# Task 1: Artificial Intelligence Strategies in Chess: A Comparative Analysis

## Introduction

Artificial Intelligence has brought revolutionary changes to chess, transforming the way analysis, playing, and perception are made. These changes witnessed in AI, from early rule-based systems to advanced self-learning algorithms, reflect the wider evolution of AI in problem-solving and decision making. The report compares two main AI strategies used in chess: the classical Minimax algorithm with Alpha-Beta Pruning, representative of symbolic AI, and the more modern data-driven approach, DeepMind's AlphaZero, powered by deep neural networks combined with reinforcement learning.

## 1. Symbolic AI: Minimax Algorithm with Alpha-Beta Pruning

**Theory and Concepts**

Minimax algorithm constitutes the basis of how AI plays two-player games like chess. It does this by trying to evaluate an optimal strategy based on predictions of opponent moves. In the Minimax framework, each move branches out into possible game states, creating a "game tree." It tries to minimize the maximum gain of the opponent-a process known as the "minimax principle." Here is how it works in brief:

* **Minimax Principle:** The players believe that their opponents will make the best possible moves, and hence every move is evaluated by considering the worst case.
* **Pruned Alpha-Beta Algorithm:** Efficiency concerns prune the algorithm using Alpha-Beta. This pruned version prunes those branches of the game tree that do not have any influence on the final decision. It therefore drastically reduces the number of positions to be evaluated with no loss of information.

Alpha-Beta Pruning enhances computational efficiency by enabling an algorithm to perform a deeper search for any given amount of time into the game. This efficiency becomes crucial during chess games, where the branching factor remains huge at each position.

**Relation to AI:**

Minimax algorithm with Alpha-Beta Pruning is representative of symbolic AI, wherein intelligence is built upon explicit rules and structured problem-solving techniques. In such an approach, the course which the AI will follow is bluntly logical to evaluate game states in order for it to make decisions. Being part of this "top-down" method of artificial intelligence, it depends upon heuristics and rules created by humans that model intelligence through logical deductions rather than experiential learning, showcasing an early era in AI.

This represents the "knowledge-based" paradigm in that the intelligence of the system emanates from structured information crafted by human experts. The Minimax approach is powerful but, in principle, cannot go beyond the scope and complexity of the rules being followed, hence limiting its adaptability to new or unforeseen patterns.

## 2. Machine Learning: Deep Neuro- and AlphaZero

**Theory and Concepts**

AlphaZero is considered a revolutionary AI discovery for chess and other strategy-type games developed by DeepMind. Unlike symbolic AI, AlphaZero bases its power on deep neural networks and reinforcement learning, allowing this program to "learn" the game from experience rather than just follow rules it has been provided with. The main components of AlphaZero are:

* **Self-play and Reinforcement Learning:** AlphaZero learns to play by playing against itself millions of times. It refines its strategies through the process of reinforcement learning, whereby moves that lead towards a win are kept and reinforce the system, while less effective ones are discarded.
* **Deep Neural Networks:** These networks allow for the valuation of board positions by AlphaZero with uncanny accuracy. This enables the neural network to learn strengths and weaknesses of positions in complex states that other traditional algorithms might not be able to recognize.
* **Monte Carlo Tree Search:** MCTS works with AlphaZero's neural network to evaluate what future moves may look like. This is different from Alpha-Beta Pruning, which simply cuts off some branches, whereas MCTS looks into the possible moves probabilistically to help guide the neural network towards good strategies.

Deep learning at AlphaZero enables him to work with chess based on the "tabula rasa" principle, learning autonomously the strategies without heuristics provided by humans or historical game data. This development is representative of a paradigm shift in AI and really serves to underline how powerful self-improving systems can be.

**Relation to AI**

AlphaZero demonstrates one currently very valid trend in AI development-namely, that of a shift from symbolic to data-driven AI. In this instance, intelligence does not need to be hand-coded but instead arises from patterns emergent via large volumes of gameplay data. This model epitomizes the revolutionary potential of machine learning and self-learning systems, whereby AI can be independently developed to have expertise in complex domains. In particular, the success of AlphaZero can be claimed as a permissible level of mastery through experiential learning, broadly evolving from rule-based systems into an adaptive model of pattern recognition.

This could have a bigger implication for the application of AI in areas other than chess, as it would be able to be customized to solve problems that involve strategy and decision-making throughout various sectors in health, finance, and logistics. It was a success which reflected how AI is boundless in performing complex tasks with all autonomy excluded from human-contrived rules, challenging our previous perception of learning and decision making in machines.

## 3. Comparative Analysis: Key Differences and Implications

Efficiency vs Adaptability

* **Minimax with Alpha-Beta Pruning:** Efficiency for structured decision-making processes bound by the depth and complexity of pre-defined rules is in its interest.
* **AlphaZero:** It offers adaptability, which solves problems using experiential learning with no domain-specific knowledge. Although computationally very intensive, adaptability provides for continuing development in the quest to understand chess strategy.

Symbolic Reasoning vs. Pattern Recognition

* **Minimax Approach:** Symbolic and logic-based, the knowledge it uses is pre-defined by humans.
* **AlphaZero's Neural Networks:** Pattern-based and data-driven, find and refine strategy through pattern recognition rather than logic.

## Implications for the Role of AI in Chess

Moving from symbolic AI (Minimax) to machine learning (AlphaZero) opens a whole new perspective for chess AI. The ability of AlphaZero to "self-learn" chess is not only a powerful example of machine learning but also transforms the role of AI from an executor of pre-defined strategies into a dynamic strategist with potential for innovation. The evolution described here underlines the broader shift in the position of AI as a full-fledged autonomous problem solver in complex and strategic environments.

## Conclusion

The evolution from Minimax with Alpha-Beta Pruning to the deep learning model in AlphaZero reveals an ever-expanding scale of AI capabilities under chess, from simple logical inference performance to the discovery of new strategies. The Minimax provided a framework through structured decisions, whereas AlphaZero elevated AI in chess to unparalleled feats of skill, truly demonstrating what an adaptive, self-learning environment is capable of. This transformation represents the wider evolution of AI from rule-based reasoning to experiential learning-a quantum leap that has redefined both its capabilities and its implications across domains.

# Task 2(a) Report: Determining a Route from Tipperary to Sligo

## Introduction

This report analyzes three pathfinding algorithms—Breadth-First Search (BFS), Dijkstra’s Algorithm, and A\* Algorithm—used to find the most efficient route between two locations on a weighted graph of cities in Ireland. The graph represents cities as nodes and distances between them as edges, with Tipperary and Sligo as the start and goal nodes.

Graph and Heuristics Setup

The code begins by importing necessary libraries (time, PriorityQueue, and deque) to handle timing, queuing, and graph traversal. The graph data structure is defined as a dictionary where each city node connects to other cities with specific distances as weights. Additionally, a heuristic dictionary is provided for the A\* algorithm, representing straight-line distances from each city to Sligo.

## Algorithm Implementations

1. **Breadth-First Search (BFS)**
   * **Purpose**: BFS explores the graph layer-by-layer to find the shortest path based on node count rather than edge weight.
   * **Implementation**: The bfs function initializes a queue to manage the traversal order and maintains a visited set to avoid revisiting nodes. The function iterates through the queue, checking each city’s neighbors until it reaches Sligo, recording both the path and execution time.
   * **Output**: Returns the path taken and the time spent to complete the search.
2. **Dijkstra's Algorithm**
   * **Purpose**: Dijkstra’s algorithm finds the shortest path by considering the edge weights, making it effective for weighted graphs.
   * **Implementation**: Using a priority queue, the dijkstra function prioritizes paths with the lowest cumulative distance. It updates the shortest path to each node as it explores neighbors, ensuring the most efficient path is found by the time it reaches Sligo.
   * **Output**: The function returns the shortest weighted path and its execution time, allowing comparison with BFS and A\*.
3. **A\* Algorithm**
   * **Purpose**: The A\* algorithm combines Dijkstra’s approach with heuristics for faster convergence to the goal.
   * **Implementation**: In the astar function, a priority queue is initialized with a starting f-cost (combination of actual and heuristic cost). It uses the heuristic values for each city to predict distance from the goal, which optimizes the search direction.
   * **Output**: Returns the calculated shortest path and time taken, providing a benchmark for evaluating efficiency alongside the other algorithms.

## Comparative Analysis

Each algorithm provides a unique approach to pathfinding:

* **BFS**: Simple but effective for unweighted pathfinding; may be slower due to lack of edge weight consideration.
* **Dijkstra**: Optimal for weighted graphs; however, it can be slower than A\* due to its exhaustive search.
* **A\***: Combines heuristics with weighted pathfinding, offering the most efficient route in this case.

## Conclusion

This comparison of BFS, Dijkstra's, and A\* algorithms highlights notable differences in time efficiency for pathfinding in a weighted graph of Irish cities. *A\* Algorithm* demonstrated the highest time efficiency by using a heuristic to focus on the most promising paths, making it ideal for complex routing tasks. Dijkstra's Algorithm, while slightly slower, effectively balanced optimal pathfinding with computation time by incorporating edge weights directly. BFS, lacking weight consideration, was the least time-efficient due to its exhaustive search method.

In summary, A\* is the recommended choice for applications requiring both speed and accuracy in weighted graphs, while Dijkstra’s remains a strong alternative for general shortest-path problems. BFS is best suited for simpler or unweighted structures, where edge weights do not impact the pathfinding outcome.

# Task 2(b) Artificial Intelligence in Video Games: Finite State Machines (FSMs) and Goal-Oriented Action Planning (GOAP)

## Introduction

Since video games were invented, from the most basic to very advanced control systems of NPCs, artificial intelligence has played an important role. Simulating complex behaviors with AI allows for much more realistic interaction with characters—a key feature of how AI contributes to player engagement. This report outlines two common AI methods now used in recent video games: the Finite State Machine and Goal-Oriented Action Planning, detailing their algorithmic methods and specific uses in popular games.

## Finite State Machines (FSMs)

**Overview of Algorithm**  
In video game development, the use of a finite state machine acts as the basic framework in AI. How FSM works: It relies on the predefined number of states an NPC cycles through according to some external stimulus or the action of the player. Each state corresponds to one specific behavior of the NPC—or NPCs—such as idle, patrol, attack, or flee behavior, with transitions specified by predetermined conditions. This is a very simple, effective model for structured and rule-based behaviors that adapt the response of NPCs to specific contexts in the game.

**Example Implementation: The Last of Us Part II**  
FSMs are an important part in building the behaviors of enemies in *The Last of Us Part II*, mostly regarding stealth and combat. Examples of such states include patrolling, searching, and attacking. These states change according to the player's movements and the way they interact with the environment. This FSM-based system acts actively in response to enemies, adding a new level of realism in the game because gameplay will be more excitingly challenging and adaptive for players.  
In this game, the used FSM shows how transitions between states provide not only interactive games but also factions of subtle behavior by the NPCs that can come as if from a human, making the game more immersive.

## Goal-Oriented Action Planning (GOAP)

**Overview of Algorithm**  
Unlike FSMs, GOAP proposes a much more flexible and adaptive architecture for AI: GOAP allows NPCs to set certain goals and then dynamically chooses and constructs sequences of actions to be executed in order to handle those goals. This will grant NPCs to react smoothly and organically against the players' actions and in-game variables. Because GOAP prioritizes objectives, rather than preconceived state transitions, it allows for more emergent behaviors that come off as goal-oriented—an illusion of autonomous decision-making, if you will.

**Sample Implementation: Middle-earth: Shadow of Mordor**  
This is notably realized in the GOAP architecture in the 2014 Nemesis System of *Middle-earth: Shadow of Mordor*, governing the behaviors of the Uruk captains. In this system, NPCs can adaptively set goals, such as seeking revenge or growing stronger, in answer to the interactions of the player. These dynamic goals spur the NPCs into the execution of sequences of actions that feel unique for each encounter. A good example of how GOAP-driven characters would create diverse personalized experiences through changes in NPC behaviors concerning the player's interaction is considering the Nemesis System. It promises unpredictable, immersive gameplay.

**Comparative Analysis of FSMs and GOAP in Video Game AI**

FSMs and GOAP are two vastly differing philosophies when it comes to video game AI, each with their respective strengths in developing games. FSMs are particularly useful for simpler, rule-based behaviors where, upon certain conditions, state transitions may take place. They are great for games that need to deploy predictable, easy-to-handle NPC behaviors. Where GOAP does have the advantage is by not only providing more advanced capabilities of its systems but goes to model and support state-driven actions towards a goal. This means that NPCs will be able to change internally what they are attempting to do at any time to react to emergent behavior from players in complex situations. This flexibility is especially useful in open-world games or scenarios requiring personalized interactions, as evidenced in the Nemesis System of *Middle-earth: Shadow of Mordor*.

## Conclusion

AI techniques like FSMs and GOAP form the core of behavioral specification and interactive specifications of NPCs in games, therefore providing the quintessential experience. FSMs present a structured state-driven approach, suited to linear and tactical situations, evidenced in *The Last of Us Part II*, as opposed to GOAP, which offers much higher adaptability in NPCs, thereby enriching emergent gameplay situations within *Middle-earth: Shadow of Mordor's* Nemesis System. Put together, these AI strategies underline how artificial intelligence is changing with each passing day in the realms of game playing—different environments, considering players' activities to intensify the feeling of involvement in reality.

# Task 3: Report on Sony’s Aibo Robotic Dog and Its AI Capabilities

## Introduction

Sony's Aibo robotic dog, first introduced in 1999 and re-envisioned in 2017, represents a unique blend of robotics and artificial intelligence designed to replicate the behaviors, emotions, and companionship of a living pet. Aibo uses AI-driven adaptability to respond to its environment, creating interactions that develop over time into a distinct identity or personality (Electronics Sony, 2023). This report evaluates Aibo's current AI capabilities and discusses potential enhancements, addressing two core questions:

* What AI characteristics does Aibo currently exhibit?
* What AI characteristics does Aibo still lack?

Aibo demonstrates advanced AI capabilities, positioning it as one of the most sophisticated robotic companions available today.

## AI Characteristics Exhibited by Aibo

**Personalized Learning and Adaptation**

Aibo's AI allows it to develop alongside its owner, creating a unique experience for each individual. Through advanced deep learning algorithms, Aibo builds a personalized memory that enables it to remember people, places, and events, fostering a bond similar to a real pet's relationship with its owner (Electronics Sony, 2023). This personalized adaptation is achieved through Sony's AI cloud system, which continuously processes Aibo's data to enhance user engagement (Wikipedia, 2024).

**Object, Facial, and Voice Recognition**

Aibo is equipped with high-precision cameras and sensors that allow it to recognize faces and objects, enabling it to distinguish individuals and engage in interactive activities like fetch. Its voice recognition also allows Aibo to respond to various commands, enhancing the user experience (Sony Robot, 2023). Over time, Aibo adapts to frequently used phrases, deepening the feeling of a genuine bond (Electronics Sony, 2023).

**Emotional Expression and Autonomous Interaction**

Aibo expresses emotions using OLED eyes and motorized components, performing actions like tail-wagging and head-tilting to convey emotions (Sony Robot, 2023). These autonomous responses foster user engagement by creating a sense of intentionality and personality, making the interaction more immersive (Wikipedia, 2024).

**Environmental Awareness and Navigation**

Through its touch sensors and cameras, Aibo detects obstacles, recognizes familiar spaces, and responds to physical touch, making it appear highly aware of its surroundings. This capability is essential for creating an engaging, immersive experience, allowing Aibo to dynamically react to its physical environment (Electronics Sony, 2023).

**Growth and Personality Development via the AI Cloud Plan**

Aibo's AI Cloud Plan allows it to store memories of interactions, refining its personality over time through data exchange with Sony's servers. This feature enables Aibo to develop preferences for specific types of petting or food, enhancing its role as a unique companion (Wikipedia, 2024; Sony Robot, 2023).

**Programmed Tricks and Visual Programming Capabilities**

Aibo can perform various tricks and dances, such as the "Shabondama" dance, adding to its charm and interactivity. The robot's visual programming interface also enables non-programmers to create custom movements and routines, enhancing its versatility as a companion (Electronics Sony, 2023).

## AI Characteristics Aibo Still Lacks

Despite its advanced AI capabilities, Aibo lacks certain features that could further enhance its functionality and lifelikeness. These include:

**Complex Language Comprehension and Contextual Reasoning**

Aibo’s language comprehension remains simple, limited to basic commands. Integrating large language models (LLMs) could allow Aibo to process complex, multi-step commands, improving its conversational abilities and contextual understanding (DeepMind, 2023). This enhancement would enable Aibo to respond naturally to complex instructions, making interactions feel more authentic.

**Advanced Task Execution Through Hierarchical Planning Models**

Aibo's task execution is currently limited to simple commands and routines. Implementing a Hierarchical Planning model would enable it to break down complex tasks into sub-goals, allowing it to perform functions such as organizing toys or locating specific items, thus enhancing its role as a versatile assistant (MIT News, 2023).

**Enhanced Real-Time Processing and Efficiency Models**

While Aibo's responsiveness is generally impressive, processing large volumes of sensory data can impact its response speed and battery life. Integrating an efficiency-focused model like Self-Adaptive Robust Attention for Robotics Transformers (SARA-RT) would optimize real-time data processing, allowing Aibo to engage more quickly without compromising performance (DeepMind, 2023).

**Vision-Language-Action Integration for Greater Versatility**

Aibo's visual recognition is currently limited to basic identification. Integrating vision-language-action models like Robotics Transformer 2 (RT-2) could enable Aibo to interpret complex visual scenes and respond to verbal cues with appropriate actions, making it more adaptable to varied environments and commands (Google DeepMind, 2023).

**Precision Motion and Lifelike Movement through RT-Trajectory Models**

Although Aibo's movements are expressive, they can sometimes lack the smoothness of a real animal. Incorporating RT-Trajectory technology could improve motor function precision, helping Aibo replicate the nuanced movements of a real pet and creating a more lifelike experience (DeepMind, 2023).