Department of Computer Science & Engineering
Independent University Bangladesh

Creating an Open-AI gym like environment for Bangladeshi game Shologuti using Unity 3D and ML-Agents

Samin Bin Karim

ID: 1720049

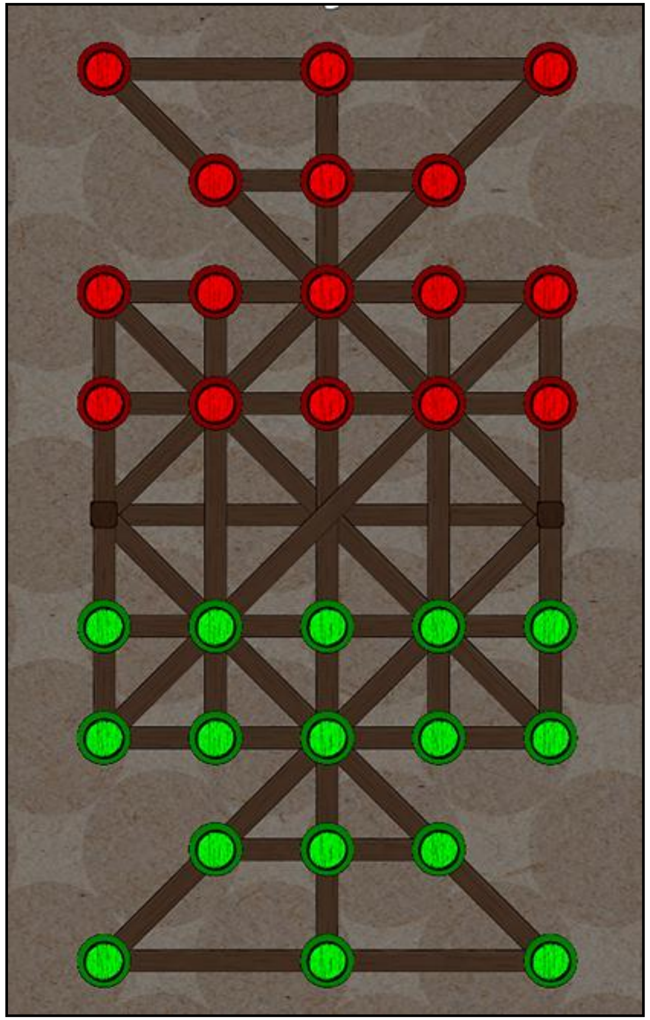
Supervised by

Dr. Amin Ahsan Ali

Outline of Presentation

- Introduction
  - Brief background of Shologuti
  - Shologuti as a research environment
  - Motivation for project
  - Objectives and Challenges
- Design and Implementation
- Results and Analysis
- Conclusion and Future work

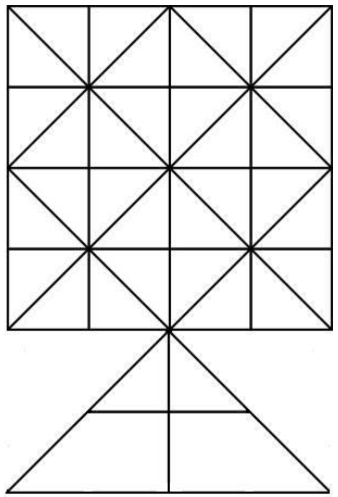
Shologuti – A traditional board game



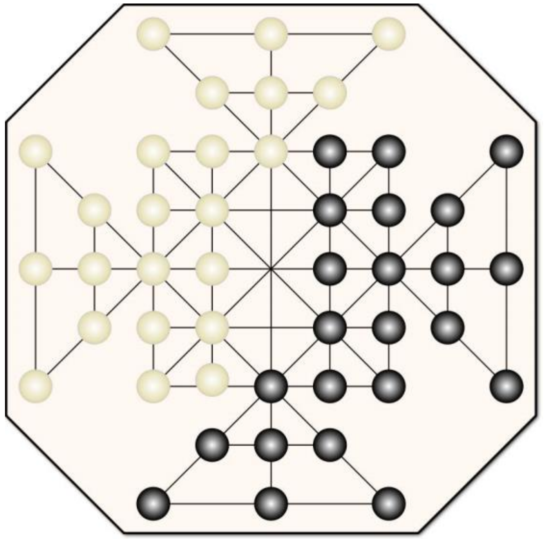
Shologuti – Bangladesh/ India

- A local board game popular in rural Bangladesh and India
- Part of a family of games including checkers
- That are all derived from Al-Quirkat

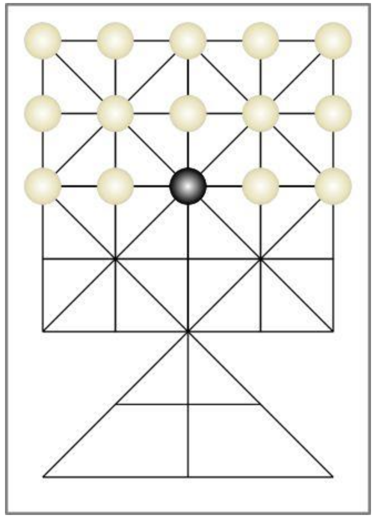
History of Shologuti



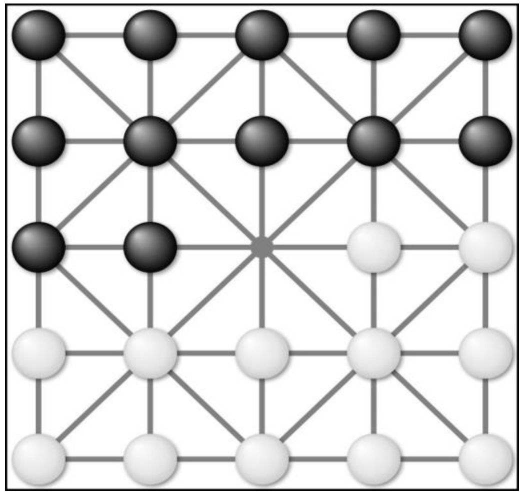
Tiger Game - Chile



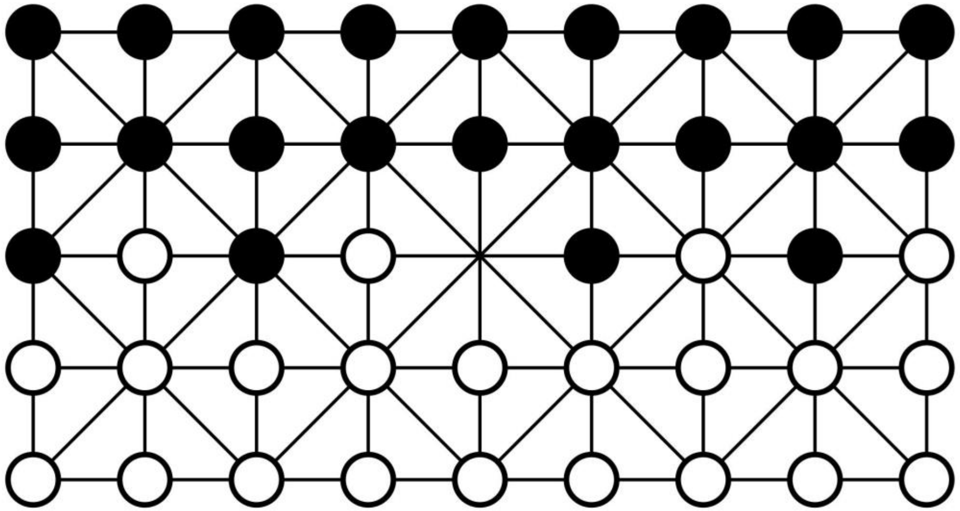
War Enclosure - Sri Lanka



Adugo Jaguar and Dogs – Brazil

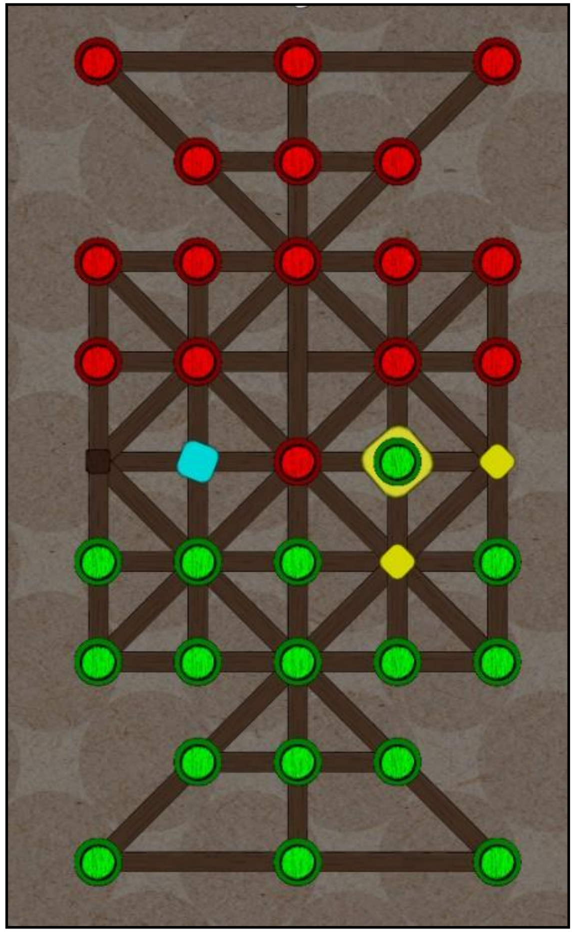


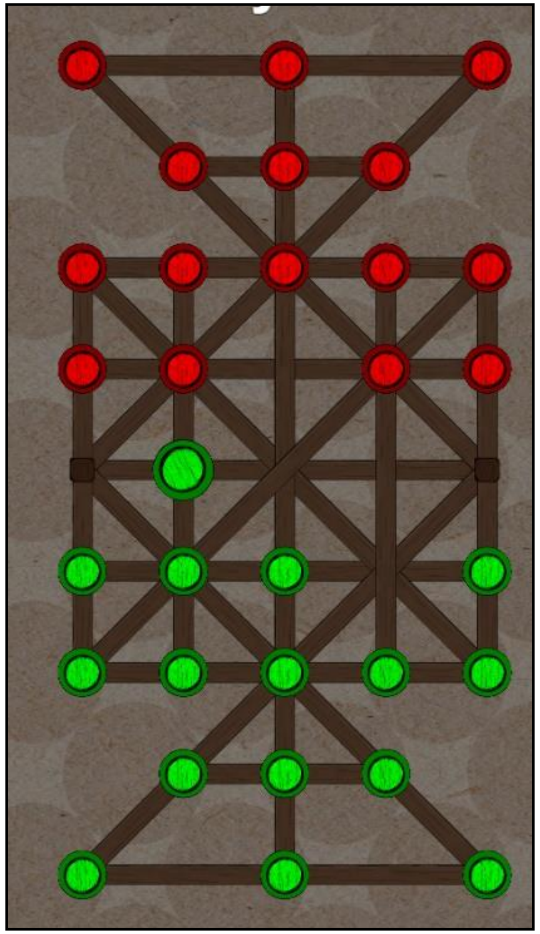
Al-Quirkat – Spain/Middle East



Fanorona-Madagascar

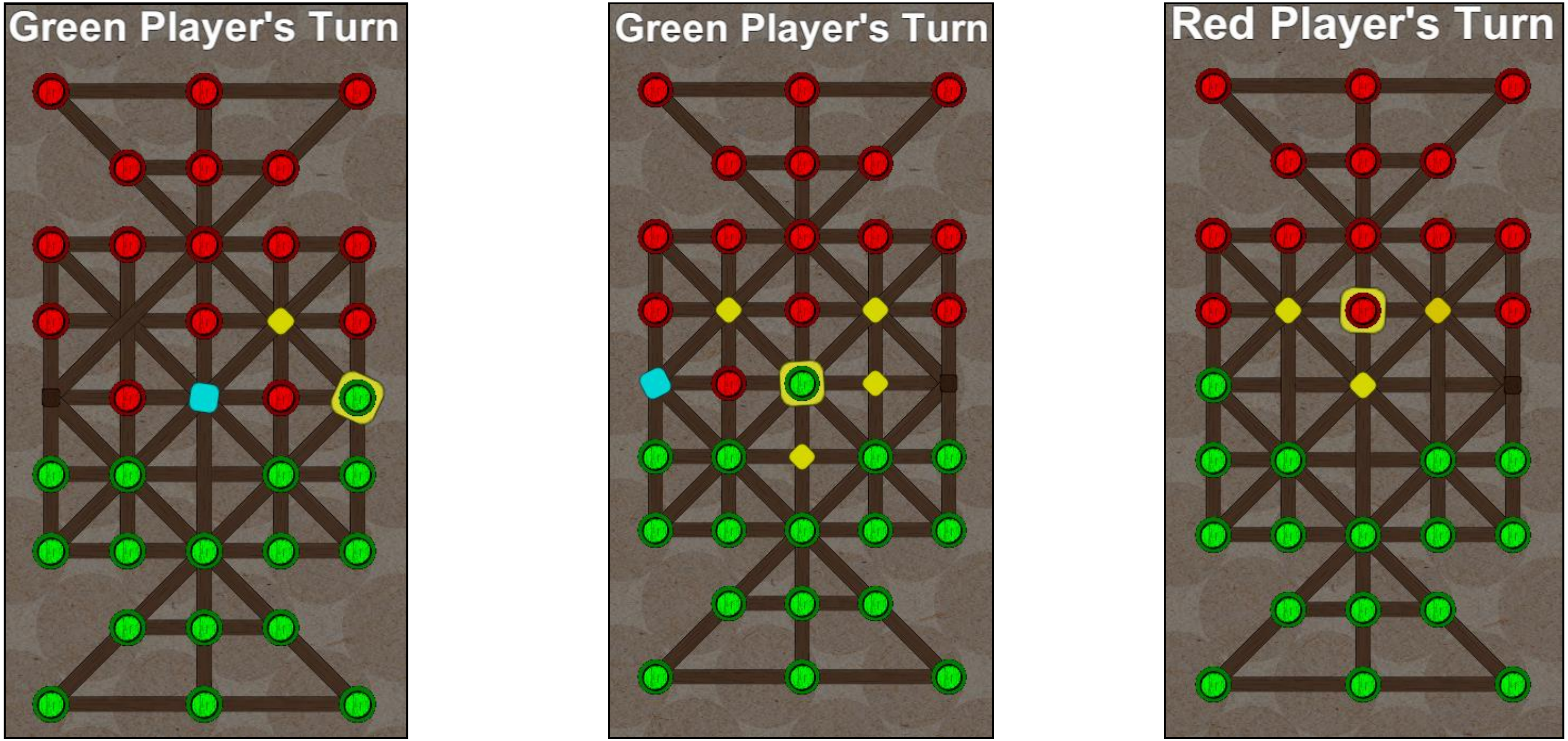
Shologuti Game Rules





- Each player starts with 16 guti
- Players make move in alternate turns
- Must chose one legal move
  - Can move to occupy adjacent empty space
  - Or jump over enemy guti to capture
- Multi capture moves are possible
- Must capture all 16 enemy guti to win

Multi-capture move



Shologuti board game as an AI research environment

$$\boldsymbol{x_1} = \textit{number of gutis owned by player } 1$$

$$\boldsymbol{x_2} = \textit{number of gutis owned by player } 2$$

$$\boldsymbol{x_3} = \textit{number of empty spaces on the board}$$

$$\boldsymbol{x_3} = \textit{board\_size} - x_1 - x_2$$

$$\textbf{board\_size} = 37$$

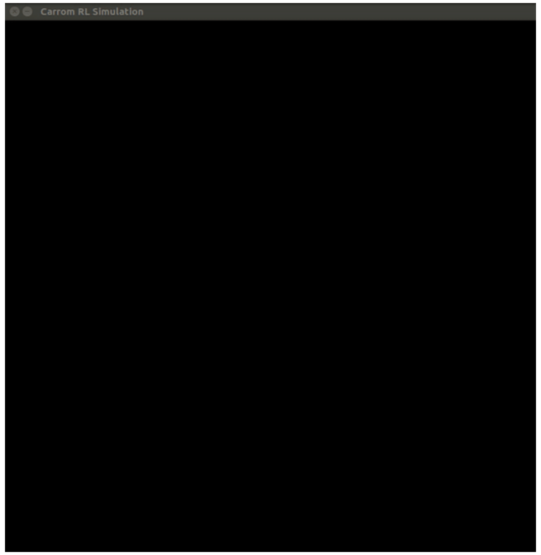
$$\log_{10}(\textbf{Statespacecomplexity}) = \log_{10} \sum_{x_1=0}^{16} \sum_{x_2=0}^{16} \frac{37!}{x_1!\,x_2!\,x_3!} = 17.58$$

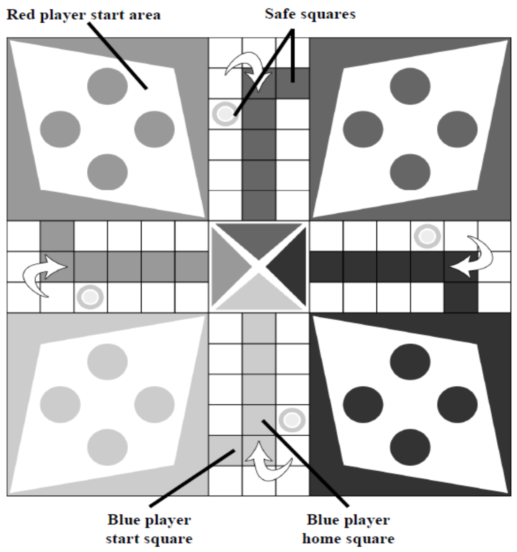
$$\log_{10}(\textbf{State space complexity}) \approx 18$$

Branching Complexity $= 14$

(empirical estimate from simulator)

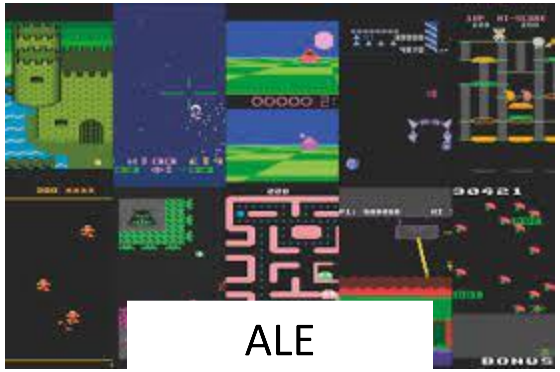
Research in local games popular in Bangladesh

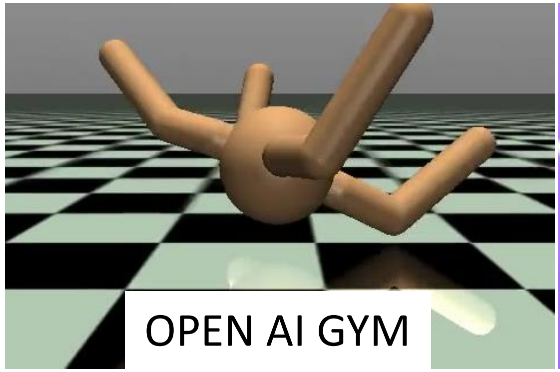




- **Ludo** – Highest amount of research done
- **Carrom** – Simulator for research available (IIT Bombay)
- **Shologuti** – Neglected with little prior research
  - Only one paper that implements MinMax tree searching algorithm for Shologuti

Simulation environments for RL





- Arcade Learning environment, ALE
- Open AI gym
- RLCard
- OpenSpiel

State of the art RL algorithms

- Temporal Difference (TD) Gammon – 1992
- AlphaGo - 2016
- Proximal Policy Optimization – 2017
- Soft Actor Critic - 2019
- Alpha Zero – 2017
- MuZero - 2019

Motivation

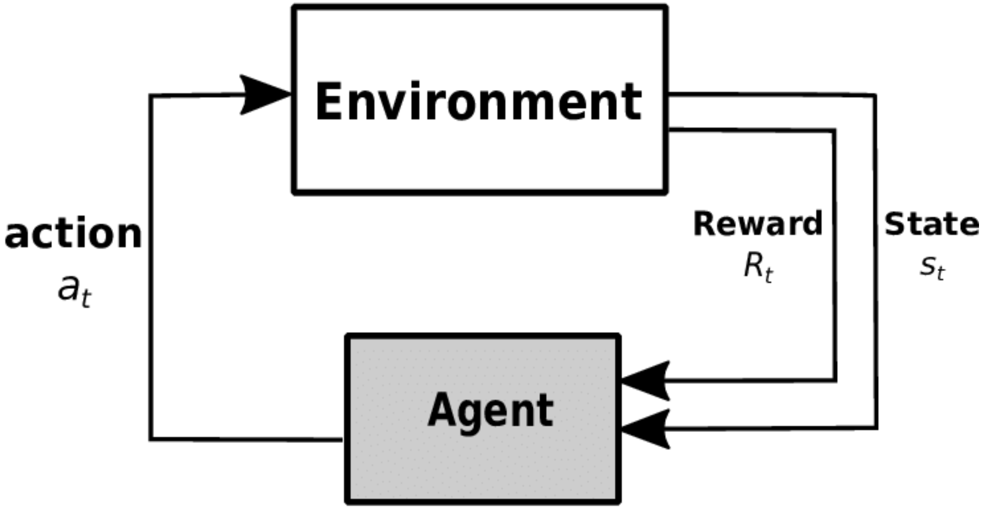
- Lack of research done for local games
- Poor availability of information for Shologuti
- Lack of availability of research environment for local games like Shologuti
- Popularity of environments like Open AI Gym

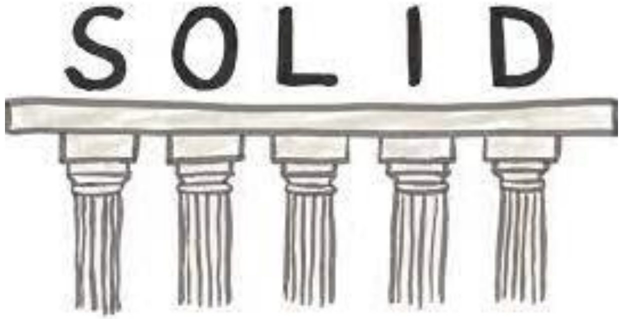
Objectives

- Python accessible learning environment
  - for training/benchmarking RL Agents
- Playable user-friendly Shologuti game
  - Running on web and windows
- Benchmark state of the art RL algorithms in Shologuti environment

Challenges faced

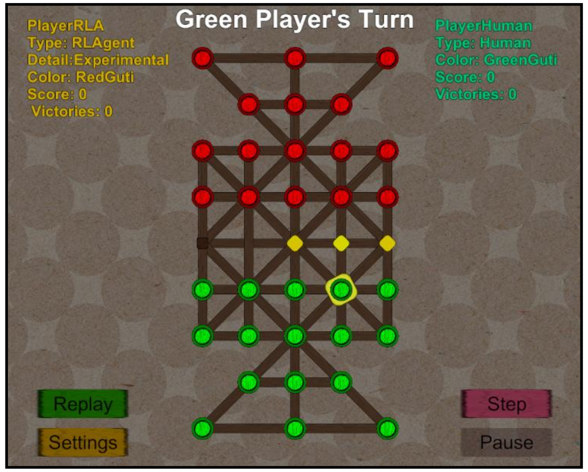
- Learning Game Development with Unity game engine
- Struggle with Unity ML-Agents toolkit
- Learning Reinforcement learning and AI Agents
- Learning and applying basics of Software engineering

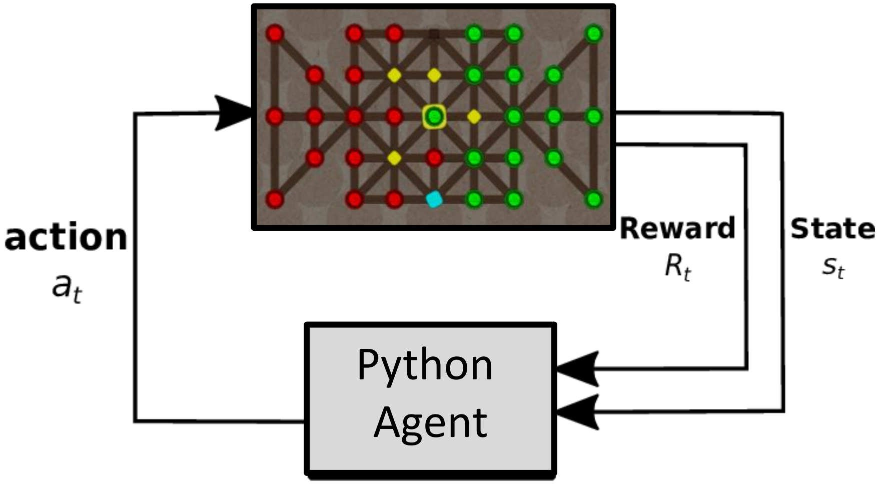


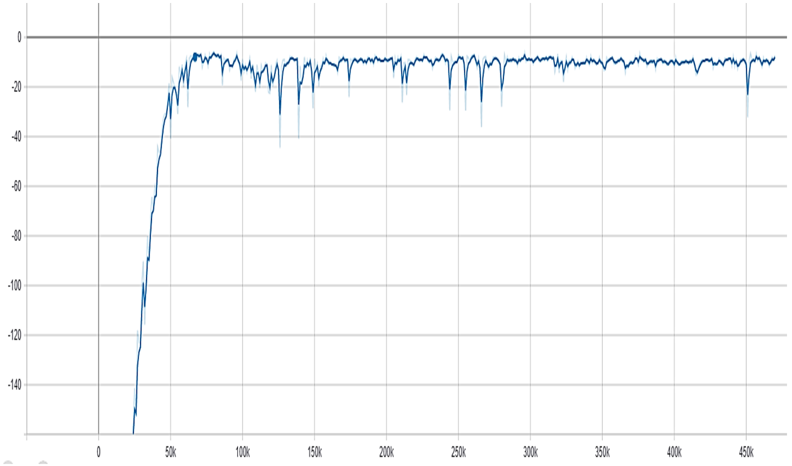


Contributions

- A playable [Shologuti game](#) for web and windows
- A research/learning environment for Shologuti
- Trained RL Agents capable of playing Shologuti
- Benchmarks of state of the art RL algorithms
- A Novel reward system Intermediate Goal States





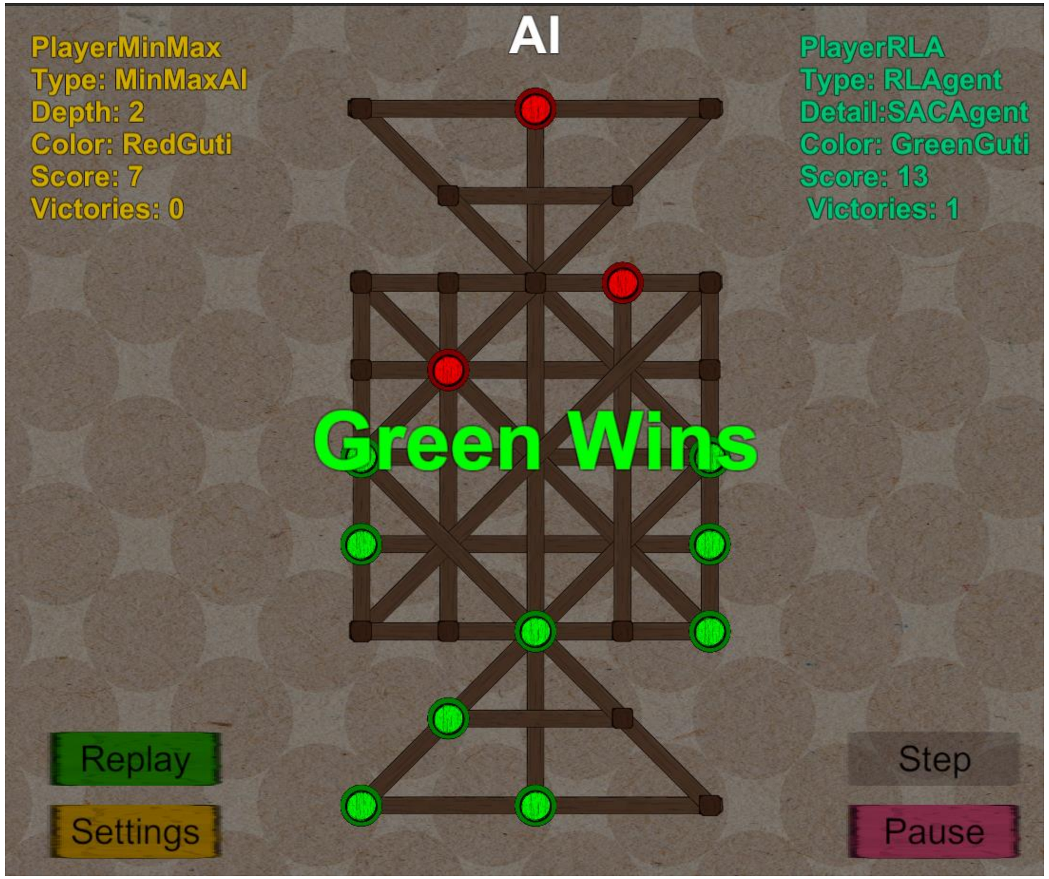


Limitations

- ML-Agents toolkit makes drastic changes in updates
- Future iterations of Unity ML-Agents may not support the Shologuti environment
- Extending features of the environment requires C# and Unity knowledge
- Installing our python learning environment is not streamlined

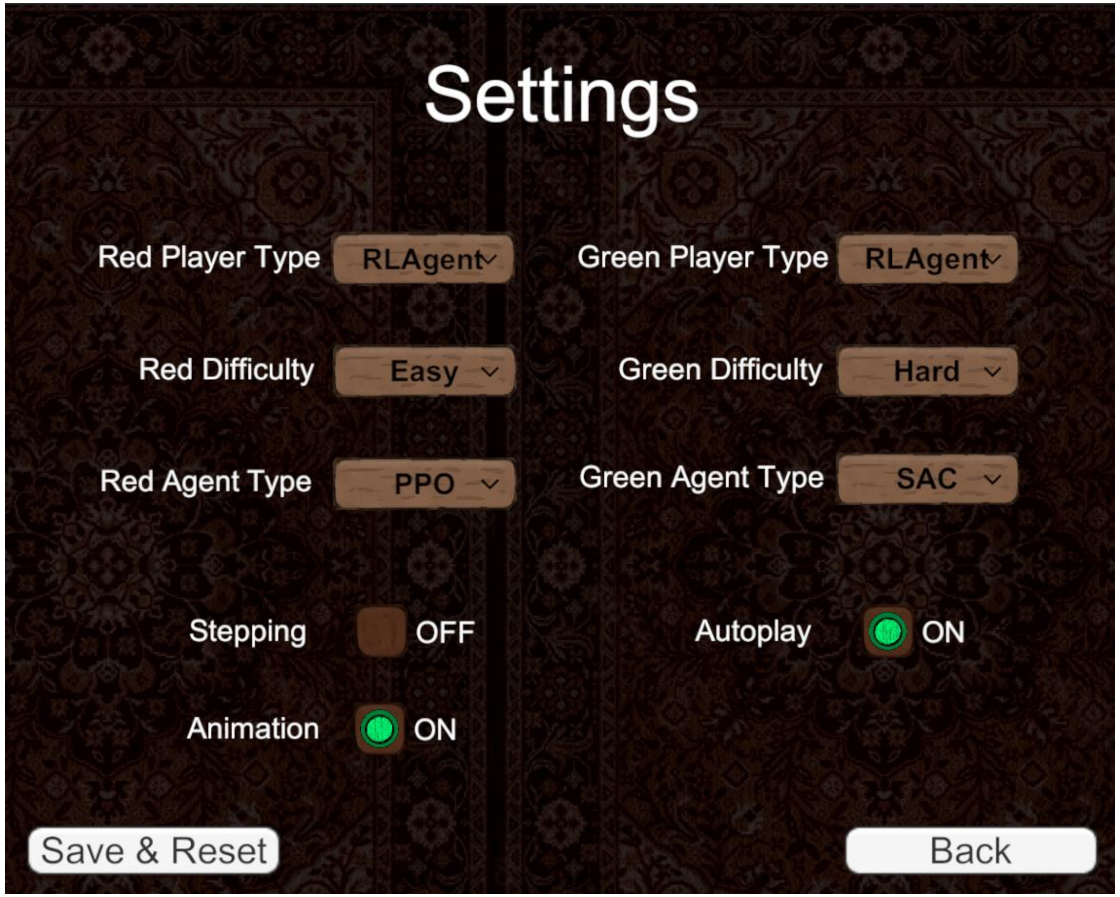
System Design and Implementation

Shologuti Game Features



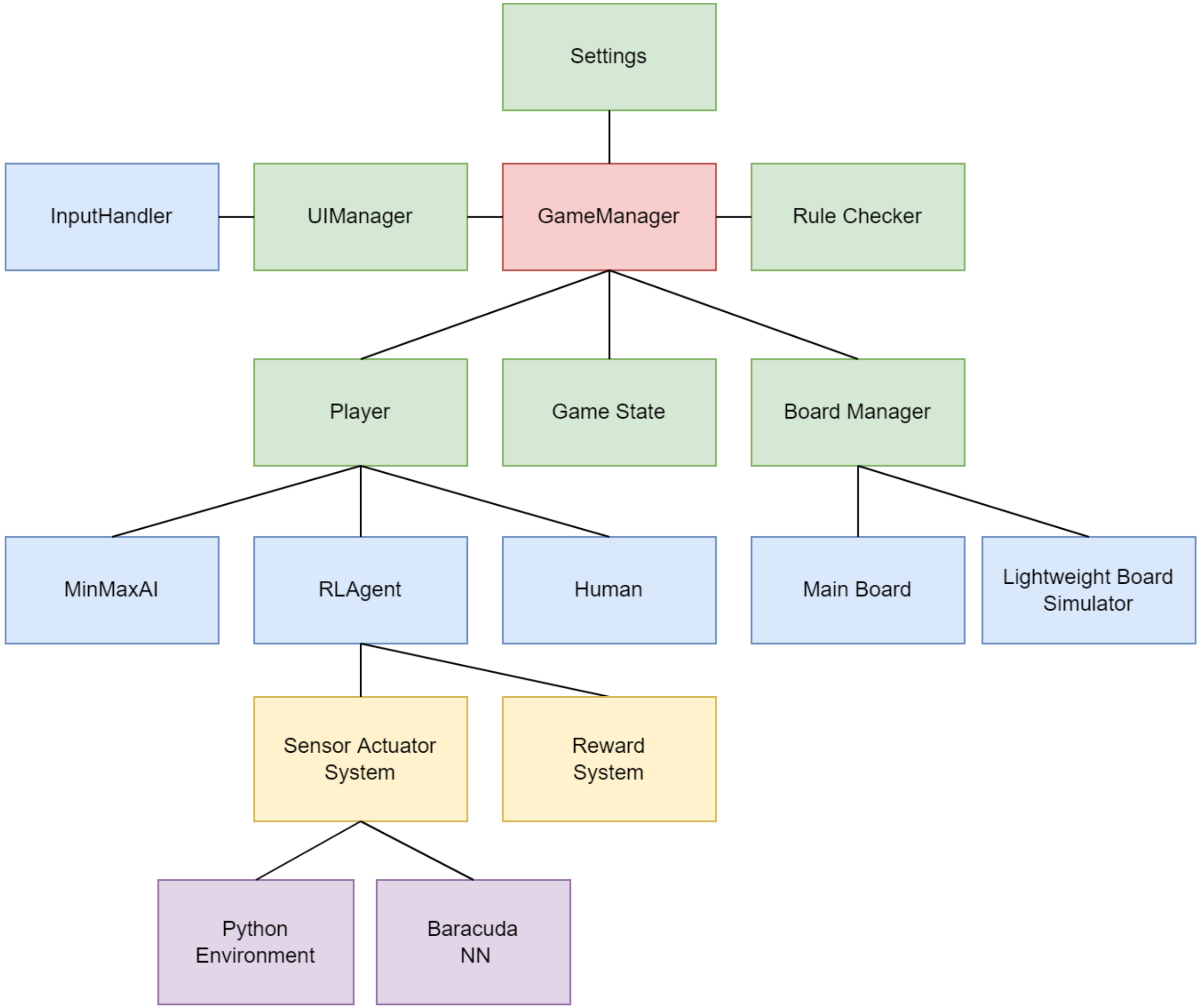
- User-friendly HUD
- GUI Interface
- Custom art and animation
- Game modes
  - Player vs Player
  - Player vs AI
  - AI vs AI

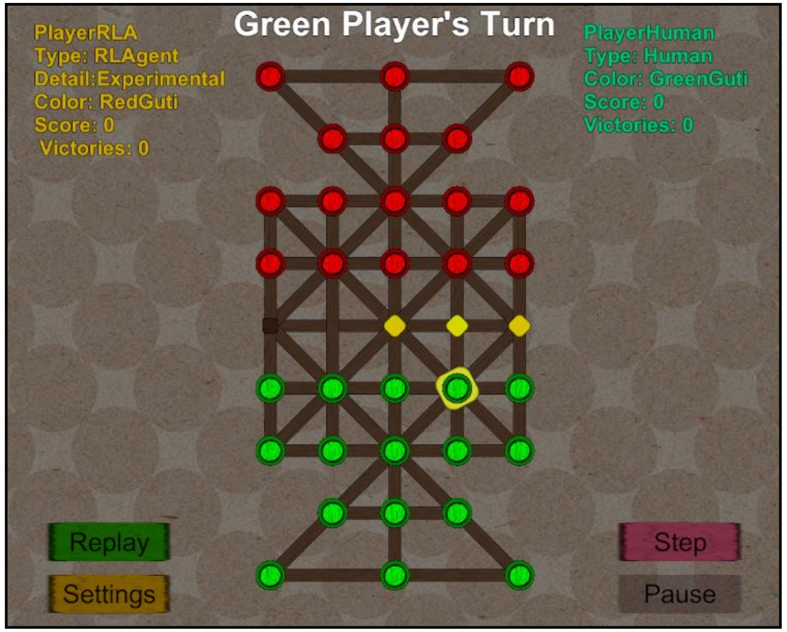
GUI Settings Menu



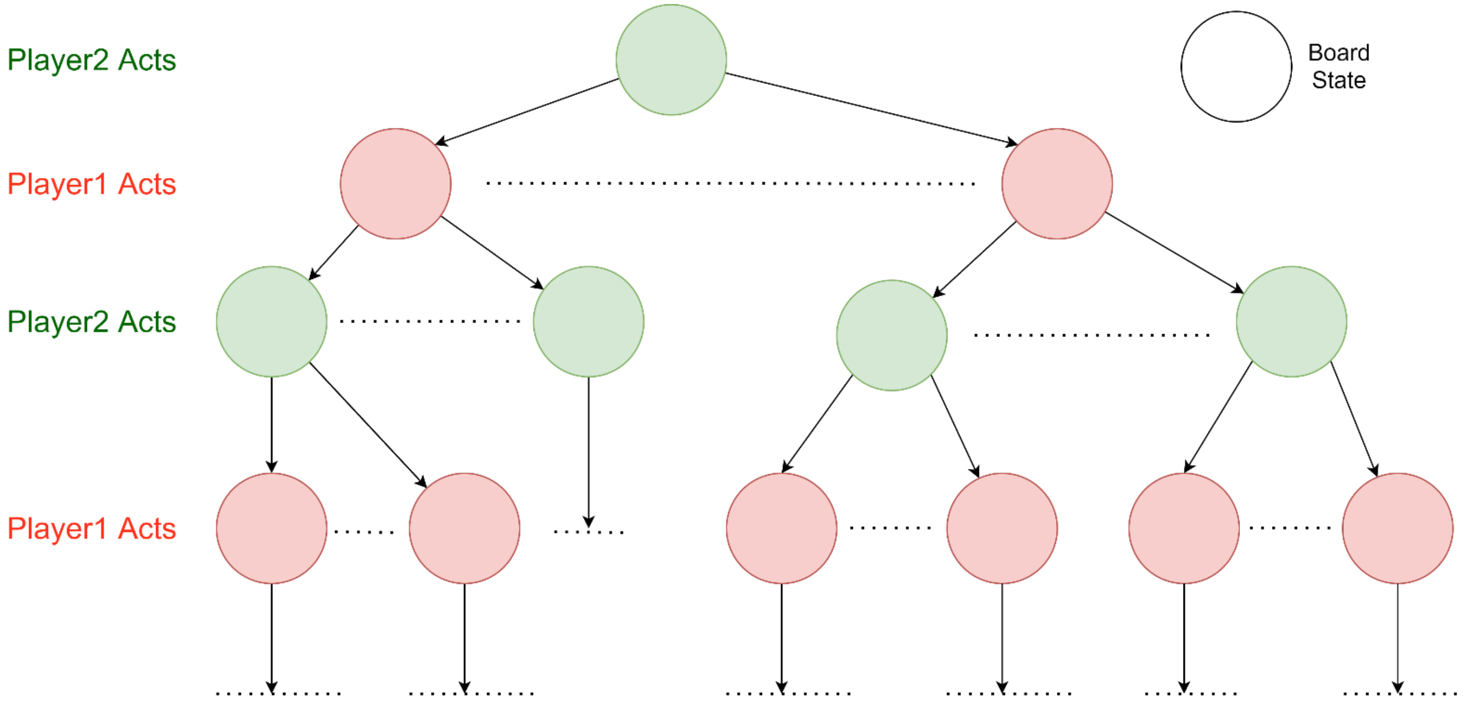
- Settings that can be controlled using the GUI Interface

System Overview

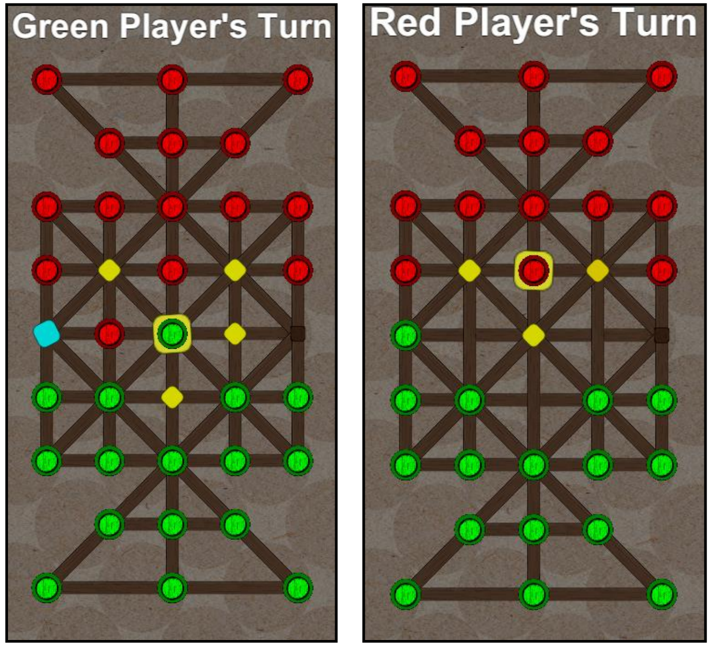




Game theoretic organization of Shologuti game

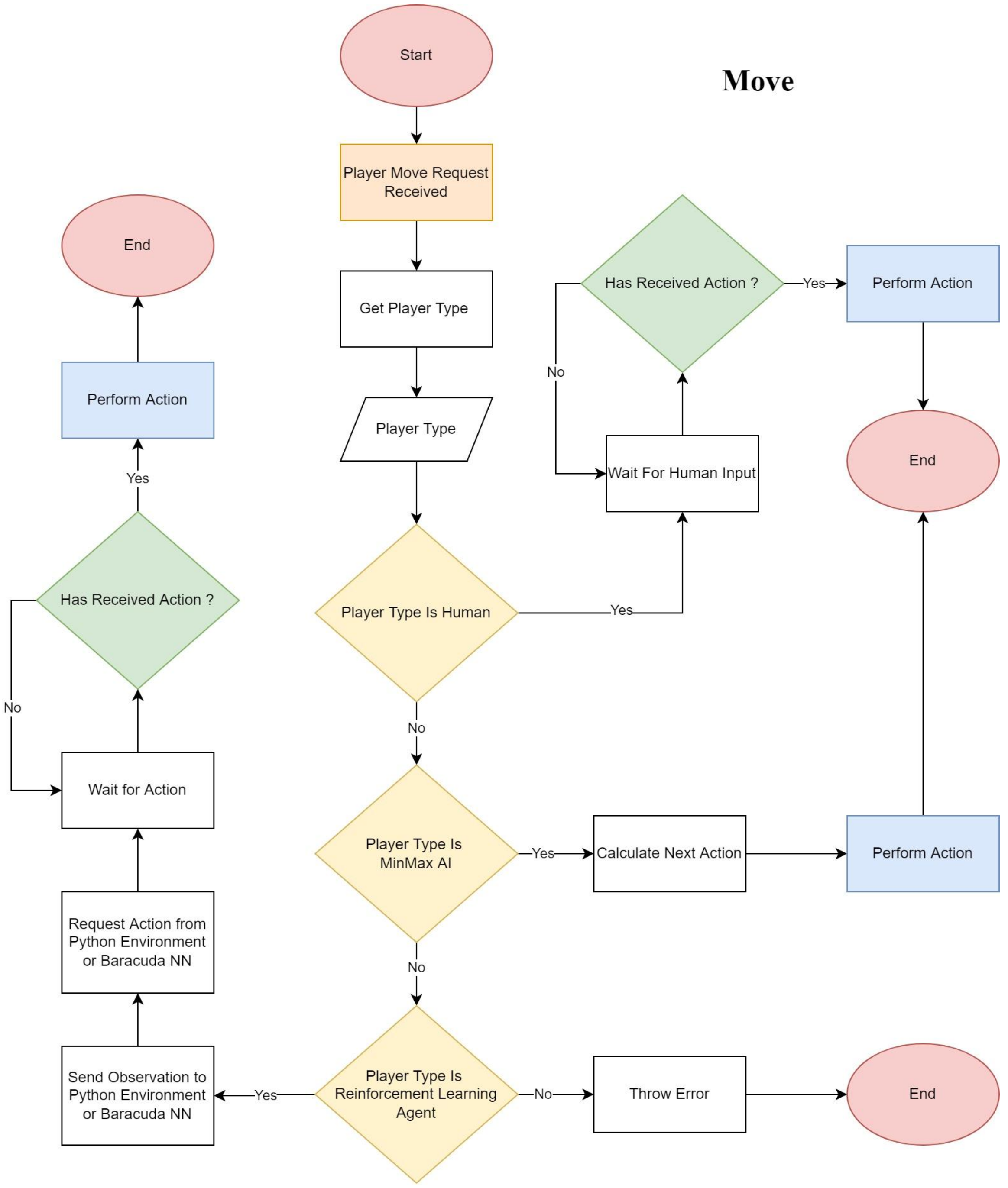


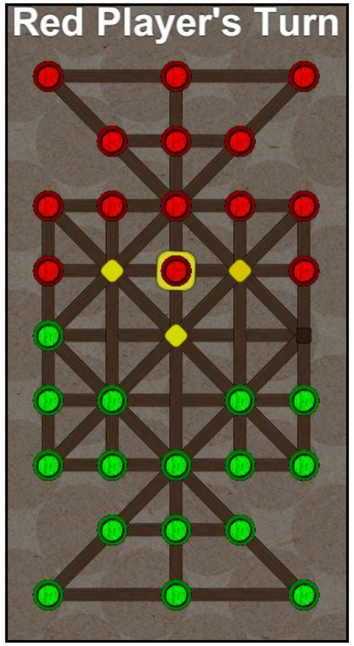
Game Tree



Board States

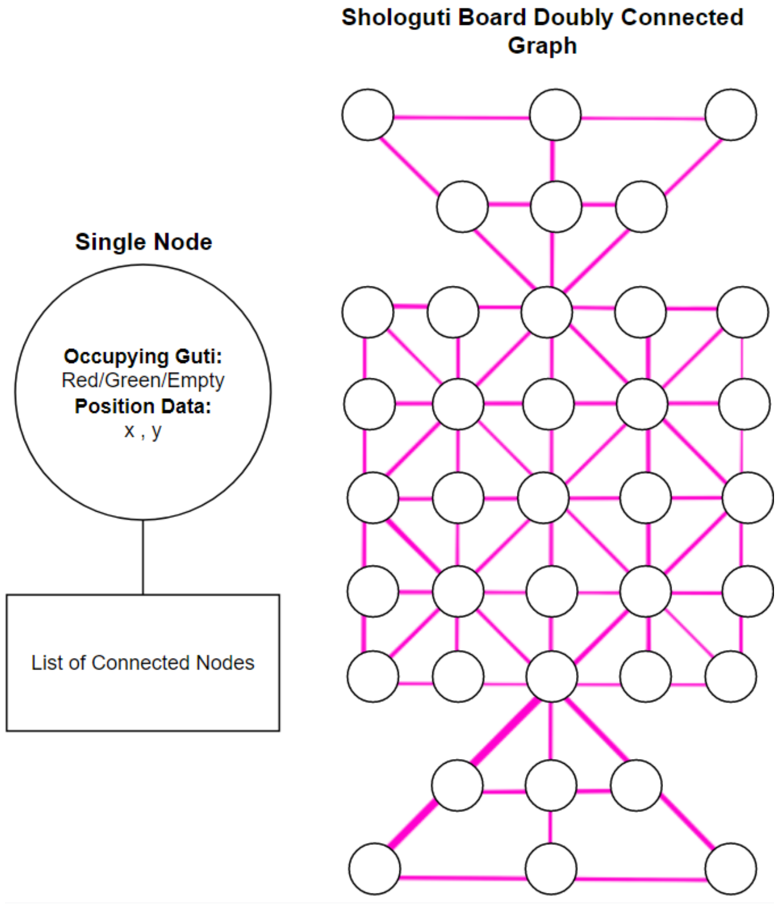
Making a Move

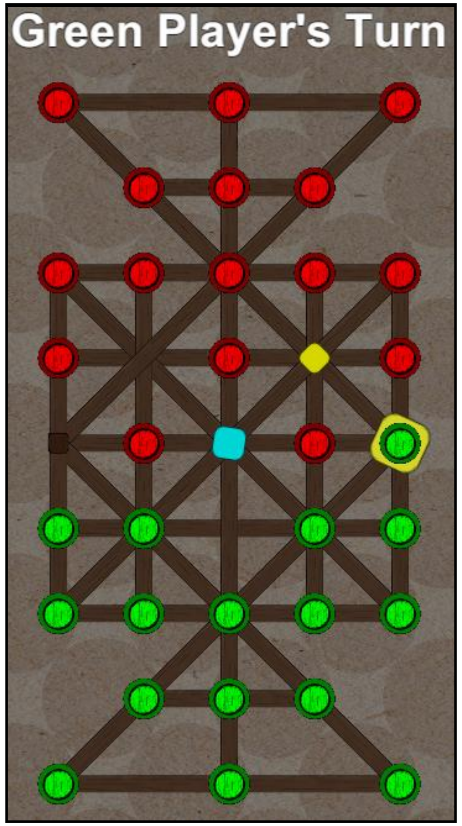




Core Game System Implementation

- Board setup
- Move generation/execution
- Rule checking
- Scoring system
- Multiple types of board game AI

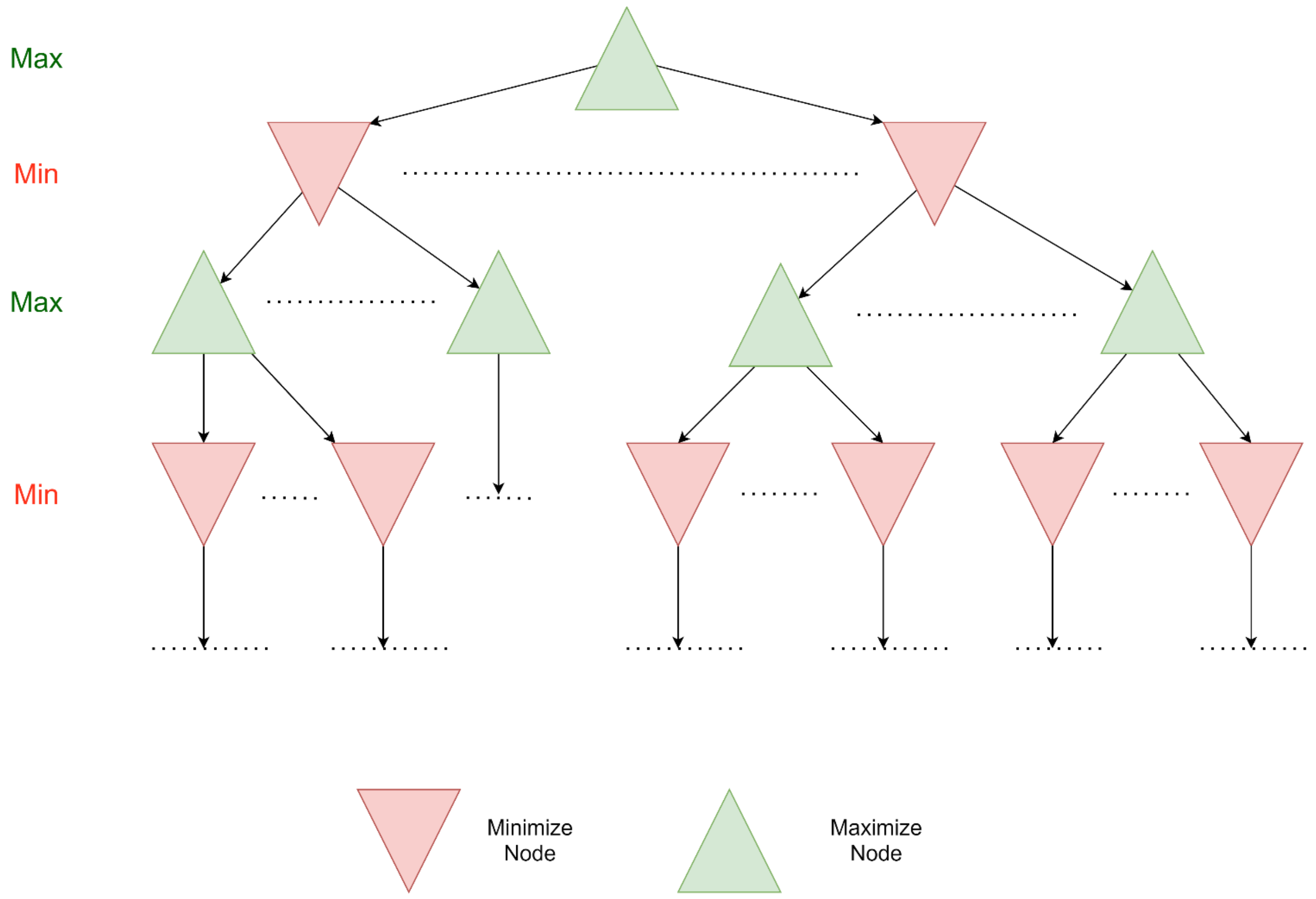


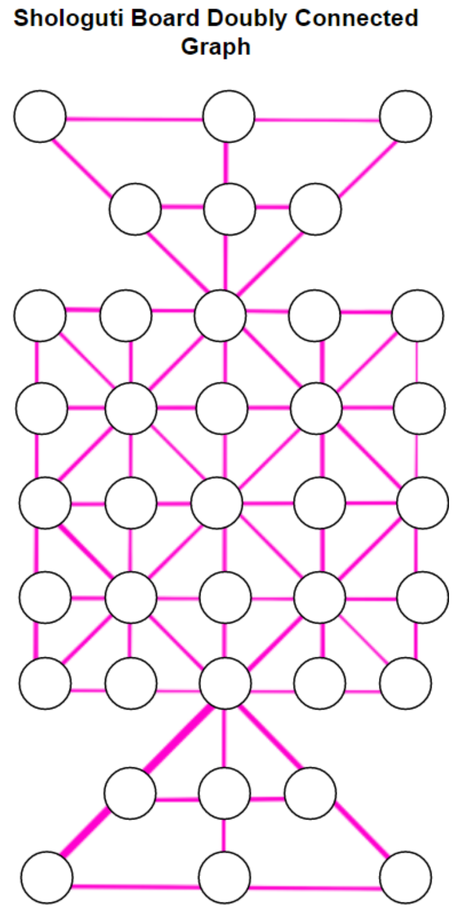


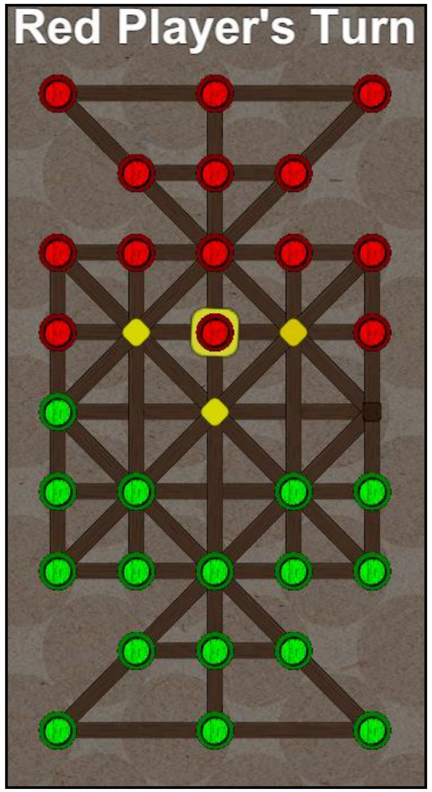
RL Component Features

- MinMax AI opponents with adjustable search depth
- RL-Agents capable of training with self-play
- Custom sensor actuator system
- Environment can train using Unity ML-Agents python training script
- Environment can be controlled through external python script

MinMax Search Based Agents







Types of RL Agents

- TD Agent
  - Temporal Difference (TD)
- Actor Critic Agents
  - Proximal Policy Optimization, PPO
  - Soft-Actor Critic, SAC

Observation Representation for Neural Networks

1	1	1	1	1	1	1	1	1	1
1	1	1	1	1	1	0	0	0	0
0	2	2	2	2	2	2	2	2	2
2	2	2	2	2	2	2			

Table 4.1 Initial board state observation

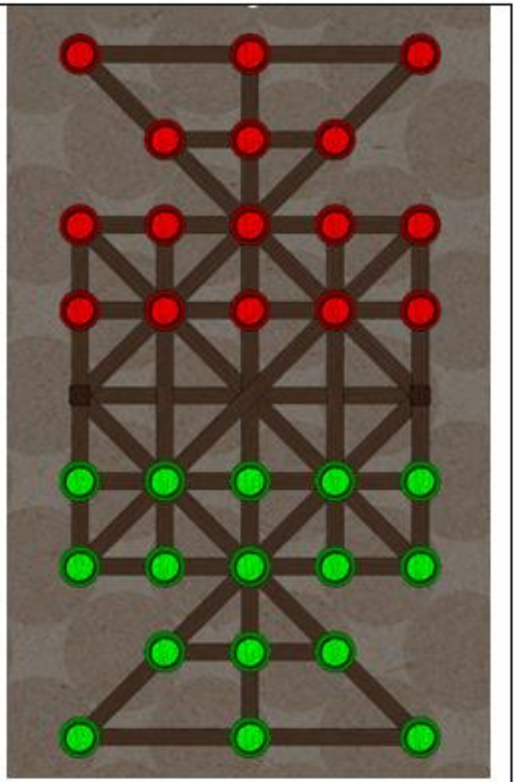
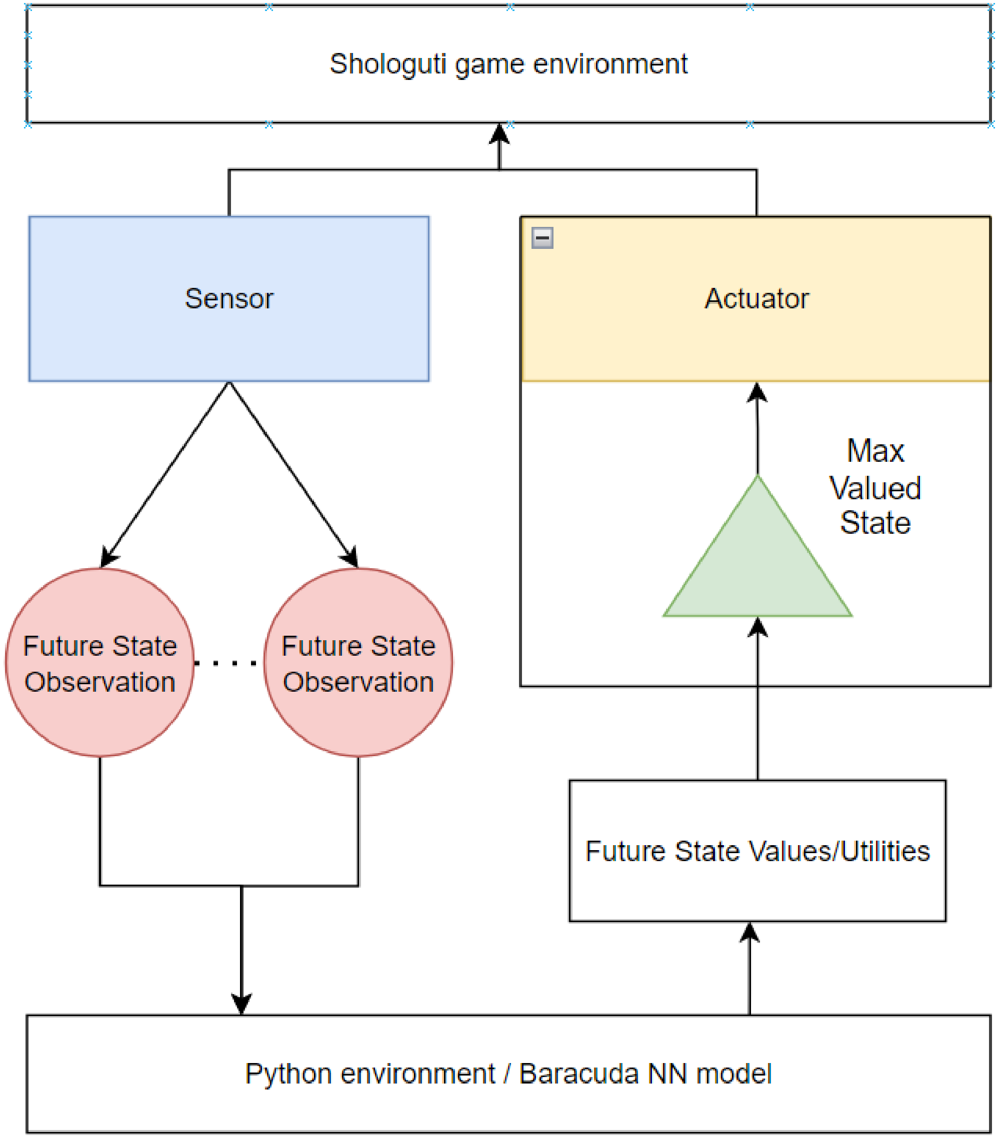
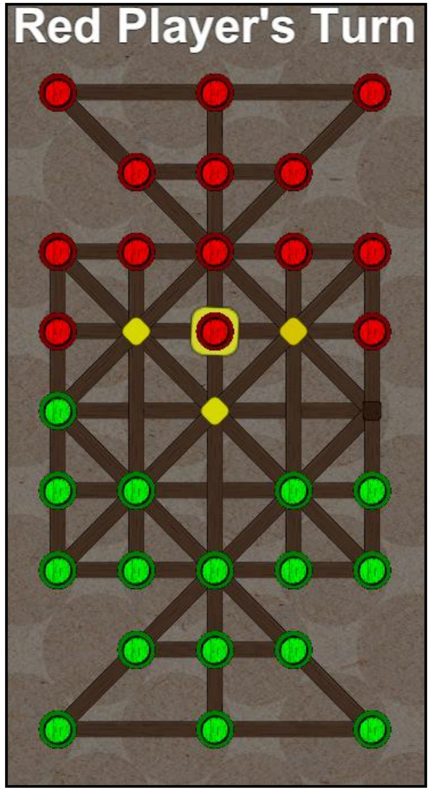


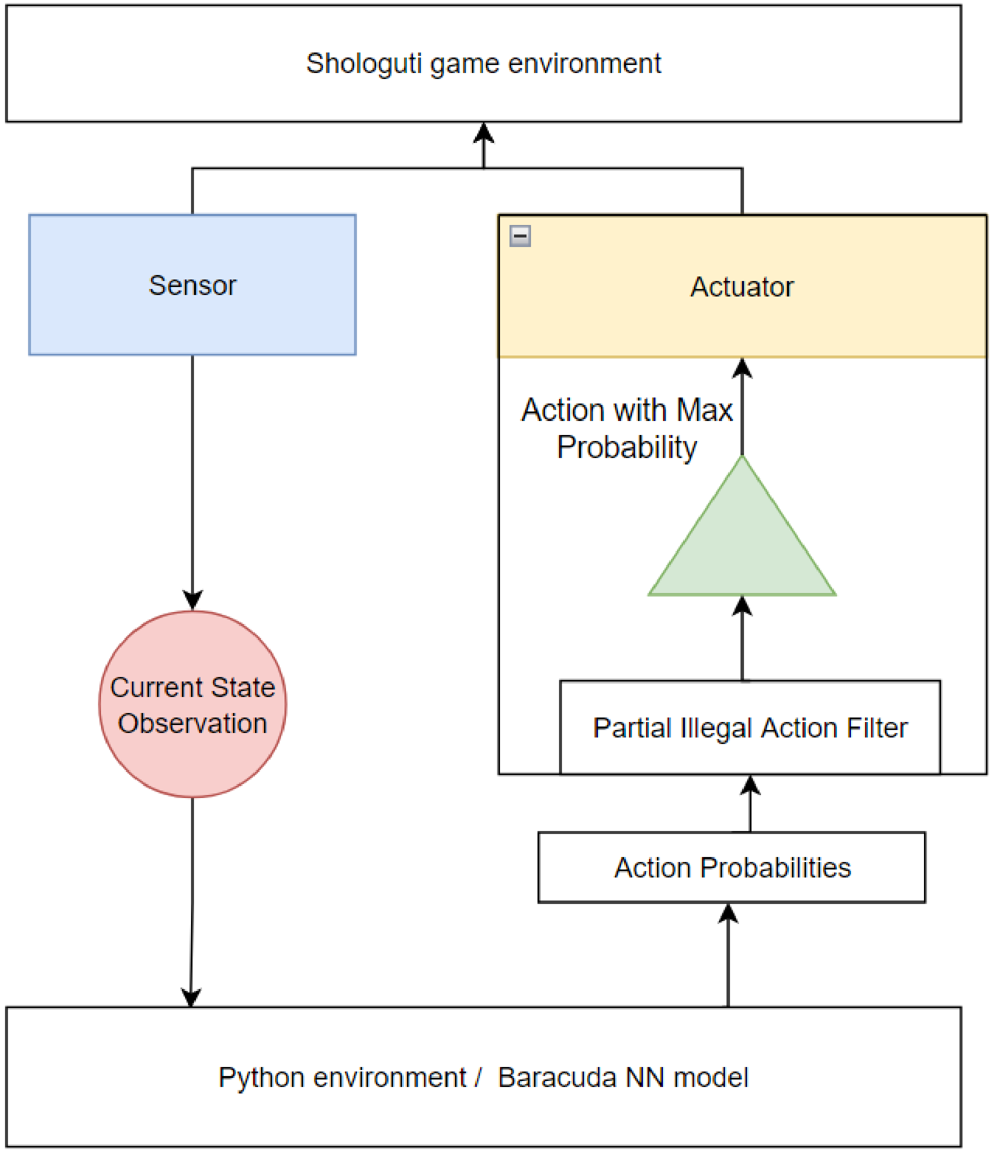
Figure 4.4 Initial board state

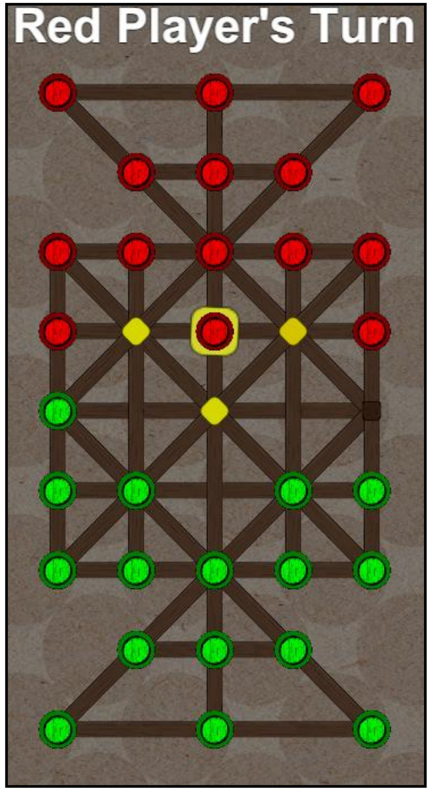
Temporal Difference (TD) Agent Decision/Inference



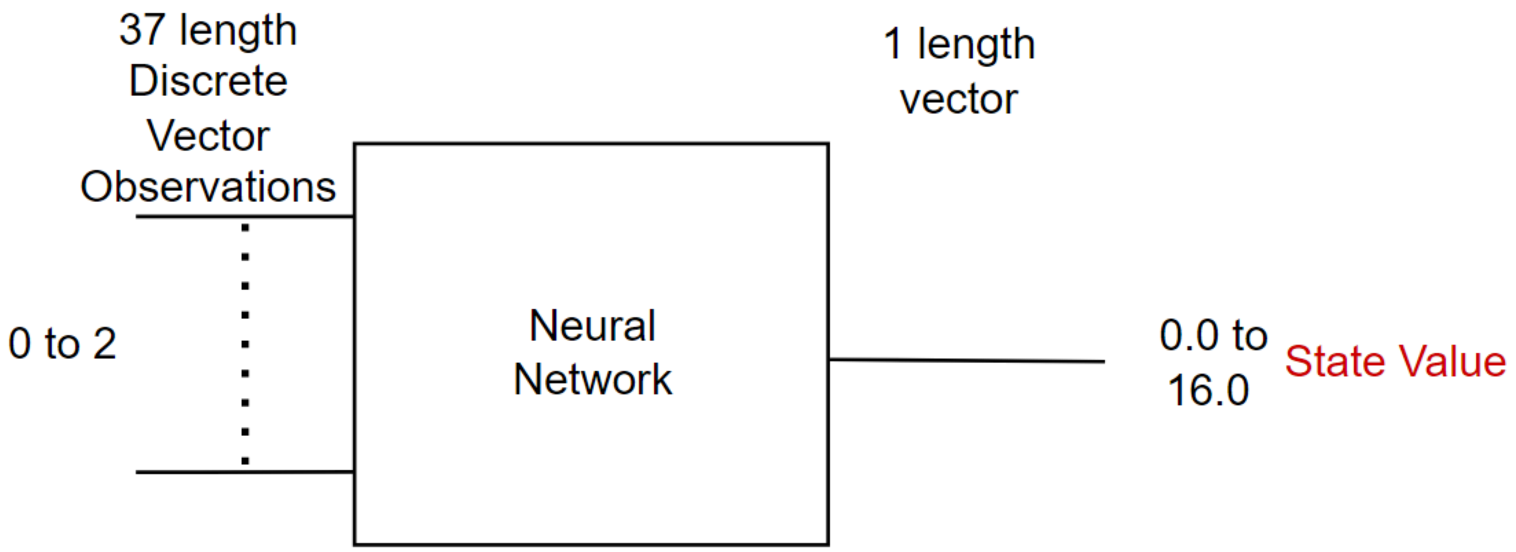


Actor Critic (AC) Agent Decision/Inference

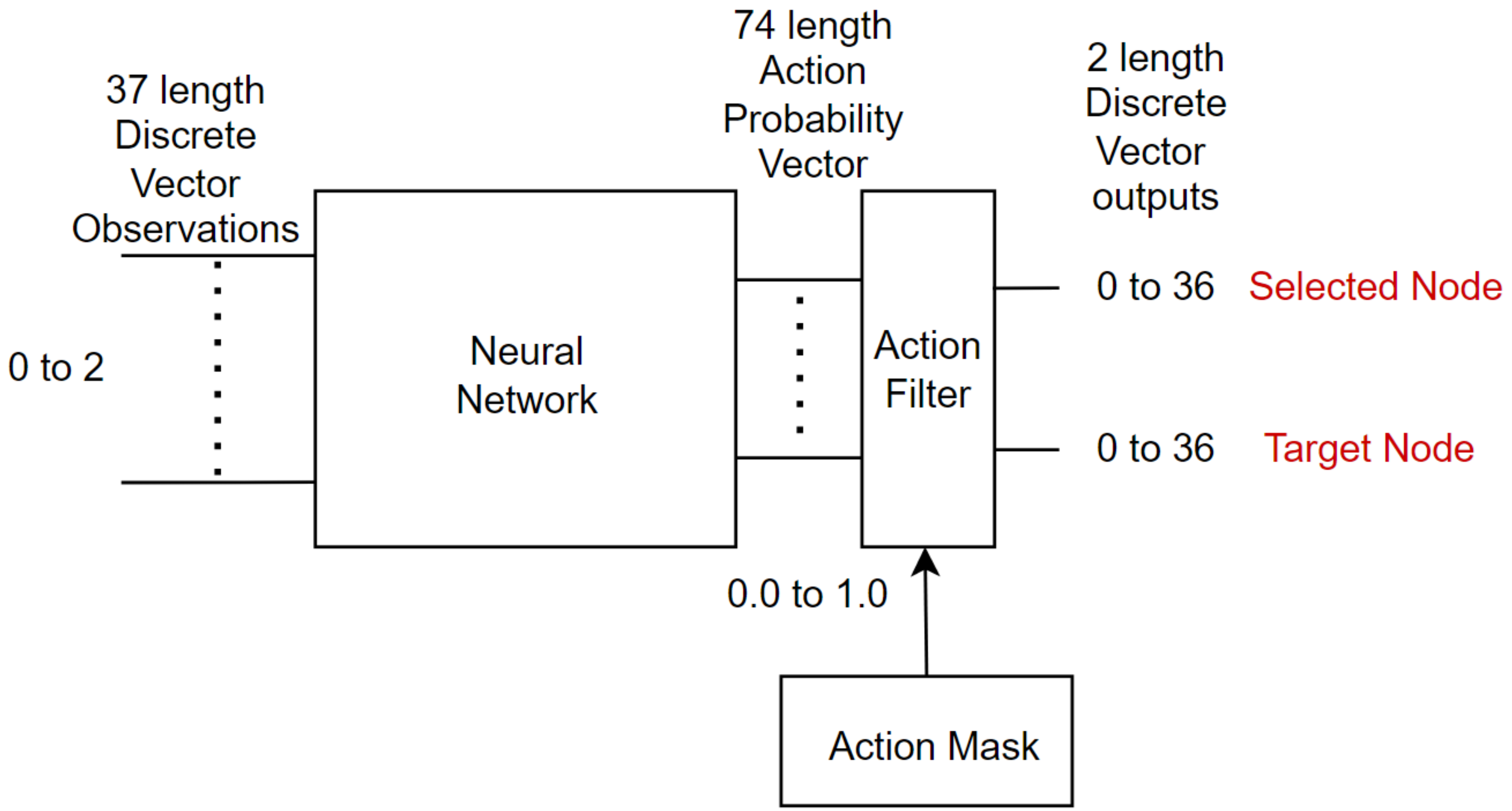




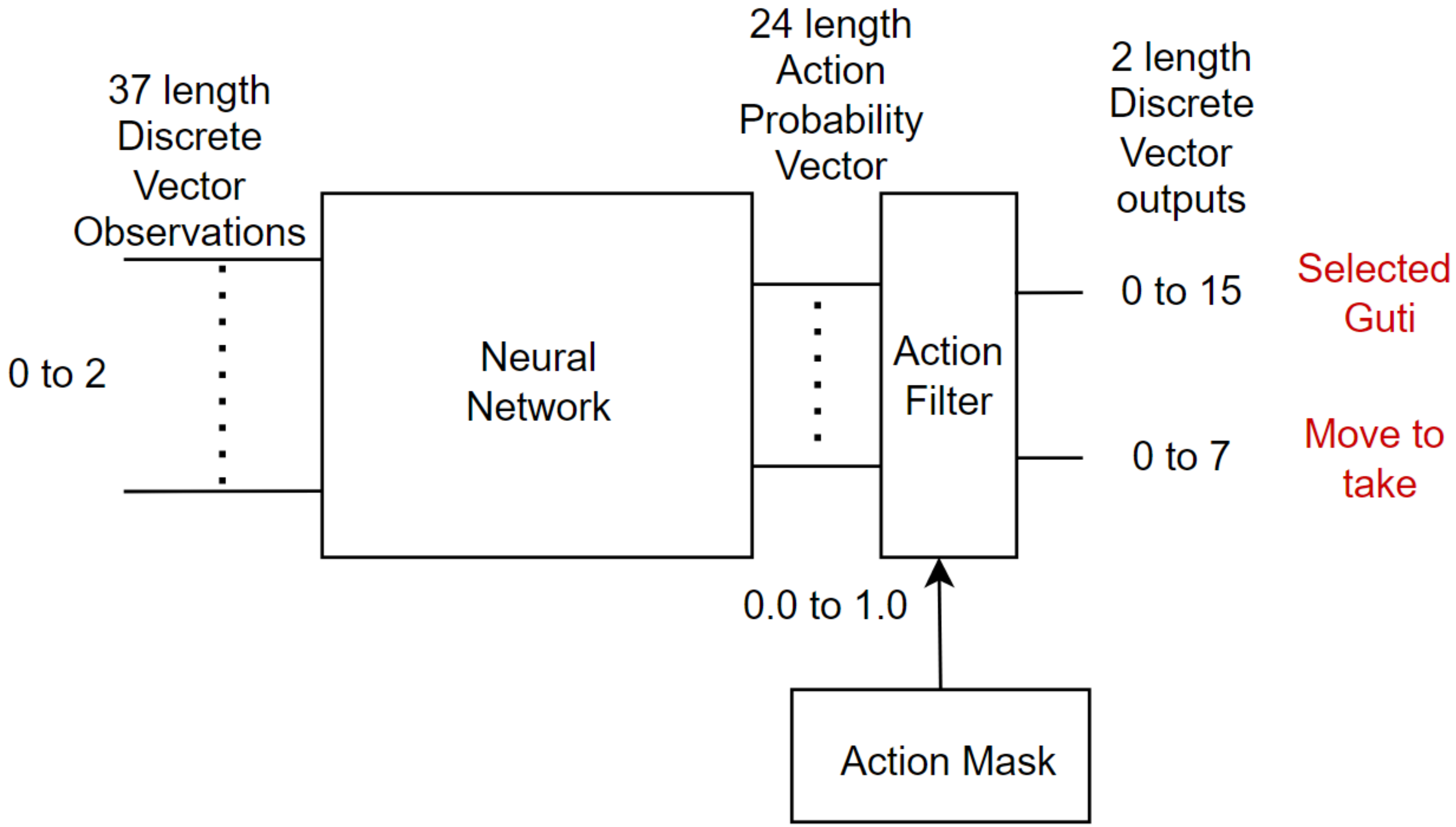
Temporal Difference (TD) Neural Network Architecture



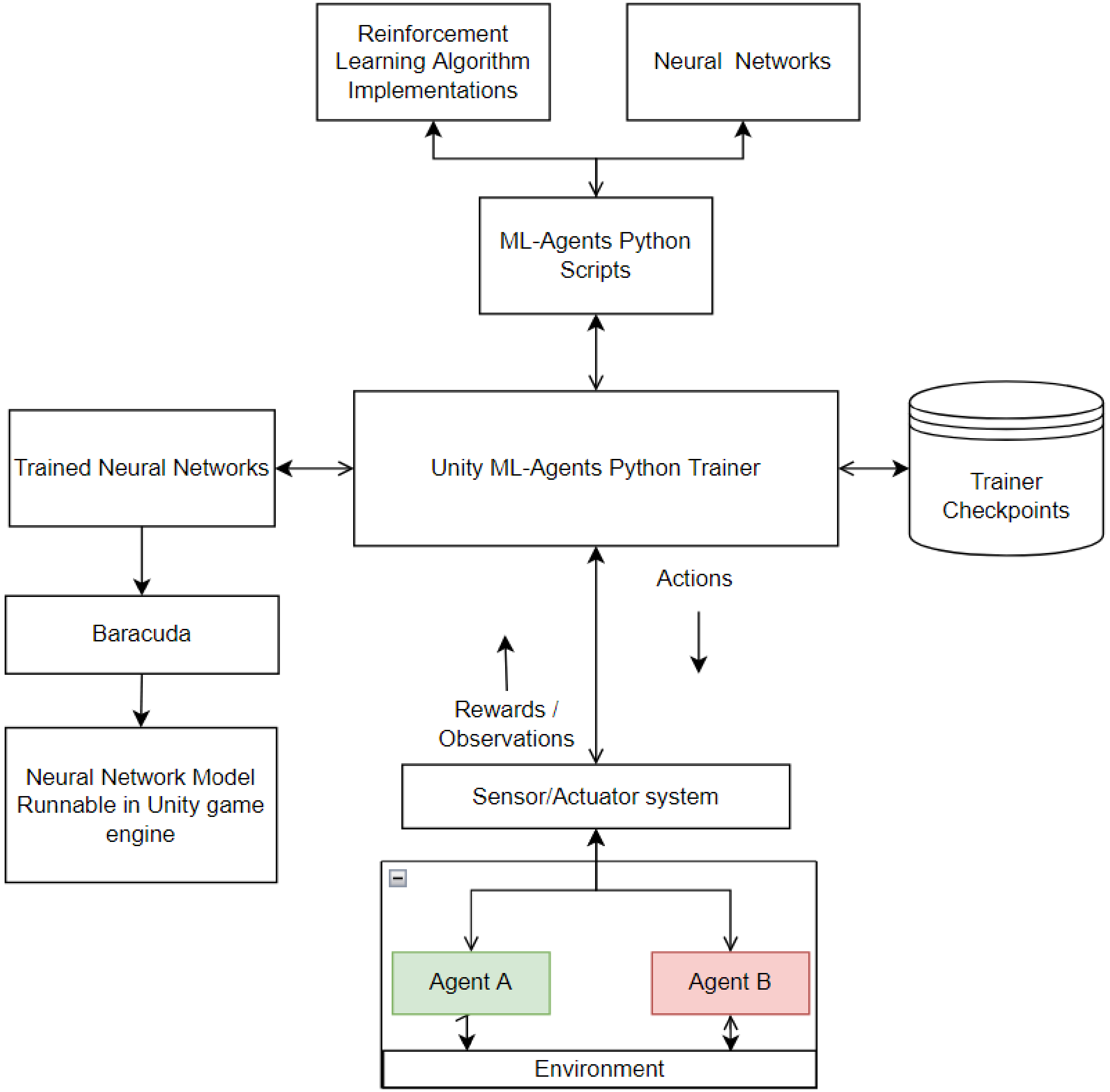
Actor Critic 1 Neural Network Architecture



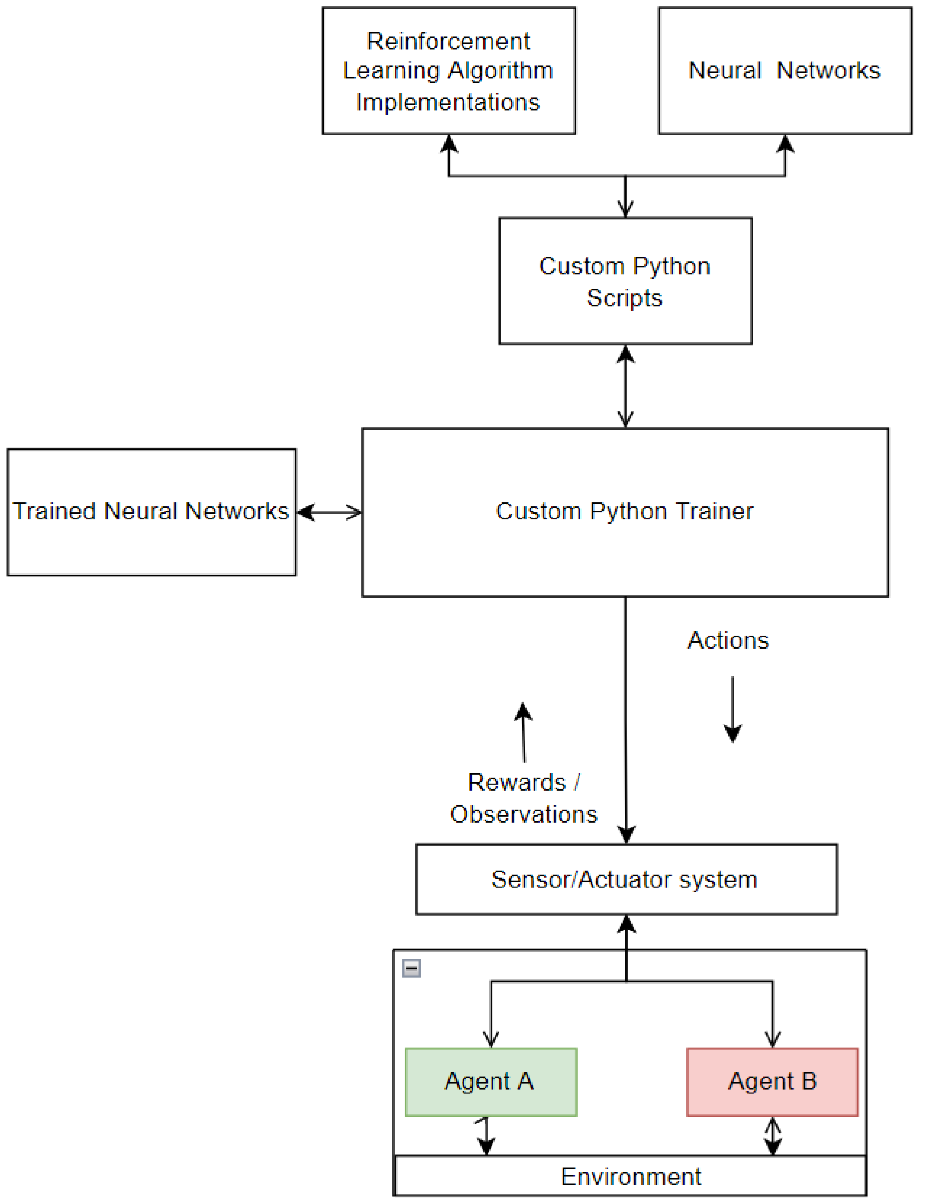
Actor Critic 2 Neural Network Architecture



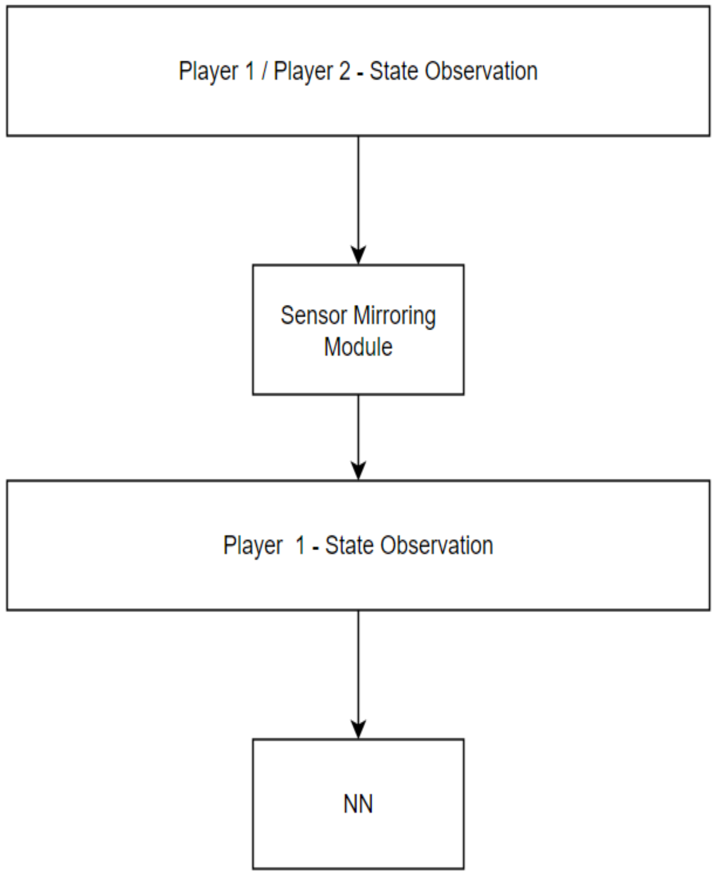
Agent Training with Unity ML-Agents Trainer

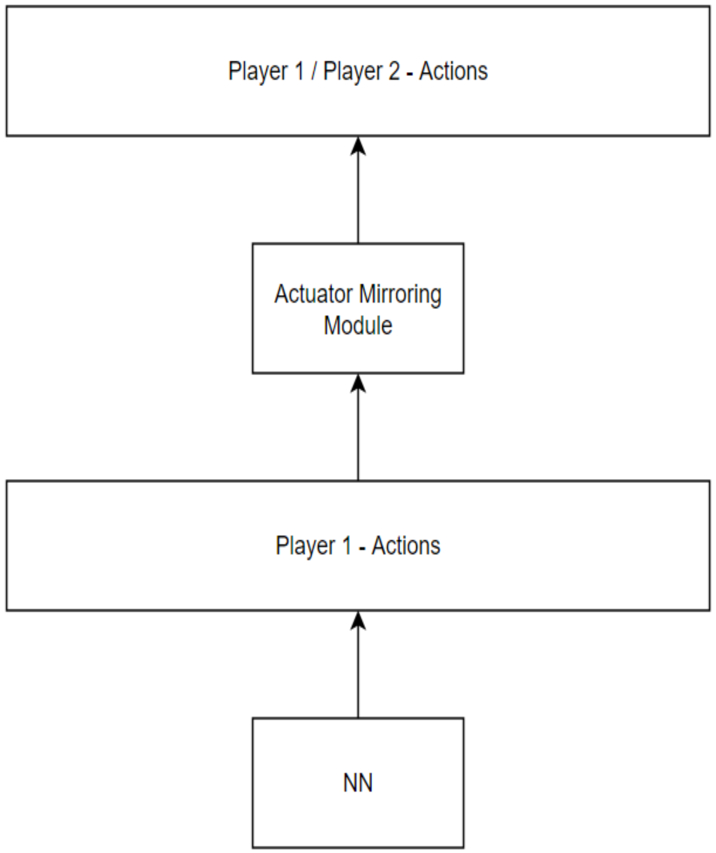


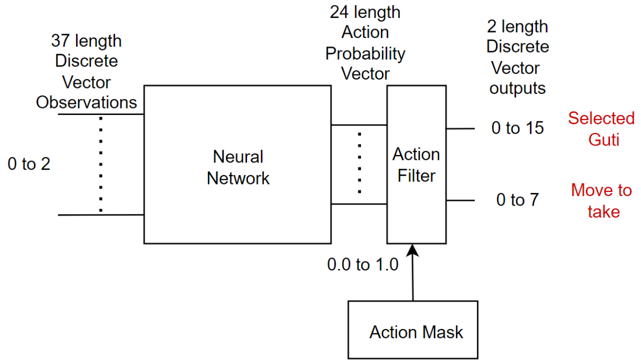
Agent Training with Custom RL Algorithms

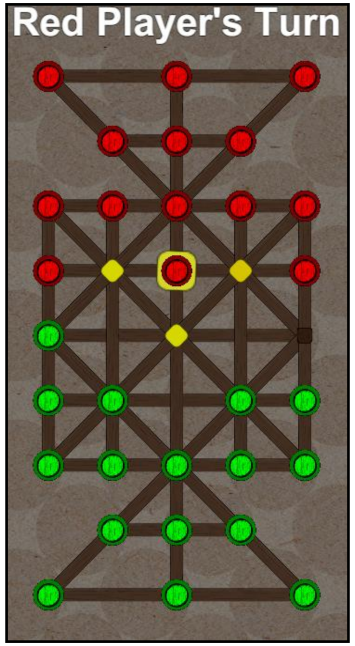


State and Action Mirroring

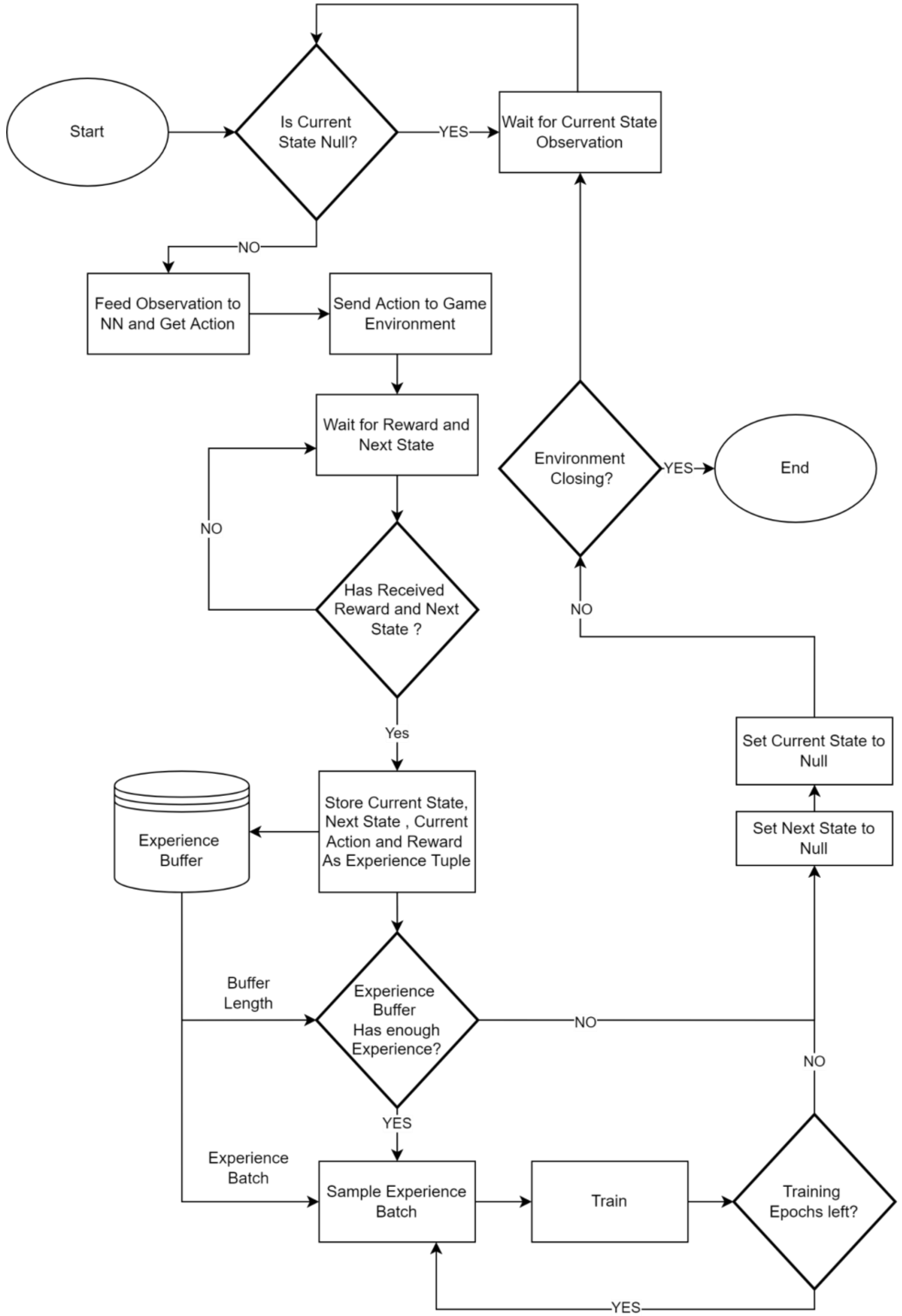


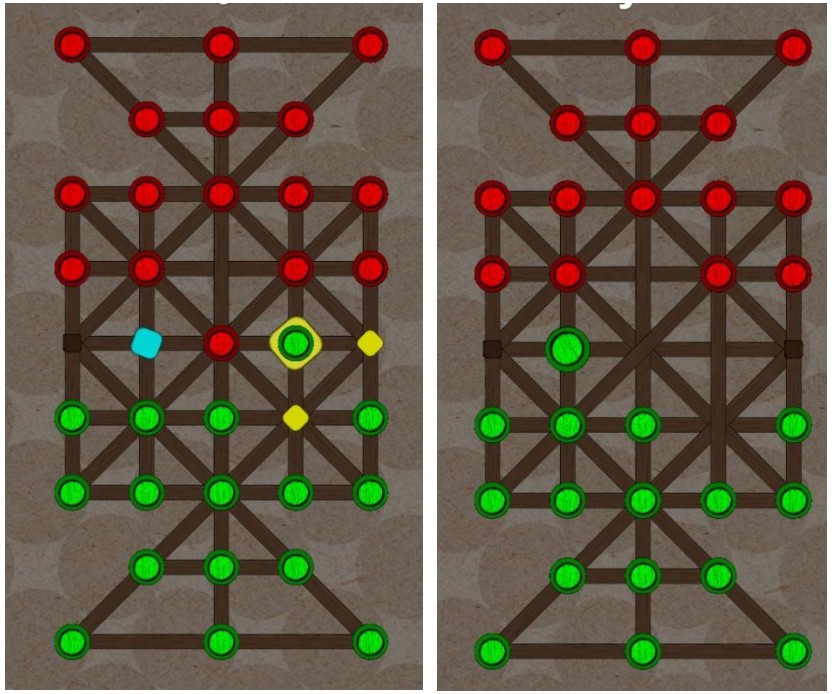




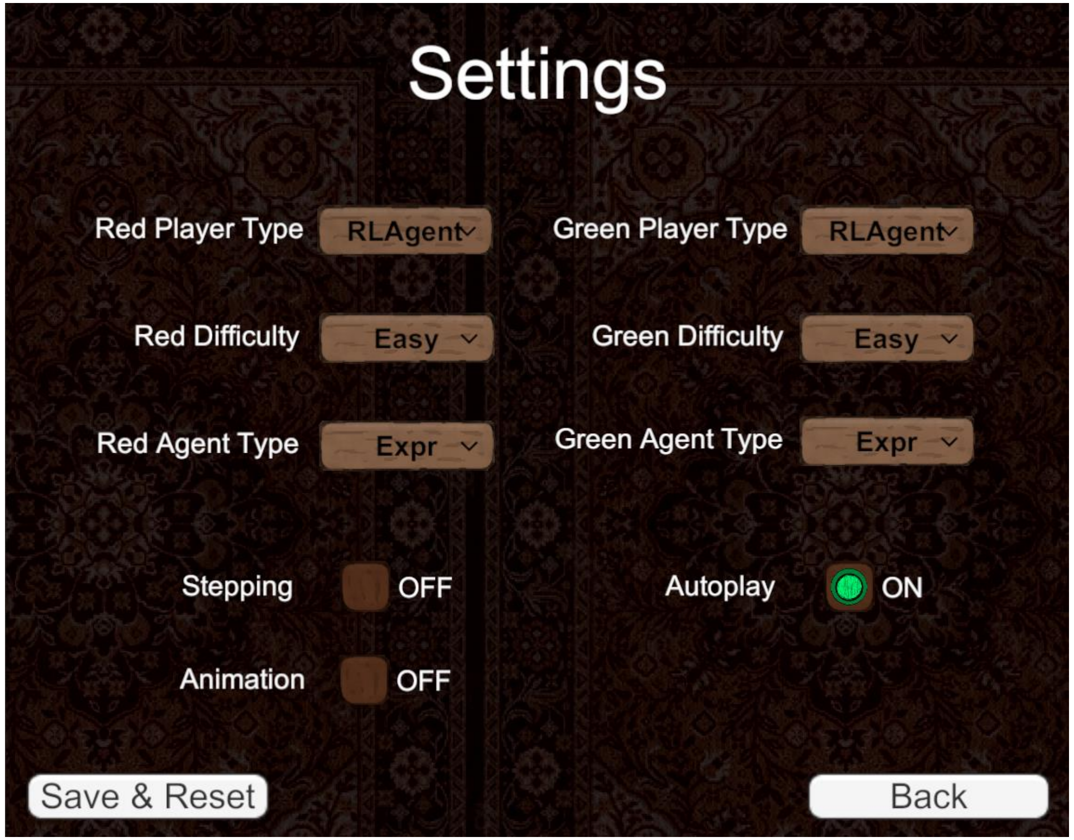


Generalized RL Algorithm



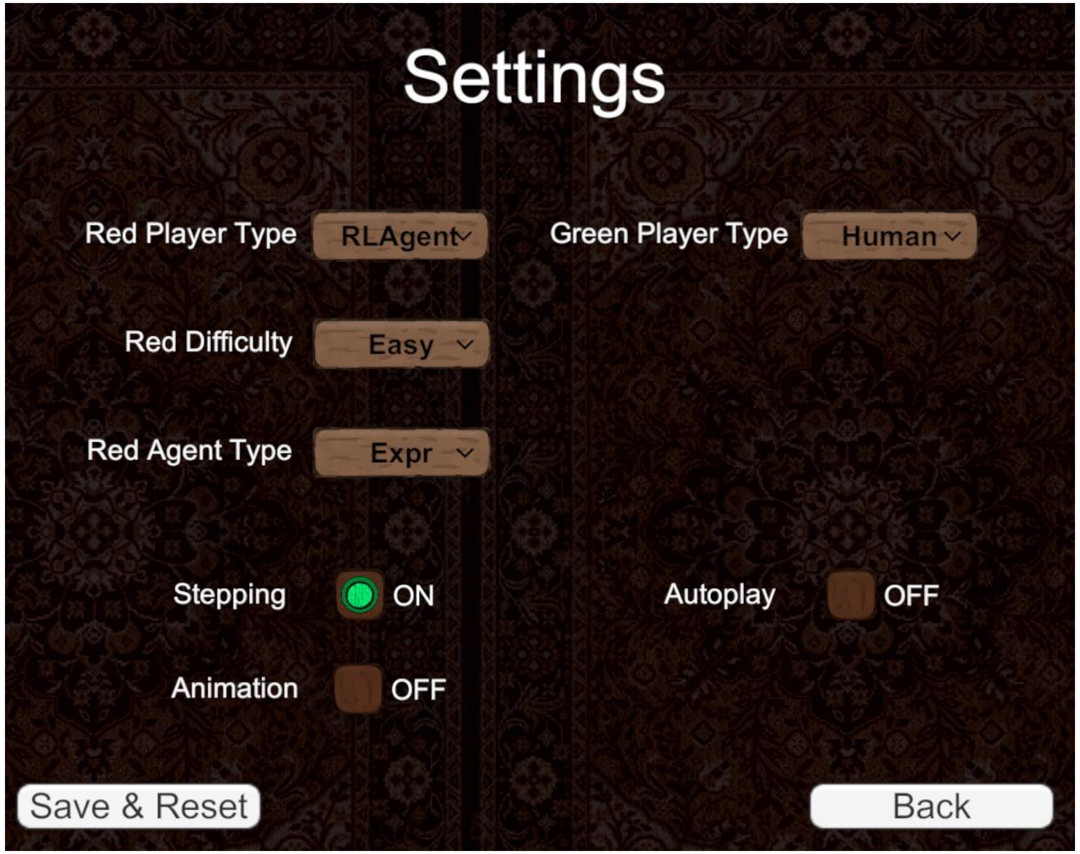


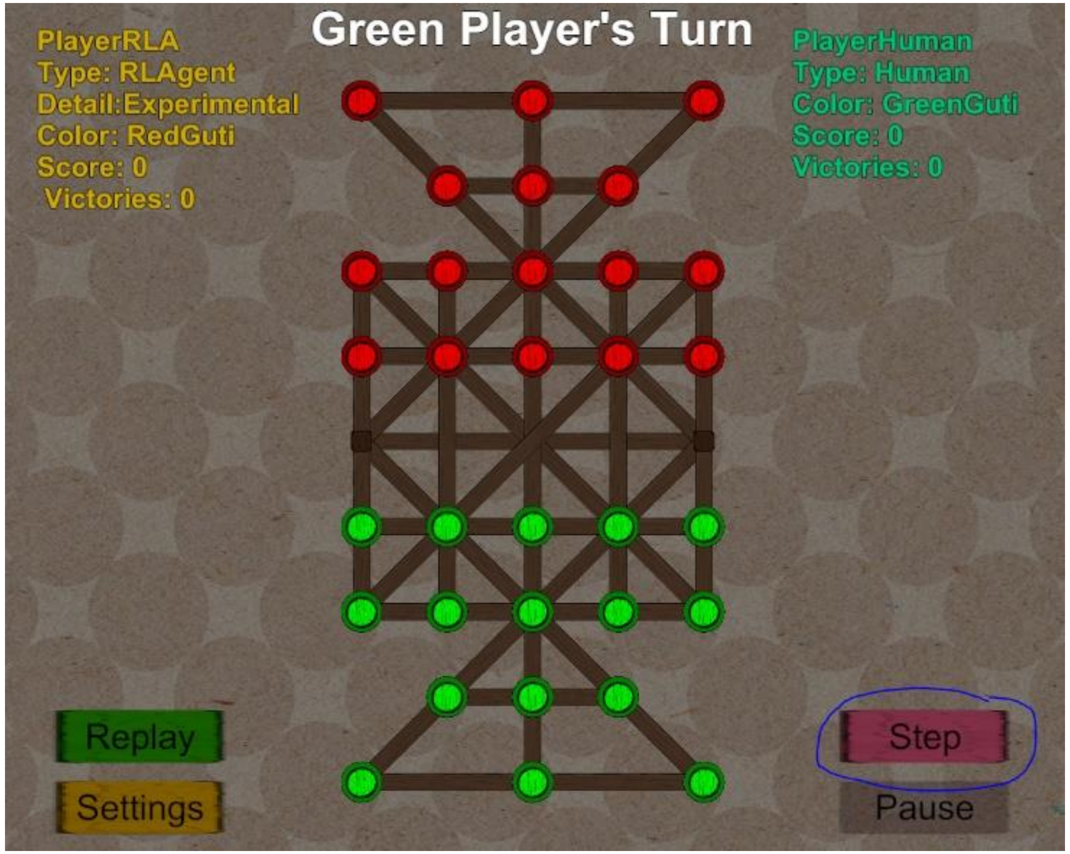
Settings for RL Component



- The type of enemy to train against
- Difficulty of the type of enemy selected
- Stepping mode toggle
- Animation toggle
- Autoplay toggle

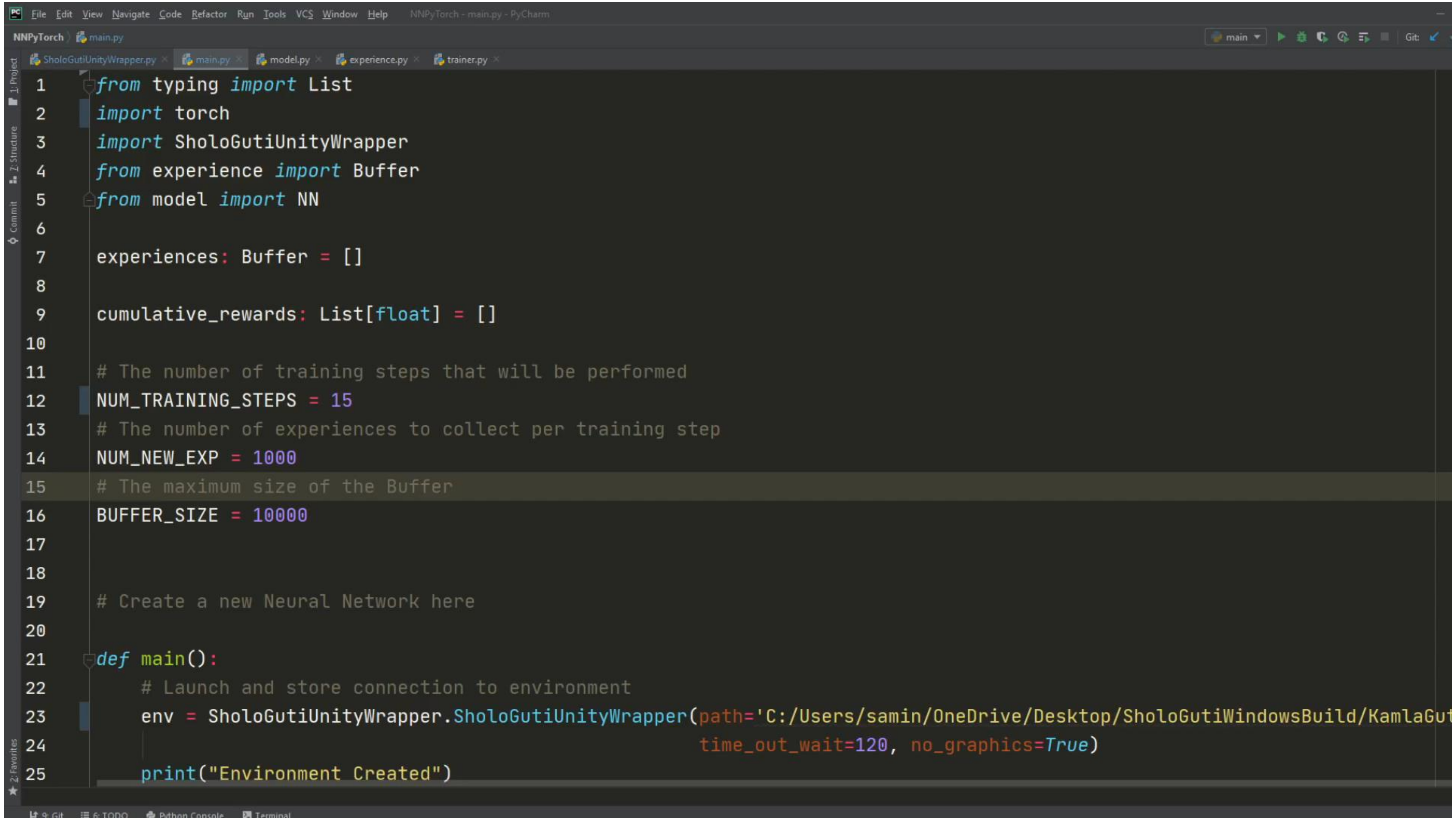
Stepping Mode





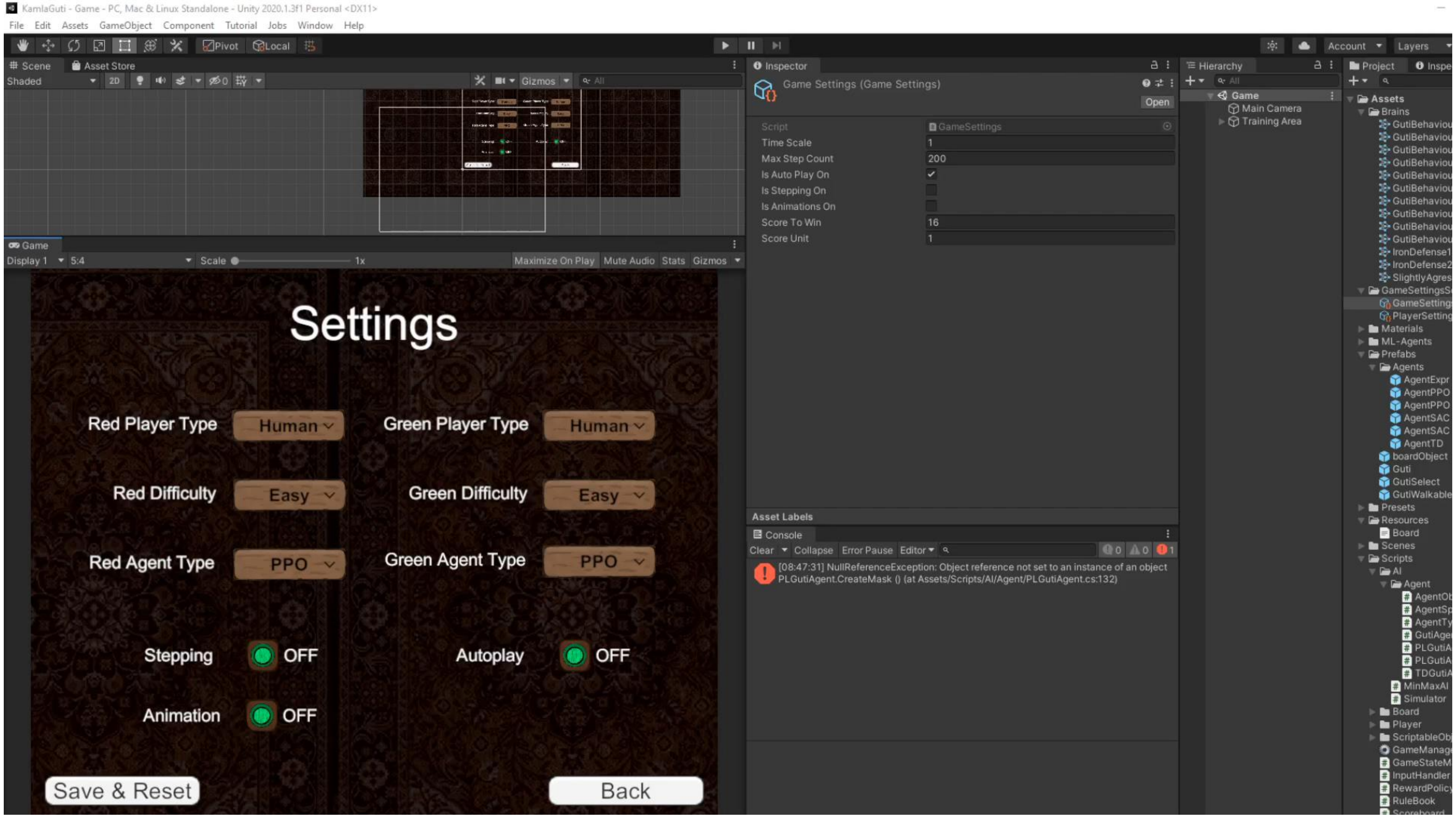
Connecting to Environment with Python

- Video or live demo



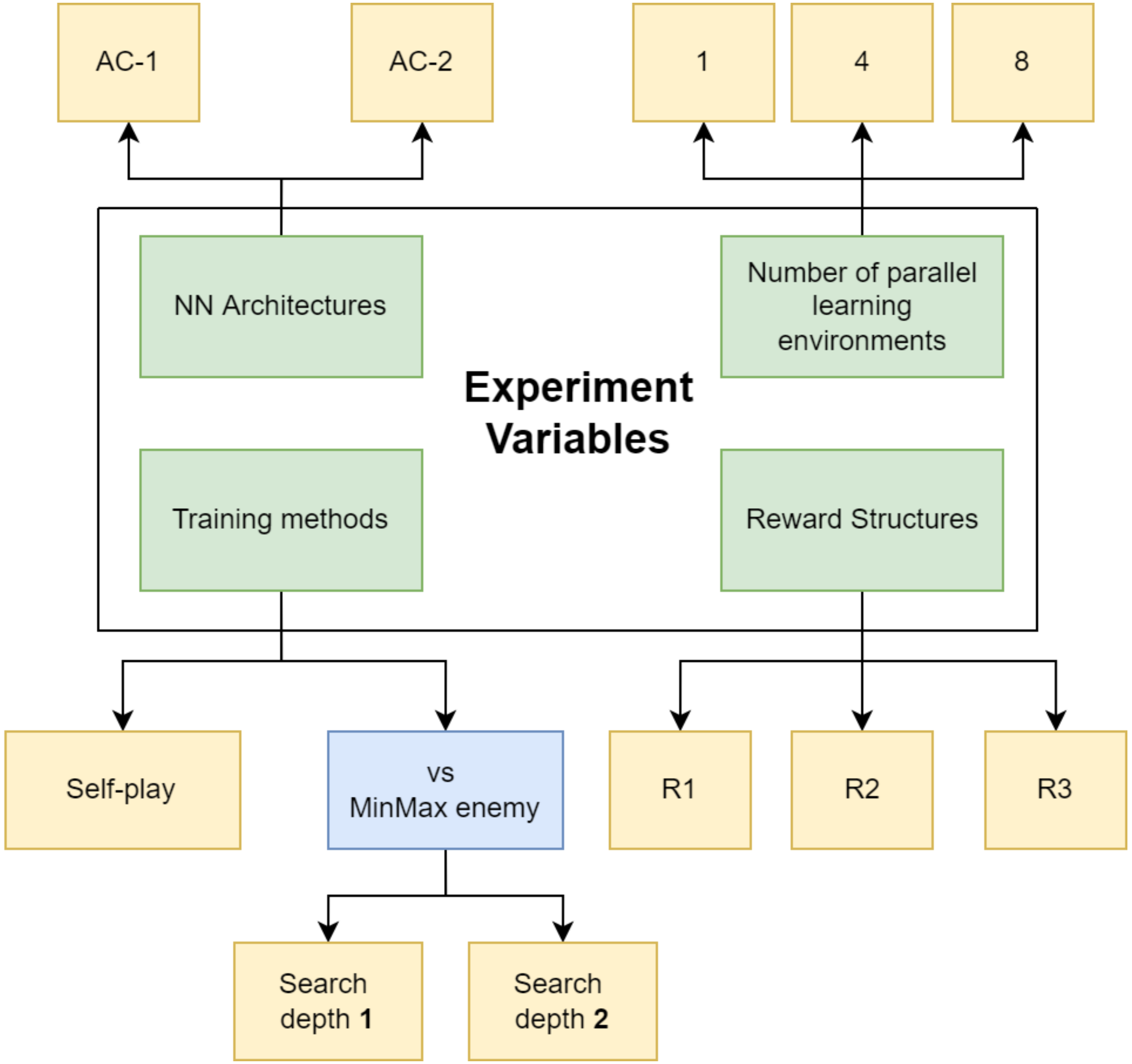
Using Unity ML-Agents Trainer to Train Agents

- Video or live demo



Results and Analysis

Training Setups and Experiments



- 7 training experiments
- Divided into 4 different training setups
- Every training setup keeps some variables constant and changes others

Training Setup 2 (TS2)

Experiment name	**NN architecture**	Training method	Reward structure	Number of parallel learning environments
TS2AC2Exp1	***AC-2***	Vs **MinMax,** search depth **1**	R1	1
TS2AC2Exp2	***AC-2***	Vs **MinMax**, search depth **2**	R1	1

Table 5.2 Training Setup 2

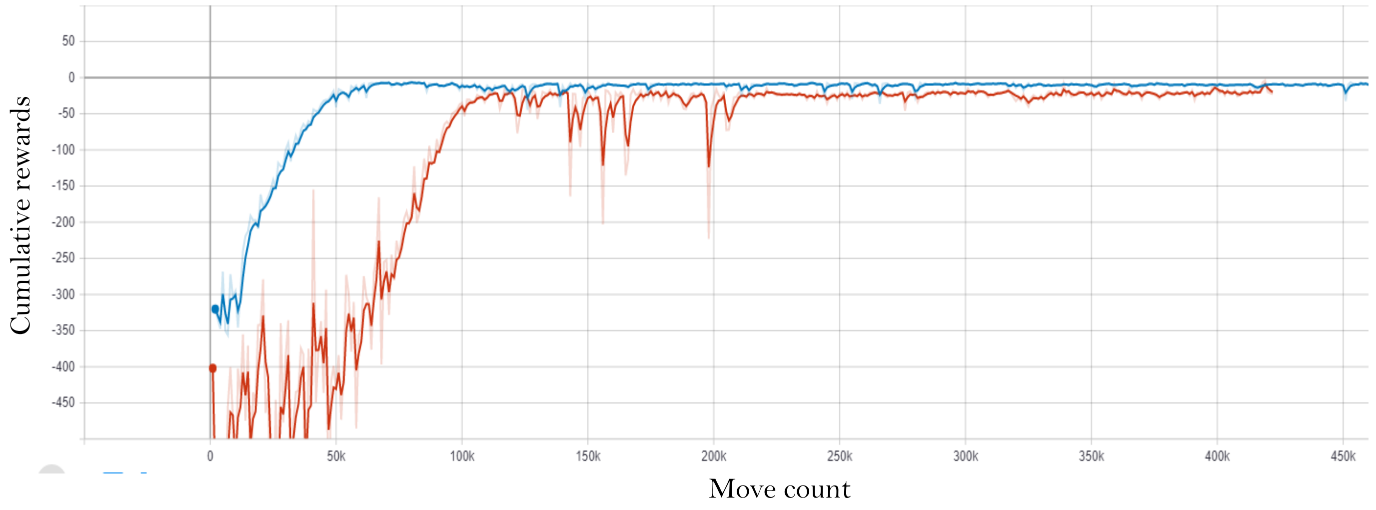
1. **Training setup 2 experiment 1 (TS2AC2Exp1):** SAC and PPO training with AC-2 architecture against agent using MinMax searching algorithm with depth 1
2. **Training setup 2 experiment 2 (TS2AC2Exp2):** SAC and PPO training with AC-2 architecture against MinMax searching algorithm with depth 2

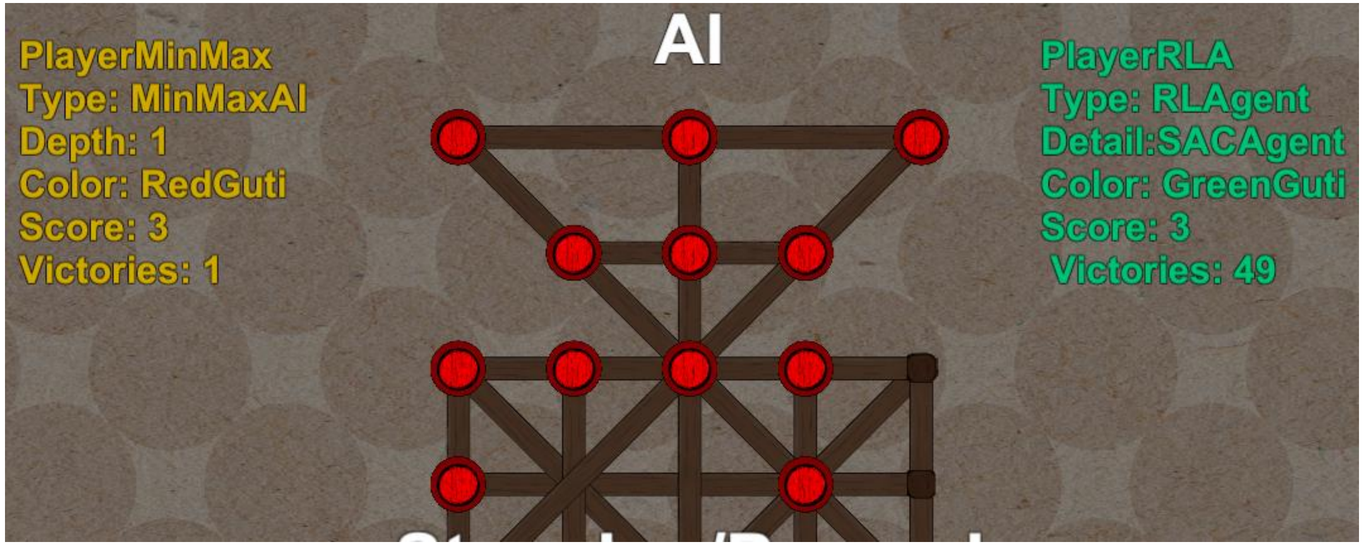
Reward Structure 1 (R1)

Reward / Penalty Condition	Reward	Ratio to maximum reward
Reward for wining a match	16	1
Reward per enemy guti captured from enemy	1	1/16th
Reward for drawing a match	0	0
Penalty for losing a match	-16	-1
Penalty per guti lost to enemy	-1	-1/16th
Penalty per legal move	-0.2	1/80th
Penalty per illegal move	-16	-1

Table 5.5 Reward Policy R1

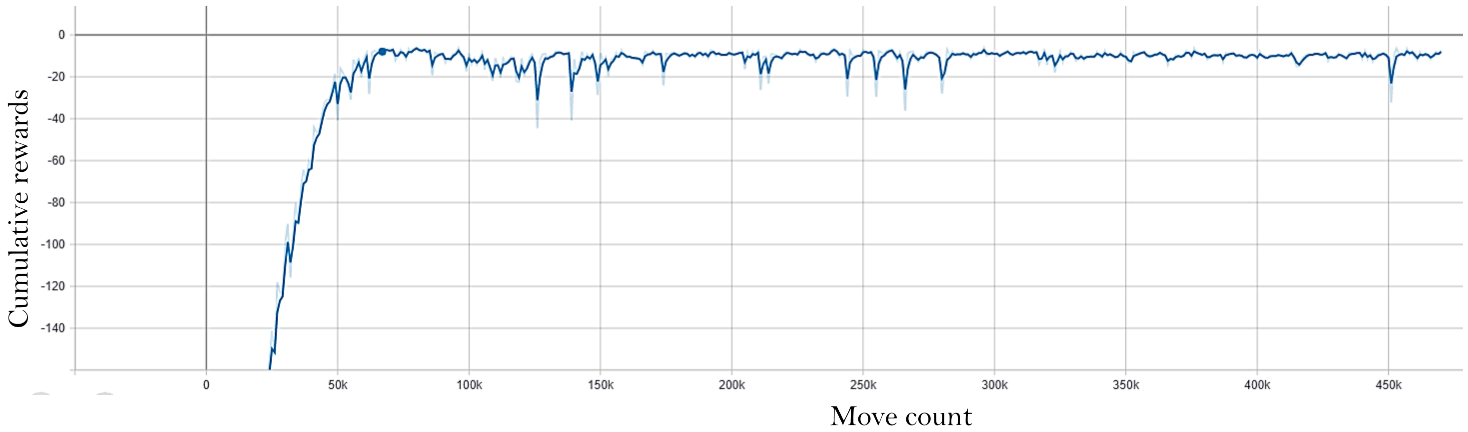
Results Experiment TS2AC2Exp1



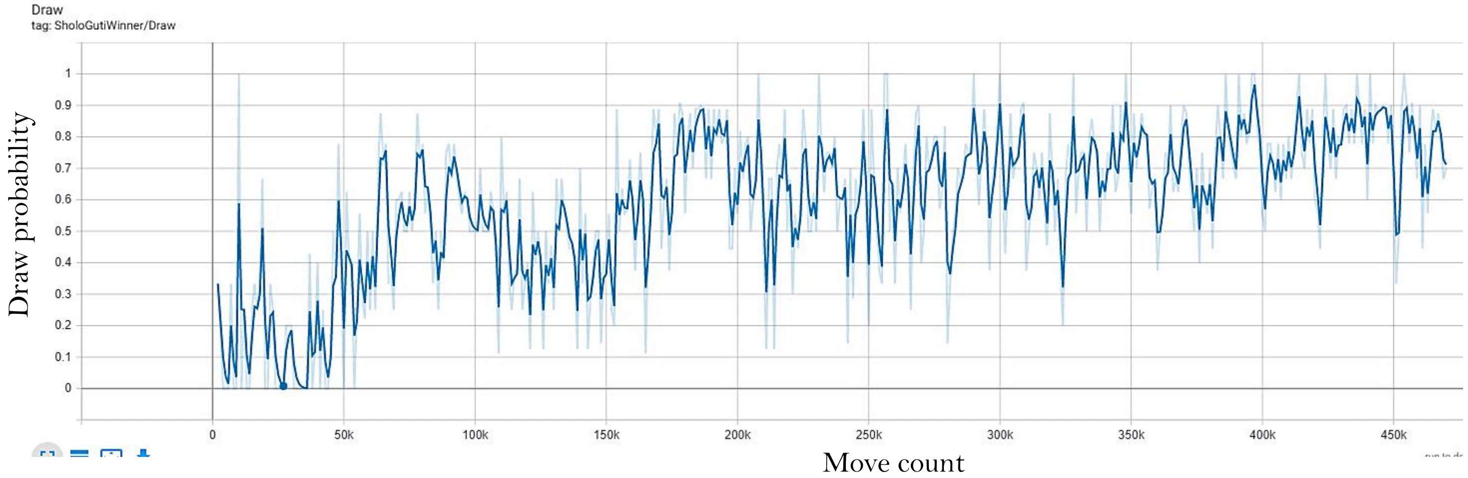


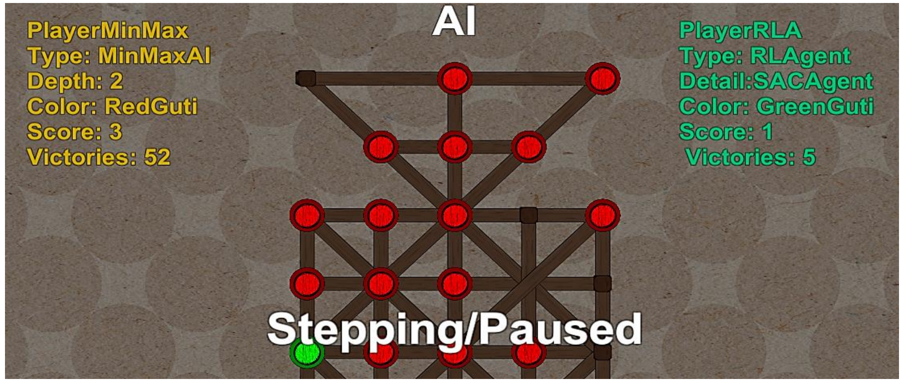
- High win rate against
  - MinMax with search depth 1
- SAC converges in fewer steps than PPO

Results Experiment TS2AC2Exp2



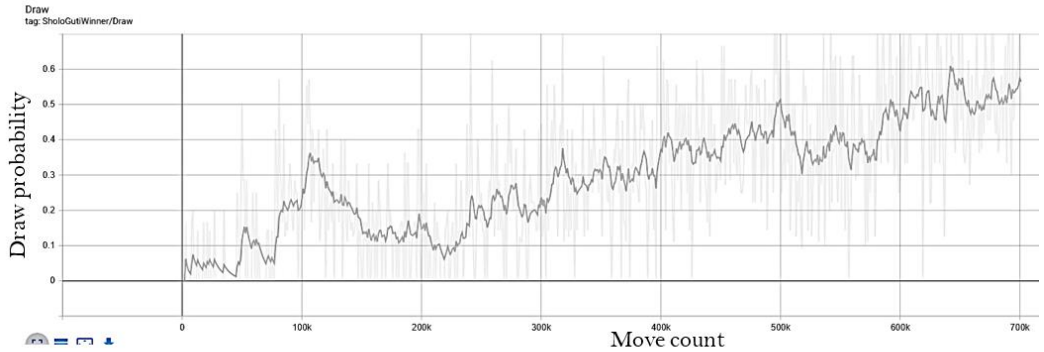
- Increasing draw rate with
- Increasing cumulative reward
- Poor performance against MinMax with search depth 2

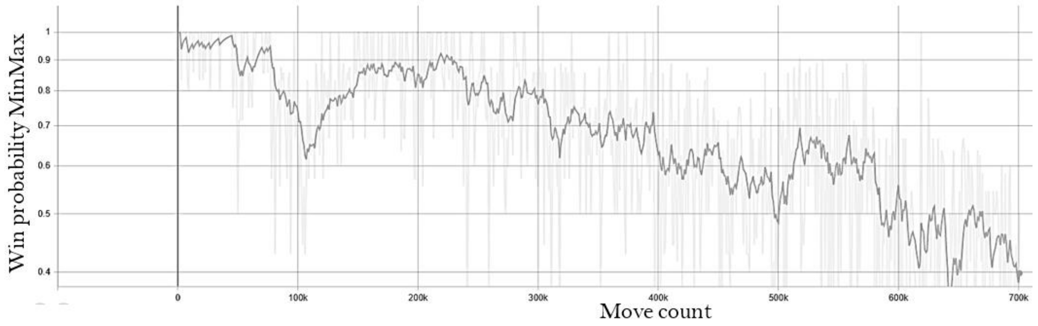


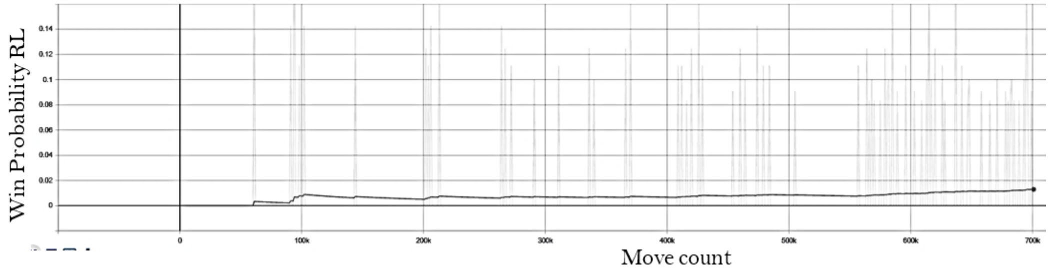


TS2 Conclusion

- Agents training against MinMax with search depth 2
  1. Produced very defensive agents
  2. The agents optimize for high draw probability instead of win probability.







Training Setup 3 (TS3)

Experiment name	NN architecture	**Training method**	**Reward structure**	Number of parallel learning environments
TS3AC2Exp1	AC-2	***Self-play***	***R2***	**4**
TS3AC2Exp2	AC-2	***Self-play***	***R2***	**8**

Table 5.3 Training Setup 3

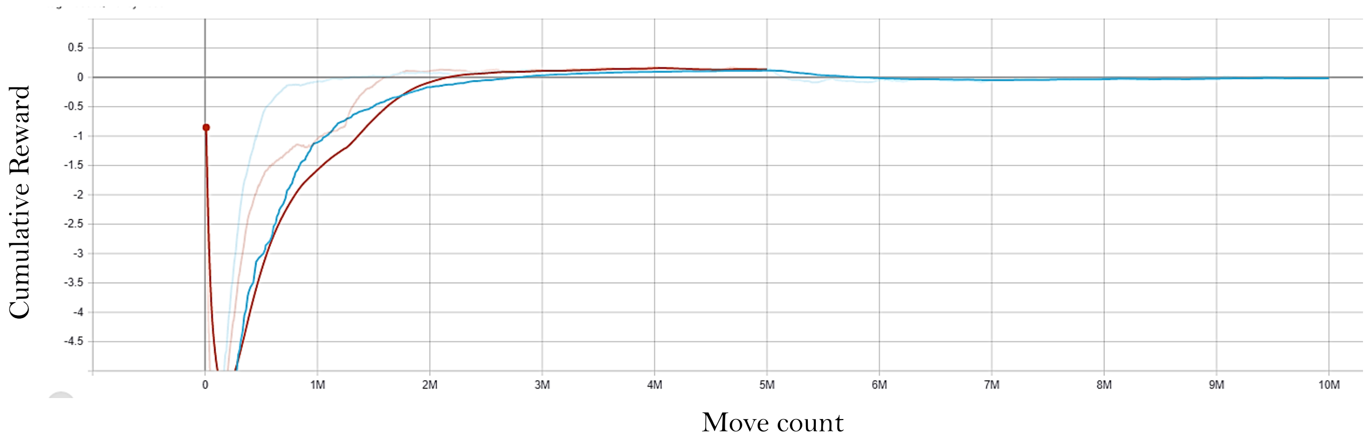
1. **Training setup 3 experiment 1 (TS3AC2Exp1):** Training agents using intermediate goal states with 4 parallel learning environments and self-play
2. **Training setup 3 experiment 2 (TS3AC2Exp1):** Training agents using intermediate goal states with 8 parallel learning environments and self-play

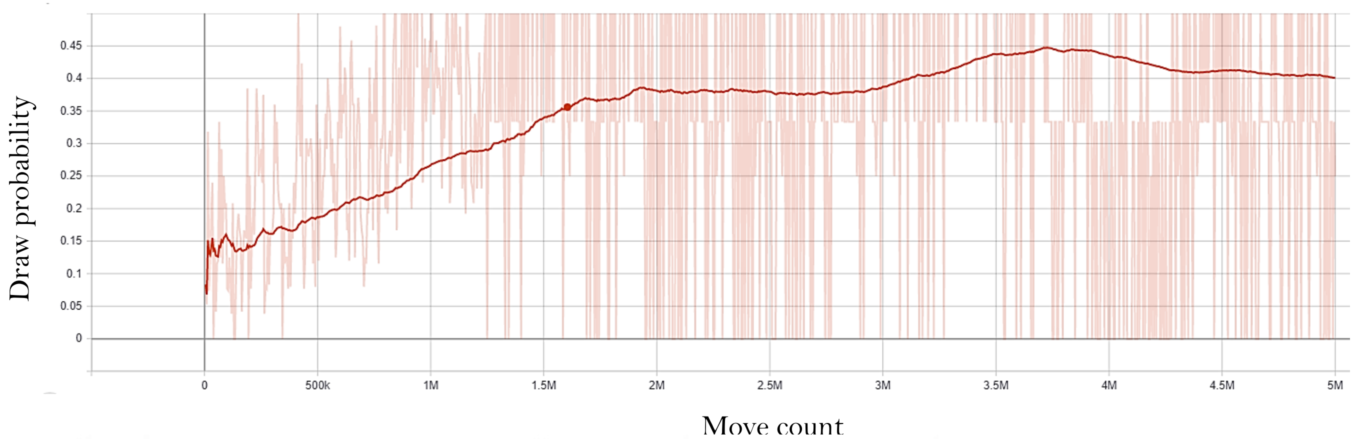
Reward Structure 2 Intermediate Goal States (R2)

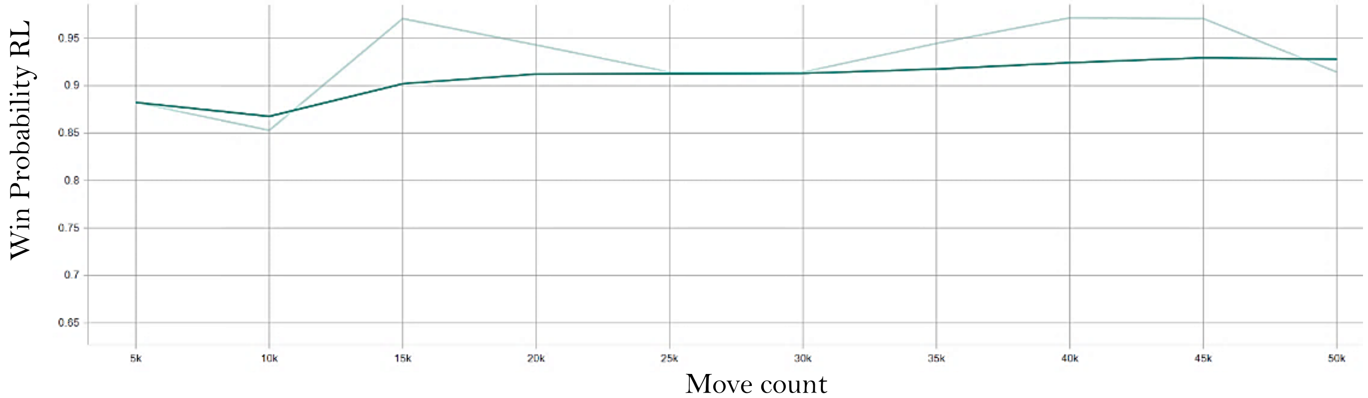
Reward / Penalty Condition	Reward	Ratio to maximum reward
Reward for wining a match after capturing 16 enemy guti	1	1
Reward for wining a match by stalling the game after reaching a higher score than enemy till move limit is reached.	0.125	1/8th
Reward per enemy guti captured from enemy	0.0625	$1/16^{th}$
Reward for reaching intermediate goal states	**0.0625**	**$1/16^{th}$**
Reward for drawing a match	0	0
Penalty for losing a match	-1	-1
Penalty per guti lost to enemy	-0.0625	$-1/16^{th}$
Penalty per legal move	-0.02	$-1/50^{th}$
Penalty per illegal move	-1	-1

Table 5.6 Reward Policy R2 with Intermediate goal states

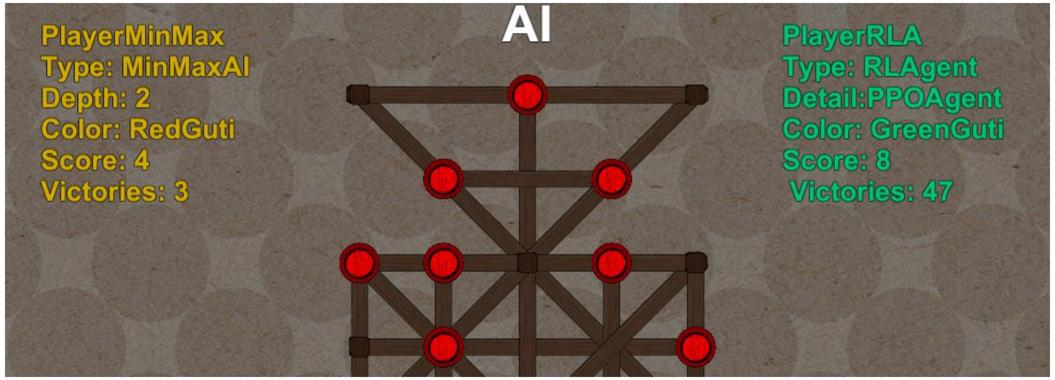
Results Experiment TS3AC2Exp1



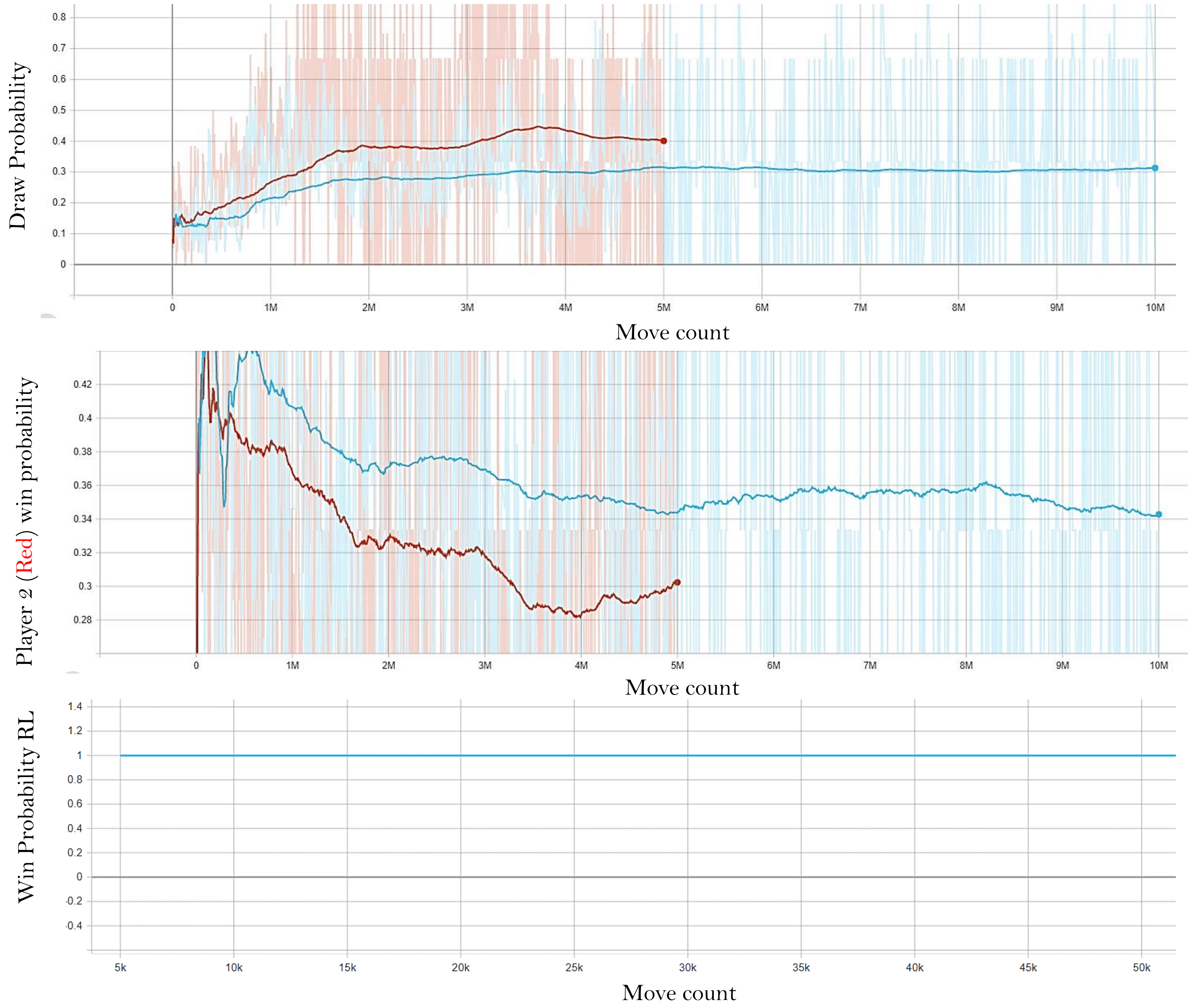




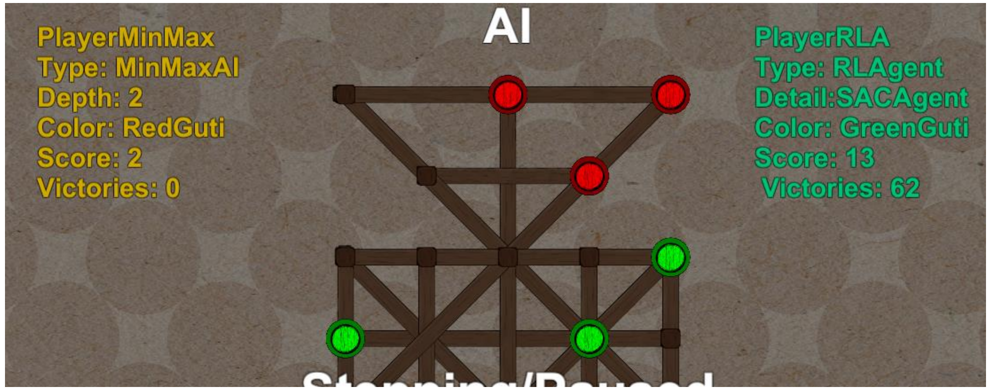
- Positive cumulative rewards
- Decreasing draw rate in self-play
- 90% win rate against MinMax search depth 2



Results Experiment TS3AC2Exp2



- Lower draw rates in self-play
- Higher win rates per agent in self-play
- 100% win rate against MinMax search depth 2



TS3 Conclusion

- Self-play is a viable training method when more that one learning environment is deployed parallelly
- RL algorithms train faster and better with increasing parallel learning environments
- Intermediate goal state reward system is effective in breaking RL agents out of local optima

Training Setup 4

Experiment	NN architecture	Training method	**Reward structure**	Number of parallel learning environments
TS4AC2Exp1	AC-2	Self-play	***R3***	4

Table 5.4 Training Setup 4

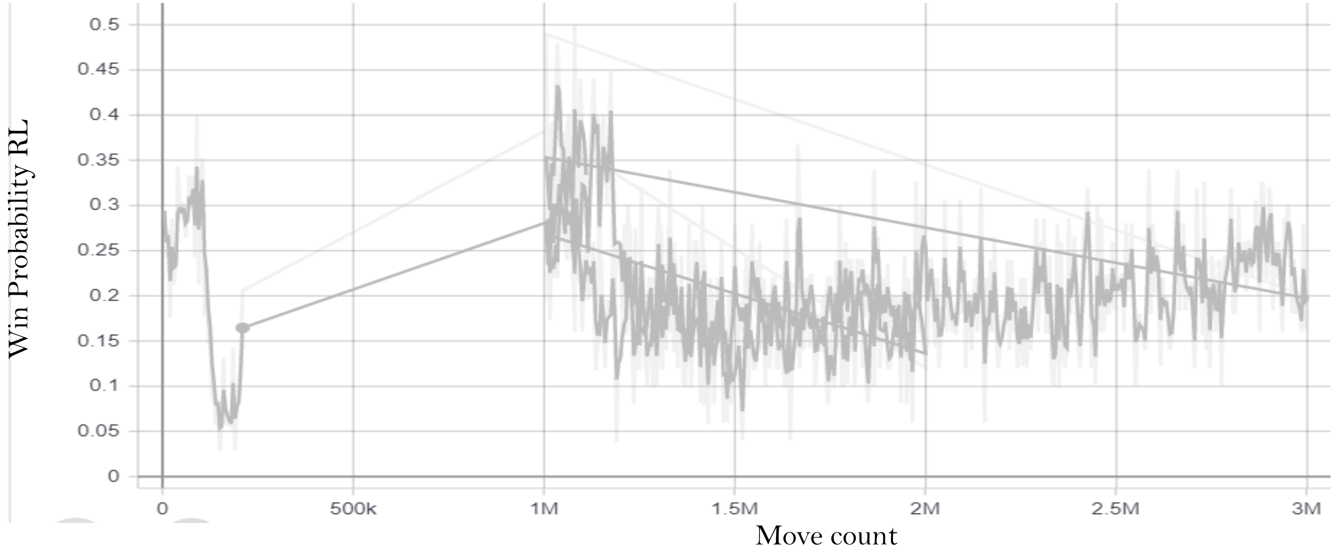
1. **Training setup 4 experiment 1 (TS1AC2Exp1):** Training agents with Curiosity rewards with 4 parallel workers and self-play

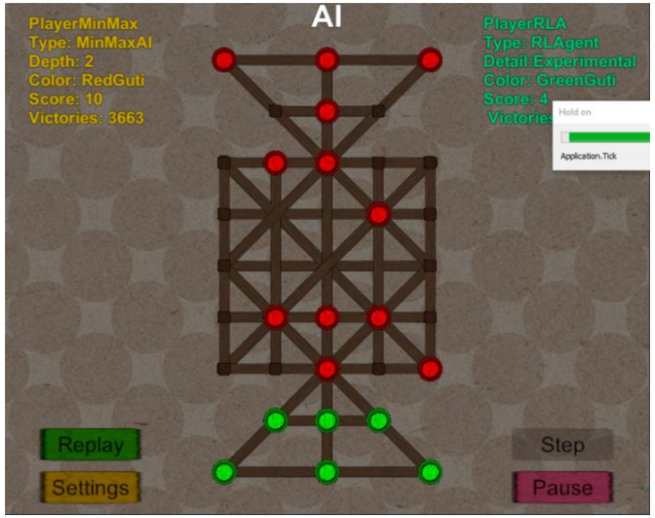
Reward Structure 3 Curiosity (R3)

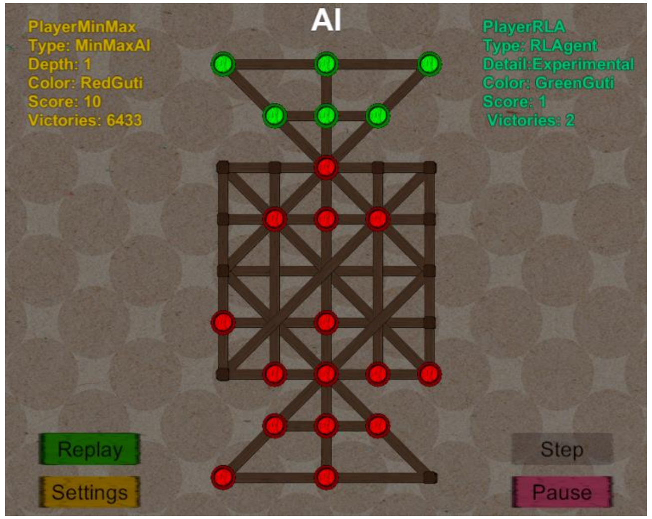
Reward / Penalty Condition	Reward	Ratio to maximum reward
Reward for wining a match after capturing 16 enemy guti	1	1
Reward for wining a match by stalling the game after reaching a higher score than enemy till move limit is reached.	0.125	1/8th
Reward per enemy guti captured from enemy	0.0625	1/16th
Reward generated by curiosity module	**Range (0 to 2)**	**0 to 2**
Reward for drawing a match	0	0
Penalty for losing a match	-1	-1
Penalty per guti lost to enemy	-0.0625	-1/16th
Penalty per legal move	-0.002	-1/50th
Penalty per illegal move	-1	-1

Table 5.7 Reward Policy R3 with Curiosity rewards

Results of Experiment TS4AC2Exp1





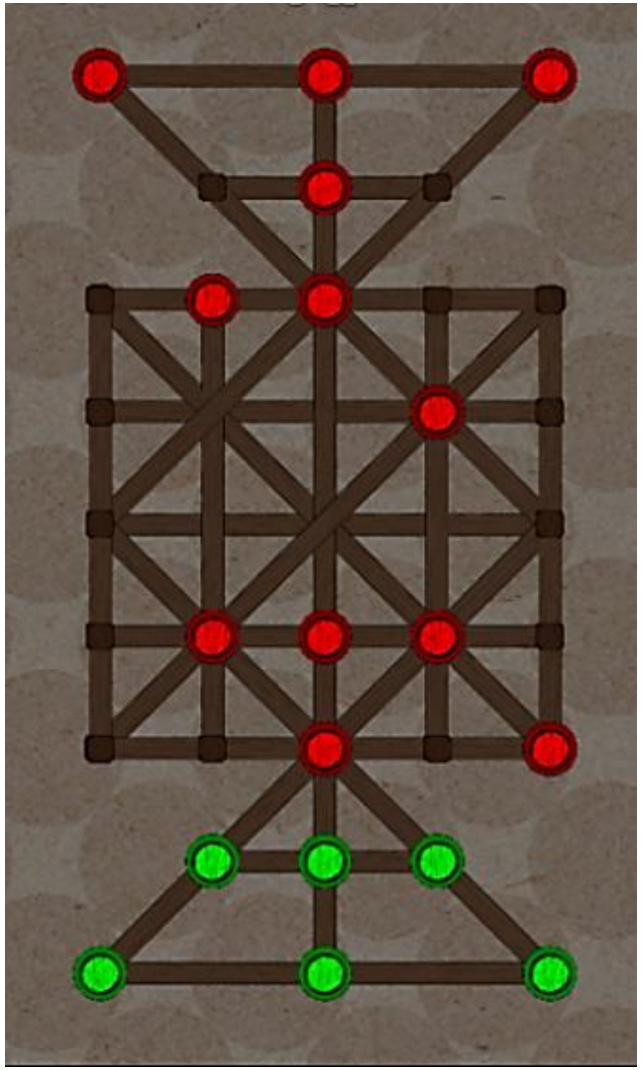


Checkmate

- Highly unstable
- Checkmate state discovered

TS4 Conclusion

- Curiosity reward generation system was highly unstable in the Shologuti environment, but it is very good at finding rare states
- It found a gap in the rules for Shologuti board game
  - It discovered a checkmate state that
  - The official rules do not account for checkmate states.



Conclusion

- Created a reinforcement learning testbench/environment for Shologuti board game
- Found an effective reward system using Intermediate Goal States
- Created a Shologuti game that runs on the Web and Windows
- Developed and trained RL Agents using state of the art RL algorithms SAC and PPO
- Created a python wrapper to access the Shologuti environment using external scripts

Future Work

- Add more games like Shologuti to the library
- Write a detailed technical documentation of the Unity project to enable future extensions.
- Streamline the installation of dependencies needed for running Shologuti environment
- Benchmark trained RL Agents against humans
- Implement and benchmark more state of the art algorithms like Alpha-Zero in the Shologuti environment
- Use GNNs to build custom RL algorithm
- Investigate and improve Intermediate Goal State reward system