Assignment 4: Appendix

Ozan Ilhan
s3559513

Qiu van Leeuwen
s3547566

1 Advantage Actor-Critic (A2C)

```python
def A2C_blackjack(model_info, hidden_size, learning_rate, n_repetitions, gamma):
    environment = gym.make("Blackjack-v1")
    total_rewards = np.zeros(n_repetitions)

    input_size = len(environment.observation_space)
    output_size = environment.action_space.n

    model = ActorCriticDiscrete(input_size, hidden_size, output_size)
    optimizer_actor = torch.optim.Adam(model.actor.parameters(), lr=learning_rate)
    optimizer_critic = torch.optim.Adam(model.critic.parameters(), lr=learning_rate)

    wins = np.zeros(n_repetitions)
    draws = np.zeros(n_repetitions)
    losses = np.zeros(n_repetitions)
    for repetition in range(n_repetitions):
        state, _ = environment.reset()
        state = torch.tensor(state, dtype=torch.float32)
        total_reward = 0
        done = False
        count = 0
        while not done:
            count += 1
            probabilities, values = model(state)
            distribution = torch.distributions.Categorical(probabilities)

            action = distribution.sample()

            next_state, reward, done, truncation, _ = environment.step(action.item())
            total_rewards[repetition] = reward
            if truncation:
                done = True
            if reward > 0:
                wins[repetition] += 1
            elif reward < 0:
                losses[repetition] += 1
            else:
                draws[repetition] += 1
            next_state = torch.tensor(next_state, dtype=torch.float32)
            _, next_values = model(next_state)
            advantage = reward + (gamma * next_values.detach() * (1 - int(done))) - values
            actor_loss = -distribution.log_prob(action) * advantage.detach()
            critic_loss = advantage.square()
            total_reward += reward

            optimizer_actor.zero_grad()
            optimizer_critic.zero_grad()

            actor_loss.backward()
            critic_loss.backward()

            optimizer_actor.step()
            optimizer_critic.step()
            state = next_state
    environment.close()
    torch.save(model.state_dict(), f'A2C_blackjack_{model_info}.pth')
    return total_rewards
```

Listing 1: The code for the A2C algorithm.

2 Deep Q-Learning

```python
def DQL_blackjack(model_info, hidden_size, learning_rate, n_repetitions, gamma):
    environment = gym.make("Blackjack-v1")
    total_rewards = np.zeros(n_repetitions)

    input_size = len(environment.observation_space)
    output_size = environment.action_space.n

    model = DeepQLearning(input_size, hidden_size, output_size)
    optimizer = torch.optim.Adam(model.parameters(), lr=learning_rate)

    batch_size = 64
    memory = deque(maxlen=10000)
    epsilon = 0.1
    for repetition in range(n_repetitions):
        state, _ = environment.reset()
        state = torch.tensor(state, dtype=torch.float32)
        total_reward = 0
        done = False
        while not done:
            if np.random.rand() < epsilon:
                action = environment.action_space.sample()
            else:
                q_values = model(state)
                action = torch.argmax(q_values).item()

            next_state, reward, done, truncation, _ = environment.step(action)
            total_rewards[repetition] = reward
            if truncation:
                done = True
            next_state = torch.tensor(next_state, dtype=torch.float32)
            memory.append((state, action, next_state, reward, done))
            state = next_state
            total_reward += reward

            if len(memory) > batch_size:
                experiences = random.sample(memory, batch_size)
                states, actions, next_states, rewards, dones = zip(*experiences)
                states = np.array(states)
                actions = np.array(actions)
                rewards = np.array(rewards)
                next_states = np.array(next_states)
                dones = np.array(dones)
                q_values = model(torch.tensor(states, dtype=torch.float32))
                next_q_values = model(torch.tensor(next_states, dtype=torch.float32))

                target_q_values = q_values.clone()
                for i in range(len(experiences)):
                    if dones[i]:
                        target_q_values[i, actions[i]] = rewards[i]
                    else:
                        target_q_values[i, actions[i]] = rewards[i] + gamma * torch.max(next_q_values[i]).item()
                mse_loss = nn.MSELoss()
                loss = mse_loss(q_values, target_q_values.clone().detach().to(dtype=torch.float32))
                optimizer.zero_grad()
                loss.backward()
                optimizer.step()
                state = next_state
    environment.close()
    torch.save(model.state_dict(), f'DQL_blackjack_{model_info}.pth')
    return total_rewards
```

Listing 2: The code for the DQL algorithm.