**Artificial Intelligence in Games Report**

**Final Year – Full Unit Project**

Chess with Artificial Intelligence

Alexander Mason

A report submitted in part fulfilment of the degree of

**BSc (Hons) in Computer Science**

**Supervisor:** Nery Riquelme Granada



Department of Computer Science

Royal Holloway, University of London

# Abstract

In this report, I will be researching about the evolution of Artificial Intelligence in video games over time. The reason for researching this topic is because it will give myself and the reader a better understanding of how AI has developed and what it was like in early concepts. This report will also discuss what makes AI good and what makes AI bad in games, as well as researching and mentioning examples of famous AI algorithms which have been applied to games.

From this, I aim to use this research to choose an Artificial Intelligence algorithm from those that I have researched, with my understand on how it works and what makes AI as a whole good in a game. This will achieve my overall goal to implement a one player mode in my Chess game with the implementation of the chosen algorithm. I also aim to give the readers of this report newfound knowledge or at the very least, a better insight to AI and its usage in games.

# Introduction

As explained in the Abstract section of this report, the main motivation for researching this topic is to enable myself and readers a better understanding of the usage and implementation of Artificial Intelligence in video games over time, from the early conceptions of AI to how it is used today in modern games.

Another motivation for the topic of ‘Artificial Intelligence in Games’ being researched is due to the huge link/relation that AI has in the Computer Science industry. Such examples include the development of the Deep Blue chess computer by IBM, which became the first computer chess-playing system to win a chess game against a world champion, whilst being under regular time controls. [[1]](#_Bibliography)

This was one of the first signs of Artificial Intelligence being effective and showing progress in the industry. It is my interest in this that led to my decision on what I am doing for my final year project, which is creating a Chess game and implementing Artificial Intelligence. Therefore, my research into Artificial Intelligence approaches is going to help with my objectives for my project.

# What is Artificial Intelligence?

To summarise, Artificial Intelligence is a field in Computer Science which demonstrates the intelligence of machines/computer systems. Not only this, but AI is also used to describe machines which can learn or study based on actions of a human player (more commonly known as machine learning but is often used in conjunction with Artificial Intelligence), and can be seen as machines which mimic normal/cognitive human functions. [2]

The basic goals of AI include:

* Reasoning: The ability for a computer system or program to automatically and fully reason.
* Knowledge Representation: The ability for a computer system or program to use knowledge about the world or state it is in which the computer can utilise to perform/complete tasks.
* Planning: The ability for a system to visualise/plan a set of actions it can take.
* Learning: The ability for the system to learn or add new information to its knowledgebase which it can then utilise to complete future tasks.
* Natural Language Processing: The ability for a computer system to interact with humans in their natural language, and how the computer can interpret and process the natural language into one it can understand itself.
* Perception: The ability for a computer system or program to gain useful information through the usage of hardware as a method of examining its surroundings and using this information to respond to the environment appropriately.

# History of Artificial Intelligence in Games

Artificial Intelligence in video games is a different aspect/subdivision of AI in general and from the summary described above. This is due to AI in video games having a different goal, which is to enhance the game experience as opposed to the academic side of AI, which is focused on the traditional goals and machine learning or decision making.

In the early 1950s, the implementation of AI has been a huge area of research since its inception at this time period. The earliest examples of AI being used in a computer game is the game Nim, which was made in 1951 and published in 1952.

In 1951, Christopher Strachey wrote a checkers program and Dietrich Prinz wrote a Chess program both using the Ferranti Mark 1 machine belonging to the University of Manchester. These were other examples of the earliest computer programs to ever be written. Developed during the middle of the 1950s to early 1960s, Arthur Samuel’s checkers program was the first to have enough skill to take on a highly regarded amateur in the game.

In the 1960s and 1970s, well-known games such as Spacewar!, Pong, and Gotcha all implemented discrete logic and involved two player game modes, meaning there wasn’t an AI implementation/one player mode.

The first one player mode games started to emerge in the 1970s, with notable games including Taito, Qwak, and Pursuit. There were also text-based (normally based in the command line) games which had enemies, the movements of which were based on coded patterns. Microprocessors were used in these games to allow more randomisation over the movement patterns as a way of making them less predictable. Two of these text-based games included Hunt the Wumpus and Star Trek.

Arguably, AI exponentially grew in popularity and development during the video arcade game era. This was mainly down to the success of Space Invaders (released in 1978) which had various difficulty levels, randomised movement patterns, and game events controlled by hash functions which used the player’s input. Pac-Man (released in 1980) also used Artificial Intelligence-controlled movement patterns for the enemies in the game, each of which having different traits to vary the gameplay.

In the 1990s, the emergence of new game genres and sub-genres used Artificial Intelligence in a more complex manner. For example, games using real time strategy used AI with multiple objects, gave the system incomplete information, pathfinding problems, and forcing the system to make decisions in real-time. In 1997, thanks to the progression of Artificial Intelligence and the building onto the early checkers and chess programs made in the 50s and 60s, IBM’s Deep Blue computer managed to beat Garry Kasparov (a chess grandmaster and former world chess champion) in the game.

In modern day games, existing techniques and methodologies are still heavily used as a way to control the actions of non-player characters (often abbreviated to NPCs). Most commonly used methodologies today are pathfinding and decision trees to guide the NPC. Artificial Intelligence is also used in processes not usually visible to the user, examples of which are data mining and procedural generation. [3]

# What Makes Good and Bad AI?

This is a short section in my report covering what is deemed good and bad AI in video games based on the opinions of those who play video games or develop them. In short, good Artificial Intelligence implementations are ones that are realistic and are fairly smart, but also not programmed perfectly in the sense that they can’t be beaten. It is important that the player feels they can still beat or get a sense of achievability from the game they are playing which includes a form of AI.

Bad AI in video games is said to be implementations which use Machine Learning. This is because the AI constantly becomes better as the player plays the game, it would be hard for developers to balance the AI so that it is still fun to play against. On a thread discussing what makes AI bad in games, one of the responses were; “The goal of machine learning AI would be to beat the player, while the goal of the game designer is to make the AI fun to beat (challenging but not overly so)”. Therefore, this backs up that it would be hard to balance as both concepts are conflicting.

Another example of bad AI implementations in video games include those that have the same movement patterns. This is because humans are good at recognising patterns and after a while the AI becomes predictable which can make the game easy and lose its appeal, especially when AI is used as a way to improve the overall game experience. This means that it is good to have an element of randomisation (RNG) in Artificial Intelligence approaches so that the AI’s patterns aren’t completely predictable. This gives a more realistic feel to the AI in the game, and in the case of enemies, means that they still pose a challenge (at least to some extent) and remain to have an element of unpredictability/ randomisation in what they do and their movement patterns. [4]

# AI Algorithms

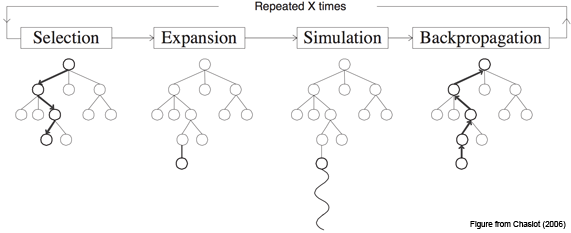
## Monte Carlo Tree Search

The Monte Carlo algorithm is used in computer systems as a way to determine the behaviour of another system, in this case a game implementing Artificial Intelligence. It mainly uses the element of randomness and statistics as a way to determine the outcome. The algorithm does not use fixed inputs and instead uses numbers to set the probability of a certain event happening in the outcome after the algorithm is run. [5]

Monte Carlo has been applied to Artificial Intelligence for games by developing it into the Monte Carlo tree search. This tree search is used to search for what is deemed the best move/outcome in a game.

The tree search has these four main steps:

1. Starting at the root node, recursively select the most optimal child nodes until a leaf node is reached.
2. If the leaf node is not a terminal node (meaning a node which doesn’t end the game), then create a child or children nodes and select one of them.
3. Simulate what would happen in the game (playout) until a result is achieved.
4. Update the tree/move sequence with the new child node with the simulation result (known as backpropagation). [6]



It is important to note that each of nodes in the tree should contain two pieces of information. These are an estimated value (payoff) based on the results of the simulation, and the amount of times the node has been visited.

### Monte Carlo Tree Search – Benefits

Aheuristic – The tree search does not need any form of strategic or tactical knowledge about the game it is used in to be able to make rational decisions. The only information the search does need of the game it is being implemented in, is what counts as a legal move and the terminating/goal states for the game. If you were to write a single implementation of the Monte Carlo tree search, you would be able to reuse that implementation into many other games with very few modifications needed.

Asymmetric – The tree search can asymmetrically grow the tree (its search space). It uses the algorithm to expand the optimal nodes much more often than the not so optimal nodes, therefore its searches are usually in the relevant parts of the tree.

This means that the Monte Carlo tree search can be used in games with large amounts of outcomes. A large combination of outcomes can be problematic or not as efficient if using other forms of tree searches such as breadth-based or depth-based tree searches. Therefore, due to how adaptive Monte Carlo is, it will find the moves that appear most optimal even if it takes a while to find and focus its searching efforts in that part of the tree.

Availability – One of the major benefits to the algorithm is that it can be stopped at any time to return the best estimate it has currently come up with. The tree that has been built so far can either be started over or kept for future use. [6]

### Monte Carlo Tree Search - Drawbacks

Playing Strength – This can be seen as a major drawback as the Monte Carlo tree search can sometimes fail to find the most reasonable move, even for games which aren’t that complex, within a reasonable amount of time. This is due to the size of the tree and that the search of the tree may not have visited all the nodes enough times to be able to give the best estimates.

Speed – Another major drawback with the Monte Carlo tree search is related to ‘Playing Strength’ listed above but presents speed as another drawback. The search can take numerous iterations to come up with the best solution, and as a result of this, it wouldn’t make the game very efficient and would make it harder to optimise. In the Chess game I am producing where there are lots of outcomes, a search which does not require numerous iterations would be more optimal/efficient on both the system’s resources and actual compute time on the system running the game. [6]

Both these drawbacks can be improved on so that the search’s performance is better. An approach known as **domain knowledge** can be used as a method to filter out implausible/unrealistic moves so that the moves made by the search are more similar to those of a human player. The benefits of this are fewer iterations, resulting in faster compute time and more realistic playoffs.

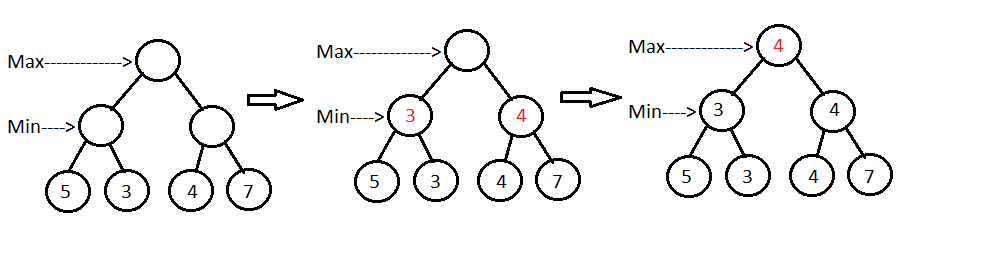
## Minimax

Minimax is a decision-based Artificial Intelligence algorithm where one player attempts to achieve the maximum payoff/result against the opponent player, who is trying to minimise the payoff by countering the move (normally the computer). IBM’s Deep Blue computer (made in 1997) was using the Minimax algorithm to defeat Garry Kasparov, who was a chess grandmaster and former world champion in the game.

In terms of the algorithm, it is assumed that the goal for the player is to play optimally in order to win the game, whereas the goal of the computer is to play strategically and minimise the payoff of the human player’s move. The algorithm does this by iterating through a search tree and selecting the nodes with the lower evaluation scores (lowest payoffs). The algorithm can be designed in such a way so that it is smart, by having the ability to look at potential moves the opponent can make in response to the player’s move. [7]

The tree search has these three main steps:

1. Traverse the tree using a depth-first search, meaning the tree looks at the lowest level nodes known as the leaf nodes. As it is the min player’s turn to choose a move in the tree, the evaluation function looks at the values of the two nodes and stores the lower value in the parent node.
2. Recursively store the scores from the leaf or children nodes (whether it is a leaf of child node depends on the level of the tree we are traversing) to their parent nodes until reaching root. When it is the min player’s turn, keep selecting the leaf or child node with the lowest score. When it is the max player’s turn, keep selecting the leaf or child node with the maximum score.
3. Once the root node is reached, the higher of the two values is selected and the move with the higher payoff is made. [8]



### Minimax Tree Search – Benefits

Decision Making – It was one of the first algorithms to make decision making based on evaluating pay off scores as a way to determine the computer’s next move. It is widely used in computer board games such as Chess, but due to its drawback being a major one, it is better used in the conjunction with the alpha-beta pruning algorithm.

### Minimax Tree Search - Drawbacks

Speed – One of the biggest drawbacks for the Minimax tree search is that it can be time consuming for the algorithm to compute which move it will make if there are a lot of branches (possible outcomes) in the game it is being implemented in. As the algorithm would be implemented in Chess, the complexity, compute time and branching factor of the algorithm would be too high.

As it can be very time consuming to build a game tree for all possible outcomes or compute the highest payoff move for a game with a high branching factor, the biggest improvement that can be made to the Minimax algorithm is pruning. The pruning process is used in the Alpha-beta pruning algorithm, which is an optimised variant of the Minimax algorithm.

## Alpha-Beta Pruning

Alpha-Beta pruning is a variant of the Minimax algorithm. It is used to optimise the compute time and overall efficiency of the Minimax algorithm by allowing the search to be faster. Alpha-Beta pruning works by cutting off branches (prune them) which do not need to be searched in future iterations of the tree because there already exists a better/more optimal move. Therefore, the algorithm does not need to take that branch of the tree into account after the branch is pruned. Alpha-Beta pruning gets its name by using alpha and beta parameters, which are additional to the parameters needed for the minimax algorithm.

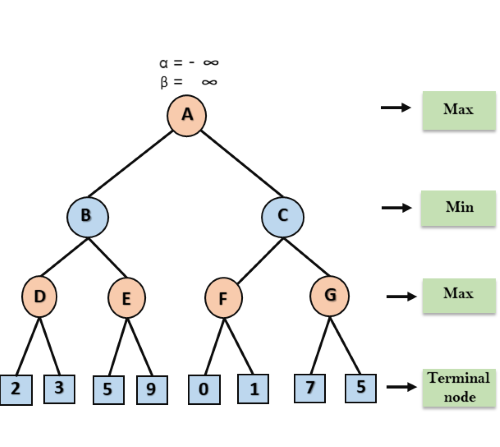
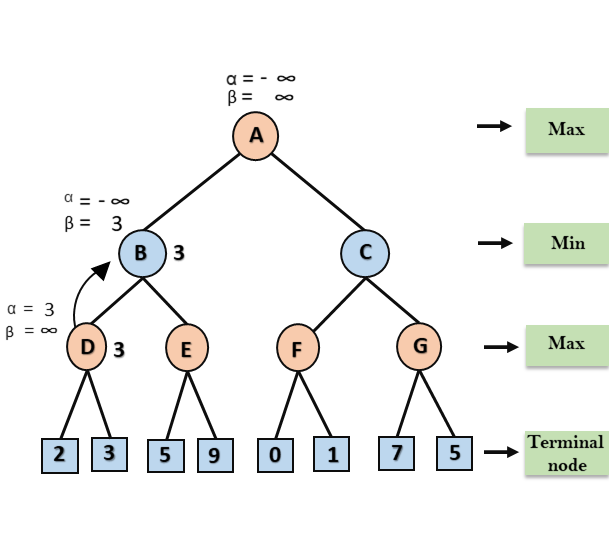
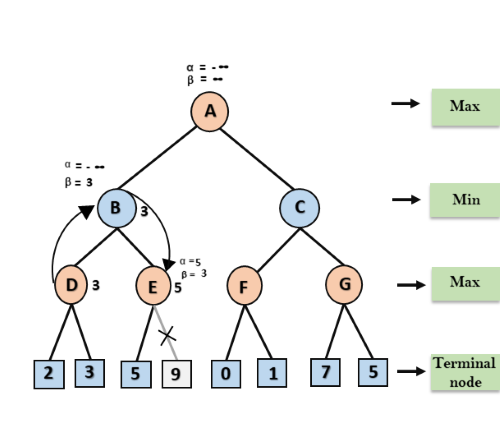
Here is what the two parameters are used for:

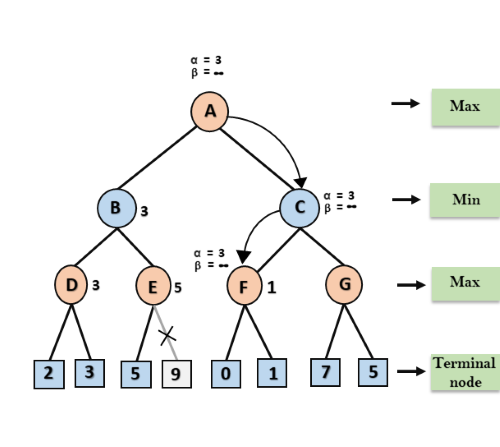
Alpha: The best value that the maximiser can guarantee at any point in its path so far.

Beta: The best value that the minimiser can guarantee at any point in its path so far.

The main pre-requisites for alpha-beta pruning are that the max/maximiser player will only update the value of alpha, the min/minimiser player will only update the value of beta, while backtracking the tree the nodes will be passed to upper/parent nodes instead of alpha and beta values, and we will only pass alpha and beta values to the child nodes. [9]

How Does Alpha-Beta Pruning Work?

1. Max player starts from node A where alpha = -∞ and beta = +∞. These alpha and beta values are passed down to node B, and then to child node D.
2. At node D, it is Max player’s turn so the value of alpha will be compared with 2 and 3. As 3 is the higher value, the value of alpha and node D’s value will both be set to 3.
3. The algorithm backtracks to node B where it is Min player’s turn. Beta will compare with the available subsequent nodes value, so for now beta is set to 3 and alpha is set to -∞ again.
4. The algorithm goes to node E, it is Max player’s turn again. The current value of alpha will be compared with 5 (so max( -∞, 5) = 5). Therefore, at node E, alpha = 5 and beta = 3.
5. The algorithm backtracks from node B to node A. At node A, the value of alpha is changed to the maximum available value which is 3 as max (-∞, 3) = 3 and beta = +∞. These values get passed to node C, and then node F.
6. At node F, the value of alpha once again gets compared with the terminal nodes. The left child is 0 so max(3, 0) = 3, and the right child is 1 so max(3, 1) = 3. As both remain 3, alpha remains the same but node F’s value is set to 1 as it’s the highest of 0 and 1.



1. Node F returns the value 1 to node C. At C the current values are alpha = 3 and beta = +∞. The value of beta will be changed as it is Min’s turn. It will compare with 1 so min(∞, 1)= 1. At C, we have alpha = 3 and beta = 1. It satisfies the condition that alpha is larger than or equal to beta. The next child of C is G, which is pruned because C contains a beta smaller than alpha/alpha larger than or equal to beta.
2. Node C returns the value 1 to node A, and Node B is returning the value 3. As it is Max’s turn, the best value for node A would be 3, as max(3, 1) = 3. This concludes that the optimal value for Max is 3. [9]

### Alpha-Beta Pruning – Benefits

Efficiency: The algorithm can reduce the number of nodes it needs to visit (via pruning) so that the optimal move is found in a lower compute time. If the algorithm uses good ordering (explained in drawbacks below), it can be twice as fast as a Minimax algorithm without any sort of alpha-beta pruning implementation.

### Alpha-Beta Pruning – Drawbacks

Ordering - The efficiency/effectiveness of the implementation of alpha-beta pruning depends on the ordering of which nodes are examined. If the algorithm does not prune any of leaves or branches in the tree, it is in worst ordering. It basically means the algorithm is the same as the Minimax algorithm with this ordering and is very slow.

For an efficient implementation, the algorithm should be using ideal ordering. Ideal ordering occurs when lots of pruning happens on the leaves and branches in the tree. The best moves are found in the left side of the tree. Depth first search is used hence the searching from the bottom left of the tree to until it recursively gets to the leaf nodes on the right-hand side of the tree.

To find good ordering:

* Occur the best move from the shallowest/leaf nodes
* Order nodes in such a way that the best nodes are examined first
* Make use of domain knowledge to find the best move. For example, for Chess the priority could be captures first, then threats, then forward moves, and then backward moves.
* Record the states as there is a possibility that states may repeat. [9]

# Bibliography

1. ***Deep Blue Chess Computer - Wikipedia***

<https://en.wikipedia.org/wiki/Deep_Blue_(chess_computer)>

This resource contained information in the chess-playing computer named “Deep Blue”, which beat the first Chess world championship under time controls.

1. ***Artificial Intelligence – Wikipedia***

<https://en.wikipedia.org/wiki/Artificial_intelligence>

This resource was used to summarise and define what Artificial Intelligence actually is, the origins of AI, and the traditional problems (known as goals) of AI.

1. ***Artificial Intelligence in Video Games – Wikipedia***

<https://en.wikipedia.org/wiki/Artificial_intelligence_in_video_games>

This resource was heavily used to talk about the history of Artificial Intelligence and its implementation in video games through time.

1. ***Why is AI Still So Bad in Games Despite the Increasing Power of Gaming Machines Over the Years – Reddit***

<https://www.reddit.com/r/pcgaming/comments/3uml0l/why_is_ai_still_so_bad_in_games_despite_the/>

This resource was used to gather opinions from other people who play games themselves or perhaps develop games regarding what makes AI bad in games, and what aspects of AI is currently done well.

1. ***Monte Carlo Algorithm – Wikipedia***

<https://en.wikipedia.org/wiki/Monte_Carlo_algorithm>

This resource was used to research the Monte Carlo algorithm and summarise what it is.

1. ***Monte Carlo Tree Search – MCTS.ai***

[https://web.archive.org/web/20151129023043/http://mcts.ai/about/index.html](https://web.archive.org/web/20151129023043/http:/mcts.ai/about/index.html)

I used this resource to explain the four main processes involved in a Monte Carlo tree search, I also used the source to explain the benefits and drawbacks of the search algorithm.

1. ***Game Theory – The Minimax Algorithm Explained – Towards Data Science***

<https://towardsdatascience.com/how-a-chess-playing-computer-thinks-about-its-next-move-8f028bd0e7b1>

I used this resource to explain/summarise the goals of the Minimax algorithm.

1. ***Introduction to Minimax Algorithm – Baeldung***

<https://www.baeldung.com/java-minimax-algorithm>

This resource was used to explain the actual process of the tree search in the Minimax algorithm as to how the computer evaluates the move it will take.

1. ***Alpha-Beta Pruning – Javatpoint***

<https://www.javatpoint.com/ai-alpha-beta-pruning>

This resource was very useful in understanding how Alpha-Beta pruning is applied to the Minimax algorithm and how it functions. The example and images used above are from this website.