# The path to the right decision: An investigation into using heuristic pathfinding algorithms for decision making

Ashley Smith

December 7, 2019

**Abstract**

Lorem ipsum dolor sit amet, consectetur adipiscing elit. Cras justo velit, vestibulum sit amet turpis in, interdum rhoncus magna. Proin pulvinar posuere iaculis. Duis vulputate tristique arcu, id pretium ante blandit ut. Vestibulum ante ipsum primis in faucibus orci luctus et ultrices posuere cubilia Curae; Nam augue tellus, mattis quis consequat id, facilisis eu lectus. Vivamus euismod non quam sed condimentum. Orci varius natoque penatibus et magnis dis parturient montes, nascetur ridiculus mus. Orci varius natoque penatibus et magnis dis parturient montes, nascetur ridiculus mus. Phasellus vitae consequat nisi. Morbi vulputate tellus ut nibh vulputate, vitae blandit ex faucibus.

# Contents

# 1 Introduction

### 1.0.1 Video games and artificial intelligence

Games are good for the global economy. Newzoo reports that in October 2019 the global games market was worth $148 billion with mobile games accounting for 46% of that. In order to remain competetive, developers are pushed to make bigger, better and more complex games. The evolution of the technologies available has allowed for a greater number of elements to be simulated in the game world which could potentially increase a game's worth (Blow, 2004). The entry barrier to creating these good looking and well executed games has been lifted with the establishment of engines like Unity (Unity Technologies, 2005) and Unreal (Epic Games, 1998) to the point where even non-programmers can get involved using textual or visual scripting.

For many years, only the game's graphics were considered important (Yap, 2002; Blow, 2004), however, good physics systems or competent AI (Artificial Intelligence) are now recognised as a way of improving the end-user's experience just as much well (Blow, 2004). Game AI is different from the normal, academic AI as it simulates behaviour and aims to be believable and fun, whereas an academic AI aims to achieve a level of intelligence or autonomy to excel at a given task (Nareyek, 2004, p.60).

### 1.0.2 The need for game AI

AI has multiple uses within a games context but the majority of use cases employ AI to control the characters featured in a level of the game. Laird and VanLent (2001, p.16) said that whether these characters are replacements for opposing players or characters that act as companions, villains and plot devices, "human-level AI can expand the types of experiences people have playing computer games". Using AI opens up the opportunity for increasing the difficulty of the game, and could be perceived as the kind of challenges that make games fun (Buro, 2004, p.2). Laird and VanLent (2001, p.16) also hypothesised that utilising such an AI is a good step towards the development of enjoyable and challenging gameplay, and potentially to "completely new genres" (Laird and VanLent, 2001, p.17).

"Customers value great AI" (Nareyek, 2004, p.60) and so it's important to choose a suitable approach that fulfills the expectations of the player thus the requirements of the game (Millington, 2019, p.19). Academic AI can be made using algorithms inspired by biology such as neural networks or genetic algorithms and trained through iteratation or with datasets. These approaches aren't used in game AI because of the high requirements to train the AI to interact with a specific game (Nareyek, 2004, p.64), moreover, it is easy to train the AI to play too strongly ruin the game (Tozour, 2002, p.13). Instead, game AI developers embrace simpler, non-learning algorithms due to them being easier to understand, implement and debug (Tozour, 2002, p.7).

### 1.0.3 Approaches to game AI development

The most basic forms of AI used in games take the form of a series of if-then statements and are known as a 'production rule systems' (Tozour, 2002). These statements are organised in a list and the AI uses the behaviour of the rules that evaluate to *true*. The result is a very basic AI that is not only limited to what actions it can take but also when it can take them. Similarly, decision trees combine the same if-then style with branching structures to create game AI (Nareyek, 2004, p.62), and the tree and subtrees are recursively traversed until a leaf node is found with the desired behaviour. This process is very easy to understand and implement (Millington, 2019, p.295), and the branching structure makes visualisation of the process more intuitive than the basic list used in a production rule system. Because of this, many consider decision trees to be one of, if not the simplest techiques to making AI (Millington, 2019, p.295; Tozour, 2002, p.7).

FSMs, or Finite State Machines, are the most common approach to game AI (Orkin, 2006, p.1; Millington, 2019, p309) due to being easy to understand and the efficacy of their output. FSMs consist of a directed graph where each node represents a state and the edges represent the transitions between them (Tozour, 2002, p.6). A character can only be in one state at time and has no memory of any previous states (Colledanchise, Marzinotto, and Ögren, 2014); each state represents an expected behaviour and determines what they do and the conditions to switch to a different state (Diller, Ferguson, Leung, Benyo, and Foley, 2004, p.3). In the right environment, the impression of a well thought out FSM could compete with that a neural network, at a fraction of the time and resource costs, despite not always arriving at the optimal decisions (Sweetser and Wiles, 2002). However, each new behaviour requires the creation of a new state and the conditions of which this state integrates and transitions into other states, making expansion and maintainance cumbersome (Sweetser and Wiles, 2002, p.2; Lim, Baumgarten, and Colton, 2010, p.3). There's no easy way to combine the tests inside of FSMs and selecting the conditions for a state transition is still very much a process that must be done by hand (Millington, 2019, p.313).

The need for more flexibility in game AI has lead to the creation adaptation of modular algorithms such as behaviour trees (Lim et al., 2010, p.1). Like decision trees, the recursive structure of a behaviour tree is simple to understand and implement while also being high level, allowing for more sophisticated AI to be created in a modular fashion through the use of subtrees and different node types (Shoulson, Garcia, Jones, Mead, and Badler, 2011, p.144), each performs an action or check and then proceeds to succeed or fail (Lim et al., 2010, p.4. It is these types that make the AI process at a higher level than standard decision trees, and when combined with leaf nodes that perform checks and actions to build trees, and then combined again to make trees containing subtrees, the simplicity and elegance of this technique certainly demonstrates why behaviour trees are getting attention (Shoulson et al., 2011, p.144).

### 1.0.4 The relationship of pathfinding and game AI

One common requirement for game AI is for the characters to be able to traverse the areas of the game in a way which meets the player's expectations logically and efficiently - a task known as pathfinding. Regardless of what an AI decides to do, a pathfinding mechanic needs to be in place to allow the AI to navigate to where it needs to go, maneuvering around obstacles while still taking a sensible route (Graham, McCabe, and Sheridan, 2003, p.60). For most games, the algorithm of choice is A* as it is the de-facto standard pathfinding algorithm (Millington, 2019, p.197; Botea, Müller, and Schaeffer, 2004, p.2; Nareyek, 2004, p.64; Leigh, Louis, and Miles, 2007, p.73).

The A* algorithm analyses the game's map and generates a path from one location to another while minimising a *cost* value - this value can represent anything but usually it represents the time or distance to travel along a given route (Yap, 2002, p.44). This means that the pathfinding algorithm itself doesn't decide where to go, only how to get there and the manner in which it does so. When asked to calculate a path, a pathfinding algorithm is provided a graph of nodes to determine which nodes can be reached from which (Nareyek, 2004, p.61). The algorithm isn't concerned with the what the data represents or in what form it is given, whether it is two-dimensional or three-dimensional, what the *cost* is of travelling from one node to another, as long as it is equipped with the right functionality to digest this information (Millington, 2019, p.277; Graham et al., 2003, p.60).

With the pathfinding process taking place after the decision has been made, the opportunity to involve the data gathered from pathfinding algorithm is missed. Often, the AI will decide to approach the nearest object, but obstacles in the way mean that the cost of navigating to the destination is greater than some alternative. While implementing the algorithm isn't difficult, ensuring the AI generates a path to the correct destination is difficult to do well (Forbus, Mahoney, and Dill, 2002). Perhaps on the way to the destination, the character has to also navigate past traps or other hazards where it will need to decide whether to avoid or pass through - decisions that A* isn't fully equipped to deal with on it's own. If the AI wanted to factor in these hazards and real cost values, it would have to perform a more advanced check on where to travel too, maybe even using the pathfinding algorithm multiple times to definitely make sure that it wants to take the generated path, potentially ruining the game's performance.

Pathfinding algorithms are actually general purpose search algorithms applied to a spacial context (Cui and Shi, 2011, p.125; Orkin, 2003, p.6; Yap, 2002, p.46); there is no such restriction that these algorithms should be restricted to pathfinding when in a games context. Millington (2019, p.197) said that "pathfinding can also be placed in the driving seat, making decisions about where to move as well as how to get there". With search algorithms having the flexibility of being able to traverse graphs of nodes representing any kind of data, it's no stretch to imagine the A* search algorithm being applied to a graph containing the same tests and actions found in decision and behaviour trees in order to generate a 'path' of actions rather than a path of spacial data (Higgins, 2002, p.114). This is paper aims to investigate and re-engineer A* to make decisions rather than paths in a game context.

## 1.1 Literature Review (11-12 pages)

### 1.1.1 Dijkstra's algorithm: A graph and tree search algorithm

A search algorithm is a recursive method designed to find a match for a piece of data within a collection such as an array, graph or tree (Friedman, Bentley, and Finkel, 1976). A piece of data is provided and the search algorithm typically returns whether it is present and it's location. (Dijkstra)'s algorithm (1959) is a search algorithm that operates on trees and graphs (which are then interpreted as trees). The algorithm calculates the shortest difference from any node on the graph to any other node. If a destination is provided, the algorithm can be terminated early to avoid unnecessary computation.

Dijkstra's algorithm works through the recursive summation and comparison of distance values (Dijkstra, 1959, p.269) - for each neighbour, the current node's distance from the start is added to the length between the current node and its neighbour. If this tentative value is lower than the current distance value of the neigbour, it replaces it. When all the neighbours have been considered, the node with the lowest tentative value on the graph is selected and permanently 'visited' (Dijkstra, 1959). This process is repeated until the destination has been visited, and thus a path and the distance from the start to the destination can be retrieved. The problem with Dijkstra's algorithm is that it always selects the node with the lowest tentative distance value, meaning that the algorithm has no notion of direction and is calculating the lowest distance to nodes that may not be relevant to getting to the destination (Millington, 2019, p.214).

### 1.1.2 A* algorithm: A heuristic best-first search algorithm

A* is an improvement of Dijkstra's algorithm in this regard (Hart, Nilsson, and Raphael, 1968, p.101) - while it doesn't stray far from how Dijkstra's algorithm works in the sense that it operates using a tentative distance value and it keeps track of the nodes that have and haven't been visited, it does extend the algorithm using what's known as a heuristic approach (Cui and Shi, 2011, p.126). "Heuristics are criteria, methods or principles for deciding which among several alternative courses of action promises to be the most effective in order to achieve some goal" (Pearl, 1984, p.3). This means that in a pathfinding situation, a heuristic function could estimate the distance to the goal, by ignoring walls and measuring in a straight line, to direct the algorithm in the right direction and avoid evaluating routes that travel in the wrong direction to make the process more efficient (Cui and Shi, 2011, p.127). Heuristics enable A* to perform a best-first search (Yap, 2002, p46), as the heuristic now has the power to select which node is the best one to evaluate and prioritises it over the others (Russell and Norvig, 2016, p.94). A good heuristic algorithm should explore nodes that have the most potential for leading to the goal node (Korf, 1985).

When identifying the next node to expand, A* will take this heuristic distance into account using the formula $f(n) = g(n) + h(n)$ (Hart et al., 1968, p.102; Russell and Norvig, 2016, p.95), where $g(n)$ is the real distance the algorithm has calculated from the start node to node $n$, $h(n)$ is the heuristic distance from the current node $n$ and the destination, and $f(n)$ is the combination of these two metrics forming an estimate of the distance from the start node to the destination if travelling through route $n$ (Hart et al., 1968; Millington, 2019; Graham et al., 2003, p.64).

This heuristic component of A* transforms it into a family of algorithms where applying a different heuristic selects a different algorithm (Hart et al., 1968, p.107), moreover, implementing A* and using a heuristic that returns a constant value for all nodes reverts A* back into Dijkstra's algorithm (Lester, 2005, p.10; Millington, 2019, p.237). Conversely, implementing a well-designed heuristic method can be used to guarantee optimal solutions, and using a heuristic that is somewhere in-between can output results with varying degrees of accuracy in exchange for faster exection (Millington, 2019, p.219). The implementation of a good heuristic can be difficult, as making the heuristic take more factors into account for accuracy has the drawback of making the algorithm less efficient overall with the heuristic being frequently used throughout the process.

On the other hand, Graham et al. (2003, p.68) argue that one of the constraints of games the industry is the "over-reliance" on the A* pathfinding algorithm and describe the development of its many extensions as a way of avoiding the discovery of new techniques. The pathfinding process can require a lot of CPU resources, sometimes to the point of stalling the game, when applied to larger graphs (Cui and Shi, 2011, p.127; Stentz, 1996, p.110; Graham et al., 2003, p.67). This indicates that the performance of A* can vary depending on various factors, and is why it is important to optimise A* by selecting an suitable storage mechanism for the graph and internal node storage as well as using a reasonable heuristic that balances efficiency and efficacy (Millington, 2019, p.228).

### 1.1.3 Decision making with A* using GOAP

Orkin (2003, p.11) expressed that expectations of AI are growing with the release of every new game, and that "we need to look toward more structured, formalized solutions to creating scalable, maintainable and re-usable decision making systems". Game AI techniques used in most games do not contain the scalability Orkin envisions (Laird and VanLent, 2001, p.17), however, Higgins (2002, p.117) declares that a pathfinding engine can be created generically, especially if you use a templated language to give the code even more reusability. Implementing A* in this way would allow it to be used for both regular pathfinding and any other uses you can get out of it(Higgins, 2002, p.120). With A* being efficient and optimisable (Millington, 2019, p.215), reusing A* for game AI has the potential to bring the benefits it usually brings to pathfinding while being adaptable enough to scale up and meet expectations.

Orkin (2006, p.1) worked on the development of the game AI for the game F.E.A.R (Monolith Productions, 2005). The approach used for this AI was called GOAP which stands for 'Goal Oriented Action Planning' and actually uses the A* algorithm as part of its decision making process and "allows characters to decide not only what to do, but how to do it" (Orkin, 2003, p.1). GOAP changes the data to be processed by A* from spacial data such as coordinates into character AI state data, such as what you'd find in FSMs. Therefore, the output of A* is no longer a sequence of movements but a sequence of actions also known as a plan (Orkin, 2003, p.2; Tozour, 2002, p.6).

GOAP essentially takes the idea of having state machines to encapsulate behaviours and replaces the hand-programming the conditions and connections of state transitions with a pathfinding process with the aim being the decoupling of states and their transitions (Orkin, 2003, p.2). The pathfinding process uses each node to represent a state and the edges between each node as the actions that lead to those states (Orkin, 2003, p.7) When GOAP is used, a goal is given for the character to achieve which gives the AI some direction and what is returned is the sequence of actions that will satisfy the goal (Orkin, 2003, p.1). An action is a representation of one thing the character will do to change the world in some way, like opening a door or picking up a weapon (Orkin, 2003, p.2); some actions have preconditions that require the execution of another action prior to it (Orkin, 2003, p.5). A* will then find the sequence of actions, the plan, that satisfies the character's goal while minimising an arbitrary *cost* value - Orkin (2003, p.4 - p.5) suggests that this process creates interesting character AI that can adapt to change while also having a code structure that is reusable, maintainable and "elegant".

A* being used in GOAP means that fundamental formula $f(n) = g(n) + h(n)$ needs to be implemented: the calculation of a node's $fitness$ to determine the *cost* of the actions taken up to that point and a heuristic function to determine the *cost* of getting to the goal from a given node (Orkin, 2003, p.7). Orkin (2003, p.7) also shares that the heuristic in GOAP can be calculated as the summation of unmet conditions of the goal node, but this is rather unclear. Given the example in figure 1, "the goal is to kill an enemy" (Orkin, 2003, p.3) could be interpreted in various ways that all seem unsuitable. If the condition is that the enemy is dead, the heuristic would not change for any action other than the goal state, making it potentially wasteful like Dijkstra's algorithm (Millington, 2019, p.214). Alternatively, if the condition was that the weapon needed to be drawn and ready to fire,

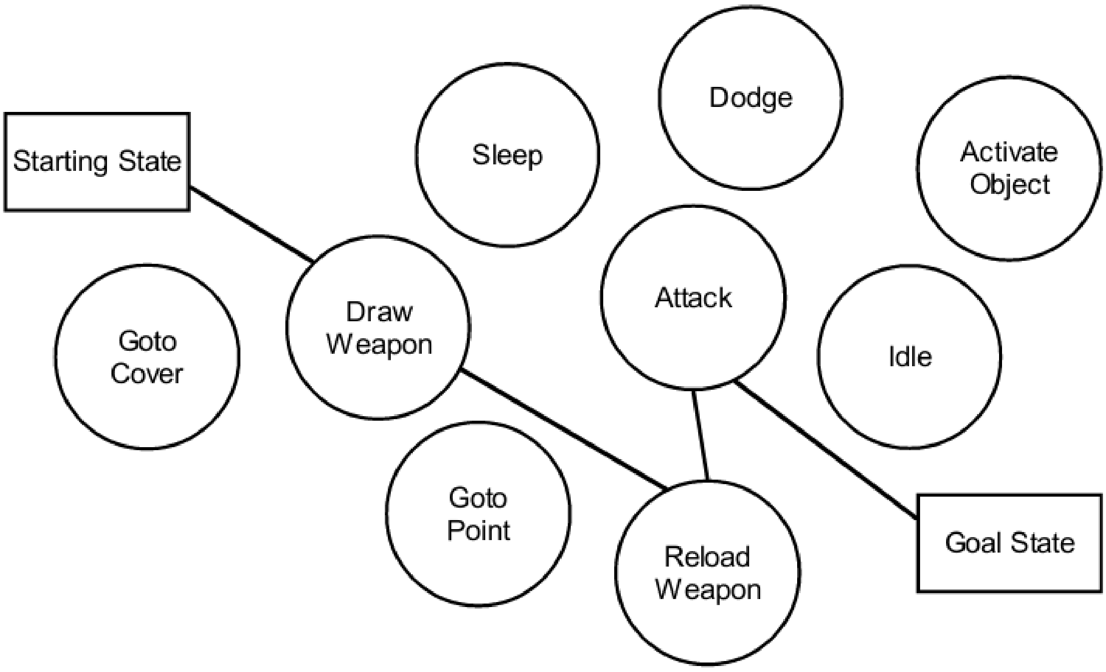


Figure 1: An example of GOAP (Orkin, 2003, p.3)

it implies either the algorithm has been ran previously to determine the requirements of killing an enemy, or that they were programmed by the programmer - how would the starting state know that the weapon needed to be reloaded before drawing it? While GOAP does seem elegant, the graph in figure 1 doesn't have a sizeable number of nodes; using dijkstra's algorithm wouldn't incur much of a performance cost with a graph this size. Orkin (2003) suggests that a regressive search by starting at the goal and pathing to the start makes sense, and in figure 1, it does, as only one action can attack and so it's clear that the decision making process boils down to the character attacking the enemy.

Another problem of GOAP is that the high-level nature of the approach may not allow for enough control of the game (Stanciu and PETRUŞEL, 2012, p.87) - while making abstractions of the world is necessary for AI to process the information (Buro, 2004, p.2) and thinking on a higher level is beneficial for making more humanlike decisions, the precision of the actions taken by the AI is both low level and very noticeable if incorrect (Graham et al., 2003, p.60). Figure 1 shows a decision making process that decides what to do but not necessarily where to do it. When the action "Goto Point" in figure 1 get executed, there's no indication of how the destinations are generated other than satisfying the preconditions for another action (for example, moving to an object to activate it) (Orkin, 2003, p.7).

This isn't specific to GOAP as it applies to every AI approach where the generation of these locations isn't part of the AI. Is the character's AI deciding where to go, or just deciding that it has to go somewhere? In order for "Goto Cover" to act in a similar way to the "Goto Point" action, a goal or action precondition would either have to require the character to be out of line-of-sight, or be using one of the designated cover spots. The latter of these would require the designation of these cover spots and then A* would be employed to find which coverspot would be the quickest to navigate to. The former would require calculating the closest position that allows the escape of line-of-sight. Both of these would have their own problems, and wrapping the entire pathfinding process as a single action would mean that the information from the pathfinding request would not be used in the decision making at all. Without utilising the AI's ability to create an impression of thought, the information given to these actions could be poorly chosen and "may be perceived as a lack of intelligence by a human player" (Graham et al., 2003, p.63). To circumvent this, multiple "Goto Point" actions would be needed so that the planning system can evaluate each point into the workflow, but this could introduce its own problems without a well-designed heuristic that can distinguish between the different locations.

### 1.1.4 Perception and processing of the game world

Every approach to designing game AI needs to find a way to digest information about the character's environment that is relevant to the decision making process, and selecting this information is just as important as the form it is delivered in (Cui and Shi, 2011, p.126). All game AI solutions need the world to be re-interpreted to be better suited for both decision making (Buro, 2004, p.2) and pathfinding (Diller et al., 2004, p.3). Particular geometry of the map may need to be interpreted as vantage points, choke points or safe spots; particular formations of enemy units may need to be not only counted but also assessed for tactical strengths, whether engaging the units head-on is better than running away to a better location, and finally, the interpretation of time such as whether there is enough time to navigate to what would otherwise be a better location for fighting the enemy (Buro, 2004).

The digestion of the world needs to be done for both performance and gameplay needs. A* can suffer from performance problems when used with a large or inefficient graph of nodes and many solutions have been discovered to prevent or reduce the impact on performance. One such solution is known as hierarchical pathfinding, where the game's map is simplified into chunks, (Cui and Shi, 2011, p.126). By simplifying a group or grid of nodes into a single node that represents the whole group (like a quadtree), the pathfinding process can be applied to larger worlds as if they were actually smaller (Botea et al., 2004).

This works well for typical pathfinding, but applying this notion to something like GOAP would require a different approach. There are large amount of actions a character can take at any given moment (Nareyek, 2004, p.62) and so applying a search algorithm to such a large set of tasks will make the process a lot slower than traditional methods if left unchecked. Hierarchical pathfinding essentially creates more nodes to expand in the short term to reduce the total number of nodes expanded in the long term and gain a net increase in performance, and so in order to apply it to decision making like GOAP, nodes would have to be grouped by other metrics that aren't necessarily distance, such as 'attacking' actions and 'defending' actions. Splitting the world state gameplay elements like this can get complicated - what if a certain unit in a strategy game can attack from a longer range, would attacking the enemy from a distance be a defensive maneuver or an offensive one, and how easy is this to change on a per-unit basis (Weber, Mateas, and Jhala, 2011).

D* was also implemented for a large node count (Stentz, 1996, p.110).

END

In this paper, the mechanisms of the A* search algorithm are examined and re-engineered, through the substitution of input and output types, to investigate the modularity and adaptability of an AI made in this way. The aim of using this approach is to bring decision-making and pathfinding closer together and therefore simplifying the overarching process of perceiving, deciding and interacting in the game world.

### 1.2 Methods and Methodologies

## 2 Design, Development and Evaluation

## 3 Design (14-15 pages when combined with development)

### 3.1 Development (14-15 pages when combined with design)

### 3.2 Results and Evaluation (11-12 pages including critical review)

## 4 Conclusions

### 4.1 Conclusions and Critical Review

## References


Blow, J. (2004). Game development: Harder than you think. Queue, 1(10), 28.

Botea, A., Müller, M., & Schaeffer, J. (2004). Near optimal hierarchical path-finding. Journal of game development, 1(1), 7–28.

Buro, M. (2004). Call for ai research in rts games. In Proceedings of the aaai-04 workshop on challenges (pp. 139–142). AAAI press.

Colledanchise, M., Marzinotto, A., & Ögren, P. (2014). Performance analysis of stochastic behavior trees. In 2014 ieee international conference on robotics and automation (icra) (pp. 3265–3272). IEEE.

Cui, X., & Shi, H. (2011). A*-based pathfinding in modern computer games. International Journal of C 11(1), 125–130.

Dijkstra, E. W. (1959). A note on two problems in connexion with graphs. Numerische mathematik, 1(1), 269–271.

Diller, D. E., Ferguson, W., Leung, A. M., Benyo, B., & Foley, D. (2004). Behavior modeling in commercial games. In Proceedings of the 2004 conference on behavior representation in m (pp. 17–20).

Epic Games. (1998). Unreal engine (Version 4.23.1). Retrieved November 30, 2019, from https://www.unrealengine.com/en-US/

Forbus, K. D., Mahoney, J. V., & Dill, K. (2002). How qualitative spatial reasoning can improve strategy game ais. IEEE Intelligent Systems, 17(4), 25–30.

Friedman, J. H., Bentley, J. L., & Finkel, R. A. (1976). An algorithm for finding best matches in logarithmic time. ACM Trans. Math. Software, 3(SLAC-PUB-1549-REV. 2), 209–226.

Graham, R., McCabe, H., & Sheridan, S. (2003). Pathfinding in computer games. The ITB Journal, 4(2), 6.

Hart, P. E., Nilsson, N. J., & Raphael, B. (1968). A formal basis for the heuristic determination of minimum cost paths. IEEE transactions on Systems Science and Cybernetics, 4(2), 100–107.

Higgins, D. (2002). Generic a* pathfinding. AI Game Programming Wisdom, 114–121.

Korf, R. E. (1985). Depth-first iterative-deepening: An optimal admissible tree search. Artificial intelligence, 27(1), 97–109.

Laird, J., & VanLent, M. (2001). Human-level ai's killer application: Interactive computer games. AI magazine, 22(2), 15–15.

Leigh, R., Louis, S. J., & Miles, C. (2007). Using a genetic algorithm to explore a*-like pathfinding algorithms. In 2007 ieee symposium on computational intelligence and games (pp. 72–79). IEEE.

Lester, P. (2005). A* pathfinding for beginners. online]. GameDev WebSite. http://www. gamedev. ne

Lim, C.-U., Baumgarten, R., & Colton, S. (2010). Evolving behaviour trees for the commercial game defcon. In European conference on the applications of evolutionary computation (pp. 100–110). Springer.

Millington, I. (2019). Ai for games. CRC Press.

Monolith Productions. (2005). F.E.A.R.

Nareyek, A. (2004). Ai in computer games. Queue, 1(10), 58.

Newzoo. (2019). 2019 global games market per device and segment. Retrieved November 30, 2019, from https://newzoo.com/key-numbers/

Orkin, J. (2003). Applying goal-oriented action planning to games. AI game programming wisdom, 2, 217–228.

Orkin, J. (2006). Three states and a plan: The ai of fear. In Game developers conference (Vol. 2006, p. 4).

Pearl, J. (1984). Heuristics: Intelligent search strategies for computer problem solving.

Russell, S. J., & Norvig, P. (2016). Artificial intelligence: A modern approach. Malaysia; Pearson Education Limited, Shoulson, A., Garcia, F. M., Jones, M., Mead, R., & Badler, N. I. (2011). Parameterizing behavior trees. In International conference on motion in games (pp. 144–155). Springer.

Stanciu, P.-L., & PETRUŞEL, R. (2012). Implementing recommendation algorithms for decision making processes. Informatica Economica, 16(3).

Stentz, A. (1996). Map-based strategies for robot navigation in unknown environments. In Aaai spring symposium on planning with incomplete information for robot problems (pp. 110–116).

Sweetser, P., & Wiles, J. (2002). Current ai in games: A review. Australian Journal of Intelligent Inform 8(1), 24–42.

Tozour, P. (2002). The evolution of game ai. AI game programming wisdom, 1, 3–15.

Unity Technologies. (2005). Unity engine (Version 2019.2.14). Retrieved November 30, 2019, from https://unity.com

Weber, B. G., Mateas, M., & Jhala, A. (2011). Building human-level ai for real-time strategy games. In 2011 aaai fall symposium series.

Yap, P. (2002). Grid-based path-finding. In Conference of the canadian society for computational stud (pp. 44–55). Springer.