IUT - Maubeuge
BUT – Informatique

Année universitaire 2024-2025
Groupe BUT 1 TPB1

Compte Rendu - SAE 101
Implémentation d’un besoin client
Mastermind

Collot Grégoire
Marecaille-Henaut Mattieu

iut VALENCIENNES
CAMBRAI • MAUBEUGE

Sommaire

Introduction

Nous sommes étudiants en 1ère année de BUT Informatique à l'IUT de Maubeuge.
Dans le cadre de la ressource SAE 101, nous avons mis en place le jeu Mastermind afin de répondre au besoin d'un client.

Nous allons donc d'abord vous montrez notre moyen utilisé pour comprendre les fonctions donné dans le fichier mm.py ainsi que du package de Pygame.

Nous continuerons avec les fonctions mise en place pour compléter le projet et répondre au besoin du client.

Par la suite, nous parlerons de la mise en place de la fonction principales ainsi que des nouvelles fonctions pour faire fonctionner Mastermind correctement.

Dernièrement, nous parlerons des différent logiciel que nous avons utiliser pour travailler ensemble sur ce projet ainsi que les problèmes rencontrer.

Nous vous remercions d'avance, pour le temps que vous nous accordez en lisant ce compte-rendu.

Comprendre le fonctionnement

Pygame :

Tout d'abord, ayant déjà une base sur laquelle travailler, nous avons dû la comprendre.

Pour cela, nous nous sommes renseigner sur le fonctionnement de Pygame.
Grâce à la documentation, nous avons pu comprendre qu'il fallait initier pygame avec .init().
Après nous avons du créer une surface avec une taille x, y.

```
pygame.display.set_mode((1000, 750))
```

Nous pouvons ainsi mettre à jour l'affichage de Pygame afin d'avoir un rendu de notre fenêtre en utilisant pygame.display.update().

Pour que la fenêtre continue à tourner avant que l'utilisateur ne quitte le programme, nous devons créer une boucle qui détectera l'action de l'utilisateur voulant fermer le programme.

```
while() :
    for event in pygame.event.get():
        if event.type == pygame.QUIT:
```

Ce qui nous permettra de fermer correctement Pygame à temps voulus.

Nous avons donc appris les bases suffisantes pour maintenant tester les fonctions données dans le fichier mm.py.

mm.py :

Pour cela nous avons tout simplement appeler chaque fonction une par une afin de comparer le rendu de la fonction avec elle-même.

Ainsi, nous savons que :

mm.afficherPlateau() permet d'afficher le modèle du plateau à l'écran.

mm.afficherChoixCouleur() montre les propositions de couleurs possibles ainsi que diverses informations pour l'utilisateur.

mm.construireProposition() permet d'ouvrir une phase de récupération de 5 couleurs de la part du joueur.

mm.afficherResultat() permet d'afficher au joueur le résultat de sa proposition. Cette fonction va nécessiter un ajout de notre part pour son bon fonctionnement.

mm.afficherSecret() permet d'afficher le code secret à trouver.

mm.afficherCombinaison() est appelé par la fonction construireProposition() pour afficher la proposition du joueur à l'écran.

mm.getChoixCouleur() est appelé par la fonction construireProposition() pour récupérer la couleur choisie.

Enfin, mm.distance() est appelé par la fonction précédente, mais celle-ci n'est pas complète nous allons voir par la suite comment nous l'avons complété.

Nous venons de comprendre la base donnée pour ce projet. Nous avons donc pris un temps pour réfléchir sur comment implémenter les besoins du client manquant, et de prévoir le nombre de fonctions nécessaire pour réaliser le projet.

Fonctions mise en place :

Pour le besoin du programme nous devons importer les bibliothèques Pygame, Random et le fichiers mm.py données au début de la SAE.

Pour le Mastermind, nous devons créer plusieurs fonctions pour réaliser son bon déroulement.

Fonction randomSecret()

Pour commencer, nous avions besoin d'une fonction permettant de créer notre combinaison secrète :

Pour cela, nous prendrons aléatoirement dans le tab couleurs (liste des couleurs disponibles) 5 aléatoirement, nous utilisons la fonction random.randint() permettant de sélectionner une couleur aléatoirement dans la liste des couleurs (grâce à leur index) puis de les ajouter dans notre liste secretCode que nous retournons.

Pseudo-code de la fonction :

```
Fonction randomSecret():Liste
        secretCode:Liste
        Pour i:0 à 5:
                secretCode ← ajouter (mm.TabCouleur[random.randint(0, 7)])
        return secretCode
FIN
```

Variable	Type	Utilité
SecretCode	List	Permet d'avoir la liste des couleurs.

Fonction checkanswerPlayer()

Le joueur devra donner une proposition et nous avons donc besoin d'une fonction qui a pour but de vérifier si la proposition du joueur à des pions corrects a notre secret et sinon combien y a-t-il de pions mal placés.

Pour cela ont créé une fonction checkanswerPlayer() qui prend pour paramètres la proposition du joueur (tuple) , et une liste (notre code secret).
Notre but est de comparer les 5 éléments des deux listes.

D'abord, nous vérifions si les deux mêmes index des deux listes sont identiques, nous ajoutons alors 1 au compteur des biens placés puis on rajoute l'index correspondant dans nos liste alreadycheckSecret et alreadyCheckPlayer, tout deux des listes vides à l'origine.

Puis nous devons vérifier s'il y a des pions mal placés en faisant attention au doublon.
Ainsi, pour chaque élément dans notre liste secretCode nous les comparons chacun à leur tour avec tous les éléments de la propositionduJoueur, si l'élément de secretCode correspond a un élément d'index différents dans notre proposition, et SI les index ne sont pas déjà dans nos listes alreadycheck et alreadycheckplayer , alors on ajoute 1 au compteur des mal placés et nous ajoutons les index de notre liste secretCode dans alreadycheckSecret et l'index de notre proposition dans notre liste alreadycheckPlayer.

Après avoir vérifié tous les éléments de Secretcode nous retournons le tuple du compteur de bien placé et mal placé.

Pseudo-code de la fonction :

```
Fonction checkAnswerPlayer(propJoueur: liste, codeSecret:liste) : entier
        variable locales:
        wellPlaced: entier
        wrongPlaced: entier
        alreadyCheckSecret: liste
        alreadyCheckPlayer: liste

        wellPlaced ← 0
        wrongPlaced ←0
        alreadyCheckSecret ← []
        alreadyCheckPlayer ← []

        pour i : 0 à 5 :
                si secret[i]  ==  propPlayer[i] alors
                        wellPlaced ← wellPlace + 1
                        alreadyCheckSecret ajouter i
                alreadyCheckPlayer ajouter i
```

```
        pour i : 0 à 5:
                pour i : 0 à 5:
                        si secret[i] == propPlayer[k] et i n'est pas dans alreadyCheckSecret et k
                        n'est dans alreadyCheckPlayer alors
                                wrongPlace ← wrongPlaced + 1
                                alreadyCheckSecret ajouter i
                        alreadyCheckPlayer ajouter k
        renvoyer (wellPlace, wrongPlaced)
FIN
```

Variable	Type	Utilité
PropJoueur	Liste contenant des tuples	Permet de savoir ce que le joueur a choisie comme couleur
CodeSecret	Liste contenant des tuples	Permet d'avoir le code secret
wellPlaced	Entier	Permet d'avoir le nombre de point bien placé
WrongPlaced	Entier	Permet d'avoir le nombre de point mal placé
alreadyCheckSecret	List	Permet de savoir les positions du codeSecret qui ont déjà été validé
alreadyCheckPlayer	List	Permet de savoir les positions de la propJoueur qui ont déjà été validé

Fonction distance() $$\mathrm{dist}(A, B) = \sqrt{(x_2 - x_1)^2 + (y_2 - y_1)^2}$$

Pour que le joueur construise sa proposition, nous avons besoin qu'il interagisse avec les boutons prévus à cet effet, il nous faut donc savoir si, quand l'utilisateur clic avec sa souris, si le clic est dans le cercle représentant la couleur qu'il a choisie.

Pour cela, nous devions compléter une fonction distance() prenant en paramètres une liste A (coordonnée x,y de l'épicentre du cercle) et la liste B contenant les coordonnées de la souris pour calculer la distance entre la souris et l'épicentre du bouton de la couleur choisie, le rayon du bouton étant de 15 pixel, si la distance entre les deux-points est supérieur à 15 alors la souris n'est pas sur le bouton et il ne se passe donc rien.

Pseudo-code de la fonction :

```
Fonction distance(a:list,b:list):reel
    renvoyer sqrt((a[0] - b[0]) ** 2 + (a[1] - b[1]) ** 2)
FIN
```

Variable	Type	Utilité
a	liste	Permet d'avoir le x et y de la souris
b	liste	Permet d'avoir le x et y d'un button

Fonction afficherFin()

Lorsque le joueur gagne ou dépasse le nombre de coup , on affiche un message grâce à notre fonction afficherfin(), qui prend en paramètres la surface de l'écran , le texte qu'il recevra en fonction de si le joueur a gagné ou perdu, et la taille du texte. De plus, elle affichera notre message à la position que nous avons choisi ici en (150,700), puis fera une mise à jour de l'affichage, dès que cette fonction est utilisée cela signifie que la partie est terminée donc nous faisons apparaître la combinaison secrète avec afficherSecret().

Pseudo-code de la fonction :

Fonction afficherFin(f: pygame.Surface, text:Texte, fontSize: Entie): None

```
    myfont: pygame.font
    label:   myfont

    myfont ← pygame.font.SysFont("monospace", fontSize)
    label ← myfont.render(text, 1, mm.noir)

    f.blit(label, (150, 700))
    pygame.display.update()
    mm.afficherSecret(screen, secretCode)
FIN
```

Variable	Type	Utilité
Myfont	Pygame.font	Permet de mettre en place le type de texte pour Pygame
Label	Myfont.render	Permet d'assembler le texte dans un élément visuel pour Pygame

Fonction principale

Venant de mettre en place toutes les fonctions nécessaires à la bonne réalisions de Mastermind. Il ne nous reste plus qu'a lier le tout à travers notre fonction principale main().

Celle-ci va être en trois partie,

La première étant juste la mise en place de Pygame pour son bon fonctionnement ainsi que l'appel de la création du code secret avec randomSecret(), suivi de la mise en place graphique du jeu avec mm.afficherPlateau() et mm.afficherChoixCouleur().

Ensuite, nous devons créer la boucle permettant le bon dérouler de la partie, celle-ci continuera à prendre des essais du joueur tant qu'il n'a pas trouvé, perdu ou quitter Pygame.

Pour vérifier si le joueur n'a pas perdu ou gagner, nous incrémentons une variable correspond au nombre d'essais du joueur, ainsi tant que celle-ci est inférieure à 16, il peut continuer. Et pour savoir s'il a gagné, il suffit de vérifier que la fonction check() renvoie bien 5 pions bien placés.

Enfin, nous affichons si le joueur à gagner ou perdu tout en révélant le nombre de coups et la combinaison secrète grâce à afficherFin(). Le joueur n'a plus qu'à quitter le jeu.

Notre jeu est maintenant terminé.

Pseudo-code de la fonction :

```
Function main():None
        Variable locales:
        secretBroken: Boole
        gameEnded: Boole
        userTry: Entier
        running: Boole
        doItOnce: Boole
        screen: pygame
        codeSecret: liste

        secretBroken ← Faux
        gameEnded ← Faux
        userTry ← 2
        running ← Vrai
        doItOnce ← Faux
        codeSecret ← randomSecret()
```

```
pygame.init()
screen ← pygame.display.set_mode((1000, 750))
screen.fill(mm.Blanc)

mm.afficherPlateau(screen)
mm.afficherChoixCouleur(screen)

Tant que running est vrai alors:
    Pour chaque évenement de pygame:
        Si evenement est pygame.QUIT alors
            afficher("C'est Fini")
            running ← Faux
            pygame.quit()
        Si SecretNotBroken est faux et gameEnded est Faux et running est vrai
        alors:
            propJoueur: list
            check: tuple
            propJoueurs ← mm.construireProposition(screen, userTry)
            check ← checkAnswerPlayer(propJoueur, SecretCode)
            mm.afficherResultat(screen, check, userTry)
            Si userTry == 16 alors:
                gameEnded ← Vrai
            Sinon si check[0] == 5 alors:
                secretBroken ← Vrai
            Sinon:
                userTry ← userTry + 1
        Si secretBroken est Vrai et doItOnce est faux:
            text: text
            text ← "Bravo ! Vous avez trouvé la combinaison en "
                + str(userTry – 1) + " coups !"
            afficherFin(screen, text, 18)
            doItOnce ← Vrai
        Sinon gameEnded est Vrai et doItOnce est faux alors
            text: text
            text ← "Perdu, vous avez dépassé vos 15 coups."
            afficherFIn(screen, text, 18)
            doItOnce ← Vrai
FIN
```

Variable	Type	Utilité
secretBroken	Boole	Permet de savoir si le joueur a trouvé la bonne solution
gameEnded	Boole	Permet de savoir si le joueur à fini la partie en dépassant les 15 essais
userTry	Entier	Permet de savoir à combien de coups, se trouve le joueur ainsi que la position de ses essais.
running	Boole	Permet de savoir si le Pygame est en cours d'utilisation
doItOnce	Boole	Permet d'effectuer une condition qu'une seule fois
screen	Pygame	"écran" pygame qui permet d'afficher la zone de jeu
propJoueur	Liste de Tuple	Liste des couleurs (tuple) que le joueur a sélectionné
CodeSecret	Liste de Tuple	Liste généré aléatoirement permettant d'avoir le code secret
check	Tuple	Permet d'avoir le nombre de pion bien placé et mal placé
text	String	Permet d'afficher le texte de fin de jeu.

Fonction Test

Test checkAnswerPlayer()

Durant notre projet nous avons créé la fonction de vérification de couleurs bien et mal placé qui nous semblâmes correct puis au fur et à mesure de nos tests, nous avons remarqué que pour certains cas particuliers étaient erronés, pour faciliter la détection du problème nous avons créé une fonction test avec une combinaison secrète fixe et des propositions diverses, notamment les propositions qui nous ont valu des erreurs.

Ce système de test nous permet ainsi de détecter plus facilement les erreurs notamment le fait qu'à l'origine nous avions un problème avec les doublons et les mal placés et de vérifier que notre solution est correcte.

Nos différentes propositions sont donc testées et si elles sont validées, c'est-à-dire si on obtient le résultat attendu, vérifié auparavant alors le teste renvoie OK sinon Kaput suivis de l'erreur renvoyer et celle attendue.

Ce qui nous a permis d'améliorer notre fonction pour qu'elle soit parfaitement fonctionnelle.

Test distance()

Pour la distance, nous avons décidé de prendre en considération le cas où le clic du joueur serait en dehors de la fenêtre, mais pour autant capter par Pygame. C'est le cas où le résultat serait inférieur ou égal à 0.

Mais aussi nous avons vérifié tous les autres cas possible où la distance les deux-points peuvent être inférieurs ou supérieurs à l'autre.

Ainsi si la fonction de test nous renvoie bien ses résultats, nous pouvons être serein à l'idée de son bon fonctionnement.

Travaille en équipe

Dès le début de notre projet, nous nous sommes dit qu'il fallait utiliser nos PC personnelles afin de pouvoir apprendre le bon fonctionnement de Python de notre côté, c'est pour cela que nous avons dû apprendre à installer un environnement Python et d'y rajouter la bibliothèque Pygame.

En effet, par défaut, Pygame n'est pas présent dans la base de Python. Pour cela sous-Windows, nous avons utilisé Anaconda qui est un logiciel permettant d'avoir une interface simple et efficace pour de la gestion d'environnement. Il a suffi de créer un nouvel environnement nommé SAE101 et d'y installer Pygame.

Sous Debian, Anaconda a créé des conflits d'environnements, nous avons dû utiliser tout simplement le terminal pour créer manuellement un environnement avec

« python3.12 -m venv SAE_101 »

Il suffit ensuite d'activer l'environnement : « source env/bin/activate »,

Et d'installer Pygame avec « pip install pygame ».

De plus pour pouvoir communiquer et voir le code de chacun facilement, nous avons mis en place un répertoire Github. Nous avons ainsi l'historique de chaque modification et la facilité de comprendre le code de chacun.

Étant donné que Grégoire à des connaissances en algorithme, nous avons décidé de faire les tâches de notre côté, mais à chaque moment nécessaire nous prenons le temps d'avoir la même vision d'ensemble et des étapes.

Grâce à cela Mattieu a pu comprendre l'entièreté du programme et a pus faire lui-même les fonctions nécessaires pour le bon fonctionnement du programme.

Nous avions au final, le même déroulé de programme, juste une typo différente.

Il nous suffis de mettre en commun les versions les plus lisibles, fonctionnel et clair.

Conclusion

Ce projet a pu nous apporter les réflexions nécessaires en arrivant sur un projet aillant une base déjà présente. À travers nos différents problèmes rencontrés, nous avons pu trouver des solutions et mettre en place les diverses fonctions nécessaires pour la bonne réponse au besoin du client.

En tant qu'étudiants en informatique, cette SAE est un passage nécessaire et important pour la suite de nos études afin de comprendre comment mettre en place un ensemble de fonction pour réaliser une tâche précise tout en respectant les besoins d'un client.

De plus, nous avons aussi appris à mettre en place un répertoire Github ainsi qu'un environnement Python.

Nous vous remercions du temps que vous avez pris pour nous lire,

Collot Grégoire,
Marecaille-Henaut Mattieu.