AKADEMIA GÓRNICZO-HUTNICZA
im. Stanisława Staszica w Krakowie
WEAIIB, Katedra Automatyki i Robotyki

Przedmiot: Interfejsy człowiek-komputer HCI2025_4_OPENFACE

Temat projektu: Tiny Creatures + Kod Źródłowy Rzeczywistości – modalność typu OPENFACE

## Spis treści



Wykonali: Tim Kusznir, Piotr Mistarz, Tomasz Morawa, Mateusz Zajda
*rok IV / AiR / Inteligentne Systemy Sterowania*
konsultant: *Jaromir Przybyło*

Wersja 1.0
Kraków, kwiecień 2025.

# **1.** Abstrakt

Celem projektu było stworzenie interfejsu sterowania w grze „Tiny Creatures + Kod Źródłowy Rzeczywistości” przy użyciu technologii rozpoznawania gestów twarzy. Rozwiązanie opierało się na generowaniu siatki punktów na twarzy gracza, umożliwiając identyfikację kluczowych wyrazów twarzy, takich jak uśmiech, otwarcie ust, zamknięcie oczu czy zmiana kierunku patrzenia. Spośród dostępnych opcji technologicznych wybrano Google MediaPipe jako podstawę implementacji interfejsu. Sygnał odzwierciedlający mimikę twarzy był determinowany przez pozycję poszczególnych punktów względem punktu centralnego twarzy lub przez wzajemną, względną pozycję punktów, w zależności od rozpoznawanej emocji lub gestu.

Wykryte gesty zostały przetłumaczone na komunikaty w postaci zer i jedynek, które następnie przesyłano do gry za pomocą socketów UDP. System zapewnił płynność oraz responsywność interakcji w czasie rzeczywistym, zwiększając immersję i intuicyjność rozgrywki.

Zastosowanie MediaPipe w projekcie podkreśla potencjał tej technologii w rozwijaniu nowych form interakcji w grach, łączących zaawansowaną analizę gestów z dynamiczną rozgrywką, co znacząco wpływa na satysfakcję gracza.

Słowa kluczowe: MediaPipe, Rozpoznawanie twarzy, Face Landmarker task, Uczenie maszynowe, Python, Sterowanie multimodalne.

## 2. Wstęp i badania literaturowe

### a) Cele i założenia projektu

Celem projektu było stworzenie interfejsu sterowania grą za pomocą gestów twarzy, wykorzystującego technologię rozpoznawania mimiki. Skoncentrowano się na detekcji gestów takich jak: uśmiech, otwarcie ust, zamknięcie oczu oraz ruch głowy w czterech kierunkach. Projekt zakładał integrację tego interfejsu z grą 2D „Tiny Creatures", w której gesty twarzy umożliwiają sterowanie atakami specjalnymi, leczeniem postaci oraz mechanizmami obronnymi.

Mimo rosnącej popularności badań nad rozpoznawaniem gestów twarzy, istnieje potrzeba dalszej optymalizacji algorytmów, aby mogły one działać skutecznie w czasie rzeczywistym, szczególnie w kontekście interaktywnych aplikacji, takich jak gry komputerowe.

### b) Zarys ogólny proponowanego rozwiązania

Proponowane rozwiązanie opiera się na wykorzystaniu rozpoznawania gestów twarzy jako interfejsu sterowania w grze. Zastosowano bibliotekę MediaPipe [1][2], wyróżniającą się wysoką precyzją oraz dużą liczbą punktów charakterystycznych twarzy, co znacząco ułatwia dokładne rozpoznawanie złożonych gestów.

Gesty takie jak uśmiech (atak specjalny), otwarcie ust (leczenie) oraz ruch głowy (obrona) były przekształcane na komendy sterujące, zapewniając użytkownikowi intuicyjną i płynną kontrolę nad postacią w grze. Komunikacja z grą, stworzoną w silniku Godot, realizowana była za pomocą protokołu UDP, umożliwiającego szybkie i niezawodne przesyłanie danych w czasie rzeczywistym. Kod serwera i klienta został opracowany w oparciu o dokumentację silnika [3].

### c) Dyskusja alternatywnych rozwiązań

W literaturze opisano różne podejścia do rozpoznawania gestów twarzy, z wykorzystaniem bibliotek takich jak OpenFace [4], Dlib [5] oraz MediaPipe. OpenFace, choć oferuje zaawansowaną analizę emocji, dysponuje ograniczoną liczbą punktów charakterystycznych, co utrudnia precyzyjne wykrywanie gestów. Z kolei Dlib zapewnia jedynie 68 punktów, co w wielu przypadkach okazuje się niewystarczające przy bardziej złożonych interakcjach.

MediaPipe okazała się najskuteczniejszym rozwiązaniem – dzięki 468 punktom detekcji, obsłudze szerokiego zakresu gestów oraz wysokiej wydajności działania w czasie rzeczywistym. Dodatkową zaletą biblioteki była rozbudowana dokumentacja oraz liczne przykłady implementacyjne, co znacznie przyspieszyło integrację systemu z grą oraz umożliwiło dostosowanie go do specyficznych wymagań projektu.

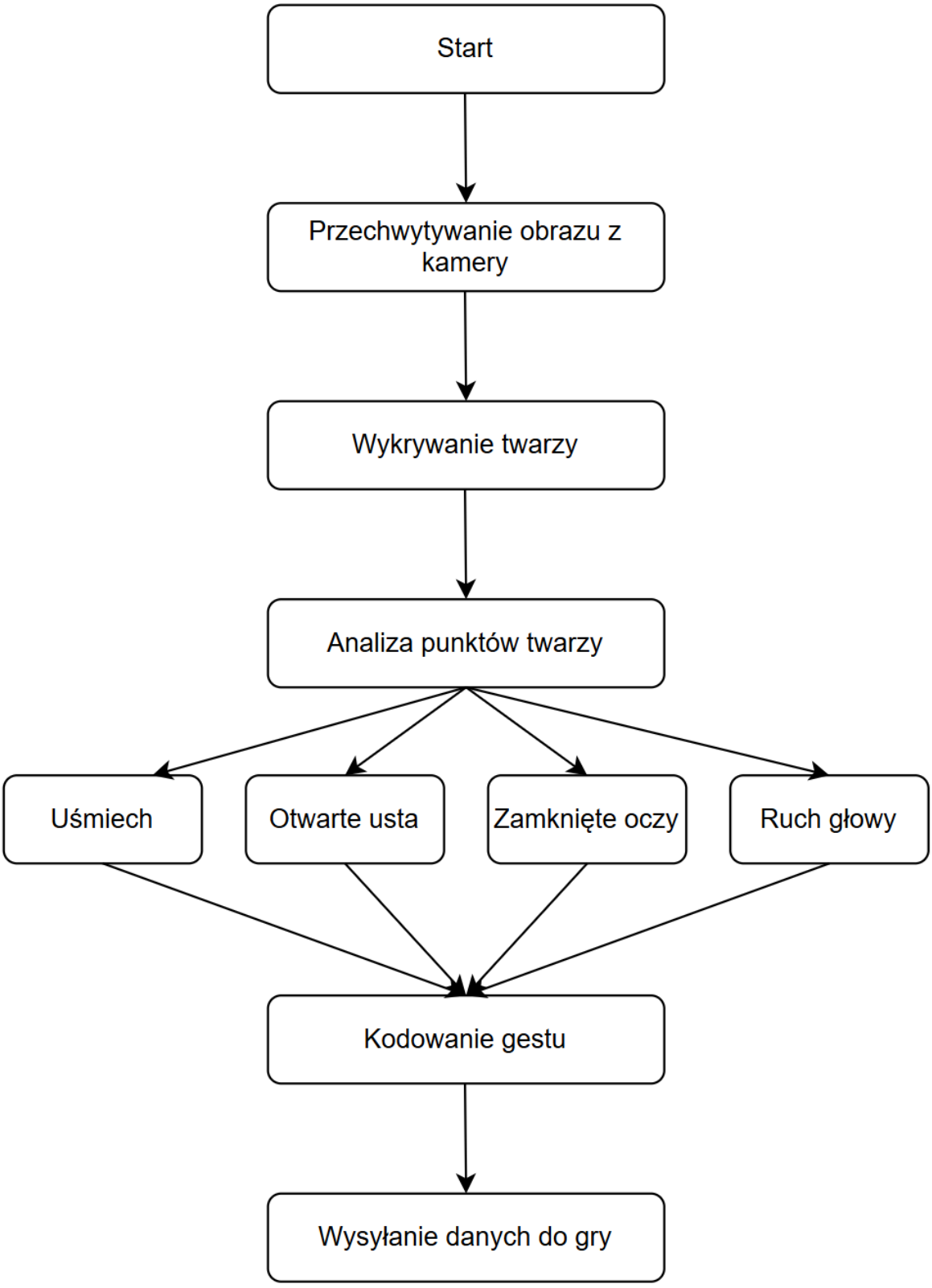
## 3. Koncepcja proponowanego rozwiązania

W proponowanym rozwiązaniu zastosowano bibliotekę MediaPipe do generowania siatki punktów charakterystycznych twarzy użytkownika w czasie rzeczywistym. Analiza odległości pomiędzy wybranymi punktami umożliwia detekcję gestów, takich jak: uśmiech, otwarcie ust, zamknięcie oczu oraz ruch głowy w czterech kierunkach (góra, dół, lewo, prawo).

Rozpoznane gesty są kodowane binarnie i przesyłane do gry „Tiny Creatures" za pośrednictwem protokołu UDP. W grze sygnały te sterują odpowiednimi akcjami rozgrywki, co umożliwia płynne i intuicyjne sterowanie mimiką twarzy, znacząco zwiększając immersję i interaktywność rozgrywki.

Poniżej przedstawiono opis głównych etapów działania systemu oraz schemat blokowy (Rys. 1):

- Przechwytywanie obrazu – ciągły strumień wideo z kamery internetowej dostarczany do systemu w czasie rzeczywistym.
- Detekcja twarzy i generowanie siatki punktów – wykorzystanie modelu Face Landmarker z biblioteki MediaPipe do uzyskania 468 punktów charakterystycznych twarzy.
- Analiza gestów – identyfikacja mimiki oraz ruchów głowy na podstawie analizy wybranych punktów siatki:
    - Uśmiech – ocena odległości między kącikami ust a ich środkiem; dodatkowo skalowanie dystansu głowy względem punktów zlokalizowanych w okolicach skroni.
    - Otwarcie ust – pomiar pionowej odległości pomiędzy górną a dolną wargą.
    - Zamknięcie oczu – analiza odległości pomiędzy górną a dolną powieką; normalizacja względem szerokości oczu.
    - Ruch głowy – identyfikacja kierunku przemieszczenia nosa względem pozycji referencyjnej (centralnej).
- Kodowanie gestu – konwersja wyników analizy na ciąg sygnałów binarnych (ciąg zer i jedynek), reprezentujących konkretne gesty.
- Wysyłanie sygnału – przesyłanie zakodowanych gestów do gry przy użyciu protokołu UDP.



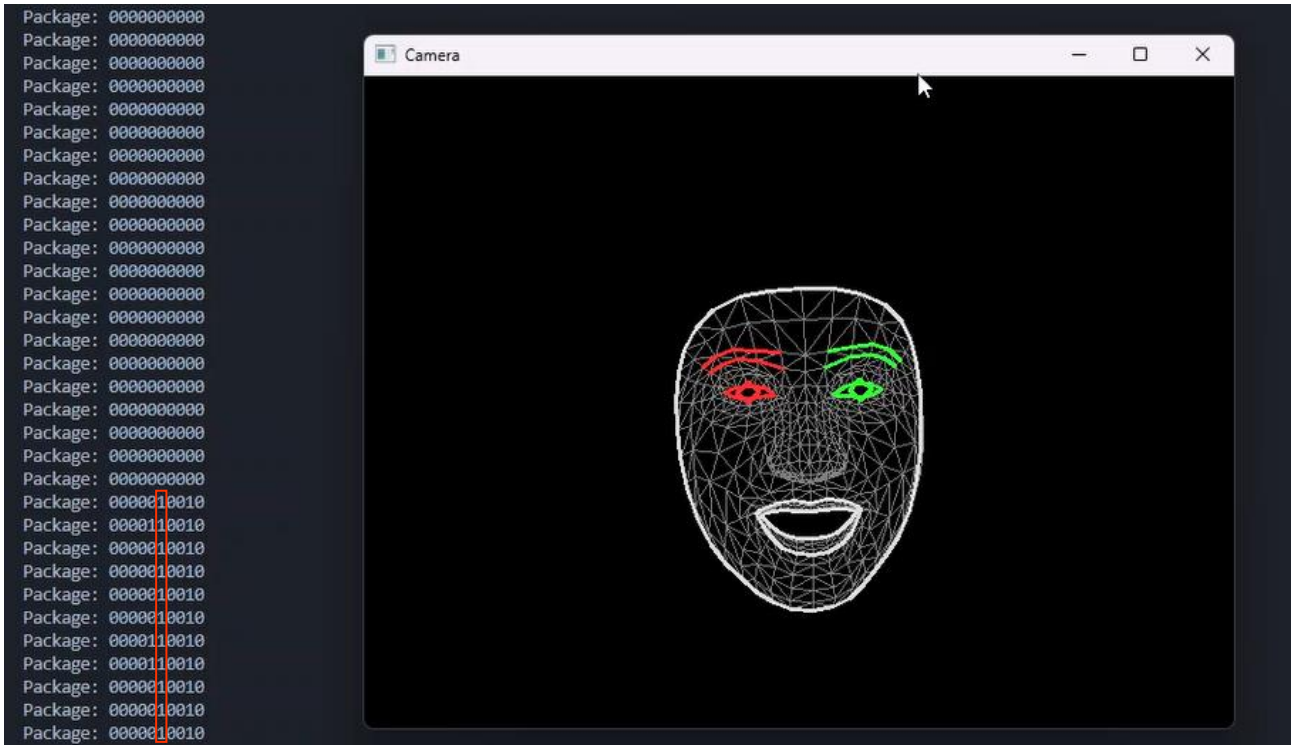

**Rys. 1 Schemat blokowy algorytmu**

## 4. Rezultaty i wyniki testów i/lub realizacja zadań projektowych

W ramach projektu opracowano system działający w czasie rzeczywistym, umożliwiający rozpoznawanie wybranych gestów twarzy (uśmiech, otwarcie ust, zamknięcie oczu, ruch głowy) przy użyciu modelu Face Landmarker z biblioteki MediaPipe. Rozwiązanie przetwarza obraz z kamery wideo i na podstawie punktów charakterystycznych twarzy generuje sygnały binarne, które są następnie przesyłane za pomocą protokołu UDP.

Analiza błędów pomiarowych wykazała, że największy wpływ na dokładność działania ma zmienność warunków oświetleniowych. W ciemnych pomieszczeniach dokładność wykrywania punktów charakterystycznych twarzy maleje.

Podczas testów zaobserwowano, że poruszanie głową poza wyznaczony obszar skutkowało wygenerowaniem odpowiedniego sygnału informującego o wykrytym ruchu. Analogiczna sytuacja miała miejsce w przypadku pozostałych gestów — uśmiech, otwarcie ust oraz zamknięcie oczu również powodowały pojawienie się sygnału w momencie przekroczenia ustalonych progów detekcji. Dzięki temu możliwe było jednoznaczne rozpoznanie, kiedy dany gest został wykonany, co potwierdza poprawność działania zastosowanych mechanizmów analizy i kodowania gestów. Wszystkie sygnały były następnie poprawnie przesyłane za pośrednictwem protokołu UDP, co umożliwiało ich bieżące odbieranie i interpretowanie przez grę. Efekty pracy pokazane zostały na Rys. 2.




**Rys. 2 Efekt działania aplikacji podczas gdy gracz się uśmiechał (piąta jedynka od końca w paczce sygnału)**

## 5. Realizacja proponowanego rozwiązania

W ramach realizacji projektu stworzenia aplikacji rozpoznającej gesty twarzy i głowy w czasie rzeczywistym, skupiono się na zapewnieniu prawidłowego działania systemu detekcji oraz komunikacji z zewnętrznymi urządzeniami, takimi jak serwer gry.

Do stworzenia projektu korzystano z systemu operacyjnego Windows oraz język programowania Python, który jest szeroko stosowany w dziedzinie analizy obrazu i uczenia maszynowego. Wykorzystano również popularne biblioteki, takie jak OpenCV do przetwarzania obrazu oraz OpenFace do wykrywania punktów charakterystycznych twarzy. Konieczne jest też posiadanie kamery do przechwytywania obrazu.

Algorytm rozpoczyna się od przechwytywania obrazy z kamery a następnie wykorzystano algorytm detekcji twarzy. Pozwala on na wykrycie punktów charakterystycznych koniecznych do dalszych części takie jak oczy, usta czy kontury twarzy. Na podstawie wykrytych punktów twarzy analizowane są gesty, takie jak zamknięcie oczu, uśmiech czy ruch głowy. Zastosowano algorytmy obliczające stosunki odległości między punktami oczu i ust, aby rozpoznać zamknięte oczy i uśmiech. Ruchy głowy natomiast są wykrywane przez sprawdzanie pozycji twarzy w przestrzeni 2D. Wykryte gesty i zmiany w mimice twarzy są przesyłane do serwera gry za pomocą protokołu UDP. Moduł komunikacyjny umożliwia szybką wymianę danych między aplikacją a serwerem, co pozwala na natychmiastową reakcję gry na zmiany w zachowaniu użytkownika. W zależności od wykrytego gestu, wysyłane są odpowiednie sygnały.

Podczas realizacji projektu jeden z członków zespołu odpowiadał za komunikację z serwerem grupy GAME oraz aplikację sterującą, podczas gdy pozostali zajmowali się opracowywaniem funkcji umożliwiających definiowanie mimiki twarzy i gestów gracza. Podział gestów na funkcje zwracające wartość logiczną pozwolił na ich równoczesne wykonywanie, a następnie synchronizację całego rozwiązania podczas jednego spotkania. Dodatkowo przygotowano plik config.json, który określał parametry takie jak czas wymagany na zamknięcie oczu oraz kolejność sygnałów.

Do komunikacji uczestnicy projektu wykorzystali platformę Microsoft Teams, natomiast do celów stricte związanych z umieszczaniem swoich rozwiązań zdecydowano się na skorzystanie z GitHub, na którym można znaleźć obecnie demo projektu oraz kod źródłowy [6].

# 6. Podsumowanie i wnioski

Udało się osiągnąć cel pracy, jakim było opracowanie systemu umożliwiający detekcję ruchów głowy oraz gestów takich jak otwieranie ust, uśmiech czy zamykanie oczu oraz wysyłanie informacji do gry za pomocą protokołu UDP. Wykonaną implementację testowano na zajęciach stacjonarnych, gdzie po połączeniu z serwerem gry można było wysyłać sygnały wykrytych gestów. Testy wykazały, że wykorzystanie modelu Face Landmarker z biblioteki MediaPipe do detekcji punktów charakterystycznych twarzy spełnia postawione wymagania i stanowi skuteczne oraz wydajne narzędzie do rozpoznawania gestów w czasie rzeczywistym. Występują jednak pewne ograniczenia takie jak zakłócenia wynikające z warunków oświetleniowych. W przyszłych pracach nad projektem można spróbować uniezależnić poprawność działania modelu od warunków oświetleniowych oraz dodać większą liczbę gestów mimiki twarzy.

## 7. Literatura

[1] https://ai.google.dev/edge/mediapipe/solutions/vision/face_landmarker/index#models (dostęp w dniu: 20.03.2025)

[2] https://github.com/google-ai-edge/mediapipe-samples/tree/main (dostęp w dniu: 21.03.2025)

[3] https://docs.godotengine.org/en/stable/classes/class_udpserver.html (dostęp w dniu: 10.04.2025)

[4] https://github.com/TadasBaltrusaitis/OpenFace (dostęp w dniu: 20.03.2025)

[5] https://dlib.net/ (dostęp w dniu: 20.03.2025)

[6] https://github.com/ICK2025OPENFACE/ICK2025 (dostęp w dniu 10.05.2025)

# 8. Opis opracowanych narzędzi i metody postępowania

Oprogramowanie zostało stworzone w języku Python i wykorzystuje biblioteki: MediaPipe (Face Landmarker), OpenCV oraz socket (komunikacja UDP). Do uruchomienia aplikacji wymagane jest:

- zainstalowanie środowiska Python 3.10 lub wyższego,
- instalacja bibliotek: media pipe, python, opencv, time, json, os requests, socket, math, typing
- kamera internetowa,

Następnie należy pobrać repozytorium Github [6]. Aby uruchomić program należy otworzyć plik face.py. System wysyła dane w postaci ciągu binarnego reprezentującego wykryte gesty (np. 01010100). Każda pozycja w ciągu odpowiada konkretnemu gestowi: uśmiech, zamknięcie oczu, otwarcie ust, ruch głowy (góra/dół/lewo/prawo). Komunikacja odbywa się przez UDP na skonfigurowany adres IP i port serwera gry. Funkcje rozpoznające konkretny gest twarzy zawarte zostały w pliku faceexpressions.py. Na przykład aby zmienić punkty opisujące uśmiech należy zmodyfikować punkty landmarks[] w funkcji detect_smile_and_open_mouth zgodnie z dokumentacją media pipe na inne.

System testowany był z wykorzystaniem systemu Windows 10, wpływ na działanie algorytmu może mieć jakość wykorzystywanej kamery oraz poziom oświetlenia.

# 9. Opis informatyczny procedur

## *Środowisko programowania*

Projekt został stworzony w języku Python 3.12, a do jego edycji i uruchamiania wykorzystywane jest Visual Studio Code. Skrypt główny (face.py) oraz pliki zawierające pomocnicze funkcje (supportfunctions.py i faceexpressions.py) znajdują się w głównym katalogu.
Dodatkowo w folderze archived przechowywane są archiwalne pliki testowe. Plik face_config.json zawiera konfiguracje dla funkcji, które zostaną później opisane szczegółowo.
Model wykorzystywany w projekcie to face_landmarker.task – plik dostarczony przez Google, znajdujący się w głównym katalogu projektu. Ze względu na jego rozmiar, został dodany do .gitignore, by nie został przesyłany do repozytorium GitHub.

## *Struktura projektu*

Pliki projektu przechowywane są w jednym directory ze względu na kompaktowość rozwiązania. Rzeczone pliki to:

- face.py – Główny plik uruchamiający cały proces.
- supportfunctions.py – Zawiera funkcje wspierające główny skrypt.
- faceexpressions.py – Moduł do analizy mimiki twarzy.
- archived/ – Folder zawierający pliki testowe i archiwalne wersje skryptów.
- face_config.json – Plik konfiguracyjny używany przez funkcje.
- face_landmarker.task – Model detekcji twarzy.
- .gitignore – Lista plików pomijanych w repozytorium, zawierająca m.in.. modele .task.

## *Biblioteki użyte w projekcie*

Projekt korzysta z następujących zależności:

- Mediapipe – Obsługa detekcji twarzy i landmarków.
- OpenCV (cv2) – Przetwarzanie obrazu z kamery.
- time – Obsługa czasu i opóźnień w kodzie.
- json – Wczytywanie i zapis danych konfiguracyjnych.
- os – Operacje na plikach i katalogach.
- requests – Wysyłanie zapytań HTTP.
- socket – Obsługa połączeń sieciowych.
- math – Obliczenia matematyczne.
- typing – Definiowanie typów danych.

## *Instrukcja uruchomienia*

W celu uruchomienia projektu należy wpisać w konsoli polecenie:

`python ./face.py`

w wyniku którego pobierany jest model o rozszerzeniu .task oraz sprawdzane jest występowanie pliku konfiguracyjnego. Jeżeli model oraz face_config.json znajdują się w katalogu to program zostanie uruchomiony i będzie przekazywał sygnały do zdefiniowanego serwera UDP.

## *Opis Funkcji*

```
Funkcja:
    camera_proc()

Przeznaczenie:
    Funkcja inicjalizująca proces przetwarzania obrazu z kamery.
    Ustawia strumień wideo, konfiguruje model FaceLandmarker oraz analizuje mimikę twarzy.

Argumenty funkcji:
    Brak

Funkcja zwraca:
    None

Używane funkcje:
    * cv2.VideoCapture() - Inicjalizacja strumienia kamery.
    * cv2.cvtColor() - Konwersja kolorów.
    * FaceLandmarker.create_from_options() - Inicjalizacja modelu detekcji twarzy.
    * landmarker.detect_async() - Detekcja punktów charakterystycznych twarzy.
    * cv2.imshow() - Wyświetlanie obrazu.
    * cv2.waitKey() - Obsługa klawisza 'q' do zamknięcia aplikacji.
    * cv2.getTickCount() - Pobranie wartości zegara systemowego.
    * cv2.getTickFrequency() - Częstotliwość zegara systemowego.
    * cam.read() - Pobranie klatki obrazu z kamery.
    * cam.release() - Zwolnienie zasobów kamery.
    * cv2.destroyAllWindows() - Zamknięcie okien wyświetlających obraz.
    * print() – Wyświetlenie komunikatów o błędach.
    * sf.draw_landmarks_on_image() - Rysowanie punktów na obrazie.

Używane zmienne:
    - cam: Uchwyt do strumienia wideo.
    - frame: Pojedyncza klatka obrazu.
    - frame_rgb: Obraz w formacie RGB (po konwersji z BGR).
    - mp_image: Obiekt obrazu dla modelu FaceLandmarker.
    - detection_result: Wynik analizy detekcji twarzy.
    - SHOW_CAMERA: flaga określająca, czy wyświetlić przetworzony obraz.
    - options: Konfiguracja dla modelu FaceLandmarker.
    - landmarker: Instancja modelu detekcji twarzy.

Uwagi:
    - Funkcja działa w trybie ciągłym, dopóki nie zostanie przerwana klawiszem 'q'.
    - Obsługa wyjątków zapobiega niekontrolowanemu zakończeniu działania skryptu.

Autor:
    Mateusz Zajda
```

```
Funkcja:
    camera_callback()

Przeznaczenie:
    Funkcja obsługująca wyniki detekcji twarzy przy użyciu modelu MediaPipe FaceLandmarker.
    Analizuje mimikę twarzy oraz przesyła odpowiednie sygnały do serwera gry przez UDP.

Argumenty funkcji:
    - result: FaceLandmarkerResult - Wynik detekcji punktów charakterystycznych twarzy.
        Zawiera listę obiektów NormalizedLandmark, które są dalej przetwarzane.
    - output_image: mp.Image - Obraz wymagany przez MediaPipe dla przetwarzania wyników.
    - timestamp_ms: int - Czas wykonania detekcji w milisekundach.

Funkcja zwraca:
    None

Używane funkcje:
    * socket.socket() - Tworzenie gniazda UDP.
    * fe.check_eyes_closed() - Analiza zamykania oczu.
    * fe.detect_smile_and_open_mouth() - Analiza uśmiechu i otwierania ust.
    * fe.detect_head_movement() - Wykrywanie ruchów głowy.
    * sf.send_msg_via_udp() - Wysyłanie sygnałów do serwera gry.
    * print() - Wyświetlenie komunikatów o błędach.
    * time.time() - Pobranie aktualnego czasu.

Używane zmienne:
    - detection_result: Globalna zmienna przechowująca wynik detekcji do wizualizacji.
    - center: Globalna zmienna przechowująca pozycję twarzy.
    - udp_socket: Globalna zmienna przechowująca gniazdo UDP do komunikacji.
    - face_config: Plik konfiguracyjny zawierający mapowanie sygnałów.
    - SERVER_IP, SERVER_PORT: Parametry połączenia z serwerem gry.

Uwagi:
    - Funkcja działa jako callback dla modelu detekcji twarzy i wywoływana jest automatycznie.
    - Wykryte ruchy i ekspresje twarzy są konwertowane na sygnały dla serwera gry.

Autor:
    Mateusz Zajda
```

```
Funkcja:
    draw_landmarks_on_image()

Przeznaczenie:
    Funkcja rysuje punkty charakterystyczne twarzy na obrazie RGB na podstawie wyników detekcji.
    Wykorzystuje narzędzia rysowania MediaPipe do wizualizacji punktów i konturów twarzy oraz oka.

Argumenty funkcji:
    - rgb_image: numpy.ndarray - Obraz wejściowy w formacie RGB.
    - detection_result - Wynik detekcji zawierający punkty twarzy.

Funkcja zwraca:
    - numpy.ndarray - Obraz z naniesionymi punktami charakterystycznymi twarzy.

Używane funkcje:
    * np.copy() - Tworzenie kopii obrazu wejściowego.
    * landmark_pb2.NormalizedLandmarkList() - Tworzenie struktury dla punktów twarzy.
    * solutions.drawing_utils.draw_landmarks() - Rysowanie punktów twarzy na obrazie.
    * mp.solutions.drawing_styles.get_default_face_mesh_tesselation_style() - Styl rysowania siatki twarzy.
    * mp.solutions.drawing_styles.get_default_face_mesh_contours_style() - Styl rysowania konturów twarzy.
    * mp.solutions.drawing_styles.get_default_face_mesh_iris_connections_style() - Styl rysowania tęczówki.

Używane zmienne:
    - face_landmarks_list: List - Lista wykrytych punktów twarzy.
    - annotated_image: numpy.ndarray - Kopia obrazu do edycji.
    - face_landmarks_proto: NormalizedLandmarkList - Obiekt przechowujący punkty twarzy do wizualizacji.

Uwagi:
    - Funkcja iteruje po wszystkich wykrytych twarzach i nakłada na obraz odpowiednie punkty.
    - Obsługuje kontury twarzy, siatkę oraz punkty tęczówki.
    - Kod pochodzi z githuba z repozytorium mediapipe-samples: https://github.com/google-ai-edge/mediapipe-
samples/blob/main/examples/face_landmarker/python/%5BMediaPipe_Python_Tasks%5D_Face_Landmarker.ipynb

Autor:
    Mateusz Zajda
```

```
Funkcja:
    send_msg_via_udp()

Przeznaczenie:
    Funkcja wysyła wiadomość przez UDP do określonego serwera gry.

Argumenty funkcji:
    - msg: str - Wiadomość do wysłania (konwertowana do stringa, jeśli to konieczne).
    - udp_socket: socket.socket - Obiekt gniazda UDP.
    - server_ip: str - Adres IP serwera docelowego.
    - server_port: str - Numer portu serwera docelowego.

Funkcja zwraca:
    - None

Używane funkcje:
    * str() - Konwersja wiadomości do formatu string.
    * udp_socket.sendto() - Wysyłanie wiadomości do serwera.
    * print() - Wyświetlenie komunikatów o błędach.

Używane zmienne:
    - msg: str - Wiadomość do przesłania (po konwersji).
    - udp_socket: socket.socket - Gniazdo do komunikacji UDP.
    - server_ip: str - Adres IP serwera gry.
    - server_port: str - Port serwera gry.

Uwagi:
    - Funkcja obsługuje wyjątki w celu zabezpieczenia przed niekontrolowanym zakończeniem działania.
    - Używa kodowania ASCII do przesyłania wiadomości.

Autor:
    Mateusz Zajda
```

```
Funkcja:
    euclideanDistance()

Przeznaczenie:
    Funkcja obliczająca odległość euklidesową pomiędzy dwoma punktami
    (landmarkami twarzy). Używana do analizy różnych aspektów mimiki i ruchów twarzy.

Argumenty funkcji:
    - pointA: NormalizedLandmark - Pierwszy punkt.
    - pointB: NormalizedLandmark - Drugi punkt.

Funkcja zwraca:
    - float - Odległość euklidesowa między punktami.

Używane funkcje:
    * math.dist() - Obliczanie dystansu pomiędzy współrzędnymi punktów.

Używane zmienne:
    - p: Tuple[float, float, float] - Współrzędne pierwszego punktu.
    - q: Tuple[float, float, float] - Współrzędne drugiego punktu.
    - distance: float - Wynik obliczeń.

Uwagi:
    - Funkcja wykorzystywana w analizie mimiki oraz ruchów głowy.

Autor:
    Piotr Mistarz
```

```
Funkcja:
    detect_smile_and_open_mouth()

Przeznaczenie:
    Wykrywa uśmiech oraz otwarcie ust na podstawie landmarków twarzy.

Argumenty funkcji:
    - landmarks: List[NormalizedLandmark] - Lista punktów twarzy.

Funkcja zwraca:
    - bool - Czy usta są otwarte.
    - bool - Czy osoba się uśmiecha.

Używane funkcje:
    * math.hypot() - Obliczanie dystansu pomiędzy landmarkami ust i twarzy.

Używane zmienne:
    - top_lip, bottom_lip: NormalizedLandmark - Punkty dla analizy otwarcia ust.
    - left_mouth, right_mouth: NormalizedLandmark - Punkty kącików ust.
    - left_cheek, right_cheek: NormalizedLandmark - Punkty policzków dla analizy szerokości twarzy.
    - open_dist: float - Odległość pionowa między górną a dolną wargą.
    - mouth_width: float - Szerokość ust.
    - face_width: float - Szerokość twarzy.
    - smile_ratio: float - Stosunek szerokości ust do szerokości twarzy.

Uwagi:
    - Uśmiech wykrywany jest na podstawie stosunku szerokości ust do szerokości twarzy.

Autor:
    Tomasz Morawa
```

```
Funkcja:
    is_eye_closed()

Przeznaczenie:
    Określa, czy oko jest zamknięte, na podstawie dystansu między powiekami,
    normalizowanego względem szerokości oka.

Argumenty funkcji:
    - landmarks: List[NormalizedLandmark] - Lista punktów twarzy.
    - eye_indices: Dict[str, int] - Indeksy kluczowych punktów oka.

Funkcja zwraca:
    - float - Współczynnik otwarcia oka.

Używane funkcje:
    * euclideanDistance() - Obliczanie dystansu pionowego i poziomego dla oka.

Używane zmienne:
    - h1, h2, v1, v2: NormalizedLandmark - Kluczowe punkty dla danego oka.
    - horizontal_distance: float - Odległość pozioma między punktami oka.
    - vertical_distance: float - Odległość pionowa między punktami oka.
    - ratio: float - Obliczony współczynnik otwarcia oka.

Uwagi:
    - Wynik jest wykorzystywany w dalszej analizie zamykania oczu.

Autor:
    Piotr Mistarz
```

```
Funkcja:
    check_eyes_closed()

Przeznaczenie:
    Sprawdza, czy oczy zostały zamknięte na wystarczająco długi czas
    oraz analizuje ich otwarcie i aktywację.

Argumenty funkcji:
    - landmarks: List[NormalizedLandmark] - Lista punktów twarzy.

Funkcja zwraca:
    - bool - Czy oczy zostały zamknięte wystarczająco długo.
    - bool - Czy oczy zostały otwarte zbyt szybko.
    - bool - Czy nastąpiła aktywacja zamknięcia oczu.

Używane funkcje:
    * is_eye_closed() - Analiza pozycji powiek.
    * time.time() - Pobranie bieżącego czasu do analizy zamknięcia.

Używane zmienne:
    - left_eye_indices, right_eye_indices: Dict[str, int] - Indeksy punktów oczu.
    - avg_left, avg_right: float - Średnie wartości współczynnika otwarcia dla oczu.
    - current_t: float - Aktualny czas dla analizy zamknięcia.
    - eyes_closed_output, eyes_failed, activate: bool - Wyniki detekcji zamknięcia oczu.

Uwagi:
    - Funkcja przechowuje historię poprzednich zamknięć oczu do analizy.

Autor:
    Piotr Mistarz
```

```
Funkcja:
    detect_head_movement()

Przeznaczenie:
    Wykrywa ruchy głowy, sprawdzając, czy jej środek wychodzi poza określony obszar.

Argumenty funkcji:
    - landmarks: List[NormalizedLandmark] - Lista punktów twarzy.
    - center: Tuple[float, float] - Środkowa pozycja twarzy (opcjonalnie).

Funkcja zwraca:
    - Tuple[bool, bool, bool, bool], Tuple[float, float] - Ruchy głowy (lewo, prawo, góra, dół) oraz nową
pozycję środka twarzy.

Używane funkcje:
    * math.hypot() - Analiza dystansu do wykrycia ruchów głowy.

Używane zmienne:
    - face_x, face_y: float - Obliczone współrzędne środka twarzy.
    - left_face_x, right_face_x: float - Pozycja krawędzi twarzy.
    - face_width: float - Szerokość twarzy.
    - center_x, center_y: float - Centralna pozycja twarzy.
    - margin_x, margin_y: float - Marginesy dla wyznaczenia obszaru ruchów głowy.
    - left_bound, right_bound, top_bound, bottom_bound: float - Granice do wykrywania ruchów.
    - is_left, is_right, is_up, is_down: bool - Detekcja kierunku ruchu.

Uwagi:
    - Mechanizm analizy dostosowany do szerokości twarzy w celu poprawnej detekcji.

Autor:
    Tim Kusznir
```

## *Opis plików*

```
Plik:
    face.py (Główny plik skryptu)

Przeznaczenie:
    Główny plik odpowiedzialny za inicjalizację aplikacji, konfigurację modelu detekcji twarzy,
    obsługę sieci UDP oraz zarządzanie wizualizacją wyników detekcji.

Pliki wymagane:
    - face_config.json - Plik konfiguracyjny przechowujący ustawienia aplikacji.
    - face_landmarker.task - Model wykrywania twarzy pobierany automatycznie w razie braku.

Używane biblioteki:
    * mediapipe - Obsługa detekcji twarzy i landmarków.
    * OpenCV (cv2) - Przetwarzanie obrazu z kamery.
    * faceexpressions - Moduł analizy mimiki twarzy.
    * supportfunctions - Funkcje pomocnicze dla głównego kodu.
    * time - Obsługa czasu i opóźnień w kodzie.
    * json - Wczytywanie i zapis danych konfiguracyjnych.
    * os - Operacje na plikach i katalogach.
    * requests - Pobieranie modelu z Google Storage.
    * socket - Obsługa połączeń sieciowych.

Główne funkcje:
    * camera_proc() - Główna funkcja obsługująca kamerę i przetwarzanie obrazu.
    * camera_callback() - Funkcja wywoływana przez model wykrywania twarzy.
    * sf.send_msg_via_udp() - Wysyłanie wiadomości do serwera gry przez UDP.

Używane zmienne globalne:
    - SERVER_IP, SERVER_PORT: Adres i port serwera gry.
    - GROUP_ID: Identyfikator grupy użytkownika.
    - udp_socket: Obiekt gniazda UDP do komunikacji sieciowej.
    - SHOW_CAMERA: Flaga określająca, czy wyświetlać obraz kamery.
    - detection_result: Wyniki detekcji twarzy przekazane do wizualizacji.
    - center: Pozycja środkowa twarzy do analizy ruchów.

Uwagi:
    - Jeśli face_config.json nie jest dostępny, aplikacja się nie uruchomi.
    - Jeśli face_landmarker.task nie jest dostępny, zostanie pobrany z Google Storage.
    - Aplikacja korzysta z UDP do wysyłania sygnałów do gry.
    - Po zakończeniu działania aplikacji gniazdo UDP jest zamykane.

Autor:
    Mateusz Zajda
```

```
Plik:
    faceexpressions.py (Moduł analizy mimiki twarzy)

Przeznaczenie:
    Plik odpowiedzialny za analizę mimiki twarzy na podstawie landmarków wykrywanych przez model
FaceLandmarker.
    Implementuje funkcje do detekcji zamknięcia oczu, uśmiechu, otwarcia ust oraz ruchów głowy.

Pliki wymagane:
    - face_config.json - Plik konfiguracyjny zawierający wartości progowe dla analizy mimiki.

Używane biblioteki:
    * math - Operacje matematyczne, obliczanie odległości.
    * time - Obsługa czasu w analizie zamknięcia oczu.
    * json - Wczytywanie danych konfiguracyjnych.
    * os - Sprawdzanie obecności plików konfiguracyjnych.
    * typing - Definiowanie typów dla list, słowników i krotek.
    * mediapipe.tasks.python.components.containers.landmark - Obsługa punktów charakterystycznych twarzy.

Główne funkcje:
    * euclideanDistance() - Obliczanie odległości euklidesowej pomiędzy dwoma punktami.
    * is_eye_closed() - Sprawdzanie, czy oko jest zamknięte na podstawie relacji powiek.
    * check_eyes_closed() - Analiza zamknięcia oczu oraz aktywacji ich długiego zamknięcia.
    * detect_smile_and_open_mouth() - Wykrywanie uśmiechu oraz otwarcia ust na podstawie proporcji twarzy.
    * detect_head_movement() - Identyfikacja ruchów głowy na podstawie przesunięcia jej centrum.

Używane zmienne globalne:
    - CLOSED_TIME: Czas wymagany do aktywacji zamknięcia oczu.
    - CLOSED_THRESH: Maksymalna tolerowana szczelina między powiekami.
    - BUF_SIZE: Liczba zapamiętanych ostatnich stanów zamknięcia oczu.
    - MAX_BLINK_DURATION: Maksymalny czas trwania mrugnięcia.

Uwagi:
    - Jeśli face_config.json nie jest dostępny, skrypt nie zostanie uruchomiony.
    - Wszystkie funkcje wykorzystują punkty charakterystyczne twarzy do analizy mimiki.
    - Funkcja check_eyes_closed() wykorzystuje mechanizm pamięciowy do wykrywania prawidłowych zamknięć oczu.
    - Detekcja uśmiechu oraz otwartych ust bazuje na analizie proporcji szerokości ust do szerokości twarzy.
    - Identyfikacja ruchów głowy opiera się na przesunięciu centrum twarzy poza określone granice.

Autorzy:
    Tim Kusznir, Piotr Mistarz, Tomasz Morawa
```

```
Plik:
    supportfunctions.py (Moduł funkcji pomocniczych)

Przeznaczenie:
    Plik zawiera funkcje wspierające główny skrypt aplikacji.
    Odpowiada za wizualizację wyników detekcji twarzy oraz komunikację sieciową poprzez UDP.

Pliki wymagane:
    - Brak dodatkowych plików wymaganych bezpośrednio przez ten moduł.

Używane biblioteki:
    * mediapipe - Obsługa detekcji twarzy i landmarków.
    * OpenCV (cv2) - Przetwarzanie obrazu z kamery.
    * numpy - Obsługa operacji na macierzach obrazu.
    * socket - Obsługa połączeń sieciowych.

Główne funkcje:
    * draw_landmarks_on_image() - Rysowanie punktów charakterystycznych twarzy na obrazie.
    * send_msg_via_udp() - Wysyłanie wiadomości do serwera gry przez UDP.

Używane zmienne globalne:
    - Brak zmiennych globalnych, funkcje operują na przekazanych argumentach.

Uwagi:
    - draw_landmarks_on_image() wykorzystuje narzędzia MediaPipe do wizualizacji punktów twarzy.
    - send_msg_via_udp() wysyła komunikaty przez UDP, obsługując ewentualne wyjątki.
    - Moduł jest używany przez face.py do przetwarzania i wysyłania danych.

Autor:
    Mateusz Zajda
```

```
Plik:
    face_config.json (Plik konfiguracyjny aplikacji)
```

```
Przeznaczenie:
    Plik zawiera ustawienia aplikacji związane z wykrywaniem twarzy,
    analizą mimiki oraz komunikacją z serwerem gry przez UDP.

Używane zmienne konfiguracyjne:
    - SERVER_IP: str - Adres IP serwera gry.
    - SERVER_PORT: int - Numer portu serwera gry.
    - SHOW_CAMERA: int - Flaga określająca, czy wyświetlać obraz kamery (1 - tak, 0 - nie).
    - GROUP_ID: int - Identyfikator grupy użytkownika.
    - BOOLEAN_MSG: int - Określa typ komunikacji z serwerem (1 - wiadomości logiczne, 0 - numeryczne).
    - CLOSED_EYES_TIME: int - Minimalny czas zamknięcia oczu wymagany do aktywacji akcji.
    - EYE_CHARGING: int - Kod sygnału dla zamkniętych oczu podczas ładowania akcji.
    - EYE_FAILED: int - Kod sygnału dla zbyt szybkiego otwarcia oczu.
    - EYE_ACTIVATION: int - Kod sygnału dla aktywacji akcji związanej z oczami.
    - MOUTH_OPENED: int - Kod sygnału dla otwartych ust.
    - SMILE: int - Kod sygnału dla wykrytego uśmiechu.
    - IS_RIGHT: int - Kod sygnału dla ruchu głowy w prawo.
    - IS_LEFT: int - Kod sygnału dla ruchu głowy w lewo.
    - IS_UP: int - Kod sygnału dla ruchu głowy w górę.
    - IS_DOWN: int - Kod sygnału dla ruchu głowy w dół.

Uwagi:
    - Jeśli plik face_config.json nie jest dostępny, aplikacja nie może się uruchomić.
    - Wartości w pliku są wykorzystywane przez funkcje analizy mimiki twarzy.
    - Kod sygnału dla każdej akcji jest mapowany w funkcji camera_callback().

Autor:
    Mateusz Zajda
```