SAE 1.01 - Implémentation d'un besoin client

Lustremant Matthis
Picouleau Louis
TPC2

La fonction distance

```python
def distance(a: list, b: list) -> float:
    return math.sqrt((a[0] - b[0]) ** 2 + (a[1] - b[1]) ** 2)
```

> Cette fonction renvoie la racine carrée de la somme des carrés des différences des coordonnées des points a et b. Cela correspond à la distance entre ces deux points.

Descriptions des fonctions

DeterminerVictoire()

```python
# Détermine le résultat de la partie
def determinerVictoire(secret: list, a: list) -> bool:
    """
    Détermine si la combinaison de l'utilisateur est similaire à la combinaison secréte
    :param secret: list
    :param a: list
    :return: bool

    >>> determinerVictoire([1, 2, 3, 4, 5], [1, 2, 3, 4, 5])
    True
    >>> determinerVictoire([1, 2, 3, 4, 5], [1, 2, 3, 4, 6])
    False
    """

    # Regarde si la combinaison secret et similaire à la combinaison de l'utilisateur
    if secret == a:
        return True
    return False
```

Description

> La fonction DeterminerVictoire() a pour but de déterminer si le joueur à trouver la bonne combinaison secrète. Elle prend en paramètre la combinaison secrète et la combinaison de l'utilisateur et renvoie un booléen en fonction de si les deux combinaisons sont identiques ou non.

Pseudo-code

```
Fonction determinerVictoire(secret: liste, a: liste) : booléen
    Variables locales :

    Si secret = a Alors
        Retourner Vrai
    Fin Si
    Retourner Faux
Fin Fonction
```

Variables

Nom	Type	Rôle
secret	list	Contient la combinaison secrète
a	list	Contient la combinaison de l'utilisateur

Tests

> Ici, nous avons deux tests pour la fonction determinerVictoire() :
>
> - Le premier test est une combinaison secrète et une combinaison de l'utilisateur identiques, la fonction doit donc renvoyer True.
> - Le deuxième test est une combinaison secrète et une combinaison de l'utilisateur différentes, la fonction doit donc renvoyer False.
>
> Une fois les doctests effectués, nous pouvons dire que la fonction determinerVictoire() fonctionne correctement.

AfficherVictoire()

```
def afficherVictoire(secret: list, a: list, screen: pygame.surface, count: int) ->
bool:
    """
    Affiche le résultat de la partie
    :param secret: list
    :param a: list
    :param screen: pygame.surface
    :param count: int
    :return: bool

    >>> afficherVictoire([1, 2, 3, 4, 5], [1, 2, 3, 4, 5], creerScreen(), 16)
    True
    >>> afficherVictoire([1, 2, 3, 4, 5], [1, 2, 3, 4, 6], creerScreen(), 16)
    True
    >>> afficherVictoire([1, 2, 3, 4, 5], [1, 2, 3, 4, 6], creerScreen(), 15)
    False
```

```
    """

    # Affiche la victoire
    if determinerVictoire(secret, a):
        msg: str = "Vous avez gagnez ! en " + str(count - 1) + " coup !"
        message = pygame.font.SysFont('monospace', 20)
        message.set_bold(True)
        label = message.render(msg, 1, mm.Noir)
        screen.blit(label, (250, 700))
        mm.afficherSecret(screen, secret)
        return True

    # Affiche la défaite
    elif count == 16:
        msg: str = "Vous avez perdue ! en " + str(count - 1) + " coup !"
        message = pygame.font.SysFont('monospace', 20)
        message.set_bold(True)
        label = message.render(msg, 1, 30)
        screen.blit(label, (250, 700))
        mm.afficherSecret(screen, secret)
        return True
    else:
        return False
```

Description

> La fonction afficherVictoire() va servir à afficher un message de victoire ou de défaite en fonction de la combinaison de l'utilisateur et de la combinaison secrète. Elle retourne un booléen en fonction de si le joueur à gagner ou non.
> Elle prend en paramètre la combinaison secrète (secret), la combinaison de l'utilisateur (a), l'écran de jeu (screen) et le nombre de coups joués (count).
>
> La fonction commence par vérifier si la combinaison de l'utilisateur est identique à la combinaison secrète, si c'est le cas, elle affiche un message de victoire et retourne True.
> Ensuite, elle vérifie si le nombre de coups joués est égal à 16, si c'est le cas, elle affiche un message de défaite et retourne True.
> Si aucune des conditions précédentes n'est remplie, elle retourne False.
>
> Le message affiché en bas de l'écran contient le nombre de coups joués par le joueur.

Pseudo-code

```
Fonction afficherVictoire(secret: liste, a: liste, screen: pygame.surface, count:
entier) : booléen
    Variables locales :
        msg : chaine de caractères
        message : police de caractères
        label : Surfacetype

    Si determinerVictoire(secret, a) Alors
```

```
        msg <- "Vous avez gagnez ! en " + (count - 1) + " coup !"
        message <- pygame.font.SysFont('monospace', 20)
        message.set_bold(True)
        label <- message.render(msg, 1, mm.Noir)
        screen.blit(label, (250, 700))
        mm.afficherSecret(screen, secret)
        Retourner Vrai

    Sinon Si count = 16 Alors
        msg <- "Vous avez perdue ! en " + (count - 1) + " coup !"
        message <- pygame.font.SysFont('monospace', 20)
        message.set_bold(True)
        label <- message.render(msg, 1, 30)
        screen.blit(label, (250, 700))
        mm.afficherSecret(screen, secret)
        Retourner Vrai

    Sinon
        Retourner Faux
    Fin Si
Fin Fonction
```

Variables

Nom	Type	Rôle
secret	list	Contient la combinaison secrète
a	list	Contient la combinaison de l'utilisateur
screen	pygame.surface	Contient l'écran de jeu
count	int	Contient le nombre de coups joués
msg	str	Contient le message à afficher
message	pygame.font	Contient la police du message
label	Surfacetype	Contient le message à afficher et la couleur du texte

Tests

> Ici, nous avons trois tests pour la fonction afficherVictoire() :
>
> - Dans le premier test, la combinaison secrète et la combinaison de l'utilisateur sont identiques, la fonction doit donc renvoyer True.
> - Dans le second test, la combinaison secrète et la combinaison de l'utilisateur sont différentes, mais le nombre de coups joués est égal à 16 (le nombre maximum de coups), la fonction doit donc renvoyer True.
> - Dans le troisième test, la combinaison secrète et la combinaison de l'utilisateur sont différentes, et le nombre de coups joués est inférieur à 16, la fonction doit donc renvoyer False.

> Une fois les doctests effectués, nous pouvons dire que la fonction afficherVictoire() fonctionne correctement.

creerScreen()

```
# Création de l'affichage graphique
def creerScreen() -> pygame.display:
    """
    Création de l'affichage graphique

    :return: pygame.display

    >>> creerScreen()
    <Surface(800x750x32 SW)>
    """

    pygame.init()
    # Taille d'affichage sur 800px de largeur et 750px de hauteur
    screen: pygame.surface = pygame.display.set_mode((800, 750))
    screen.fill((255, 255, 255))
    return screen
```

Description

> La fonction creerScreen() a pour but de créer un écran de jeu de 800 pixels de largeur et 750 pixels de hauteur. La fonction ne prend aucun paramètre et retourne l'écran de jeu. Avant de retourner l'écran de jeu, la fonction remplit l'écran de jeu avec la couleur blanche.

Pseudo-code

```
Fonction creerScreen() : pygame.display
    Variables locales :
        screen : pygame.surface

    Initialiser pygame
    screen <- pygame.display.set_mode((800, 750))
    screen.fill((255, 255, 255))
    Retourner screen
Fin Fonction
```

Variables

Nom	Type	Rôle
screen	pygame.surface	Contient l'écran de jeu

Tests

> Ici, nous avons un test pour la fonction creerScreen() :
>
> - Le test consiste à créer un écran de jeu, la fonction doit donc retourner un écran de jeu de 800 pixels de largeur et 750 pixels de hauteur.
>
> Une fois le doctest effectué, nous pouvons dire que la fonction creerScreen() fonctionne correctement. L'écran de jeu est bien créé et rempli de couleur blanche.

creerCombinaisonSecrete()

```python
# Création de la combinaison secréte en aléatoire
def creerCombinaisonSecrete() -> list:
    """
    Création de la combinaison secréte en aléatoire

    :return: list
    """

    secret: list = []
    i: int
    for _ in range(5):
        i = random.randint(1, 6)
        secret.append(mm.TabCouleur[i])
    # print(secret)
    return secret
```

Description

> La fonction creerCombinaisonSecrete() a pour but de créer une combinaison secrète aléatoire.
> La fonction ne prend aucun paramètre et retourne une liste contenant la combinaison secrète. La fonction commence par créer une liste vide (secret). Puis dans elle choisir 5 fois un nombre aléatoire entre 1 et 6 qu'elle ajoute à la liste secret.

Pseudo-code

```
Fonction creerCombinaisonSecrete() : liste
    Variables locales :
        secret : liste
        i : entier

    secret <- []
    Pour _ : 1 à 5
        i <- random.randint(1, 6)
        secret.ajouter(mm.TabCouleur[i])
    Fin Pour
    Retourner secret
Fin Fonction
```

Variables

Nom	Type	Rôle
secret	list	Contient la combinaison secrète
i	int	Contient un nombre aléatoire entre 1 et 6

Tests

> Ici, nous n'avons pas de test pour la fonction creerCombinaisonSecrete() car la fonction utilise des nombres aléatoires et les doctests ne peuvent pas être utilisés pour tester des fonctions qui utilisent des nombres aléatoires.
> Cependant, nous pouvons dire que la fonction creerCombinaisonSecrete() fonctionne correctement, pour en être sûr, on l'a `print()`puis testé manuellement plusieurs fois.

tuples()

```
# Fonction nb bien placé / nb mal placé
def tuples(a,secret) -> tuple:
    """
    Permet de déterminer le nombre de bien placé et de mal placé
    :param a: list
    :param secret: list
    :return: tuple

    >>> tuples([1, 2, 3, 4, 5],[1, 2, 3, 4, 5])
    (5, 0)
    >>> tuples([1, 2, 3, 4, 5],[1, 2, 3, 4, 6])
    (4, 0)
    >>> tuples([1, 2, 3, 4, 5],[2, 1, 3, 4, 5])
    (3, 2)
    >>> tuples([1, 2, 3, 4, 5],[1, 1, 6, 6, 6])
    (1, 0)
    """

    countGood: int = 0
    countBad: int = 0
    counts: dict = {}

    for i in secret:
        if i not in counts:
            counts[i] = 0
        counts[i] += 1

    for i,j in zip(a, secret):
        if i == j:
            countGood += 1
            counts[i] -= 1

    for i in a:
```

```
            if i in counts and counts[i] > 0:
                countBad += 1
                counts[i] -= 1


        return (countGood, countBad)
```

Description

> La fonction tuples() a pour but de déterminer le nombre de couleurs bien placées et le nombre de couleurs mal placées.
> Elle prend en paramètre la combinaison de l'utilisateur (a) et la combinaison secrète (secret) et retourne un tuple contenant le nombre de couleurs bien placées et le nombre de couleurs mal placées.
>
> La fonction commence par initialiser les variables countGood et countBad à 0, et la variable counts à un dictionnaire vide.
> Ensuite, elle parcourt la combinaison secrète et compte le nombre de fois que chaque couleur apparaît dans la combinaison secrète.
> Puis, elle parcourt les deux combinaisons (a et secret) en même temps et compare les couleurs, si elles sont identiques, elle incrémente countGood et décrémente le nombre de fois que la couleur apparaît dans la combinaison secrète.
> Enfin, elle parcourt la combinaison de l'utilisateur et compte le nombre de couleurs mal placées en vérifiant si la couleur est dans la combinaison secrète et si elle n'a pas déjà été utilisée.

Pseudo-code

```
Fonction tuples(a, secret) : tuple
    Variables locales :
        countGood : entier
        countBad : entier
        counts : dictionnaire

    Pour i dans secret
        Si i n'est pas dans counts Alors
            counts[i] <- 0
        counts[i] += 1
    Fin Pour

    Pour i, j dans zip(a, secret)
        Si i = j Alors
            countGood += 1
            counts[i] -= 1
        Fin Si
    Fin Pour

    Pour i dans a
        Si i dans counts et counts[i] > 0 Alors
            countBad += 1
            counts[i] -= 1
```

```
            Fin Si
        Fin Pour

        Retourner (countGood, countBad)
    Fin Fonction
```

Variables

Nom	Type	Rôle
a	list	Contient la combinaison de l'utilisateur
secret	list	Contient la combinaison secrète
countGood	int	Contient le nombre de couleurs bien placées
countBad	int	Contient le nombre de couleurs mal placées
counts	dict	Contient le nombre de fois qu'appaît chaque couleur dans la combinaison secrète

Tests

> Ici, nous avons quatre tests pour la fonction tuples(), la fonction ne vérifiant pas l'intérieur des tuples contenu dans les listes, on peut donc se permettre de tester la fonction avec des int et non des tuples dans les listes.
> Les tests sont les suivants :
>
> - Dans le premier test, la combinaison de l'utilisateur et la combinaison secrète sont identiques, la fonction doit donc renvoyer (5, 0).
> - Dans le deuxième test, la combinaison de l'utilisateur et la combinaison secrète ont 4 couleurs identiques, la fonction doit donc renvoyer (4, 0).
> - Dans le troisième test, la combinaison de l'utilisateur et la combinaison secrète ont 3 couleurs bien placées et 2 couleurs mal placées, la fonction doit donc renvoyer (3, 2).
> - Dans le quatrième test, la combinaison de l'utilisateur et la combinaison secrète ont 1 couleur bien placée et 0 couleurs mal placées, la fonction doit donc renvoyer (1, 0).
>
> Une fois les doctests effectués, nous pouvons dire que la fonction tuples() fonctionne correctement.

Prog()

```python
# Programme principale
def prog() -> None:
    """
    Programme principale

    :return: None
    """

    # Initialisation des variables
    count = 1
```

```
    screen: pygame.display = creerScreen()
    secret: list = creerCombinaisonSecrete()
    mm.afficherPlateau(screen)
    mm.afficherChoixCouleur(screen)
    victory: bool = False

    # Tant que le jeu est en cours
    while True:
        for event in pygame.event.get():
            if event.type == pygame.QUIT:
                pygame.quit()
                sys.exit()

        while not victory:
            count += 1
            a: list = mm.construireProposition(screen, count)
            victory: bool = afficherVictoire(secret, a, screen, count)
            mm.afficherResultat(screen, tuples(a, secret), count)

            if victory:
                # Affiche la combinaison secréte en cas de victoire
                mm.afficherCombinaison(screen, a, count)
                mm.afficherSecret(screen, secret)
```

Description

> La fonction prog() est le programme principal du jeu.
> Elle commence par initialiser les variables nécessaires au jeu, puis elle crée l'écran de jeu, la combinaison secrète, affiche le plateau de jeu, les pastilles de choix de couleurs et initialise la variable victory à False.
>
> Ensuite, elle entre dans une boucle infinie qui va permettre de jouer au jeu. Dans cette boucle, elle vérifie si l'utilisateur a cliqué sur la croix pour fermer la fenêtre, si c'est le cas, elle ferme la fenêtre et quitte le programme.
>
> Ensuite, elle entre dans une autre boucle qui va tourner tant que la variable victory est à False, c'est-à-dire tant que le joueur n'a pas gagné ou perdu. Dans cette boucle, le programme va a chaque tour :
>
> - Incrémenter le nombre de coups joués. `count += 1`
> - Construire la proposition de l'utilisateur. `a = mm.construireProposition(screen, count)`
> - Vérifier si le joueur a gagné ou perdu. `victory = afficherVictoire(secret, a, screen, count)`
> - Afficher le résultat de la proposition de l'utilisateur. `mm.afficherResultat(screen, tuples(a, secret), count)`
>
> Si le joueur a gagné ou perdu, le programme affiche la combinaison secrète et la combinaison de l'utilisateur.

Pseudo-code

```
Procédure prog :
    Variables locales :
        count : entier
        screen : pygame.display
        secret : liste
        victory : booléen
        a : liste

    count <- 1
    screen <- creerScreen()
    secret <- creerCombinaisonSecrete()
    mm.afficherPlateau(screen)
    mm.afficherChoixCouleur(screen)
    victory <- Faux

    Tant que Vrai
        Pour event dans pygame.event.get()
            Si event.type = pygame.QUIT Alors
                pygame.quit()
                sys.exit()
            Fin Si
        Fin Pour

        Tant que non victory
            count += 1
            a <- mm.construireProposition(screen, count)
            victory <- afficherVictoire(secret, a, screen, count)
            mm.afficherResultat(screen, tuples(a, secret), count)

            Si victory Alors
                mm.afficherCombinaison(screen, a, count)
                mm.afficherSecret(screen, secret)
            Fin Si
        Fin Tant que
    Fin Tant que
Fin Procédure
```

Variables

Nom	Type	Rôle
count	int	Contient un compteur de coups joués, il est initialisé à 1
screen	pygame.display	Contient l'écran de jeu
secret	list	Contient la combinaison secrète
victory	bool	Contient un booléen qui indique si la partie est terminée ou non
a	list	Contient la combinaison de l'utilisateur

Tests

> Ici, nous n'avons pas de test pour la fonction prog() car c'est le programme principal du jeu et il est difficile de tester un programme principal. Cependant, nous pouvons dire que la fonction prog() fonctionne correctement, elle permet de jouer au jeu Mastermind.

Répartition des tâches

Répartition des tâches dans le code

- **Picouleau Louis**
    - Fonction `distance`
    - Fonction `determinerVictoire`
    - Fonction `afficherVictoire`
    - Fonction `prog
- **Lustremant Matthis**
    - Fonction `creerScreen`
    - Fonction `tuples`
    - Fonction `prog`
    - Fonction `creerCombinaisonSecrete`

Répartition des tâches dans le rapport

- **Picouleau Louis**
    - Introduction
    - Description de la fonction `distance`
    - Description de la fonction `determinerVictoire`
    - Description de la fonction `afficherVictoire`
- **Lustremant Matthis**
    - Description de la fonction `creerScreen`
    - Description de la fonction `creerCombinaisonSecrete`
    - Description de la fonction `tuples`
    - Description de la fonction `prog`