RAG-2

Rzeszów University of Technology Games for Artificial Intelligence 2.0

Tworzenie i łączenie zewnętrznego źródła sterowania (modeli sztucznej inteligencji)

Marcin Bator 173592

Paweł Buczek 173599

Bartłomiej Krówka 173650

Opiekun: **dr inż. Dawid Kalandyk**

Politechnika Rzeszowska 2025

Tworzenie i łączenie zewnętrznego źródła sterowania (modeli sztucznej inteligencji)

Wstęp

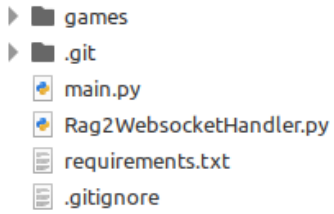
Instrukcja opisuje proces zintegrowania aplikacji dostarczającej sterowanie do gry z systemem RUT-AI Games 2.0. Aplikacja będzie działać lokalnie na komputerze użytkownika i będzie komunikować się z systemem za pomocą protokołu websocket. Będzie ona implementowała algorytm sterujący jedną z paletek w grze Pong na podstawie danych otrzymywanych w czasie rzeczywistym z systemu.

Klonowanie repozytorium

Należy sklonować lub pobrać repozytorium z kodem źródłowym aplikacji dostarczającej sterowanie. Jest ono dostępne pod adresem: `https://github.com/KN-GEST-ongit/rag-2-ai/tree/quick-start`. Należy użyć gałęzi `quick-start` (domyślnie na nią prowadzi powyższy link). Jeśli po sklonowaniu repozytorium nie znajduje się na gałęzi `quick-start`, należy ją ustawić za pomocą komendy:

```
git checkout quick-start
```

Sklonowane repozytorium na gałęzi `quick-start` powinno mieć strukturę katalogów przedstawioną na rysunku 1.



Rysunek 1: Struktura katalogów aplikacji

Instalacja zależności i uruchomianie aplikacji

Do uruchomienia aplikacji wymagane jest zainstalowanie środowiska Python w wersji 3.9. Można je pobrać ze strony `https://www.python.org/downloads/release/python-390/`. Potrzebny będzie też menadżer pakietów Pythona (PIP), który należy zainstalować według dokumentacji dostępnej pod adresem `https://pip.pypa.io/en/stable/installation/`. Kolejnym krokiem jest zainstalowanie zależności aplikacji z użyciem komendy:

```
pip install -r requirements.txt
```

Aplikację można uruchomić za pośrednictwem interfejsu IDE jak Pycharm lub Spyder albo za pomocą komendy:

```
python main.py
```

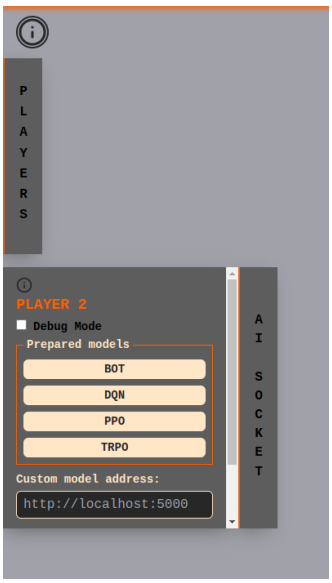
Po uruchomieniu na konsoli powinien pojawić się komunikat `Started`. Oznacza to, że aplikacja działa i jest gotowa do nawiązania połączenia z systemem RUT-AI Games 2.0.

Podłączenie modelu sterującego do RUT-AI Games 2.0

Należy otworzyć w przeglądarce witrynę `http://rutai.kia.prz.edu.pl/game/pong`. Ważne jest, aby otworzyć stronę używając protokołu niezabezpieczonego (HTTP), gdyż umożliwi to połączenie do systemu z poziomu aplikacji działającej na adresie lokalnym nieposiadającym certyfikatu SSL.

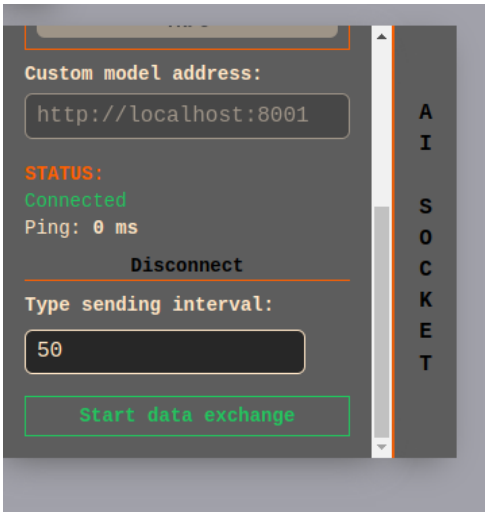
W menu `PLAYERS` po lewej stronie należy zmienić źródło sterowania jednego z graczy na `SOCKET`, jak pokazano na rysunku 2. Poniżej pojawi się menu `AI SOCKET` dla gracza `PLAYER 2` przedstawione na rysunku 3.

Rysunek 2: Wybór źródła sterowania



Rysunek 3: Menu połączenia

W menu `AI SOCKET` można wybrać i połączyć się z jednym z zaimplementowanych na serwerze modeli sterujących. Aby połączyć się z własnym modelem trzeba ręcznie podać adres lokalnego modelu uruchomionego w poprzednim kroku. W tym celu należy wpisać w pole `Custom model address` adres `http://localhost:8001/ws/pong/` i nacisnąć znajdujący się poniżej przycisk `connect`. Po udanym połączeniu status połączenia powinien zmienić się na `Connected` jak na rysunku 4. Użyty adres URL modelu zostanie zapisany w pamięci podręcznej przeglądarki i będzie dostępny przy korzystaniu z pola `Custom model address`.



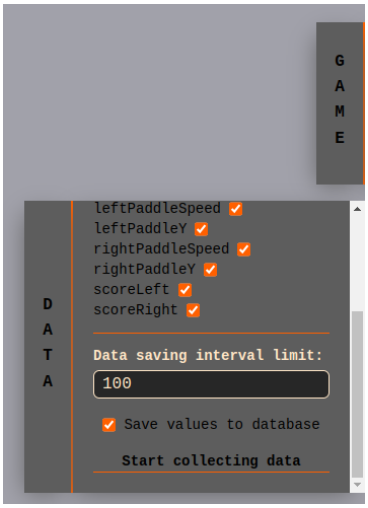
Rysunek 4: Udane połączenie z modelem

Po wykonaniu powyższych kroków model sterujący jest gotowy do działania. Pole zmiany interwału najlepiej pozostawić w wartości domyślnej (50ms). Aby rozpocząć wymianę danych z modelem należy nacisnąć zielony przycisk `Start data exchange`. Wybrana paletka powinna zacząć się poruszać. Do drugiej z nich można podłączyć inny model sterujący lub sterować nią za pomocą klawiatury (przyciski podane są w oknie pomocy w lewym górnym rogu strony z grą). Aby zatrzymać wymianę danych z modelem należy nacisnąć przycisk `Stop data exchange` bądź rozłączyć całkowicie model za pomocą przycisku `Disconnect`.

W przypadku pojawienia się komunikatu o przekroczonej liczbie połączonych gości należy utworzyć konto w systemie za pomocą adresu e-mail z domeny PRz, a następnie zalogować się na nie i ponowić próbę połączenia.

Zbieranie danych z gry

System RUT-AI Games 2.0 umożliwia zbieranie danych o stanie gry w czasie rzeczywistym podczas wymiany danych z modelem sterującym oraz alternatywnie z użyciem menu zapisu danych z wybranego etapu rozgrywki znajdujego się po prawej stronie ekranu (rys. 5). Funkcjonalność ta jest dostępna jedynie dla zalogowanych użytkowników.



Rysunek 5: Menu zbierania danych

W menu można odznaczyć dane, które nie są konieczne do zebrania. Aby rozpocząć zbieranie danych należy nacisnąć przycisk `Start collecting data`. Zakończenie ich zbierania umożliwia przycisk `Stop collecting data`. Dane zostaną zapisane w bazie danych systemu i będą dostępne w panelu użytkownika dostępnym w pasku nawigacji (dashboard). Można też od razu pobrać plik JSON, zawiera on jednak jedynie czyste dane zebrane z gry, bez dodatkowych informacji o zapisie.

Pobrany plik JSON zawiera zestaw danych o stanie gry zebranych w czasie kolejnych interwałów podczas rozgrywki. Dla każdej gry zbierane parametry różnią się, zaś ich wartości w czasie rzeczywistym można podejrzeć w oknie konsoli (rys. 6) rozwijanym na dole strony z grą (tylko dla zalogowanych). Zebrane dane można wykorzystać do analizy rozgrywki, trenowania modeli sztucznej inteligencji czy też do oceny jakości sterowania.

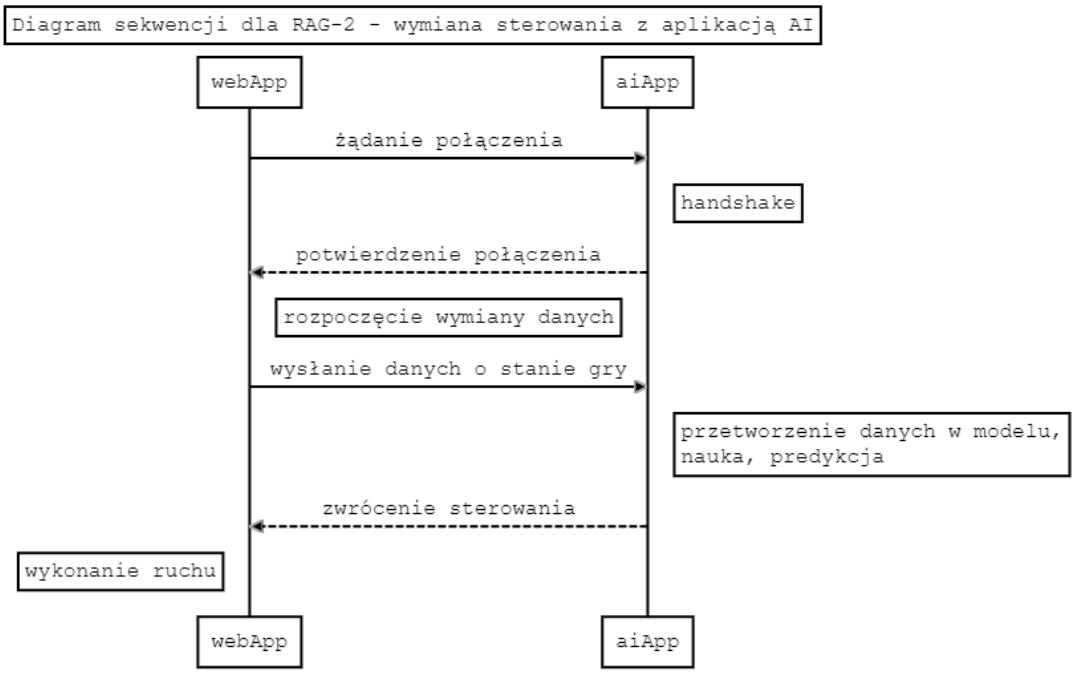
Rysunek 6: Konsola diagnostyczna

Warto zaznaczyć, że dane o stanie gry w czasie rzeczywistym są również przesyłane przez menu AI SOCKET po nawiązaniu połączenia z serwerem websocket, aby umożliwić algorytmowi bądź agentowi AI wybór odpowiedniej akcji. Omówione wyżej menu zapisu danych może okazać się pomocne przy uczeniu modeli takich jak MLP czy CNN, które wymagają danych treningowych w postaci zbioru danych, jednak nie znajdzie zastosowania przy uczeniu metodami uczenia ze wzmocnieniem. W tym przypadku agent powinien korzystać z danych przesyłanych w czasie rzeczywistym przez websocket.

Sposób wymiany danych z modelem

Po rozpoczęciu wymiany danych RUT-AI Games 2.0 wysyła do aplikacji sterowania dane w formacie JSON przedstawiające obecny stan rozgrywki. W odpowiedzi na to aplikacja odsyła dane zgodne z wymaganym przez grę schematem. Gra oczekuje od aplikacji danych, które będą zawierały informacje o wybranym przez algorytm/agenta ruchu w odpowiedzi na obecny stan rozgrywki. Przykładowa paczka danych wysyłana przez grę do aplikacji w każdym interwale znajduje się na listingu 1.

Diagram sekwencji (rys. 7) przedstawia schemat przepływu danych i sterowania pomiędzy postacią w grze zdefiniowaną w systemie (webApp) a uruchomioną na komputerze lokalnym aplikacją (aiApp).



Rysunek 7: Diagram sekwencji - wymiana sterowania

Listing 1: Przykładowa paczka danych wysłana przez grę

```
{
  "name": "pong",
  "playerId": 1,
  "state": {
    "leftPadleY": 120,
    "rightPadleY": 225,
    "leftPaddleSpeed": 0,
    "rightPaddleSpeed": 0,
    "ballX": 817.9999999999986,
    "ballY": 493.6666666666661,
    "ballSpeedX": 8,
    "ballSpeedY": 6.333333333333334,
    "ballSpeedMultiplier": 1.05,
    "scoreLeft": 7,
    "scoreRight": 3
  },
  "players": [
    {
      "inputData": { "move": 0 },
      "expectedDataDescription": "Value of {-1, 0, 1}, -1: down, 0: stop, 1: up
          ",
      "isObligatory": true,
      "name": "Player 1",
      "id": 1,
      "isActive": true,
      "playerType": "SOCKET"
    },
    {
      "inputData": { "move": 0 },
      "expectedDataDescription": "Value of {-1, 0, 1}, -1: down, 0: stop, 1: up
          ",
      "isObligatory": true,
      "name": "Player 2",
      "id": 2,
      "isActive": true,
      "playerType": "KEYBOARD"
    }
  ],
  "expectedInput": { "move": 0 }
}
```


Paczka zawiera informacje o stanie rozgrywki (`state`), informacje o graczach (`players`) oraz oczekiwane od aplikacji dane sterowania (`expected_input`). Dane wysyłane przez aplikację powinny być zgodne z formatem oczekiwanym przez system. Słowny opis danych oczekiwanych znajduje się w polu `players.expectedDataDescription`.

Przykładowa paczka danych oczekiwana przez grę znajduje się na listingu 2. W tym przypadku aplikacja zwraca wartość 0, co oznacza brak ruchu paletką. Zgodnie ze specyfikacją `expectedDataDescription` wartość -1 oznacza ruch w dół, 0 - brak ruchu, a 1 - ruch w górę.

Listing 2: Przykładowa paczka danych sterowania oczekiwana przez grę


```
{ "move": 0 }
```


Omówienie kodu źródłowego aplikacji websocket

Plik `./main.py` (listing 3) zawiera kod źródłowy aplikacji, która obsługuje połączenie websocket. W linii 8 tworzony jest obiekt aplikacji tornado, który nasłuchuje na porcie 8001. W linii 9 dodawany

jest handler obsługujący połączenie websocket pod adresem `http://localhost:8001/ws/pong/`. W linii 11 uruchamiana jest aplikacja, a w linii 14 uruchamiana jest pętla główna aplikacji.

Listing 3: Plik ./main.py

```
1  import tornado.ioloop
2  import tornado.web
3  import tornado.websocket
4
5  from games.pong import PongWebSocketHandler
6
7
8  app = tornado.web.Application([
9          (r"/ws/pong/", PongWebSocketHandler) #http://localhost:8001/ws/pong
10     ])
11 app.listen(8001)
12
13 print("Started")
14 tornado.ioloop.IOLoop.current().start()
```

Konstruktor obiektu `app` (linia 9) jest miejscem, w którym należy zarejestrować `handlery`, czyli klasy, które będą przetwarzały dane wysłane przez klienta websocket na podanym w tej linii adresie. W tym przypadku jest to klasa `PongWebSocketHandler` zdefiniowana w pliku `./games/pong.py`. Tak zdefiniowany kod utworzy serwer websocket, który będzie nasłuchiwał na adresie `http://localhost:8001/ws/pong/`, zaś logikę obsługi sterowania przekaże do klasy `PongWebSocketHandler`, której kod znajduje się na listingu 4. Takich handlerów w aplikacji na tym samym porcie może być więcej, jednak muszą mieć one różne adresy URL.

Plik `./games/pong.py` (listing 4) jest handlerem obsługującym połączenie websocket. Dziedziczy on po klasie `Rag2WebsocketHandler`, która jest klasą bazową dla handlerów websocket w systemie RUT-AI Games 2.0. W linii 5 definiowana jest metoda `send_data`, która jest wywoływana za każdym razem, gdy serwis wysyła dane o stanie gry i oczekuje na dane sterujące od aplikacji. W metodzie tej powinna znajdować się logika parsowania danych otrzymanych od aplikacji, wybranie odpowiedniego ruchu (na przykład za pomocą algorytmu, losowo bądź za pomocą systemu ekspertowego czy agenta) i wysłanie go w formacie JSON za pomocą metody `self.write_message`.

Listing 4: Plik ./games/pong.py

```
1  import json
2
3  from Rag2WebsocketHandler import Rag2WebsocketHandler
4
5  class PongBot(Rag2WebsocketHandler):
6      def send_data(self, data):
7          if data['playerId'] == 0:
8              if data['state']['ballY'] < data['state']['leftPaddleY'] + 50:
9                  move = 1
10             else:
11                 move = -1
12         else:
13             if data['state']['ballY'] < data['state']['rightPaddleY'] + 50:
14                 move = 1
15             else:
16                 move = -1
17         self.write_message(json.dumps({'move': move}))
```

Parametr `data` dostępny w metodzie to paczka danych przysłana przez grę, taka jak ta pokazana na listingu 1. Algorytm zawarty w liniach od 7 do 16 jest przykładowym algorytmem sterującym, który wybiera ruch paletką w zależności od położenia piłki. W tym przypadku algorytm wybiera ruch w górę, jeśli piłka jest wyżej niż środek paletki, w przeciwnym wypadku wybiera ruch w dół. Warto zauważyć, że algorytm ten działa jedynie dla gracza o `playerId` równym 0, czyli dla gracza po lewej stronie ekranu.

Dla gracza po prawej stronie algorytm działa analogicznie, jednak z uwzględnieniem położenia paletki po prawej stronie ekranu.

Na końcu, w linii 17 ma miejsce stworzenie obiektu JSON o strukturze zgodnej z tą oczekiwaną przez grę (listing 2). Kiedy serwis otrzyma dane o wymaganym formacie, postać w grze (paletka) wykona ruch zlecony przez powyższy algorytm. Warto zauważyć, że w przypadku bardziej skomplikowanych algorytmów sterujących, takich jak te wykorzystujące sieci neuronowe, metoda `send_data` może być bardziej rozbudowana i skomplikowana, a nawet może używać innych klas i metod.

Metoda `send_data` może też wyglądać o wiele prościej. Na listingu 5 został przedstawiony kod metody wysyłający do gry pong losowy ruch ze zbioru {-1, 0, 1}.

Listing 5: Przykładowa metoda send_data

```
1 def send_data(self, data):
2     move = random.choice([-1, 0, 1])
3     self.write_message(json.dumps({'move': move}))
```

Przykłady implementacji aplikacji sterujących

1. Aplikacja przetwarzająca gesty z kamery internetowej na ruchy postaci w grach: `https://github.com/marcinbator/rag-2-gestures`
2. Mikroserwis zaimplementowany na serwerze RUT-AI Games 2.0 z modelami wyświetlanymi w menu `AI SOCKET`: `https://github.com/KN-GEST-ongit/rag-2-ai`