Deep Search:
Stochastic N-Player Planning
in Shared-Resource Environments

2025

Author
george.manolachi@s.unibuc.ro

Abstract

Recent advancements in Game AI have largely focused on two-player, zero-sum, deterministic environments (e.g. Chess, Go). However, multi-player environments with stochastic elements present distinct challenges in state-space complexity and opponent modeling. This paper presents **Deep Diver**, a comprehensive research testbed and AI agent designed for Deep Sea Adventure, a Japanese board game characterized by a shared resource constraint and stochastic movement. I propose a Monte Carlo Tree Search algorithm, modified to accommodate N-player dynamics and stochastic transitions via chance nodes.

1 Introduction

The field of Game AI has changed significantly over the past decade, driven by major breakthroughs in systems mainly designed for two-player, zero-sum games. Successes such as AlphaGo Zero have shown how powerful deep reinforcement learning and search algorithms can be when operating in deterministic, head-to-head environments.

This binary "win-loss" dynamic fails to capture the chaotic complexity of real world decision making. When an environment transitions from two players to $N > 2$ players and introduces stochastic elements, the adversarial model fractures. And agent must now not only determine how to win, but whom to oppose and when to align with others. This challenge is deepened further in environments with shared resource constraints, where a "Tragedy of the Commons" dynamic forces agents to balance individual greed against the colective survival of the group.

To investigate these complexities, this research utilizes the board game **Deep Sea Adventure** as a testbed, in which, unlike games with independent resources (e.g. Catan), it couples the utility of all players to a single, shared oxygen tank. A greedy move by one player depletes the time and resources available to all others, instantly altering the viability of every other player's strategy.

1.1 Contributions

- **Game environment:** Complete game implementation in C++, used as a research testbed and lightweight game engine simulating Deep Sea Adventure.
- **Stochastic MCTS:** WIP
- **N-Player Backpropagation:** WIP
- **Behavioral Benchmarking:** WIP

1.2 Summary of approach

My approach focuses on modifying the standard Monte Carlo Tree Search to accommodate the specifics of Deep Sea Adventure. Standard MCTS assumes a deterministic transition between states; I replace this with a stochastic model where actions lead to Chance Nodes, which do not represent a decision, but rather a probabilistic branching point (simulating the dice roll) that resolves into a final state. A simulation phase that mimics "greedy" human play-outs is implemented to provide more accurate heuristic evaluations at the leaf nodes, ensuring that the agent does not underestimate the aggression of its opponents.

1.3 Motivation

I chose this project because Deep Sea Adventure offers a unique intersection of game theory and probability that is currently under-explored in AI literature. While other "Push-Your-Luck" games exist, very few enforce a coupled constraint as strictly as this game does. The shared oxygen tank creates a semi-cooperative dependency that dissolves into cutthroat competition as resources become scarce. Solving this requires an agent that is not just calculating odds, but actively managing a shared economy of risk, a problem set with broad applications in multi-agent autonomous systems and resource management.

1.4 Related Work

- (Browne et al., 2012)
- (Cazenave and Jouandeau, 2010)
- (Sturtevant, 2008)
- (Schrittwieser et al., 2021)

2 Approach

2.1 Domain Description: The Mechanics of Deep Sea Adventure

To evaluate the efficacy of stochastic tree search, I utilized the Japanese board game *Deep Sea Adventure* as the research environment. Before detailing the AI architecture, it is necessary to understand the unique constraints this game imposes, which differ significantly from standard testbeds like Chess or Go.

The game is a multi-player "push-your-luck" race consisting of N players (2-6) and a linear path of treasure chips. The core mechanics are:

- **Shared Resource Constraint:** All players share a single oxygen supply, initialized at 25 units. The oxygen depletes at the start of every turn based on the total number of treasures a player is currently holding. This creates a "Tragedy of the Commons" dynamic where one player's greed penalizes the entire group.
- **Stochastic Movement:** Players move by rolling two 3-sided dice. The effective movement M is calculated as $M = \text{Roll}(2d3) - T_{held}$, where T_{held} is the number of treasures carried. This introduces a negative feedback loop: greedier players move slower and consume more oxygen.
- **The Goal:** The objective is to maximize collected treasure points (P) while returning to the submarine before the oxygen reaches zero. Failure to return results in $P = 0$.

2.2 Data Source & Exploratory Data Analysis

Unlike traditional supervised learning tasks relying on static datasets, this project utilizes Self-Play Data Generation.
The preliminary data source consists of observational data from manual human play and theoretical probability analysis of the game mechanics. Rather than training on a pre-existing dataset, I derived the constraints and heuristics from an analytical decomposition of the game rules.

2.2.1 Observation A: The Fairness of State Zero

Based on manual gameplay sessions, it was observed that the Order of Play (the starting sequence of players) appears to have negligible impact on the final win probability. Unlike Chess or Connect-4 (First-move advantage), the stochastic nature of the dice and the shared Oxygen pool balances the initiative. Consequently, the state space at $t = 0$ is assumed to be neutral, requiring no artificial handicaps for model training.

2.2.2 Observation B: The "Greed Limit" Probability

Through probability analysis of the movement mechanics, a critical threshold for carrying capacity was identified. Movement is determined by rolling $2d3$, resulting in an expected value of $E[move] = 4$. Since carrying treasure chips (b) subtracts directly from movement speed ($v = roll - b$), we derive a mathematical "Greed Limit":

- **Safe Load ($b \leq 2$):** The player retains a positive expected velocity.
- **Critical Load ($b \geq 3$):** The expected velocity drops to 1 or lower. Given the variance of $2d3$, carrying 3 treasures makes the probability of a "stall" **33%**.

This mathematical boundary will serve as a hard heuristic for the "Random" agent in our baseline comparisons, preventing it from making mathematically suicidal moves. From my tests so far, random agents almost always drown.

2.3 Models & Hardware Implementation

The core solver is a custom C++ implementation of Monte Carlo Tree Search.

2.3.1 The Model: Parallelized MCTS

- **Language:** C++20
- **Algorithm:** MCTS with UCT (Upper Confidence Bound for Trees). The implementation distinguishes between Decision Nodes (Player choices) and Chance Nodes (dice rolls).
- **Policy:** The simulation phase (rollout) will use the simple greedy heuristics derived in the EDA to terminate bad branches early.

2.3.2 Hardware Specifications

The implementation is tailored to use the **AMD Ryzen 9 9950x** CPU.
This CPU provides 16 cores and 32 threads. As MCTS is a highly parallelizable algorithm, I plan to utilize Root Parallelization, where multiple search trees are executed concurrently on separate threads, and their statistics are aggregated to select the final move. This approach also minimizes synchronisation overhead compared to Leaf Parallelisation.

2.4 Evaluation & Comparison Methods

To validate the effectiveness of the MCTS agent, I am considering the following metrics:

- **Baseline Dominance:** The MCTS agent will play 1000 games against a Random (Heuristic based) Agent. The targeted win rate is >95%
- **Search Efficiency:** Another benchmark will be done on the engine's performance by measuring Nodes Per Second, a standard practice for MCTS-based algorithms.
- **Heuristic Impact:** The Vanilla MCTS, using random rollouts, will be thoroughly compared to the Heuristic MCTS

3 Limitations

WIP

4 Conclusions and Future Work

WIP

References

Cameron B Browne, Edward Powley, Daniel Whitehouse, Simon M Lucas, Peter I Cowling, Philipp Rohlfshagen, Stephen Tavener, Diego Perez, Spyridon Samothrakis, and Simon Colton. 2012. A survey of monte carlo tree search methods. *IEEE Transactions on Computational Intelligence and AI in games*, 4(1):1–43.

Tristan Cazenave and Nicolas Jouandeau. 2010. Monte-carlo tree reductions for stochastic games. In *Proceedings of the 2010 conference on Communcations and Networking in China*.

Julian Schrittwieser, Thomas Hubert, Amol Mandhane, Mohammadmin Barekatain, Julian Schrittwieser, and David Silver. 2021. Planning in stochastic environments with a learned model. *International Conference on Learning Representations (ICLR)*.

Nathan R Sturtevant. 2008. An analysis of monte carlo tree search in multiplayer games. In *Computers and Games: 6th International Conference, CG 2008, Beijing, China, September 29-October 1, 2008. Proceedings 6*, pages 37–46. Springer.