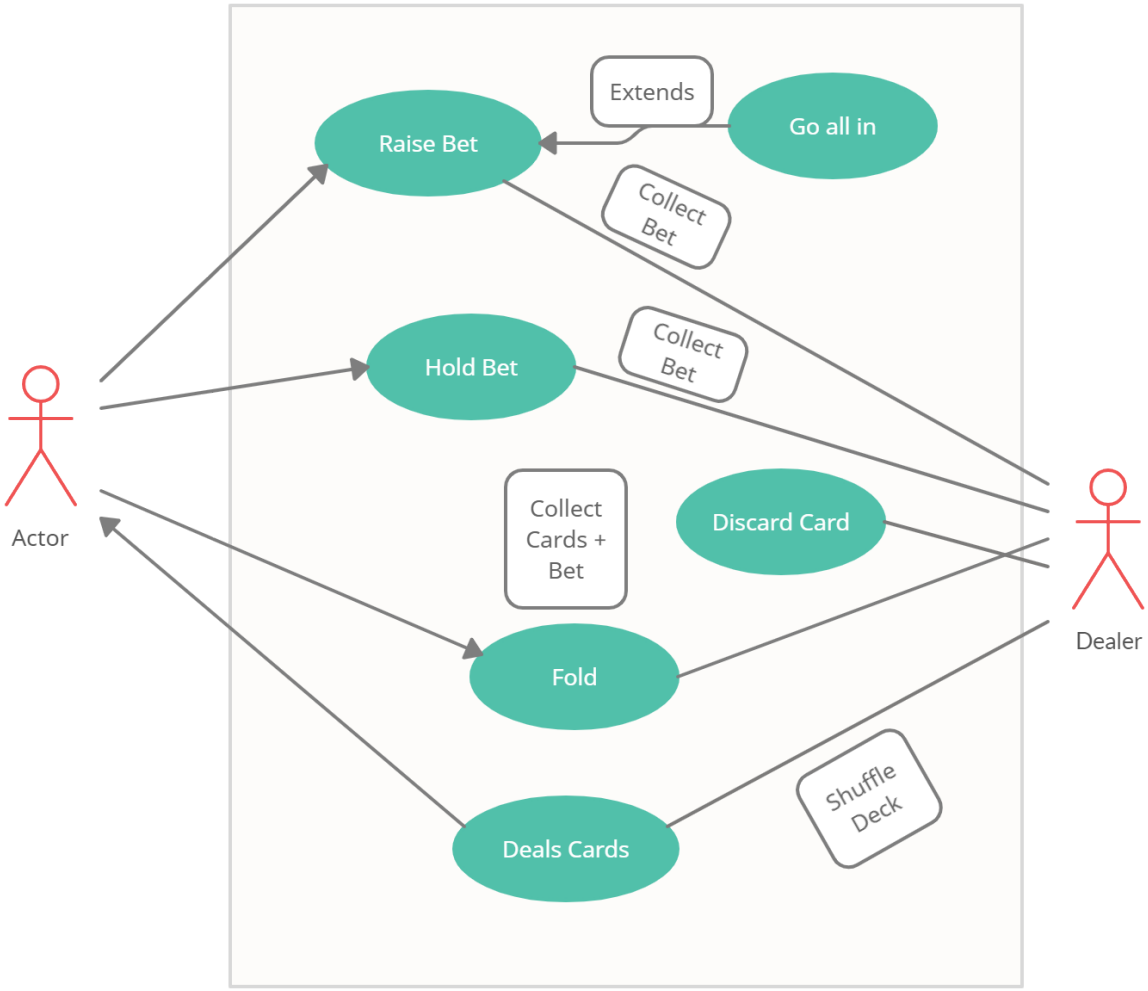
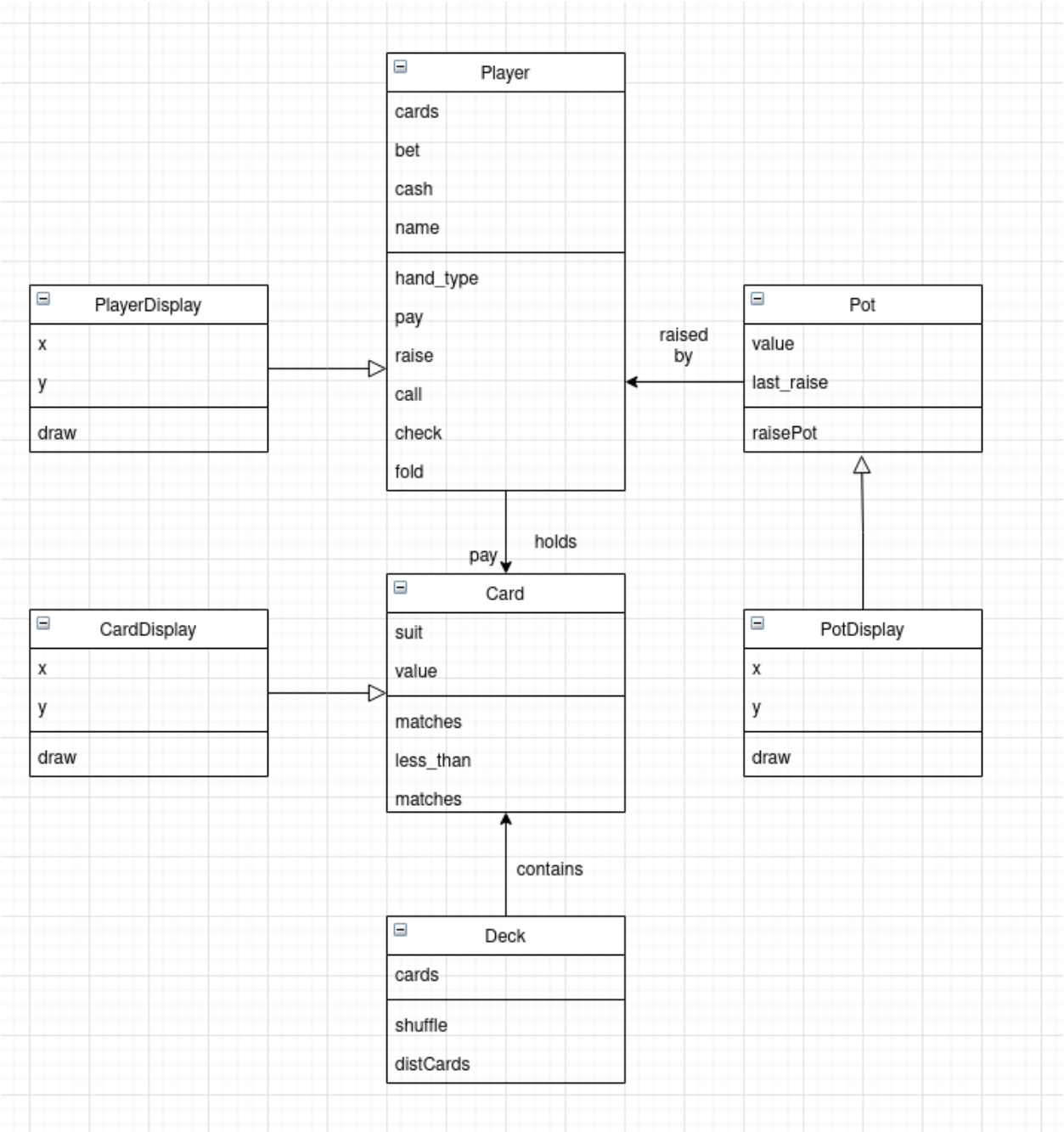
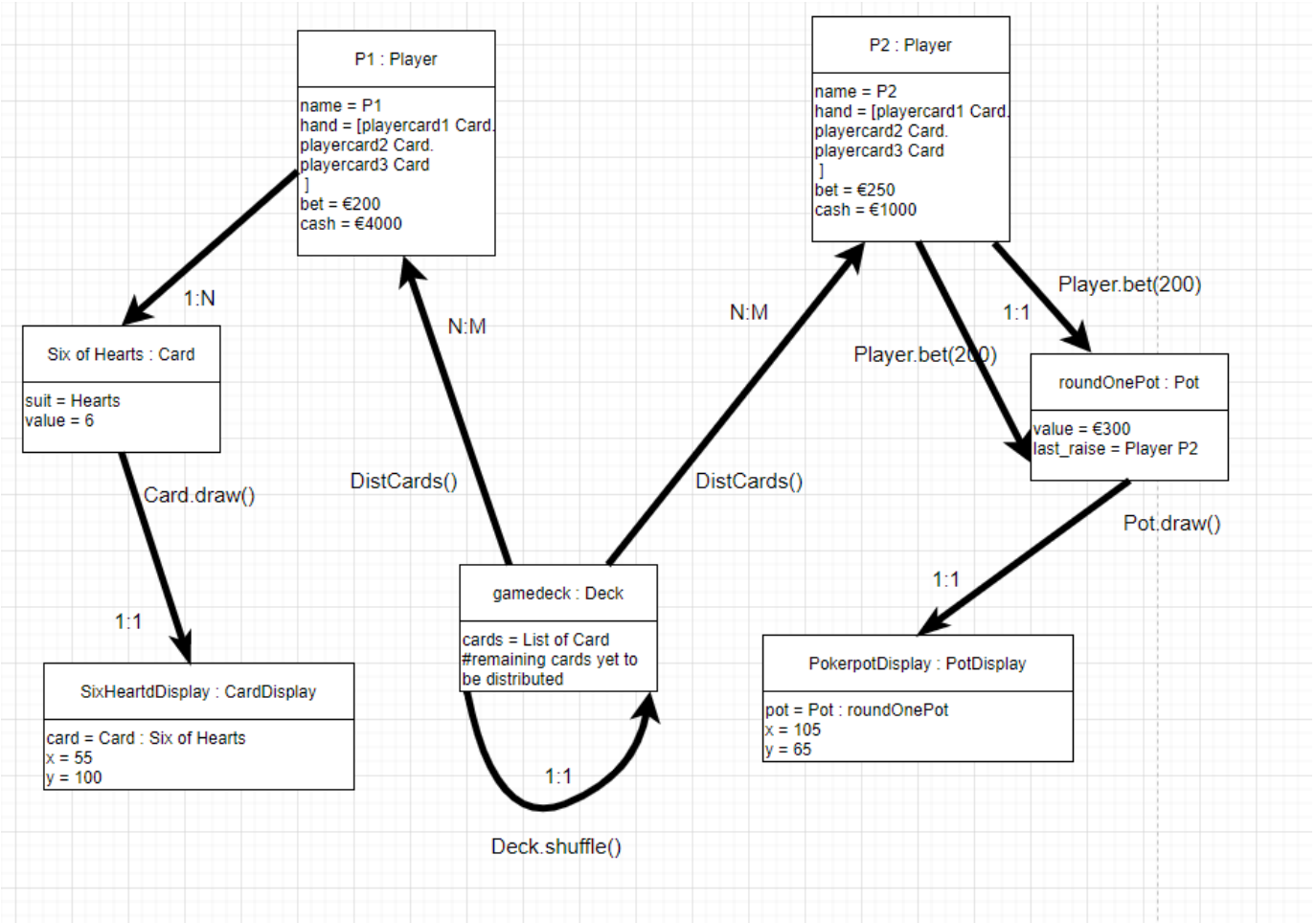
0. If you had problematic Use cases or the Diagram add them in here revised.



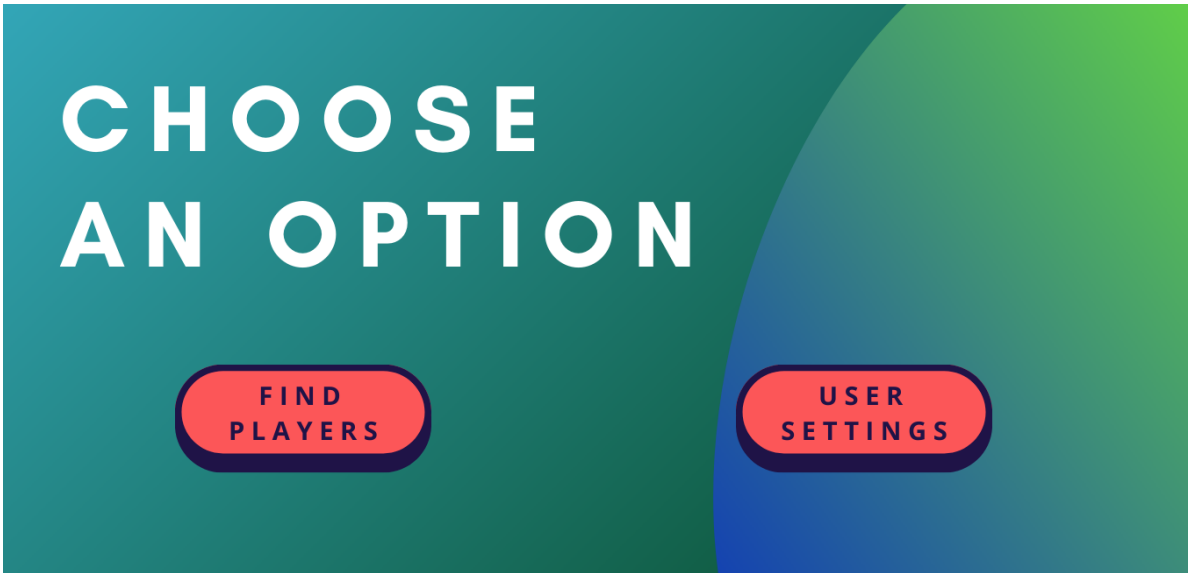
1. Refined class diagrams

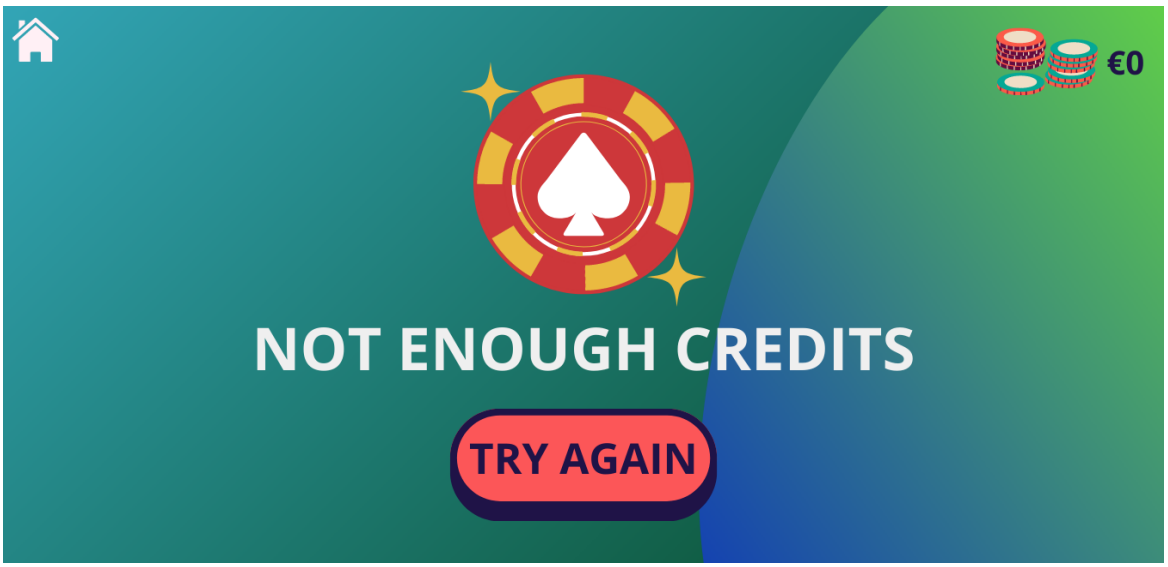


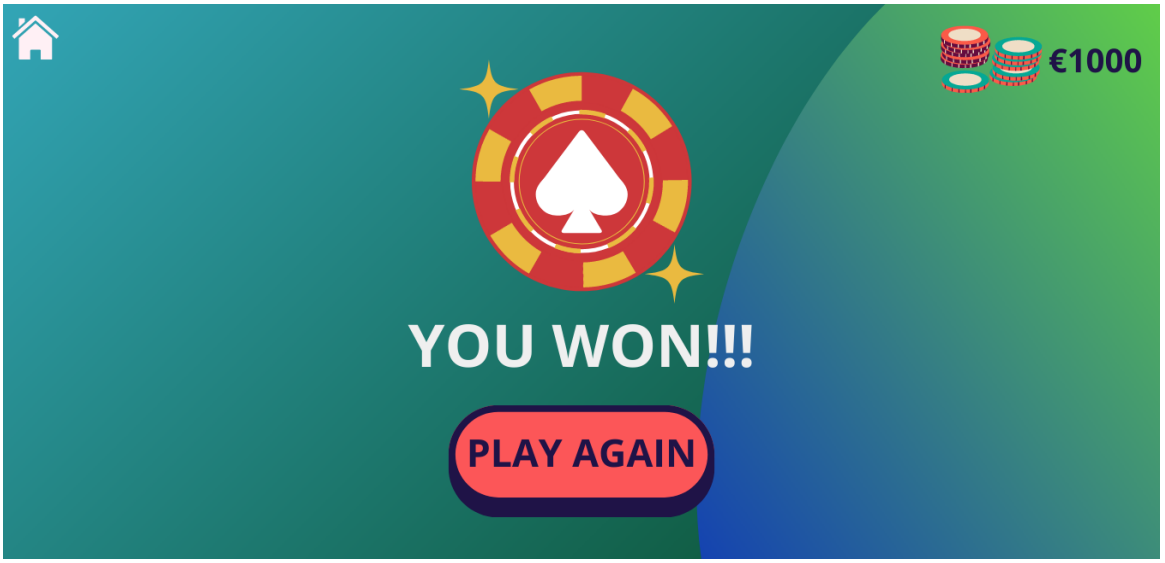
2. Object diagrams



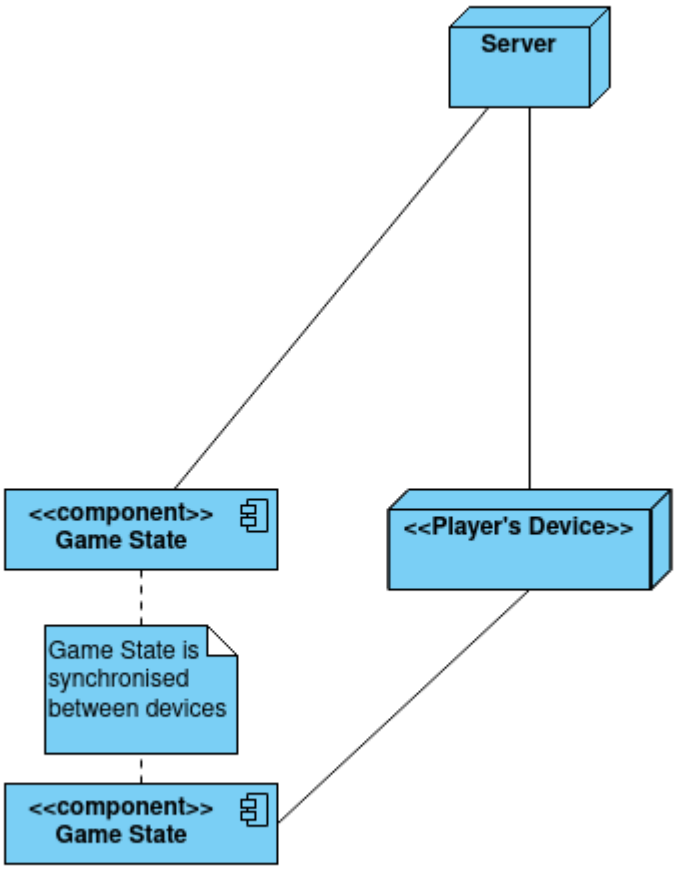
3. User interface mock-ups



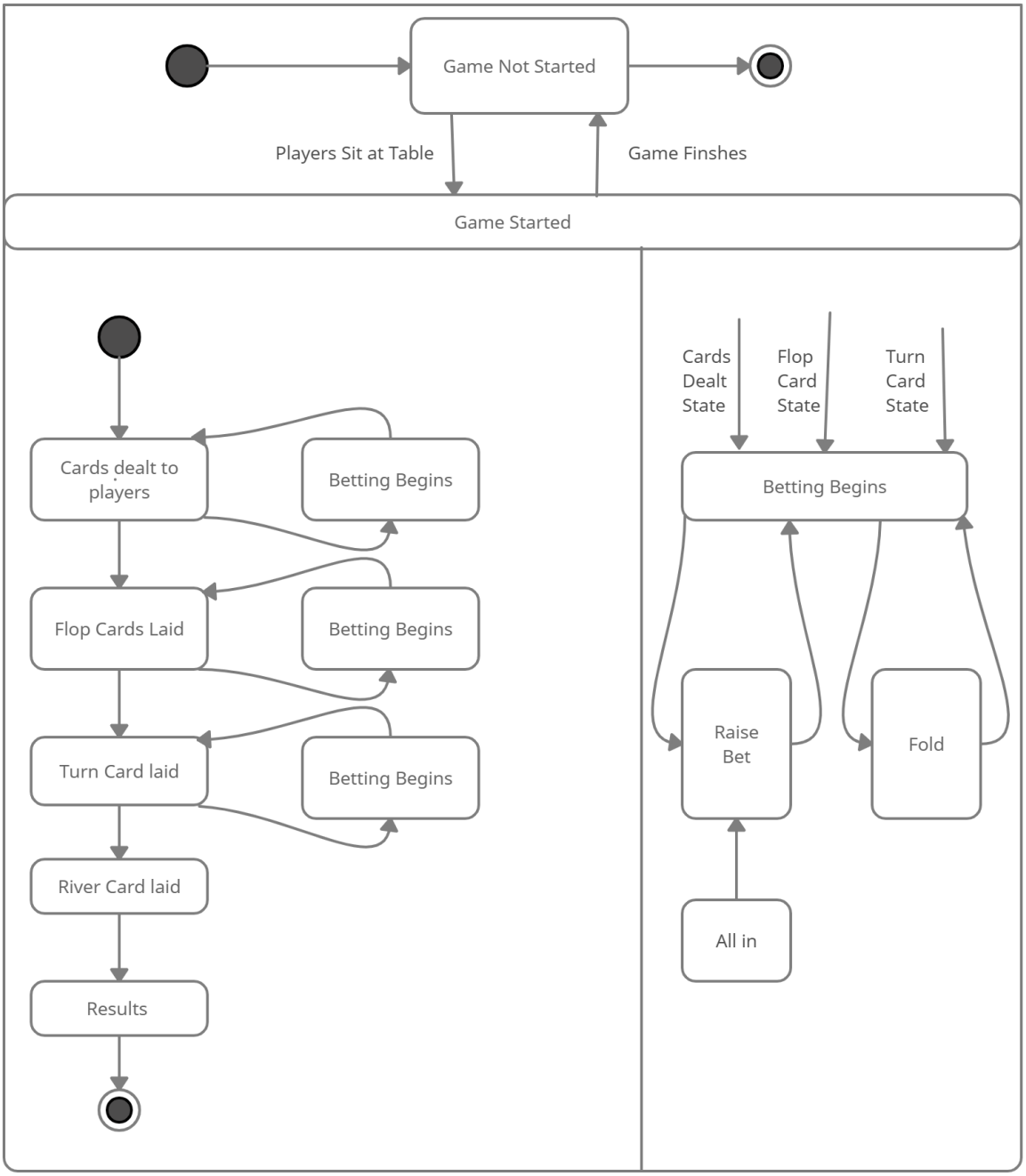

€0
NOT ENOUGH CREDITS
TRY AGAIN


€1000
YOU WON!!!
PLAY AGAIN

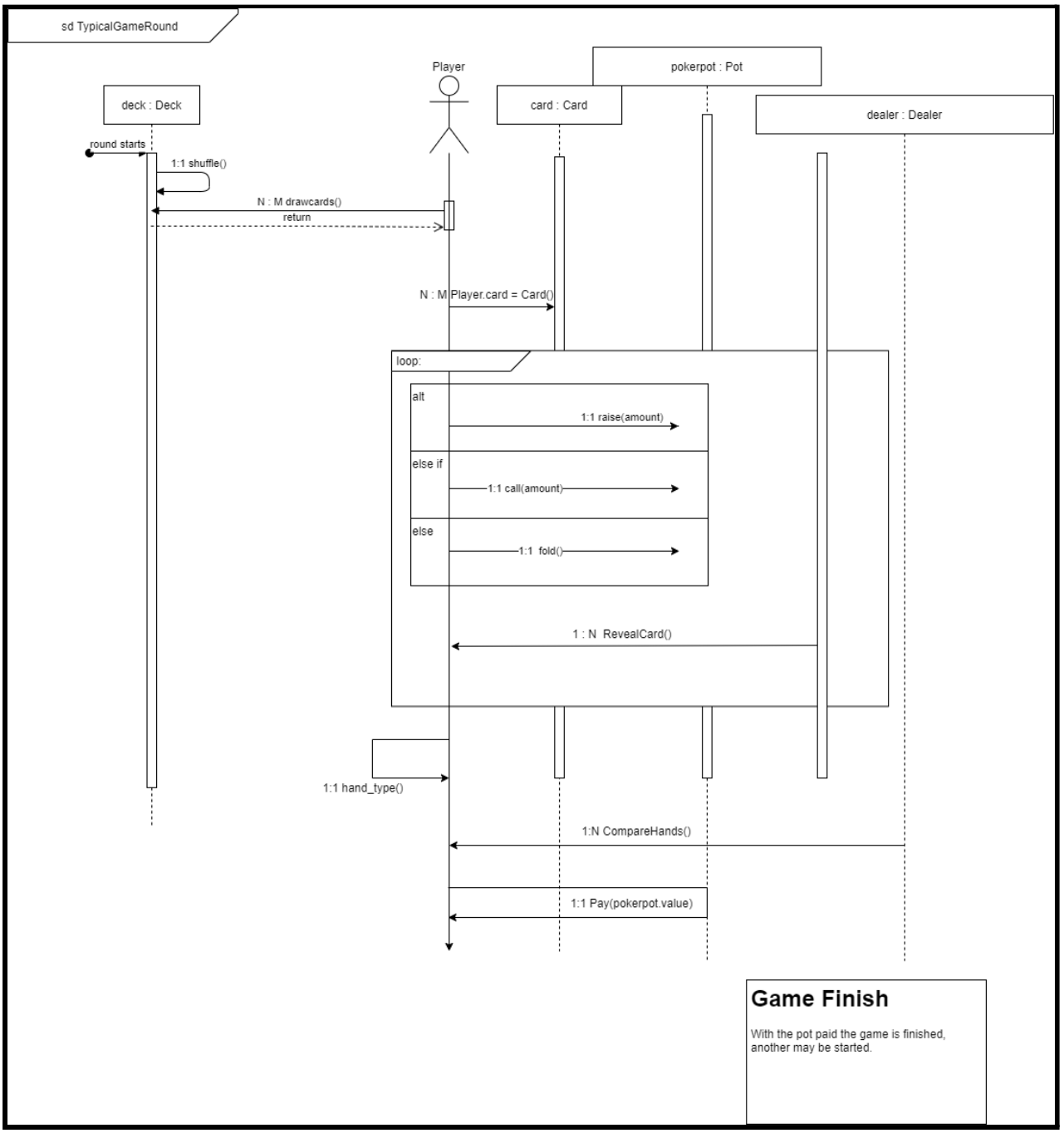
3A. Network ideas

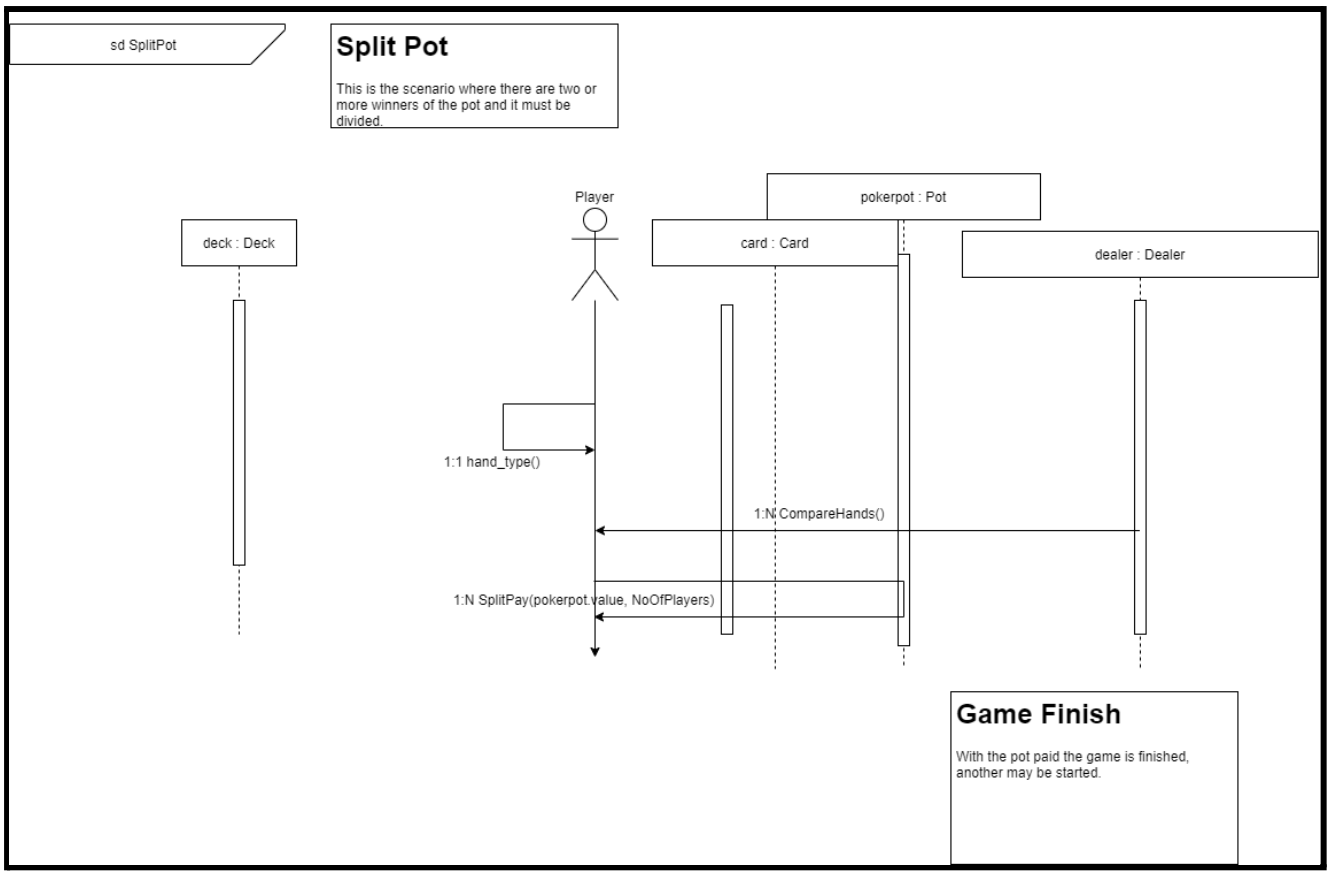


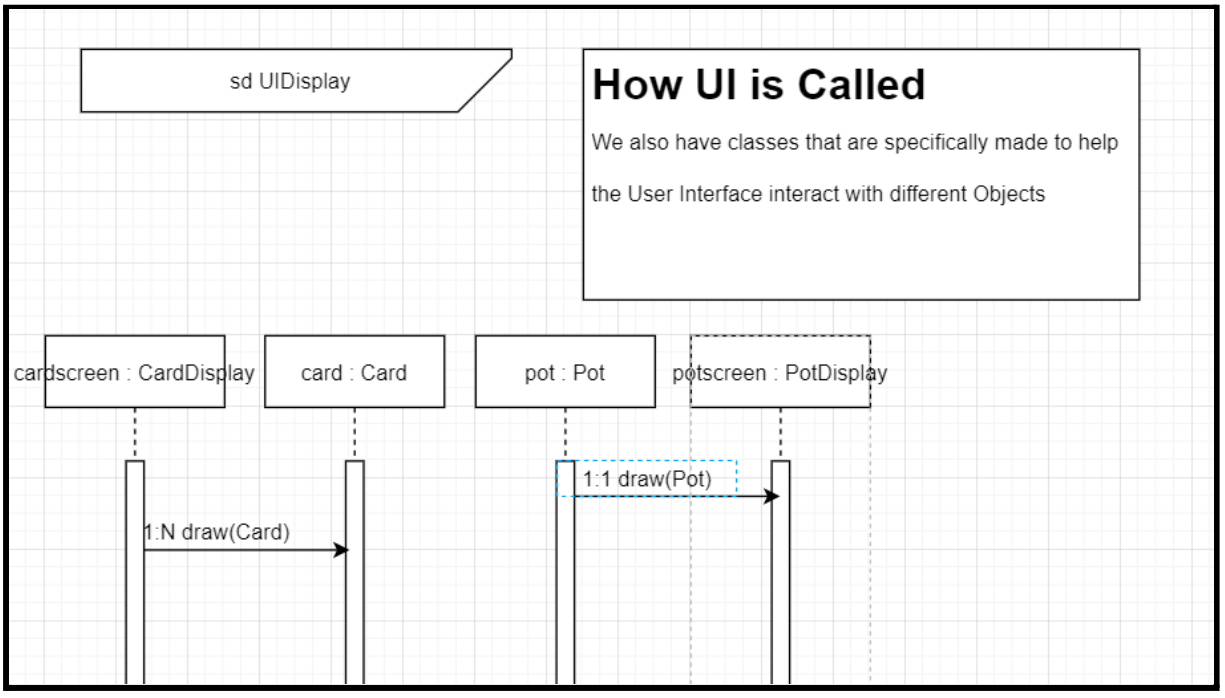
4. State machines



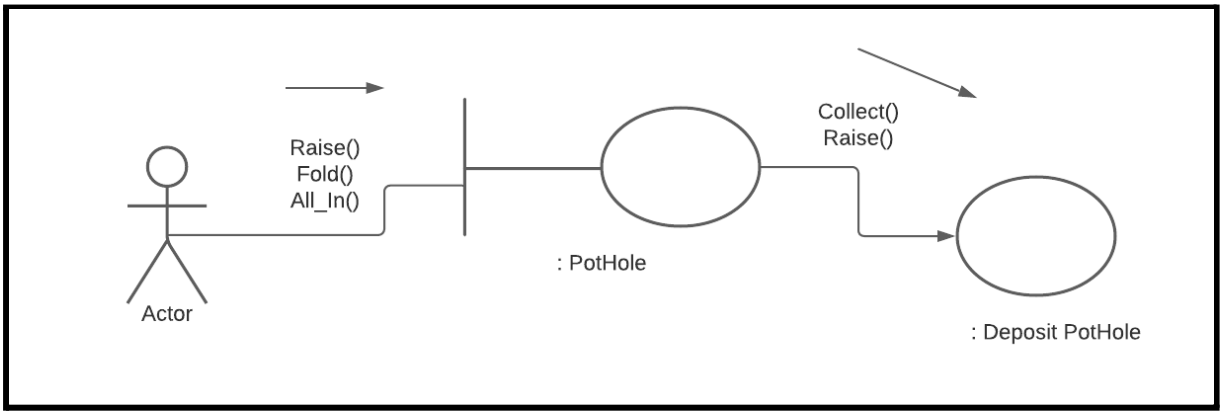
5. Sequence diagrams

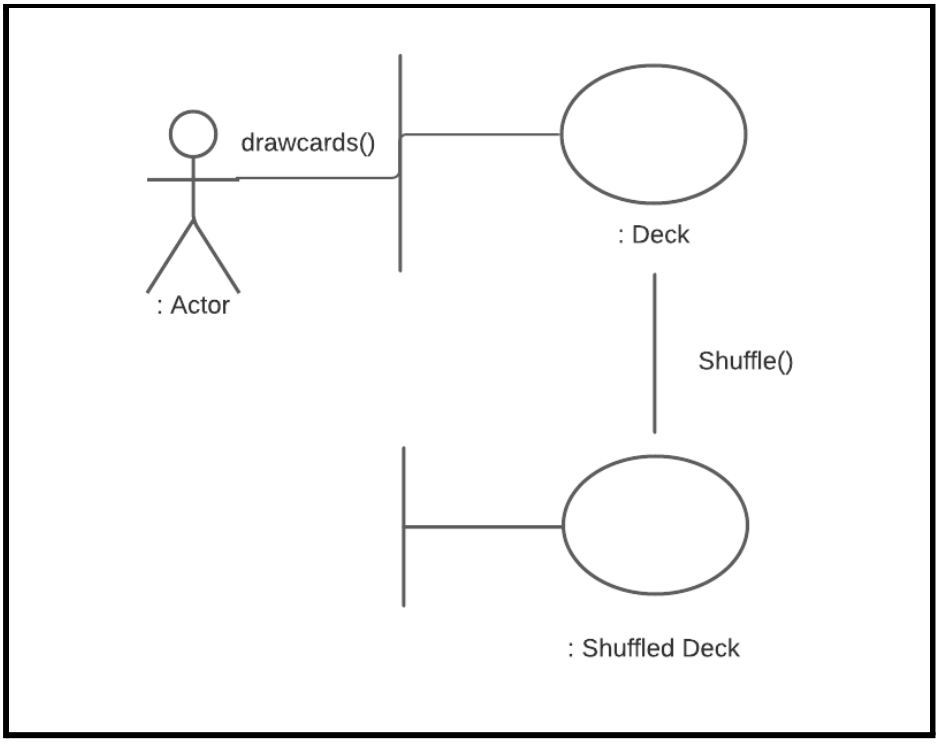


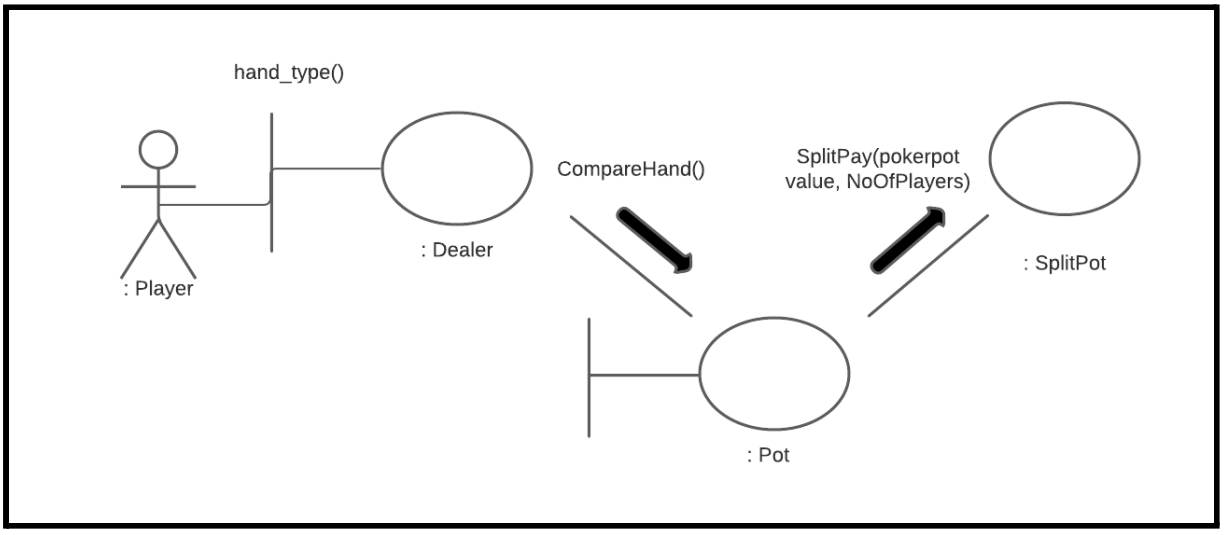

sd SplitPot
Split Pot
This is the scenario where there are two or more winners of the pot and it must be divided.
Player
pokerpot : Pot
deck : Deck
card : Card
dealer : Dealer
1:1 hand_type()
1:N CompareHands()
1:N SplitPay(pokerpot.value, NoOfPlayers)
Game Finish
With the pot paid the game is finished, another may be started.

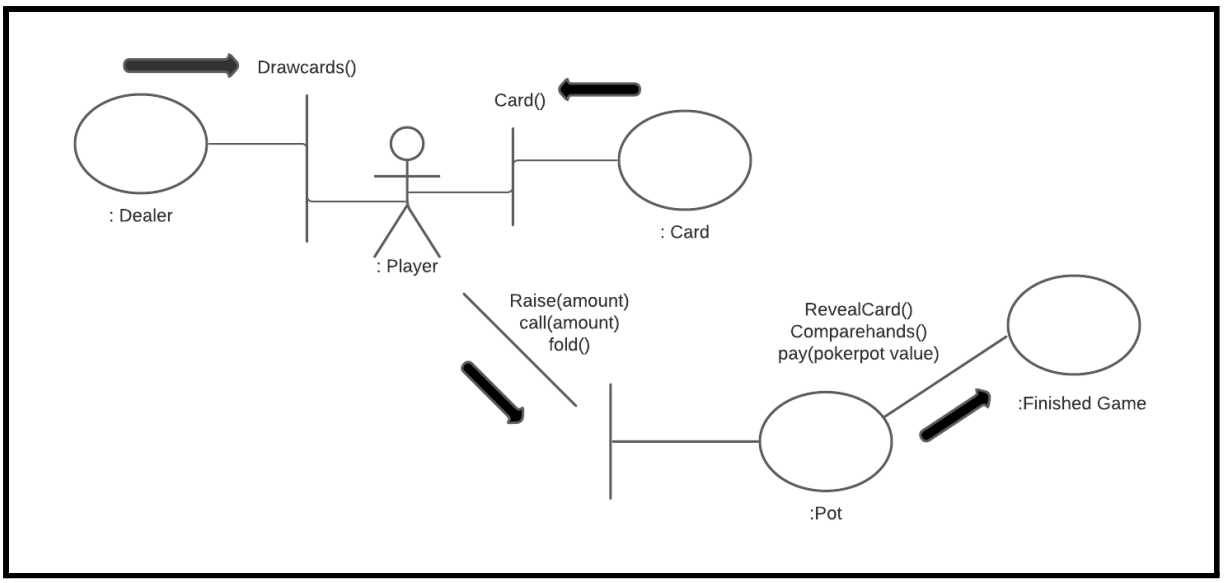

sd UIDisplay
How UI is Called
We also have classes that are specifically made to help
the User Interface interact with different Objects
cardscreen : CardDisplay
card : Card
pot : Pot
potscreen : PotDisplay
1:1 draw(Pot)
1:N draw(Card)

6. Draw 4 Collaboration (now known as communication) based completely on your sequence diagrams in Step 5

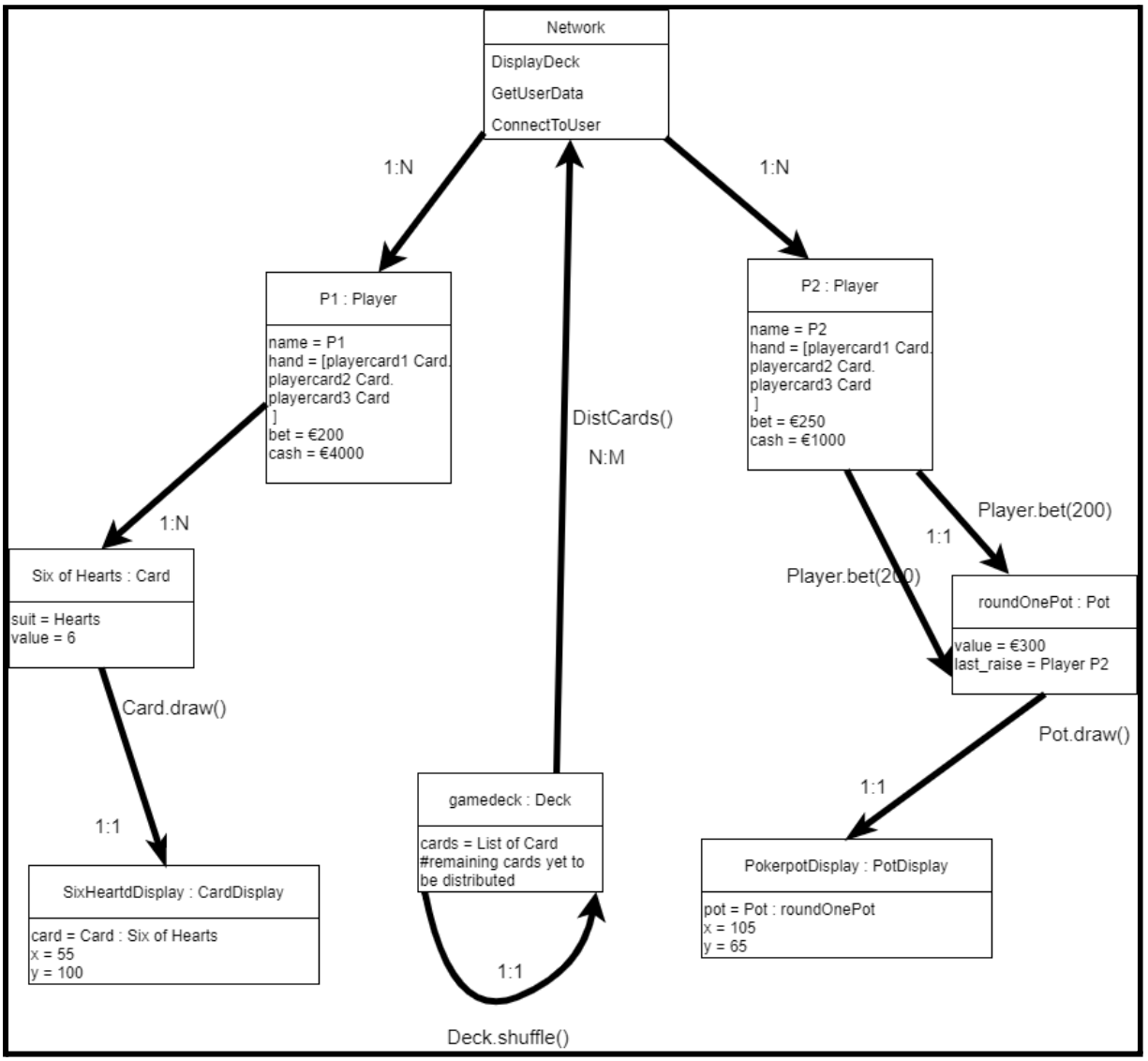








7. Revised Object diagrams



8. More Refined class diagrams

Check associations, try to make them complete with information, naming, cardinality, type, et cetera)
(notes from other members:

- renamed Pot raise -> Pot raisepot
- Add a player attribute called name.
- added Deck DistCards: a method that draws the players cards from Deck,
- give methods to Player for Raise, Call, Check or Fold,
- Add a dealer class to reveal cards
    - Also compares hands and decides the winner.
- For the “CardDisplay” class, replace value and suit with the literally Card being represented. E.g CardDisplay.card = Card(Hearts, 6)
- Added Network class to allow information to send information between players.
    - DisplayDeck deals cards to each player
    - GetUserData checks each user’s hand, turn status, and money
    - ConnectToUser initialises a new player into the game

)

9. Class skeletons from 8. (i.e. code prototypes, based on your actual implementation language, docstrings and comments can help here for some languages) - Use GitHub ?? Malachy & Ryan

```python
class Card:
    def __init__(self, suit: str, value: int):
        self.suit = suit
        if value == 1:
            self.value = 14
        else:
            self.value = value

    def __str__(self) -> str:
        Names = {
        11: "Jack", 12: "Queen", 13: "King", 14: "Ace",
        }
        return "{} of {}".format(Names[self.value], self.suit)

    def __eq__(self, other: Card) -> bool:
        return self.value == other.value

    def __gt__(self, other: Card) -> bool:
        return self.value > self.other

    def __lt__(self, other: Card) -> bool:
        return self.value > self.other
```

```python
class CardDisplay(Card):
    def __init__(self, x: int, y: int, *args, **kwargs):
        super().__init__
        self.x = x
        self.y = y

    def draw(self):
        """
        Draw card to screen
        """
```

```python
class Deck:
    def __init__(self):
        self.cards = [[Card(suit, val) for suit in ["Hearts", "Diamonds", "Spades", "Clubs"]] for val in range(1, 15)]
        self.shuffle()

    def shuffle(self):
        random.shuffle(self.cards)


    def dist_cards(self, players: List[Player], amount: int):
        for player in players:
            for _ in range(amount):
                player.draw(self.cards.pop())
```

```python
class Player:
    def __init__(self, name: str, hand: List[Card], bet: int=0, cash: int=0):
        self.name = name
        self.hand = hand
        self.bet = bet
        self.cash = cash

    def draw(self, card: Card):
        self.hand += card

    def hand_type(self) -> Tuple[int, int]:
        """
        Calculates the value of the player's hand and returns the value of the player's hand and their highest card
        """

    def pay(self, payment: int):
        self.cash += int

    def raise_pot(self, amount: int):
        self.bet += amount

    def call(self):
        """
        Call bet
        """

    def check(self):
        """
        Check bet
        """

    def fold(self):
        """
        Fold
        """
```

```python
class playerDisplay(Player):
    def __init__(self, x: int, y: int, *args, **kwargs):
        super().__init__
        self.x = x
        self.y = y


    def draw(self):
        """
        Draw player to screen
        """
```

```python
class Pot:
    def __init__(self, value: int=0):
        self.value = 0
        self.last_raise: Player = None

    def raise_pot(self, amount: int, player: Player):
        self.pot += amount
        self.last_raise = player
```

```python
class PotDisplay(Pot):
    def __init__(self, x: int, y: int, *args, **kwargs):
        super().__init__
        self.x = x
        self.y = y

    def draw(self):
        """
        Draw pot to screen
        """
```

10. Minutes or notes of team meetings. - Date, Attendance, Actions.

01/11/21 - Meetup: All in attendance

- Met and assigned titles to subgroups of people (2 or 3 people)

08/11/21 - Meetup: All in attendance

- Approaching the Assignment 2 Deadline, we met to further discuss Ass 2
- We tried to combine any common updates to the previously established models, esp in terms of synthesizing the terms and phrasing

Consistent correspondence in our group chat was kept concerning the project