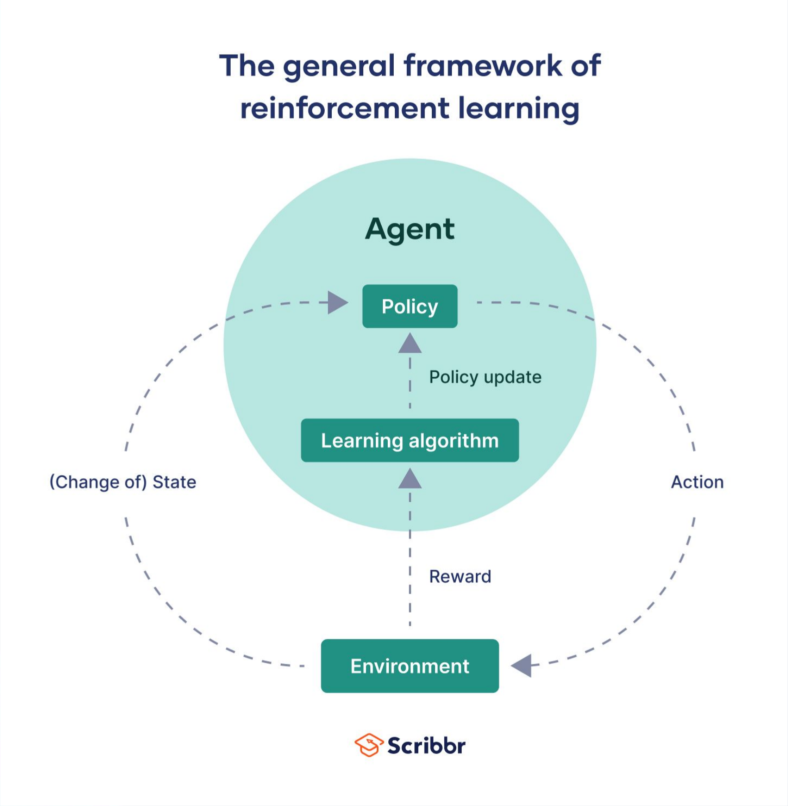
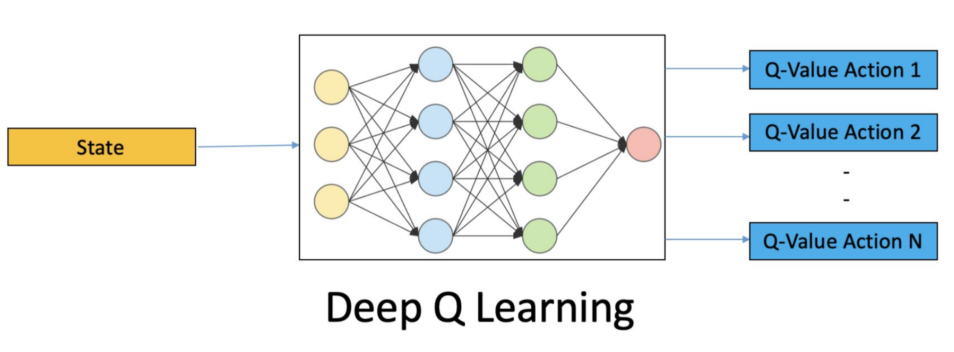
Air-Hockey Bot

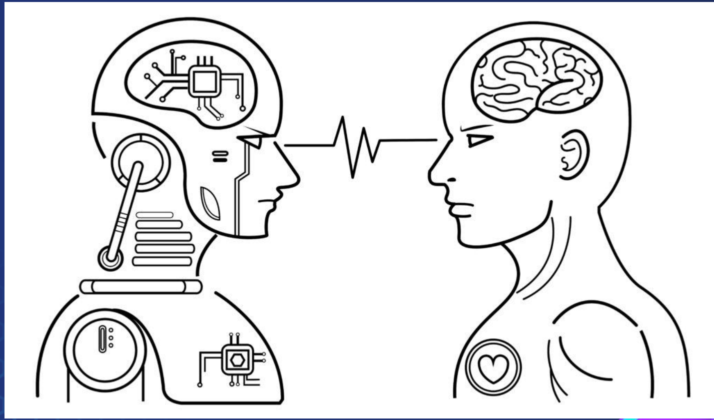
Jorge Bris Moreno, Eric Dapkus, Brian Kwon, Kang Liu, and Billy McGloin
DSAN 6600, Final Project, April 24, 2024

Contents

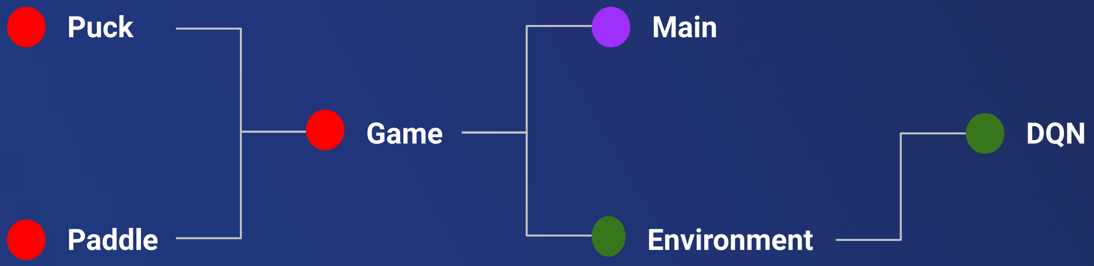
Introduction

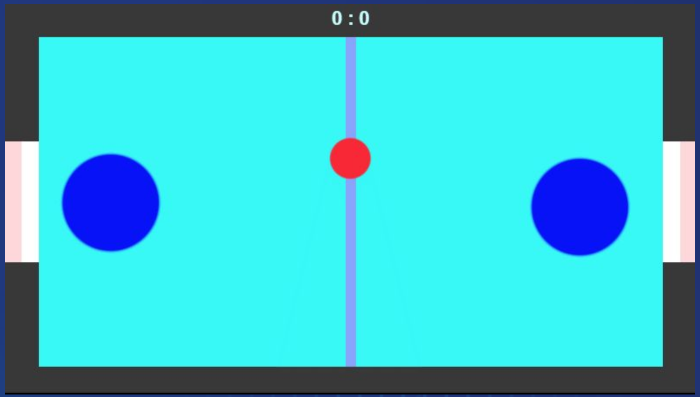






Air Hockey Game





Environment

Deck.html
(https://kang.georgetown.domains/dsan-website/6600-website/project-presentation/deck.html#/title-slide)

Models

Convolutional Neural Network

```python
class DQN(nn.Module):
    def __init__(self, outputs):
        super(DQN, self).__init__()
        self.conv1 = nn.Conv2d(3, 16, kernel_size=3, stride=1, padding=1)  # Small kernel, stride 1
        self.bn1 = nn.BatchNorm2d(16)
        self.pool1 = nn.AvgPool2d(2)  # Pooling to reduce dimension

        self.conv2 = nn.Conv2d(16, 32, kernel_size=3, stride=1, padding=1)  # Small kernel, stride 1
        self.bn2 = nn.BatchNorm2d(32)
        self.pool2 = nn.AvgPool2d(2)  # Pooling to reduce dimension

        # Calculate the size of the output from the pooling layer
        self._to_linear = None
        self._get_conv_output([1, 3, 16, 8])  # Input shape sample

        self.head = nn.Linear(self._to_linear, outputs)  # Linear layer to output the Q-values for each action

    def _get_conv_output(self, shape):
        input = torch.rand(shape)
        output = self.pool1(self.bn1(self.conv1(input)))
        output = self.pool2(self.bn2(self.conv2(output)))
        self._to_linear = int(torch.numel(output) / output.shape[0])

    def forward(self, x):
        x = F.relu(self.pool1(self.bn1(self.conv1(x))))
        x = F.relu(self.pool2(self.bn2(self.conv2(x))))
        x = x.view(x.size(0), -1)  # Flatten the features for the linear layer
        return self.head(x)
```

Linear Neural Network

```python
# Linear Neural network model
class DQN_linear(nn.Module):
    def __init__(self, input_dim, output_dim):
        super(DQN_linear, self).__init__()
        self.linear1 = nn.Linear(input_dim, 16)  # First hidden layer
        self.linear2 = nn.Linear(16, 32)  # Second hidden layer
        self.linear3 = nn.Linear(32, 16)  # Third hidden layer
        self.head = nn.Linear(16, output_dim)  # Output layer to produce the final outputs

    def forward(self, x):
        x = F.relu(self.linear1(x))  # Activation function for first layer
        x = F.relu(self.linear2(x))  # Activation function for second layer
        x = F.relu(self.linear3(x))  # Activation function for third layer
        x = self.head(x)  # Output layer does not need an activation for Q-value estimation
        return x
```

CNN & Linear Combo

```
class DQN(nn.Module):
    def __init__(self, outputs):
        super(DQN, self).__init__()
        self.conv1 = nn.Conv2d(3, 16, kernel_size=4)  # Small kernel, stride 1
        self.bn1 = nn.BatchNorm2d(16)

        self.conv2 = nn.Conv2d(16, 32, kernel_size=4)  # Small kernel, stride 1
        self.bn2 = nn.BatchNorm2d(32)

        # Calculate the size of the output
        self._to_linear = None
        self._get_conv_output([1, 3, 16, 8])  # Input shape sample

        self.linear = nn.Linear(self._to_linear, 18)  # Linear layer to output the Q-values for each action
        self.head = nn.Linear(18, outputs)  # Head layer to output the final Q-values

    def _get_conv_output(self, shape):
        input = torch.rand(shape)
        output = self.bn1(self.conv1(input))
        output = self.bn2(self.conv2(output))
        self._to_linear = int(torch.numel(output) / output.shape[0])

    def forward(self, x):
        x = F.relu(self.bn1(self.conv1(x)))
        x = F.relu(self.bn2(self.conv2(x)))
        x = x.view(x.size(0), -1)  # Flatten the features before the linear layer
        x = F.relu(self.linear(x))
        return self.head(x)
```

Training Process (Training environment)

Google Colab Pro
- L4 GPU
- 12.7GB system ram
- 4.1 credit per hour

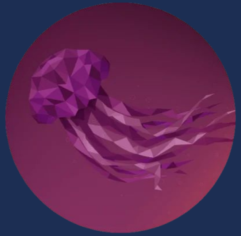
```
#check GPU
gpu_info = !nvidia-smi
gpu_info = '\n'.join(gpu_info)
if gpu_info.find('failed') >= 0:
  print('Not connected to a GPU')
else:
  print(gpu_info)
```

Google Colab

```
#install required packages for colab
!pip install gymnasium
#run code
!python dqn.py
```

Intel NUC
- 4 x NUC11
— i5
— 64GB ram
— 500 GB m.2

Ubuntu Server (22.04.4 LTS)
Software
- Screen
- miniconda
- pip



Hostnames: James, Ben, Jeff, Nakul

Training Process Cont. (Training Data)

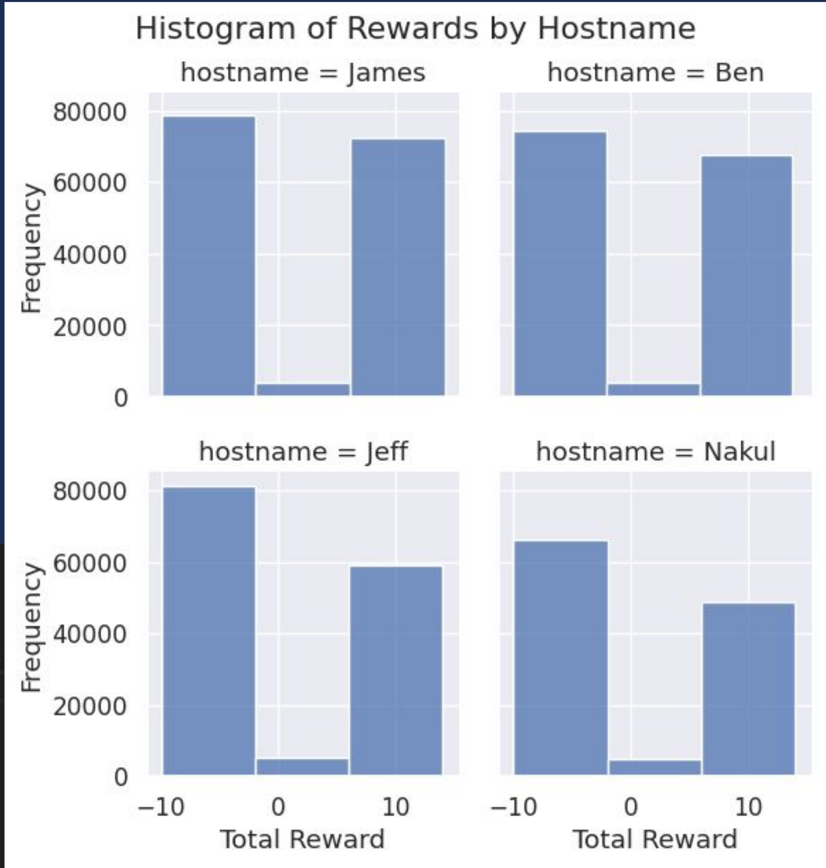
james (155257 episodes)
- Opponent new bot logic

ben (146760 episodes)
- Opponent new bot logic
- TARGET_UPDATE = 750

jeff (145268 episodes)
- Opponent old bot logic

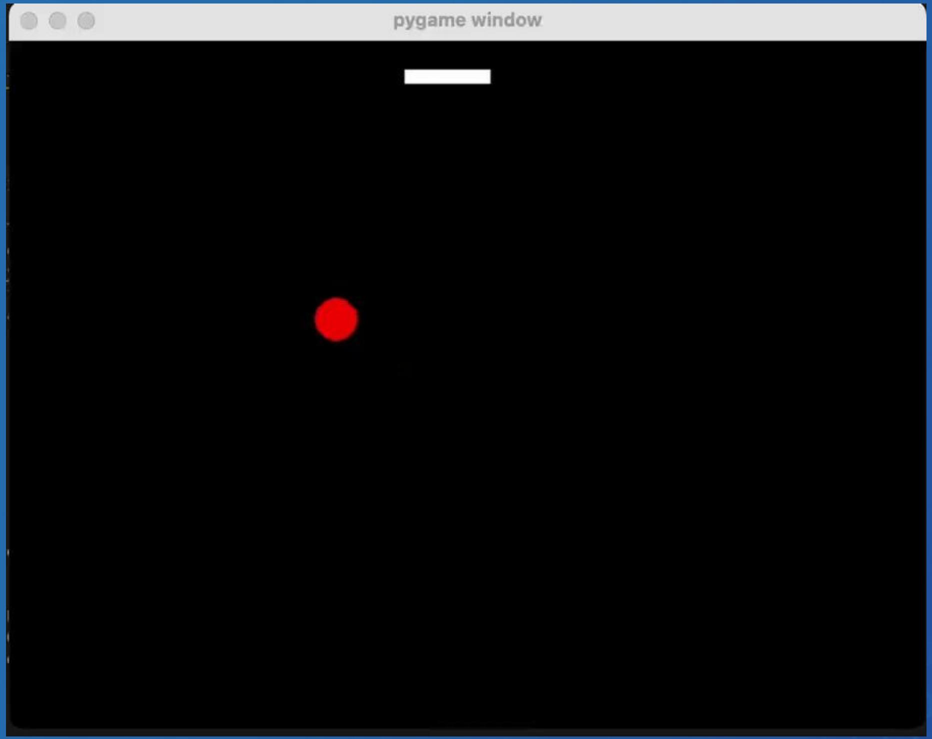
nakul
- Opponent old bot logic
- TARGET_UPDATE = 750

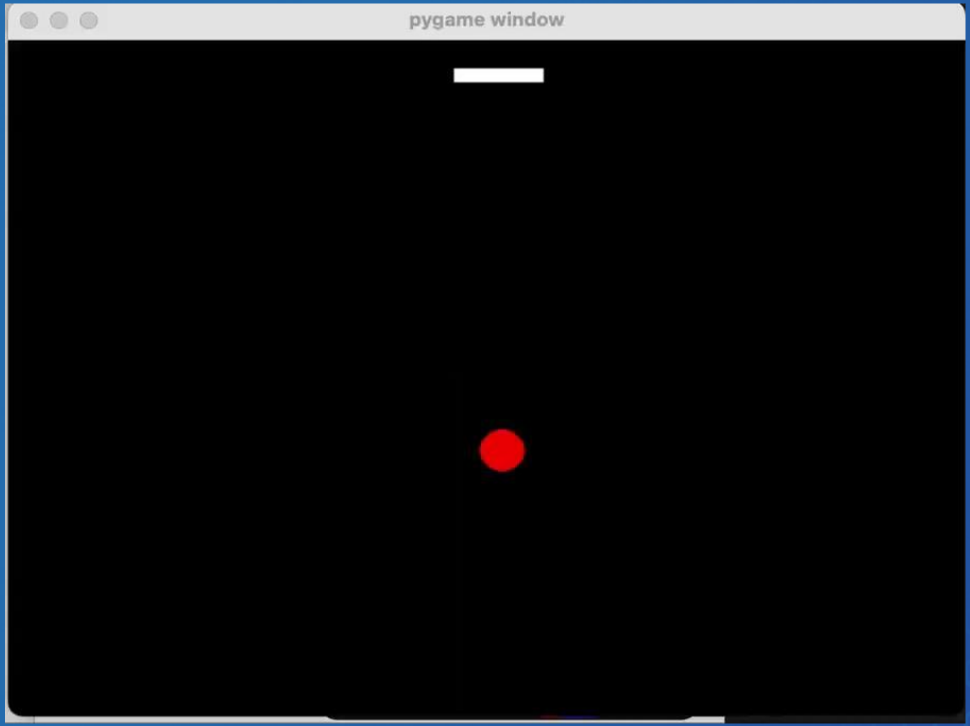
```
# hyperparameters
BATCH_SIZE = 64
GAMMA = 0.99  # Discount factor for future rewards
EPS_START = 0.95  # Initial epsilon value for epsilon-greedy action selection
EPS_END = 0.05  # Final epsilon value for epsilon-greedy action selection
EPS_DECAY = 80000  # Rate at which epsilon decreases
LR = 1e-4  # Learning rate for the optimizer
TARGET_UPDATE = 1000  # How often to update the target network
MEMORY_CAPACITY = 10000  # Capacity of the replay memory
NUM_EPISODES = 500000  # Number of episodes to train
```

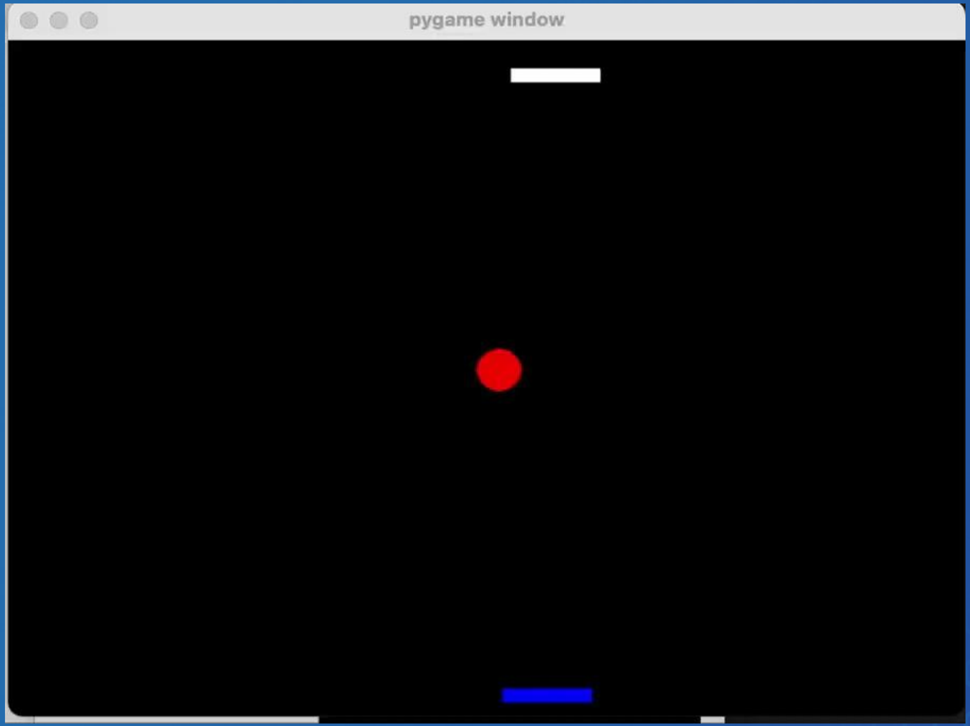


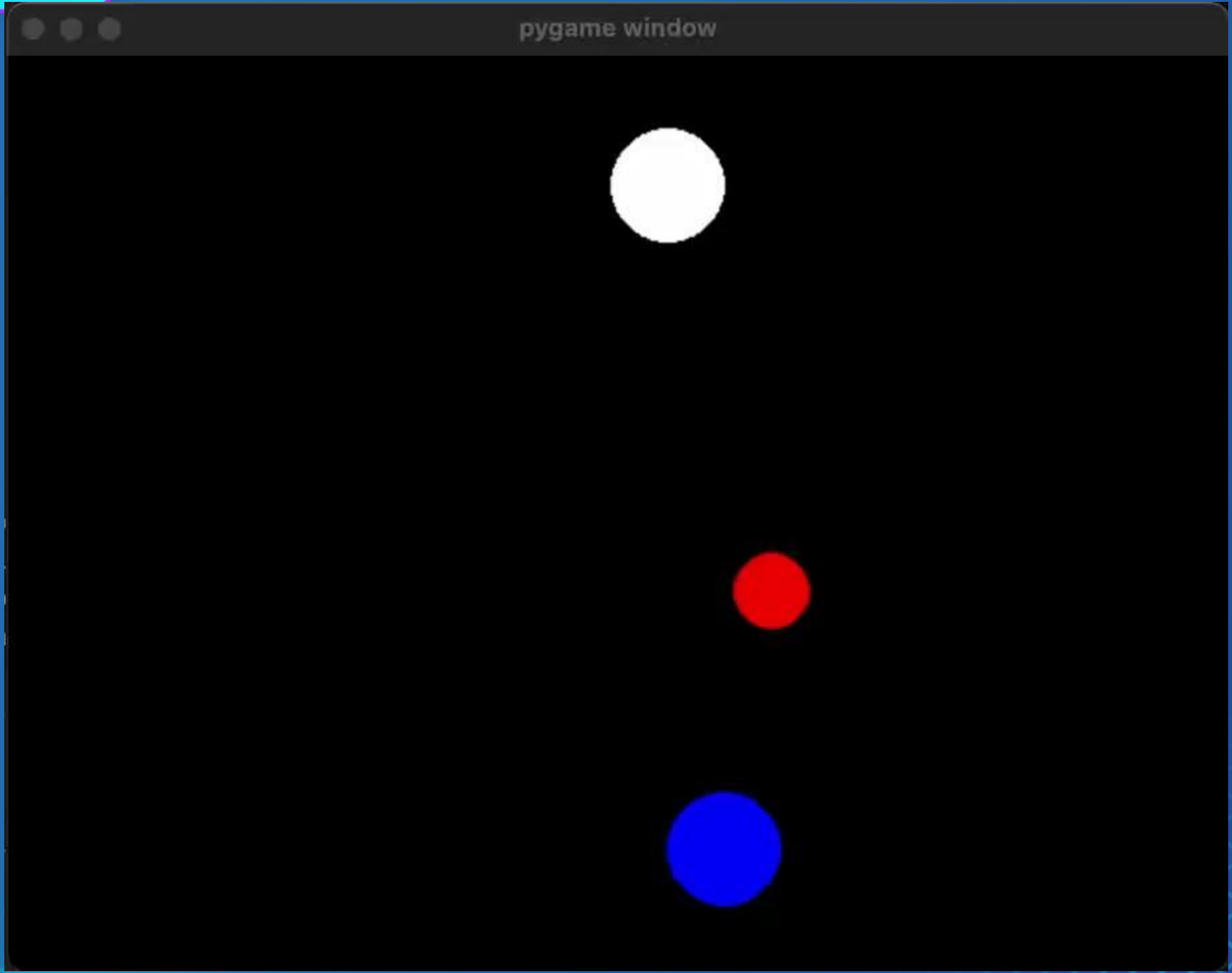
Training Process Cont. (Training Data)

Proof of Concept


pygame window


pygame window


pygame window


pygame window

Air Hockey Implementation

Future Plans

1. Extend Training Duration
2. Evaluate Alternate Rewards
3. Human v. RL?
4. Implement Supervised Learning
5. RL v. RL?
6. Upgrade RL Network

Conclusion

Thanks!

Special thanks to our professors!

Reference:

Google Colab. https://colab.research.google.com/#

Questions?