Gra w makao od 2 do 4 graczy.

Piotr Sławek

2022-06-21

Contents

1 Wstęp **1**
1.1 Doprecyzowanie zasad 1
1.2 Zasady opcjonalne 2

2 Opisy klas **2**
2.1 Card 2
2.2 CardDraw 3
2.3 CardWait 4
2.4 CardDemand 5
2.5 CardExchange 6
2.6 CardNoExchange 6
2.7 Deck 6
2.8 DeckPosition 7
2.9 Discarded 7
2.10 DiscardPosition 8
2.11 Player 8
2.12 Game 9

3 Diagram UML **11**

4 Działanie programu **11**

1 Wstęp

Napisany program to gra w makao. Komunikacja z graczami odbywa się przez wiersz poleceń. Nie ma możliwości zagrać z komputerem - nie ma sztucznej inteligencji.

Zasady gdy w makao można znaleźć tutaj: https://pl.wikipedia.org/wiki/Makao_(gra_karciana).

1.1 Doprecyzowanie zasad

Występuje wiele wersji tej gry, różniących się drobnymi szczegółami, więc wypiszę tutaj pewne założenia:

1. *makao* lub *po makale* można napisać zawsze, gdy gra zadaje pytanie (faktyczne pytanie ze znakiem zapytania). Żeby nie trywializować gry, napisanie tych wyrażeń w złym momencie skutkuje dobraniem 5 kart.
2. Nie ma schodków (rosnące lub malejące ułożenie kart).
3. W dowolnej sytuacji można położyć dowolnie wiele kart tej samej rangi (z wyjątkiem króli).

- Jeśli gracz decyduje się położyć nową kartę, to technicznie prosi o dodatkową turę. Jego tura jeszcze trwa, więc jeśli dopiero w tej *następnej* pozbędzie się przedostatniej lub ostatniej karty, nie powinien pisać *makao*.

4. Obrona przed kartami funkcyjnymi jest tego samego koloru lub tej samej rangi.
5. Jedyna funkcja damy to *dama na wsyzstko, wszystko na damę*, bez obrony przed kartami funkcyjnymi.

1.2 Zasady opcjonalne

Dodałem też opcjonalne zasady, które kiedyś opracowałem ze znajomymi. Nacisk na *opcjonalne*, to znaczy przed rozpoczęciem gry, gracz dostaje wybór czy się na nie zgadza:

1. Króle trefl i karo, normalnie bezfunkcyjne, dostają nowe funkcje.
2. Król trefl może zainicjować zamianę wszystkimi kartami z dowolnie wybranym graczem.
3. Król karo blokuje zamianę. Tylko król karo.
4. 'makao' lub 'po makale' inicjującego zamianę nie ma żadnego efektu. Nie ma żadnych kar.
5. Cel zamiany musi powiedzieć co trzeba (*makao*, *po makale*) zarówno gdy blokuje zamianę, jak i gdy do zamiany dochodzi.
6. Zablokowanie zamiany powoduje, że jej inicjator dobiera 5 kart.

2 Opisy klas

2.1 Card

Klasa przechowująca informacje o każdej karcie i sposobie w jaki oddziałuje na przebieg gry. Jej parametry to: `game` (**Game**), nominał karty `rank` (liczba naturalna), kolor `color` (string), `universal` (wartość logiczna) i `color_shift`. `universal` ma wartość *true* tylko i wyłącznie dla dam, a `color_shift` - dla asów.

Wartości parametrów `rank` i `color` są przekazywane jako argumenty przy tworzeniu obiektu tej klasy.

Jej metody to:

2.1.1 get_rank(), get_color(), is_universal()

Te metody służą do zwrócenia wyżej wymienionych informacji, które przechowuje karta.

2.1.2 direct_next_turn()

Domyślny sposób ustalenia następnego gracza, który odbędzie swoją turę. Modyfikuje atrybut obiektu klasy **Game**: `current_player` tak, żeby odpowiadał następnemu graczowi na liście graczy (lub pierwszemu, jeśli była tura ostatniego).

2.1.3 card_function()

Domyślny sposób interakcji z graczem podczas jego tury.

Najpierw sprawdza, czy gracz ma dostępny legalny ruch (sprawdzana jest każda karta z listy `hand`, która jest parametrem **Player**):

1. Jeśli tak, pada pytanie, czy gracz chce zagrać kartę:
    - Jeśli tak, gracz jest proszony o wpisanie karty, którą ma na ręce (parametr `hand` obiektu **Player**) do momentu, gdy będzie to legalny ruch. Wykonuje się ponowne sprawdzenie, czy gracz ma legalny ruch i jeśli tak - pojawia się pytanie, czy chce zagrać jeszcze jedną kartę.
    - Jeśli nie, gracz dobiera kartę.

2. Jeśli nie, gracz dobiera kartę i jeśli jest legalnym ruchem, pada pytanie, czy ma zostać zagrana. Jeśli tak - patrz punkt 1, bez wpisywania nazwy karty. Jeśli nie, nic się nie dzieje.

Jeśli zagrana karta jest klasy **CardDraw**, ilość kart do dobrania jest dodana do parametru `draw_queue` obiektu **Game**. Analogicznie w przypadku **CardWait**, ale modyfikowany jest parametr `wait_queue`.

Jeśli nie została wybrana żadna karta, obiekt **Game** jest informowany, że specjalna funkcja karty na wierzchu stosu karty ma nie być wykonywana i że gracz nie powtarza tury. Odbywa się to przez ustawienie parametru `action` obiektu **Game** na *false*, a parametru `continue_turn` na *false*. Następnie utworzona zostaje 7 kier (**Card**), na której zostaje wywołana metoda `direct_next_turn`. Parametry `makao_said` i `po_makale_said` obiektu **Game** są ustawione na *false*.

Jeśli zagrana karta jest klasy **CardExchange**, wyświetla się lista imion graczy i liczb ich kart, tworzona na podstawie parametru `players` obiektu **Game** i pada pytanie, czy gracz chce wymienić się z kimś kartami:

1. Jeśli tak, gracz wpisuje imię celu zamiany, parametry `exchange_initiator` i `exchange_victim` obiektu **Game** są ustawiane na reprezentujące odpowiednich graczy, a parametry `makao_said` i `po_makale_said` obiektu **Game** są ustawione na *false*.
2. Jeśli nie, nic się nie dzieje.

Jeśli aktywny gracz nie zdecydował się wymienić kartami i została zagrana jakaś karta, następuje sprawdzenie, czy gracz powiedział *makao* lub *po_makale* poprzez wywołanie metody `check_makao` obiektu **Game**. Jeśli należy się kara, gracz dobiera 5 kart.

Jeśli gracz zdecydował się kontynuować turę po położeniu karty: `action` na *false*, `continue_turn` na *true*.

Jeśli nie ma kontynuowania tury, karta **CardExchange** została zagrana, ale nie było chęci wymiany kart, to `continue_turn` na *false*, `action` na *false*. Następnie utworzona zostaje 7 kier (**Card**), na której zostaje wywołana metoda `direct_next_turn`.

Jeśli powyższe dwa warunki nie są spełnione: `continue_turn` na *false*, `action` na *true*. Na wybranej karcie zostaje wywołana metoda `direct_next_turn`. Następnie sprawdzane są dwa warunki:

1. Jeśli parametr `color_shift` zagranej karty (**Card**) był równy *true*, gracz wybiera jeden z czterech kolorów i parametr `top_color` obiektu **Discarded** jest ustawiany na ten kolor.
2. Jeśli karta była klasy **CardDemand**, gracz wybiera jakiej karty chce żądać lub decyduje się nic nie żądać.
    - Jeśli nic nie było żądane, `action` na *false* oraz każdemu obiektowi **Player** parametr `active_demand` jest ustawiony na *None*.
    - Jeśli coś jest żądane każdemu obiektowi **Player** parametr `active_demand` jest ustawiony na to żądanie (liczba naturalna).

2.2 CardDraw

Podklasa **Card**, reprezentująca karty powodujące dobieranie kart (dwójki, trójki, król kier, król pik). Ma dodatkowe parametry `amount` (liczba naturalna) i `reverse` (wartość logiczna), które odpowiednio oznaczają ilość dobieranych kart i kierunek ataku. Różni się od **Card** metodami `direct_next_turn` i `card_function`.

2.2.1 direct_next_turn()

Jeśli nie `reverse`, wykonuje się wersja z **Card**. Jeśli `reverse` (król pik), poprzedni gracz na liście graczy jest ustalany jako ten, który ma grać turę jako następny.

2.2.2 card_function()

Następuje sprawdzenie, czy gracz ma legalny ruch (karta **CardDraw** do koloru lub tej samej rangi).

1. Jeśli tak, gracz jest informowany ile kart dobierze, jeśli się nie obroni. Może zagrać kartę lub zdecydować się nie bronić. Jeśli zagrywa kartę, wybiera legalną kartę do skutku, a następnie może kontynuować turę, analogicznie do **Card**. Jeśli gracz się nie broni, dobiera tyle kart ile wynosi wartość parametru `draw_queue` obiektu **Game**.
2. Jeśli nie ma legalnego ruchu, wyświetla się informacja, że gracz dobierze karty i ich ilość. Najpierw sprawdzana jest pierwsza dobrana karta, a `draw_queue` zmniejsza się o 1. Jeśli karta jest legalnym ruchem, mamy sytuację z punktu 1. Jeśli nie, dobierana jest reszta kart, a `draw_queue` jest wyzerowane.

Jeśli zagrana karta jest klasy **CardDraw**, ilość kart do dobrania jest dodana do parametru `draw_queue` obiektu **Game**.

Jeśli nie została wybrana żadna karta, utworzona zostaje 7 kier (**Card**), na której zostaje wywołana metoda `direct_next_turn`.

Jeśli nie została wybrana żadna karta, obiekt **Game** jest informowany, że specjalna funkcja karty na wierzchu stosu karty ma nie być wykonywana i że gracz nie powtarza tury. Odbywa się to przez ustawienie parametru `action` obiektu **Game** na *false*, a parametru `continue_turn` na *false*. Następnie utworzona zostaje 7 kier (**Card**), na której zostaje wywołana metoda `direct_next_turn`. Parametry `makao_said` i `po_makale_said` obiektu **Game** są ustawione na *false*.

Jeśli powyższy warunek nie jest spełniony:

1. Następuje sprawdzenie, czy gracz powiedział *makao* lub *po_makale* poprzez wywołanie metody `check_makao` obiektu **Game**. Jeśli należy się kara, gracz dobiera 5 kart.
2. Jeśli gracz zdecydował się kontynuować turę po położeniu karty: `action` na *false*, `continue_turn` na *true*.
3. Jeśli powyższy warunek nie jest spełniony, `action` na *true*, `continue_turn` na *false*. Na zagranej karcie zostaje wywołana metoda `direct_next_turn`.

2.3 CardWait

Podklasa **Card**, reprezentująca karty powodujące czekanie tury (czwórki). Różni się od **Card** metodą `card_function`.

2.3.1 card_function()

Następuje sprawdzenie, czy gracz ma legalny ruch (karta **CardWait**):

1. Jeśli tak, gracz jest informowany, że jeśli się nie obroni, czeka tyle tur ile wynosi wartość parametru `wait_queue` obiektu Game. Może wybrać, czy się broni:
   - Jeśli tak, wybiera do skutku kartę do zagrania. Jeśli ma więcej legalnych ruchów, może zdecydować się kontynuować turę.
   - Jeśli nie, do parametru `n_wait` obiektu **Player** dodawane jest *wait_queue - 1*, a następnie `wait_queue` jest zerowane. Gracz nie wykonuje więcej akcji.
2. Jeśli nie, gracz jest informowany ile czeka i dostaje możliwość dobrać kartę, żeby móc się obronić. Jeśli nie chce dobrać - po prostu czeka. Jeśli dobiera:
   - Jeśli się nie uda, dostaje motywujący komentarz i czeka, zatrzymując kartę.
   - Jeśli się uda, karta jest grana automatycznie, a gracz jest informowany, że się udało.
     - Jeśli to spowodowało, że jego liczba kart zmieniła się 1 -> 2 -> 1, wyświetla się dodatkowe pytanie, nie podpowiadające w jakiej jest sytuacji, ale dające możliwość powiedzenia *makao*. To pytanie jest czymś w rodzaju easter-egga i dlatego gracz nie dostaje standardowej możliwości wyboru.

Jeśli nie została zagrana karta, obiekt **Game** jest informowany, że specjalna funkcja karty na wierzchu stosu karty ma nie być wykonywana i że gracz nie powtarza tury. Odbywa się to przez ustawienie parametru `action` obiektu **Game** na *false*, a parametru `continue_turn` na *false*. Następnie utworzona zostaje 7 kier (**Card**), na której zostaje wywołana metoda `direct_next_turn`. Parametry `makao_said` i `po_makale_said` obiektu **Game** są ustawione na *false*.

Jeśli powyższy warunek nie jest spełniony:

1. `wait_queue` zwiększa się o 1.
2. Następuje sprawdzenie, czy gracz powiedział*makao* lub *po_makale* poprzez wywołanie metody `check_makao` obiektu **Game**. Jeśli należy się kara, gracz dobiera 5 kart.
3. Jeśli gracz zdecydował się kontynuować turę po położeniu karty: `action` na *false*, `continue_turn` na *true*.
4. Jeśli powyższy warunek nie jest spełniony, `action` na *true*, `continue_turn` na *false*. Na zagranej karcie zostaje wywołana metoda `direct_next_turn`.

2.4 CardDemand

Podklasa **Card**, reprezentująca karty żądające położenia kart bezfunkcyjnych (walety). Różni się od **Card** metodą `card_function`.

2.4.1 card_function()

Gracz jest informowany o swoim aktywnym żądaniu, określonym przez parametr `active_demant` obiektu **Player**, oraz o możliwości ewentualnego przebicia żądania (tak - jeśli ostatnio zagrany był walet, nie - w innym przypadku)

Następuje sprawdzenie, czy gracz posiada legalny ruch (żądana karta lub karta tego samego nominału):

1. Jeśli tak, gracz może zagrać kartę. Jeśli nie chce, dobiera kartę. Jeśli chce zagrać, wybiera do skutku kartę z ręki. Jeśli ma więcej legalnych ruchów, może zdecydować się wziąć następną turę.
2. Jeśli nie, dobiera kartę. Jeśli jest legalnym ruchem, może podjąć decyzję ją zagrać. Nie jest możliwe mieć więcej legalnych ruchów, więc to sprawdzenie się nie odbywa.

`active_demand` zostaje zmienione na *None*.

Jeśli nie została zagrana karta, obiekt **Game** jest informowany, że specjalna funkcja karty na wierzchu stosu karty ma nie być wykonywana i że gracz nie powtarza tury. Odbywa się to przez ustawienie parametru `action` obiektu **Game** na *false*, a parametru `continue_turn` na *false*. Następnie utworzona zostaje 7 kier (**Card**), na której zostaje wywołana metoda `direct_next_turn`. Parametry `makao_said` i `po_makale_said` obiektu **Game** są ustawione na *false*.

Jeśli powyższy warunek nie jest spełniony:

1. Następuje sprawdzenie, czy gracz powiedział *makao* lub *po_makale* poprzez wywołanie metody `check_makao` obiektu **Game**. Jeśli należy się kara, gracz dobiera 5 kart.
2. Jeśli gracz zdecydował się kontynuować turę po położeniu karty: `action` na *false*, `continue_turn` na *true*.
3. Jeśli powyższy warunek nie jest spełniony, `action` na *true*, `continue_turn` na *false*. Na zagranej karcie zostaje wywołana metoda `direct_next_turn`. Dodatkowo jeśli zagrana karta była klasy **CardDemand**, gracz wybiera jakiej karty chce żądać lub decyduje się nic nie żądać:
    - Jeśli nic nie było żądane, `action` na *false* oraz każdemu obiektowi **Player** parametr `active_demand` jest ustawiony na *None*.

- Jeśli coś jest żądane, każdemu obiektowi **Player** parametr `active_demand` jest ustawiony na to żądanie (liczba naturalna).

2.5 CardExchange

Podklasa **Card**, reprezentująca kartę inicjującą wymianę kart między graczami (król trefl, jeśli gracze wyrazili zgodę na tę mechanikę). Różni się od **Card** metodami `direct_next_turn` i `card_function`.

2.5.1 direct_next_turn()

Jeśli parametr `exchange_initiator` obiektu **Game** ma wartość *None*, utworzona zostaje 7 kier (**Card**), na której zostaje wywołana metoda `direct_next_turn`

Jeśli powyższy warunek nie jest spełniony, parametr `current_player` obiektu **Game** jest ustawiony na wartość parametru `exchange_victim` tego obiektu.

2.5.2 card_function()

Następuje sprawdzenie, czy gracz posiada legalny ruch (karta **CardNoExchange**):

Jeśli tak, gracz jest informowany o tym, który gracz chce zamienić się z nim kartami i że odparcie ataku będzie skutkowało dobraniem przez niego pięciu kart. Pada pytanie, czy gracz chce zagrać kartę:

1. Jeśli karta jest zagrana, inicjator wymiany dobiera 5 kart.
2. Jeśli nie ma legalnego ruchu lub karta nie została zagrana, wyświetla się informacja, że dochodzi do zamiany i obiekty **Player** rezprezentujące tych graczy wymieniają się wartościami parametru `hand`.

Zmieniane są parametry obiektu **Game**:

1. `current_player` na wartość parametru `exchange_initiator`
2. `action` na *false*.
3. `exchange_initiator` i `exchange_victim` na *None*.

2.6 CardNoExchange

Podklasa **Card**, reprezentująca kartę blokującą efekt karty **CardExchange** (król karo, jeśli gracze wyrazili zgodę na tę mechanikę). Nie różni się niczym od klasy **Card** i mogła zostać zastąpiona parametrem tej klasy, ale taką implementację uznałem za bardziej estetyczną i przejrzystą.

2.7 Deck

Obiekty tworzone z argumentem **Game**.

Klasa reprezentująca talię kart w grze jako listę łączoną. Jej parametry to `top_card` (**DeckPosition**), `bottom_card` (**DeckPosition**) i `game` (**Game**).

Istnieje tylko jeden obiekt tej klasy.

Jej metody to:

2.7.1 get_top(), get_bottom()

Metody zwracające wartości odpowiednich parametrów

2.7.2 remove_top()

Krok przejściowy w procesie dobierania kart.

1. Jeśli `top_card` nie jest *None*:
   - Na obiekcie `top_card` wywołana jest metoda `get_next`, a zwrócony obiekt jest ustawiony na wartość `top_card`
   - Metoda zwraca starą wartość `top_card` oraz *false* jako informację, że talia i stos się nie skończyły i są jeszcze karty do dobierania
2. Jeśli `top_card` jest *None*:
   - Jeśli karta (**DiscardPosition**) na szczycie stosu (**Discarded**) ma następnika (wywołanie `get_next` na tej karcie nie zwraca *None*), to na stosie (**Discarded**) wywołana jest metoda `restock_deck`. Następnie wywołana jest rekurencyjnie metoda `remove_top`.
   - Jeśli powyższy warunek nie jest spełniony, zwracane są *None* i *true*. *true* jest informacją, że nie da się dobierać kart.

2.7.3 add_card(card: Card)

Metoda używana w parze z metodą `restock_deck` obiektu **Discarded**.

card służy do utworzenia nowego obiektu **DeckPosition**.

Jeśli `top_card` jest *None*, `top_card` i `bottom_card` zostają ustawione na ten nowy obiekt

Jeśli nie jest, `bottom_card` jest ustawione na nowy obiekt, stare `bottom_card` ustawia go jako następnika metodą `set_next` i jest ustawione jako poprzednik nowego obiektu metodą `set_previous`

2.8 DeckPosition

Obiekty tworzone z argumentem **Card**.

Klasa reprezentująca elementy listy łączonej **Deck**, reprezentujące karty w talii.

Parametry to `card` (**Card**), `previous_card` (**DeckPosition**) i `next_card` (**DeckPosition**).

Jej metody to:

2.8.1 get_card(), get_previous(), get_next()

Metody zwracające wartości odpowiednich atrybutów.

2.8.2 set_previous(card: DeckPosition), set_next(card: DeckPosition)

Metody przypisujące nowe wartości odpowiednim atrybutom.

2.9 Discarded

Obiekty tworzone z argumentem **Game**.

Klasa reprezentująca stos kart zagranych w grze. Jej parametry to `top_card` (**DiscardPosition**), `top_color` (string) i `game` (**Game**).

Istnieje tylko jeden obiekt tej klasy.

Jej metody to:

2.9.1 get_top(), get_color()

Metody zwracające wartości odpowiednich atrybutów.

2.9.2 set_top(card: DiscardPosition)

Przypisuje atrybutowi `top_card` argument. Kolor karty przechowywanej przez ten argument jest ustawiony jako wartość `top_color`.

2.9.3 place_card (card: Card)

Tworzony jest obiekt typu **DiscardPosition** z argumentem *card*.

`top_card` jest ustawione jako następnik nowego obiekt metodą `set_next`.

Wywołana jest metoda `set_top` z nowym obiektem jako argumentem.

2.9.4 restock_deck()

Wszystkie elementy stosu **Discarded** oprócz pierwszego zostają ułożone w listę. Lista jest wymieszana losowo z użyciem funkcji `shuffle` biblioteki `random`. Każdy obiekt wymieszanej listy jest dodany do talii **Deck** metodą `add_card`.

Następnik pierwszego elementu zostaje ustawiony na *None*.

2.10 DiscardPosition

Obiekty tworzone z argumentem **Card**.

Klasa reprezentująca elementy stosu **Discarded**, czyli zagrane w grze karty. Parametry to `card` (**Card**) i `next_card` (**DiscardPosition**).

Jej metody to:

2.10.1 get_card(), get_next()

Metody zwracające wartości odpowiednich atrybutów.

2.10.2 set_next(card: DiscardPosition)

Ustala wartość parametru `next_card` na *card*

2.11 Player

Obiekty tworzone z agumentami **Game** i string (imię gracza).

Klasa reprezentująca graczy.

Jej atrybuty to:

- `game` (**Game**)
- imię gracza `player_name` (string)
- liczba tur do przeczekania - `n_wait` (integer)
- ewentualne aktywne żądanie przez waleta - `active_demand` (integer)
- karty na ręce - `hand` (lista obiektów **Card**)

Jej metody to:

2.11.1 play_card(card: Card)

Z listy `hand` usuwany jest element *card*, a następnie jest kładziony na stosie **Discarded** metodą `place_card`.

2.11.2 draw_card()

Metoda wywoływana za każdym razem, gdy dowolny gracz dobiera kartę

Na talii **Deck** wywoływana jest metoda `remove_top`, która zwraca nową kartę (**DeckPosition**) i wartość logiczną *all_empty* mówiącą, czy dobranie karty jest możliwe (czy talia i stos się nie wyczerpały)

Jeśli karty się wyczerpały, wyświetla się informacja, że dobranie karty jest niemożliwe.

Jeśli karty się nie wyczerpały, z nowej karty zstaje odczytany atrybut `card`, a jego wartość (**Card**) zostaje dodana do listy `hand`. Karty są sortowane w trakcie dodawania najpierw po kolorach, potem po wielkości.

Zwracana jest dobrana karta (**Card**) i *all_empty*.

2.12 Game

Obiekty tworzone bez argumentów.

Przy utworzeniu:

1. Wyświetla instrukcję gry przed jej rozpoczęciem i pyta gracza, czy zgadza się na nową funkcję króli trefl i karo, po czym odpowiednio ustala wartość parametru `with_a_twist`
2. Tworzone są grywalne karty, czyli obiekty **Card** (i wszystkich podklas). Karty ułożone są w listę, która jest wymieszana z wykorzystaniem funkcji `shuffle` biblioteki `random`.
3. Utworzone są: talia (**Deck**) i stos kart zagranych (**Discarded**). Jedna z kart z utworzonej listy jest umieszczna na wierzchu stosu metodą `place card`. Pozostałe są dodane do talii metodą `add_card`.

Klasa ta reprezentuje obiekt rozpoczynający i kończący grę oraz przechowujący wszystkie potrzebne parametry i metody, które nie znajdują się w innych klasach.

Istnieje tylko jeden obiekt tej klasy.

Jej atrybuty to:

- `players` (lista obiektów **Player**)
- liczba graczy `n_players` (integer)
- numer aktywnego gracza - `current_player` (integer)
- liczba kart do dobrania przy następnym efekcie karty **CardDraw** - `draw_queue` (integer)
- liczba tur do przeczekania przy następnym efekcie karty **CardWait** - `wait_queue` (integer)
- parametr dyktujący, czy ma zostać wykonana funkcja ostatnio zagranej karty - `action` (bool)
- parametr dyktujący, czy aktywny gracz kładzie następną kartę - `continue_turn` (bool)
- informacja, czy gracz powiedział *makao* - `makao_said` (bool)
- informacja, czy gracz powiedział *po makale* - `po_makale_said` (bool)
- informacja, czy gra ma się odbyć z nową funkcją króli - `with_a_twist` (bool)
- numer gracza inicjującego wymianę kart królem trefl - `exchange_initiator` (integer)
- numer gracza wybranego do wymiany kart - `exchange_victim` (integer)
- informacja, czy któryś z graczy odniósł zwycięstwo - `over` (bool, domyślnie *false*)

Jej metody to:

2.12.1 set_action(bool)

Ustawia wartość parametru `action`.

2.12.2 get_n_players()

Zwraca wartość parametru `n_players`.

2.12.3 add_player(player_name: string)

Tworzy nowego gracza (**Player**) z parametrem `player_name` i dodaje go do listy `players`. Następnie zwiększa `n_players` o 1.

2.12.4 set_players()

Prowadzi interakcję z graczem przed rozpoczęciem gry, żeby ustalić listę graczy.

Gracz wpisuje imiona dopóki nie zasygnalizuje, że skończył lub do momentu, gdy jest już czterech graczy.

Wpisanie imienia wywołuje metodę `add_player` z imieniem jako argumentem, jeśli nie ma jeszcze gracza o tym imieniu.

Gracz nie może powiedzieć, że skończył, jeśli jest mniej niż dwóch graczy.

Na koniec gracz może zdecydować, że nie jest zadowolony z imion, które wpisał i zacząć od nowa, co resetuje wartości parametrów `players` i `n_players` oraz wywołuje metodę `set_players`

2.12.5 start_game()

Każdy gracz (**Player**) z listy `players` dobiera 5 kart metodą `draw_card`.

2.12.6 start_turn()

Jeśli aktywny gracz nie kładzie kolejnej karty:

- Wyświetla się informacja, czyja jest tura.
- Wyświetla się informacja, jaka karta leży na wierzchu stosu. Jeśli jej (**Card**) parametr `color_shift` jest równy *true*, dodane jest na jaki kolor zdecydował się poprzedni gracz, kładąc asa.
- Wyświetlają się karty na ręce w kolejności najpierw kolorami, potem numerami.

Jeśli parametr `n_wait` aktywnego gracza (**Player**) jest większy od zera, wyświetla się liczba tur do przeczekania, `n_wait` jest zmniejszone o 1, parametr `active_demand` jest ustawiony na *None*, a na karcie z wierzchu stosu (**Card**) jest wywołana metoda `direct_next_turn`.

Jeśli powyższy warunek nie jest spełniony i gracz ma aktywne żądanie przez waleta (parametr `active_demand`), utworzony zostaje walet kier (**CardDemand**) na którym zostaje wykonana metoda `card_function`.

Jeśli powyższe dwa warunki nie są spełnione i parametr `action` ma wartość *true*, na karcie ze szczytu stosu (**Card**) zostaje wywołana metoda `card_function`.

Jeśli powyższe trzy warunki nie są spełnione, utworzona zostaje 7 kier (**Card**), na której zostaje wykonana metoda `card_function`.

2.12.7 check_makao()

Sprawdza, czy aktywnemu graczowi należy się przypomnienie w jaką grę gra - w postaci pięciu kart.

Karty się należą jeśli (każda z opcji wyświetla inną informację):

- gracz ma jedną kartę i nie powiedział *makao*,
- gracz nie ma kart i nie powiedział *po makale*,
- gracz pomylił formułki,
- gracz powiedział jedną z formułek, gdy nie powinien był mówić żadnej.

Następnie parametry `makao_said` i `po_makale_said` są ustawione na *false*.

Zwracana zostaje wartość logiczna dyktująca, czy gracz dobiera karne karty, czy nie.

2.12.8 check_win()

Sprawdza, czy któryś z graczy nie ma kart. Jeśli tak, zwraca zwycięzcę (**Player**).

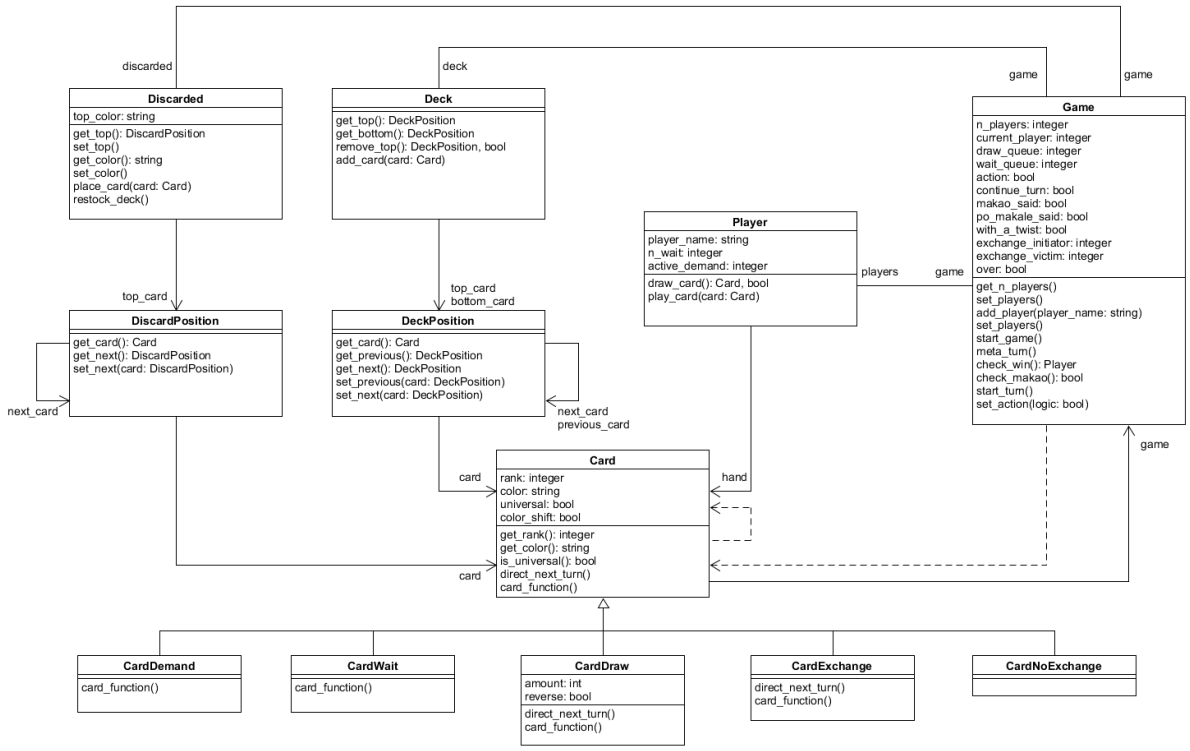
2.12.9 meta_turn()

Sprawdza, czy gra się skończyła, wywołując metodę `check_win`.

Jeśli dostaje gracza, wyświetla stosowną informację i ustawia parametr `over` na *true*.

Jeśli nie, wywołuje metodę `start_turn`.

3 Diagram UML



4 Działanie programu

Program zaczyna od utworzenia obiektu **Game** i wywołania na nim kolejno metod `set_players` i `start_game`.

Następnie do momentu gdy wartość parametru `over` będzie równa *true*, wywołuje na obiekcie **Game** metodę `meta_turn`.