Integrating Parallel Computation of Nash Equilibria into N-Player Role-Playing Games

1st Geela Margo Ramos
COP4520-22 Spring 0V01
University of Central Florida
Orlando, FL, USA
gelamargo.ramos@knights.ucf.edu

***Abstract*—When we consider its definition, a role-playing game (RPG) is a game in which participants assume the roles of characters in a fictional setting; its definition, however, can be extended by analysis of randomness or unpredictability in the game. In this project, we will delve into how we can define a role-playing game by stochastic randomness and how we can utilize Nash equilbira to assist action-choice selection by the program to act similarly to the role of a Dungeon Master (DM) in a simulation of Dungeons and Dragons (DnD). The implementation of parallel structures and algorithms to support calculation of Nash Equilibrium throughout each state of the game will be analyzed by observing how performative randomness conducted by the participant interacting with the program is affected, and how efficiency achieved by the program is affected when more than 2 players are introduced in the simulation.**

***Index Terms*—parallel computation, nash equilibria, stochastic games**

I. Introduction

The key elements of a role-playing game include story and setting; exploratio and quests; items and inventory; character actions and abiltiies; experience and levels; combat; and interface and graphics. These in mind, we can expect that a role-playing game ultimately exists in a dynamic-state affected by each involved player's choices. A game where "the play proceeds by steps from position to position" can be defined as a stochastic game; however, this definition normally applies when transition probabilities are controlled jointly by two players. If we were to scale the definition of game states throughout a role-playing game when more than two players are involved, it is important to understand the amount of randomness that can be experienced.

Randomness is contributed through the decisions that each player makes. Decisions affect the payoff other players receive and the environment interactions take place within. Likewise, randomness also occurs when a role-playing game implements rolling dice. By rolling dice, a certain level of uncertainty is produced for success of a player's choice of action in a scenario.

Because of the stochastic nature of role-playing games, consistent game-play where players make the most-optimal decisions at every state may is not normally achievable. The first objective of this research looks at how we can optimize game-play by a player through implementation of Nash equilbrium calculation. Following, we will explore how parallel structures and algorithms affect the effiency of overall gameplay, observing how performative randomness develops over time in-game and as more than two players are introduced to scenarios.

II. Background Information

A. Current Research in this Topic

Developments in this research normally looks at how to optimize the process of searching through all possible states of a game to find the most optimal towards your objective. This has led to the utilization of "game trees," especially for two-person sequential games like tic-tac-toe or chess. "Game trees" as a structure involve the root vertex as the unique starting position for the game, followed by the representation of legal moves or actions in game by edges and the representation of end states for the game through leaves. To realize a specific outcome in the game, a particular path must be followed where alternating edges represent moves between the two players; to achieve optimal gameplay for a specified player, the path taken through both players' decisions takes into consideration if their opponent takes counter actions that will best lead them towards their respective goal. Because of this, the utilization of game tree searching algorithms is necessary to find said-path.

Brute Force Algorithms through a Game Tree: Sequential algorithms were the first approach towards game tree searching. In terms of brute-force, these algorithms include MiniMax, NegaMax, and Negascout. Minimax observes in a game between two players that one acts as a maximizer, aiming to get the highest score possible, while the other acts as a minimizer, aiming to get the lowest score possible. Provided through this condition, it assumes that in a given state, values of the board are calculated by some heuristics unique to the type of game and that the path chosen by a player takes into consideration the most optimal play your opponent can make in the following state of the game. As a result of this strategy, two subroutines must be implemented where one branch exists for utility-maximizing and another exists for utility-minmizing.

Comparatively, Negamax implements the Minimax algorithm, but its form is variant due to its reliance on the zero-sum property of a two-player game; the zero-sum property can be

simply defined that if one player gains a certain amount, their opponent loses the same amount.

$$\max(a, b) = -\min(-a, -b)$$

It simplifies the implementation of the Minimax algorithm by "switching perspectives" and can employ alpha-beta pruning, a search algorithm that decreases the number of nodes the algorithm evaluates. Finally, there is Negascout, which relies on the Negamax algorithm framework. It utilizes an enhanced version of alpha-beta pruning to increase the number of nodes cut-off from evaluation. In general, research in this field has resulted in various improvements towards alpha-beta and forward pruning techniques, including iterative deepening, transposition tables, the killer move heuristic, and the history heuristic.

Parallel Algorithms through a Game Tree: Following improvements to brute-force algorithms for game tree searching, modern parallel computer architecture has given rise to new algorithms for optimal decision making. Specifically, considerable limitations associated with the size of search space for a game tree (which is determined through the number of strategies a player can choose and the number of steps it takes to complete a game) can be overcome. The first of these algorithms was ABDADA, which was a loosely-synchronized distributed search algorithm. To be employed, all processors were to start the search simultaneously through the game tree's root such that analysis of a position was conducted in three phases: the eldest son is analyzed; all other sons not being currently analyzed by other processors are analyzed; and if there are remain sons not yet analyzed, they are searched. As a result, ABDADA retains a recursive control structure over a sequential algorithm, and when implemented with a recursive NegaScout framework, it was observed to be thirty percent faster in performance, in contrast to a non-recursive search.

Continuously, other parallel algorithms that have been included for game tree searching include YWBC, Jamboree, and Dynamic Tree Splitting. The foundation behind each algorithm relies on using parallel processes to search different paths – whether through splitting the search ree into sub-trees or vising the first node and running the remaining siblings in parallel. A key issue to application of parallel computing in this topic is synchronization. The size of search space can be demanding, as previously mentioned, but synchronization is required to calculate optimization. As will be discussed in the following section, the choice of one's actions will affect another's, so paths may be dependent on one another to reach a particular conclusion; to optimize this dependency, we must consider the zero-sum payoff between all players.

Future Updates to this Section: This section will later include a table summarizing the differences between the different algorithms.

III. Discussion of Algorithm

Future Updates to this Section: This section will later include an explanation of Nash Equilibrium and how it is calculated. Will also include sample code for Nash equilibrium calculation

IV. Experimental Design

An important thing to consider is thhat we separate the definition of a player and a participant for experimentation. A participant is an individual direclty interacting with the program. A player is considered to be any character produced by the either individual or the program. Continuously, the following conditions will be met:

- All players in the simulation will have the same number of pure strategies; the program will be designed such that an arbitrary number of strategies is supported.
- Payoff values will be defined by integer values.
- Combinations of players and pure strategies are to be constrained by the maximum number of totally mixed equilibria supported by an n-k game, where n represents the number of players and k represents the number of strategies available to each player.

Future Updates to this Section: This section will include a discussion of the scenario design, and how it is implemented throughout the program.

V. Appendix

A. Challenges

One challenge to conducting this research is my computer. At the moment, my laptop holds an 8th Generation Intel Core i7 Processor, which supports 4 cores and 8 threads. This limits the scale of game states I can explore. In general, my computer already has a hard time running homework assignments and it is hard to check if the runtime has improved, as a result. The second challenge involves having suddenly switched topics a week before the midterm report was due in combination to now being the only person in the group; I have limited time to create a full simulation; however, with enough time, the goal is to create a Discord Bot that others can interact with to play the virtual simulation.

B. Set of Completed Tasks

Literature Review for Background Information: All notes for literature review, prior to the midterm report deadline, was originally kept in the following Google Doc Link; these notes include sample code to look at as references for calculating Nash equilibria and for implementation of different game tree searching algorithms, as well as possible game design for the actual scenario to be simulated: Notes through Google Doc

C. Set of Remaining Tasks for the Project

Tasks include to finish designing the scenario for simulation and writing code to simulate the scenario both sequentially and in parallel. Following I will run the code, alongside having other participants and myself interact with the program to analzye the efficiency and performative randomness generated through the scenario. Time-permitting, I will create a Discord

Bot to house this simulation and have it available for others to interact with.

REFERENCES

[1] LS Shapley, Stochastic games. Proc Natl Acad Sci USA 39, 1095–1100 (1953)

Future Updates to this Section: More references will be added as the paper continues to develop.