Foundations of Intelligent & Learning Agents

Project Proposal

Intelligent Pool Player

Atharv Nandapurkar	160050004
Rajat Rathi	160050015
Nikhil Tummidi	160050096
Gurparkash Singh	160050112

Problem Statement

The aim of our project is to model the game of Pool and to build an intelligent agent that could learn to play the game efficaciously. Pool is a very complex game with several nuances that can make developing a generalized agent a challenging task. Therefore, before we can hope to build an omnipotent agent, it is wiser to relax some of the constraints and learn in a more simplified model of the game. However, the proposed relaxations don't interfere with the overall working of the game and are mostly related to the corner-cases in the rules of the game. There is some existing work that also trains a model in a relaxed environment, however the restraints in the previous work are too loose. Our goal is to train our model in a more constrained environment than before, even though it is still relaxed as compared to the actual game.

Existing Work

In the existing work [1], [2], the authors have developed a model that can learn to sink all the balls on the pool table sequentially. They have used a modified version of the original pool game simulator [3] (alternatively, [4]) in order to train their model. In the paper, the agent has been formulated to treat all the balls equally and to sink them sequentially using the white ball. The reward function used by them is greedy in the sense that it makes the agent to try to sink the maximum number of balls in one turn. This might not be optimum, because in the long run, it might be better to sink less number of balls in the current turn and instead set up the positions of the other balls such that it takes less number of turns to sink all the balls on the table.

Novelty of our work

From the above description of the existing work, it is clear that this is a very loose model of the game, as it ignores the fact that there are two types of balls (solid and striped), out of which a player can only sink one type of balls and directly touching or sinking the other type of balls results in a penalty. Also, their model ignores the existence of the black ball (or the 8 ball), which is to be sunk only after all the balls of a particular type have been sunk. Not conforming to this rule also leads to a penalty. These are the kinds of constraints that we plan to tackle in our project. We plan to come up with the reward functions such that the agent chooses one type of balls to sink and tries to sink them before moving on to the black ball which is to be sunk at last.

However, there are various other constraints that we will relax. In the original version of the game, the very first shot is valid only if at least four balls touch the perimeter of the pool. Also, it is necessary to declare the hole in which the player is going to pocket the black ball before attempting to sink it. If the player sinks the black ball in some other hole, it is considered a foul. We are also going to relax this constraint, mostly because it is cumbersome to have the agent "declare" the hole automatically only for the final turn. As we can see, relaxing these constraints do not interfere with the overall functioning of the game.

Problem Formulation

The problem is formulated as an MDP with the positions of all the balls (including the white and the black ball) as its state. The states can be both continuous and discrete depending on the method used to solve MDP. The set of actions will (θ,F) where θ is the angle at which we strike the white ball and F is the corresponding force. Both θ and F will be discrete.

The reward function will be formulated based on the following points:

1. Reward when the ball of the correct type is sunk
2. Penalize if the ball of the other type is sunk
3. Reward/Penalty should be proportional to the number of balls sunk
4. Penalize if the white ball is sunk
5. Penalize if the black ball is sunk before sinking the other balls

Methodology

We will model the game as a Markov Decision Process by experimenting with different types of reward functions and various definitions of states and actions, both discrete and continuous. In order to solve the MDPs, we plan to implement and compare the following two algorithms:

1. Q-Learning Algorithm :

Q-learning is an algorithm that learns a policy which maximizes the expected value of total reward in the future. To learn the Q-value, which is the expected total future reward by taking a certain action at a certain state, we iteratively update the Q-value with the following equation upon each experience:

$$\hat{Q}(s,a) := \hat{Q}(s,a) + \alpha(r + \gamma \max_{a' \in action(s')} \hat{Q}(s',a') - \hat{Q}(s,a))$$

where $\hat{Q}(s,a)$ is the current estimate of the Q-value, s is the current state, a is the current action, r is the reward received by taking action a at state s, s' is the transitioned next state, α is the learning rate, and γ is the discount factor of the rewards. The state and action pairs will be discrete for this method.

2. Deep Q-Network (DQN)

In our second method, we will use Deep Q-Networks (DQN) to learn the Q-value of all the actions in a given state. Once we have the network to give these mappings, we can get the optimal policy by using the methods discussed in class. We will experiment with various network architectures as well as methods like drop-out and batch-normalization. The states will be continuous and the actions will be discrete for this method.

Expected Outcomes

If all goes well, we should have an intelligent agent that can pocket all the balls of one type without sinking any ball of the other type. The model should pocket the black ball at the end. We will analyse the differences between the two methods discussed above. Even though this does not capture all the subtleties of the real game, it captures enough nuances and take a long stride towards the complete model, which can be implemented as a simple extension of our work.

References

[1] Deep Cue Learning: A Reinforcement Learning Agent for Playing Pool

[2] Implementation of above paper

[3] Pool simulator used in paper

[4] Pymunk library