Macau Game Assistant

System monitorowania przestrzegania zasad podczas gry w Makao

Opis aplikacji webowej realizowanej w ramach przedmiotu Podstawy Teleinformatyki

Arkadiusz Dokowicz

Łukasz Świdziniewski

Maciej Anglart

Maciej Drzewiecki

1. Spis treści

2. Wstęp

Celem niniejszego dokumentu jest przedstawienie założeń projektu realizowanego w ramach zajęć projektowych Podstawy Teleinformatyki. Projekt zakłada stworzenie aplikacji webowej pomagającej w kontroli przestrzegania zasad podczas rozgrywki w grze karcianej “Makao”. W celu zapewnienia pełnej funkcjonalności aplikacja dzieli się na trzy moduły - moduł wykrywania kart, moduł kontroli zasad oraz moduł komunikacji z użytkownikiem. Projekt wraz z niniejszą dokumentacją ma stanowić formę zaliczenia przedmiotu. W skład zespołu projektowego realizującego system o nazwie Macao Game Assistant, wchodzą Arkadiusz Dokowicz, Maciej Anglart, Łukasz Świdziniewski oraz Maciej Drzewiecki.

Powodem podjęcia się tego tematu, jest wyjątkowa sympatia oraz znajomość gier karcianych przez członków zespołu. Ponadto, temat pozwala na rozwijanie umiejętności z dziedziny przetwarzania obrazu, która jest niewątpliwie ciekawym i współczesnym zagadnieniem. Zadanie pozwoliło również na kształcenie umiejętności programowania w języku Python, który również wydaje się być prężnie rozwijającą technologią.

Podczas rozgrywki w “Makao”, można napotkać wiele problemów, takich jak spory dotyczące prawidłowej interpretacji zasad przez graczy. Kombinacje symboli, kolorów oraz kart specjalnych zwanych “kartami bojowymi”, wraz z szybkim tempem rozgrywki potrafią przysporzyć kłopotów. Z pomocą przychodzą kamery i algorytmy przetwarzania obrazu oraz wykrywania poszczególnych jego cech. Z ich pomocą można skutecznie określić kolejno położone karty oraz zweryfikować zgodność zagrań graczy co do zasad.

Systemy takie jak ten, działają w oparciu o szereg działań związanych z wstępnym przetwarzaniem analizowanego obrazu. Obraz najczęściej reprezentowany jest jako macierz wartości pikseli o wymiarach jego rozdzielczości. Każdy piksel może zawierać informacje o wartości swojej jasności (w przypadku obrazu czarno-białego) lub poszczególnych wartościach składowych, zależnych od modelu barw (wartości kolorów czerwonego, zielonego oraz niebieskiego w przypadku modelu RGB).

Obraz taki jest często lekko rozmazywany, progowany (aby oddzielić zawartość od tła) oraz poddawany operacji zamknięcia (w celu poprawienia wyniku progowania)

3. Opis aplikacji

Macao Game Assistant to oprogramowanie mające na celu usprawnienie przebiegu gry w karcianą grę “Makao” poprzez weryfikację oraz informowanie o poprawności zagrań podczas prowadzenia rozgrywki. Rozgrywka śledzona jest przez kamerę wiszącą nad stosem kart. Kolejno położone karty są rozpoznawane, sprawdzane pod kątem poprawności zasad oraz wysyłane do aplikacji frontendowej wyświetlającej przyjazny dla użytkownika interfejs wraz z komunikatami dotyczącymi rozgrywki. Dzięki temu gracze mają stały wgląd do historii zagranych kart oraz przejrzystych informacji dotyczących tego, czy zasady gry nie zostały złamane.

Aplikacja daje do dyspozycji użytkownikowi wygodny, przejrzysty interfejs, w którym wydzielić można dwie główne sekcje - sekcja, w której wyświetlana jest graficzna reprezentacja ostatnio położonych kart oraz sekcja wyglądem przypominająca terminal systemowy, w którym wyświetlane są poszczególne komunikaty poprzedzone rysunkiem czerwonej (w przypadku złamania zasad), bądź zielonej (w przypadku prawidłowego zagrania) diody. Aby zachować przejrzystość serwisu, użytkownik ma dostęp do jedynie dwóch przycisków - rozpoczęcia oraz zakończenia gry.

4. Przeznaczenie

Aplikacja przeznaczona jest dla każdego, kto chciałby zacząć grać w Makao, bądź potrzebuje przypomnieć sobie zasady. Tak naprawdę system użyteczny jest zarówno dla doświadczonych graczy, jak i nowicjuszy. Weryfikacja przestrzegania zasad przez uczestników, jest przydatna przy każdym stopniu rozgrywki, nie ma więc większej różnicy czy gra ma charakter towarzyski, czy też poważny, turniejowy. System do działania potrzebuje sewera backendowego, zdolnego do uruchomienia skryptu w języku programowania Python oraz aplikacji backendowej (wymagającej pakietu Java Development Kit w wersji 8+) i serwera frontendowego z

zainstalowanym środowiskiem Node.js. Taki serwer można zaimplementować w ramach wielu, lub jednej maszyny (tak zwany localhost).

5. Plan prac

Poniższa tabela zawiera opis wszystkich prac związanych z implementacją systemu asystenta gry w Makao. Tabela opisuje poszczególne zadania oraz określa osoby odpowiedzialne za ich realizację.

Zadanie	**Opis**	**Osoba/y odpowiedzialna za realizację**
Stworzenie modułu wykrywającego karty	Kluczowym elementem całego projektu jest element wykrywania kart	Maciej Anglart, Maciej Drzewiecki
Walidacja wykrywania kart po stronie modułu rozpoznającego karty	Wymagane było walidowanie, oraz pomijanie błędnie wykrytych kart	Maciej Anglart Arkadiusz Dokowicz
stworzenie komunikacji z API	Stworzenie komunikacji z modułem analizy poprawności rozgrywki w module rozpoznawania kart	Arkadiusz Dokowicz Maciej Anglart
Stworzenie modułu walidacji gry w makao	Stworzenie API komunikującego się z modułem rozpoznawania i modułem komunikacji z użytkownikiem	Arkadiusz Dokowicz Łukasz Świdziniewski
Stworzenie skryptu testującego API	Wymagane było stworzenie skryptu do symulowania rozgrywki, żeby przyspieszyć testowanie modułu poprawności rozgrywki	Maciej Anglart Arkadiusz Dokowicz
Stworzenie modułu komunikacji z użytkownikiem	Aplikacja webowa relacjonująca stan gry i historię ruchów	Łukasz Świdziniewski Maciej Drzewiecki

Prace nie miały jasno określonych ram czasowych, pierwszym i najważniejszym zadaniem było opracowanie modułu rozpoznawania kart. Istnienie pierwszej wersji tego modułu pozwoliło na łatwiejsze opracowanie modułu sprawdzania poprawności przebiegu rozgrywki oraz modułu komunikacji z użytkownikiem. Pracą zarządzano w oparciu o metodologie programowania zwinnego (*ang. agile software development*) oraz scrum.

6. Wykorzystane technologie

System został zaprojektowany z myślą o łatwości rozszerzenia jego funkcjonalności i zachowania czytelności kodu źródłowego, z tego powodu zastosowane zostały wiodące technologie wykorzystywanie w zakresie przetwarzania obrazów oraz tworzenia aplikacji internetowych. Uwzględniając podział systemu na moduły możemy wyróżnić:

- Moduł rozpoznawania kart (skrypt Python):
    - Python 3.7.5 - język programowania o rozbudowanym pakiecie bibliotek standardowych. Python jest językiem z typami dynamicznymi. Wszystkie wartości przekazywane są przez referencję.
    - opencv-python 4.3.0.36 - biblioteka funkcji wykorzystywanych podczas obróbki obrazu. Służy do przetwarzania obrazu w czasie rzeczywistym.
- Moduł analizy poprawności rozgrywki (Aplikacja Java):
    - BouncyCastle - biblioteka zawierająca mechanizmy odpowiadające za bezpieczeństwo danych, wymagana była, obliczać wartość funkcji skrótu dla danej karty w rozgrywce
    - Spring Boot Framework - podstawowe narzędzia frameworku spring pozwalające na zbudowanie rest-api
- Moduł komunikacji z użytkownikiem (aplikacja Angular):
    - Angular 9 - framework i platforma do tworzenia aplikacji webowych SPA (Single Page Application)

- TypeScript 3.8 - język programowania będący nadzbiorem języka JavaScript
- SCSS - preprocesor języka kaskadowych arkuszy stylów CSS, wykorzystany do implementacji interfejsu graficznego

7. Architektura

Współdziałanie modułów opiera się na REST (Representational state transfer), będącym stylem architektury oprogramowania, definiującym zestaw zasad i procedur, które mają być używane do tworzenia usług internetowych. Opiera się on na bezstanowej komunikacji (oznacza to, że każde zapytanie od klienta musi zawierać komplet informacji oraz, że serwer nie przechowuje stanu sesji użytkownika po swojej stronie). Komunikacja klient-serwer odbywa się z wykorzystaniem metod protokołu HTTP (GET do pobierania zasobów i POST do wysyłania danych na serwer).

8. Funkcjonalności

System zapewnia użytkownikowi dostęp do następujących funkcjonalności:

- Pobieranie obrazu z kamery telefonu w czasie rzeczywistym (aplikacja DroidCam dla systemów Android)
- Rozpoznawanie kart wraz z ich kolorem i dokładnym symbolem
- Weryfikacja poprawności kolejnych zagrań
- Przechowywanie historii rozgrywki w trakcie jej trwania z podziałem na ruchy dozwolone i niedozwolone i sygnalizowaniem ich
- Prezentowanie na żywo stanu gry w formie graficznej (ostatnia i przedostatnia poprawnie położona karta)

9. Błędy i trudności w implementacji

Trudnością w implementacji był fakt, że projekt powstawał w czasie pandemii COVID-19, i nasze moduły musiałby być replikowane na każdym z komputerów, i projektowane z czasowym wyprzedzeniem, które pozwoliło odpowiednio dostosować punkty łączące nasze 3 aplikacje.

10. Instrukcja użytkowania aplikacji

- Moduł rozpoznawania kart (skrypt Python)

Aby uruchomić skrypt rozpoznający karty należy mieć zainstalowane:

- Python w wersji 3.x
- PyCharm

Pierwszym krokiem jest sklonowanie projektu z GitHuba, który znajduje się pod linkiem: https://github.com/Dornfelder07/CameraOperator. Można to zrobić przy pomocy polecenia git clone.

Po uruchomieniu IDE PyCharm, należy odczekać do dwóch minut do momentu aż w prawym dolnym rogu wyświetli się komunikat o brakujących bibliotekach. Gdy to nastąpi należy nacisnąć na auto-import, aby pobrać brakujące biblioteki.

Dodatkowo, ze sklepu Play na urządzeniach z Android OS należy pobrać program DroidCam i połączyć się z tą samą siecią, w której jest urządzenie, na którym uruchamiamy program. W klasie CardDetector.py należy podać adres i port. Przykładowo:

```
videostream =
VideoStream.VideoStream((IM_WIDTH,IM_HEIGHT),FRAME_RATE,2,'http://192.
168.0.56:4747/video?640x480').start()
```

- Moduł analizy poprawności rozgrywki (backend)

  Do poprawnego działania aplikacji w wersji deweloperskiej potrzebne będzie:

  IntelliJ IDEA 2019.2.3 ULTIMATE lub wersja wyższa

  Java 1.8.0_231

  Maven 3

  GitBash 2.23.0 lub wyższy

Należy pobrać projekt z repozytorium :
https://github.com/ArkadiuszDokowicz/PlayingCardsRulesChecker.git
Należy wpisać w konsolę git bash polecenie

git clone https://github.com/ArkadiuszDokowicz/PlayingCardsRulesChecker.git
Pozwoli to sklonować projekt, należy go uruchomić IDE IntelliJ w wersji ultimate,

IDE rozpozna, że jest to projekt wspierany frameworkiem maven, i po prawej stronie ekranu pojawi się informacja, że IDE może pobrać zależności projektu, można zrobić to ręcznie wpisując w ścieżce projektu w CMD **mvn dependency:resolve**, jednak do tego potrzebny jest maven, dodany jako zmienna środowiskowa, IDE IntelliJ pozwala ominąć ten problem, ponieważ wersja ULTIMATE posiada wbudowany maven, z graficznym interfejsem jego obsługiwania, po pobraniu zależności wystarczy uruchomić projekt w prawym górnym rogu przyciskiem run.
Aplikacja działa domyślnie na porcie 8080.

- Moduł komunikacji z użytkownikiem (frontend)
  Należy sklonować projekt z repozytorium:
  git clone https://github.com/swidziniak/macau-frontend.git
  Do uruchomienia aplikacji frontendowej należy najpierw zainstalować **Node.js** https://nodejs.org/en/
  Następnie pobrać projekt i w wierszu poleceń - po wejściu do katalogu z projektem - należy wpisać następującą komendę:
  npm install, aby zainstalować pozostałe wymagane do działania aplikacji zależności i biblioteki
  Aby uruchomić projekt należy posłużyć się komendą ***ng serve*** (ewentualnie ***npm start***), która spowoduje kompilację projektu i uruchomi serwer pod domyślnym adresem http://localhost:4200/, jeżeli chcemy uruchomić go na innym porcie należy skorzystać z flagi ***--port***, np. ***ng serve --port 4201***.

Po wejściu w przeglądarce pod podany wcześniej adres w oknie pojawi się nasza aplikacja, wystarczy kliknąć przycisk “Start Game”, aby rozpocząć relacjonowanie analizowanej rozgrywki. Jeżeli chcemy zresetować stan gry, wystarczy, że klikniemy “End Game”, a następnie ponownie ją rozpoczniemy.

11. Możliwa rozbudowa aplikacji.

Działanie modułu rozpoznawania kart można by było skorygować o dodatkową obróbkę obrazu (określanie poziomu progowania na podstawie wartości tła, filtr gaussa dla redukcji zniekształceń takich jak rysy czy sęki na drewnianym stole). Ponadto odpowiednio nauczona sieć neuronowa pozwala na uzyskanie większej skuteczności i elastyczności (zmiany w oświetleniu, brak stabilizacji obrazu) w wykrywaniu kart. Ciekawe efekty mogłoby przynieść rozpoznawanie kart za pomocą ORB (Oriented FAST and Rotated BRIEF, ponieważ detektor opiera się na algorytmie FAST oraz zmodyfikowanej wersji deskryptora wizualnego BRIEF), czyli nowoczesnego algorytmu detekcji cech szczególnych obrazu.

Aplikację frontendową można rozwinąć pod kątem wyświetlania większej ilości kart (obecnie są to dwie ostatnio położone poprawnie karty) oraz wyświetlania obrazu z kamery wykorzystywanej do analizy rozgrywki. Ponadto można wprowadzić funkcję zapamiętywania gier (np. do późniejszej dokładniejszej analizy), eksportu historii do pliku, manipulowanie kolejnymi krokami, np. powrotu do stanu gry sprzed kilku położonych kart.

Aplikację napisaną w języku Java, można rozwinąć o dodatkowe gry karciane, ponieważ ten moduł odpowiada za walidację ruchu, otrzymuje informacje o kartach i wysyła do aplikacji front-endowej czy dana karta jest poprawna, oraz umieszcza odpowiednią treść komunikatu.

12. Opis najważniejszych implementacji wraz z krótkim kodem źródłowym.

- Moduł rozpoznawania kart (skrypt Python)

Klasa VideoStream.py:

def start(self) - uruchamia wątek odpowiedzialny za odczytywanie klatek z kamery.

def update(self):

```
for f in self.stream:
        self.frame = f.array
        self.rawCapture.truncate(0)

        if self.stopped:
            self.stream.close()
            self.rawCapture.close()
            self.camera.close()
```

Pętla for odpowiada za ciągłe działanie odczytywania klatek dopóki wątek nie dostanie zakończony.

def read(self) - zwraca ostatnią zapisaną klatkę filmu.

Klasa Cards.py:

Stałe:

```
BKG_THRESH = 60
CARD_THRESH = 30

CORNER_WIDTH = 32
CORNER_HEIGHT = 84

RANK_WIDTH = 70
RANK_HEIGHT = 125

SUIT_WIDTH = 70
SUIT_HEIGHT = 100

RANK_DIFF_MAX = 2000
SUIT_DIFF_MAX = 700

CARD_MAX_AREA = 120000
CARD_MIN_AREA = 25000
```

W stałych określone są wartości progowe, szerokość i wysokość narożnika karty, gdzie znajduje się ranga oraz kolor. Reszta stałych wspomaga ocenę czy karta jest rozpoznana poprawnie. Im większe limity, tym większa pewność, że karta zostanie rozpoznana poprawnie, jednak kosztem szybkości działania.

def load_ranks(filepath):

```
train_ranks = []
i = 0

for Rank in ['Ace','Two','Three','Four','Five','Six','Seven',
             'Eight','Nine','Ten','Jack','Queen','King']:
    train_ranks.append(Train_ranks())
    train_ranks[i].name = Rank
    filename = Rank + '.jpg'
    train_ranks[i].img = cv2.imread(filepath+filename, cv2.IMREAD_GRAYSCALE)
    i = i + 1

return train_ranks
```

Wszystkie obrazy, które znajdują się w folderze ‘img’ są przygotowywane do porównywania z przechwyconymi klatkami. Zostają one załadowane do tablicy o nazwie train_ranks.

def load_suits(filepath):

```
train_suits = []
i = 0

for Suit in ['Spades','Diamonds','Clubs','Hearts']:
    train_suits.append(Train_suits())
    train_suits[i].name = Suit
    filename = Suit + '.jpg'
    train_suits[i].img = cv2.imread(filepath+filename, cv2.IMREAD_GRAYSCALE)
    i = i + 1

return train_suits
```

Podobna operacja jak w funkcji powyżej, lecz tym razem obiektem obrazów są typy kart, a nie numery.

def preprocess_image(image):

Odpowiada za zwrócenie zamazanego obrazu przechwyconego z kamery z adaptacyjnym progiem, który został ustawiony w stałych.

def preprocess_card(contour, image):

Funkcja preprocess_card używa konturu zwróconego przez funkcję find_cards() do znalezienia informacji o karcie. Dodatkowo oddziela informację o kolorze i wartości karty z obrazu.

def match_card(qCard, train_ranks, train_suits):

Znajduje najlepsze dopasowania rangi i koloru dla karty zapytania.

```
for Trank in train_ranks:

        diff_img = cv2.absdiff(qCard.rank_img, Trank.img)
        rank_diff = int(np.sum(diff_img)/255)

        if rank_diff < best_rank_match_diff:
            best_rank_diff_img = diff_img
            best_rank_match_diff = rank_diff
            best_rank_name = Trank.name
```

Pętla for odpowiedzialna za znalezienie najlepiej pasującej rangi karty. Karta, która zostaje dopasowana jako najlepsza jest to karta z najmniejszą różnicą względem obrazu bazowego.

```
for Tsuit in train_suits:

        diff_img = cv2.absdiff(qCard.suit_img, Tsuit.img)
        suit_diff = int(np.sum(diff_img)/255)

        if suit_diff < best_suit_match_diff:
            best_suit_diff_img = diff_img
            best_suit_match_diff = suit_diff
            best_suit_name = Tsuit.name
```

Ten sam proces, jednak dla koloru karty.

def draw_results(image, qCard):

Funkcja odpowiedzialna za wyświetlanie konturów i informacji o karcie na obrazie rzeczywistym wyświetlonym na ekranie komputera z kamery telefonu. Funkcja ta jest napisana dzięki bibliotece OpenCV.

Klasa CardDetector.py

Klasa ta wykorzystuje funkcje zaimplementowane w klasach Cards.py i VideoStream.py.

Stałe:

```
IM_WIDTH = 1280
IM_HEIGHT = 720
FRAME_RATE = 10
```

def most_frequent(List):

```
def most_frequent(List):
    return max(set(List), key = List.count)
```

Funkcja pomocniczna pozwalająca na lepsze określenie dopasowania karty. Po 100 odczytach predykcyjnych karty do aplikacji backendowej wysyłana jest informacja o karcie najczęściej powtarzającej się.

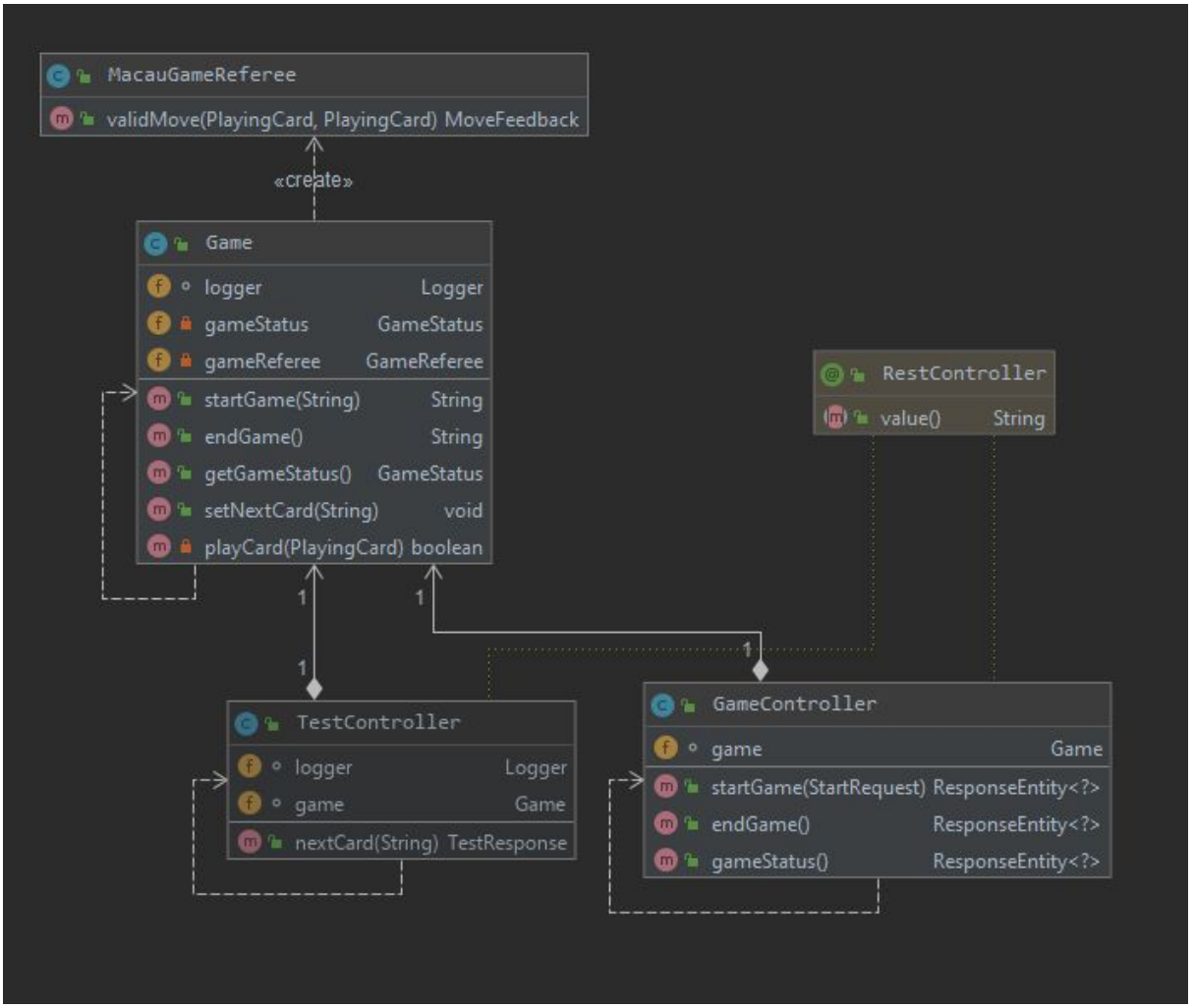
while cam_quit == 0:

Pętla główna działająca do momentu wyłączenia kamery.

```
image = Cards.draw_results(image, cards[k])
if(cards[k].best_rank_match != "Unknown" and cards[k].best_suit_match != "Unknown"):
    #print(cards[k].best_rank_match, cards[k].best_suit_match)
    if(len(cards_history)!=0):
        if(cards_history[len(cards_history) - 1] != (cards[k].best_rank_match + " " + cards[k].best_suit_match)):
            last_card.append(cards[k].best_rank_match + " of " + cards[k].best_suit_match)
    else:
        last_card.append(cards[k].best_rank_match + " of " + cards[k].best_suit_match)
    #print(last_card)
k = k + 1
if(len(last_card)==100):
    cards_history.append(most_frequent(last_card))
    r = requests.get(url='http://192.168.0.82:8080/test/pyConnectTest?card='+most_frequent(last_card))
    data = r.json()
    print("CARDS HISTORY", str(cards_history))
    last_card=[]
```

Część kodu odpowiedzialna za obsługę informacji predykcyjnych, które zostały odczytane przez program. Jeśli dobrze odczytana jest ranga i kolor, wtedy predykcja zostaje dodana do tablicy z odpowiednim opisem. Jeśli długość tablicy osiągnie wartość 100. Wtedy użyta zostaje funkcja most_frequent(), która podaje najczęściej występujący obiekt w tablicy. Taka karta zostaje wysłana do aplikacji backendowej i zapisana do tablicy, w której znajduje się historia kart w grze.

- Moduł analizy poprawności rozgrywki (backend)



Zdjęcie przedstawia diagram najważniejszych klas projektu Java.

Cała aplikacja backendowa polega na walidowaniu i poprawnym obsługiwaniu zapytań HTTP, możemy wyróżnić dwa kontrolery, TestController (komunikacja z aplikacją python) , GameController (komunikacja z aplikacja frontendową), obydwa mają endpointy, do komunikacji z 2 modułami. Wszystkie requesty obsługuje usługa o nazwie Game, która w przypadku obsługiwania zapytania HTTP od aplikacji python (dostarczającej kolejne karty), wywołuje metody klasy MacauGameReferee.

```
@GetMapping("/pyConnectTest")
public TestResponse nextCard(@RequestParam String card){
    game.setNextCard(card);
    return new TestResponse(card);
}
```

Zdjęcie przedstawia endpoint /test/pyConnectTest

(Dostosowaliśmy już pracę 2 modułów do komunikacji na tym endpoincie, więc nazwa testowa została oficjalną)

Jest to endpoint, który odpowiada, za odbieranie rozpoznanych kart od modułu rozpoznawania kart, trzeba wykonać zapytanie typu GET, z parametrem o nazwie karty typu String. Zwracana jest odebrana karta.

```
@PostMapping
@RequestMapping("/startGame")
public ResponseEntity<?> startGame(@RequestBody StartRequest request){

    String message = game.startGame(request.getGameName());
    return ResponseEntity.ok(message);
}
@PostMapping
@RequestMapping("/endGame")
public ResponseEntity<?> endGame(){
    String message = game.endGame();
    return ResponseEntity.ok(message);
}
```

Zdjęcie przedstawia endpoint /api/game/startGame oraz /api/game/endGame

Są to endpointy odpowiadające za rozpoczęcie i zakończenie gry, endpoint startGame przyjmuje w ciele zapytania obiekt typu StartRequest, który w postaci .json wygląda tak:

```
{
  "gameName" : "mojaGra"
}
```

Jest tam tylko jedno pole, ponieważ jest furtka na dalszy rozwój, można na przykład dodać pole ilość ruchów, albo liczba graczy, które byłoby potrzebne do innych gier.

Zwracana jest wiadomość, “Game is already started” lub, “Game Started”, w zależności od tego czy grę można uruchomić.

Endpoint kończący grę nie zawiera parametrów, ani obiektów w ciele zapytania. Zwraca wiadomość “game is not running” jeżeli gra nie jest uruchomiona lub “end game”, jeżeli jest w stanie się zakończyć.

```
@GetMapping
@RequestMapping("/gameStatus")
public ResponseEntity<?>gameStatus(){
    return ResponseEntity.ok(game.getGameStatus());
}
```

Zdjęcie przedstawia endpoint /api/game/gameStatus

Jest to endpoint, który aplikacja frontendowa odpytuje cyklicznie, aby wychwycić zmiany w statusie gry, oraz w tym jakie karty, są aktualnie na stole.

W ciele odpowiedzi zwraca obiekt, który zawiera wszystkie potrzebne informacje dla aplikacji front endowej :


```
{
  "id": 16,
  "currentPlayingCard": {
    "label": "Clubs",
    "number": "Ten",
    "date": "2020-07-10T14:04:34.053+0000",
    "id": "5f59a04b0389ced70783d7028483781cc513cadd07f26c08c5181c6b5b5b830a"
  },
  "previousPlayingCard": {
    "label": "Hearts",
    "number": "Ace",
    "date": "2020-07-10T14:04:27.041+0000",
    "id": "91efd8d77033c389e98f1b4f8c0d7c3ff28ded0dae085eaf6c465b5168cad71d"
  },
  "gameEvent": "GOOD_MOVE",
  "message": "Keep playing",
  "name": "mojaGra",
  "gameRunning": true
}
```


id- unikalne id zmieniane dopiero, gdy zmieni się stan gry;

currentPlayingCard - aktualnie zagrana karta z unikalnym id, datą zagrania, numerem i kolorem

previousPlayingCard - poprzednio zagrana karta z unikalnym id, datą zagrania, numerem i kolorem

gameEvent - informacja czy ruch został wykonany poprawnie: wyróżniamy Eventy takie jak:

RUNNING - gra uruchomiona, ale nie wykonano jeszcze ruchu;

ERROR - błędny ruch;

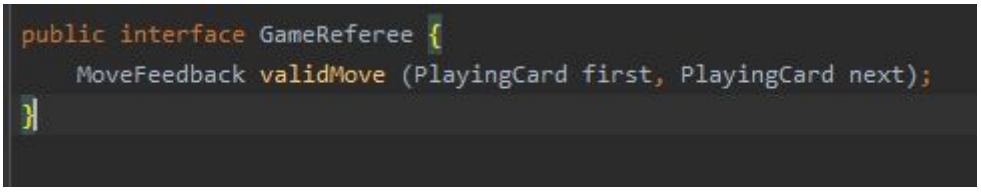
NOT_STARTED - gra, która jeszcze się nie rozpoczęłą;

GOOD_MOVE - wykonano poprawny ruch

Message - przedstawiona informacja do wyświetlenia użytkownikowi w formie tekstowej

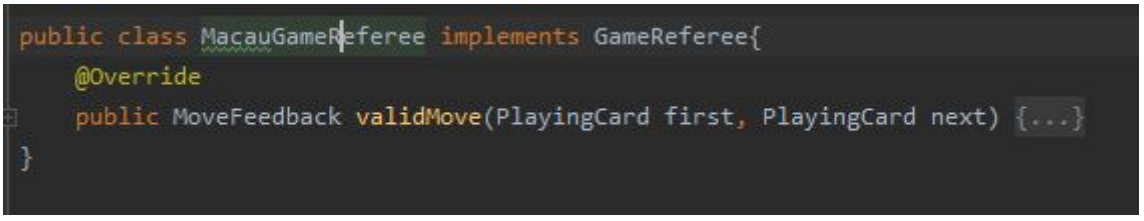
Name - nazwa aktualnej gry

gameRunning - flaga zawierająca wartości true, false, która mówi czy gra jest włączona



Zdjęcie przedstawia interfejs GameReferee

Implementowanie tego interfejsu pozwala tworzyć nowe zasady gry



Zdjęcie przedstawia klasę MacauGameReferee

Klasa, która implementuje interfejs GameReferee, musi nadpisać metodę validMove(), która sprawdza poprawność wykonanego ruchu, można tworzyć wiele klas implementujących ten interfejs w zależności od gry, którą chcemy obsłużyć tym modułem. W ciele metody validMove ustalamy zasady gry.

```java
public void setNextCard(String card){

    if(gameStatus.isGameRunning()){
        logger.info("Card received :" + card);
        PlayingCard newCard = new PlayingCard(card);
        if(gameStatus.getCurrentPlayingCard()==null || !gameStatus.getCurrentPlayingCard().isTheSame(newCard)){
            boolean isMoveValid = this.playCard(newCard);
            if(isMoveValid){
                this.gameStatus.setCurrentPlayingCard(newCard);
            }
        }else{
            logger.info("PyCamera Send The same card");
        }
    }
    else{
        logger.info("Card received, but game is not running");
    }
}
```

Zdjęcie przedstawia fragment klasy Game.java

Metoda setNextCard() waliduje czy ruch mógł się odbyć, jeżeli mógł to wywołuje metodę playCard().

```java
private boolean playCard(PlayingCard nextCard){
    MoveFeedback feedback = gameReferee.validMove(gameStatus.getCurrentPlayingCard(),nextCard);
    if(feedback.getGameEvent().equals(GameEvent.GOOD_MOVE)){
        this.gameStatus.setGameEvent(GameEvent.GOOD_MOVE);
        this.gameStatus.setMessage(feedback.getMesssage());
        return  true;
    }else{
        this.gameStatus.setGameEvent(GameEvent.ERROR);
        this.gameStatus.setMessage(feedback.getMesssage());
        return false;
    }
}
```

Zdjęcie przedstawia fragment klasy Game.java

Metoda playCard() wywołuje metodę validMove(), i obsługuje wyjście tej metody ustawiając odpowiedni stan gry oraz wiadomość dla modułu frontendowego

- Moduł komunikacji z użytkownikiem (frontend)

  Aplikacja frontendowa opiera swoje działanie na komunikacji z API serwera backendowego poprzez endpoint **http://localhost:8080/api/game/**, na który wysyłane są zapytania HTTP (REST API) w celu prezentowania aktualnego stanu gry

Za komunikację z serwerem odpowiada usługa o nazwie **CardsService**. Zawiera ona stałe, w których przechowywany jest adres endpointu oraz nagłówek dołączany do wysyłanych zapytań protokołu HTTP.

```
const CARDS_API = 'http://localhost:8080/api/game/';
const httpOptions = {
   headers: new HttpHeaders({ 'Content-Type': 'application/json'
}),
};
```

Nagłówek ten oznacza, że dane przesyłane są w postaci JSON (Java Script Object Notation).

```
export class CardsService {
  constructor(private http: HttpClient) {}

  startGame(gameName) {
    return this.http.post(
      CARDS_API + 'startGame',
      { gameName },
      { headers: httpOptions.headers, responseType: 'text' }
    );
  }

  endGame() {
     return this.http.get(CARDS_API + 'endGame', { responseType:
'text' });
  }

  getGameStatus() {
           return  this.http.get(CARDS_API  +  'gameStatus',  {
responseType: 'text' });
  }
}
```

Usługa ta zawiera trzy metody realizujące zapytania HTTP za pomocą wstrzykniętego do usługi klienta HTTP, dostępnego we frameworku Angular.

startGame(gameName) - ta metoda przyjmuje jako argument unikatową nazwę gry, tak aby rozróżnić między statusami różnych gier, wysyła ona do serwera zapytanie POST informujące serwer o rozpoczęciu gry,
endGame() - ta metoda wysyła zapytanie GET i wywołuje na serwerze zakończenie aktualnie trwającej rozgrywki,
getGameStatus() - jest to metoda wysyłająca zapytanie GET, w odpowiedzi uzyskuje od serwera aktualny status gry.

Każda z powyższych metod zwraca tzw. obserwablę, którą należy zasubskrybować, gdy pojawia się strumień danych odpowiedź jest emitowana - wszystko działa asynchronicznie.
Usługa ta realizuje jedynie komunikację z serwerem, w tym celu jest używana w głównym komponencie aplikacji o nazwie **AppComponent**, w którym natomiast realizowana jest już sama obsługa odpowiedzi serwera i główna logika działania aplikacji.

Na początku, w tzw. lifecycle hook **ngOnInit()**, czyli metodzie wykonywanej przy inicjalizacji komponentu (przy starcie aplikacji) pobierany jest status gry. Jeżeli w odpowiedzi serwera obecne jest pole **gameRunning**, wywołujemy zakończenie gry (aby uniknąć sytuacji, gdy gra z jakiegoś powodu działała w tle na serwerze, mimo niedziałającej aplikacji klienta)

```
ngOnInit() {
  this.cardsService.getGameStatus().subscribe((res) => {
    if (JSON.parse(res).gameRunning) {
      this.onEndGame();
    }
  });
}
```

Metoda **onEndGame()** wywołuje zakończenie rozgrywki, przy poprawnej odpowiedzi resetowane są wszystkie zmienne pomocnicze, a także emitowany jest tzw. Subject **destroyIntervalPoll**, który wykorzystywany jest

przy nasłuchiwaniu statusu gry. W przypadku błędu wyświetlany jest on wraz z pozostałymi wiadomościami w historii gry.

```
  onEndGame() {
    this.cardsService.endGame().subscribe(
      () => {
        this.gameStarted = false;
        if (this.destroyIntervalPoll) {
          this.destroyIntervalPoll.next();
          this.destroyIntervalPoll.complete();
        }
        this.previousRequestId = null;
        this.previousCard = null;
        this.currentCard = null;
        this.messages = [];
      },
      (err) => {
this.messages.push(`🔴 ERROR: ${JSON.parse(err.error).message}`);
      }
    );
  }
```

Metoda **onEndGame()** wywołuje zakończenie rozgrywki, przy poprawnej odpowiedzi resetowane są wszystkie zmienne pomocnicze, a także emitowany jest tzw. Subject **destroyIntervalPoll**, który wykorzystywany jest przy nasłuchiwaniu statusu gry. W przypadku błędu wyświetlany jest on wraz z pozostałymi wiadomościami w historii gry.

Metoda **onStartGame()** powoduje rozpoczęcie rozgrywki, w przypadku poprawnej odpowiedzi serwera wywoływana jest metoda **listenToStatus()**, w przypadku błędu wyświetlana jest odpowiednia wiadomość.

```
  onStartGame() {
    this.cardsService.startGame().subscribe(
      (res) => {
        this.gameStarted = true;
        this.messages.push(res);
        this.listenToStatus();
      },
```

```
      (err) => {
this.messages.push(`🔴 ERROR: ${JSON.parse(err.error).message}`);
      }
    );
  }
```

Najważniejszą metodą w aplikacji frontendowej jest właśnie **listenToStatus()**, czyli metoda odpowiedzialna za odbieranie statusu rozgrywki z serwera i odpowiednią aktualizację stanu gry.

Na początku inicjalizowany jest tzw. Subject, który wykorzystywany będzie do "zabijania" subskrypcji (działającej asynchronicznie), w celu uniknięcia przecieków pamięci i istnienia więcej niż jednej subskrypcji wykonującej operacje jednocześnie w tym samym czasie, co powodowałoby przekłamania w statusie gry.

Następnie poprzez **interval(3000)** powodujemy wysyłanie zapytania nieustannie co 3 sekundy i przekazujemy tzw. operatory:

takeUntil() - oznacza, że **interval** będzie "żył" tak długo, aż Subject **destroyIntervalPoll** wyemituje jakąkolwiek wartość

concatMap() - nie pozwoli na utworzenie nowej subskrypcji tak długo, aż poprzednia subskrypcja nie zostanie zakończona.

Następnie dokonujemy subskrypcji i w niej znajduje się funkcja strzałkowa obsługująca odpowiedź. Gdy otrzymamy poprawną odpowiedź z serwera sprawdzamy czy jej **id** jest inne, niż **id** poprzedniego zapytania (aby uniknąć aktualizacji statusu w przypadku, gdy stan gry się nie zmienił - każde zapytania ma inne **id**). Następnie jeżeli ruch był poprawny ustawiane są wszystkie zmienne pomocnicze, takie jak **currentCard**, **previousCard**, **currentCardName**, a także zdjęcie jakie reprezentuje daną kartę, które wybierane jest na podstawie mapy (opisanej poniżej). W przypadku niepoprawnego ruchu wypisywana jest odpowiednia wiadomość w historii gry.

```
listenToStatus() {
    this.destroyIntervalPoll = new Subject<null>();
```

```
    interval(3000)
      .pipe(
        takeUntil(this.destroyIntervalPoll),
        concatMap(() => {
          return this.cardsService.getGameStatus();
        })
      )
      .subscribe(
        (res) => {
          const parsed = JSON.parse(res);
          if (parsed.id !== this.previousRequestId) {
            if (parsed.gameEvent !== 'ERROR') {
              if (parsed.currentPlayingCard) {
                if (this.currentCard) {
                  this.previousCard = this.currentCard;
                }
this.currentCard = parsed.currentPlayingCard;
const currentCardName = `${parsed.currentPlayingCard.number} of
${parsed.currentPlayingCard.label}`;
this.currentCard.img = this.cardsMap.get(currentCardName);
this.messages.push(`□ ${currentCardName}: ${parsed.message}`);
              }
            } else {
              this.messages.push(`🔴 WRONG! ${parsed.message}`);
            }
            this.previousRequestId = parsed.id;
          }
        },
        (err) => {
this.messages.push(`🔴 ERROR: ${JSON.parse(err.error).message}`);
        }
      );
  }
```

Grafiki wektorowe reprezentujące poszczególne karty przypisane są do nazw kart, np. “Two of Clubs” -> “2C.svg” i na tej podstawie wyświetlane w aplikacji.

```
  cardsMap = new Map([
    ['Two of Clubs', '2C.svg'],
    ['Two of Diamonds', '2D.svg']...
```

```
    ['Ace of Spades', 'AS.svg'],
  ]);
```

Gdy aplikacja zostanie odświeżona lub gra zostanie zakończona sam komponent umiera i wywoływany jest lifecycle hook **ngOnDestroy()**, w którym jeżeli istnieje **destroyIntervalPoll**, wtedy metodą **next()** powodujemy wyemitowanie pustej wartości, co z kolei powoduje uruchomienie się **takeUntil()** w metodzie **listenToStatus()**, co wywołuje przerwanie nasłuchiwania, następnie używamy **complete()**, co powoduje samo zamknięcie Subjectu tak, aby również jego instancje nie duplikowały się w przypadku nowej gry.

```
  ngOnDestroy() {
    if (this.destroyIntervalPoll) {
      this.destroyIntervalPoll.next();
      this.destroyIntervalPoll.complete();
    }
  }
```