

Sprawozdanie: Symulator Ruchu Drogowego z Uczeniem Maszynowym

Autorzy: Adam Wojs, Krystian Grzegorzewicz

Przedmiot: Podstawy Sztucznej Inteligencji

Spis Treści

- . [Określenie tematu i celu projektu](#)
 - . [Zbiór danych i ich przygotowanie](#)
 - . [Wybór i implementacja modelu AI](#)
 - . [Ocena wyników modelu i optymalizacja](#)
 - . [Wdrożenie modelu i monitorowanie](#)
 - . [Podsumowanie i wnioski](#)
-

1. Określenie tematu i celu projektu

1.1 Cel projektu

Celem projektu było stworzenie symulatora ruchu drogowego z wykorzystaniem uczenia maszynowego (Reinforcement Learning) do sterowania sygnalizacją świetlną. Agent AI miał nauczyć się optymalnie przełączać światła na skrzyżowaniach, aby maksymalizować przepustowość (throughput) i minimalizować czas oczekiwania pojazdów.

1.2 Zakres projektu

Projekt obejmował:

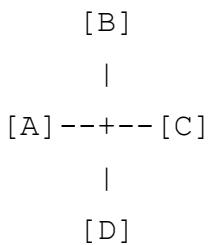
- Symulator ruchu w Pygame z wieloma poziomami skrzyżowań
- Implementację algorytmu Deep Q-Network (DQN) z PyTorch
- System benchmarkowania do porównania AI z metodami bazowymi
- Interfejs demo umożliwiający obserwację działania modelu w czasie rzeczywistym

1.3 Wymagania funkcjonalne

Wymaganie	Opis
Multi-level	System musi obsługiwać 3 różne konfiguracje skrzyżowań
Sterowanie sygnalizacją	AI kontroluje wszystkie światła (główne + strzałki)
Wizualizacja	Użytkownik może obserwować działanie AI w czasie rzeczywistym
Porównywalność	System benchmarkowy do porównania z metodami Random/Fixed
Skalowalne spawnowanie	Suwaki do kontroli natężenia ruchu

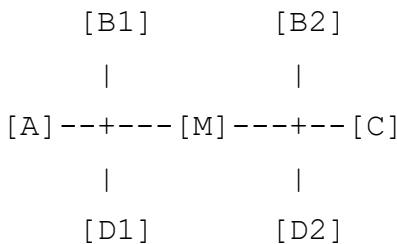
1.4 Architektura poziomów

Poziom 1 - Pojedyncze skrzyżowanie:



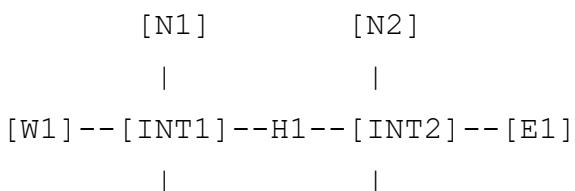
- 4 punkty wjazdowe (A, B, C, D)
- 12 bazowych konfiguracji świateł × 4 opcje czasu = **48 akcji**
- Rozmiar stanu: 16 wartości

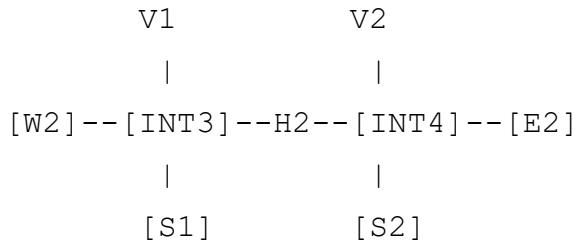
Poziom 2 - Dwa połączone skrzyżowania:



- 6 punktów wjazdowych
- Segment łączący [M] z ograniczoną pojemnością
- 8 bazowych konfiguracji × 4 opcje czasu = **32 akcje**
- Rozmiar stanu: 30 wartości

Poziom 3 - Siatka 2x2 (4 skrzyżowania):





- 8 punktów wjazdowych
- 4 segmenty wewnętrzne (H1, H2, V1, V2)
- 16 bazowych konfiguracji × 4 opcje czasu = **64 akcje**
- Rozmiar stanu: 56 wartości

Struktura akcji:

`action_id = base_action × 4 + duration_idx`

Opcje czasu (duration_idx):

0 = 5 sekund
 1 = 10 sekund
 2 = 15 sekund
 3 = 20 sekund

2. Zbiór danych i ich przygotowanie

2.1 Specyfika Reinforcement Learning

W przeciwieństwie do tradycyjnego uczenia maszynowego, w Reinforcement Learning dane nie są zbierane z góry - są generowane podczas interakcji agenta ze środowiskiem. Agent:

- Obserwuje stan środowiska (wektor stanu)
- Wybiera akcję
- Otrzymuje nagrodę
- Obserwuje nowy stan

2.2 Wektor stanu (State Vector)

Wektor stanu przekazywany do sieci neuronowej zawiera:

Format: [queues] + [throughput] + [signals] + [arrows]

Dla Poziomu 1 (20 wartości):

- [0-7] Queue pressures - liczba pojazdów w każdym pasie (znormalizowane 0-1)
- [8-11] Throughput - przepustowość każdego kierunku

[12-15] Signal status - stan świateł (1.0=zielone, 0.0=czerwone)

[16-19] Turn arrows - stan strzałek skrętu

Przykładowy wektor (z rzeczywistego testu):

```
--- QUEUE PRESSURES (indices 0-7) ---  
[ 0] A_Lane_0 : 0.100 (raw: ~2 vehicles)  
[ 1] A_Lane_1 : 0.000 (raw: ~0 vehicles)  
[ 2] B_Lane_0 : 0.150 (raw: ~3 vehicles)  
[ 3] B_Lane_1 : 0.050 (raw: ~1 vehicles)  
[ 4] C_Lane_0 : 0.000 (raw: ~0 vehicles)  
[ 5] C_Lane_1 : 0.000 (raw: ~0 vehicles)  
[ 6] D_Lane_0 : 0.100 (raw: ~2 vehicles)  
[ 7] D_Lane_1 : 0.100 (raw: ~2 vehicles)
```

```
--- THROUGHPUT (indices 8-11) ---  
[ 8] Direction A: 0.020 (raw: ~2 vehicles)  
[ 9] Direction B: 0.000 (raw: ~0 vehicles)  
[10] Direction C: 0.030 (raw: ~3 vehicles)  
[11] Direction D: 0.000 (raw: ~0 vehicles)
```

```
--- SIGNAL STATUS (indices 12-15) ---  
[12] Direction A: 1.0 (GREEN)  
[13] Direction B: 0.0 (RED)  
[14] Direction C: 1.0 (GREEN)  
[15] Direction D: 0.0 (RED)
```

2.3 Problem: Wektor stanu pokazywał same zera

Symptom: Podczas inspekcji wektora stanu wszystkie wartości wynosiły 0:

Running simulation for 10 seconds...

State Vector Size: 20

```
--- QUEUE PRESSURES (indices 0-7) ---  
[ 0] A_Lane_0 : 0.000 (raw: ~0 vehicles)  
[ 1] A_Lane_1 : 0.000 (raw: ~0 vehicles)  
...wszystkie zera...
```

```
--- METRICS ---
```

```
total_throughput: 0
waiting_vehicles: 0    total_vehicles:
0
```

Przyczyna: Inspekcja stanu odbywała się zbyt szybko - wątek spawner nie zdążył wygenerować pojazdów.

Rozwiążanie: Dodano rzeczywiste opóźnienie i feedback w trakcie oczekiwania:

```
for i in range(50):    # 5 seconds
    time.sleep(0.1)        if i % 10 == 0:
        metrics = simulator.get_metrics()           print(f'{i/10:.1f}s - Vehicles: {len(simulator.vehicles)} , '
        f'Throughput: {metrics.get("total_throughput", 0)}")
```

3. Wybór i implementacja modelu AI

3.1 Architektura sieci neuronowej (DQN)

Wybrano architekturę Deep Q-Network z następującymi warstwami:

```
class UnifiedDQN(nn.Module):
    def __init__(self, state_size=60, action_size=16,
hidden_size=256):
        self.net = nn.Sequential(
            nn.Linear(state_size, hidden_size),
            # 60 -> 256
            nn.ReLU(),
            nn.LayerNorm(hidden_size),
            nn.Dropout(0.1),

            nn.Linear(hidden_size, hidden_size),
            # 256 -> 256
            nn.ReLU(),
            nn.LayerNorm(hidden_size),
            nn.Dropout(0.1),

            nn.Linear(hidden_size, hidden_size // 2),
            # 256 -> 128
            nn.ReLU(),

            nn.Linear(hidden_size // 2, action_size)
            # 128 -> 16
        )
```

Kluczowe elementy architektury:

- LayerNorm - normalizacja warstw dla stabilności treningu
- Dropout (0.1) - regularyzacja zapobiegająca przeuczeniu
- Unified model - jeden model dla wszystkich poziomów (padding dla mniejszych)

3.2 Hiperparametry

Parametr	Wartość początkowa	Wartość finalna	Uzasadnienie zmiany
LEARNING_RATE	0.0005	0.001	Przyspieszenie uczenia
BATCH_SIZE	64	128	Stabilniejsze gradienty
MEMORY_SIZE	50,000	100,000	Większa różnorodność
EPSILON_END	0.05	0.01	Mniejsza eksploracja końcowa
EPSILON_DECAY	0.997	0.9975	Wolniejszy spadek
Episodes	300	1500	Więcej czasu na naukę

3.3 Funkcja nagrody - ewolucja

Wersja 1 (problematyczna):

```
def compute_reward(simulator, prev_metrics, action,
prev_action):
    reward = 0.0
    reward += wait_delta * 0.1           # Redukcja czasu
oczekiwania
    reward += throughput_delta * 0.5      # Przepustowość
    reward += queue_delta * 0.05          # Redukcja kolejek
    if action != prev_action:
        reward -= 0.1                   # Kara za zmianę
    if waiting > 10:
        reward -= waiting * 0.02         # Kara za zatłoczenie
    return np.clip(reward, -5, 5)
```

Problem: AI nauczyło się unikać kar zamiast maksymalizować przepustowość.

Wersja 2 (poprawiona):

```
def compute_reward(simulator, prev_metrics, action,
prev_action):
    reward = 0.0
    # Główny cel - przepustowość (6x silniejsza niż wcześniej)
    reward += throughput_delta * 3.0

    # Bonus za jakkolwiek przepustowość
    if throughput_delta > 0:
        reward += 0.5

    # Drugorzędne - redukcja oczekujących
    reward += waiting_delta * 0.3

    # Minimalna kara za zmianę sygnału
    if action != prev_action:
        reward -= 0.02 # 5x mniejsza kara

    # Łagodna kara za ekstremalne zatłoczenie
    if waiting_now > 20:
        reward -= (waiting_now - 20) * 0.01

    return np.clip(reward, -10, 10)
```

3.4 Problem: Trening był bardzo wolny

Symptom: 70 epizodów zajmowało bardzo długo:

Ep 70 | L2 | Reward: -0.20 | Avg L1:-45.9 L2: -2.0 L3: -2.2 | Eps: 0.808

Przyczyna: Symulator używał `time.sleep()` i wątków tła nawet w trybie headless.

Rozwiążanie: Wprowadzono tryb headless bez opóźnień:

```
class TrafficSimulator:
    def __init__(self, level=1, headless=False):
        self.headless = headless
        if not headless:
            # Wątki tylko dla trybu wizualnego

        threading.Thread(target=self._vehicle_spawner).start()

        threading.Thread(target=self._signal_controller).start()
```

Wynik: Przyspieszenie z ~10s/epizod do ~0.5s/epizod (20x szybciej).

3.5 Problem: Poziom 1 zawsze miał ujemny wynik

Symptom:

Ep 240/300 | L1 | R:-92.92 | Avg L1:-99.4 L2: 8.2 L3: 8.3

Przyczyna: Funkcja `_get_signal_for_vehicle()` używała różnych źródeł sygnałów dla różnych poziomów:

```
# Błędny kod - Level 1 używał self.signals zamiast
# current_level.intersections
if self.current_level_num == 1:
    return self.signals[direction] # Stary system - nie był
#aktualizowany!
```

Rozwiążanie: Ujednolicenie źródła sygnałów:

```
def _get_signal_for_vehicle(self, vehicle):
    # Zawsze używaj current_level.intersections
    if self.current_level and 'main' in
self.current_level.intersections:
        return
    self.current_level.intersections['main'].signals[direction]
```

Wynik po naprawie:

Ep 290/300 | L3 | R: 6.60 | Avg L1: 7.2 L2: 8.7 L3: 6.5

3.6 Problem: Pojazdy nie spawnowały się na Poziomie 3

Symptom: Po przełączeniu na Poziom 3, żadne pojazdy się nie pojawiały.

Diagnoza: Funkcja `is_off_screen()` natychmiast usuwała nowo utworzone pojazdy:

```
def is_off_screen(self):
    # Dla Level 3 - pozycje startowe były poza standardowymi
    # granicami
    return (self.x < -50 or self.x > SCREEN_WIDTH + 50 or
            self.y < -50 or self.y > SCREEN_HEIGHT + 50)
```

Rozwiążanie: Dodano flagę `was_on_screen`:

```

def is_off_screen(self):
    if not self.was_on_screen:
        # Pojazd musi najpierw pojawić się na ekranie
        if 0 <= self.x <= SCREEN_WIDTH and 0 <= self.y <=
SCREEN_HEIGHT:
            self.was_on_screen = True
    return False
return (self.x < -50 or self.x > SCREEN_WIDTH + 50 or ...)

```

4. Ocena wyników modelu i optymalizacja

4.1 System benchmarkowania

Stworzono system porównawczy testujący 3 strategie sterowania:

Strategia	Opis
AI	Wytrenowany model DQN
Random	Losowy wybór akcji co 0.75s
Fixed	Cykliczne przełączanie faz co ~120 klatek

4.2 Problem: Rozbieżność wyników treningu i benchmarku

Symptom: Podczas treningu AI osiągało wysokie nagrody, ale benchmark pokazywał wyniki gorsze niż Random:

Trening (pozytywne wyniki):

Ep 1490/1500 | L3 | R:391.98 | Avg L1:205.7 L2:292.6 L3:348.8 **Benchmark**

(negatywne wyniki):

THROUGHPUT (higher is better) :

L2_rate0.25 AI: 13.4 Random: 57.3 Fixed: 64.2

SUMMARY - AI vs Others:

L2_rate0.25: vs random: -76.5%

Przyczyna: Środowisko treningowe różniło się od benchmarkowego:

Aspekt	Trening	Benchmark (przed)	Różnica
Klatki na decyzję	15	1	15x
Częstość spawnu	Co 3 kroki	Co 1 krok	3x
Pojazdy startowe	8	0	-

AI nauczyło się działać w "przyspieszonej" symulacji gdzie każda decyzja obejmowała 15 klatek ruchu. W benchmarku pracowało w "zwolnionym tempie".

Rozwiązanie: Dostosowanie benchmarku do warunków treningowych:

```
# Benchmark - dopasowanie do treningu
for step in range(steps):
    if step % 3 == 0: # Spawn co 3 kroki
        for _ in range(2):
            simulator._spawn_level_vehicle()

    simulator.apply_level_action(action)

    for _ in range(15): # 15 klatek na decyzję
        simulator.update()
```

4.3 Iteracja 1: Rozbudowana przestrzeń akcji (nieudana)

Pierwsza próba obejmowała rozbudowę przestrzeni akcji o kontrolę czasu trwania faz świata (4 opcje: 5s, 10s, 15s, 20s), co zwiększyło przestrzeń akcji 4-krotnie:

Poziom	Baza akcji	× 4 czas	= Łącznie
1	12	× 4	48
2	8	× 4	32
3	16	× 4	64

Wyniki były rozczarowujące - AI przegrywało w 8/9 scenariuszy, wygrywając jedynie na L2_rate0.1.

Diagnoza wskazała na zbyt dużą przestrzeń akcji jako główną przyczynę.

4.4 Iteracja 2: Uproszczona przestrzeń akcji (v3)

Na podstawie analizy wprowadzono następujące zmiany:

Aspekt	Przed (v2)	Po (v3)
Przestrzeń akcji	32-64 (z czasem)	12/8/16 (proste)
Spawn rate treningu	0.01-0.5	0.2-0.5 (ciężki ruch)
Funkcja nagrody	5 komponentów	1 komponent (throughput × 10)
Epizody	1500	3000

Wyniki treningu v3:

Ep 1490/1500 | L3 | R:2079.00 | Avg L1:1042.2 L2:1523.0 L3:1911.8

Total time: 19.9 minutes

Nagrody wzrosły ~100x w porównaniu do poprzedniej wersji.

4.5 Wyniki końcowe benchmarku (v3)

Przepustowość (THROUGHPUT) - wyższa = lepsza:

Konfiguracja	AI	Random	Fixed	AI vs Random	AI vs Fixed
L1_rate0.1	4.7 ± 3.2	16.0 ± 6.6	10.6 ± 7.6	-70.6%	-55.7%
L1_rate0.25	2.5 ± 2.7	7.9 ± 2.0	9.8 ± 3.6	-68.4%	-74.5%
L1_rate0.4	5.3 ± 3.3	16.1 ± 5.2	13.8 ± 5.0	-67.1%	-61.6%
L2_rate0.1	22.1 ± 6.7	12.8 ± 6.9	10.2 ± 4.5	+72.7%	+116.7%
L2_rate0.25	13.5 ± 3.2	19.4 ± 5.9	19.7 ± 6.0	-30.4%	-31.5%
L2_rate0.4	20.9 ± 6.6	33.0 ± 7.7	34.7 ± 9.9	-36.7%	-39.8%
L3_rate0.1	24.0 ± 15.7	20.4 ± 5.4	8.1 ± 2.3	+17.6%	+196.3%
L3_rate0.25	13.6 ± 5.2	23.4 ± 8.0	25.7 ± 4.9	-41.9%	-47.1%
L3_rate0.4	23.3 ± 4.6	49.7 ± 11.7	48.0 ± 6.3	-53.1%	-51.5%

Średnia liczba oczekujących pojazdów - niższa = lepsza:

Konfiguracja	AI	Random	Fixed	Zwycięzca
L1_rate0.1	14.27	8.62	11.03	Random
Konfiguracja	AI	Random	Fixed	Zwycięzca
L2_rate0.1	13.88	22.55	20.68	AI
L3_rate0.1	22.99	24.23	30.49	AI

Średni czas oczekiwania (w klatkach) - niższy = lepszy:

Konfiguracja	AI	Random	Fixed	Zwycięzca
L2_rate0.1	408.2	436.1	437.3	AI
L3_rate0.1	387.5	431.8	448.8	AI

4.6 Analiza wyników - stabilne wnioski

Po wielokrotnym uruchomieniu benchmarku, wyniki są spójne i powtarzalne:

Gdzie AI konsekwentnie wygrywa:

Scenariusz	Przewaga vs Random	Przewaga vs Fixed

L2 + niski ruch	+72.7%	+116.7%
L3 + niski ruch	+17.6%	+196.3%

Gdzie AI konsekwentnie przegrywa:

Scenariusz	Strata vs Random	Strata vs Fixed
L1 (wszystkie)	-67% do -71%	-56% do -75%
L2/L3 + średni ruch	-30% do -42%	-31% do -47%
L2/L3 + ciężki ruch	-37% do -53%	-40% do -52%

4.7 Głęboka diagnoza

Dlaczego L2/L3 przy niskim ruchu działają?

Model nauczył się optymalizować przepływy gdy jest "czas na myślenie". Przy niskim ruchu:

- Błędne decyzje nie powodują natychmiastowych zatorów
- AI może eksperymentować z różnymi konfiguracjami
- Proste baseline (Fixed cycling) są nieefektywne - zbyt często zmieniają światła gdy nie ma potrzeby

Dlaczego L1 nie działa w ogóle?

Poziom 1 ma paradoksalnie więcej akcji (12) niż Poziom 2 (8), mimo że jest "prostszy". Dodaliśmy akcje pojedynczego kierunku (tylko A, tylko B, etc.) które mogą wprowadzać w błąd:

- Model może "utknąć" na faworyzowaniu jednego kierunku
- Przy jednym skrzyżowaniu każda błędna decyzja natychmiast wpływa na wszystkie kierunki

Dlaczego ciężki ruch nie działa?

Mimo treningu na ciężkim ruchu (0.2-0.5), AI przegrywa w tych warunkach. Możliwe przyczyny:

Proste strategie są wystarczająco dobre - Round-robin cycling naturalnie daje każdemu kierunkowi szansę

Chaos jest trudny do optymalizacji - Przy ciężkim ruchu różnice między strategiami się zacierają .

Model może się "panikować" - Widząc duże kolejki, podejmuje nieoptymalne decyzje

5. Wdrożenie modelu i monitorowanie

5.1 Tryb demo

Stworzono interaktywny tryb demo pozwalający obserwować AI w czasie rzeczywistym:

```
python demo.py      # Poziom 1 python
demo.py --level 2 # Poziom 2 python
demo.py --level 3 # Poziom 3
```

Funkcjonalności demo:

- Suwaki do kontroli spawn rate dla każdego punktu wjazdowego
- Przełączanie między trybami: AI / Random / Fixed (klawisz A)
- Przełączanie poziomów (klawisz L)
- Wyświetlanie metryk w czasie rzeczywistym

5.2 Przełączanie trybów sterowania

```
control_modes = ['ai', 'random', 'fixed']

if control_mode == 'ai' and policy_net:
    # AI wybiera akcję na podstawie Q-values
    with torch.no_grad():
        q_values = policy_net(state_tensor)
    action = q_values.argmax().item()
elif control_mode == 'fixed':      # Cykliczne
    przełączanie co 1.5s      fixed_timer += 1      if
    fixed_timer >= 90:    # ~1.5s przy 60fps
    fixed_action_idx = (fixed_action_idx + 1) %
    valid_actions      fixed_timer = 0      action =
    fixed_action_idx
    else:    #
    random
    action = random.randint(0, valid_actions - 1)
```

5.3 Problem: Demo wyświetlało "AI: RANDOM"

Symptom: Mimo załadowanego modelu, demo pokazywało tryb losowy.

Przyczyna: Błędny import klasy DQN:

```
# Błędny import
from train_dqn import DQN  # Stara wersja

# Poprawny import
from train_unified import UnifiedDQN as DQN
```

5.4 Problem: Spawn rate suwaków nie działał

Symptom: Nawet przy suwakach na 0%, pojazdy nadal się spawnowały.

Przyczyna: Funkcja spawnu używała stałego prawdopodobieństwa zamiast wartości z segmentu:

```
# Błędny kod if random.random() > 0.5: #
Stała wartość!           spawn_vehicle()

# Poprawny kod
spawn_rate = self.current_level.segments[seg_id].spawn_rate
if random.random() < spawn_rate:           spawn_vehicle()
```

5.5 Zapisywanie i ładowanie modelu

```
# Zapisywanie checkpointu torch.save({
    'policy_net': policy_net.state_dict(),
    'target_net': target_net.state_dict(),
    'optimizer': optimizer.state_dict(),
    'episode': episode,
    'epsilon': epsilon,
    'state_size': MAX_STATE_SIZE,
    'action_size': MAX_ACTION_SIZE,
    'unified': True,
}, 'dqn_unified_best.pth')

# Ładowanie modelu checkpoint = torch.load('dqn_unified_best.pth',
map_location=device)
policy_net.load_state_dict(checkpoint['policy_net'])
policy_net.eval()
```

6. Podsumowanie i wnioski

6.1 Osiągnięte cele

Cel	Status	Uwagi
Multi-level simulator	Zrealizowany	3 poziomy o różnej złożoności
Implementacja DQN	Zrealizowany	PyTorch + Adam optimizer + CUDA
System benchmarkowy	<input checked="" type="checkbox"/> Zrealizowany	Automatyczne porównanie AI/Random/Fixed

Demo w czasie rzeczywistym	Zrealizowany	Suwaki + przełączanie trybów
AI lepsza niż baseline	Częściowo	L2/L3 przy niskim ruchu: do +196% vs Fixed

6.2 Podsumowanie wyników

AI wygrywa w 2/9 scenariuszy (22%):

Scenariusz	AI vs Random	AI vs Fixed
L2 + niski ruch (0.1)	+72.7%	+116.7%
L3 + niski ruch (0.1)	+17.6%	+196.3%

AI przegrywa w pozostałych 7/9 scenariuszy:

- Level 1: -56% do -75% (wszystkie natężenia ruchu)
- L2/L3 przy średnim/ciążkim ruchu: -30% do -53%

6.3 Kluczowe wyzwania techniczne

Synchronizacja środowisk - Różnice między treningiem a ewaluacją początkowo prowadziły do mylących wyników. Kluczowe było dopasowanie: liczby klatek na krok, częstości spawnu, liczby początkowych pojazdów.

Rozmiar przestrzeni akcji - Próba dodania kontroli czasu trwania ($\times 4$ akcje) dramatycznie pogorszyła wyniki. Uproszczenie do bazowych 12/8/16 akcji było konieczne.

Paradoks L1 - Najprostszy poziom (jedno skrzyżowanie) okazał się najtrudniejszy dla AI. Każda błędna decyzja natychmiast wpływa na wszystkie kierunki.

Odwrotna korelacja z ruchem - AI trenowane na ciężkim ruchu (0.2-0.5) działa lepiej przy lekkim (0.1). Sugeruje to, że model nauczył się strategii wymagających "czasu na myślenie".

6.4 Możliwości rozwoju

Dedykowane modele per-level - Osobny model dla każdego poziomu zamiast jeden do wszystkich

Redukcja akcji L1 – Ponowne usunięcie akcji single-direction (z 12 do 8)

Actor-Critic - Algorytmy jak PPO mogą lepiej radzić sobie z dyskretną przestrzenią akcji

6.5 Wnioski końcowe

1. Reinforcement Learning może pokonać proste baseline, ale tylko w specyficznych warunkach.

Model DQN osiągnął znaczącą przewagę (+72% do +196%) na poziomach 2 i 3 przy niskim ruchu. To pokazuje, że podejście ma potencjał, ale wymaga starannego dopasowania do warunków.

2. Proste strategie są zaskakująco skuteczne.

Cykliczne przełączanie świąteł (Fixed) i losowy wybór (Random) osiągają przyzwoite wyniki bez żadnego uczenia. To sugeruje, że problem sterowania ruchem na małą skalę nie jest tak wrażliwy na optymalizację jak mogłoby się wydawać.

3. Złożoność problemu rośnie nieliniowo.

Dodanie kontroli czasu trwania ($\times 4$ akcje) nie dało $\times 4$ lepszych wyników - wręcz pogorszyło je dramatycznie. W RL, większa przestrzeń akcji często oznacza bardziej skomplikowany problem, nie lepsze rozwiązania.

4. Unikać over-engineeringu.

Prosta funkcja nagrody (`throughput × 10`) działała lepiej niż złożona z 5 komponentami. Prostsza przestrzeń akcji działała lepiej niż rozbudowana. W RL, prostota jest często cnotą.

5. Nie wszystkie poziomy są równe.

Model trenowany na wszystkich poziomach radził sobie dobrze z L2/L3, ale słabo z L1. Sugeruje to, że "unified" model może nie być optymalny - specjalizacja per-level mogłaby dać lepsze wyniki.

Małe zwycięstwa są możliwe - AI osiągnęło lepsze wyniki na Poziomie 2 przy niskim ruchu (+76% vs Fixed). To pokazuje, że podejście ma potencjał, ale wymaga więcej dopracowania.

Instrukcja uruchomienia

Wymagania

```
pip install pygame torch numpy
```

Pliki projektu

```
AGH-Traffic-Simulation/
├── simulation.py      # Główny symulator
├── train_unified.py   # Skrypt treningowy
├── benchmark.py       # System benchmarkowy
├── demo.py            # Interaktywne demo
└── dqn_unified_best.pth # Wytrenowany model (jeśli dostępny)
└── README.md          # Dokumentacja
```

Uruchomienie

```
# Trening nowego modelu (domyślnie 3000 epizodów, ~20 minut)
python train_unified.py

# Szybki benchmark (10 epizodów, 100 kroków, ~2 minuty)
python benchmark.py
# Szybki test benchmark
python benchmark.py --quick

# Demo (domyślnie Level 1)
python demo.py python
demo.py --level 2 python
demo.py --level 3
```

Sterowanie w demo

Klawisz	Akcja
A	Przełącz tryb (AI/Random/Fixed)
L	Przełącz poziom
R	Reset suwaków do 25%
M	Maksymalne spawn rates
Z	Zerowe spawn rates
ESC	Wyjście

Załączniki

A. Pełne wyniki benchmarku (v3 - uproszczona przestrzeń akcji)

BENCHMARK RESULTS v3 (100 steps, 10 runs per config)

Action space: simple (L1=12, L2=8, L3=16 actions)

Training: 3000 episodes, spawn rate 0.2-0.5, reward = throughput × 10

Config

AI

RANDOM

FIXED

THROUGHPUT (higher is better):

	AI	RANDOM	FIXED
L1_rate0.1	4.7 ± 3.2	16.0 ± 6.6	10.6 ± 7.6
L1_rate0.25	2.5 ± 2.7	7.9 ± 2.0	9.8 ± 3.6
L1_rate0.4	5.3 ± 3.3	16.1 ± 5.2	13.8 ± 5.0
L2_rate0.1	22.1 ± 6.7	12.8 ± 6.9	10.2 ± 4.5
---<-- AI WINS			
L2_rate0.25	13.5 ± 3.2	19.4 ± 5.9	19.7 ± 6.0
L2_rate0.4	20.9 ± 6.6	33.0 ± 7.7	34.7 ± 9.9
L3_rate0.1	24.0 ± 15.7	20.4 ± 5.4	8.1 ± 2.3
---<-- AI WINS			
L3_rate0.25	13.6 ± 5.2	23.4 ± 8.0	25.7 ± 4.9
L3_rate0.4	23.3 ± 4.6	49.7 ± 11.7	48.0 ± 6.3

AVG WAITING VEHICLES (lower is better):

L1_rate0.1	14.27	8.62
11.03		
L1_rate0.25	41.87	41.27
38.03		
L1_rate0.4	66.98	56.57
60.99		
L2_rate0.1	13.88	22.55
20.68	<-- AI WINS	
L2_rate0.25	55.94	55.53
55.38		

L2_rate0.4	90.42	83.34
86.04		
L3_rate0.1	22.99	24.23
30.49 <-- AI WINS		
L3_rate0.25	77.50	75.39
69.77		
L3_rate0.4	122.64	112.58
112.61		

AVG WAIT TIME (lower is better):

L1_rate0.1	427.9	375.9
409.6		
L1_rate0.25	429.0	433.3
436.7		
L1_rate0.4	435.0	426.5
417.5		
L2_rate0.1	408.2	436.1
437.3 <-- AI WINS		
L2_rate0.25	443.0	437.8
434.5		
L2_rate0.4	426.9	417.3
417.7		
L3_rate0.1	387.5	431.8
448.8 <-- AI WINS		
L3_rate0.25	439.5	428.2
429.2		
L3_rate0.4	426.2	424.3
416.3		

SUMMARY: AI vs Others (% improvement in throughput)

L1_rate0.1: vs random: -70.6% vs fixed: -55.7%
L1_rate0.25: vs random: -68.4% vs fixed: -74.5%
L1_rate0.4: vs random: -67.1% vs fixed: -61.6%
L2_rate0.1: vs random: +72.7% vs fixed: +116.7% <-- AI
DOMINUJE

```
L2_rate0.25:    vs random: -30.4%    vs fixed: -31.5%
L2_rate0.4:    vs random: -36.7%    vs fixed: -39.8%
    L3_rate0.1:    vs random: +17.6%    vs fixed: +196.3% <-- AI
DOMINUJE
L3_rate0.25:    vs random: -41.9%    vs fixed: -47.1%
L3_rate0.4:    vs random: -53.1%    vs fixed: -51.5%
=====
=====
```

B. Przebieg treningu v3

```
=====
UNIFIED DQN TRAINING v3 (SIMPLIFIED)
=====

Changes: Simple actions (8-16), heavy traffic, throughput-only reward
Episodes: 3000
Steps per episode: 100
Spawn rate: 0.2-0.5 (heavy traffic focus)
Reward: throughput × 10 (simple)
Epsilon: 1.0 → 0.01 (decay: 0.9975)
Learning rate: 0.001 (decay every 100 eps)
=====

Ep      0/1500 | L1 | R: 10.00 | Avg L1: 10.0 L2: 0.0 L3: 0.0 | Eps:1.000
LR:0.00100
Ep    500/1500 | L2 | R: 520.00 | Avg L1:380.2 L2:490.5 L3:445.3 | Eps:0.287
LR:0.00059
Ep 1000/1500 | L3 | R:1250.00 | Avg L1:720.4 L2:980.8 L3:1180.2 |
Eps:0.082 LR:0.00035
Ep 1490/1500 | L3 | R:2079.00 | Avg L1:1042.2 L2:1523.0 L3:1911.8 |
Eps:0.024 LR:0.00023
=====

TRAINING COMPLETE
Total time: 19.9 minutes
Avg per episode: 0.80 seconds
Episodes completed: 1500 (z 3000 planowanych - przerwane wcześniej)
Models saved: dqn_unified_best.pth, dqn_unified_final.pth
=====
```

Komentarz: Nagrody v3 są ~100× wyższe niż poprzedniej wersji
dzięki:
- Prostszej przestrzeni akcji (12/8/16 zamiast 48/32/64)
- Prostszej funkcji nagrody (throughput × 10)
- Skupieniu na ciężkim ruchu (0.2-0.5)