart = serpent.get_liste_pos(l_arene["serpents"][num_joueur-1])[0]
    liste_cases = trouver_cases_distance(case_depart,dist_max,l_arene) #-1 car à l'affichage on dit que premier joueur est le joueur 1 !!!
    print(liste_cases)
    for (ligne,col) in liste_cases:
        if arene.get_val_boite(l_arene, ligne, col) != 0 or arene.est_bonus(l_arene, ligne, col) :#il y a une boite ou un bonus
            print(ligne,col)
            deltalignes  = ligne-case_depart[0]
            deltacol = col-case_depart[1] # on a les coordonées qui nous séparent de l'objet
            chemin = ''
            possible = True
            while (deltalignes,deltacol) != (0,0) and possible : #on va chercher le chemin pour atteindre l'objet
                if deltalignes > 0 :
                    if not arene.est_mur(l_arene,ligne-1, col) :
                        deltalignes -= 1
                        chemin += 'S'
                    else :
                        possible = False
                elif deltalignes < 0 :
                    if not arene.est_mur(l_arene,ligne+1, col) :
                        deltalignes += 1
                        chemin += 'N'
                    else :
                        possible = False
                elif deltacol > 0 :
                    if not arene.est_mur(l_arene,ligne, col-1) :
                        deltacol -= 1
                        chemin += 'E'
                    else :
                        possible = False
                elif deltacol < 0 :
                    if not arene.est_mur(l_arene,ligne, col+1) :
                        deltacol += 1
                        chemin += 'O'
                    else :
                        possible = False
                else:
                    print('case pas accessible en {dist_max} cases')
                    break
            pos_objets[chemin]=(len(chemin),arene.get_val_boite(l_arene,ligne,col),arene.get_proprietaire(l_arene,ligne,col))
    return pos_objets
```

Nous avions plusieurs idées, possibles ou pas possibles.

```
481    """
482    Idée : il ne faut pas que les serpents rentrent dans les trous d'E et S car ils ne peuvent pas sortir sans perdre des points (sauf s'ils récupèrent le point d'eux même)
483    mais ça a l'air trop compliqué de coder donc mis à côté
484
485    Idée : le serpent doit se diriger que vers les surpuissances et les défenses pour gagner des points et ne pas en perdre, finalement jouer agressivement
486
487    Idée : le serpent doit se diriger que vers les bonus pour gagner des points, et ne pas se diriger vers les autres serpents pour ne pas perdre de points,
488    finalement jouer défensivement
489
490    Idée : le serpent va essayer d'enfermer les autres serpents pour les manger, finalement jouer agressivement (trop dur à coder je ne crois même pas que c'est possible)
491
492    Idée : une fois que le serpent est le premier, il va focus sur le deuxième, pour rester premier. S'il est deuxième, il va focus sur le troisième, pour ne pas être dépassé.
493    S'il est troisième, il va focus sur le quatrième, pour ne pas être dépassé. S'il est quatrième, il va focus sur les points et les bonus mais ça a aussi l'air trop compliqué donc
494    aussi mis à côté.
495
496    Idée : le serpent va qu'attaquer l'élément le plus proche de lui sans se soucier des autres éléments, finalement jouer agressivement
497    """
```

Bien sûr que nous n'avons pas pu tout faire avec du temps limité, mais nous avons quand même réussi à faire plusieurs stratégies.

Comme ici, avec une fonction de complexité N^2, un programme qui retourne une liste avec les positions des cases autour de nous.

```python
def trouver_cases_distance(coordonné, dist_max, l_arene):
    """
    Retourne la liste des cases accessibles à partir de la coordonnée donnée
    Args:
        coordonné (tuple): les coordonnées du point de départ
        dist_max (int): la distance maximale à laquelle
        l_arene (dict): l'arène considérée
    Returns:
        list: la liste des cases accessibles


    """
    cases_accessibles = []

    longueur = l_arene["matrice"]["nb_lig"]
    hauteur = l_arene["matrice"]["nb_col"]
    x, y = coordonné
    for i in range(longueur):
        for j in range(hauteur):
            if abs(x - i) + abs(y - j) <= dist_max:
                cases_accessibles.append((i, j))


    return cases_accessibles
```

Ici, une fonction de Nlog(N) qui trie les objets en fonction de leur valeur, ainsi que leur longueur et direction, puis une deuxième fonction N^2 qui retourne l'objet le plus proche.

```python
def trier_objet(dico_objet: dict) -> list:
    """
    Trie les objets en fonction de leur type (valeur) et de la longueur de la direction.

    Args:
        dico_objet (dict): Dictionnaire des objets avec les directions comme clés.

    Returns:
        list: Liste triée des objets.
    """
    return sorted(dico_objet.items(), key=type_et_longueur)

def plus_proche_boite_bonus(l_arene:dict, num_joueur:int, dist_max:int):
    liste_unique = []
    dico_toute_pose = objets_voisinage(l_arene, num_joueur, dist_max)
    liste_triée = trier_objet(dico_toute_pose)
    if len(dico_toute_pose) == 0:
        return liste_unique
    i = 0
    for i in range(-5,3):
        for j in range(len(dico_toute_pose)):
            if liste_triée[j][1][1]==i:
                liste_unique.append(liste_triée[j])
                break
    return liste_unique
```

Ici nous avons une fonction de complexité N, qui est aussi pour trier les valeurs, qui sert pour le « key » de l'autre fonction.

```python
def type_et_longueur(item):
    """
    Fonction pour trier les objets en fonction de leur type et de la longueur de la direction.
    Args:
        item: Tuple contenant les informations de l'objet.
    Returns:
        Tuple: Tuple contenant la valeur de l'objet et la longueur de la direction.
    """
    direction, (distance, val_objet, prop) = item
    return (val_objet, len(direction))
```

Ici, nous avons une fonction qui renvoie la liste des serpents autour du serpent du joueur « num_joueur ».

```python
def plus_proche_boite_serpent(l_arene:dict, num_joueur:int, dist_max:int):
    """renvoie la liste de chaque boite de serpent (hormis notre joueur) le plus proche de nous dans le cas où il y en a un autour de nous

    Args:
        l_arene (dict): l'arene considérée
        num_joueur (int): numéro du joueur
        dist_max (int): distance max à laquelle on cherche des objets

    Returns:
        list : liste des serpents
    """
    liste_unique = []
    liste_carree = [1,2,4,8,16,32,64,128,256]
    dico_toute_pose = objets_voisinage(l_arene, num_joueur, dist_max)
    liste_triée = trier_objet(dico_toute_pose)
    if len(dico_toute_pose) == 0:
        return liste_unique
    i = 0
    for i in liste_carree:
        for j in range(len(dico_toute_pose)):
            if liste_triée[j][1][1]==i and liste_triée[j][1][2]!= num_joueur and liste_triée[j][1][2] != 0 :
                liste_unique.append(liste_triée[j])
                break
    return liste_unique
```

Ici, nous voyons une liste des boîtes numérotés, ce qui veut dire les points, avec une complexité de Nlog(N)

```python
def plus_proche_boite_spe(l_arene:dict, num_joueur:int, dist_max:int, spe:int):
    """renvoie la liste de chaque bonus de numero spe le plus proche de nous dans le cas où il y en a un autour de nous

    Args:
        l_arene (dict): l'arene considérée
        num_joueur (int): numéro du joueur
        dist_max (int): distance max à laquelle on cherche des objets

    Returns:
        list : liste des objets
    """
    liste_unique = []
    dico_toute_pose = objets_voisinage(l_arene, num_joueur, dist_max)
    liste_triée = trier_objet(dico_toute_pose)
    if len(dico_toute_pose) == 0:
        return liste_unique
    for j in range(len(dico_toute_pose)):
        if liste_triée[j][1][1]==spe and liste_triée[j][1][2]==0:
            liste_unique.append(liste_triée[j])
            break
    for position in liste_unique:
        if position== "":
            liste_unique.remove(position)
    return liste_unique
```

Ensuite, nous avons fait une stratégie (qui est je pense la plus importante de notre projet), avec 3 phases, premier 50 tours, deuxième 75 tours et troisième 50 tours. (0-50, 50-75, 75-150). (1,

prend des points, 2, prend des bonus, 3, attaque les autres)

```python
def premier_50_tours(num_joueur, arene_partie, liste_boite_manger_1, liste_boite_manger_2,directions_possibles):
    """renvoie une position vers laquelle le serpent doit aller
    si la tete du serpent vaut 2 ou plus, il va aller chercher des
    boites de 2, sinon il prend une direction aleatoire parmi celles possibles, si aucune n'est possible il prend une direction aléatoire.
    si la tete vaut moins que 2, il va aller chercher des
    boites de 1, sinon il prend une direction aleatoire parmi celles possibles, si aucune n'est possible il prend une direction aléatoire.

    args :

    num_joueur (int): numero du joueur
    arene_partie (dict): l'arene de la partie
    liste_boite_manger_1 (list) : liste des boites de valeurs 1 autour du joueur num_joueur
    liste_boite_manger_2 (list) : liste des boites de valeurs 2 autour du joueur num_joueur
    directions_possibles (str) : chaine de caractères avec les lettres des positions possibles

    return :
     str : une lettre symbolisant la direction que doit prendre le serpent
    """
    if arene.get_val_boite(arene_partie, arene.get_serpent(arene_partie, num_joueur)[0][0], arene.get_serpent(arene_partie, num_joueur)[0][1]) >= 2:
        if liste_boite_manger_2 and liste_boite_manger_2[0][0][0] in arene.DIRECTIONS: # Si une boite de valeur 2 est trouvée

            direction = liste_boite_manger_2[0][0][0]
            if direction in directions_possibles: #Si la direction est possible
                return direction
            else :
                return random.choice("NSEO")

    if liste_boite_manger_1 and liste_boite_manger_1[0][0][0] in arene.DIRECTIONS:
        direction = liste_boite_manger_1[0][0][0]

        if direction in directions_possibles:
            return direction
        return random.choice("NSEO")
    elif directions_possibles:
        return random.choice(directions_possibles)
    else:
        return 'N'
```

Bilans Personnels :

Murat :

Durant cette SAE, je me suis occupé de presque toute la page « serpent.py » (J'ai laissé quelques fonctions à Tifany car je ne comprenais pas), ainsi que les fonctions copy de partie, case et arène mais comme l'utilisation de « .copy » était interdite, nous l'avons refait. De plus, les idées de stratégies et finalement le rapport.

Cette SAE m'a permis de développer des compétences comme par exemple, la communication, collaboration, et plus spécifiquement, comme « Conduire un projet », « Travailler dans une équipe informatique », « Réaliser un développement d'applications », mais également « Optimiser des applications informatiques »

Nous étions bien un groupe ensemble, pour cela, nous avons utilisé l'extension « Live Share » sur Visual Studio Code pour coder en temps réel sur des différents ordinateurs, ensemble, plus efficace. J'ai bien apprécié mon groupe car, vu la différence de niveaux, ils m'ont bien aidé, et je ne me suis pas senti « inutile », et aussi j'ai pu les aider sur quelques fonctions donc je ne les ai pas « ralenti ».

Je pense qu'on ne pourrait jamais y arriver si on avait choisi les groupes nous-mêmes car sans se connaitre, nous étions encore plus sérieux, de plus, nous avons pu aussi se connaitre donc je dois vous remercier pour cela aussi.

Cette SAE m'a fait comprendre que je dois travailler plus pour « rattraper » mon groupe.

De ce qui est projet, je ne pense pas que j'ai « totalement » compris cette SAE puisqu'on avait qu'une semaine et vu la différence des niveaux, je pense que j'ai besoin d'avoir un peu plus de temps, pour pouvoir *re*-réaliser ces fonctions et pouvoir les faire moi-même, sans avoir besoin d'aide de quelqu'un d'autre.

Mais en tout cas j'ai bien aimé mon groupe, notre communication et notre collaboration. On s'est tous bien amusé et personnellement, je suis bien excité pour le deuxième semestre, le travail que nous avons réalisé tous ensemble, était bien amusant. J'espère qu'on va bien se mettre, avoir des bonnes relations. Comme on l'a fait, pendant ce projet SAE.

Tifany :

De mon côté, j'ai pu coder quelques fonctions dans serpents.py et j'ai principalement travaillé dans le fichier IA.py où j'ai cherché à corriger les fautes potentielles dans les programmes et j'ai également essayé de comprendre les autres fichiers tels que serveur.py ou affiche.py pour comprendre nos erreurs d'affichage du jeu et comment le jeu en lui-même fonctionnait.

Grâce à cette SAE on a pu apprendre à réaliser un projet en groupe tout en s'attribuant des tâches à faire et en apprenant à se connaître. Ce projet est semblable à ce qu'on pourra faire en entreprise donc c'est une bonne opportunité de s'entraîner et prendre un peu d'expérience sur des projets plus faciles.

Un point compliqué sur cette SAE était le fait qu'on soit avec d'autres élèves que nous ne connaissions pas et que les niveaux entre nous pouvaient être assez différents, de ce fait on devait veiller à ce que tout le monde comprenne tout en avançant ensemble et que personne ne se sente "mis de côté" à cause de son niveau moins élevé.

Louis :

Pour ma part, je me suis occupé de refaire certaines fonctions de copie car, comme l'a mentionné mon camarade, nous avions utilisé .copy() alors que cela n'était pas autorisé. J'ai donc retravaillé ces fonctions afin de les réaliser sans recourir à .copy() .

J'ai plutôt bien apprécié cette SAE, même si je trouve que nous avons perdu trop de temps sur des choses simples. En effet, nous n'avons codé que le fichier serpent et IA, ce qui a rendu la rédaction des autres fonctions particulièrement compliquée. Par exemple, pour afficher la queue d'un serpent à partir de la partie, il fallait analyser ce que renvoyait la partie (qui envoie une arène), puis ce que renvoyait l'arène, etc. Cela nous a pris énormément de temps.

Quand quelque chose n'allait pas, le simple fait de comprendre d'où venait le problème était laborieux. Si nous avions tout codé nous-mêmes dès le début, nous aurions su corriger les erreurs plus rapidement (mais il nous aurait fallu plus d'une semaine bien sûr...).

Sinon, j'ai bien aimé la SAE car créer du code Python de façon concrète, ça change, même si en cours on ne fait pas que de la théorie. Le fait de créer un mini-jeu visuel m'a beaucoup plu.

Cette SAE m'a permis de développer des compétences en communication, car il a été nécessaire de collaborer avec mon groupe pour avancer. À cet effet, nous avons créé un groupe ensemble et utilisé Live Share, une extension de VS Code, pour coder en temps réel sur deux ordinateurs différents. J'ai pu également découvrir de nouveaux camarades à qui je n'avais jamais parlé, ce que je trouve assez sympa.

De plus, cette SAE m'a permis de développer la COMPÉTENCE 1 : Réaliser un développement d'applications, mais également la COMPÉTENCE 2 : Optimiser des applications informatiques. Même si ça n'a pas toujours été parfait, j'ai pu réaliser une application, ou plutôt une partie du code d'une application, et nous avons dû optimiser nos algorithmes car nous avions 0.1 seconde pour jouer, donc il fallait que notre algorithme soit optimisé.

Louis et Tifany : Nous nous sommes occupés de tous les autres algorithmes à deux (objet_voisin, IA, etc.).

Nous avons fait les programmes à deux en regardant le code à deux et en réfléchissant à deux, ce qui nous a permis d'être bien plus efficaces. Par exemple, lorsque Louis a fait une erreur bête, comme arene[serpent] au lieu de arene["serpent"], Tifany a pu immédiatement lui signaler l'erreur, ce qui est bien plus efficace que de perdre 10 minutes seul pour une erreur comme celle-ci.

Ce travail nous a donc permis de collaborer et de travailler en équipe.