A班　花札

プログラミングⅡ

- 主な設計
- ユーザーインターフェース

花札のルール

1. 山札、手札、場札、出来札が存在
2. 手札から1枚カードを出し、場札に同じ月の札があれば獲得
3. 山札から1枚引き、場札と同じ月の札があれば獲得
4. 2. 3どちらも月が合わない場合は場札として置く
5. 手札がなくなるまで2～4を繰り返す

Cardクラス

```
class Card:
def __init__(self, month, kind):
    self.month = month
    self.kind = kind

def __str__(self):
    return f"{self.month}月の{self.kind}"

def __repr__(self):
    return self.__str__()
```

カードが置かれる場所

① 山札

② 場札

③ 手札(2つ)

④ 出来札(2つ)

→カードが独立した状態で扱うのは大変

Cardsetクラス

```
class Cardset:
    def generate_cards(self):
    kinds_by_month = {
    1: ["光", "短冊", "カス", "カス"],
       ~~~~~各月のカードの設定~~~~~~
    12: ["光", "カス", "カス", "カス"],
    }
    self.cards = [Card(month, kind)
    for month, kinds in kinds_by_month.items()
    for kind in kinds]
```

カード管理

```
def __init__(self, cards=None):
    self.cards = cards if cards else []
    self.card_dict = defaultdict(list)
    self.kind_dict = defaultdict(list)
    for card in self.cards:
        self.card_dict[card.month].append(card.kind)
        self.kind_dict[card.kind].append(card.month)
```

役判定

```
def check_yaku(self):
    yaku = []
    match len(self.kind_dict.get("光", [])):
        case 5:
            yaku.append("五光")
        case 4 if 11 in self.kind_dict["光"]:
            yaku.append("雨四光")
        case 4:
            yaku.append("四光")
        case 3:
            yaku.append("三光")
```

役判定

```
match self.kind_dict:
    case hanami if 3 in hanami.get("光", [])
        and 9 in hanami.get("タネ", []):
        yaku.append("花見で一杯")
    case tsukimi if 8 in tsukimi.get("光", [])
        and 9 in tsukimi.get("タネ", []):
        yaku.append("月見で一杯")
```

ユーザーインターフェース


Hanafuda Game