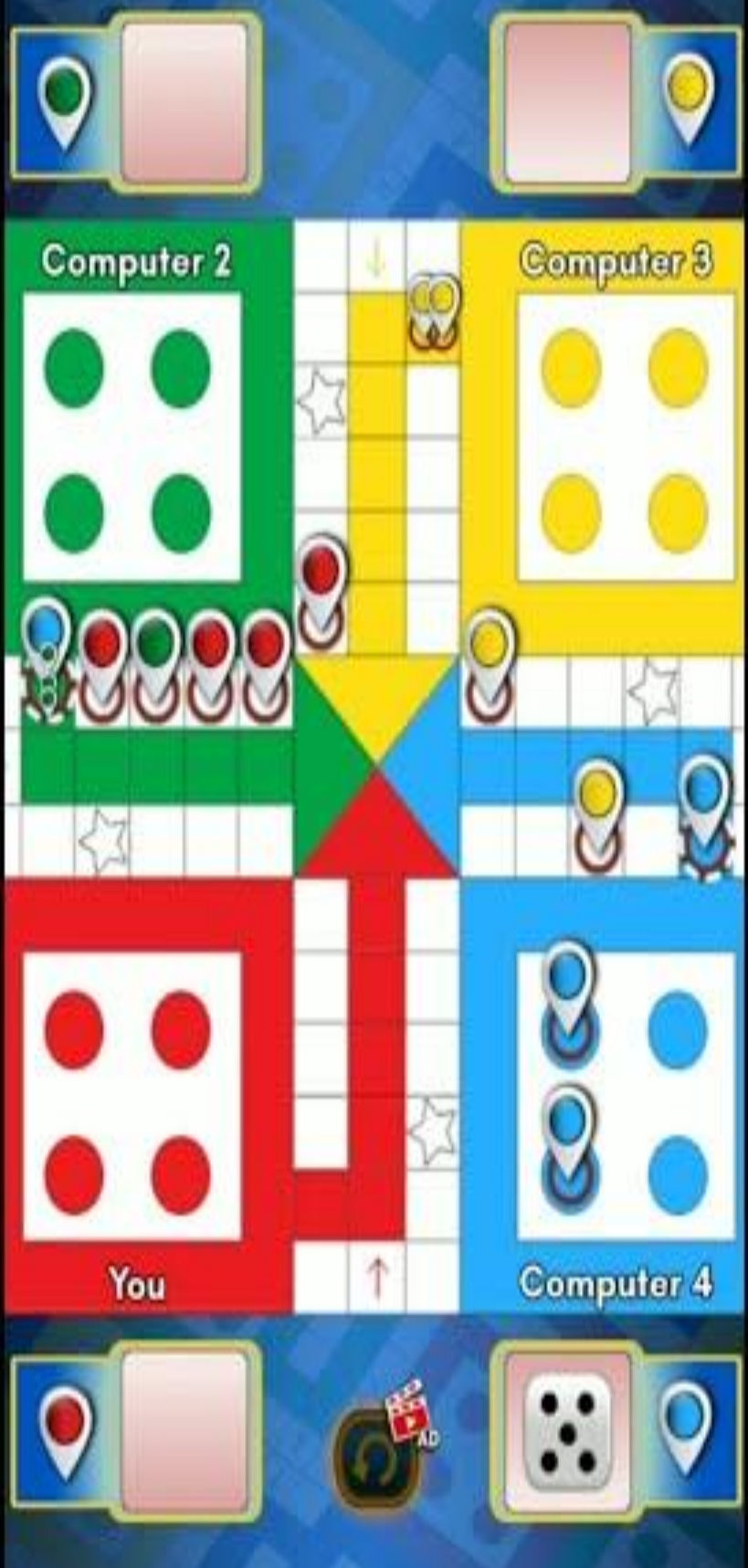
AI based Ludo Player

Uditi Goyal, Tanya Jain, Ritish Shailly, Manas Shukla
Mini Project | Intro to Artificial Intelligence
April 28th, 2022

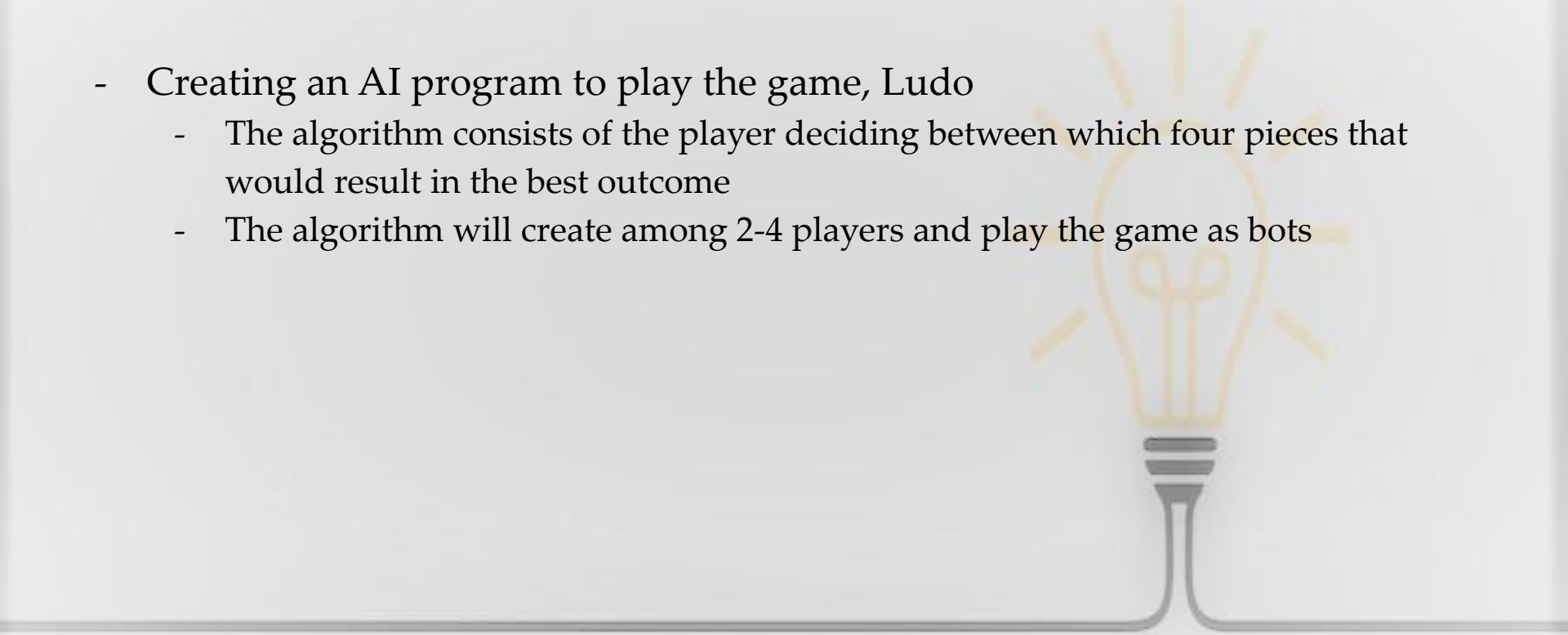

Computer 2
Computer 3
You
Computer 4
AD

Research Topic

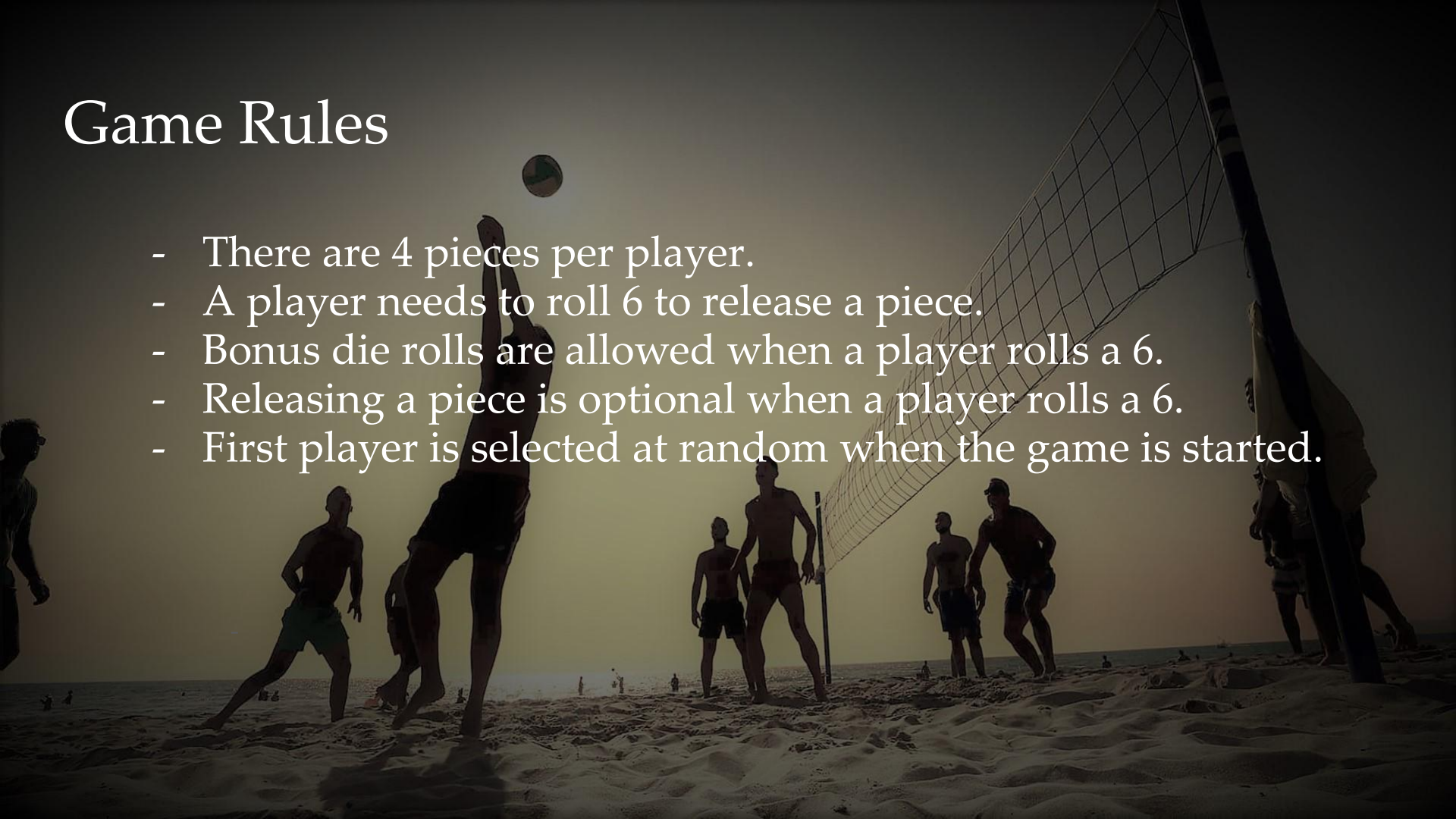
Creating an AI based player
to play the game Ludo

Problem Statement

- Creating an AI program to play the game, Ludo
  - The algorithm consists of the player deciding between which four pieces that would result in the best outcome
  - The algorithm will create among 2-4 players and play the game as bots



Game Rules



- There are 4 pieces per player.
- A player needs to roll 6 to release a piece.
- Bonus die rolls are allowed when a player rolls a 6.
- Releasing a piece is optional when a player rolls a 6.
- First player is selected at random when the game is started.

Concepts used from AI class

- **MDPs (Markov Decision Process)**
- **Reinforcement learning**
- **Q-learning**
- **Neural Networks (FANN)**

Mathematical Model

- Markovian Decision Process

$$<S, A, P_a(s,s'), R_a(s,s')>$$

  - S: all possible states in environment
  - A: set of all possible actions in an environment
  - P_a: Transition model (Probability of moving from s to s')
  - R_a: Immediate reward received when agent goes to s'

-

Mathematical Model

- Markovian Decision Process

$$<S, A, P_a(s,s'), R_a(s,s')>$$

  - S: all possible states in environment
  - A: set of all possible actions in an environment
  - P_a: Transition model (Probability of moving from s to s')
  - R_a: Immediate reward received when agent goes to s'

Reinforcement learning devises a policy π which maps states to actions

Optimal policy π^* produces maximum cumulative rewards of all states

Q-Learning

Q-learning focuses on finding the optimal policy by estimating quality values for each state/action combination (known as Q-Values), and updates them

$$Q(s, a) = r(s, a) + \gamma \max_{a} Q(s', a)$$

Q-Learning

Q-learning focuses on finding the optimal policy by estimating quality values for each state/action combination (known as Q-Values), and updates them

$$Q(s,a) = r(s,a) + \gamma \max_a Q(s',a)$$

Policy Update:

$$Q^{new}(s_t,a_t) \leftarrow \underbrace{Q(s_t,a_t)}_{\text{old value}} + \underbrace{\alpha}_{\text{learning rate}} \cdot \overbrace{\Big(\underbrace{\underbrace{r_t}_{\text{reward}} + \underbrace{\gamma}_{\text{discount factor}} \cdot \underbrace{\max_a Q(s_{t+1},a)}_{\text{estimate of optimal future value}}}_{\text{new value (temporal difference target)}} - \underbrace{Q(s_t,a_t)}_{\text{old value}}\Big)}^{\text{TD error}}$$

Q-Learning

Q-learning focuses on finding the optimal policy by estimating quality values for each state/action combination (known as Q-Values), and updates them

$$Q(s,a) = r(s,a) + \gamma \max_a Q(s',a)$$

Policy Update:

$$Q^{new}(s_t,a_t) \leftarrow \underbrace{Q(s_t,a_t)}_{\text{old value}} + \underbrace{\alpha}_{\text{learning rate}} \cdot \overbrace{\Big(\underbrace{\underbrace{r_t}_{\text{reward}} + \underbrace{\gamma}_{\text{discount factor}} \cdot \underbrace{\max_a Q(s_{t+1},a)}_{\text{estimate of optimal future value}}}_{\text{new value (temporal difference target)}} - \underbrace{Q(s_t,a_t)}_{\text{old value}}\Big)}^{\text{TD error}}$$

For our problem, we have set the QL learning rate as 0.5 and discount factor as 0.95

Framework

Learning Objective: Select best applicable action for a given state

A = {defensive, aggressive, fast, random, preferRelease}

State Representation:

- Each piece has (52+5) squares
- Each square state represented by real number (0 - no pieces, 1- four pieces)
- 4 unary inputs which indicate current player turn
- So, total inputs = (59 X 4) + 4 = 294 inputs

Training and Testing:

- The model was trained using 4 QL players and was tested in 2 scenarios
  - 1 QL vs 3 random players
  - 1 QL vs 3 expert players

Framework

Rewards: Rewards were designed to pick moves in descending order:

- Win the game
- Release a piece.
- Defend a vulnerable piece
- Knock an opponent piece
- Move pieces closest to home
- Form a blockade

Loss: the agent is penalized in the following situations:

- Getting one of its pieces knocked in the next turn.
- Losing the game

Policy:

- The epsilon-greedy policy is used to balance between exploration and exploitation with $\varepsilon = 0.9$

Expert Player

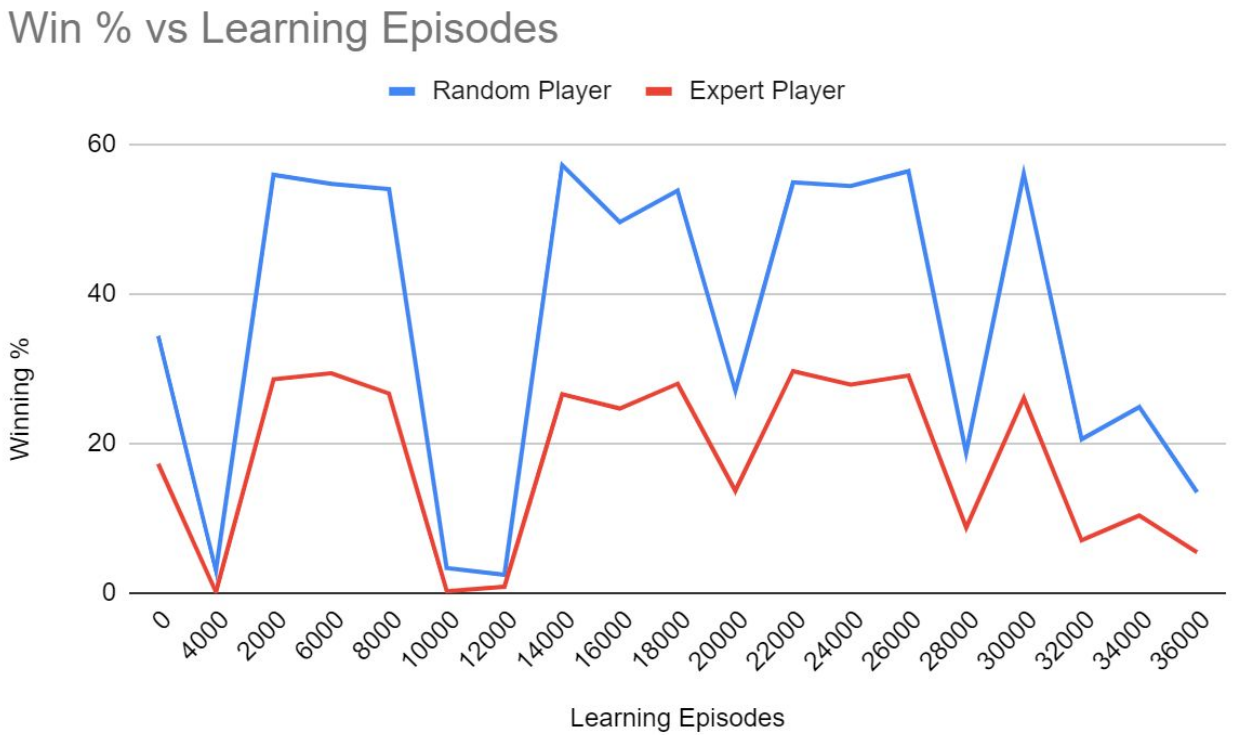
Priority of Strategies:

- Defensive - Maximize knocking range from opponents
- Aggressive - Prefer knocking opponent’s piece
- Fast - Move pieces to home location first
- Random

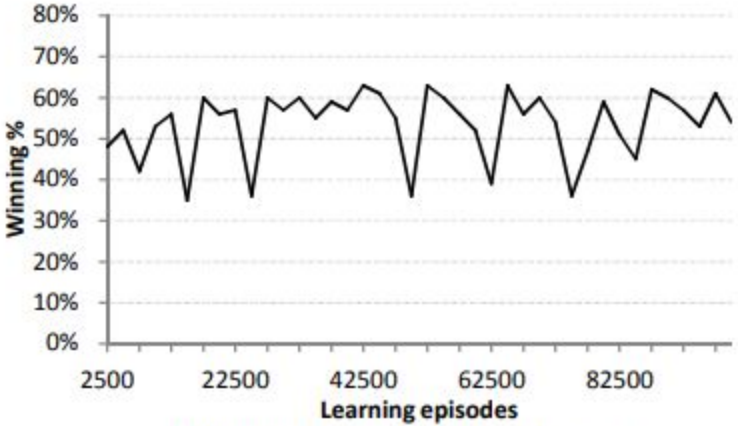
Approaches

- Implemented with Reinforcement Learning
  - Similar approach to the Pac-man game
  - Uses Deep Learning which includes our agent and the environment
- Every time the agent performs an action, the environment gives a positive or negative reward to the agent
- Goal: For the agent to learn which actions maximizes the reward

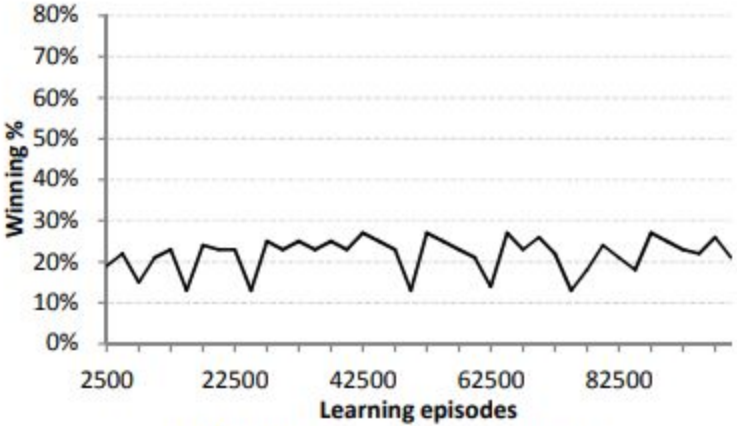
Results


Win % vs Learning Episodes
Random Player
Expert Player
60
40
20
0
Winning %
0
4000
2000
6000
8000
10000
12000
14000
16000
18000
20000
22000
24000
26000
28000
30000
32000
34000
36000
Learning Episodes

Research Paper Results



(a) Winning rate against 3 random players



(b) Winning rate against 3 expert players

Limitations

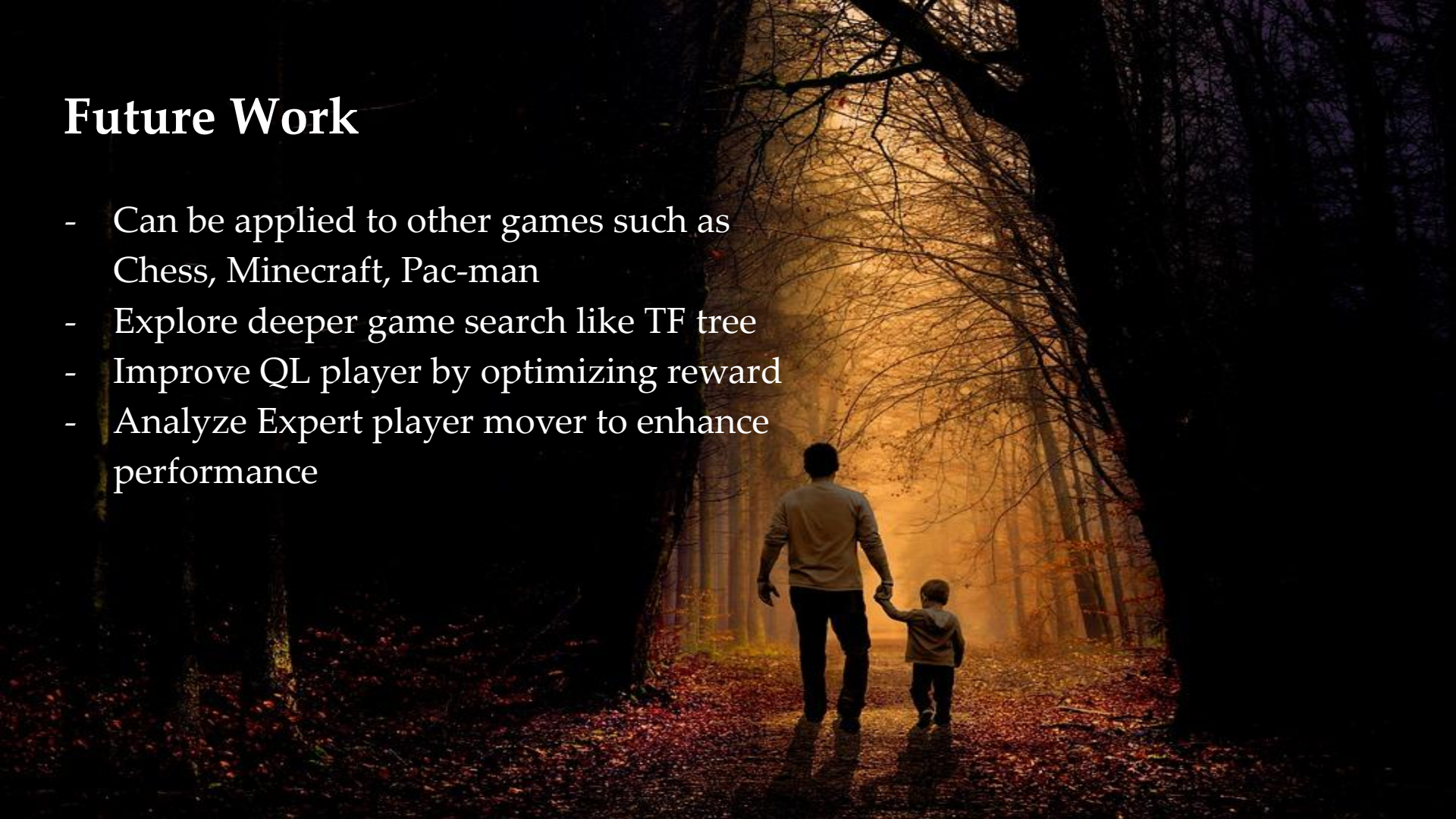
- We tried to reproduce the research done in the paper. However due to lack of computing power we weren't able to reproduce it completely.
- The results against random players were comparable to research paper. However There were slightly more deviations when tested against expert players.
- This game is based on luck (dice rolls) which adds a complexity to evaluation of results.

Lessons learned

- Reinforcement learning for AI agent
- Implementation of Q-learning for multiagent AI games
- Effect of learning parameter and discount factor in Q-learning
  - Learning becomes better with lower learning rate
  - Learning is quicker with higher learning rate
  - Low discount factor is quicker because it learned policy that doesn't consider future states
  - High discount factor considers future states and is therefore slower
- Effect of learning against expert vs random players
  - Algorithm performs better with random players as compared to expert players

Future Work

- Can be applied to other games such as Chess, Minecraft, Pac-man
- Explore deeper game search like TF tree
- Improve QL player by optimizing reward
- Analyze Expert player mover to enhance performance




Questions?