A Comparative Analysis of Agent Performance in Santorini

INFO 450 Final Project

Austin Young

1. **Introduction**

This project presents a comparative analysis of four types of agents' performance in the game Santorini. Santorini is a two-player, turn-based, pure strategy game where each player has two worker tokens placed on a 5x5 grid. Players take turns moving their workers and then adding levels onto buildings, both existing and new. The first player to move a worker onto the third level of a building wins the game. This simplistic primary objective is complicated by the conditions for moving a worker, namely it must be moved to an unoccupied spot, the worker can only move one level at a time, and the worker cannot move onto the third level of a building if a fourth level has been built on it. In optimal play, this results in a tradeoff between pursuing your own victory, keeping your options open, and blocking your opponent’s victory. This varied strategy in a stochastic, deterministic, and simple state space allows for a variety of agents to succeed, but agents that are best able to weigh these different dynamics are the most successful.

The goal of the project is to determine which agent performs the best in the game of Santorini by comparing their performance in a series of games. The agents under consideration are a random agent, a heuristic agent, a Q-learning agent, and a DQN agent.

1. **AI Concepts and Methods**

The project employs several AI concepts and methods in the creation and operation of the four types of agents.

* **Random Agent**: This agent selects one of the legal moves at random. It does not follow any specific strategy or learning method, and its moves are entirely stochastic.
* **Heuristic Agent**: This agent follows a heuristic approach. It will try to win if it has a move to do so or it will try to block the opponent from winning if their opponent’s next turn would otherwise result in a win. If no heuristic move is possible, the agent will revert to a random move.
* **Q-learning Agent**: This agent uses a Q-table for selecting optimal moves. Q-learning is a model-free reinforcement learning algorithm that seeks to find the best action to take given the current state.
* **DQN Agent**: This agent uses a Q-network and a target network to perform value iteration. Deep Q-Networks (DQN) is a reinforcement learning technique that combines Q-Learning with deep neural networks at its core.

1. **Code and Programming Challenges**

The project is implemented using three main files:

* `Board.py` defines the board game class by specifying the state spaces as a 2x5x5 tensor representing the position of workers in the 5x5 grid, and the level of buildings in the same 5x5 grid. It also defines methods for getting all legal moves at a given state, progressing the game given a selected action, and returning a reward at a given state.
* `Agents.py` defines the four agents: a random agent, a heuristic agent, a Q-learning agent, and a DQN agent. Each agent has a different method for selecting moves, as described in the AI Concepts and Methods section.
* `Arena.py` is used to compare the performance of these agents against each other. Each agent is put into a bracket where they play 10 back-to-back games with another agent, before being switched out for the next 10 games with another agent until all agents have faced each other 100 times.

The primary difficulty in this implementation was refactoring the state space as specified for the Q-learning and DQN agents. The board game class returns a 2d list of move tuples specifying how to move a worker piece as a python array. Correctly specifying the dimension modifications necessary to get this list of moves into a format recognizable for the NumPy Q-learning agent and Torch DQN agent proposed a significant challenge, that was largely overcome through trial-and-error debugging.

Additionally, figuring out how to balance and create a tournament of different agents in a functionalized and clever way required a fair bit of experimentation, as did determining how to set up the brackets to provide every agent a fair chance and prove its competency.

1. **Test Results**

**Fig.1 Agent Performance Heatmap
**

*Figure 1. Agent Performance Heatmap*

As previously mentioned, agents were paired up with each other over a series of hundred games. The wins of each agent were recorded in Figure 1. The average performance of each agent was as follows: Random win rate: 8.34%, Heuristic win rate: 34%, Q win rate: 63% DQN win rate: 94.67%.

1. **Analysis**

The Random agent performed the worst but won a fair share of games against the Q and DQN agents early on while they were still learning and improving. The Heuristic agent performed better, consistently beating the Random agent with a higher level of success than the Q-based agents. The reason for this performance is due to the Heuristic agent having a stochastic move selection process that did not require training to improve. This stochasticity in turn resulted in the Q-based agents learning to exploit the predictability of the Heuristic algorithm. Ultimately the DQN agent came out well in the lead with an almost 95%-win rate as it was able to explore with the Random agent, learn a basic formulation of strategy from the Heuristic agent, and compare developed tactics with the Q agent. The size and flexibility of the DQN network alongside the power of value iteration with the target network proved to be the best suited for learning this toy problem.

**Github Repository Link**: [450-Final-Project](https://github.com/rileyounga/Final-Project)