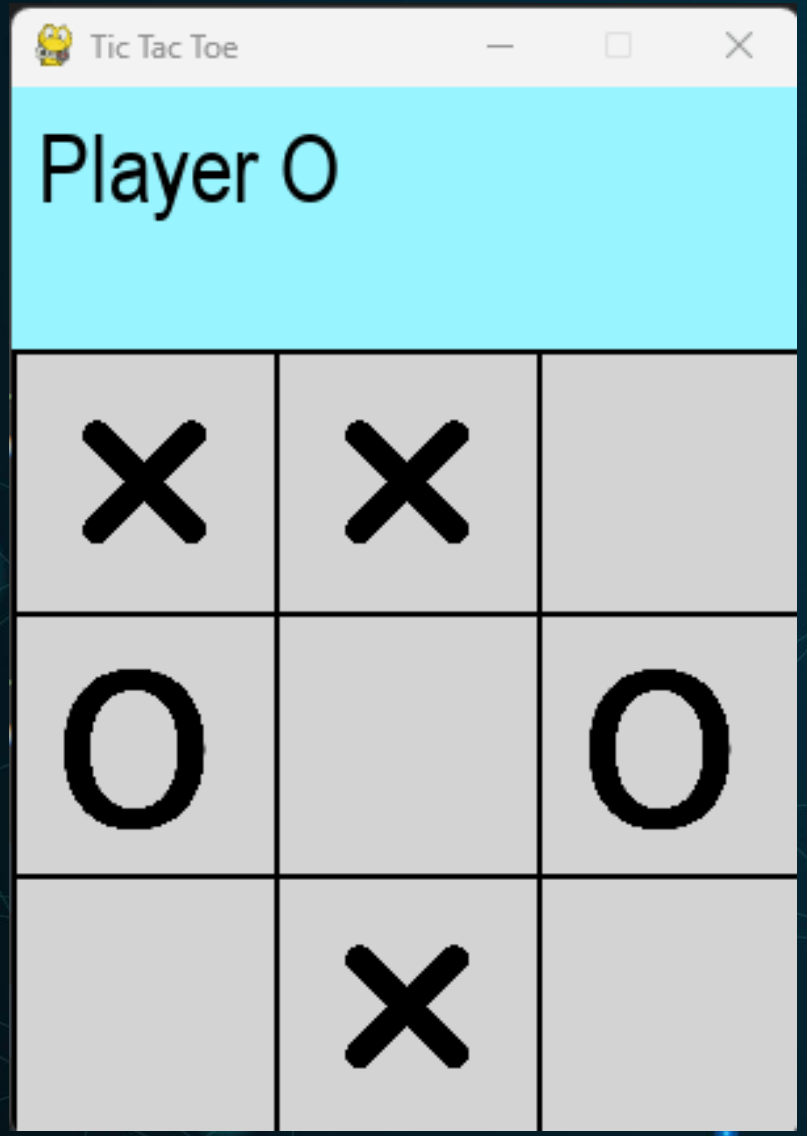


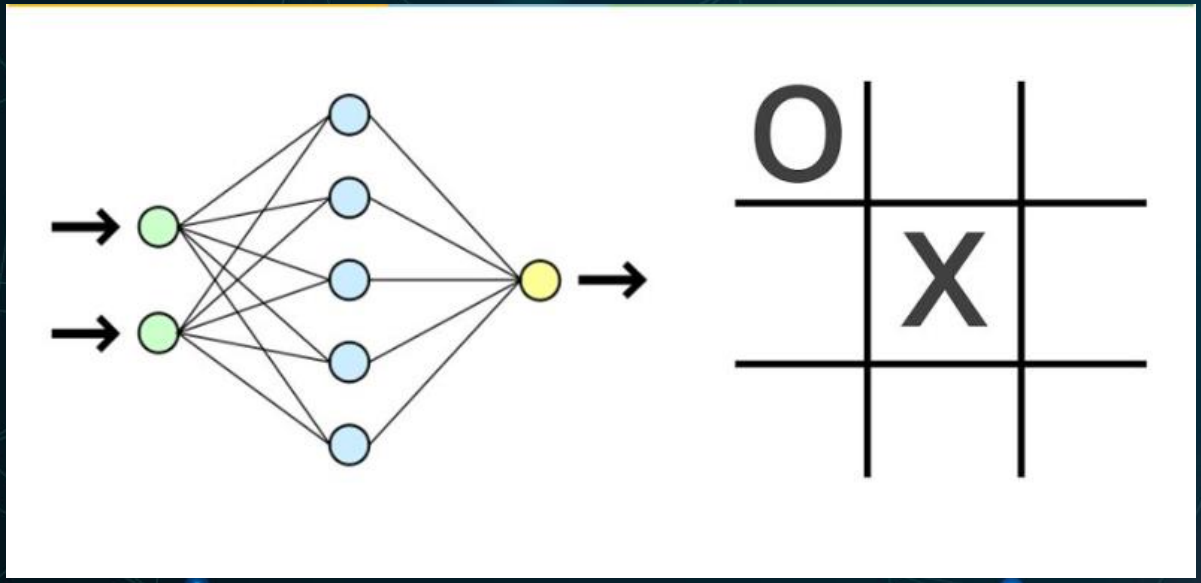
הדגמה משחק
איקס עיגול

DQN

Deep Q-learning
Neural Network

תוכן השיעור

- חזרה על האלגוריתם DQN
- המחלקה ReplayBuffer
- המחלקה DQN – רשת הנוירונים.
- המחלקה DQN_Agent – סוכן AI.
- DQN_Trainer – אימון באמצעות Deep Q-learning.
- המחלקה Tester לבדיקת המודל.



Deep Q-learning (DQN)

Initialize Q(s,a,w) with arbitrary w.

Initialize $\hat{Q}$(s,a,w^-) with arbitrary w^-.

Initialize replay-buffer RB with size N.

For epoch in epochs (for each game):

- Initialize s
- While s is not Terminal:
  - Choose A From S using Q and e-greedy
  - Interact with environment and get s, a, r, s'
  - Store transition (s, a, r, s') in RB
  - $s = s'$
  - sample random minibatch from RB
  - Calculate $Q(s, a, w)$ - forward
  - Calculate loss with MSELoos :
    - If s' is Terminal: $loss = \left(r - Q(s, a, w)\right)^2$
    - else: $loss = \left(r + \gamma \cdot max_a \hat{Q}(s', a, w^-) - Q(s, a, w)\right)^2$
  - Calculate gradients (backwards): $loss.backward()$
  - Update W : $Optim.SGD.step()$
- Every C epochs $w^- \leftarrow w$

Replay buffer - המחלקה

קריית החינוך
פארק המדע
בית לערכים
למצוינות ולחדשנות

ReplayBuffer.py > ReplayBuffer

```python
from collections import deque
import random
import torch
import numpy as np

capacity = 10000

class ReplayBuffer:
    def __init__(self, capacity= 10000) -> None:
        self.buffer = deque(maxlen=capacity)

    def push (self, state_tensor, reward_tensor, next_state_tensor, done):
        self.buffer.append((state_tensor, reward_tensor, next_state_tensor, done))

    def sample (self, batch_size):
        if (batch_size > self.__len__()):
            batch_size = self.__len__()
        state_tensors, reward_tensors, next_state_tensors, dones = zip(*random.sample(self.buffer, batch_size))
        states = torch.vstack(state_tensors)
        rewards = torch.vstack(reward_tensors)
        next_states = torch.vstack(next_state_tensors)
        done_tensor = torch.tensor(dones).long().reshape(-1,1)
        return states, rewards, next_states, done_tensor

    def __len__(self):
        return len(self.buffer)
```

Replay Buffer

- Deque – תור דו כיווני.
- Maxlen – מגביל את אורך התור כך שאם מגיעים למקסימום כל הוספה של איבר מוציאה איבר ישן. כך אנחנו שומרים את האיברים החדשים.
- האיברים ברשימה כוללים tuple של: $(state, action, reward, state', done)$
  - Done = ערך בוליאני אם מצב סופי.
- Sample – בוחר ערכים רנדומליים מהתור, ומסדר אותם ב tensors של עמודות, המתאים לאימון רשת הנוירונים.

State	Action	Reward	State’	done

DQN class

```
# Parameters
input_size = 11 # state: board = 3 * 3 + action 1 * 2
layer1 = 128
layer2 = 64
output_size = 1 # Q(s,a)
gamma = 0.99
MSELoss = nn.MSELoss()

class DQN (nn.Module):
    def __init__(self) -> None:
        super().__init__()
        if torch.cuda.is_available:
            self.device = torch.device('cpu') # 'cuda'
        else:
            self.device = torch.device('cpu')

        self.linear1 = nn.Linear(input_size, layer1, device=self.device)
        self.linear2 = nn.Linear(layer1, layer2, device=self.device)
        self.output = nn.Linear(layer2, output_size, device=self.device)
```

```
def forward (self, x):
    x = self.linear1(x)
    x = F.relu(x)
    x = self.linear2(x)
    x = F.relu(x)
    x = self.output(x)
    return x

def load_params(self, path): ...

def save_params(self, path): ...

def copy (self): ...

def loss (self, Q_value, rewards, Q_next_Values, Dones ):
    Q_new = rewards + gamma * Q_next_Values * (1- Dones)
    return MSELoss(Q_value, Q_new)

def __call__(self, states, actions):
    state_action = torch.cat((states,actions), dim=1)
    return self.forward(state_action)
```

DQN class

קריית החינוך
פארק המדע
בית לערכים
למצוינות ולחדשנות

DQN_Agent

קריית החינוך
פארק המדע
בית לערכים
למצוינות ולחדשנות

```python
class DQN_Agent:
    def __init__(self, player = 1, parametes_path = None, train = True, env= None) -> None:
        self.DQN = DQN()
        if parametes_path:
            self.DQN.load_params(parametes_path)
        self.train(train)
        self.player = player
        self.env = env

    def train (self, train):
        self.train = train
        if train:
            self.DQN.train()
        else:
            self.DQN.eval()
```

DQN_Agent

```python
def get_action (self, state: State, epoch = 0, events= None, train = True):
    epsilon = self.epsilon_greedy(epoch)
    rnd = random.random()
    actions = self.env.legal_actions(state)
    if self.train and train and rnd < epsilon:
        return random.choice(actions)

    state_tensor = state.toTensor()
    action_np = np.array(actions)
    action_tensor = torch.from_numpy(action_np)
    expand_state_tensor = state_tensor.unsqueeze(0).repeat((len(action_tensor),1))
    # state_action = torch.cat((expand_state_tensor, action_tensor ), dim=1)
    with torch.no_grad():
        Q_values = self.DQN(expand_state_tensor, action_tensor)
    max_index = torch.argmax(Q_values)
    return actions[max_index]

def get_actions (self, states, dones):
    actions = []
    for i, state in enumerate(states):
        if dones[i].item():
            actions.append((0,0))
        else:
            actions.append(self.get_action(State.tensorToState(state), train=False))
    return torch.tensor(actions)
```

Trainer

```
epochs = 30000
C = 1000
batch = 64
learning_rate = 0.1
path = "Data\DQN_PARAM_3_30K.pth"

def main ():
    env = TicTacToe()
    player1 = DQN_Agent(1, env=env)
    player2 = Random_Agent(-1, env=env)
    replay = ReplayBuffer()
    Q = player1.DQN
    Q_hat :DQN = Q.copy()
    Q_hat.train = False
    optim = torch.optim.SGD(Q.parameters(), lr=learning_rate)
```

DQN_Trainer

```python
for epoch in range(epochs):
    print (epoch, end="\r")
    state = State()
    while not env.end_of_game(state):
        action = player1.get_action(state, epoch=epoch)
        after_state, reward = env.next_state(state, action)
        if env.end_of_game(after_state):
            replay.push(state, action, reward, after_state, env.end_of_game(after_state))
            break
        after_action = player2.get_action(state=after_state)
        next_state, reward = env.next_state(after_state, after_action)
        replay.push(state, action, reward, next_state, env.end_of_game(next_state))
        state = next_state

        if epoch < batch:
            continue
        states, actions, rewards, next_states, dones = replay.sample(batch)
        Q_values = Q(states, actions)
        next_actions = player1.get_actions(next_states, dones)
        with torch.no_grad():
            Q_hat_Values = Q_hat(next_states, next_actions)

        loss = Q.loss(Q_values, rewards, Q_hat_Values, dones)
        loss.backward()
        optim.step()
        optim.zero_grad()
        if epoch % C == 0:
            Q_hat.load_state_dict(Q.state_dict())

player1.save_param(path)
```

קריית החינוך
פארק המדע
בית לערכים
למצוינות ולחדשנות

Tester

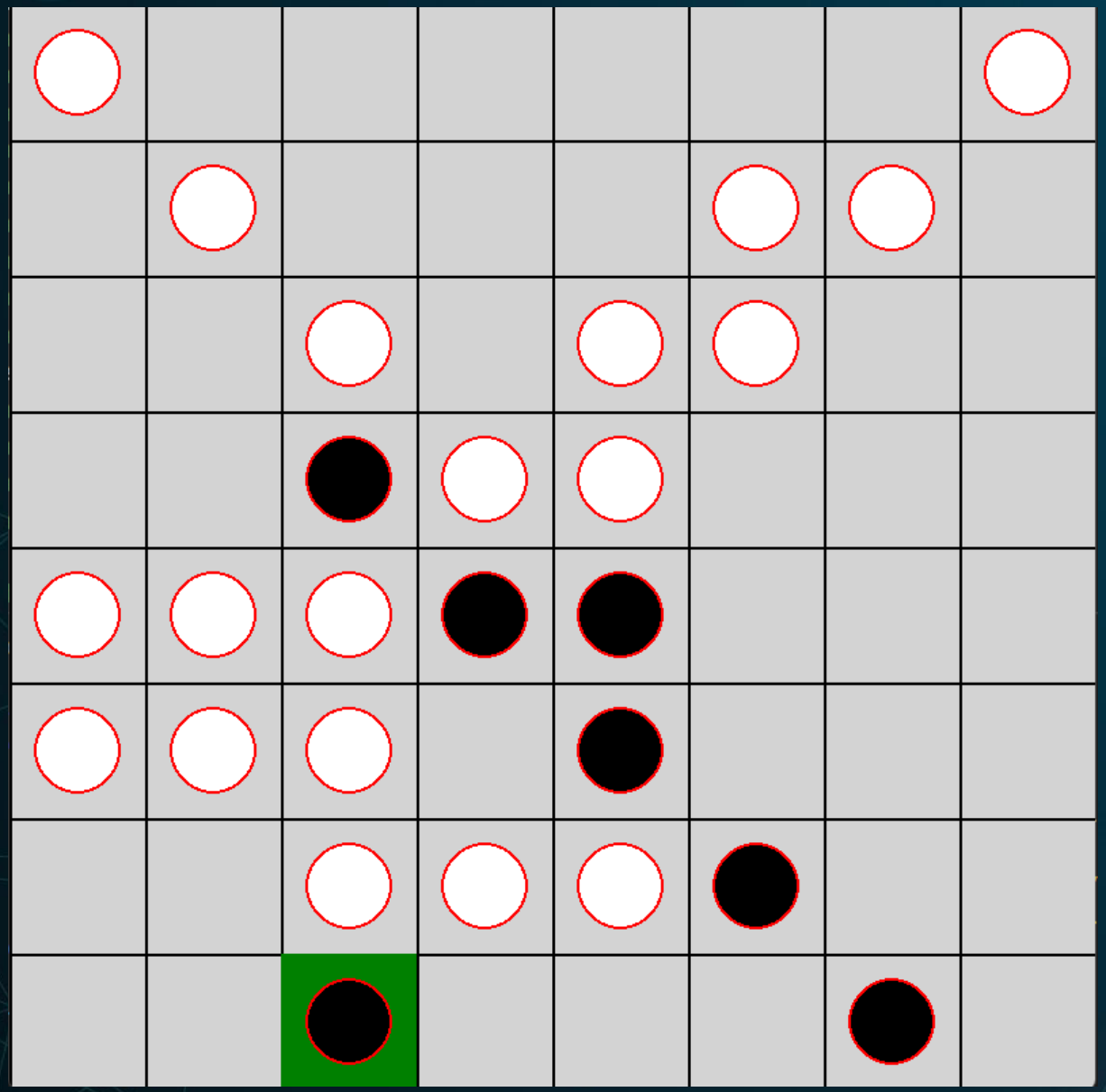
```python
PATH = 'Data\DQN_PARAM_3_30K.pth'

env = TicTacToe(State())
player1 = DQN_Agent(1, env=env, parametes_path=PATH, train=False)
# player1 = Random_Agent(1, env,graphics=None)
player2 = Random_Agent(-1, env,graphics=None)
num = 1000
```

```python
def main ():

    x_win = 0
    o_win = 0
    tie = 0

    for n in range(num):
        state = State()
        player = player1
        while not env.end_of_game(state):
            action = player.get_action(state=state)
            state, _ = env.next_state(state,action)
            player = switch_players(player)
        if state.end_of_game == 1:
            x_win +=1
        elif state.end_of_game == -1:
            o_win += 1
        else:
            tie +=1
        state.reset()
        print(n, end = "\r")
    print()
    print(x_win, o_win, tie)
```



קריית החינוך
פארק המדע
בית לערכים
למצוינות ולחדשנות

בשיעור הבא:

הדגמת DQN
משחק רברסי

Othelo - Reversi

קריית החינוך
פארק המדע
בית לערכים
למצוינות ולחדשנות

קריית החינוך
פארק המדע
בית לערכים
למצוינות ולחדשנות