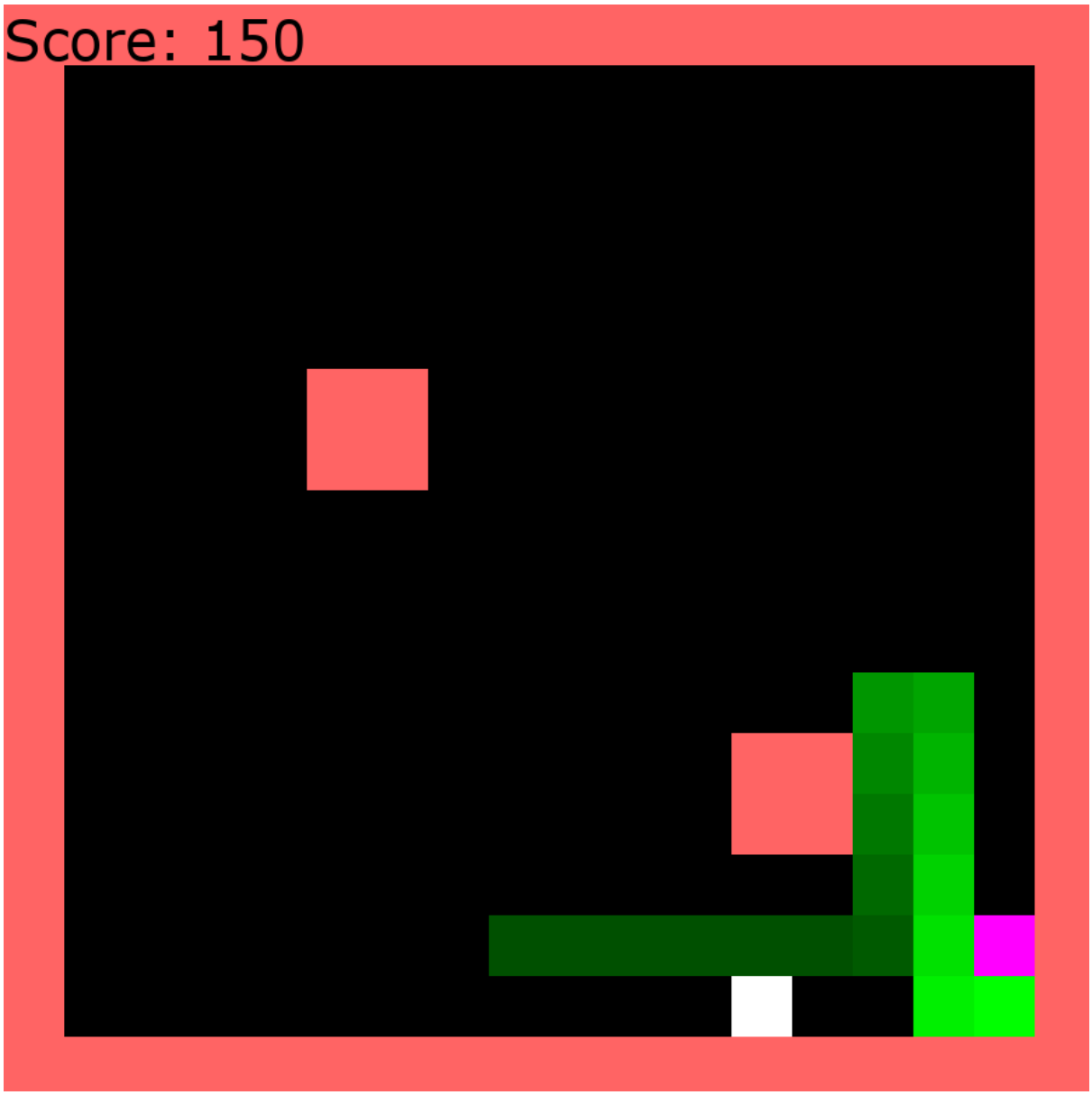
Neuroninio tinklo apmokymas žaisti žaidimą „gyvatėlė“



Darbą atliko – Matas Valiūnas

Galutinis tikslas – gyvatėlė pasiekia 500 taškų per žaidimą, kas reikštų kad ji turi suvalgyti vaisių 100 kartų.

1. Žaidimo aplinkos paruošimas

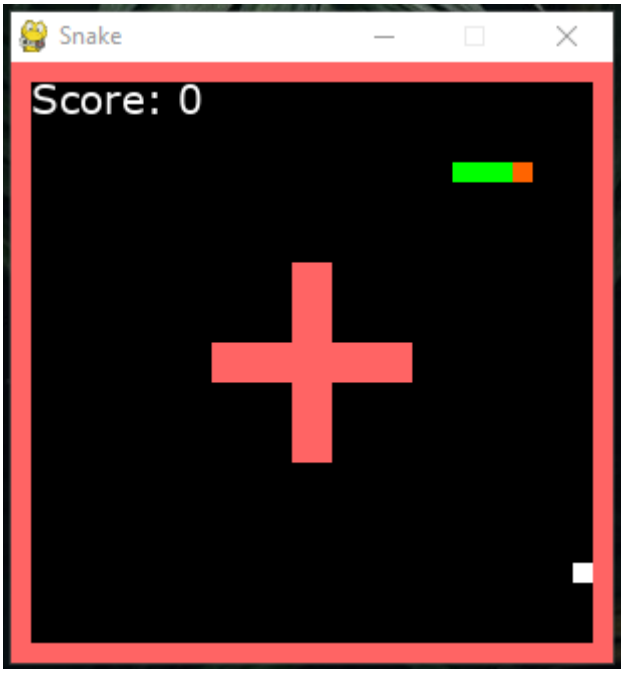
Gyvatėlės žaidimas buvo pasiskolintas iš svetainės *GeeksforGeeks*. Pradžioje jis atrodė taip:

Kodas yra labai paprastas ir ganėtinai neoptimizuotas, kas yra labai svarbu neuroninių tinklų apmokymui.

Pagrindiniai pakeitimai, kurie buvo atlikti šiam kodui:

- Sukurtos išorinės ir 2 sienos per vidury, į kurias atsitrenkus baigiasi žaidimas
- Žaidimo dydis padarytas santykiu 1:1 dėl paprastumo
- Gyvatėlės galva išskirta kita spalva
- Sukurta funkcija vaisiaus atsiradimui, kad būtų tikrinama kad jis neatsirastų sienoje arba pačioje gyvatėje
- Optimizuotas kodas
- Pralaimėjus žaidimą, jis iškart užsidaro (vietoj to, kad būtų rodoma pralaimėjimo langas)
- Kiti smulkūs pakeitimai (pakeistas šriftas ir dydis, lango pavadinimas ir t.t.)

Po pakeitimų žaidimas atrodė taip:



2. Agento aplinkos paruošimas

Agento aplinkai paruošti buvo naudojama ***Gymnasium*** biblioteka. Ši biblioteka padeda paruošti žaidimą neuroninio tinklo apmokymui.

Perkėlus žaidimą ir sukūrus papildomą logiką, kodas atrodo taip:

- Importuojamos reikalingos bibliotekos:

```
import gymnasium as gym
from gymnasium import spaces
import numpy as np
import pygame
```

- Klasė (paveldėta iš *gym.Env* klasės) ir jos kintamieji:

```
class SnakeEnv(gym.Env):
    _scale = 50                 # The size of each grid cell in pixels
    _snake_speed = 50           # The speed of the snake (FPS)
```

_scale kintamasis nurodo žaidimo atvaizdavimo dydį.
_snake_speed kintamasis nurodo gyvatės greitį (kiek kartų per sekundę pajudės gyvatė)

- Inicializavimo funkcija:

```
def __init__(self, render_mode=False, size=18):
        self.render_mode = render_mode          # The render mode
        self.size = size                        # The size of the square grid
        self.window_size = size * self._scale   # The size of the PyGame window
```

__init__ funkcija priima argumentus ***render_mode***, kuris nurodo ar aplinka turi būti atvaizduojama, ir ***size***, kuris nurodo aplinkos dydį.

```
# The observation space is a box of 8 elements:
[left_obst, up_obst, right_obst, down_obst, food_left, food_up, food_right, food_down]
        self.observation_space = spaces.Box(low=0, high=1, shape=(8,), dtype=bool)

        # The action space is a discrete space of 4 elements: left, up, right, down
        self.action_space = spaces.Discrete(4)
        self._action_to_direction = {
            0: np.array([-1, 0]),    # Left
            1: np.array([0, -1]),    # Up
            2: np.array([1, 0]),     # Right
            3: np.array([0, 1])      # Down
        }

        if self.render_mode:
            pygame.init()
```

```
            pygame.display.set_caption('Snake')
            self.window = pygame.display.set_mode((self.window_size, self.window_size))

        self._walls = [pygame.Rect(0, 0, size, 1),                # Top wall
                       pygame.Rect(0, size - 1, size, 1),         # Bottom wall
                       pygame.Rect(0, 0, 1, size),                # Left wall
                       pygame.Rect(size - 1, 0, 1, size),         # Right wall
                       pygame.Rect(0, 0, 0, 0),                   # Random wall #1
                       pygame.Rect(0, 0, 0, 0)]                   # Random wall #2
```

self.observation_space kintamasis nurodo kintamuosius, kurie bus perduodami neuroniniui tinklui kaip įvestis (apie tai bus kalbama plačiau ***_observation*** funkcijoje), nurodomos minimalios (*0*) ir maksimalios (*1*) jų vertės, dydis (*8*) ir tipas (*Bool*).

self.action_space kintamasis nurodo kiek yra galimų neuroninio tinklo išvesčių (*4*).

self._action_to_direction žodynas nurodo galimas neuroninio tinklo išvestis (kairė, viršus, dešinė, apačia). Tai atvaizduojama per *numpy* masyvus, kadangi pagal juos galima daug greičiau ir efektyviau pakeisti gyvatėlės poziciją.

Jei ***self.render_mode*** kintamasis yra *True*, tuomet inicializuojamas pygame langas.

Self._walls masyvas laiko *pygame.Rect* tipo objektus ir tai yra sienos.

- Susidūrimo aptikimo funkcija:

```
# Function to check if xy collides with walls (or snake)
def _collision(self, xy, ignore_head=True, ignore_snake=False):
    for wall in self._walls:
        if wall.colliderect(pygame.Rect(xy, (1, 1))):
            return True

    if not ignore_snake:
        if ignore_head:
            for block in self._snake_body[1:]:
                if np.array_equal(xy, block):
                    return True
        else:
            for block in self._snake_body:
                if np.array_equal(xy, block):
                    return True
    return False
```

_collision funkcija aptinka ar xy (numpy masyvo objektas) susiduria su sienomis ir/ar gyvatėle ir/ar jos galva pagal tai ar ***ignore_head*** ir ***ignore_snake*** yra *True* ar *False*.

- Naujos tuščios pozicijos radimo funkcija:

```
# Function to generate a new position which does not collide with walls (or snake)
def _new_position(self, ignore_snake=False):
    while True:
        xy = np.random.randint(1, self.size, size=(2,))

        if not self._collision(xy, ignore_head=False,  ignore_snake=ignore_snake):
            return xy
```

_new_position funkcija priima argumentą ar ignoruoti gyvatę ieškant naujos tuščios pozicijos ir randa naują tuščią poziciją. *while* ciklas sukasi kol randamos atsitiktinės koordinatės (***xy***), kurios nesiduria su sienomis ir gyvate (pagal nurodymus).

- Naujos gyvatėlės sudarymo funkcija:

```
# Function to generate a new snake which does not collide with walls
def _new_snake(self, lenght=4):
    # Generates coordinates for snake's head
    self._snake_location = self._new_position(ignore_snake=True)
    # Assigns snake's head to snake's body
    self._snake_body = np.array([self._snake_location.copy()])

    # Possible directions (left, up, right, down)
    directions = np.array([[-1, 0], [0, -1], [1, 0], [0, 1]])

    for i in range(0, lenght-1):
        # Adds a new block to snake's body in direction if it does not collide with walls or itself
        for direction in directions:
            if not self._collision(self._snake_body[i] + direction, ignore_head=False):
                self._snake_body = np.vstack([self._snake_body, self._snake_body[i] + direction])
                break
        else:
            break
```

_new_position funkcija sugeneruoja naują gyvatę ir priima gyvatės ilgį kaip argumentą. Pradžioje sugeneruojama nauja gyvatės galvos pozicija (***self._snake_location***) ir priskiriama gyvatės kūnui (***self._snake_body***). Paskui sukamas *for* ciklas (tiek kartų, koks nurodytas gyvatės ilgis), kuris suka kitą *for* ciklą kuris eina per galimas kryptis (***directions***) ir tikrina ar ten laisva vieta. Jei laisva, tuomet tą vietą priskiria gyvatės kūnui.

- Stebėjimų funkcija:

```
# Function to get the observations
def _observation(self):
    left_obst =  self._collision([self._snake_location[0] - 1, self._snake_location[1]])
    up_obst = self._collision([self._snake_location[0], self._snake_location[1] - 1])
    right_obst = self._collision([self._snake_location[0] + 1, self._snake_location[1]])
    down_obst = self._collision([self._snake_location[0], self._snake_location[1] + 1])

    food_left = self._target_location[0] < self._snake_location[0]
    food_up = self._target_location[1] < self._snake_location[1]
    food_right = self._target_location[0] > self._snake_location[0]
    food_down = self._target_location[1] > self._snake_location[1]

    return [left_obst, up_obst, right_obst, down_obst, food_left, food_up, food_right, food_down]
```

_observation funkcijos reikšmė yra surinkti kokius duomenis perduoti neuroniniui tinklui kaip įvestį. Šiuo metu ši funkcija grąžina 8 *Bool* dydžio masyvą: 4 vertės nurodančios ar kairėj, viršuj, dešinėj, apačioj yra kliūtis ir 4 vertės nurodančios ar maistas yra kairėj, viršuj, dešinėj, apačioj.

- Perkrovimo funkcija:

```
# Function to reset (or start) the environment
def reset(self):
    self._score = 0

    # Generates 2 random walls
    self._walls[-2] = pygame.Rect(np.random.randint(2, self.size-3, size=(2,)), (2, 2))
    self._walls[-1] = pygame.Rect(np.random.randint(2, self.size-3, size=(2,)), (2, 2))

    self._new_snake()

    self._target_location = self._new_position()

    return self._observation(), {}
```

reset funkcija suveikia kiekvieną kartą, kai gyvatėlė miršta ir aplinkos kintamieji nustatomi iš naujo arba kai naujai sukuriama aplinka.

self.score kintamasis nurodo gyvatėlės taškus ir jam priskiriamas 0.

Atsitiktinai sugeneruojamos dvi 2 × 2 dydžio sienos ir priskiriamos sienų masyvui (***self._walls***).

Sugeneruojama nauja gyvatė su ***self._new_snake*** funkcija ir sugeneruojama nauja vaisiaus pozicija su ***self._new_position*** funkcija.

reset funkcija grąžina (perduoda neuroniniam tinklui) stebėjimus ir tuščią sąrašą (galima papildoma informacija, bet šiuo atveju viskas perduodama per ***_self.observation*** funkciją).

- Žingsnio funkcija:

```
def step(self, action):
    terminated = False      # Episode termination
    reward = 1              # Episode reward

    # Applying direction given by the model
    self._snake_location += self._action_to_direction[action]

    # Elongating the snake
    self._snake_body = np.insert(self._snake_body, 0, self._snake_location, axis=0)
    if np.array_equal(self._snake_location, self._target_location):
        self._target_location = self._new_position()
        reward = 50
        self._score += 10
    else:
        self._snake_body = np.delete(self._snake_body, -1, axis=0)

    # Checking if the snake has hit a wall or itself
    if self._collision(self._snake_location):
        terminated = True
        reward = -50

    if self.render_mode:
        self.render()

    return self._observation(), reward, terminated, False, {}
```

step funkcija yra vykdoma kiekvieną episodą (kadrą), per kurį gyvatėlė pajuda 1 atstumą. Ji turi 1 argumentą ***action***, kuris yra neuroninio tinklo išvestis (veiksmas, kurį atlieka gyvatėlė).

terminated kintamasis nurodo ar episodas baigiasi gyvatėlės mirtim.

reward nurodo episodo atlygį. Jei gyvatėlė suvalgo vaisių atlygis yra 50, jei miršta atlygis -50 ir jei neįvyksta niekas, tuomet atlygis yra 1 (už tai kad išgyveno).

Pradžioje gyvatėlės pozicijai (***self._snake_location***) pritaikomas neuroninio tinklo paduotas veiksmas (***action***). Paskui gyvatėlės kūnas (***self._snake_body***) yra prailginamas 1 bloku, kuriuo pajudėjo gyvatėlė. Jei gyvatėlė suvalgė vaisių, tuomet sugeneruojama nauja vaisiaus pozicija, skiriamas atlygis (***reward***) ir pridedama taškų (***self._score***), priešingu atveju iš gyvatėlės kūno atimamas paskutinis elementas.

Jei gyvatėlė susiduria su siena ar savim, ***terminated*** kintamasis nustatomas kaip *True* ir skiriama nuobauda. Taip baigiasi vienos gyvatėlės iteracija.

Jei ***self.render_mode*** kintamasis yra *True*, tuomet kadras yra atvaizduojamas (paprastai kadrai neatvaizduojami jei vyksta neuroninio tinklo mokymas, kadangi tai užimtų daugiau laiko).

step funkcija grąžina 5 kintamuosius:

1) Stebėjimus (***self._observation***)
2) Atlygį (***reward***)
3) Ar iteracija baigiasi (***terminated***)
4) Ar agentas užstrigęs (šiuo metu visada grąžinama *False*)
5) Papildoma informacija (kaip buvo minėta – viskas paduodama per stebėjimus)

- Atvaizdavimo funkcija

```
def render(self):
    self.window.fill(pygame.Color(0, 0, 0))

    # Drawing walls
    for wall in self._walls:
        pygame.draw.rect(self.window, (255, 100, 100),
                         pygame.Rect(wall.x * self._scale, wall.y * self._scale,
                         wall.width * self._scale, wall.height * self._scale))

    # Drawing snake
    green = 255
    for pos in self._snake_body[1:]:
        pygame.draw.rect(self.window, pygame.Color(0, green, 0), pygame.Rect(pos * self._scale,
(self._scale, self._scale)))
        green = max(80, green - 15)
    pygame.draw.rect(self.window, pygame.Color(255, 0, 255), pygame.Rect(self._snake_location *
self._scale, (self._scale, self._scale)))

    # Drawing fruit
    pygame.draw.rect(self.window, pygame.Color(255, 255, 255), pygame.Rect(self._target_location
* self._scale, (self._scale, self._scale)))

    # creating font object score_font
    score_font = pygame.font.SysFont('verdana', int(self._scale * 0.9))
    # create the display surface object score_surface
    score_surface = score_font.render('Score: ' + str(self._score), True, pygame.Color(0, 0, 0))
    # displaying text
    self.window.blit(score_surface, (0, 0, 0, 0))

    # Updating the window
    pygame.display.update()
    pygame.time.Clock().tick(self._snake_speed)
```

render funkcija atvaizduoja kadrą.

Pradžioje kadras užpildomas juoda spalva. Vėliau atvaizduojamos sienos (***self._walls***), gyvatėlė (***self._snake_body***) ir vaisius (***self._targer_location***), kurių dydis yra dauginamas iš ***self._scale***. Galiausiai atvaizduojami taškai ir atnaujinamas kadras.

- Uždarymo funkcija

```
def close(self):
    if self.window is not None:
        pygame.display.quit()
        pygame.quit()
```

Ši funkcija uždaro pygame langą jei jis yra atvaizduojamas. Šią funkciją įvykdo neuroninis tinklas, jei viskas baigta.

3. Apmokymas

Šis etapas užėmė daugiausiai laiko, kadangi buvo ieškoma biblioteka, kuri tinkamai veiktų su ***Gymnasium*** biblioteka.

Pradžioje bandyta dirbti su ***keras-rl*** ir ***keras-rl2*** bibliotekomis, bet ji buvo nesuderinama tiek su ***Gymnasium*** biblioteka, tiek su kitomis mano turimomis bibliotekų versijomis. Vėliau buvo bandytos kelios paprastos ***keras*** bibliotekos implementacijos, bet arba joms nepavyko teisingai perduoti duomenų, arba mokymas nevyko teisingai. Galiausiai buvo naudojama ***tianshou*** biblioteka, kurią pavyko teisingai implementuoti:

- Importuojamos bibliotekos:

```
import torch
from torch.utils.tensorboard import SummaryWriter
import tianshou as ts
from tianshou.utils.net.common import Net

from SnakeEnv import SnakeEnv
```

Importuojamos reikalingos bibliotekos ir prieš tai minėta ***SnakeEnv*** klasė.

- Hiperparametrai:

```
# Hyperparameters
lr = 1e-3
epoch = 1000
batch_size = 128
gamma = 0.9
n_step = 3
target_freq = 320
buffer_size = 20000
train_num, test_num = 1000, 50
eps_train, eps_test = 0.1, 0.01
step_per_epoch = 100
step_per_collect = 10
reward_threshold = 2000
```

- **lr** (*learning rate*) – mokymosi greitis.
- **epoch** – maksimalus epochų skaičius.
- **batch_size** – paketų skaičius, nusakantis kiek perėjimų iš vieno kadro į kitą yra saugoma atmintyje.
- **gamma** – atlygio nuolaidos koeficientas (skaičius, nusakantis koks svarbus yra)
- **n_step** – skaičius, nusakantis po kas kiek skirtingų aplinkų pabaigos įvyksta tinklo atnaujinimas.
- **targer_freq** – skaičius, nusakantis kaip dažnai atnaujinamas neuroninis tinklas.

- **buffer_size** – atmintyje saugomų episodų skaičius į kuriuos atsižvelgus vyksta neuroninio tinklo mokymasis.
- **train_num, test_num** – skaičius, nusakantis kiek skirtingų aplinkų yra sukuriama treniravimuisi ir testavimui.
- **eps_train, eps_test** – treniravimo ir testavimo epsilon-godumas (tikimybė atlikti atsitiktinį veiksmą).
- **step_per_epoch** – maksimalus iteracijų per epochą skaičius.
- **step_per_collect** – veiksmų, kuriuos agentas atlieka prieš surinkdamas patirtį į pakartojimo buferį (***batch_size***), skaičius.
- **reward_threshold** – atlygis, kurį pasiekus neuroninio tinklo mokymas stabdomas.

- Mokymo duomenų statistika

```
logger = ts.utils.TensorboardLogger(SummaryWriter('log/dqn'))
```

Renka tokius duomenis kaip iteracijų ilgius, atlygius ir t.t.

- Treniravimo ir testavimo aplinkos

```
train_envs = ts.env.DummyVectorEnv([lambda: SnakeEnv() for _ in range(train_num)])
test_envs = ts.env.DummyVectorEnv([lambda: SnakeEnv(render_mode=True) for _ in range(test_num)])
```

Sukuriama tiek treniravimo ir testavimo aplinkų, kiek buvo nurodyta hiperparametruose (***train_num*** ir ***test_num***). Treniravimo aplinkos nėra atvaizduojamos, kadangi jų yra labai daug, bet testavimo aplinką patogu atvaizduoti (***render_mode=True***), jei testavimą atlieka tik viena aplinka (***test_num***). Po kiekvieną epochą vyksta treniravimas ir paskui – testavimas. Taip yra patogu pažiūrėti kokį progresą atlieka neuroninis tinklas.

- Neuroninis tinklas

```
env = SnakeEnv(render_mode=True)

state_shape = env.observation_space.shape or env.observation_space.n
action_shape = env.action_space.shape or env.action_space.n

net = Net(state_shape=state_shape, action_shape=action_shape, hidden_sizes=[64, 64])
```

Sukuriama nauja aplinka (***env***), iš jos gaunami stebėjimų (***state_shape***) ir veiksmų (***action_shape***) dydžiai. Šie dydžiai yra atitinkamai perduodami neuroniniui tinklui kaip įvesties ir išvesties dydžiai. Šiame neuroniniame tinkle yra 2 paslėpti neuroniniai sluoksniai po 64 neuronus, todėl neuroninio tinklo forma yra tokia – 8, 64, 64, 4.

- Strategija ir kolektoriai

```
optim = torch.optim.Adam(net.parameters(), lr=lr)

policy = ts.policy.DQNPolicy(net, optim, gamma, n_step, target_update_freq=target_freq)

train_collector = ts.data.Collector(policy, train_envs,
ts.data.VectorReplayBuffer(buffer_size, train_num), exploration_noise=True)
test_collector = ts.data.Collector(policy, test_envs, exploration_noise=True)
```

optim yra optimizavimo funkcija, kuri šiuo atveju yra *Adam.*

policy yra strategija, atsakinga už agento pasirenkamus veiksmus ir neuroninio tinklo modelio atnaujinimą.

train_collector ir ***test_collector*** yra kolektoriai, generuojantys duomenis treniravimui ir testavimui, pagal kuriuos atnaujinamas neuroninio tinklo modelis.

- Modelio treniravimas

```
result = ts.trainer.offpolicy_trainer(
    policy, train_collector, test_collector, epoch, step_per_epoch, step_per_collect,
    test_num, batch_size, update_per_step=1/step_per_collect,
    train_fn=lambda epoch, env_step: policy.set_eps(eps_train),
    test_fn=lambda epoch, env_step: policy.set_eps(eps_test),
    stop_fn=lambda mean_rewards: mean_rewards >= reward_threshold,
    logger=logger)
print(f'Finished training! Use {result["duration"]}')
```

ts.trainer.offpolicy_trainer funkcija vykdo treniravimą pagal visus jai duotus parametrus. Treniravimas nutraukiamas kai pasiekiamas nustatytas atlygis (***reward_threshold***).

- Modelio išsaugojimas ir paruošimas įvertinimui

```
torch.save(policy.state_dict(), 'dqn.pth')
policy.eval()
```

- Atvaizduojamas galutinis rezultatas

```
env = ts.env.DummyVectorEnv([lambda: env])
collector = ts.data.Collector(policy, env)
collector.collect(n_episode=10, render=1/60)
```

env aplinka paruošiama atvaizdavimui ir 10 kartų (***n_episode***) atvaizduojama galutinė neuroninio tinklo versija.

4. Eksperimentavimas

Po kiekvieno pilno neuroninio tinklo treniravimo vyko 10 bandymų ir jų rezultatai buvo užrašomi lentelėse. Kiekvienam testavimui neuroninis tinklas buvo treniruojamas 3 skirtingus kartus dėl patikimumo.

Lentelėse **paryškinti** rezultatai reiškia aukščiausią to treniravimo bandymo rezultatą, o rezultatai su raudonu šriftu reiškia, kad gyvatėlė užstrigo amžinam cikle gaudydama savo uodegą ir tą testavimą teko nutraukti paliekant surinktus taškus. Bendras viso eksperimento vidurkis yra **ir paryškintas, ir pabrauktas**.

Paprastas mokymas

Šiam testavimui niekas nebuvo pakeista nuo jau prieš tai aiškinto kodo:

- Hiperparametrai:

```
lr = 1e-3
epoch = 200
batch_size = 128
gamma = 0.9
n_step = 3
target_freq = 320
buffer_size = 20000
train_num, test_num = 1000, 10
eps_train, eps_test = 0.1, 0.01
step_per_epoch = 100
step_per_collect = 10
reward_threshold = 5000
```

- Neuronio tinklo forma – [8, 64, 64, 4]
- Neuroniniam įvesčių sluoksniui duodami duomenys:

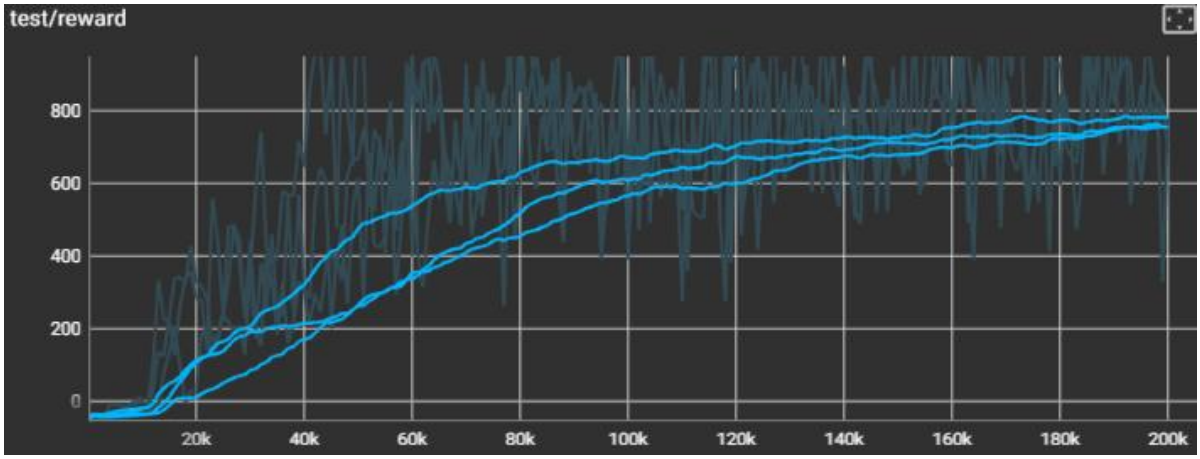
```
[left_obst, up_obst, right_obst, down_obst, food_left, food_up, food_right, food_down]
```

Rezultatai

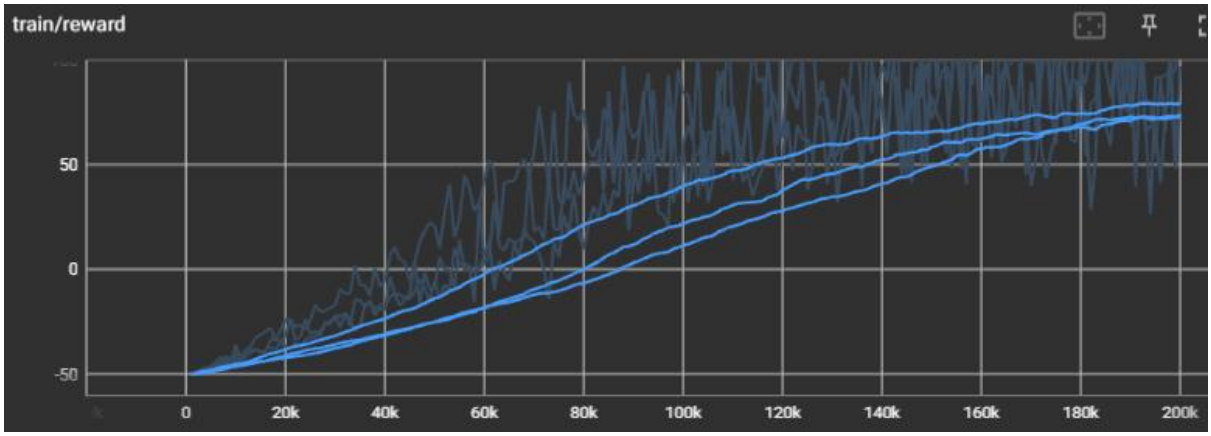
		Bandymai										Vidurkis
		#1	#2	#3	#4	#5	#6	#7	#8	#9	#10	
Treniravimai	#1	240	**350**	300	260	170	290	310	160	240	120	244
	#2	210	**320**	260	200	270	70	220	280	270	290	239
	#3	180	180	130	200	90	280	260	250	**320**	170	206
												230

Iš lentelės galima matyti, kad ištreniruota gyvatėlė vidutiniškai surinko 230 taškų. Beveik visais atvejais gyvatėlė užsispaudė save į spąstus ir dėl to mirdavo.

Toliau esančiose lentelėse atvaizduotos 3 treniravimo linijos, kuriose atvaizduoti testavimo rezultatai. X ašis atspindi atliktų žingsnių skaičių, o 1 epocha atitinka 1000 žingsnių.



Iš šios lentelės galima pastebėti, kad progresas matomas visu treniravimo metu, nors pabaigoje jis ganėtinai lėtėja. Visi 3 treniravimai pasiekė labai panašius rezultatus.



Eksperimentas #1

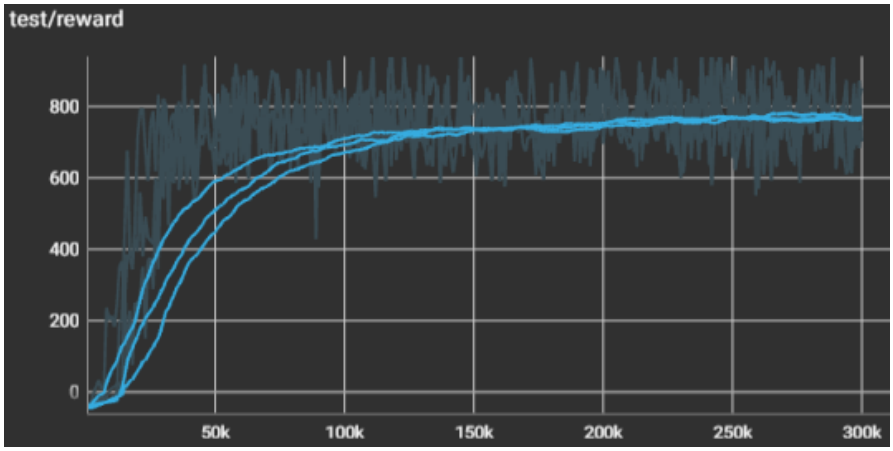
Atlikti pakeitimai:

- Padidintas maksimalus iteracijų skaičius per epochą (***step_per_epoch*** **100 → 300**), kad treniruojamos gyvatėlės geriau žinotų kaip elgtis kai jos yra ilgesnės.
- Padidintas epochų skaičius (***epoch*** **200 → 300**), kad būtų daugiau laiko progresui ir pastebėti kada neuroninis tinklas pasiekia savo potencialą su turimais hiperparametrais ir aplinka.
- Padidintas testavimo aplinkų skaičius (***test_num*** **10 → 50**) dėl tikslesnių duomenų analizės.

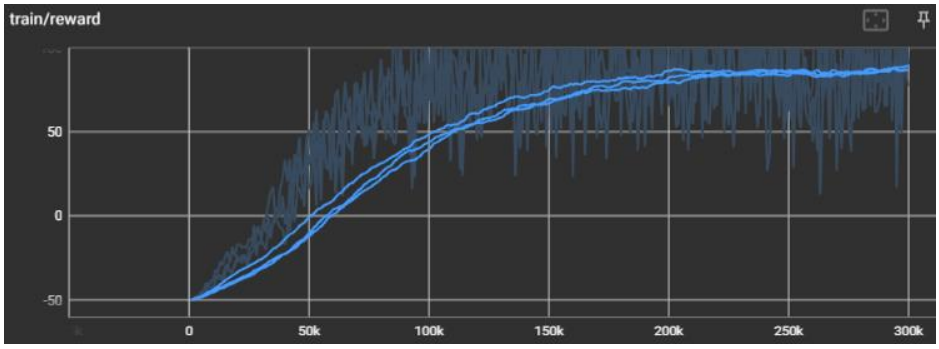
Rezultatai

		Bandymai										**Vidurkis**
		#1	#2	#3	#4	#5	#6	#7	#8	#9	#10	
Treniravimai	#1	30	80	70	310	220	30	110	**320**	180	160	151
	#2	60	320	300	350	30	280	210	160	190	**380**	228
	#3	220	160	70	110	280	270	**450**	60	210	210	204
												194

Iš lentelės galima pastebėti, kad gyvatėlė daug dažniau užstringa amžinai gaudydama savo uodegą. Tai turbūt gali įvykti dėl persimokymo. Taip pat, bendras taškų vidurkis nukrito 36 taškais. Iš kitos pusės, padidėjo rekordas iš 350 į 450.



Ties maždaug 200 epocha mokymasis labai smarkiai sulėtėjo.



Eksperimentas #2

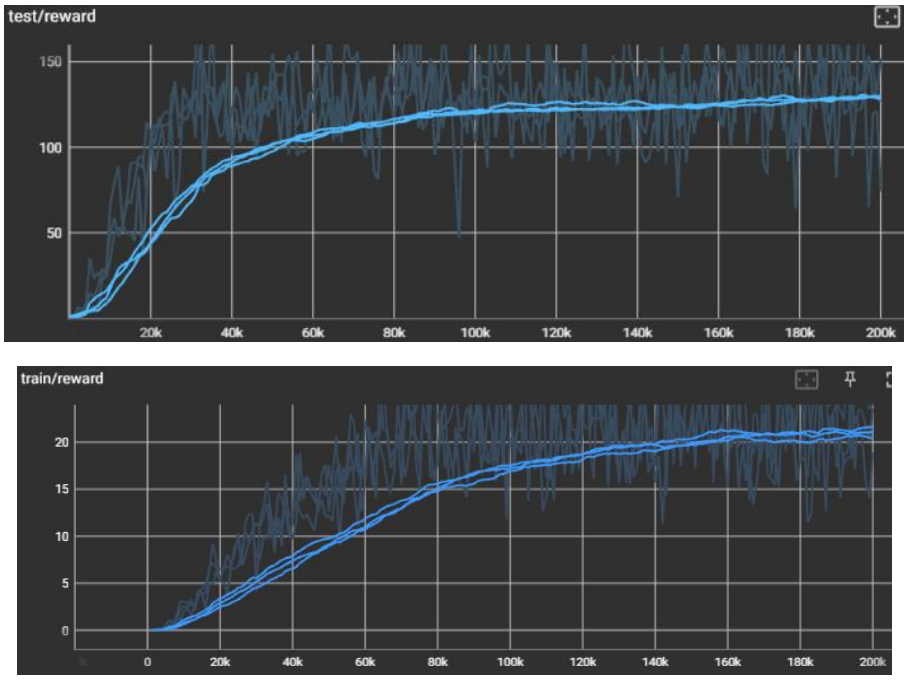
Atlikti pakeitimai:

- Kadangi pastebėta, kad nuo 200 epochos praktiškai nevyksta progresas ir neuroninis tinklas galimai tik persimoko (*overfitting*), todėl epochų skaičius vėl sumažintas iki 200 (***epoch* 300 → 200**).
- Taip pat dėl didelio kiekio gyvačių gaudančių savo uodegą pakeistos atlygio taisyklės:
    - Pralaimėjimas neatima atlygio (-50 → 0)
    - Suvalgytas vaisius duoda 10 taškų (50 → 10)
    - Sėkmingas pajudėjimas daugiau atlygio neduoda (1 → 0)
- Žingsnių per epochą skaičius sumažintas (***step_per_epoch* 300 → 150**).
- Padidintas **gamma**, nusakantis kaip agentas turi stengtis dėl ateities (***gamma* 0.9 → 0.95**).

Rezultatai

		Bandymai										Vidurkis
		#1	#2	#3	#4	#5	#6	#7	#8	#9	#10	
Treniravimai	#1	140	100	280	330	130	190	**410**	100	180	150	201
	#2	370	330	290	290	**480**	270	120	250	230	440	307
	#3	80	180	220	220	200	270	280	**300**	140	250	214
												241

Rezultatai rodo, kad atlikti pakeitimai turėjo teigiamą poveikį, kadangi taškų vidurkis padidėjo 11 taškų lyginant su originaliu mokymu ir 47 taškais lyginant su eksperimentu #1. Taip pat pasiektas naujas 480 taškų rekordas ir patirtas tik 1 atvejis kai gyvatė be pabaigos gaudė savo uodegą.



Kadangi buvo pakeista atlygio sistema, *test/reward* grafikai atitinka taškus.

Eksperimentas #3

Atlikti pakeitimai:

- Vietoje 4 galimų krypčių (į kairę, viršų, dešinę, apačią) dabar yra 3 galimos kryptys (į kairę, tiesiai, į dešinę). Tai panaikina galimybę gyvatėlei suvalgyti save, kas yra didelė problema treniravime, kai tinklas pasirenka atsitiktinį veiksmą pagal epsilon hiperparametrus (***eps_train***, ***eps_test***). Taip pat dėl šio keitimo neuroniniui tinklui reikia paduoti tik 7 parametrus, nei prieš tai buvusius 8, nes nereikia paduoti duomenų ar yra kliūtis už gyvatės.

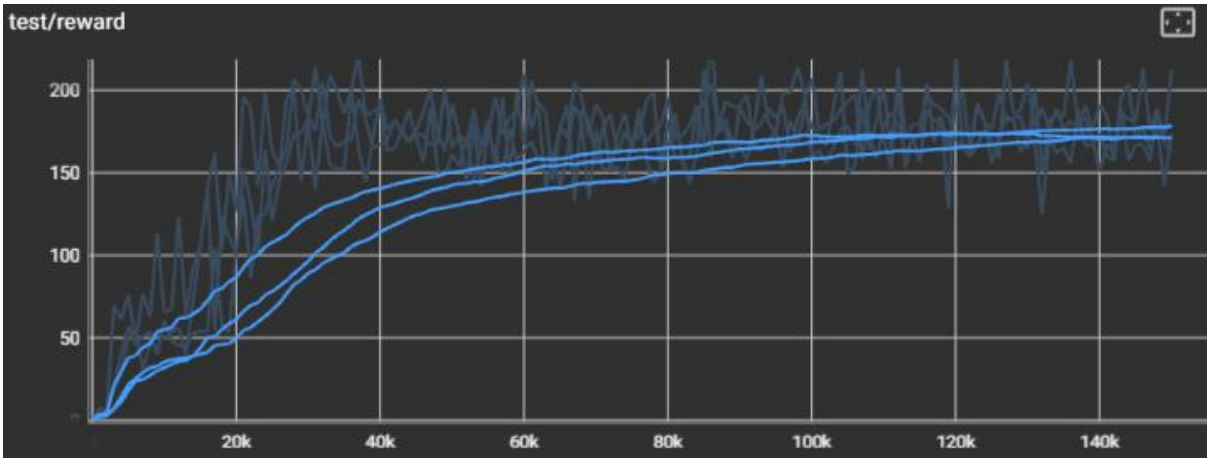
```
self.action_space = spaces.Discrete(3)
self._action_to_direction = {
    0: np.array([[0, 1], [-1, 0]]),   # Left
    1: np.array([[1, 0], [0, 1]]),    # Straight
    2: np.array([[0, -1], [1, 0]])    # Right
}
```

```
[obst_left, obst_straight, obst_right, food_left, food_straight, food_right, food_back]
```

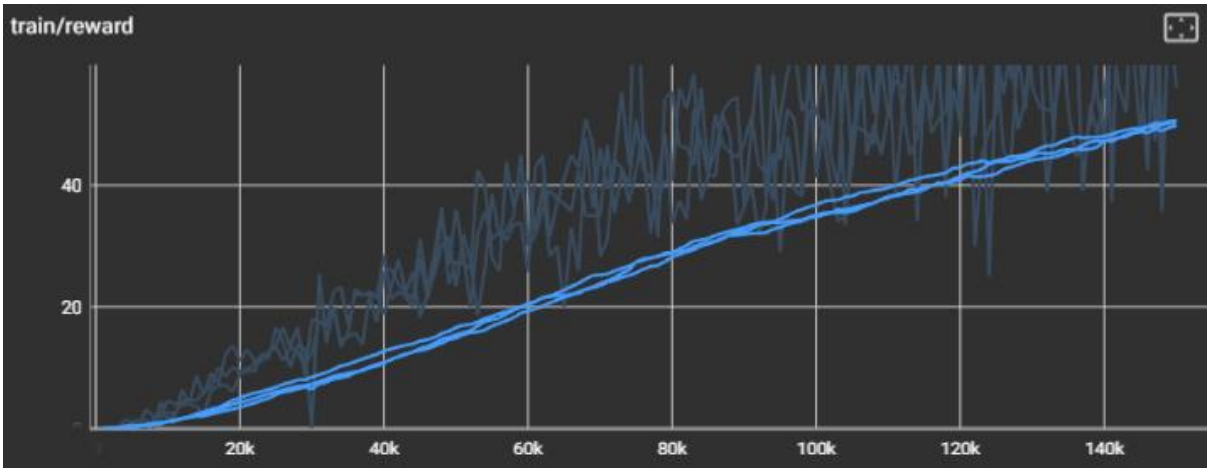
Rezultatai

		Bandymai										Vidurkis
		#1	#2	#3	#4	#5	#6	#7	#8	#9	#10	
Treniravimai	#1	230	240	210	190	60	110	**320**	210	140	130	184
	#2	**490**	250	270	290	380	210	120	220	440	160	283
	#3	260	130	30	380	90	180	220	200	180	220	189
												219

Iš rezultatų lentelės galima matyti, kad pasiektas naujas rekordas 490, kuris beveik pasiekė užsibrėžtą 500 taškų tikslą. Bendras vidurkis sumažėjo 32 taškais, bet manau, kad šio eksperimento pakeitimai turi geresnį šansą pasiekti norimus rezultatus. 1# treniravimo gyvatėlė parodė neįprastą elgesį – vietoje paprastai atliekamo 1 ar dviejų atliekamų posukių judant link maisto, ji darė posūkius kiekvieną ėjimą judėdama zigzagais ir tai nėra optimalus būdas šiame žaidime.



Lyginant su praeitu eksperimentu, testavimo atlygis vidutiniškai padidėjo nuo 130 iki 170 taškų.



Lyginant su praeitu eksperimentu, treniravimo atlygis vidutiniškai padidėjo nuo 22 iki 48 taškų.

Eksperimentas #4

Atlikti pakeitimai:

- Neuroniniui tinklui perduodami 3 papildomi duomenys:
  - 2 nauji kintamieji apie esamas kliūtis priekyje kairėje ir priekyje dešinėje
  - Kintamasis, nusakantis gyvatėlės ilgį:

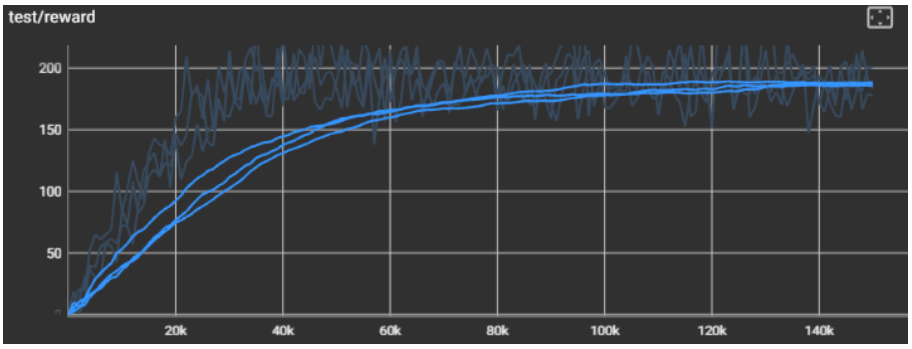
```
lenght = len(self._snake_body) / (self.size * (self.size - 4))
```

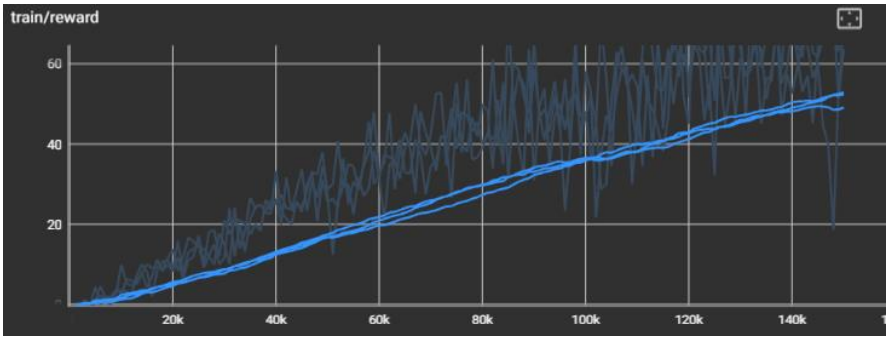
Kad ***lenght*** kintamasis būtų normalizuotas [0, 1], gyvatėlės ilgis padalinamas iš maksimalaus galimo gyvatėlės ilgio.

Rezultatai

		Bandymai										Vidurkis
		#1	#2	#3	#4	#5	#6	#7	#8	#9	#10	
Treniravimai	#1	320	350	280	270	150	**420**	140	360	200	210	270
	#2	170	160	**290**	230	200	170	250	140	160	220	199
	#3	170	190	120	150	230	110	180	**260**	200	**260**	187
												<u>219</u>

Rezultatų vidurkis išliko lygiai toks pats, bet pavyko visiškai sustabdyti nesibaigiančius ciklus kai gyvatėlė gaudo savo uodegą.





Tiek testavimo, tiek treniravimo rezultatai tapo šiek tiek geresni nei praeito eksperimento.

Eksperimentas #5

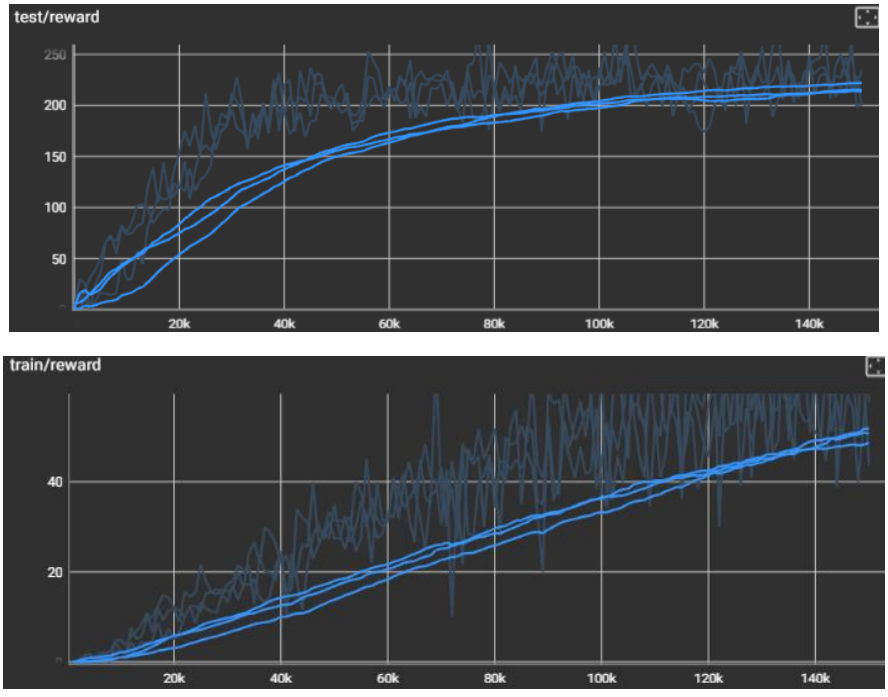
Atlikti pakeitimai:

- Neuroniniam tinklui perduodami 2 papildomi duomentys – jei prieš pat gyvatę yra kliūtis, tikrinama ar po kaire ir dešine yra jos kūnas. Tai gali išpręsti didžiausią problemą – savęs užsispendimą.

Rezultatai

		Bandymai										Vidurkis
		#1	#2	#3	#4	#5	#6	#7	#8	#9	#10	
Treniravimai	#1	370	320	290	330	320	340	270	**480**	240	330	329
	#2	340	250	160	180	200	210	140	210	**420**	340	245
	#3	**380**	220	270	360	200	290	300	170	360	310	286
												287

Bendras vidurkis išaugo net 68 taškais.



Testavimo atlygis vidutiniškai padidėjo apie 30 taškų, o treniravimo - išliko panašus.

Eksperimentas #6

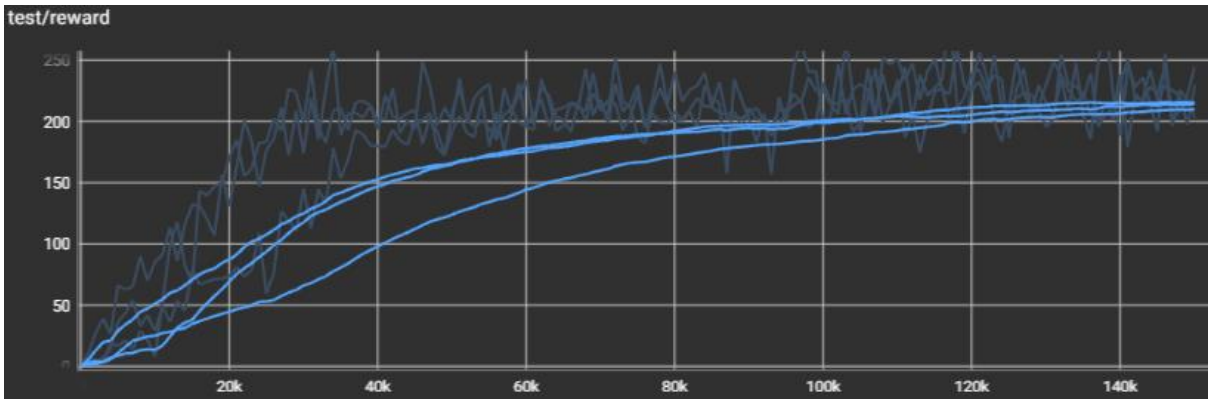
Atlikti pakeitimai:

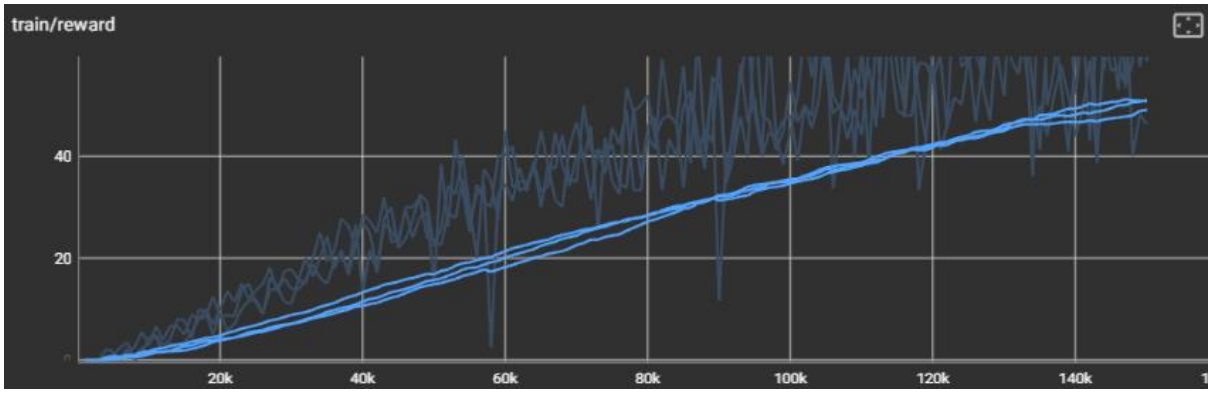
- Į neuroninį tinklą įterptas papildomas neuroninis sluoksnis (**[12, 64, 64, 3] → [12, 64, 128, 64, 3]**). Tai galimai padės modeliui priimti geresnius sprendimus.

Rezultatai

		Bandymai										Vidurkis
		#1	#2	#3	#4	#5	#6	#7	#8	#9	#10	
Treniravimai	#1	510	310	170	200	**670**	370	320	250	390	510	370
	#2	330	220	210	270	170	490	290	**510**	**510**	100	310
	#3	340	300	340	390	290	360	200	380	190	**410**	320
												333

Bendras taškų vidurkis perkopė 300 ribą ir išaugo 46 taškais. Pasiektas naujas 670 rekordas, kuris gerokai viršijo užsibrėžtą 500 taškų ribą.





Testavimo ir treniravimo atlygiai išliko panašūs.

Eksperimentas #7

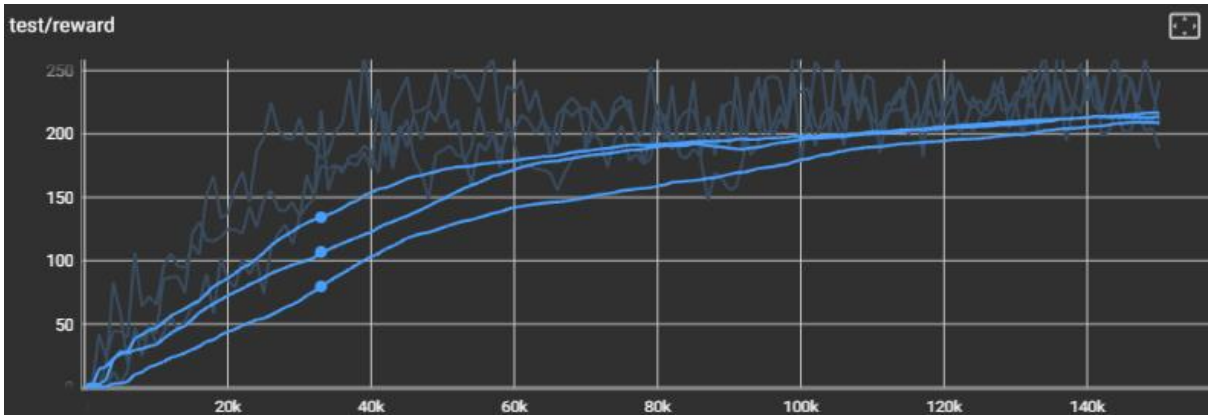
Atlikti pakeitimai:

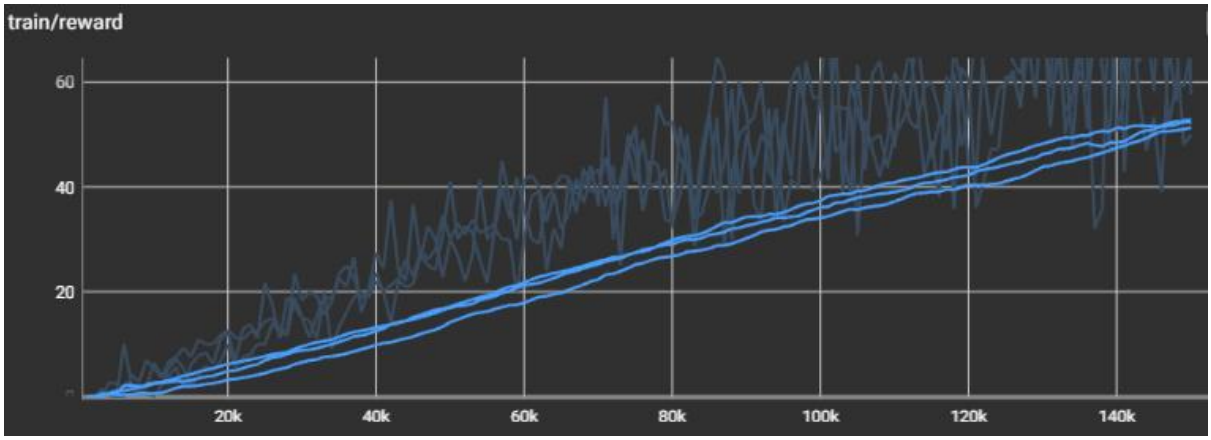
- Padidintas neuronų skaičius neuroniniame tinkle (**[12, 64, 128, 64, 3] → [12, 128, 256, 128, 3]**). Tai galimai padės modeliui priimti dar geresnius sprendimus.

Rezultatai

		Bandymai										Vidurkis
		#1	#2	#3	#4	#5	#6	#7	#8	#9	#10	
Treniravimai	#1	100	**430**	170	300	260	370	40	260	290	250	247
	#2	190	320	220	290	**360**	270	240	70	340	310	261
	#3	230	210	330	300	360	250	360	**370**	350	350	311
												<u>273</u>

Iš rezultatų matosi, kad bendras vidurkis nukrito 60 taškų.





Išvados

Geriausius rezultatus pasiekė eksperimentas #6, kuriam buvo atlikti tokie pakeitimai lyginant su orginaliu modeliu:

- Padidintas maksimalus iteracijų skaičius per epochą (***step_per_epoch*** **100 → 150**)
- Pamažintas epochų skaičius (***epoch*** **200 → 150**)
- Pakeisti atlygiai
- Padidintas **gamma**, nusakantis kaip agentas turi stengtis dėl ateities (***gamma*** **0.9 → 0.95**)
- Vietoje 4 galimų krypčių (į kairę, viršų, dešinę, apačią) padarytos 3 galimos kryptys (į kairę, tiesiai, į dešinę)
- Duodama papildoma informacija neuroniniam tinklui
- Į neuroninį tinklą įterptas papildomas neuroninis sluoksnis (**[12, 64, 64, 3] → [12, 64, 128, 64, 3]**)

Šis modelis ne tik pasiekė 500 taškų ribą, bet ir ganėtinai ją viršijo iš 10 bandymų surinkęs 670 taškų rekordą.

Galimas patobulinimas būtų pradžioje įterpti konvulucinius sluoksnius, paduoti visos informacijos paprastai nėra praktiška ar net logiška.