Implementation of Parallel Computation of Multiple Subgame Perfect Equilibrium in 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 efficiency achieved by the program is affected when traversing through a game tree during simulation. In future work, it is suggested to also observe how such implementation can affect performative randomness conducted by a player, and how efficiency achieved by the program is affected with an increasing number of players in the game.**

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

I. Introduction

The key elements of a role-playing game include story and setting; exploration 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 the time it takes to traverse across a game tree to calculate Nash equilibrium.

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 tree into sub-trees or visiting 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 experienced payoff between all players.

III. Discussion of Algorithm

To compare and calculate payoff between all interacting players within a non-cooperative game, one can use Nash equilibrium to determine the optimal solution. A non-cooperative game refers to a game where rational economic agents utilize available strategies to achieve their own, individual goals. There is a lack of external authority to enforce cooperative behavior; and when Nash equilibrium is employed, as a result of the game's nature, no player gains anything from deviating from their initially chosen strategy. It is important to note that there is no formal algorithm to calculate for normal Nash equilibrium. Looking for Nash equilibrium usually involves making comparisons between each player's available actions to find a compromise between tradeoffs. The process behind finding Nash equilibrium intensifies when situations where tradeoff is variable and subject to elements of randomness in the game are introduced.

For example, there are times when multiple strategies may be available to a player, but a player will not be able to increase their expected payoff by choosing between the given actions. In other words, there is no "pure" strategy that is available and leads to performative randomness by the player. A player's next move can be unpredictable. When we find ourselves in this situation, we instead look for what is known as mixed strategy Nash equilibrium. Between two agents and two strategies, we can form a 2x2 matrix which provides a cell for each combination of strategies taken by both players. Resulting in four possible cells, we then calculate the overall probability of each combination (each cell) to be executed by multiplying the probability of a player's corresponding strategy to the probability of an opponent's corresponding strategy. Finally, after choosing which player whose payoff you would like to calculate, we would then multiply each combined probability with the player's tradeoff in the respective cells. These numbers are summed together, and what results is the expected payoff in the mixed strategy Nash equilibrium for that player.

Another variation, and what is also known as a refinement, of Nash equilibrium applicable to role-playing games is multiple subgame perfect equilibrium. Role-playing games often take up the structure of rounds and involve sequential behavior (taking turns) between a given number of players. The shifting states of a role-playing game, from a player's health to how close they may be to their target destination, may cause each round to have a different mixed strategy Nash equilibrium. We can categorize each round, then, as its own subgame, and when there are multiple rounds, calculating for the overall tradeoff is what is known as multiple subgame perfect equilibrium.

We can map behavior and interaction in extensive form games like role-playing games through game trees. As we will explore in the experimental design of this research, the more strategies each player has available to them, especially when in response to an opponent's strategies, the larger the depth will become of the game tree. The number of interacting players will also affect the depth, so long as they behave

sequentially - taking turns to take action, through elements like rolling a dice for initiative (for which determines turn order for all players). When deciding the optimal action of a player through calculation of Nash equilibrium, investigating how a binary tree can be used to store the tradeoffs was key to the design of this algorithm. Thus, implementation of parallel computation for multiple subgame perfect equilibrium was explored through creating a concurrent binary tree. Nodes were assigned separate threads to conduct calculation for mixed-strategy perfect equilibria. In terms of traversal of the tree, nodes were called in post order, since calculations were dependent on children nodes' tradeoff values.

Two methods were attempted to create a coarse-grained concurrent binary tree, the first of which was applying a Semaphore lock to the overall tree. A Semaphore is a generalized mutex, and is normally used as a signaling mechanism. Each thread, upon accessing and calculating the individual tradeoff for a node, would attempt to acquire the Semaphore lock. This method was chosen in consideration to how traversal through the tree occurred. The second method attempted was applying the exponential backoff algorithm to individual nodes as each thread conducted calculation of a specified nodes' tradeoff. If a thread responsible for calculating a child node's tradeoff had yet to be completed, the corresponding parent node's tradeoff would have to wait a randomized period of time before attempting to access the child node.

IV. Experimental Design

An important thing to consider is that we separate the definition of a player and a participant for experimentation. A participant is an individual direcltly 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.

In the provided simulation, a participant is spawned onto a location within an 8x8 map. The objective of the simulation is to find a target location by traversing through the provided map, while an enemy player also seeks for you. One wins the game by either finding the target location or defeating the enemy player. To evaluate the proposed algorithms, the approach to calculate mixed subgame perfect equilibria was compared to iterative and recursive methods for the same calculations. Overall, both methods for parallel implementation failed to succeed the efficiency of both the iterative and recursive approaches.

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 and was unable to complete the Minimax implementation for game play. Despite ths, NashHunt.java is available as a sample to the type of games algorithms like these could be utilized in.

References

[1] Borovska, P.,& Lazarova, M. (n.d.). Efficiency of Parallel Minimax Algorithm for Game Tree Search.

[2] Gao, Y., & Marsland, T. A. (n.d.). Multithread Pruned Tree Search in Distributed Systems.

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

[4] Spaniel, W. (n.d.). Game theory 101. Game Theory 101. Retrieved from http://gametheory101.com/

[5] Widger, J., & Grosu, D. (n.d.). Parallel Computation of Nash Equilibria in N-Player Games.