

Inspiracja – symulowany samochód autonomiczny

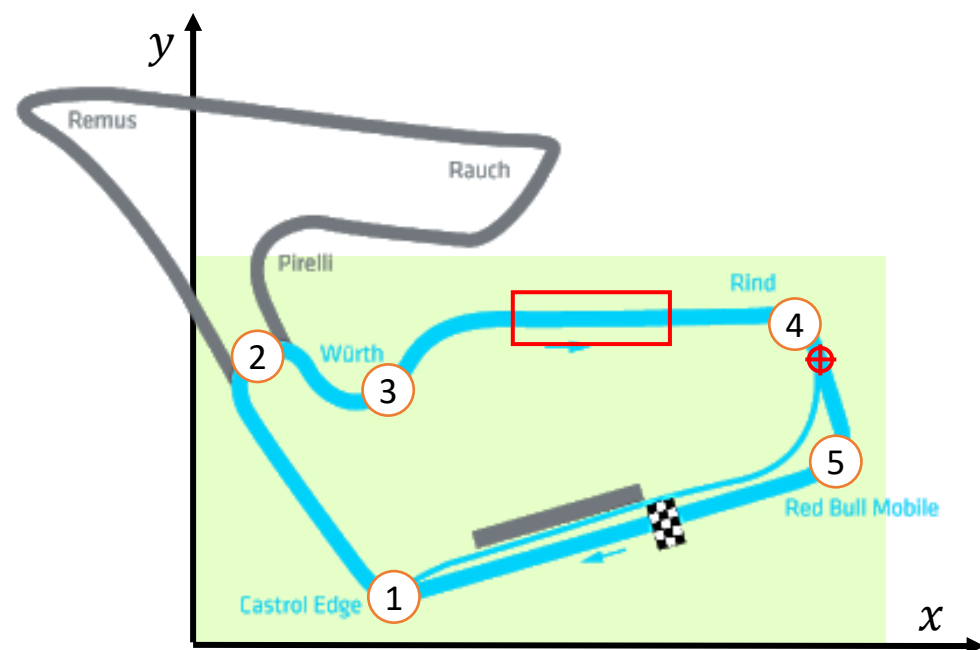
pythonprogramming.net



Tor, scenariusze

- PNG 1920x1080, kanały RGB
- skala 22 piksele = 1 m
- szerokość toru i margines od innych elementów 3-4 m
- pętla jednokierunkowa, możliwe skrzyżowania

- scenariusz = sekwencja manewrów
- manewr jako element treningu pokonywania typowego fragmentu topologii
- manewr = obszar startowy + cel
- obszar startowy musi obejmować fragment toru, może obejmować trawnik
- cel zawsze „z przodu”
- CSV, 1 scenariusz, maks. 8 manewrów



1

2

3

4

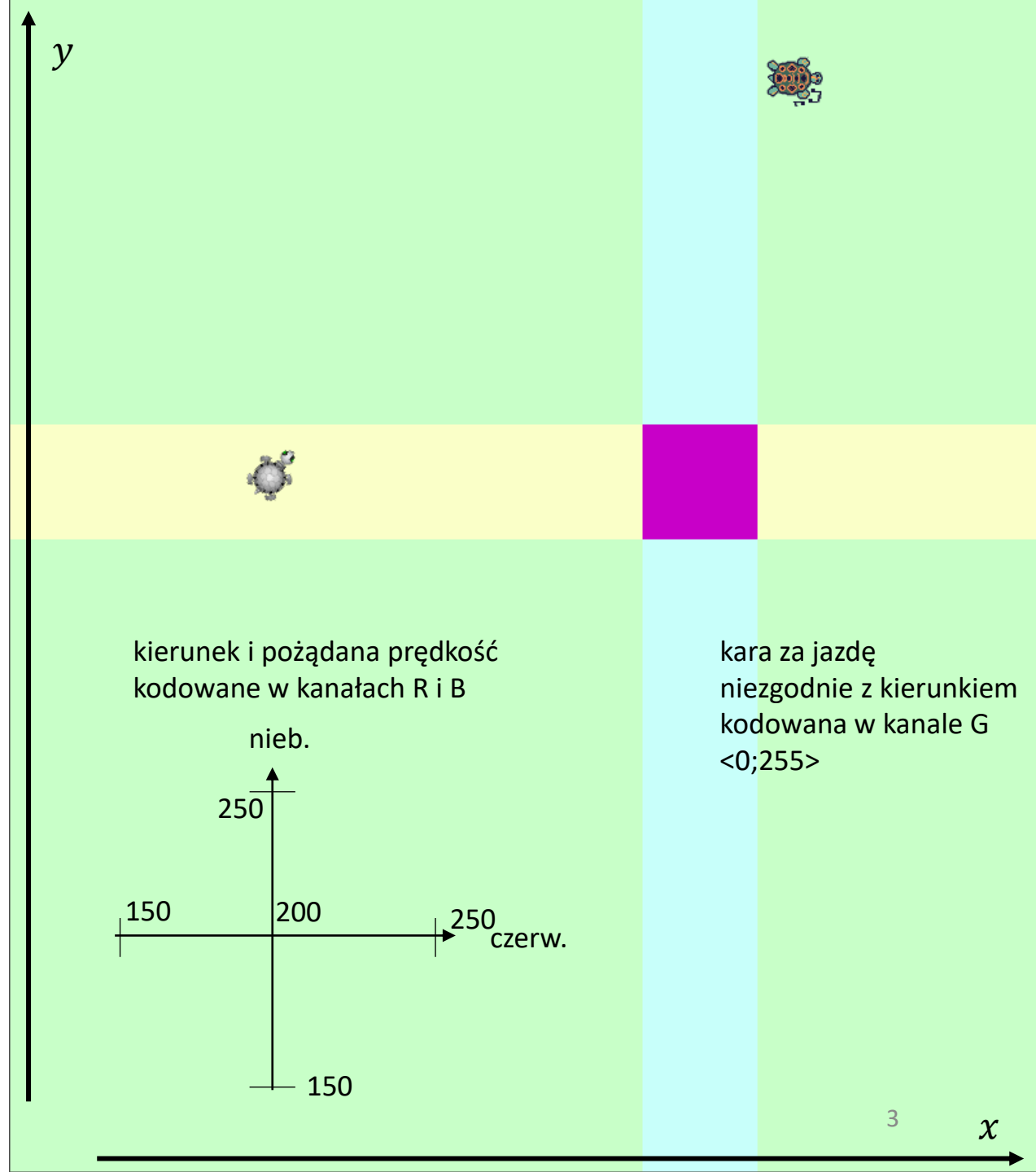
5

liczba agentów
uruchamianych w tym
obszarze – aktualnie 1

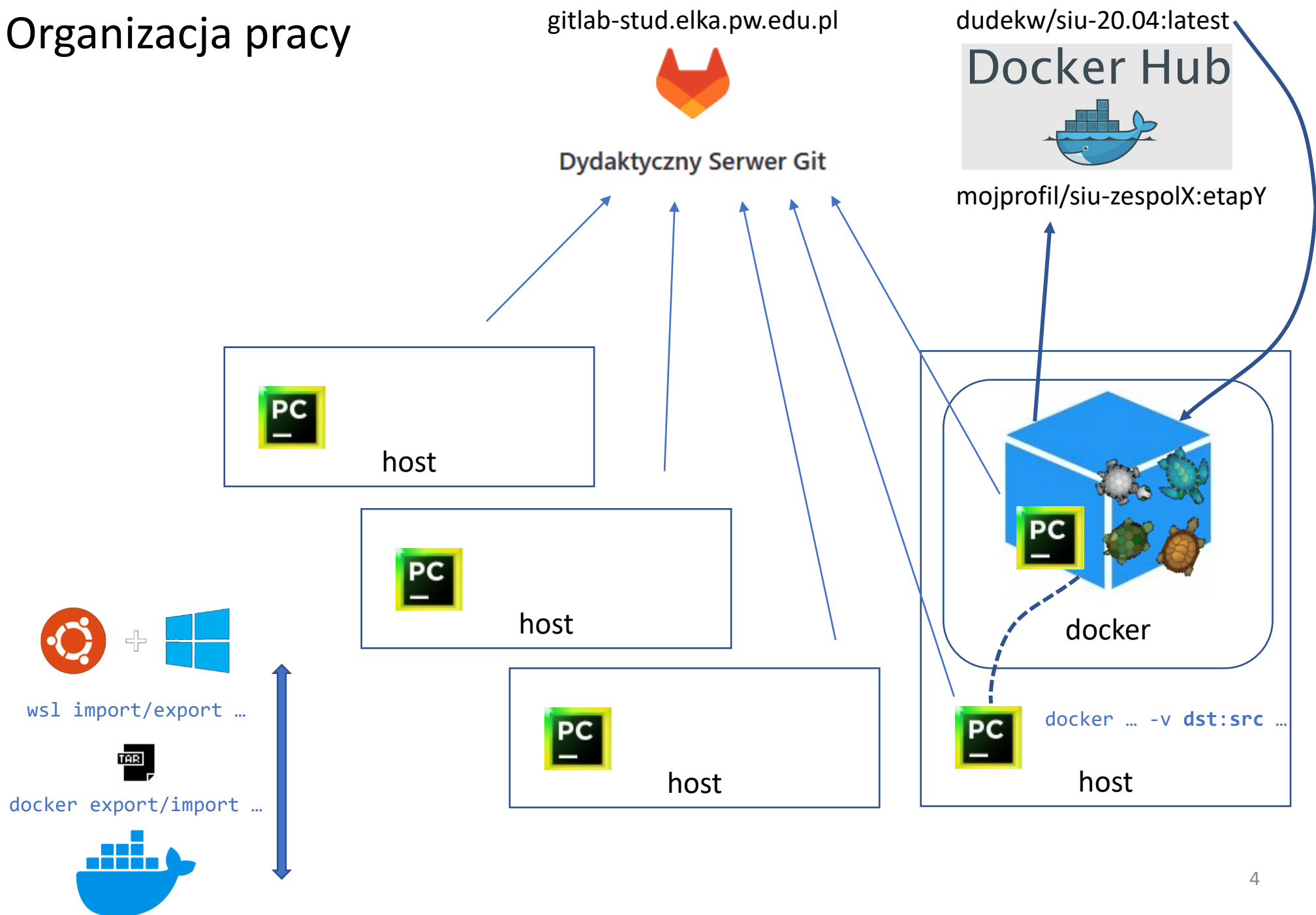
$id_scenariusza; 1; x_{min}^{(1)}; x_{max}^{(1)}; y_{min}^{(1)}; y_{max}^{(1)}; x_g^{(1)}; y_g^{(1)}$

Kodowanie kolorów

- Trawnik ma kolor 200,200,255
- Preferowana gradacja na łukach
- Kanał G koduje istotność zalecanego kierunku i prędkości jazdy



Organizacja pracy



Organizacja kodu

- Kontrakty dla środowiska symulacyjnego i uczącego ustanowione w klasach bazowych
 - TurtleSimEnvBase
 - DqnBase
- Przykładowa implementacja środowiska 1-agentowego
 - TurtleSimEnvSingle
- Ucząc, sprawdzić wpływ wybranych parametrów na rozwiązanie
 - * - nie zmieniać
 - > - nie zwiększać
 - < - nie zmniejszać
- Uzupełnić brakujące fragmenty kodu:

TODO STUDENCI

...

```
class TurtleAgent:
    pass

class TurtleSimEnvBase(metaclass=abc.ABCMeta):
    # określa parametry symulacji (poniższy zestaw i wartości)
    def __init__(self):
        # parametry czujnika wizyjnego i interakcji z symulacją
        self.GRID_RES = 5          # liczba komórek siatki
        self.CAM_RES = 200         # dł. boku siatki
        self.SEC_PER_STEP = 1.0    # *okres dyskretyzacji
        self.WAIT_AFTER_MOVE = .01 # oczekiwanie po ruchu
        # parametry oceny sytuacyjnej
        self.SPEED_RWRD_RATE = 0.5 # >wzmocnienie nagrody
        self.SPEED_RVRS_RATE = -10.0 # <wzmocnienie kary
        self.SPEED_FINE_RATE = -10.0 # <wzmocnienie kary
        self.DIST_RWRD_RATE = 2.0   # >wzmocnienie nagrody
        self.OUT_OF_TRACK_FINE = -10 # <ryczałtowa kara
        self.COLLISION_DIST = 1.5   # *odległość wykrycia
        self.DETECT_COLLISION = False # tryb wykrywania
        self.MAX_STEPS = 20         # maksymalna liczba kroków
        self.PI_BY = 6              # *dzielnik zakresu

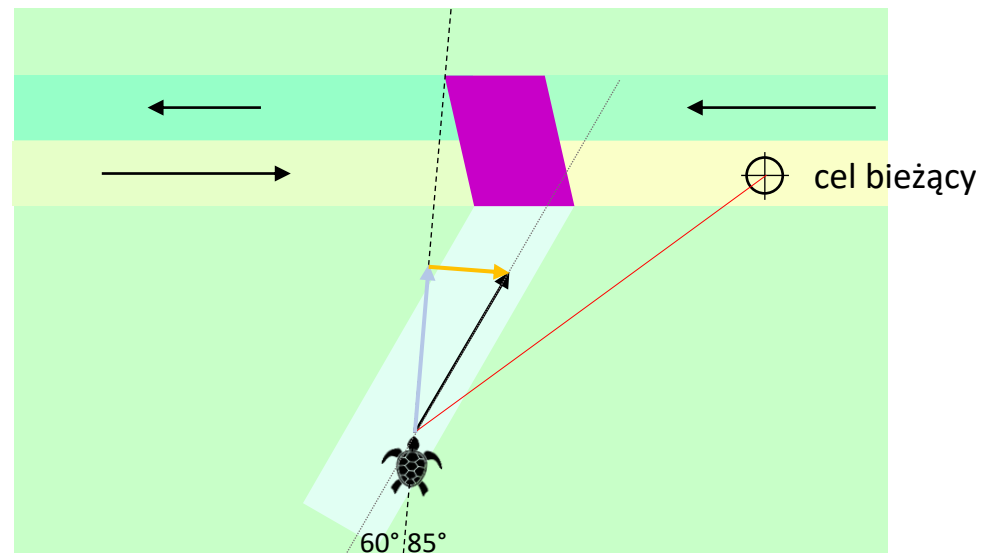
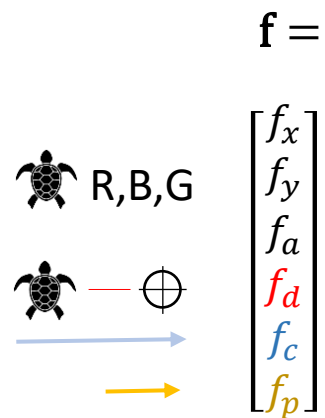
    # uzupełnić setup() i reset()
```

```
class TurtleSimEnvSingle(TurtleSimEnvBase):
    def __init__(self):
        super().__init__()
    def step(self, actions, realtime=False):...

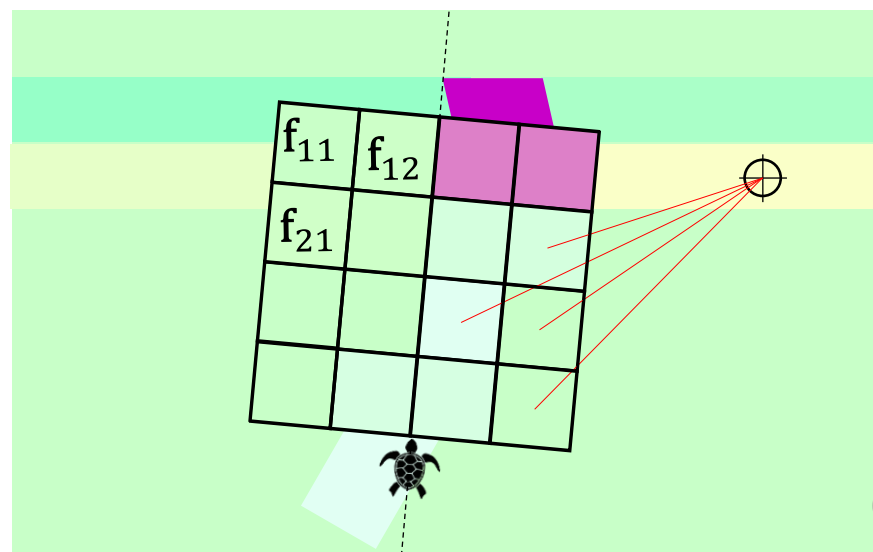
    # uzupełnić step()
```

Świadomość sytuacyjna (klasa bazowa)

get_road(tname)



get_map(tname)

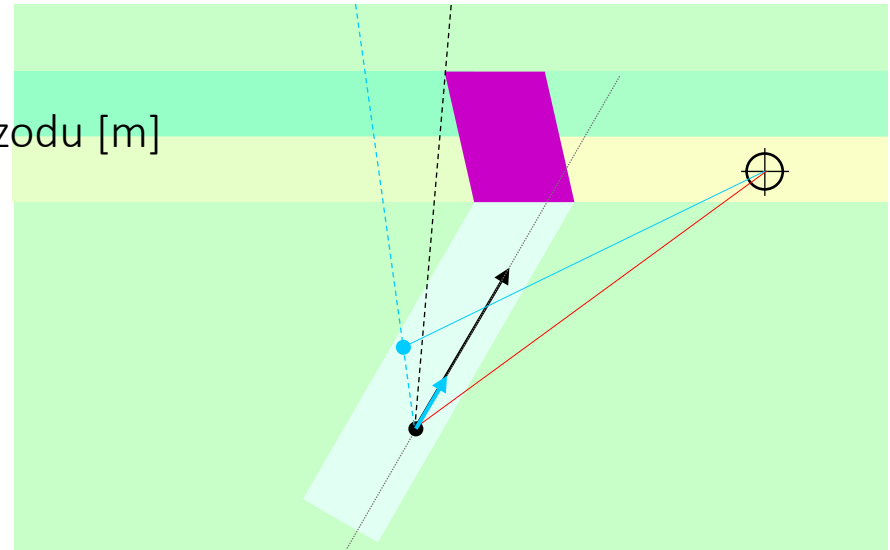


Symulacja kroku sterowania (klasa pochodna)

`step(actions, realtime)`

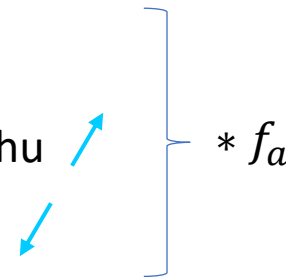
`action[0]` – wartość przesunięcia/prędkości do przodu [m]

`action[1]` – wartość skrętu w lewo [rad]



Składniki nagrody:

- kara proporcjonalna do przekroczenia prędkości
- nagroda proporcjonalna do przemieszczenia w zalecanym kierunku ruchu
- kara proporcjonalna do jazdy pod prąd
- nagroda za zbliżanie się do celu, $f_d(t+1) - f_d(t)$
- kara za wypadnięcie z trasy



Uczenie się ze wzmocnieniem w dyskretnej przestrzeni stanów

	a			
s	UP	DOWN	LEFT	RIGHT
0				
1				
2				
3				
4				
5				
6				
7				
8				

$Q(s, a)$ – ocena akcji a w stanie s

$r(s, a)$ – nagroda za akcję a w stanie s

f – funkcja stanu obiektu, $s' = f(s, a)$

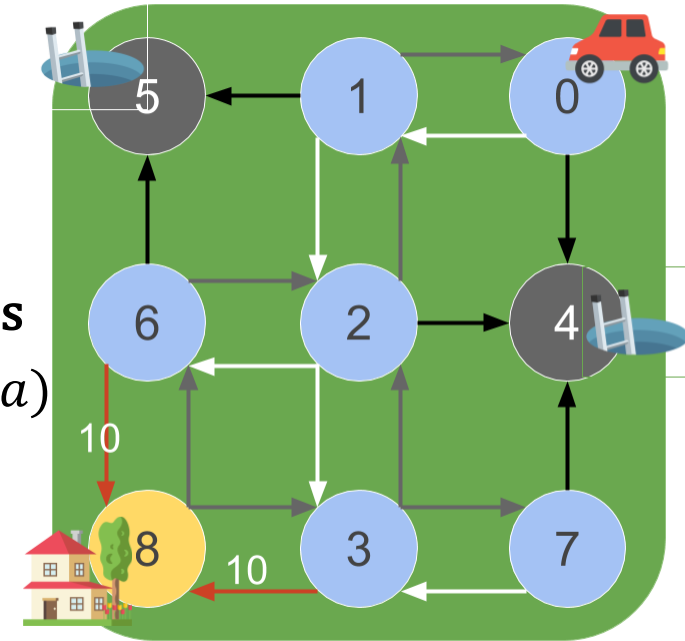
$\max_{c'} Q(s', a')$ – najlepsza z ocen
w wynikowym stanie s'

α – szybkość uczenia

γ – dyskonto (uwzględnianie przyszłych nagród)

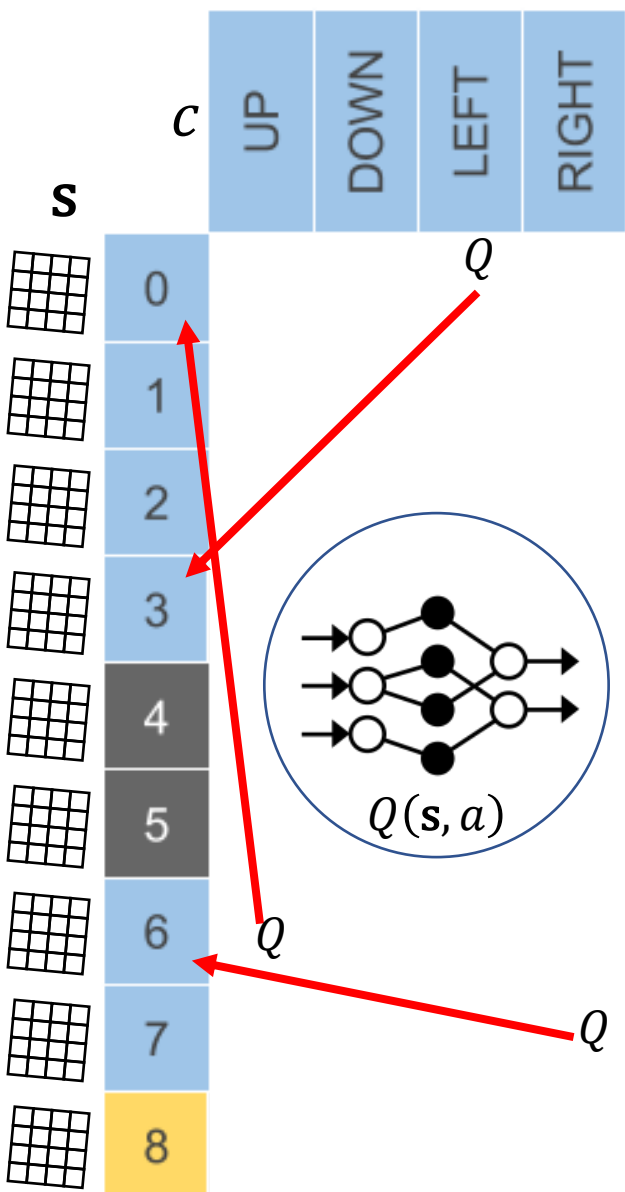
Funkcja Q ocenia każde możliwe sterowanie w każdym możliwym stanie. Jeśli stanów i sterowań jest mało, można (i należy) przechowywać jej wartości w tablicy.

Jeśli przestrzeń stanów jest ciągła, zakładamy że Q nie jest tabelą, ale funkcją określonej klasy (np. definiowaną przez sieć głęboką) i poszukujemy jej parametrów.



$$Q(s, a) := (1 - \alpha)Q(s, a) + \alpha \left[r(s, a) + \gamma \max_{a'} Q(s', a') \right]$$

Uczenie się ze wzmocnieniem w ciągłej przestrzeni stanów = poszukiwanie $Q(s, a)$



`DqnSingle.train_main()`

Strojenie (ulepszanie) funkcji $Q(s, a)$

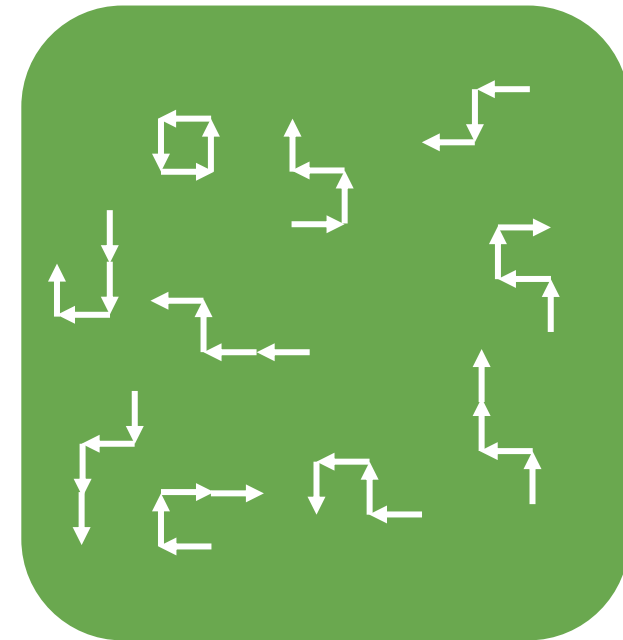
Powtarzaj:

- zrób kilka kroków, dodaj do historii
- wylosuj próbkę kroków z historii
- wyznacz $\max_{a'} Q(s', a')$ dla każdego kroku z próbki
- wyznacz nowe pożądane wartości $Q(s, a)$
- doucz sieć na wybranej próbce

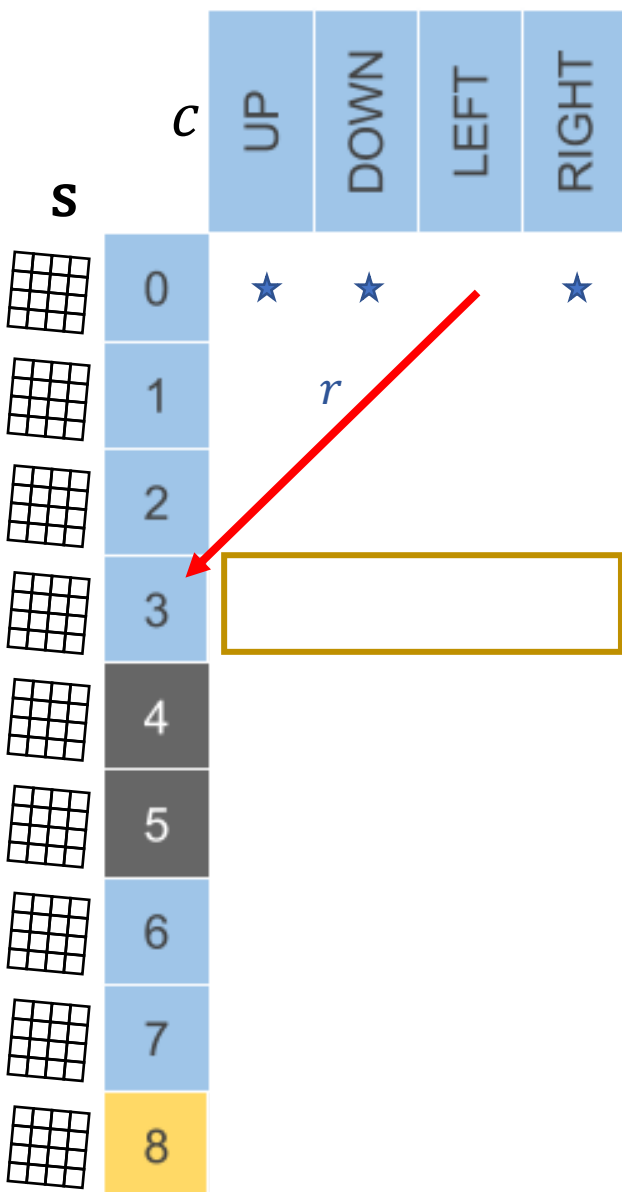
`DqnSingle.replay_memory=[(s, a, r, s'), ...]`

$$Q(s, a) = r(s, a) + \gamma \max_{a'} Q(s', a')$$

$$Q(s, a) := (1 - \alpha) Q(s, a) + \alpha \left[r(s, a) + \gamma \max_{a'} Q(s', a') \right]$$



Dwie sieci – doraźna i docelowa

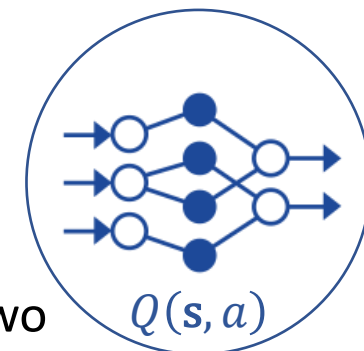


Strojenie (ulepszanie) funkcji $Q(s, a)$

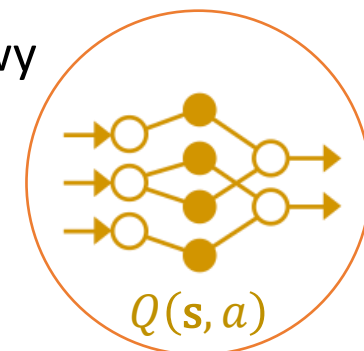
Powtarzaj:

- zrób kilka kroków, dodaj do historii
- wylosuj próbkę kroków z historii
- wyznacz $\max_{a'} Q(s', a')$ dla każdego kroku z próbki
- wyznacz nowe pożądane wartości $Q(s, a)$
- doucz sieć na wybranej próbce

sieć bieżąca



Okresowo aktualizuj model docelowy



sieć docelowa

$$Q(s, a) = r(s, a) + \gamma \max_{a'} Q(s', a')$$

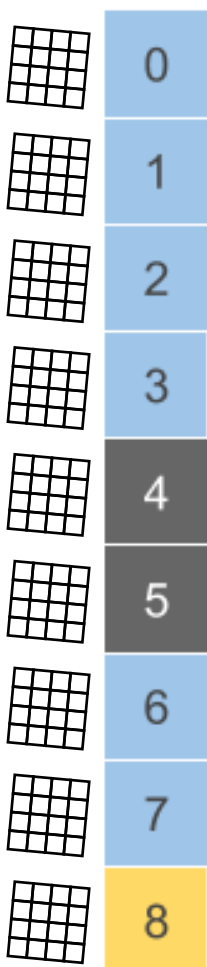


$$Q(s, a) = r(s, a) + \gamma \max_{a'} Q(s', a')$$

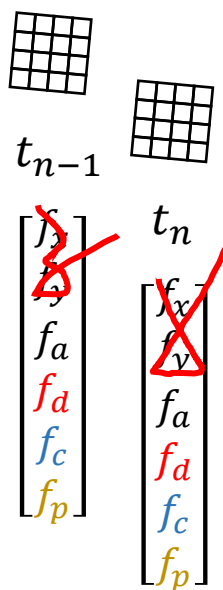
Architektura sieci

W etapie 3:

- uzupełnij kod w klasach środowiska
- wytrenuj sieć na własnej planszy
- popraw wyniki, zmieniając co najmniej 2 parametry klas i co najmniej 1 parametr lub strukturę sieci neuronowej



DqnSingle.
inp_stack()



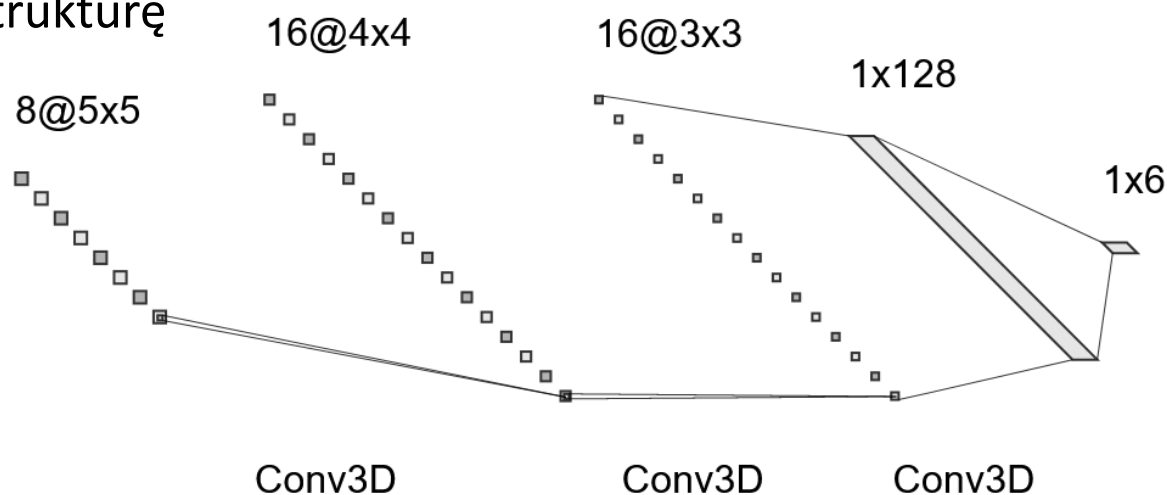
DqnSingle.
ctl2act()

action[0] – wartość przesunięcia do przodu [m]

action[1] – wartość skrętu w lewo [rad]

	prędkość\skręt	-0,1 rad	0	0,1 rad
$c =$	0,2 m/s	c[0]	c[1]	c[2]
	0,4 m/s	c[3]	c[4]	c[5]

DqnSingle.make_model()



<https://storage.googleapis.com/deepmind-media/dqn/DQNNaturePaper.pdf>

<https://pythonprogramming.net/reinforcement-learning-self-driving-autonomous-cars-carla-python/?completed=reinforcement-learning-agent-self-driving-autonomous-cars-carla-python/>

Jak przygotować i przekazać wyniki etapu 3

W katalogu projektu:

- pełna implementacja klas środowiska i uczenia DQN
- plansza i scenariusz
- model lub modele godne pokazania, w formacie tf
- skrypt w Pythonie symulujący zachowanie agenta, np. `play_single_handout.py`

W uzgodnieniu z prowadzącym:

- zawartość katalogu projektowego ALBO
- obraz maszyny wirtualnej ALBO
- obraz Dockera

Wersja wieloagentowa środowiska

Uzupełnia klasę bazową:

- liczniki kroków indywidualne dla agentów
- step() przesuwa wybrane agenty
- wykrywanie kolizji
- zwraca słownik sytuacji poruszonych agentów

```
agent1;2;18;22.54545455;9.045454545;13.59090909;16.8  
agent1;2;16.13636364;20.68181818;17;21.54545455;25.0  
agent1;2;15.95454545;20.5;33.5;38.04545455;22.772727  
agent1;2;38.5;43.04545455;40.77272727;45.31818182;49  
agent1;2;45.13636364;49.68181818;28.09090909;32.6363  
agent1;2;40.95454545;45.5;9.227272727;13.77272727;33
```

```
actions = {dict: 12} {'agent1_4_0': [0.2  
> 'agent1_4_0' = {list: 2} [0.2, 0.0]  
> 'agent1_1_0' = {list: 2} [0.4, 0.0]  
> 'agent1_2_1' = {list: 2} [0.2, -0.25]  
> 'agent1_3_0' = {list: 2} [0.2, 0.0]
```

```
scene = {dict: 12} {'agent1_4_0':  
> 'agent1_4_0' = {tuple: 4} ((arra  
> 'agent1_1_0' = {tuple: 4} ((arra  
> 'agent1_2_1' = {tuple: 4} ((arra  
> 'agent1_3_0' = {tuple: 4} ((arra  
> 'agent1_2_0' = {tuple: 4} ((arra  
> 'agent1_1_1' = {tuple: 4} ((arra  
0 = {tuple: 7} (array([[0.  
0 = {ndarray: (5, 5)} [[0.  
1 = {ndarray: (5, 5)} [[0.  
2 = {ndarray: (5, 5)} [[1.  
3 = {ndarray: (5, 5)} [[4.  
4 = {ndarray: (5, 5)} [[1.  
5 = {ndarray: (5, 5)} [[1.  
6 = {ndarray: (5, 5)} [[1.  
__len__ = {int} 7  
1 = {float64} 0.6723305908
```

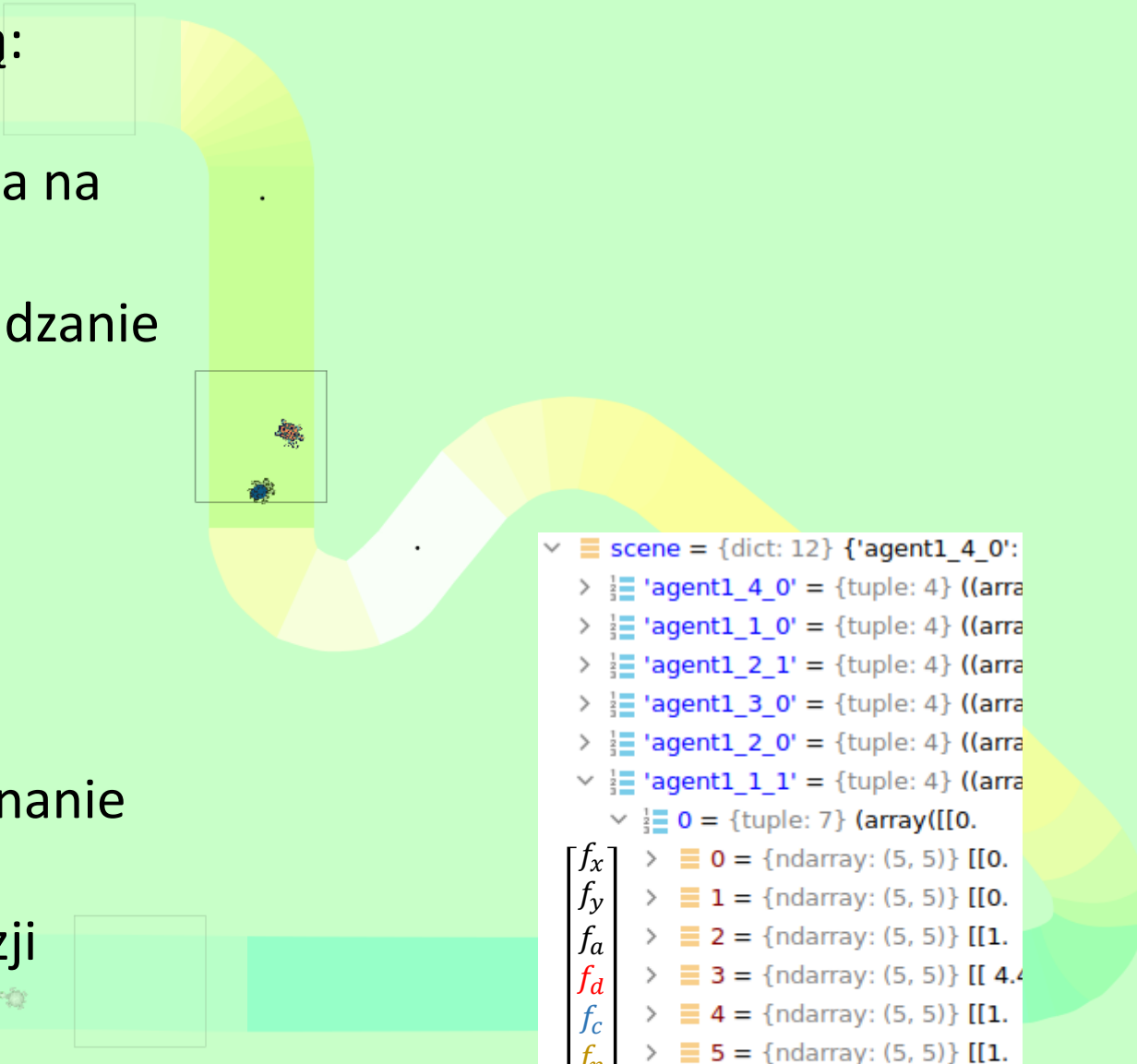
Wersja wieloagentowa klasy uczącej

Uzupełnia klasę jednoagentową:

- wejście sieci 5x5x10 (+inni)
- pętla treningowa zorientowana na kroki symulacji
- wykrywanie i selektywne odradzanie agentów zatrzymanych przez środowisko
- douczanie z bieżących ruchów wszystkich agentów

Jak uczyć:

- model bezkolizyjny (tylko pokonanie trasy)
- douczanie z wykrywaniem kolizji



```
scene = {dict: 12} {'agent1_4_0':  
> 'agent1_4_0' = {tuple: 4} ((arra  
> 'agent1_1_0' = {tuple: 4} ((arra  
> 'agent1_2_1' = {tuple: 4} ((arra  
> 'agent1_3_0' = {tuple: 4} ((arra  
> 'agent1_2_0' = {tuple: 4} ((arra  
v 'agent1_1_1' = {tuple: 4} ((arra  
v 0 = {tuple: 7} (array([[0.  
[f_x] > 0 = {ndarray: (5, 5)} [[0.  
[f_y] > 1 = {ndarray: (5, 5)} [[0.  
[f_a] > 2 = {ndarray: (5, 5)} [[1.  
[f_a] > 3 = {ndarray: (5, 5)} [[ 4.  
[f_c] > 4 = {ndarray: (5, 5)} [[1.  
[f_p] > 5 = {ndarray: (5, 5)} [[1.  
[f_o] > 6 = {ndarray: (5, 5)} [[1. 1  
01 __len__ = {int} 7  
01 1 = {float64} 0.6723305908
```