Projet: UNO

Point historique

À l'origine du jeu de société, il y a Merle Robbins, barbier américain dans la ville de Reading dans l'Ohio. Selon les rumeurs populaires, il fut créé avec l'aide de son fils Ray Robbins suite à une querelle qu'ils avaient à propos des règles du jeu du 8 américain, mais également de sa femme Marie. Il décida alors de créer ce jeu en se basant sur ce jeu, mais y ajoutant une nouvelle touche esthétique. Les cartes originales du jeu ont été dessinées sur la table de leur salle à manger.

Voyant que le jeu plait à sa famille, il décide d'économiser et d'investir dans la fabrication de la première version du jeu. Le budget initial était de 8000 dollars pour 5000 jeux de cartes. Ils ont commencé à le vendre de son salon de coiffure et dans des boutiques locales. Après 10 ans, le jeu Uno commence à devenir populaire et cède les droits à la société International Games pour la modique somme de 50.000$ avec une commission de 10 cents par copie vendue. L'inventeur Merle Robbins décédera seulement 3 ans ensuite.

https://jeu-uno.info/conseils-et-techniques/histoire-du-uno/

Sommaire

Règles officielles

Au début de chaque manche, chaque personne tire une carte au hasard dans la pioche. Celle qui a le chiffre le plus élevé devient le donneur de la manche. Toute carte avec un symbole compte pour zéro.

Le donneur bat les cartes et en distribue 7 aux autres joueurs. Les cartes restantes, placées faces cachées, forment la pioche. La dernière carte de celle-ci est retournée face visible au centre de la table. Elle constitue le talon et la base de la défausse.

Attention : les joueurs n'ont pas le droit de regarder le recto de leurs cartes (distributeur compris) avant que le donneur n'ait retourné la première carte. Toute personne qui enfreint cette règle se voit contrainte d'ajouter deux cartes supplémentaires à son jeu. Si elle récidive et regarde ces deux nouvelles cartes, la sanction se répète.

Le jeu démarre dans le sens des aiguilles d'une montre, en partant du premier joueur à gauche de celui qui a distribué. La carte supérieure du talon doit être recouverte par une carte ayant :

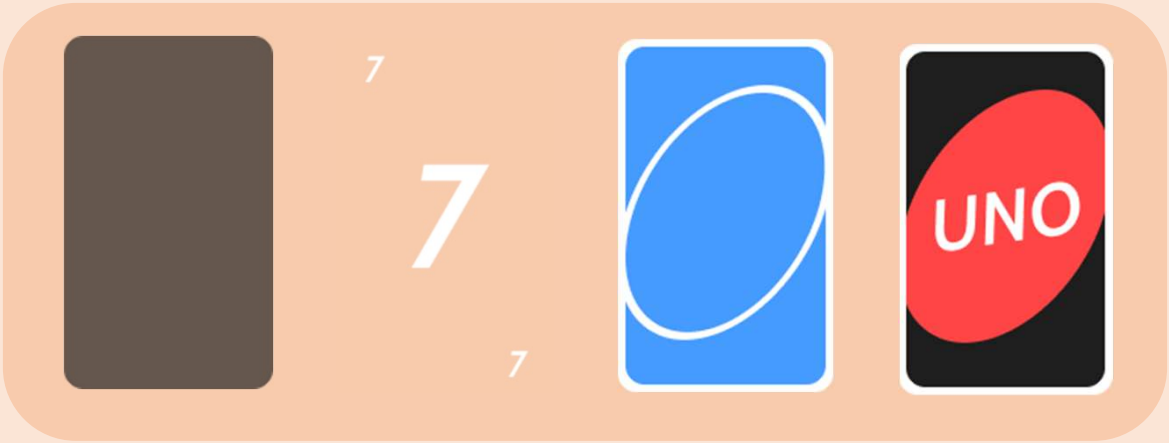
le même numéro ou la même couleur. Par exemple, si la pile de défausse contient une carte rouge qui est un 3, le participant peut placer soit une carte rouge, soit une carte avec un 3 de n'importe quelle couleur ;
le même symbole que la défausse (s'il s'agit d'une carte spéciale).
Remarque : si un joueur a une carte « Joker » ou « Super joker », il peut la poser sur n'importe quelle carte.

Lorsqu'un participant ne peut pas jouer, il pioche. S'il peut jouer la carte qu'il vient de piocher, il le fait. S'il ne peut pas, il la garde dans son jeu et passe son tour. Il est également possible de ne pas jouer une carte jouable. Dans ce cas de figure, il convient de tirer une carte de la pioche. Si elle est jouable, le participant peut l'utiliser immédiatement.

https://www.joueclub.fr/contenu/les-regles-officielles-du-uno.html

Fichier:	unocard.py
Objet:	UNOCard
Hérité de:	tkinter.Canvas
Méthodes:	12
Fichier importé:	tkinter, constant

Les cartes sont composées de 4 parties superposables



Etat désactivé

Valeur

Fond(couleur)

Dos

Constructeur(__init__)

Paramètres:

parent: objet de tkinter dans lequel se place une carte
symbol: valeur de la carte
color: couleur de la carte

Constructeur de base

```
Canvas.__init__(
    self, parent,
    height=CardConstant.Height.value,
    width=CardConstant.Width.value,
    highlightthickness=0,
    bg = '#557C55'
)
```

constructeur de la classe de base. Initialisation des dimensions de la fenêtre et de la couleur de fond

Variable membre

Stockage des paramètres du constructeur et creation d'un attribut pour conserver l'état (actif ou non) de la carte

```
self.symbol: Symbol = symbol
self.color: Color = color
self.disableState : bool = None
```

Import des images

```
self.background = PhotoImage(file=f'{bgPath}{self.color.value}')
self.symbolCard = PhotoImage(file=f'{smblPath}{self.symbol.value}')
self.backCard = PhotoImage(file=f'{smblPath}{Symbol.back.value}')
self.disableFilter = PhotoImage(file=f'{bgPath}{Symbol.disable.value}')
```

Nous importons chaque partie de la carte, elles sont stockée dans un objet **PhotoImage** (tkinter class)

Insertion des images

```
self.backwardId: int = self.create_image((w, h), image=self.background)
self.forwardId: int = self.create_image((w, h), image=self.symbolCard)
self.backId: int = self.create_image((w, h), image=self.backCard)
self.disableId: int = self.create_image((w, h),image=self.disableFilter)
```

Nous chargeons toutes les images dans la classe

```
self.disable(False)
```

Enfin, nous rendons la carte active

https://www.youtube.com/watch?v=GrsUwOiuotY&ab_channel=TipsJazzInferno

Méthodes de vérification

```
def __eq__(self, card) -> bool: ...
def isValid(self) -> bool: ...
def isSpecialCard(self) -> bool: ...
def isJokerCard(self) -> bool: ...
def isSkipCard(self) -> bool: ...
```

```
def isDisable(self) -> bool: ...
def isSameColor(self, card) -> bool: ...
def isSameSymbol(self, card) -> bool: ...
```

Nous avons ensuite implémenter d'autres méthodes booléennes afin d'identifier la nature de la carte

hide

Paramètres:

hidden: booléen qui indique la face de la carte à afficher

```
a, b = ('hidden', 'normal') if hidden else ('normal', 'hidden')

self.itemconfigure(self.forwardId, state=a)
self.itemconfigure(self.backId, state=b)
```

Cette fonction vérifie le paramètre **hidden**:

- Si **hidden** = True alors le dos de la carte est affiché
- Sinon le dos de la carte est masqué

https://stackoverflow.com/questions/53499669/how-to-hide-and-show-canvas-items-on-tkinter

disable

Paramètres:

disable: booléen qui indique si la carte dois être désactivé

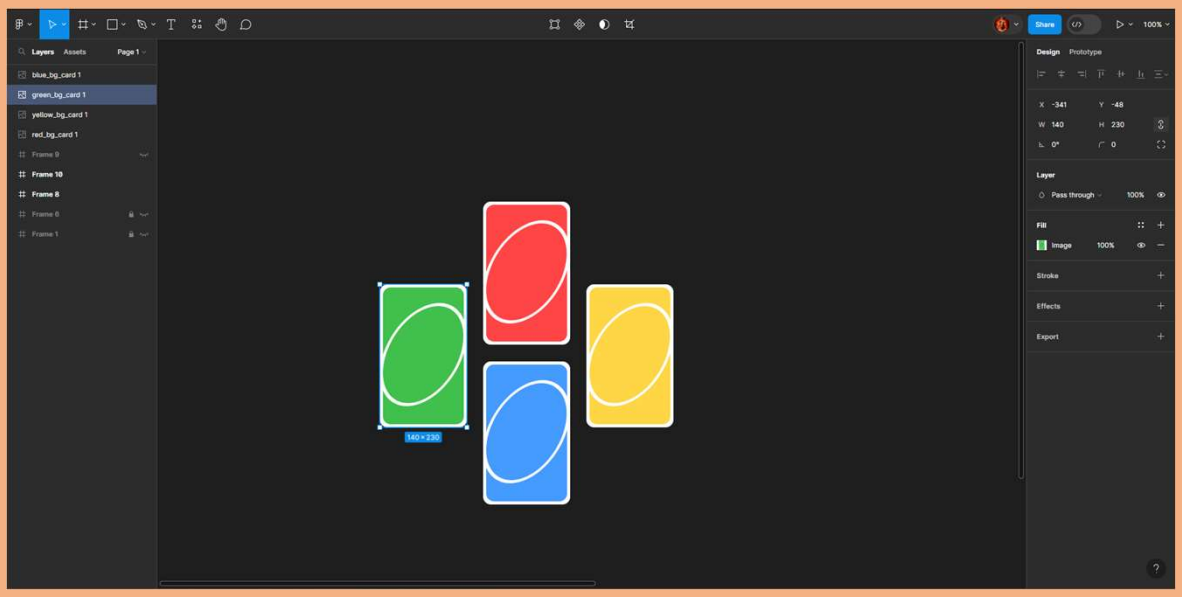
```
a = 'normal' if disable else 'hidden'
self.itemconfigure(self.disableId, state=a)
self.disableState = False
```

Cette fonction vérifie le paramètre **disable**:

- Si **disable** = True alors la carte est grisée et deviens non jouable

Création des cartes

Pour réaliser le visuel des cartes, nous avons utilisé le site internet Figma dédié a la création d'interface graphique
https://www.figma.com



Notre palette de couleur est la suivante:

Jaune:	FFD646	Rouge:	FF4646
Vert:	41C04E	Bleu:	469BFF
Noir:	1C1C1C	Blanc:	FFFFFF

Police d'écriture: Cabin Semi Bold Italic 64pixels

Fichier:	deck.py
Objet:	Deck
Hérité de:	list
Méthodes:	7
Fichier importé:	unocard

validCards

Paramètres:

card: carte de référence

```
self.disable()

for i in self:
    sameColor: bool = i.isSameColor(card)
    sameSymbol: bool = i.isSameSymbol(card)
    playerJokerCard: bool = i.isJokerCard()

    if sameSymbol: i.enable()
    elif sameColor: i.enable()
    elif playerJokerCard: i.enable()
```

Désactive toutes les cartes dans un premier temps puis active les cartes du deck qui sont des jokers/ +4 sont de même couleur ou de même valeur que la carte de référence

getLast

```
return self[-1]
```

Renvoi le dernier élément du deck

empty

```
return not(self)
```

Renvoi True si le deck est vide sinon False

disable

```
for i in self:
    i.disable()
```

Désactive toutes les cartes du deck

Remove

Paramètres:

card: carte à retirer

```
index: int = self.getIndex(card)
if not(index == None):
    return self.pop(index)
return None
```

Récupère l'indice de la carte dans le deck. S'il existe alors La carte présente à cet indice est retirée du deck

getIndex

Paramètres:

card: carte à chercher

```
index = 0
for i in self:
    if i == card:
        return index
    index+=1
return None
```

Parcours toutes les cartes en incrémentant le compteur **index** si la carte à chercher est dans le deck alors **index** est retourné

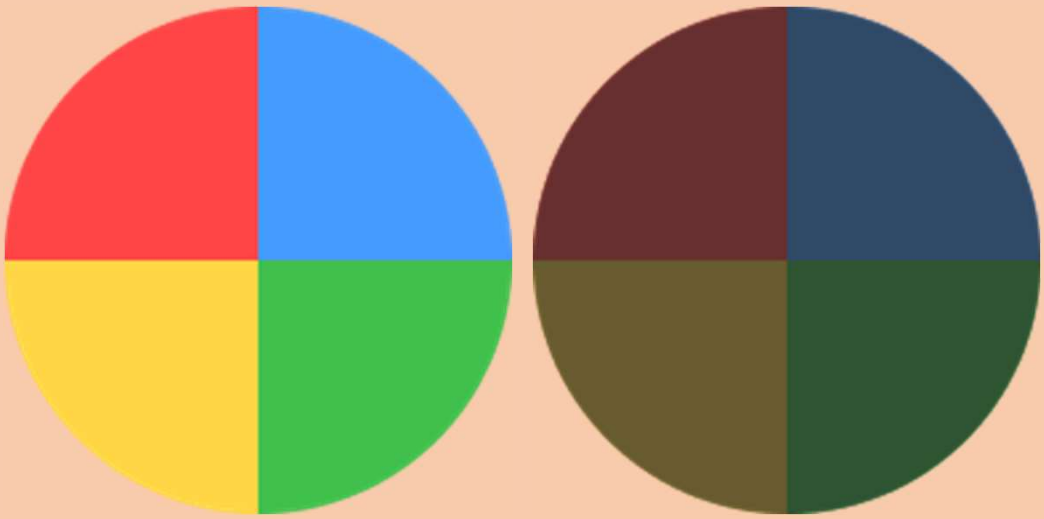
destroy

```
for i in self:
    i.place_forget()
    i.pack_forget()
self.clear()
```

Parcours toutes les cartes et les retires de l'interface puis détruit les cartes

Fichier:	selector.py
Objet:	ColorSelector
Hérité de:	tkinter.Canvas
Méthodes:	8
Fichier importé:	tkinter, constant

Choisir une couleur après avoir joué un Joker ou un +4



Activé

Désactivé

Constructeur(__init__)

Paramètres:

parent: objet de tkinter dans lequel se place une carte

Variable membre

```
self.bgColor : str = '#A6CF98'
self.value = IntVar()

self.l : list[Canvas] = [None] * 4
self.corner : list[PhotoImage] = [None] * 4
self.disableImg : list[PhotoImage] = [None] * 4
self.disableId : list[int] = [None] * 4
self.enableId : list[int] = [None] * 4
```

Initialisation des attributs. **self.l** est une liste contenant chaque bouton

Constructeur de base

```
Canvas.__init__(
    self, parent,
    height=250, width=250,
    highlightthickness=0, bg=self.bgColor
)
```

Attribue les dimensions et la couleur de fond du sélecteur de couleur

Variable interne

```
color : list[str] = ['Red', 'Green', 'Blue', 'Yellow']
pos : list[tuple] = [(30, 33), (125, 128), (125, 33), (30, 128)]
path : str = 'src/ColorSelection/'
disableColor : list[str] = ['R', 'G', 'B', 'Y']
```

Ces variables seront utile dans le bloc de code suivant elles décrivent la couleur et la position e chaque bouton

Insertion des images

```
for i in range(4):
    self.l[i] = Canvas(
        self, height=95, width=95,
        highlightthickness=0, bg=self.bgColor
    )
```

Pour chaque élément de la liste de boutons nous créons un **Canvas** au dimensions du boutons

```
self.disableImg[i] = PhotoImage(file=f'{path}disable{disableColor[i]}.png')
self.corner[i] = PhotoImage(file=f'{path}{color[i]}.png')

self.enableId[i] = self.l[i].create_image((47.5, 47.5), image=self.corner[i])
self.disableId[i] = self.l[i].create_image((47.5, 47.5), image=self.disableImg[i])

self.l[i].place(x=pos[i][0], y=pos[i][1])
```

Nous importons les images puis les insérons dans le **Canvas** créé précédemment
Enfin nous plaçons chaque élément à une position x, y défini dans la liste de tuple **pos**

```
self.disableAll()
```

Nous désactivons tout les boutons

Placement du sélecteur

```
self.place(
    x = ScreenSize.Width.value - 250,
    y = (ScreenSize.Height.value - 250) / 2
)
```

Nous affichons le sélecteur dans la fenêtre principale à droite et centré verticalement

disable

Paramètres:

color: nombre associé à la couleur

disable: booléen qui indique l'état du bouton

```
a = 'normal' if disable else 'hidden'
self.l[color].itemconfigure(self.disableId[color], state=a)
```

Désactive la couleur associé si disable = True

disableAll

Paramètres:

disable: booléen qui indique l'état du sélecteur

```
for i in range(4):
    self.disable(i, disable)
```

Désactive tout les boutons si disable = True

onClick

Paramètres:

index: indice associé à la couleur d'un bouton

```
self.value.set(index)
self.disableAll()
self.enable(index)
```

Affecte l'indice à l'attribut **value** puis désactive toutes les couleur et active la couleur sélectionnée

activate

Paramètres:

activate : booléen qui indique si le sélecteur est cliquable

```
if activate:
    for i in range(4):
        self.l[i].bind('<Button>', lambda e, j=i: self.onClick(j))
else:
    for i in range(4):
        self.l[i].unbind('<Button>')
```

Si **activate** = True alors affecter le callback correspondant sinon retirer les callbacks

getValue

```
return self.value
```

Renvoi le chiffre associé à la couleur sélectionner

Fichier:	constant.py
Objets:	ScreenSize, CardConstant, Color, Symbol, PositionY
Hérité de:	Enum
Fichier importé:	enum

Dans ce fichier est listé toutes les valeurs constantes utilisée dans le programme

```
#@verify(UNIQUE) # uncomment only if python > python 3.7
class ScreenSize(Enum): ...

#@verify(UNIQUE) # uncomment only if python > python 3.7
class CardConstant(Enum): ...

#@verify(UNIQUE) # uncomment only if python > python 3.7
class Color(Enum): ...

#@verify(UNIQUE) # uncomment only if python > python 3.7
class Symbol(Enum): ...

#@verify(UNIQUE) # uncomment only if python > python 3.7
class PositionY(Enum): ...
```

Le décorateur **@verify(UNIQUE)** est à décommenter uniquement si votre version de python est supérieur à python3.7, il sert de sécurité supplémentaire afin de s'assurer que chaque élément d'une énumération est unique

Fichier:	unoparty.py
Objets:	UNOParty
Méthodes:	23
Fonctions:	3
Hérité de:	tkinter.Canvas
Fichier importé:	random, Tkinter, constant, selector, deck, pygame

Préparation des outils audio

```
mixer.init()
takeCardSound = mixer.Sound('src/song/takecard.wav')
playCardSound = mixer.Sound('src/song/playcard.wav')
winSound = mixer.Sound('src/song/win.wav')
failSound = mixer.Sound('src/song/fail.wav')
unoSound = mixer.Sound('src/song/uno.wav')
unoVoiceSound = mixer.Sound('src/song/unovoice.wav')
music = mixer.Sound('src/song/music.wav')
```

Nous initialisons le mixer de **pygame** puis nous chargeons chaque fichier dans un canal du mixer. Nous avons choisi d'utiliser les modules audio de **pygame** car ils permettent de jouer simultanément deux sons

NB: les fichiers audio sont en .wav pour garantir la compatibilité avec edupython

Les sons du jeu proviennent de ce site:
https://pixabay.com

La voix « UNO » proviens de YouTube:
https://www.youtube.com/shorts/9EWiEZC8Jrs

createStack

Paramètres:

draw: deck symbolisant la pioche

```
temp = Deck()
for i in range(len(draw)):
    if not (draw[i].isSpecialCard() or draw[i].isJokerCard()):
        temp.append(draw.pop(i))
        return temp, draw
```

Ici nous créons la pile de carte, nous prenons la première carte de la pioche. Si cette carte comporte une action spéciale alors nous l'ignorons et vérifions la carte suivante. Sinon nous l'ajoutons sur la pile et renvoyons la pile et la pioche

createDeck

Paramètres:

draw: deck symbolisant la pioche

```
temp = Deck()
temp.extend([draw.pop(0) for i in range(7)])
return temp, draw
```

Cette fonction retire 7 cartes du début de la pioche et les place dans le deck puis le deck et la pioche sont renvoyé

createDraw

Paramètres:

parent: widget parent

variables

```
draw = Deck()
validSymbol = list(Symbol)[1:13]
validColor = list(Color)[:-1]
```

Nous créons la pioche, puis deux listes, respectivement, tout les symboles du UNO (0-9, +2, +4, Joker, passer, changer de sens) et les 4 couleurs de cartes (rouge, bleu, jaune, vert)

Liste de carte

Création de la liste de carte, Nous commençons par la remplir des 4 cartes « 0 » de chaque couleur

```
symbolColorList = [
        (Symbol.n0, Color.Blue),
        (Symbol.n0, Color.Red),
        (Symbol.n0, Color.Green),
        (Symbol.n0, Color.Yellow),
    ]
```

Remplissage de la liste

```
for _ in range(2):
    symbolColorList.extend(
        [
            (Symbol.joker, Color.Black),
            (Symbol.joker, Color.Black),
            (Symbol.plus4, Color.Black),
            (Symbol.plus4, Color.Black)
        ]
    )
    for symbol in validSymbol:
        for color in validColor:
            symbolColorList.append((symbol, color))
```

Nous créons les 4 cartes « Joker » et les 4 « +4 » ainsi que deux exemplaires de chaque cartes (1-9, +2, passer, changer de sens)

Liste de carte

```
random.shuffle(symbolColorList)
draw.extend([UNOCard(parent, i, j) for i, j in symbolColorList])
return draw
```

Nous mélangeons la liste de carte, l'ajoutons au deck créé puis nous retournons le deck

Constructeur(__init__)

Paramètres:

parent: instance principale de tkinter

Constructeur de base

```
Canvas.__init__(
    self, parent,
    width=ScreenSize.Width.value,
    height=ScreenSize.Height.value,
    highlightthickness=0
)
```

constructeur de la classe de base. Initialisation des dimensions de la fenêtre

Variable membre

Définition des attributs pour gérer les interactions du joueur, créé des constantes utiles et les widgets qui serons utilisé

```
self.checker = IntVar()
self.MAX_CARDS_IN_ROW : int = 20
self.AMOUNT_OF_DELAY : int = 700
self.background_image : PhotoImage = None
self.UNOdraw : Deck = None
self.playerDeck : Deck = None
self.botDeck : Deck = None
self.stack : Deck = None
self.uno : Canvas = None
self.unoImg : PhotoImage = None
```

Affichage des widgets statiques

```
self.__setBackground()
self.drawCardImg : UNOCard = UNOCard(self, Symbol.back, Color.Black)
self.selector = ColorSelector(self)
```

Nous affichons le fond du jeu, la pioche et le sélecteur de couleur

__hideCard

```
self.stack.destroy()
self.UNOdraw.destroy()
self.drawCardImg.place_forget()
```

Détruit le deck du joueur et du bot et retire la pioche

__delay

Paramètres:

time_us: temps en milliseconde

```
checker = IntVar()
self.after(time_us, checker.set, 1)
self.wait_variable(checker)
```

Marque une pause de time_us millisecondes

__skipTurn

Paramètres:

deck: deck du personnage entrain de jouer
func: fonction à appeler après avoir passer le tour
delay: booléen qui indique si un délai doit être ajouté

```
self.__refreshGame()
if delay: self.__delay(self.AMOUNT_OF_DELAY)
if not deck.empty(): func()
```

Nous rafraichissons les données du jeu, puis ajoutons un délai, si **delay** = True et si le deck n'est pas vide alors nous appelons **func**

__setBackground

```
self.background_image = PhotoImage(file='src/background.png')
self.create_image((500, 450), image=self.background_image)
```

Nous affichons le fond du jeu

__disablePlayer

```
self.playerDeck.disable()
for i in self.playerDeck: i.unbind('<Button>')
self.drawCardImg.unbind('<Button>')
```

Nous désactivons toutes les cartes du joueur et la pioche

__enablePlayer

```
for i in range(len(self.playerDeck)):
    if self.playerDeck[i].isValid():
        self.playerDeck[i].bind(
        '<Button>', lambda e, j=i: self.checker.set(j),add='+'
        )

self.drawCardImg.bind('<Button>', lambda e : self.checker.set(-1))
```

Nous activons toutes les cartes et actions du joueur ainsi que la pioche

__getFirstValidCard

Paramètres:

deck: deck à vérifier

```
return next((i for i in deck if i.isValid()), None)
```

Retourne la première carte jouable présente dans le deck si rien n'est trouvé alors **None** est retourné

__showDraw

```
self.drawCardImg.place(
    x = (ScreenSize.Width.value
        - CardConstant.Width.value-240)/2,
    y = PositionY.Center.value
)

self.drawCardImg.configure(bg='#9AD0C2')
```

Nous positionnons la pioche sur l'écran et mettons la même couleur d'arrière plan que le fond du jeu

__showStack

```
for i in range(len(self.stack) - 1):
    self.stack[i].place_forget()

self.stack.getLast().unbind("<Button>")
self.stack.getLast().unbind("<Enter>")
self.stack.getLast().unbind("<Leave>")
self.stack.getLast().hide(False)
self.stack.getLast().place(
    x = (ScreenSize.Width.value
        - CardConstant.Width.value + 240) / 2,
    y = PositionY.Center.value
)
self.stack.getLast().configure(bg='#9AD0C2')
```

Nous montrons que la dernière carte de la pile de carte puis nous retirons tout les évènement attaché à cette carte, enfin nous positionnons la carte sur l'écran et faisons coïncider les arrière plans

__setHoverEvent

```
yPos = PositionY.Bottom.value
shift = CardConstant.Shift.value
for i in self.playerDeck:
    i.bind("<Enter>", lambda e,j=i: j.place(y=yPos-5))
    i.bind("<Leave>", lambda e,j=i: j.place(y=yPos))

if len(self.playerDeck) <= self.MAX_CARDS_IN_ROW:
    return

for i in range(len(self.playerDeck) % self.MAX_CARDS_IN_ROW):
    k = self.playerDeck[i]
    k.bind("<Enter>", lambda e, j=k: j.place(y=yPos-shift -5))
    k.bind("<Leave>", lambda e, j=k: j.place(y=yPos-shift))
```

Déplace la carte du joueur de quelques pixels vers le haut lorsqu'elle est survolé par la souris

__showCard

Paramètres:

deck: deck a sélectionner
index: indice de la carte à afficher
position: position verticale
shiftX: décalage horizontale des cartes
shiftY: décalage verticale des cartes
hidden: état actif ou non de la carte

Variable interne

```
_x = (len(deck) - 1) * CardConstant.Shift.value
_x += CardConstant.Width.value
_x = (ScreenSize.Width.value - _x) / 2
```

Calcul du positionnement horizontal du deck

Désactivation de la carte

```
if hidden:
    deck[index].itemconfigure(deck[index].disableId, state='hidden')
```

Nous désactivons la carte si **hidden** = True

positionnement

```
deck[index].hide(hidden)
deck[index].place(
    x = _x  + (index - shiftX) * CardConstant.Shift.value,
    y = position.value - shiftY
)
deck[index].configure(bg='#557C55')
return deck
```

Nous positionnons la carte selon les valeurs précédemment fournies et calculée

interrupt

```
UNOParty.running = False
self.checker.set(-2)
```

Ici nous interrompons la boucle principal du jeu lorsque la fenêtre est fermée

__showDeck

Paramètres:

position: position verticale
deck: deck à afficher
hide: masquer ou non de la carte

Variable interne

```
length: int = len(deck)

shiftY = -CardConstant.Shift.value
if position.value != 0: shiftY = CardConstant.Shift.value
if length < self.MAX_CARDS_IN_ROW: shiftY = 0

iteration: int = length % self.MAX_CARDS_IN_ROW
```

Nous définissons les positions des cartes en fonction de leur nombre

Affichage des cartes

```
for i in range(iteration):
    deck = self.__showCard(deck, i, position, 0, shiftY, hide)

if length < self.MAX_CARDS_IN_ROW : return

for i in range(self.MAX_CARDS_IN_ROW):
    deck = self.__showCard(deck, i + iteration, position, iteration, 0, hide)
```

Nous affichons le premier rang de carte, si le nombre de carte est plus grand que 20 alors on affiche les autres cartes au deuxième range

__UNO

Paramètres:

deck: deck à vérifier

```
if len(deck) != 1: return

mixer.Channel(1).play(unoSound)
mixer.Channel(3).play(unoVoiceSound)

self.uno = Canvas(
    self, width=490, height=340,
    highlightthickness=0, bg='#A6CF98'
)
self.unoImg = PhotoImage(file='src/UNO.png')
self.uno.create_image((245,170), image=self.unoImg)
self.uno.place(relx=.5, rely=.5,anchor= CENTER)
self.__delay(self.AMOUNT_OF_DELAY * 2)
self.uno.place_forget()
```

Nous vérifions s'il reste une carte dans le deck, si oui alors nous jouons les sons **unoSound** et **unoVoiceSound** Puis nous affichons pendant quelques secondes un popup «UNO»

https://stackoverflow.com/questions/260738/play-audio-with-python

__endGame

```
self.__disablePlayer()
out = 'player' if self.playerDeck.empty() else 'bot'
self.__hideCard()
return out
```

Désactive les actions du joueur puis vérifie si le deck du joueur est vide alors il est déclaré gagnant sinon le vainqueur est le bot, enfin nous cachons les cartes puis retournons le vainqueur

__refreshGame

```
self.__showDeck(PositionY.Bottom, self.playerDeck)
self.__showDeck(PositionY.Top, self.botDeck, True)
self.__showStack()
self.__showDraw()
self.__setHoverEvent()
```

Nous affichons le deck du joueur, du bot, la pile de carte, la pioche et activons les évènement de survol de la souris

setupGame

```
self.pack()
UNOParty.running = True
self.UNOdraw = createDraw(self)
self.playerDeck, self.UNOdraw = createDeck(self.UNOdraw)
self.botDeck, self.UNOdraw = createDeck(self.UNOdraw)
self.stack, self.UNOdraw = createStack(self.UNOdraw)
self.__showDraw()
self.__refreshGame()
```

Nous créons la pioche de 108 cartes,
Deux decks de 7 cartes et une pile de 1 carte puis nous rafraichissons les données du jeu

__playCard

Paramètres:

cardplayer: carte à poser
deckPlayer: deck d'où proviens la carte

```
mixer.Channel(0).play(playCardSound)
card: UNOCard = deckPlayer.remove(cardPlayer)
self.stack.append(card)
self.__refreshGame()
return deckPlayer
```

Nous jouons le son **playCardSound** puis nous retirons la carte du deck du joueur puis nous l'ajoutons à la pile de carte. Enfin nous rafraichissons les données du jeu

__takeCard

Paramètres:

deck: deck a remplir

n: nombre de carte à ajouter au deck

```
mixer.Channel(0).play(takeCardSound)
if not(self.UNOdraw):
    self.UNOdraw = self.stack.copy()

for i in range(n):
    card = self.UNOdraw.pop(0)
    card.disable()
    deck.append(card)
self.__refreshGame()
return deck
```

Nous jouons le son **takeCardSound** puis nous vérifions si la pioche est vide alors nous la remplissons avec la pile puis nous ajoutons **n** cartes au deck

__botTurn

Préparation du tour

```
self.botDeck.validCards(self.stack.getLast())

self.__disablePlayer()
```

Nous vérifions s'il y a des cartes jouable dans le deck du bot et désactivons les actions du joueur

Piocher une carte

Nous récupérons la 1er carte valide dans le deck, s'il n'y en as aucune alors le bot pioche

```
card = self.__getFirstValidCard(self.botDeck)

if card is None:
    self.botDeck = self.__takeCard(self.botDeck)
    return
```

Évènement de pioche

```
if self.stack.getLast().symbol == Symbol.plus2:
    self.playerDeck = self.__takeCard(self.playerDeck, 2)

if self.stack.getLast().symbol == Symbol.plus4:
    self.playerDeck = self.__takeCard(self.playerDeck, 4)
```

Si la carte est un **+2** alors le joueur prend 2 cartes
Si la carte est un **+4** alors le joueur prend 4 cartes

Passer un tour

Nous vérifions que la carte posée est une carte **skip**, si oui alors le bot rejoue

```
if self.stack.getLast().isSkipCard():
    self.__skipTurn(self.botDeck, self.__botTurn)
```

Changer la couleur

```
if self.stack.getLast().isJokerCard():
    color: Color = random.choice(list(Color)[:-1])
    self.stack.getLast().color = color
```

Nous vérifions que la carte posée est un **joker** ou un **+4** si oui alors une couleur est tirée au hasard et affectée à la dernière carte posée

__playerTurn

Préparation du tour

```
self.playerDeck.validCards(self.stack.getLast())

self.__enablePlayer()

self.wait_variable(self.checker)
```

Nous vérifions les cartes jouable dans le deck et activons les actions du joueur puis attendons que le joueur clique sur une carte ou la pioche

Fermeture de la fenêtre

Si le joueur ferme la fenêtre alors le tour est annulé

```
if self.checker.get() == -2:
    return False
```

Piocher une carte

```
if self.checker.get() == -1:
    self.playerDeck = self.__takeCard(self.playerDeck)
    return True
```

Si le joueur clique sur la pioche alors une carte lui est donné et le tour est annulé

Jouer la carte

```
card: UNOCard = self.playerDeck[self.checker.get()]
self.playerDeck = self.__playCard(card, self.playerDeck)
```

Passer un tour

```
if card.isSkipCard():
    self.__skipTurn(self.playerDeck, self.__playerTurn, False)
```

Si la c'est une carte **skip** alors le joueur rejoue

Évènement de pioche

```
if card.symbol == Symbol.plus2:
    self.botDeck = self.__takeCard(self.botDeck, 2)

if card.symbol == Symbol.plus4:
    self.botDeck = self.__takeCard(self.botDeck, 4)
```

Si la carte est un **+2** alors le bot prend 2 cartes
Si la carte est un **+4** alors le bot prend 4 cartes

Choisir une couleur

```
self.playerDeck.disable()
self.selector.activate(True)
self.selector.enableAll()

self.selector.wait_variable( self.selector.getValue())
v = list(Color)[self.selector.getValue().get()]
self.stack.getLast().color = v
self.__delay(500)
self.selector.disableAll()
self.selector.activate(False)
self.playerDeck.validCards(self.stack.getLast())
```

Nous désactivons les cartes du joueur puis activons les couleurs et les interactions avec le sélecteur de couleur.

Nous attendons que le joueur clique sur une couleur puis l'affectons a la dernière carte posée

Nous attendons 500 millisecondes

Enfin nous désactivons le sélecteur de couleur et vérifions a nouveau les cartes jouables dans le deck du joueur

loop

`while UNOParty.running:` Boucler tant que l'application est ouverte

Tour du joueur

```
if not(self.playerDeck.empty() or self.botDeck.empty()):
    if not self.__playerTurn():
        return 'exit'

    self.__refreshGame()
    self.__delay(self.AMOUNT_OF_DELAY)
    self.__UNO(self.playerDeck)
else: break
```

Nous vérifions si aucun des decks n'est vide alors Le joueur joue son tour, s'il ne quitte pas le jeu alors les données sont rafraichie, un laps de temps s'écoule puis nous appelons la méthode **__UNO**

Tour du bot

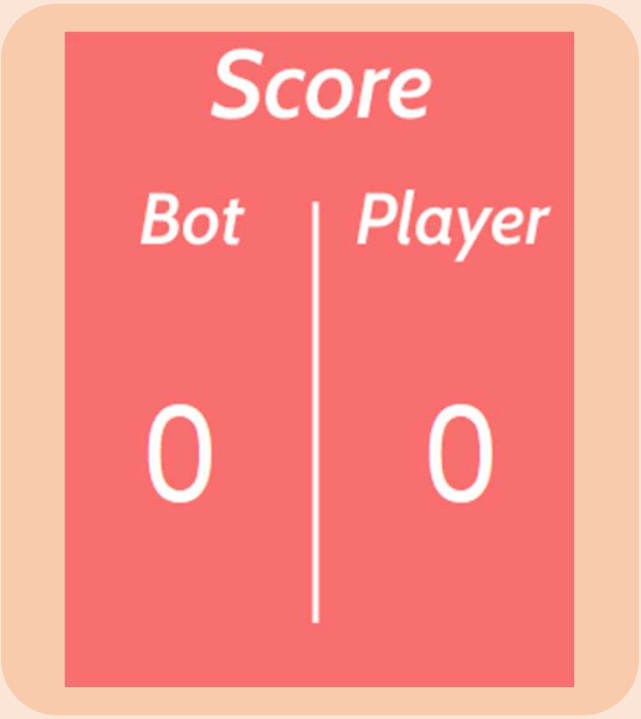
```
if not(self.playerDeck.empty() or self.botDeck.empty()):
    self.__botTurn()
    self.__refreshGame()
    self.__delay(self.AMOUNT_OF_DELAY)
    self.__UNO(self.botDeck)
else: break
```

Nous vérifions si aucun des decks n'est vide alors le bot joue son tour, les données du jeu son rafraichies et la méthode **__UNO** est appelée

`return self.__endGame()` Nous retournons le vainqueur de la partie

Fichier:	main.py
Objets:	ScoreBoard
Méthodes:	2
Hérité de:	tkinter.Canvas
Fichier importé:	Tkinter, unoparty

Tableau des scores



Constructeur(__init__)

Paramètres:

parent: widget dans lequel afficher le score

Constructeur de base

```
Canvas.__init__(
    self, parent,
    width=120,
    height=190,
    highlightthickness=0,
    bg='#FA7070'
)
```

constructeur de la classe de base.
Initialisation des dimensions de la fenêtre et de la couleur de fond

Initialisation de l'attribut score

```
self.score = StringVar(value='0')
```

Affichage du score

```
self.pack_propagate(0)
self.scoreLabel = Label(
    self,
    height=190,
    font=('Cabin', 50),
    bg='#FA7070',
    fg='white',
    textvariable=self.score
)
self.scoreLabel.pack()
```

Nous rendons la taille du label indépendante de c'elle du widget parent (voir lien ci dessous)

Nous créons un **Label** avec la police d'écriture « Cabin » et une taille de police de 50 pixels et de couleur blanche avec comme texte le **score**
Enfin nous affichons le tableau des score

https://stackoverflow.com/questions/16363292/label-width-in-tkinter

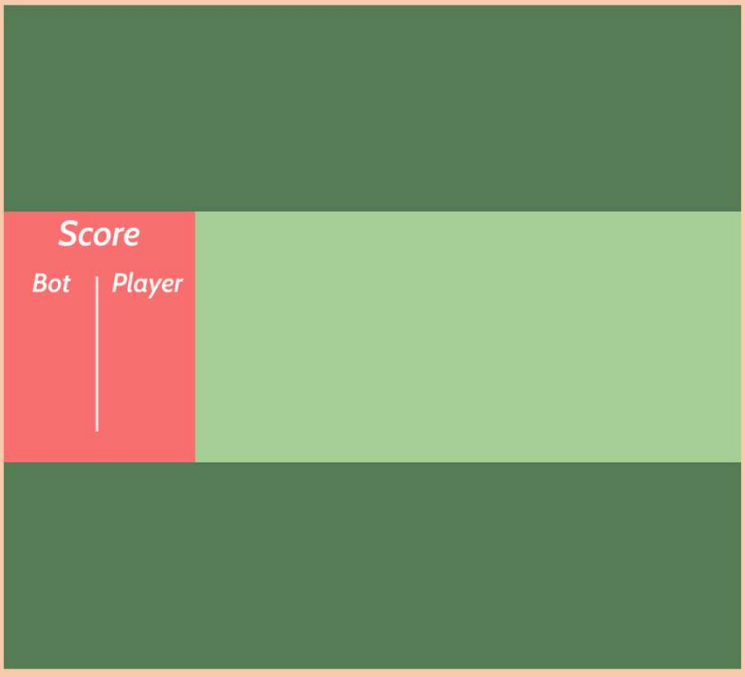
plus1

```
self.score.set(int(self.score.get()) + 1)
```

Nous incrémentons le score de 1

Fichier:	main.py
Objets:	mainWindow
Méthodes:	3
Hérité de:	tkinter.Tk
Fichier importé:	Tkinter, unoparty

Fenêtre principale



Palette de couleur

https://colorhunt.co/palette/f2ffe9a6cf98557c55fa7070

Constructeur(__init__)

Constructeur de base

```
Tk.__init__(self)
self.title('UNO')
self.geometry(f'{ScreenSize.Width.value}x{ScreenSize.Height.value}')
self.resizable(width=0, height=0)
```

constructeur de la classe de base. Initialisation des dimensions de la fenêtre et du titre

Variable membre

Nous créons une partie et l'affichons

```
self.uno = UNOParty(self)
self.uno.pack()
```

Bouton play

```
self.playBtn = Canvas(
    self, width=300, height=153,
    highlightthickness=0, bg ='#A6CF98'
)
self.playImg = PhotoImage(file='src/play.png')
self.playBtn.create_image((150, 76), image = self.playImg)
self.playBtn.place(
    x=(ScreenSize.Width.value - 300) / 2,
    y = (ScreenSize.Height.value - 153) / 2
)
```

Nous importons l'image du bouton Play et le positionnons au milieu de la fenêtre

```
self.playBtn.bind('<Button>', lambda e: self.launchGame())
```

Nous lions la méthode **launchGame** au clique du bouton

Tableau des scores

```
self.botScore = ScoreBoard(self.uno)
self.botScore.place(x=0, y=400)

self.playerScore = ScoreBoard(self.uno)
self.playerScore.place(x=140, y=400)
```

Nous créons l'affichage du score pour le joueur et le bot

Musique

```
mixer.Channel(4).play(music, loops=-1)
```

Nous jouons en boucle la musique principale du jeu

https://stackoverflow.com/questions/42393916/how-can-i-play-multiple-sounds-at-the-same-time-in-pygame

Fermeture de la fenêtre

```
self.protocol("WM_DELETE_WINDOW", self.close)
```

Nous lions la méthode **close** au clique du bouton fermer

close

```
self.uno.interupt()
music.stop()
self.quit()
self.destroy()
```

Nous interrompons la boucle principale, arrêtons la musique et détruisons la fenêtre

launchGame

Mise en place du jeu

```
mixer.Channel(0).play(playCardSound)
self.playBtn.place_forget()
self.uno.setupGame()
winner: str = self.uno.loop()
```

Nous jouons le son d'une carte quand le bouton Play est cliqué puis retirons le boutons et lançons une partie. Enfin nous gardons en mémoire le vainqueur de la partie

Score gagnant/perdant

```
if winner == 'exit': return
elif winner == 'player':
    mixer.Channel(2).play(winSound)
    self.playerScore.plus1()
else:
    mixer.Channel(2).play(failSound)
    self.botScore.plus1()
```

- Si le joueur interrompt la partie alors nous quittons la méthode.

- Sinon si le gagnant est le joueur alors nous jouons le sons **winSound** signifiant la victoire du joueur et incrémentons de 1 son score.

- Sinon nous jouons le son **failSound** et incrémentons de 1 le score du bot

Positionnement du jeu

```
self.playBtn.place(
    x=(ScreenSize.Width.value - 300) / 2,
    y = (ScreenSize.Height.value - 153) / 2
)
```

Nous plaçons le jeu au milieu de la fenêtre

```
uno = mainWindow()
uno.mainloop()
```

Nous créons la fenêtre du jeu et exécutons la boucle principale