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

Ashley Smith

December 1, 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.

1 Introduction (3-4 pages)

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 in the growing and changing market, developers are pushed to make bigger, better and more complex games. Blow (2004) describes how the evolution of the technologies available allowed for a greater number of elements to be simulated in the game world at the same time, which in turn could increase a game's overall potential worth. Video games are just as much about their design as they are about their programming, and the entry barrier to creating 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, making a good game has been about improving graphics to be higher quality and more realistic (Yap, 2002; Blow, 2004). However, it is becoming clear that graphics aren't everything and that good physics systems or competent AI (Artificial Intelligence) can improve the end-user's experience just as much as graphics if not more (Blow, 2004). When referring to AI in a games context, there is a significant difference between the approaches used to create academic AI and the AI used in games. The average game AI simulates a decision making process where the actions to be taken are as believable and fun as possible in the game's context whereas the objective of an academic AI is typically to achieve a level of intelligence or autonomy that enables the AI to optimally excel at a given task (Nareyek, 2004, p.60). It's not difficult to make a simply AI that always wins against the player, but player's wouldn't enjoy a game they couldn't win (Tozour, 2002), and so the distinction between the objectives of these approaches is very important.

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 (nicknamed 'Bots') or used to create NPCs (Non Player Characters) to act as companions, villains and plot devices in a role-playing or narrative context, "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, so long as the AI can be adapted to the player's strength, this difficulty will be perceived as the same 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). While "customers value great AI" (Nareyek, 2004, p.60), the fact that marketing material cannot advertise enjoyable game AI as easily as appealing graphics means that, in comparison to graphics or physics, AI rarely gets the same time or resource investment it needs to improve (Nareyek, 2004, p.60).

When developing game AI, it is very important to choose a suitable approach that fulfills the requirements of the game. Academic AI can be made using algorithms inspired by biology such as neural networks or genetic algorithms and trained through iteratation or with datasets. Genetic algorithms simulate natural selection and evolution using data to immitate gene crossover and mutation (Tozour, 2002). Similarly, neural networks simulate neuron patterns in the brain to map inputs to outputs (Tozour, 2002). These approaches aren't used in game AI due to the requirements necessary to train, tune and test the AI, moreover, the AI would need to be tailored to play a given game in order to take advantage of game specific requirements (Nareyek, 2004, p.64). 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 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). The AI is divided into two parts: the first being the ruleset that the AI has to follow and the second is the execution of the response to said rules. 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. AI created in this way is perfectly suited to simple games with a small number of scenarios that the programmer can check for and respond to by hand. Additionally, games where there are a lot of agents needing to make decisions simultaneously would also benefit from a basic AI.

Decision trees are another way to create AI that have applications for games. Building on the same fundamentals as production rule systems, decision trees accept a series of inputs such as the game environment and ask questions in the same if-then style (Tozour, 2002). Decision trees are branching structures containing nodes, where each node represents a test condition that leads either a sub-tree or a leaf, and the leaves are dead-end nodes that represent an action the AI should take (Nareyek, 2004, p.62). Traversal of a decision tree begins at the root node, where each conditional statement selects which child node to investigate next (Nareyek, 2004). This process repeats until a leaf node is reached containing the action the AI should take. 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 (Millington, 2019, p309) due to the balance between their ease of understanding and the efficacy of their output. Jeff Orkin (2006, p.1), a programmer who worked on F.E.A.R (Monolith Productions, 2005), said "If the audience of the Game Developers Conference were to be polled on the most common A.I techniques applied to games, one of the top answers would be Finite State Machines". FSMs consist of a directed graph where each node represents a state and the edges represent the transitions between them (Tozour, 2002, p.6). What the states represent depend on the game, but they usually represent an expected behaviour of the character. A character can only be in one state at time and the state determines how they act at any given moment as well as the conditions necessary to switch to a different state (Diller, Ferguson, Leung, Benyo, and Foley, 2004, p.3).

INSERT IMAGE OF STATE MACHINE EXAMPLE HERE

When using FSMs, controlling character behaviour can become quite straightforward as long as there is a clear idea about what is to be expected of a character and that these expectations of the character aren't too demanding. The process is still simple to implement like a decision tree due to the significant usage of rules that describe when the current state should switch to another (Nareyek, 2004, p.61), however, with a well designed FSM, the notion that a character's thought process is stateful gives off the impression that the character is indeed thinking. Given the correct game environment, this impression could appear to be good enough to compete with a neural network at a fraction of the time and resource cost despite not always being arriving at the optimal solution to the character's situation (Sweetser and Wiles, 2002).

END

As video games have evolved, so too have strategies used to design and implement artificial intelligence (AI) into the characters and logic necessary for the player's enjoyment. The requirements for game AI are not the same as academic AI; the characters involved simply need to interact with the environment in a manner which makes the game fun to play. 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.

Typically, the pathfinding side of AI is separate from the decision making side. Some algorithm will perceive the world and come to a conclusion about what to do and then pass this information on to

the pathfinding algorithm in charge of navigating to the desired area. This means that the pathfinding algorithm doesn't do any 'thinking' and instead just generates a path from one point to another. The path is generated after a decision has already been made, and so any information gathered that could otherwise be useful to the decision-maker is lost.

Without knowing the path or any specific details about points of interests along said path beforehand, a decision-making process could easily make mistakes that could be considered wrong or 'glitchy' by both players and developers. Similarly, factoring in environmental details like these into the decision runs the risk of overcomplicating the process and making it harder to manage and maintain, or repeating calculations made in the following pathfinding process.

In this paper, the mechanisms of a typical pathfinding algorithm are examined and re-engineered, through the substitution of input and output types, with the aim of bringing decision-making and pathfinding closer together.

1.1 Literature Review (11-12 pages)

Decision trees are the simplest way to implement AI but are basic.

FSMs are one of the most common ways of implementing AI (Three states and a plan: the AI of FEAR (228 cites), but they also don't provide a simple way of combining these tests together to make more complex queries.

Behaviour trees are an improvement over FSMs.

Current techniques do not scale up (Human-level AI's killer application: Interactive computer games)

A solution is an action sequence, so search algorithms work by considering various possible action sequences (Artificial Intelligence: A modern approach). These pathfinding algorithms can search for a solution for traveling from A to B, but could potentially do more with the redefinition of what an action is. In theory, you could apply 'best first' to game interactions in a scenario.

Normally, pathfinding takes a backseat when it comes to decision making and just generates the requested path. However, since pathfinding algorithms are just search algorithms being applied to graphs, there's no restriction that these nodes and edges have to correspond to locations and distances.

Dijkstra's algorithm is a simple pathfinding algorithm that finds the shortest route to every node in the graph. It's an uninformed algorithm.

AStar is the defacto algorithm for pathfinding. Its actually a family of algorithms where the heuristic dictates how it functions. AStar with no heuristic is dijkstras, and could also be used to become a best-first, breadth first and depth-first.

Changing how A* evaluates actions and applies a heuristic will allow it to operate differently. A good heuristic will allow the algorithm to discard 'bad moves' (Depth-first iterative-deepening: an optimal admissable tree search).

A* becomes inefficient on bigger maps and lots of techniques have been created to adjust to these problems. Having lots of edges per node could make A* very inefficient.

This paper will investigate the effects of different cost and heuristic functions when changing the types involved with A* to see how agents perform.

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.

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

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 modeling and simulation (brims)* (pp. 17–20).

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

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

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

Sweetser, P., & Wiles, J. (2002). Current ai in games: A review. *Australian Journal of Intelligent Information Processing Systems*, *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

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