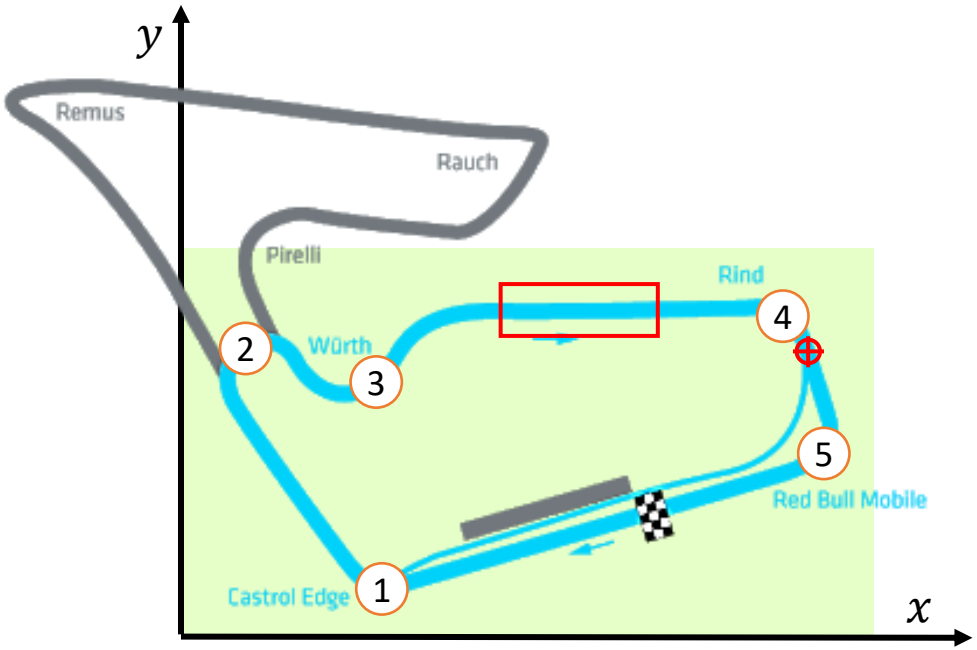
Inspiracja – symulowany samochód autonomiczny

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 = ciąg etapów
- w etapie – trening pokonywania fragmentu topologii
- etap = obszar startowy + cel
- obszar startowy musi obejmować fragment toru, może obejmować trawnik
- cel zawsze „z przodu”
- CSV, 1 scenariusz, maks. 8 etapów

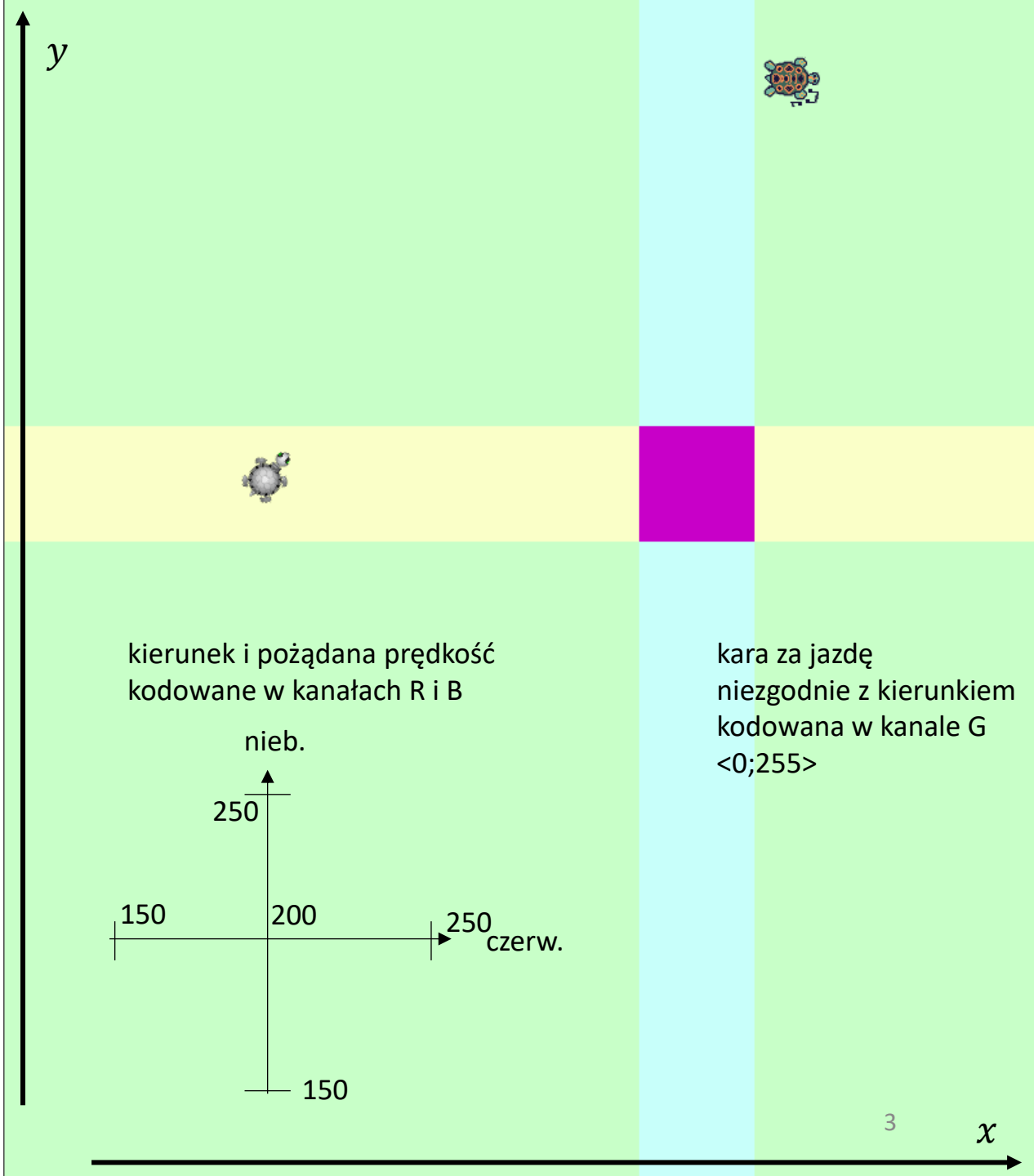
```
1;1;41.77;45.55;18.14;32.05;47.32;37.5
1;1;51.00;55.95;31.91;38.00;55.64;27.7
1;2;36.64;50.55;22.64;26.36;32.73;21.3
```

liczba agentów uruchamianych w tym obszarze – aktualnie 1

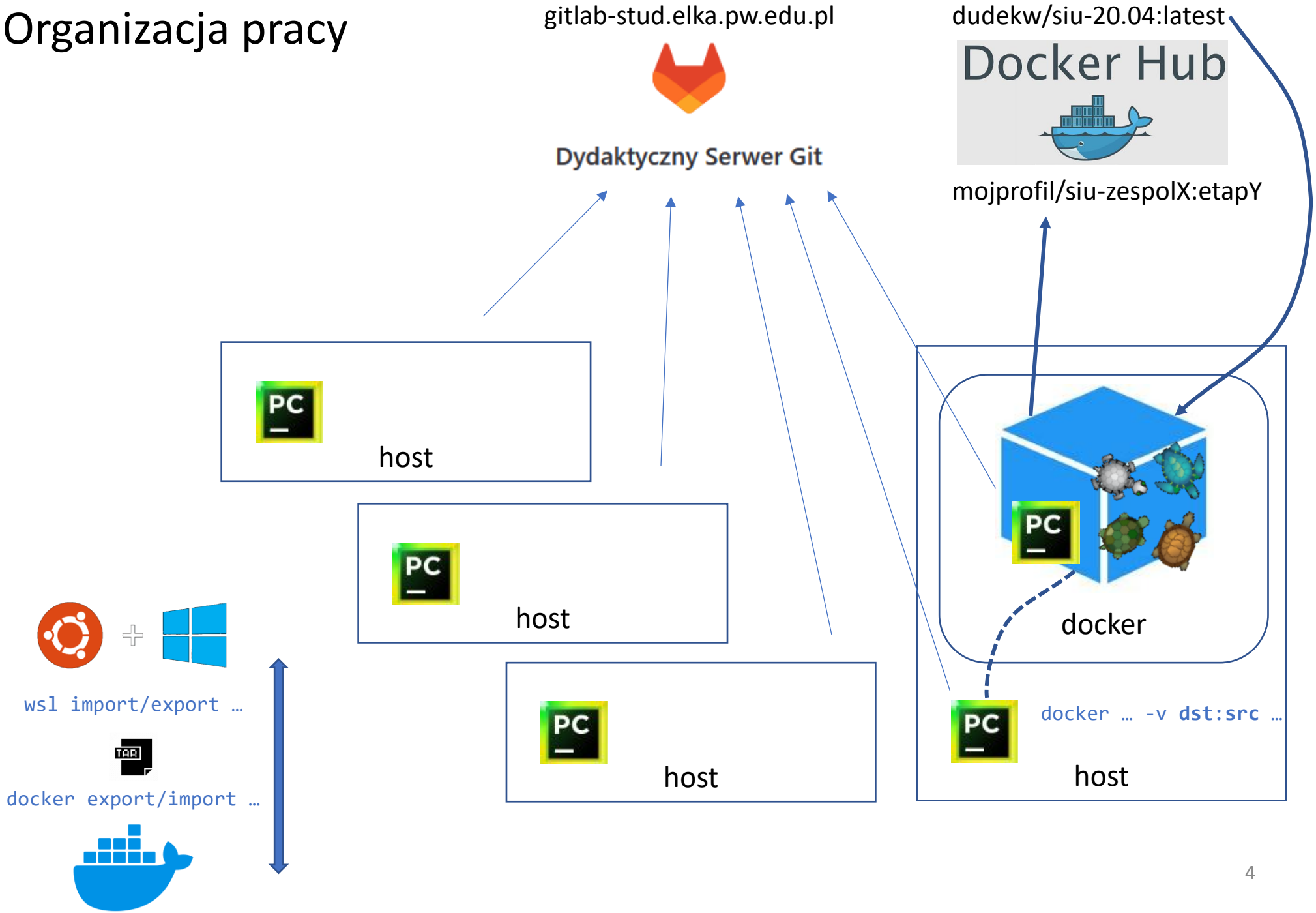
id_scenariusza;1;$x_{min}^{(1)}$;$x_{max}^{(1)}$;$y_{min}^{(1)}$;$y_{max}^{(1)}$;$x_g^{(1)}$;$y_g^{(1)}$

Kodowanie kolorów

- Trawnik ma kolor R=200,G=255,B=200
- Przynajmniej 1 zakręt wyoblony
- Preferowana gradacja na łukach
- Kanał G koduje istotność zalecanego kierunku i prędkości jazdy



Organizacja pracy



Organizacja pracy – suplement – praca bez Dockera

Alternatywą dla Dockera jest Ubuntu na maszynie wirtualnej albo bezpośrednio na sprzęcie.
Aby zainstalować symulator, należy wykonać poniższe kroki (Ubuntu 20.04):

1. zainstalować ROS Noetic według instrukcji http://wiki.ros.org/noetic/Installation/Ubuntu
2. mkdir -p \$HOME/siu_ws/src
3. cd \$HOME/siu_ws/src
4. sudo apt update
5. sudo apt upgrade
6. sudo apt install ros-noetic-teleop-twist-keyboard ros-noetic-moveit-simple-controller-manager ros-noetic-imu-sensor-controller ros-noetic-force-torque-sensor-controller ros-noetic-twist-mux ros-noetic-four-wheel-steering-controller ros-noetic-image-proc ros-noetic-range-sensor-layer ros-noetic-pointcloud-to-laserscan ros-noetic-laser-filters ros-noetic-teb-local-planner ros-noetic-gmapping ros-noetic-octomap-server ros-noetic-octomap-mapping ros-noetic-people-msgs ros-noetic-rosbridge-server ros-noetic-rospy-message-converter ros-noetic-sound-play ros-noetic-object-recognition-msgs ros-noetic-moveit-ros-planning-interface ros-noetic-move-base-msgs bison flex ros-noetic-navigation ros-noetic-moveit-planners-ompl python-scipy python-catkin-tools ros-noetic-turtlebot3* ros-noetic-ddynamic-reconfigure python3-catkin-pkg-modules python3-rospkg-modules python3-opencv python3-matplotlib python-rosinstall git
7. git clone -b noetic-devel https://github.com/RCPRG-ROS-pkg/ros_tutorials
8. cd $HOME/siu_ws
9. source /opt/ros/noetic/setup.bash
10. catkin build

Przy tej instalacji plansza wykorzystywana przez żółwie znajduje się pod ścieżką:
$HOME/siu_ws/src/ros_tutorials/turtlesim/images/roads.png

Uwaga: w przypadku pracy *z Dockerem*, po zainstalowaniu tensorflow może zaistnieć potrzeba aktualizacji bibliotek obsługujących połączenie przez VNC, poleceniem:
pip install --upgrade Jinja2 flask

Organizacja kodu

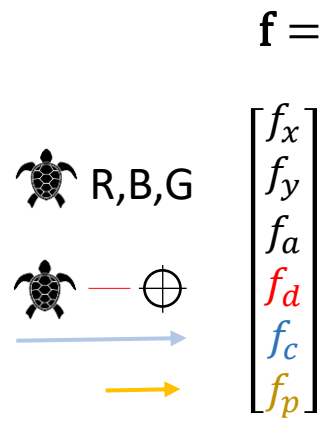
- Kontrakty dla środowiska symulacyjnego i uczącego ustanowione w klasach bazowych
  - `TurtlesimEnvBase`
  - `DqnBase`
- Przykładowa implementacja środowiska 1-agentowego
  - `TurtlesimEnvSingle`
- Ucząc, sprawdzić wpływ wybranych parametrów na rozwiązanie
  - * - nie zmieniać
  - > - nie zwiększać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

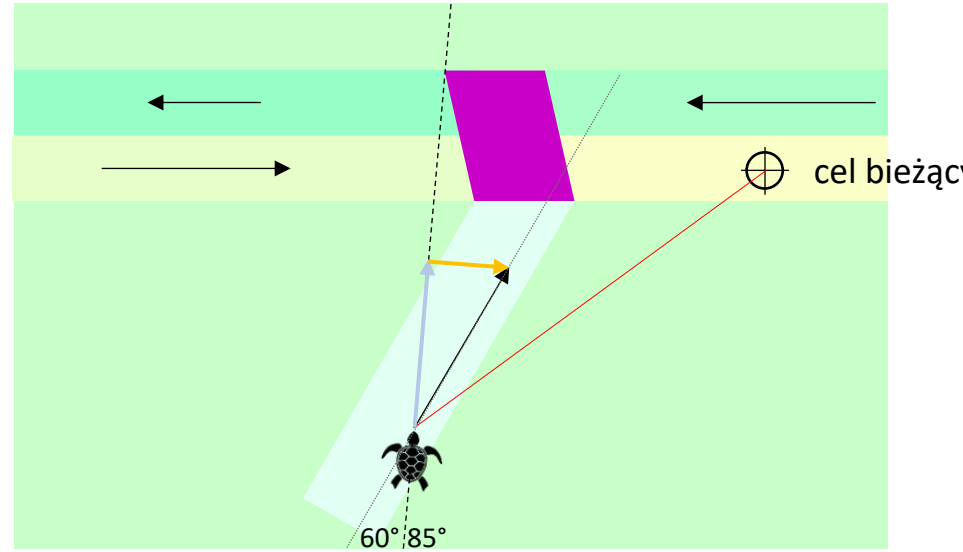
```
# TODO STUDENCI
...
```

```
class TurtleAgent:       # struktura ze stałymi i bieżącymi
    pass                 # route, sec_id, seq, color_api, g

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

class TurtlesimEnvSingle(TurtlesimEnvBase):
    def __init__(self):
        super().__init__()
    def step(self,actions,realtime=False):...
    uzupełnić step()
```

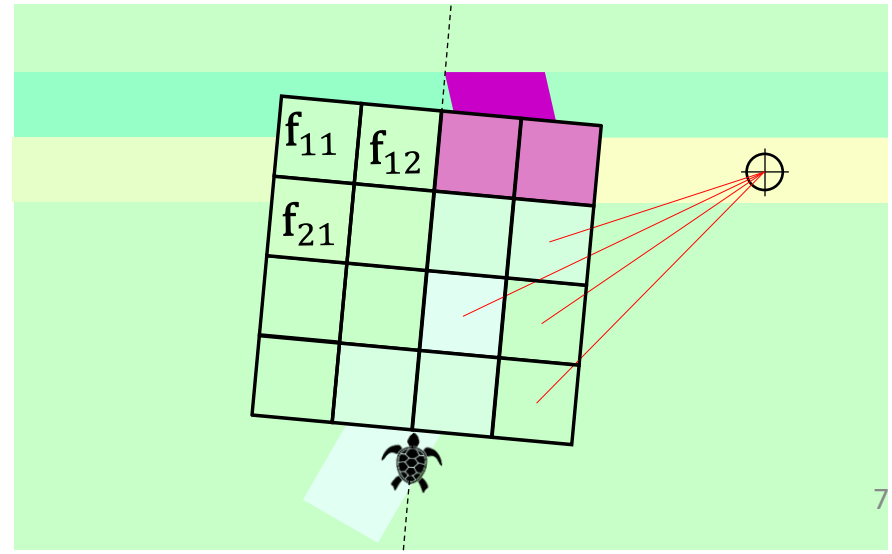
Świadomość sytuacyjna (klasa bazowa)

`get_road(tname)`





`get_map(tname)`

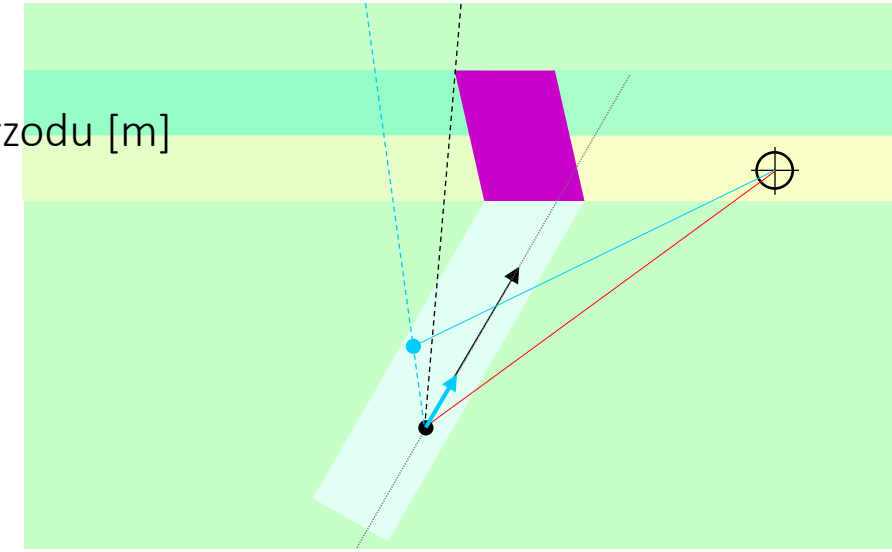


Symulacja kroku sterowania (klasa pochodna)

`step(actions,realtime)`

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

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



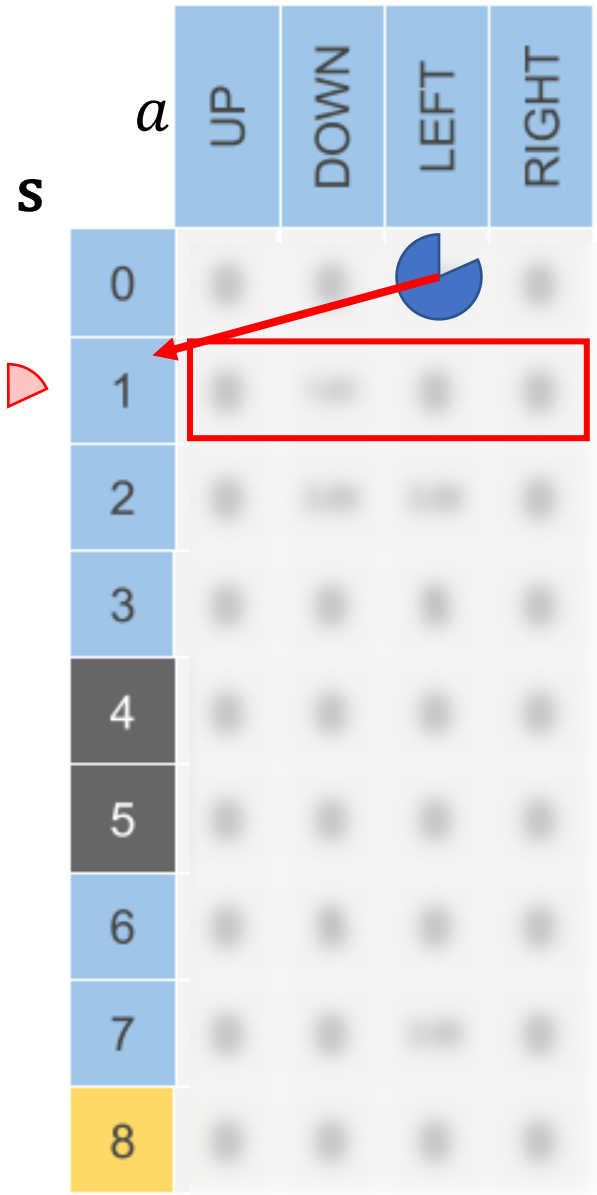
Składniki nagrody:

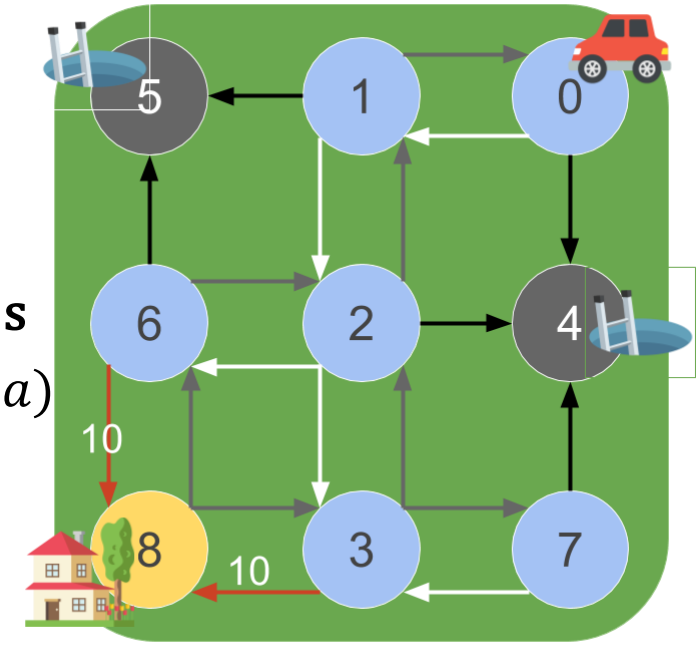
- kara proporcjonalna do przekroczenia prędkości
- nagroda proporcjonalna do przemieszczenia w zalecanym kierunku ruchu
- kara proporcjonalna do jazdy pod prąd

$* f_a$

- 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





$Q(\boldsymbol{s}, a)$ – ocena akcji a w stanie $\mathbf{s}$

$r(\mathbf{s}, a)$ – nagroda za akcję a w stanie $\mathbf{s}$

f – funkcja stanu obiektu, $\mathbf{s}' = f(\boldsymbol{s}, a)$

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

α – szybkość uczenia

γ – dyskonto (uwzględnianie przyszłych nagród)

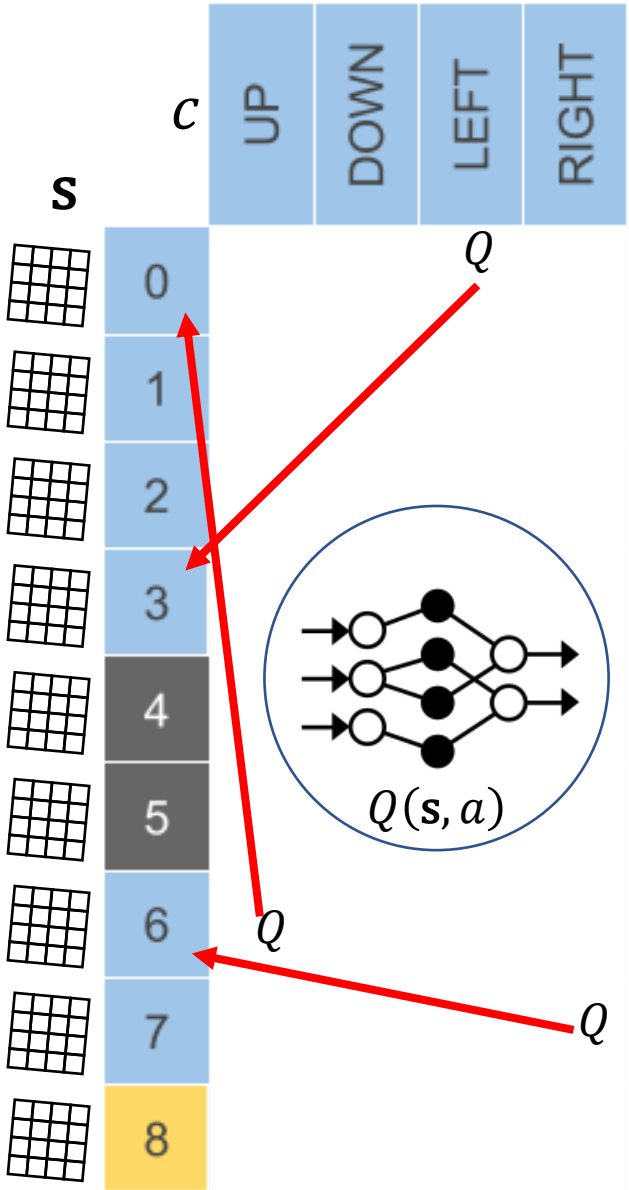
Funkcja Q ocenia każde możliwe sterowanie w każdym możliwym stanie. Jeśli stanów i sterowań jest mało, można (i należy) przechowywać jej wartości w tablicy.

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

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

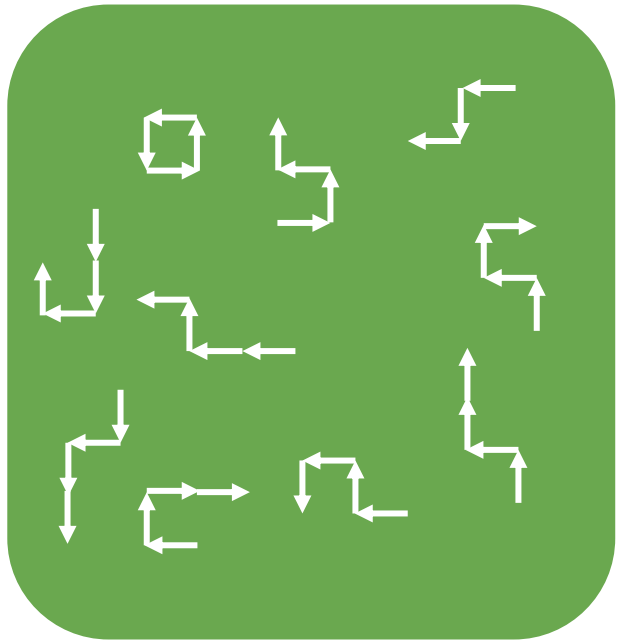
https://towardsdatascience.com/practical-reinforcement-learning-02-getting-started-with-q-learning-582f63e4acd9

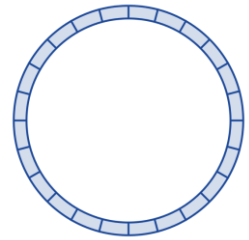
Uczenie się ze wzmocnieniem w ciągłej przestrzeni stanów = poszukiwanie $Q(\mathbf{s}, a)$



`DqnSingle.train_main()`

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



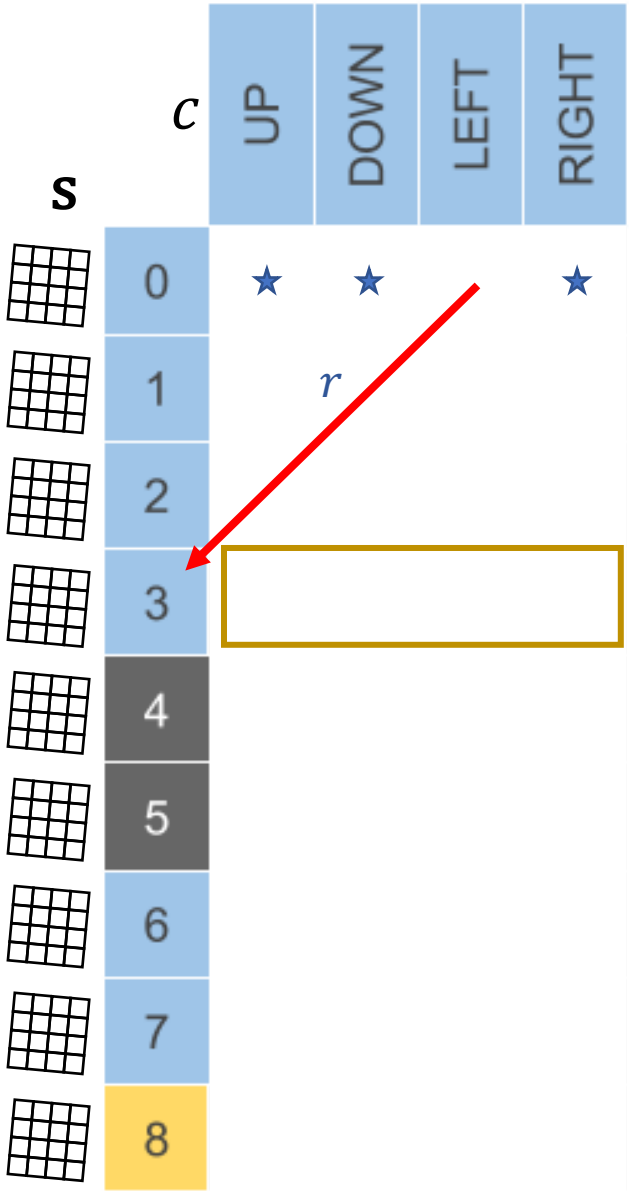


`DqnSingle.replay_memory=[(s,a,r,s'),…]`

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

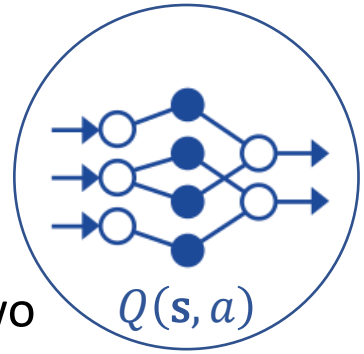
$$Q(\mathbf{s}, a) := (1 - \alpha) Q(\mathbf{s}, a) + \alpha \left[r(\mathbf{s}, a) + \gamma \max_{a'} Q(\mathbf{s}', a') \right]$$

Dwie sieci – doraźna i docelowa

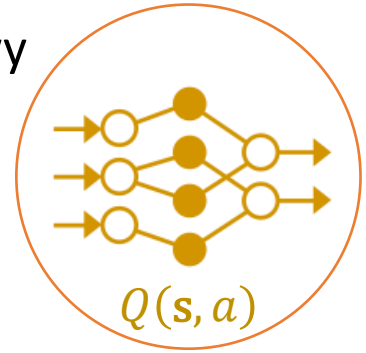


Strojenie (ulepszanie) funkcji $Q(\mathbf{s}, a)$
Powtarzaj:
- zrób kilka kroków, dodaj do historii
- wylosuj próbkę kroków z historii
- wyznacz $\max_{a'} Q(\mathbf{s}', a')$ dla każdego kroku z próbki
- wyznacz nowe pożądane wartości $Q(\mathbf{s}, a)$
- doucz sieć na wybranej próbce

sieć bieżąca



Okresowo aktualizuj model docelowy



sieć docelowa

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

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

Architektura sieci

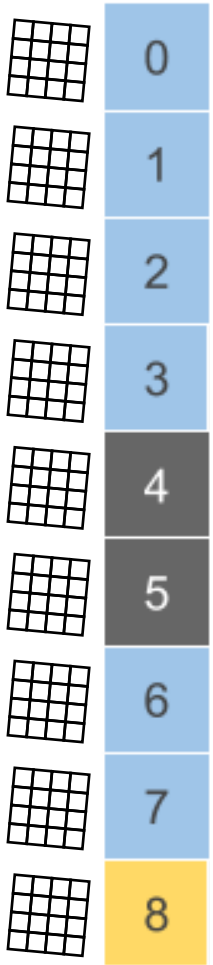
`action[0]` – wartość przesunięcia do przodu [m]

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

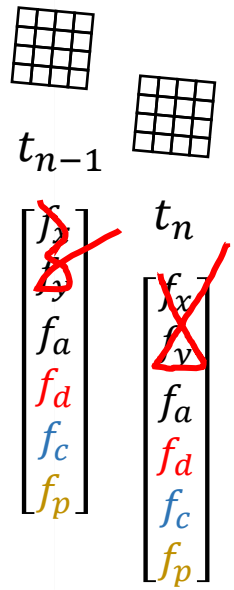
`DqnSingle.ctl2act()`

$c =$

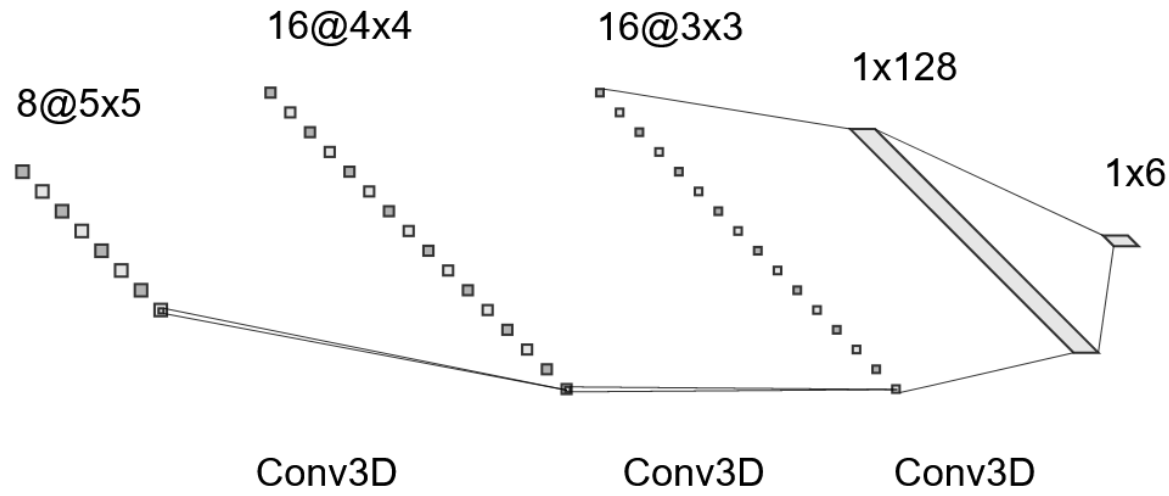
prędkość\skręt	-0,1 rad	0	0,1 rad
0,2 m/s	0	1	2
0,4 m/s	3	4	5



`DqnSingle.inp_stack()`



`DqnSingle.make_model()`



http://alexlenail.me/NN-SVG/LeNet.html

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 2

Zadania:
- uzupełnij kod w klasach (`# TODO STUDENCI…`)
- wytrenuj sieć na własnej planszy
- popraw wyniki, zmieniając co najmniej 2 parametry klas środowiska i klasy uczącej, i co najmniej 1 parametr lub strukturę sieci neuronowej

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