Damian Kolaska
Jan Prugarewicz
Łukasz Tumialis
Dawid Wysocki

Warszawa, 2023

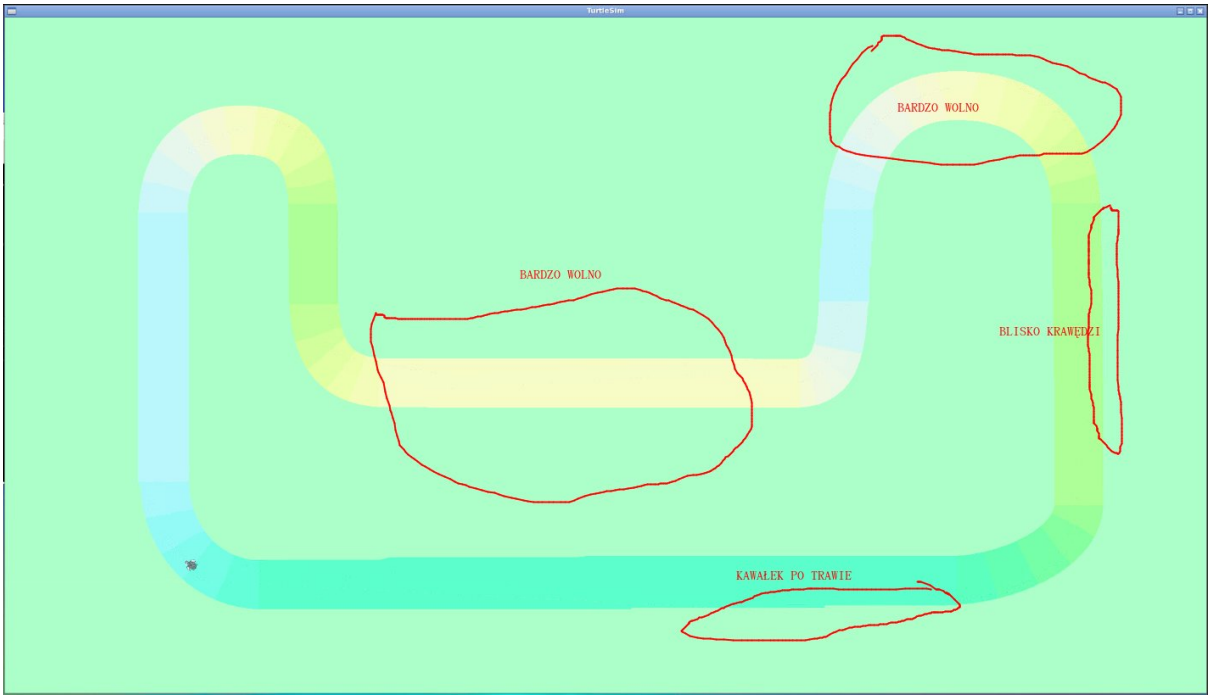
SIU

2. etap projektu

Proces treningu agenta

Trening wstępny

Na samym początku postanowiliśmy uruchomić trening dla domyślnych parametrów uczenia. Agent już po 450 epokach był niemal w stanie w pełni pokonać trasę. Obserwację z testowego przejazdu agenta przedstawia *Rysunek 1.* Przede wszystkich zauważyliśmy, że agent w pewnych miejscach miał tendencję do zbytniego zbliżania się do krawędzi toru. Poza tym niektóre części pokonywał w niezrozumiale powolnym tempie. Wytrenowany model bez zmian został zapisany jako dqns_model_X.tf, gdzie X oznacza ilość epok.



Rysunek 1. Obserwacje z przejazdu agenta wytrenowanego przez 450 epok przy domyślnych parametrach uczenia.

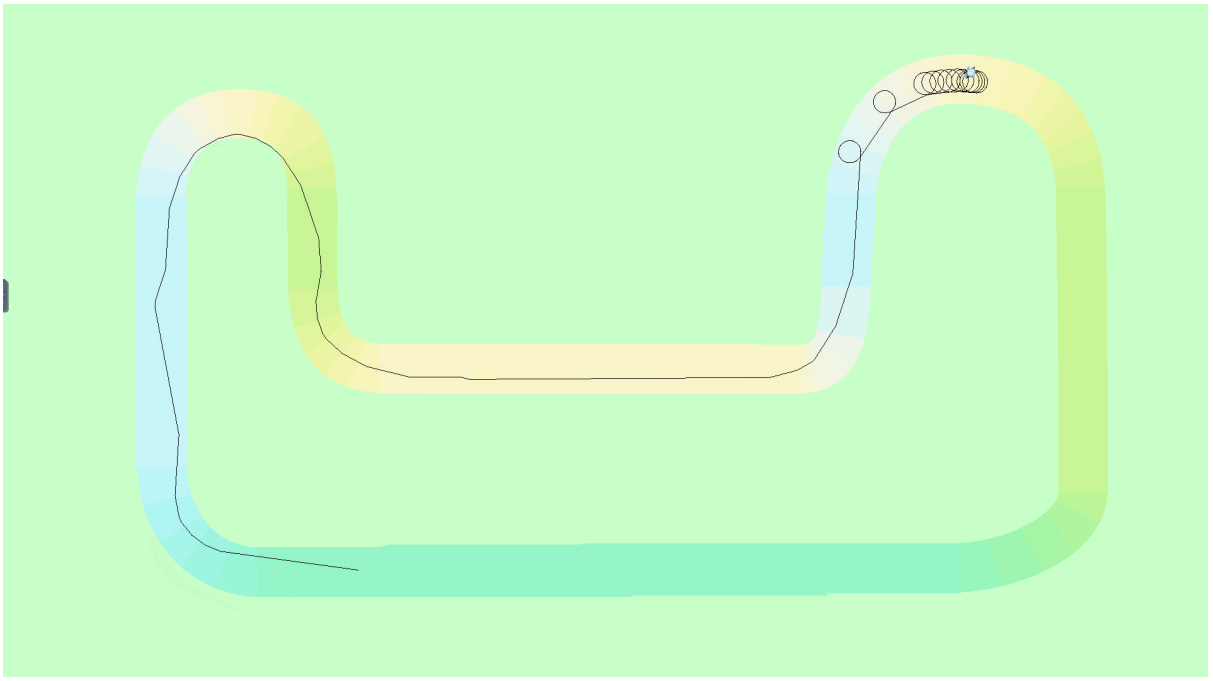
Zmiana parametrów środowiska

W dalszym kroku uruchomiliśmy kolejny trening. Skorzystaliśmy z wcześniej wytrenowanego modelu. Postanowiliśmy jednak zmienić niektóre parametry środowiska:

- Zwiększyliśmy nagrodę za jazdę w poprawnym kierunku z 0.5 do 6.0 (SPEED_RWRD_RATE).
- Zwiększyliśmy karę z jazdę pod prąd z -10 do -14 (SPEED_RVRS_RATE).
- Zwiększyliśmy karą za wypadnięcie z trasy z -10 do -15 (OUT_OF_TRACK_FINE).

Powyższe zmiany mają na celu przede wszystkich jeszcze bardziej zachęcić agenta do jazdy w odpowiednim kierunku oraz zniechęcić przed zbaczaniem z trasy.

Szczególnie trudny dla agenta jest zakręt w prawym górnym rogu planszy. Zdarzało się, że agent próbując pokonać zakręt wykonywał, tzw. bączki. Efekt ten prezentuje *Rysunek 2.* Spekulujemy, że wynika to z dużego kąta pomiędzy kierunkiem startowym, a położeniem celu.



Rysunek 2. Trudności z pokonywaniem zakrętu w prawym górnym rogu trasy.

Trening trwał przez 6000 epok jednak już po 1500. epoce otrzymaliśmy agenta, który z łatwością pokonywał zadany tor. Modele po zmianie parametrów środowiska zapisane są jako dqns2_model_X.tf, gdzie X oznacza liczbę epok.

Zmiana parametrów sieci

Do modyfikacji parametrów sieci podeszliśmy z następującym rozumowaniem:

- Agent nie miał problemów z wyszukaniem przyzwoitej polityki akcji, dlatego też modyfikacja parametru epsilon raczej nie jest konieczna.
- Podobnie częstsza aktualizacja wag modelu nie przyniesie korzyści.
- Proces uczenia był stabilny dlatego nie jest konieczne zwiększenie parametru *MINIBATCH_SIZE*.

Jedyną zmianą jaka według nas mogłaby przynieść jakiekolwiek korzyści jest modyfikacja architektury sieci neuronowej. Trudno jest jednak stwierdzić dokładnie jakiej zmiany należałoby dokonać. Dlatego postanowiliśmy eksperymentalnie rozszerzyć sieć o kolejną warstwę splotową.

Architekturę sieci po dodaniu warstwy splotowej przedstawia *Rysunek 3.*

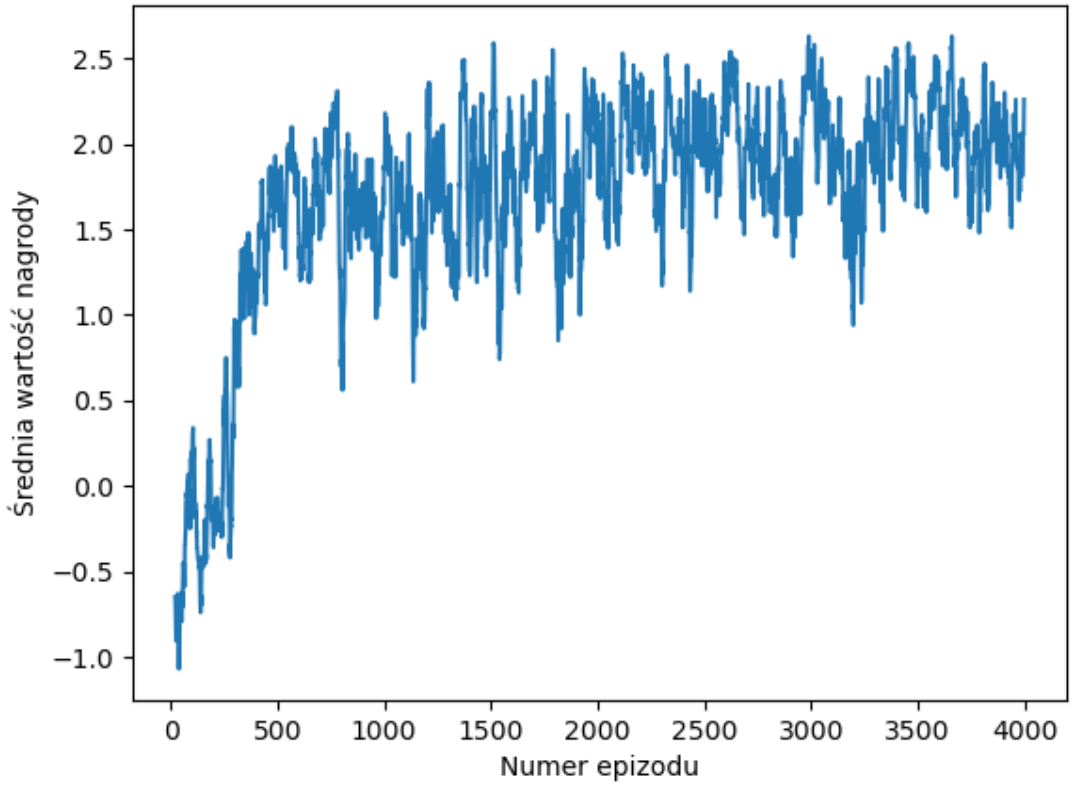
```
Model: "sequential"
_________________________________________________________________
 Layer (type)                Output Shape              Param #
=================================================================
 conv3d (Conv3D)             (None, 4, 4, 1, 16)       528

 permute (Permute)           (None, 4, 4, 16, 1)       0

 conv3d_1 (Conv3D)           (None, 3, 3, 1, 16)       1040

 permute_1 (Permute)         (None, 3, 3, 16, 1)       0

 conv3d_2 (Conv3D)           (None, 2, 2, 1, 16)       1040

 permute_2 (Permute)         (None, 2, 2, 16, 1)       0

 conv3d_3 (Conv3D)           (None, 1, 1, 1, 16)       1040

 flatten (Flatten)           (None, 16)                0

 dense (Dense)               (None, 32)                544

 dense_1 (Dense)             (None, 6)                 198

=================================================================
Total params: 4,390
Trainable params: 4,390
Non-trainable params: 0
_________________________________________________________________
```

Rysunek 3. Architektura sieci po dodaniu warstwy splotowej.

Trening modelu z dodatkową warstwą splotową

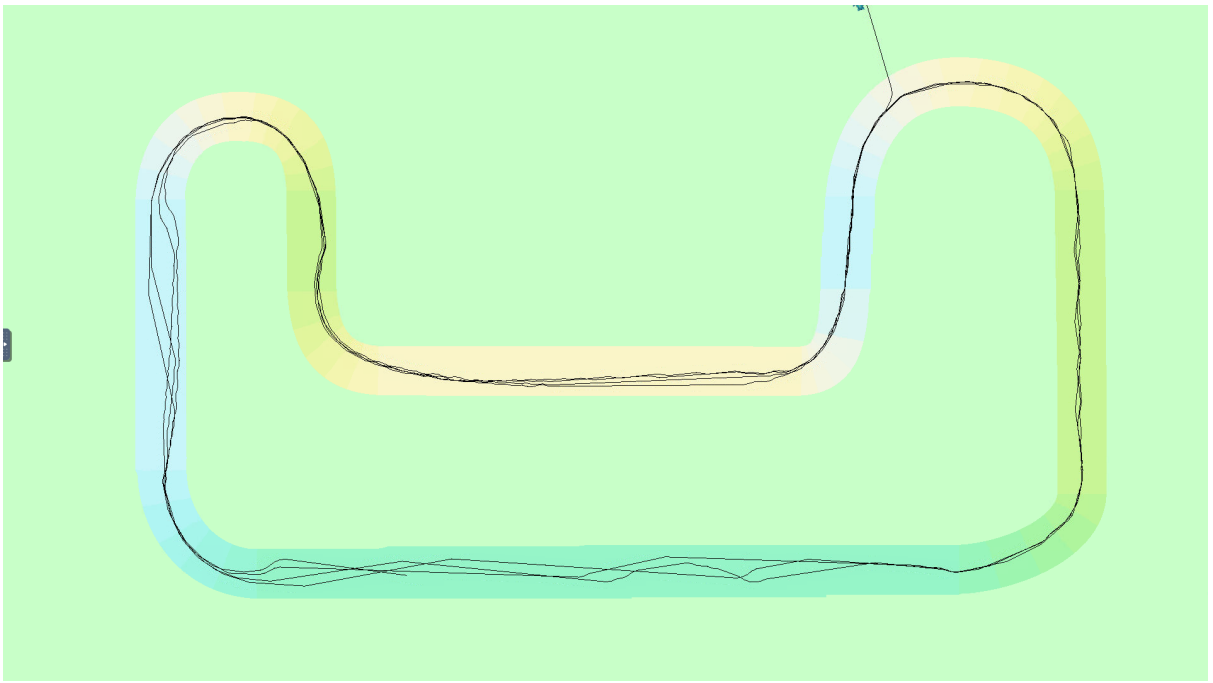
Wykres nagrody dla uczenia modelu z dodatkową warstwą splotową przedstawia *Rysunek 4.* Widzimy, że model osiąga szczytową nagrodą po ok. 1500 epizodach. Pokrywa się to z naszymi obserwacjami. Model wytrenowany na 500 epizodach był w stanie pokonać 2.5 okrążenia zanim zjechał z trasy. Z kolei po 1500 epizodach było to już 5.5 okrążenia. Model po 3950 okrążeniach był w stanie pokonać jedynie 1.5 okrążenia.

Damian Kolaska
Jan Prugarewicz
Łukasz Tumialis
Dawid Wysocki



Rysunek 4. Wykres nagrody dla procesu uczenia modelu (po dodaniu warstwy splotowej).

Zarejestrowany scenariusz uruchomienia agenta



Rysunek 5. Zarejestrowany przebieg scenariusza uruchomienia agenta. Czarną linią

zaznaczono kolejne kroki które wykonał agent w czasie ruchu.

Trasę przykładowego, w pełni pokonanego przejazdu agenta przedstawia *Rysunek 5.* Widzimy, że agent ani razu nie wyjechał poza granicę toru. Momentami ruch agenta jest niestabilny, jednak wciąż oceniamy ostateczną jakość modelu jako satysfakcjonującą.

Wskaźnik jakości modelu

Ostatnim krokiem oceny jakości modelu jest wyznaczenie wskaźnika *n*:

$$n = \frac{s}{l}$$

l - liczba okrążeń
s - liczba prób

Wskaźnik został wyliczony dla *models/exp_2/dqns_conv_model_1500.tf.*
Przejazd 1. - 8 (zatrzymano z powodu zbyt długiego czasu eksperymentu).
Już pierwszy przejazd pokazał, że model z łatwością pokonuje zadaną trasę.
Zatem współczynnik n jest z pewności większy od 1.

Artefakty

Pliki *_handout.py

turtlesim_env_base_handout.py

```
167     def _load_routes(self, routes_filename: str):
168         with open(routes_filename, encoding='utf-8-sig') as f:  # załadowanie tras agentów
169             for line in f.readlines():
170                 route_id, agent_cnt, x_min, x_max, y_min, y_max, x_g, y_g = line.strip().split(";")
171                 route_section = (
172                     int(agent_cnt),
173                     float(x_min), float(x_max),
174                     float(y_min), float(y_max),
175                     float(x_g), float(y_g),
176                 )
177                 if route_id in self.routes.keys():
178                     self.routes[route_id].append(route_section)
179                 else:
180                     self.routes[route_id] = [route_section]
```

Rysunek 6. Fragment klasy TurtleSimEnvBase odpowiedzialny za załadowanie tras agentów.

```
184     def _roulette_selection(self, sections: list) -> int:
185         max = sum([section[0] for section in sections])
186         selection_probs = [section[0]/max for section in sections]
187         return np.random.choice(len(sections), p=selection_probs)
```

Rysunek 7. Fragment klasy odpowiedzialny za losowanie pozycji żółwi. Jest to implementacja algorytmu selekcji ruletkowej, gdzie poszczególne sekcje toru są losowane proporcjonalnie do planowanej liczby żółwi w sekcji.

turtlesim_env_single_handout.py

```
20            # TODO STUDENCI przejechać 1/2 okresu, skręcić, przejechać pozostałą 1/2
21            if realtime:                                    # jazda+skręt+jazda+skręt
22                twist = Twist()
23                twist.linear.x = action[0]
24                twist.linear.y = 0
25                twist.angular.z = action[1]
26                self.tapi.setVel(tname, twist)
27                rospy.sleep(.5 * self.SEC_PER_STEP)
28                self.tapi.setVel (tname, twist)
29                rospy.sleep(.5 * self.SEC_PER_STEP)
```

Rysunek 8. Algorytm pokonywania zakrętu.

dqn_single_handout.py

```
144         def _save_model(self, episode: int):
145             self.model.save(f"./models/{self.id_prefix}_model_{episode}.tf")
```

Rysunek 9. Funkcja odpowiedzialna za zapisywanie modelu w formacie .tf.

Obraz dockera z wytrenowanym modelem

https://hub.docker.com/r/tortillazhawaii/siu_etap1

Repozytorium GitHuba

https://github.com/GRO4T/SIU_autonomous_driving_agent

Skrypt testowy

W katalogu /root/ znajduje się skrypt test_model.py. Pozwala on na przetestowanie działania wybranej wersji modelu.

Przykład użycia
python3 test_model.py models/exp_2/dqns_conv_model_1500.tf