**Compte rendu**

**Table des matières**

[I – Introduction 1](#_Toc151908570)

[**Dobble** 1](#_Toc151908571)

[**Démarche du plan** 1](#_Toc151908572)

[II – Création du jeu 1](#_Toc151908573)

[**Explication de la class « Cartes »** 1](#_Toc151908574)

[**Explication de la class « Dobble »** 1](#_Toc151908575)

[III - Conclusion 1](#_Toc151908576)

# **I – Introduction**

Notre projet consiste à programmer un jeu de cartes en python parmi tant d’autres en utilisant le paradigme de programmation orienté objet.

## 

## **Dobble**

Le Dobble est un jeu de cartes basé sur la perception visuelle et la rapidité qui se compose de cartes rondes avec différentes images. Chaque carte a plusieurs images, et deux cartes quelconques ont exactement une image en commun. Le but du jeu est d'identifier plus rapidement que les autres joueurs la paire d'images communes entre deux cartes. Les joueurs doivent chercher la paire d'images communes entre leur carte du dessus et celle au centre de la table. Le premier joueur qui identifie le nom de l'image commune et pose sa carte au centre, recouvrant ainsi la carte précédente. Une nouvelle carte est ainsi exposée au centre pour les autres joueurs. Les joueurs répètent le processus, cherchant la paire entre leur nouvelle carte et celle au centre. Le jeu continue ainsi jusqu'à ce qu'un joueur ait posé toutes ses cartes. Le joueur qui pose toutes ses cartes en premier est déclaré vainqueur.

**Démarche du plan**

Pour commencer, notre Dobble sera constitué de 40 images différentes, parmi lesquelles 5 images seront choisies aléatoirement pour chaque carte. Chaque joueur possédera 15 cartes, et étant donné qu'il y a 4 joueurs dans notre jeu, le jeu sera composé de 60 cartes au total. Nous allons organiser le code en deux classes : la classe 'Carte', qui sera responsable de la création des cartes Dobble, et la classe 'Dobble', qui gérera le lancement du jeu en suivant toutes les règles établies.

# **II – Création du jeu**

## **Explication de la class « Cartes »**

Une image contenant texte, capture d’écran, Police

Description générée automatiquement

## 

## Tout d'abord, nous importons le module **random,** dont nous aurons besoin par la suite. Ensuite, nous créons la classe **"Cartes".** Cette classe prend un paramètre, **"nomJoueur",** qui spécifie le nom du joueur associé au paquet de cartes. La liste **"symboles"** contient les différents symboles qui peuvent apparaître sur les cartes.

Nous effectuons un choix aléatoire, **"choix",** pour sélectionner un indice dans la liste des symboles, et nous utilisons une liste temporaire appelée **"carte\_temp".** Ensuite, une boucle externe **("for i in range(15)")** est utilisée pour créer **15 cartes.**

À l'intérieur de cette boucle externe, une boucle interne **("for j in range(5)")** est employée pour ajouter 5 symboles à chaque carte. Avant d'ajouter un symbole à la carte, nous vérifions qu'il n'est pas déjà présent dans la liste temporaire "**carte\_temp".** Une fois la carte complétée, elle est ajoutée à la liste **"paquetDeCartes"** de l'instance de la classe **"Cartes".**

Enfin, la **liste** temporaire **"carte\_temp"** est réinitialisée pour la prochaine carte. Ainsi, la classe **"Cartes"** représente un paquet de cartes avec des symboles générés de manière aléatoire.

## **Explication de la class « Dobble »**

Une image contenant texte, capture d’écran, Police, affichage

Description générée automatiquement

On crée la classe "Dobble". Cette classe a un constructeur, "**init**", qui prend les joueurs de la partie comme paramètres, à savoir **"p1, p2, p3, p4"**. Chacun de ces joueurs est associé à un attribut correspondant, tel que **self.p1, self.p2, self.p3, et self.p4**. Lors de la création d'une instance de la classe Dobble, les noms des joueurs sont passés en tant qu'arguments.

L'attribut **self.carteDepart** représente la carte de départ pour la partie. Il est initialement une liste vide et sera complété ultérieurement par la méthode **carteDeDepart.** L'attribut **self.symboles** est une liste de symboles possibles sur les cartes. Cette liste contient une variété de symboles représentés par des chaînes de caractères, tels que "cœur", "sandale", etc. Ces symboles seront utilisés pour générer les cartes de départ ainsi que les cartes des joueurs.

La méthode **carteDeDepart** de la classe Dobble a pour responsabilité de créer une carte initiale pour le jeu. Elle sélectionne au hasard cinq symboles parmi une liste prédéfinie, en évitant les doublons dans la carte. Voici comment cela fonctionne :

La variable **choixSymbole = random.randint(0, 39)** génère un index aléatoire entre 0 et 39 inclus, correspondant à un symbole dans la liste prédéfinie **symboles**. Ensuite, la boucle **for i in range(5)** parcourt cette liste cinq fois pour choisir cinq symboles.

Le **while self.symboles[choixSymbole] in self.carteDepart** vérifie si le symbole correspondant à l'index choisi aléatoirement est déjà présent dans la carte de départ (**carteDepart**). Si c'est le cas, cela signifie qu'un doublon a été trouvé. Dans ce cas, un nouvel index aléatoire est généré jusqu'à ce qu'un symbole unique soit trouvé.

En utilisant **self.carteDepart.append(self.symboles[choixSymbole])**, le symbole unique est ajouté à la carte de départ. En fin de compte, **return self.carteDepart** renvoie la carte de départ mise à jour, qui est maintenant une liste de cinq symboles uniques.

Une image contenant texte, capture d’écran, Police

Description générée automatiquement

La méthode **tour** de la classe Dobble représente le mécanisme du jeu, décrivant comment le jeu fonctionne en prenant le "joueur" comme argument. Tout d'abord, le programme affiche le nom du joueur (**joueur.nomJoueur**) et les cartes qu'il possède. Pour cela, une boucle **for i in range(len(joueur.paquetDeCartes))** est utilisée pour itérer sur chaque carte du paquet du joueur, affichant son numéro et son contenu.

Ensuite, une variable temporaire, **choixCarte\_tempo**, est créée pour obtenir le numéro de la carte que le joueur souhaite jouer via une saisie utilisateur. Une boucle **while choixCarte\_tempo...** garantit que le choix est valide, c'est-à-dire dans la plage des cartes disponibles. Après avoir obtenu le numéro de la carte choisie, la carte correspondante est récupérée (**choixEnCours**) en utilisant l'index obtenu à partir du choix du joueur.

Un booléen, **detecteur**, est initialisé à False pour vérifier la présence de symboles communs. Une autre boucle **for i in range(len(choixEnCours))** itère sur les symboles de la carte choisie. Si un symbole est présent dans la carte de départ, le détecteur devient True.

Ensuite, le programme affiche un message indiquant que le joueur a posé sa carte, met à jour la carte de départ (**self.carteDepart = choixEnCours**), retire la carte jouée du paquet du joueur (**joueur.paquetDeCartes.pop(choixCarte\_tempo - 1)**), et affiche la nouvelle carte de départ. Si aucun symbole commun n'est trouvé, le joueur pioche une nouvelle carte en utilisant un système similaire à celui utilisé dans la méthode **carteDeDepart**.

Enfin, la méthode renvoie la nouvelle carte de départ.

Une image contenant texte, capture d’écran, logiciel

Description générée automatiquement

# 

La méthode **partie** de la classe Dobble coordonne l'intégralité d'une partie en prenant en paramètre les quatre joueurs, notés **p1, p2, p3, p4**. Le processus commence par une boucle principale débutant avec l'instruction **while len(self.vainqueurs) != 4:**. Cette boucle continue à s'exécuter tant que le nombre de joueurs déclarés vainqueurs n'atteint pas quatre, ce qui signifie que trois joueurs doivent vider leur paquet de cartes, tandis que le quatrième joueur continue de jouer. À chaque itération :

* Si un joueur a vidé son paquet de cartes, il est ajouté à la liste des vainqueurs.
* Si le paquet de cartes d'un joueur n'est pas vide, la méthode **tour** est appelée pour simuler son tour de jeu.
* Si le nombre de joueurs déclarés vainqueurs atteint trois et que le joueur actuel n'est pas déjà parmi eux, il est ajouté à la liste des vainqueurs.

Cette boucle se répète jusqu'à ce que le nombre de joueurs déclarés vainqueurs atteigne quatre, indiquant que trois joueurs ont vidé leur paquet, tandis que le quatrième joueur continue de jouer. La méthode **partie** se termine en renvoyant la liste des joueurs déclarés vainqueurs à l'aide de l'instruction **return self.vainqueurs**.

# **III - Conclusion**

Faire un jeu comme Dobble en programmation n'est pas une tâche simple, et ce code propose une approche pratique pour résoudre le problème. Il peut être utilisé comme point de départ pour des améliorations futures, surtout en ce qui concerne la facilité d'utilisation, la flexibilité et la gestion des erreurs. Comprendre comment fonctionne le jeu et être capable de le transformer en code sont des étapes vraiment importantes dans le processus de création de jeux. Ce code montre bien ces compétences. En le retravaillant un peu plus et en faisant quelques ajustements, il pourrait devenir encore plus solide et plus facile à comprendre pour d'autres développeurs.