Reinforcement Learning Based Artificial Intelligence to Play Ultimate Tic-Tac-Toe

Khoa Bui

Concordia University of Edmonton
Department of Mathematical and Physical Sciences

Introduction

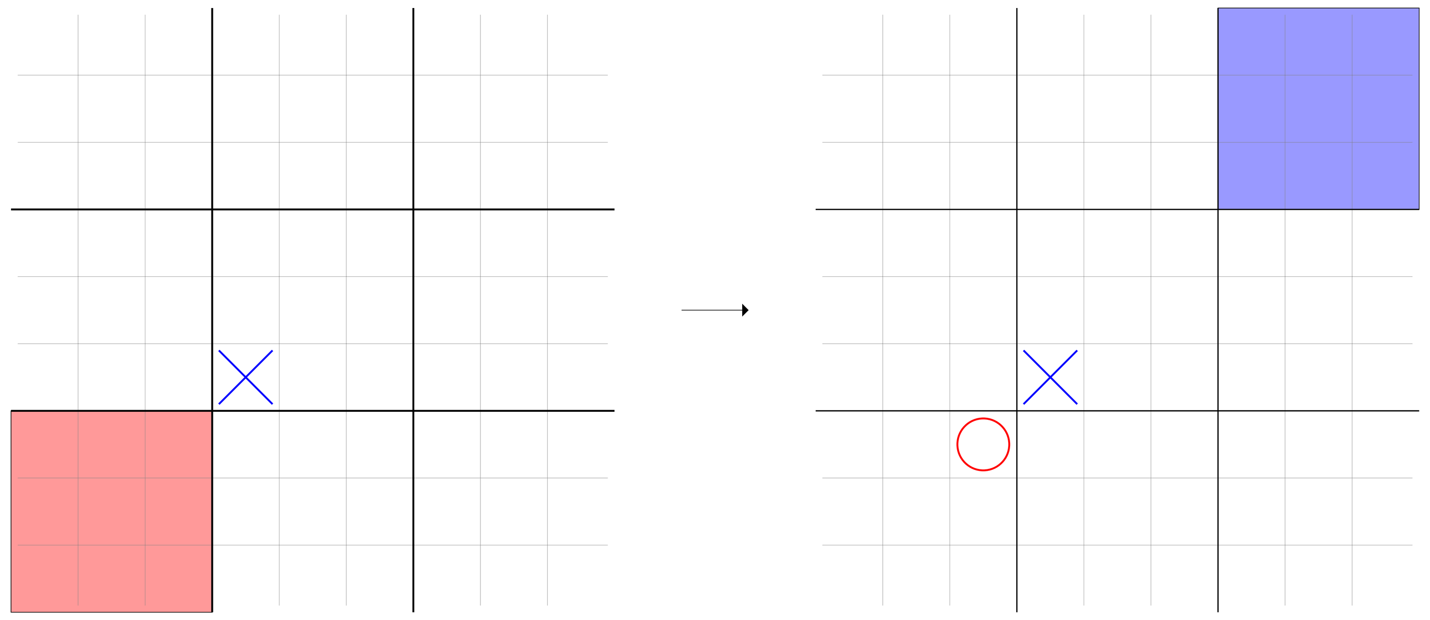
- In 2016, Google developed the program AlphaGo to beat the best human player in Go.
- Go was thought to be impossible for computers to play efficiently.
- In this project, I apply the methodology from AlphaGo to design an A.I. to play and explore stacking games such as Ultimate Tic-Tac-Toe [1].

Fig. 1: The Game Go. Source: Wikimedia Commons

- The game of Go has approximately 2.1×10^{170} legal moves, which makes it 10^{100} more complex than chess.

Ultimate Tic-Tac-Toe

- The game is play on a global 9×9 board that consists of 9 small boards of size 3×3. The coordinate of the previous move on a small board determines which small board the next move will be.



- If a player wins a small board, then they get that board for the global board.

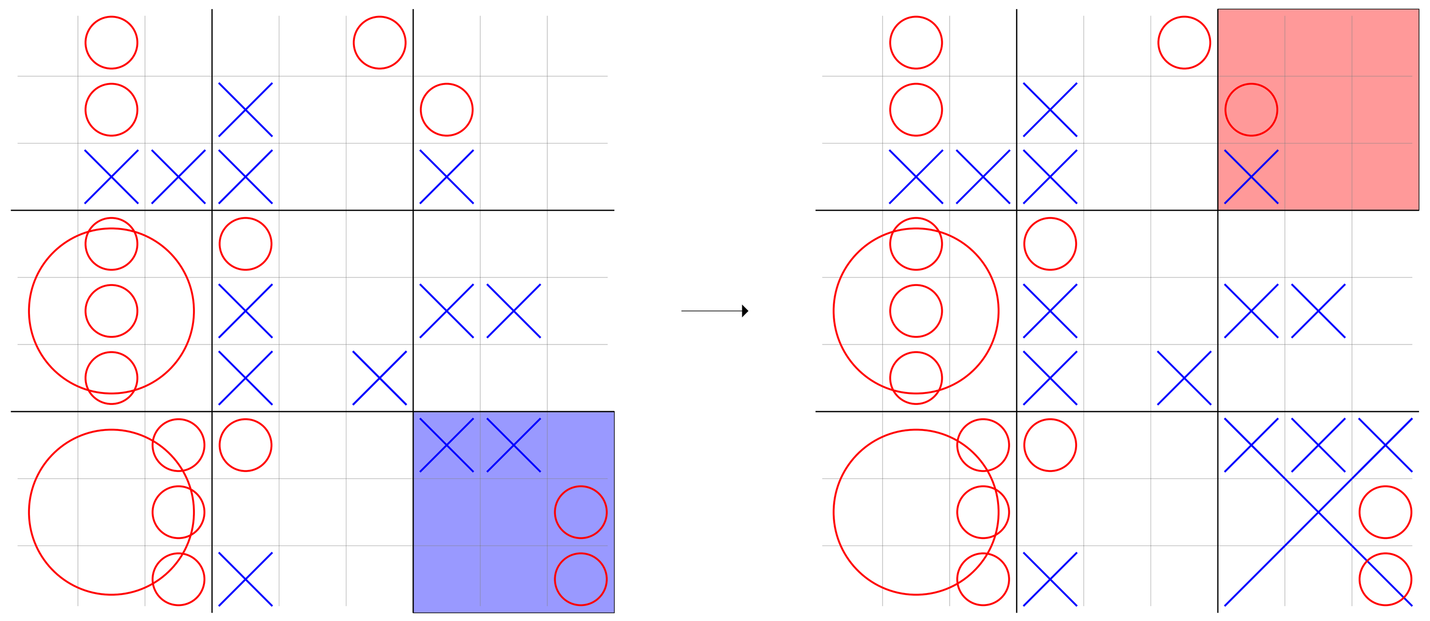


Fig. 3: Winning a Small Board and the Subsequent Move.

- The game ends when either a player places three consecutive small boards on the global board or there are no legal moves remaining in the small boards.
- Ultimate Tic-Tac-Toe is computationally interesting due to its higher complexity but still maintains the simplicity of Tic-Tac-Toe.

Reinforcement Learning

Reinforcement learning is the most intuitive method of learning associated with stimuli. There are 4 components in reinforcement learning:

- Agent: The learning system. Our A.I. player in this case.
- Environment: The system that the agent operates in.
- Reward: The value that the agent receives dependent on the action.
- Action policy: The set of actions that maximize the agent's award [3].

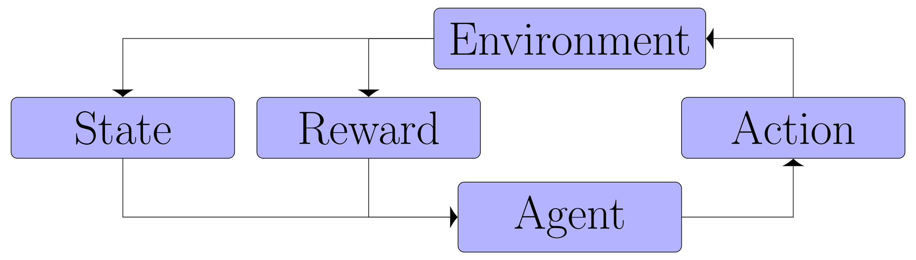
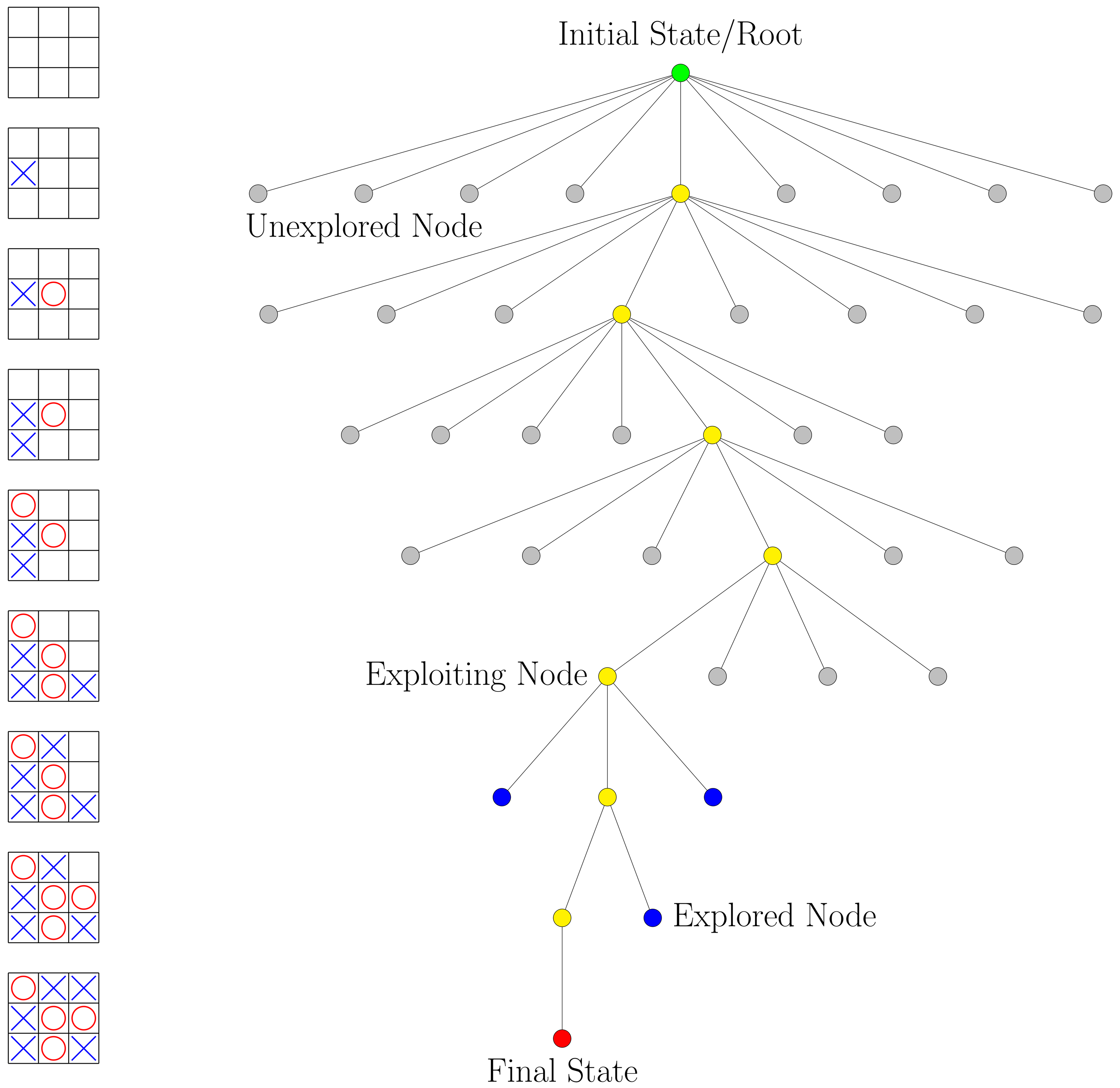


Fig. 4: Reinforcement Learning Visualized

Problems to Consider

- What to say that a move is the best move? Appropriate reward?
- How do we eliminate the bad moves?
- When does the computer know to stop calculating?
- How to balance between exploiting the best current move and exploring other possibilities?



Monte Carlo Tree Search

- Each simulation, the tree search treats a fully expanded node, or a visited node as a root and explores its children.
- The result of the simulation is propagate back to the root as **total simulation reward** $Q(v)$ and **total number of visits** $N(v)$ for a fully explored node.
- Upper Confidence Bound function to choose the next node among visited nodes to traverse through [2]:

$$\mathrm{UCT}(v_i, v) = \frac{Q(v_i)}{N(v_i)} + c\sqrt{\frac{\log_2(N(v))}{N(v_i)}}$$

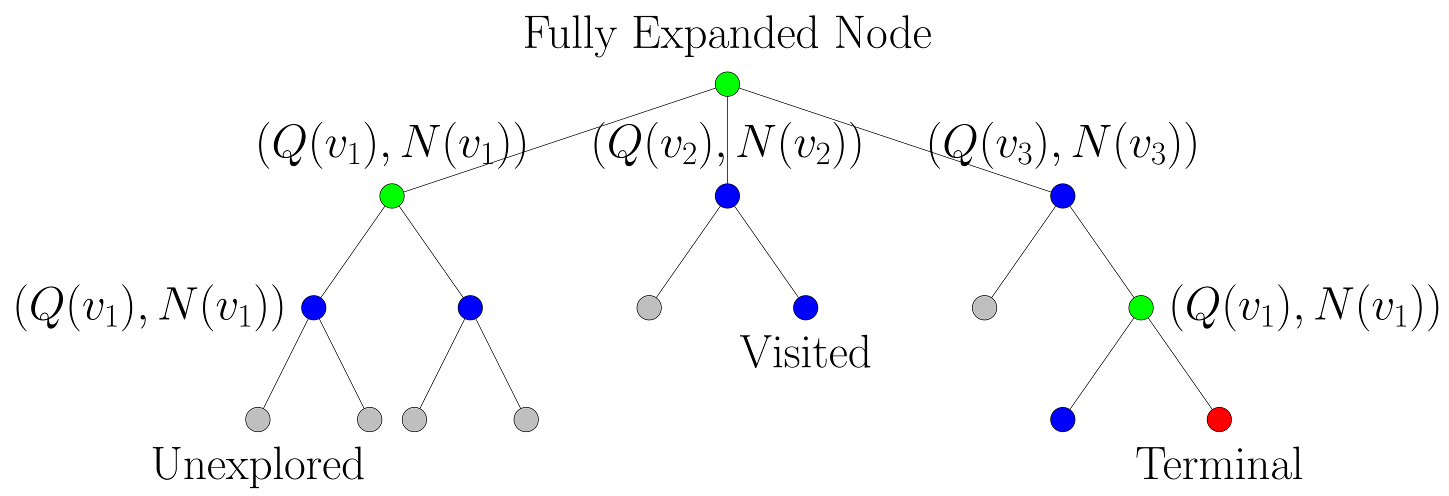


Fig. 6: Monte Carlo Tree Search on a Game Tree

Conclusion and Future Work

- There are other methods to implement such as Q-learning, ϵ-greedy algorithm, and the alpha-beta pruning method for reinforcement learning.
- There are other stacking games to apply the methodology to as well.
- Considering these methods to problems in neurology, finance, and logistic.

Acknowledgement

I would like to acknowledge Dr. Svenja Huntemann for allowing me to work on the stacking game project, and providing the background in game theory; Dr. Baidya Saha for supervising the machine learning aspect; Lexi Nash for working with me on this project; and other professors and peers in the Math & Information Technology department at the Concordia University of Edmonton. Finally, this project was supported by grant number CRG-SEED-2105-01.

References

[1] Elwyn R Berlekamp, John H Conway, and Richard K Guy. *Winning ways for your mathematical plays, volume 4*. AK Peters/CRC Press, 2004.

[2] Kamil Czarnogórski. "Monte Carlo Tree Search – beginners guide". In: (2021). Accessed: 2022-04-04. URL: https://int8.io/monte-carlo-tree-search-beginners-guide.

[3] Ivan Vasilev et al. *Python Deep Learning: Exploring deep learning techniques and neural network architectures with Pytorch, Keras, and TensorFlow*. Packt Publishing Ltd, 2019.