Bachelor's thesis

Information and Communications Technology

2024

Jesse Friberg

Artificial Intelligence of Non-Playable Characters in Video Games

TURKU AMK
TURKU UNIVERSITY OF APPLIED SCIENCES

Bachelor’s Thesis | Abstract

Turku University of Applied Sciences

Information and Communications Technology

2024 | 32 pages

Jesse Friberg

Artificial Intelligence of Non-Playable Characters in Video Games

Artificial Intelligence created by game developers for non-playable characters is one of the most important parts of developing a fully fleshed video game. Even though the subject is an important part of the industry, it is not talked about enough and documentation on the subject in general is rather lacking. The objective of this thesis was to seek out the most common solutions to creating an artificial intelligence that feels intuitive and helps the player immerse themselves in the game they are playing. The goal was to study these solutions and find out how they are used.

The method of research used in this thesis started with looking at developer logs, articles, and conferences held by game developers, and then gathering these sources to a spreadsheet. Data would then be extracted from these sources and be summarized. After enough data was collected, conclusion was drawn on which solutions were the most common and would be studied further in this thesis.

As a result of this study, it was shown that the four most common solutions used were Goal-Oriented Action Planning (technique used to create goal-oriented AI), Hierarchical Finite State Machines (nested Finite State Machines), Navigation mesh (abstract data structure), and A* (A-star, search algorithm). These four solutions were the most common and many of the other solutions were derived from one of these four which shows that the reason they are so popular is their flexibility and ability to be derived.

Keywords: artificial intelligence, video game, character, immersion, developer

Opinnäytetyö (AMK) | Tiivistelmä

Turun ammattikorkeakoulu

Tieto- ja viestintätekniikan koulutus

2024 | 32 sivua

Jesse Friberg

Ei-pelattavien hahmojen tekoäly videopeleissä

Videopelien ei-pelattaville hahmoille luotu tekoäly on yksi tärkeimmistä asioista videopelin kehityksessä. Vaikka aihe on tärkeä, se on jäänyt vähäiselle huomiolle. Opinnäytetyön tavoitteena oli tutkia yleisimpiä tekoälyratkaisuja, jotka tuntuisivat luontevalta ja auttaisivat pelaajaa immersoitumaan pelaamansa peliin. Näitä ratkaisua oli siis tarkoitus oppia ymmärtämän ja saada selville, miten niitä käytetään.

Tekoälyratkaisuja tutkittiin etsimällä kehittäjien lokeja, artikkeleja ja kehittämisen alan konferensseja, minkä jälkeen löydökset taulukoitiin. Löydettyjen lähteiden pohjalta kerättiin dataa, josta selvitettiin yleisimmin käytössä olevat ratkaisut. Yleisimpien löydettyjen ei-pelattavien hahmojen tekoälyratkaisutyyppien ominaisuudet ja toteutustavat selvitettiin yksityiskohtaisesti.

Tutkimustulokset osoittivat, että neljä yleisintä ratkaisua olivat Goal-Oriented Action Planning (tekniikka tavoitepohjaisen tekoälyn luontiin), Hierarchical Finite State Machine (äärellinen automaatti, joka koostuu muista äärellisistä automaateista), Navigation mesh (abstrakti tietorakenne), ja A* (A-star, polunetsintäalgoritmi). Löydetyt neljä ratkaisuja olivat yleisimmät ja useat muut ratkaisut olivat johdettu näistä neljästä, mikä osoittaa sen, että nämä ratkaisut ovat suosittuja niiden joustavuuden ja johdettavuuden vuoksi.

Asiasanat: tekoäly, videopeli, hahmo, immersio, kehittäjä

Contents

Appendices

Pictures

Tables

List of abbreviations

AC	Artificial Consciousness
AGI	Artificial General Intelligence
AI	Artificial Intelligence
ANI	Artificial Narrow Intelligence
ASI	Artificial Super Intelligence
FPS	First-Person Shooter
FSM	Finite State Machine
GDC	Game Developers Conference
GM	Game Master
GOAP	Goal-Oriented Action Planning
HFSM	Hierarchical Finite State Machine
MMORPG	Massively Multiplayer Online Role-Playing Game
NPC	Non-Playable Character
RPG	Role-Playing Game

1 Introduction

Non-playable characters (NPC) in video games have always been one of the most important aspects of creating an immersive and engaging experience. Immersion is one of the most desired aspects in video games (Michailidis et al. 2018). However just existing is not enough, as having a functional and smart artificial intelligence (AI) is one of the most important parts of constructing a good NPC. This thesis focuses on researching what kind of solutions game developers use to craft the perfect AI for their different kinds of characters in order to immerse the player in the game.

The motivation for this research stems from the lack of scientific research on AI regarding video games. Creating a working artificial intelligence for a specific character is something that is critically important in the development of video games, but it is something that is not talked about enough when discussing the video game industry. Finding documentation on how certain games handle AI is possible, as will be seen in this thesis, but finding collection of the most common solutions that developers use, is much harder. Game development evolves constantly year by year and especially for young and upcoming indie video game developers, documentation on game development is vital.

This thesis will seek to explore the most common solutions that video game developers use in their AI for their NPCs. This will be done by compiling data from developers and finding what tools are commonly used and for what kind of NPCs. Data will be based mostly on developer held conferences, in which they go in depth on what kind of AI solutions they have used in past or are currently using.

While there are a massive number of different solutions for AI development in games, this thesis will only be focusing on few of the most commonly used ones. There are a lot of niche systems, that are usually derived from one of the common tools, and a lot of game studios' in-house systems that will not be covered in this thesis.

2 Artificial Intelligence and Its Evolution

Artificial intelligence, also referred to as AI, is technology that enables machines, particularly computers, to exhibit intelligence of sorts. Even though artificial intelligence was officially coined as a term in 1956, the concept of AI has been in the minds of researchers from the late 1930s.

2.1 Early Foundations

Turing (1950) in his book Computing Machinery and Intelligence asks the question: “Can machines think?” and as an answer presented a simple test, which later became known as the Turing Test. This test’s purpose was to determine whether a computer could be able to replicate human intelligence. In 1956 the Darthmouth workshop, where the term AI was officially coined, was held by Marvin Minsky, John McCarthy, Claude Shannon, and Nathan Rochester. During this workshop John McCarthy coined the term Artificial Intelligence as “the science and engineering of making intelligent machines.”

2.2 Successful Beginnings

Since the Darthmouth workshop to all the way to 1974, science of artificial intelligence would thrive and prosper with computers becoming more powerful at a rapid rate. Machine learning algorithms were improving and people's understanding of them got only deeper as time went on. New and upcoming concepts, that would demonstrate the progress of the field were brought to life during this era:

- *Unimate*, the first industrial robot installed on a General Motors assembly plant during 1961. Invented first by George Devol in the 1950s.
- *ELIZA,* a computer program developed by Joseph Weizenbaum from 1964 to 1967, that would try to simulate responses that a therapist would give in an intimate conversation.
- *General Problem Solver,* a computer program to simulate human problem solving developed by Herbert A. Simon and Allen Newell during 1957.

2.3 First AI Winter

1974 would mark the beginning of the first "AI winter", meaning a period of time where the whole field of artificial intelligence faced a lot of scepticism towards the limitations of AI, and this subsequently would lead to funding of AI related projects being reduced tremendously. Because of this there would be fewer significant developments. The lack of funding and there simply not existing enough computing power would be the fundamental issues of the first AI winter that would last until 1980. (Kaul et al. 2020)

2.4 Resurgence

During the 1980's, the popularity of AI was reinvigorated due to AI once again having new breakthroughs and more funding. The introduction of "deep learning", introduced by John Hopfield and David Rumelhart, allowed computer to learn from experience. During these times Edward Feigenbaum introduced the so called "expert systems", which allowed the program to ask questions from experts in specific fields and learn from answers given. This allowed the program to mimic the decision-making process of experts. These expert systems were widely used in different industries. The Japanese government established the "Fifth Generation Computer Project", during which they massively funded the expert systems and other AI related projects. Aim for this project was to revolutionize computer processing and to improve artificial intelligence. This funding allowed expert systems to thrive, but most of the other projects invested in were not able to meet the ambitious goals set for them. (Rockwell, 2017)

2.5 Second AI Winter

Even though expert systems enjoyed a wide range of success in various industries, it wasn't without a fault and in 1984 John McCarthy criticized expert systems for its lack of common sense and it not being able to see its own limitations. This combined with other AI projects failing to meet expectations led to decrease in funding and in public interest. (Schuchmann, 2019)

The second AI winter has no clear start or end date, but it is most commonly said to have spanned from the late 1980s to early 1990s. The public had lost interest, and this led to fear of funding AI research and such fear eventually led to actually the funding ending.

2.6 Modern Day AI

In modern day AI has exploded in popularity, largely due to much more powerful computing power and understanding of more advanced algorithms. AI technology is currently being widely utilized in many different fields, such as industry, science, and software development. Some examples of AI, that almost everyone has used at some point include web search engines (such as Google), recommendation algorithms (such as YouTube or Netflix), virtual assistants (such as Siri or Google Assistant), generative AI tools (such as ChatGPT), or inhuman bot players in chess.

Artificial intelligence can generally be categorized into two different categories: weak AI and strong AI. Weak AI, commonly also known as artificial narrow intelligence (ANI) is specifically trained and made to focus on performing specific tasks. Most of the AIs that are utilized today are categorized as weak AIs. Strong AI is a theoretical concept, that can be defined in many ways, but the most common ones are made up of Artificial General Intelligence (AGI), Artificial Consciousness (AC), and Artificial Super Intelligence (ASI).

Theoretically a machine using strong AI could achieve consciousness and intelligence rivalling or even surpassing humans and perform vast range of cognitive tasks. While strong AI is purely theoretical as of today, there are still many different research projects doing research on it and its possibilities in the future. (Bringsjord and Govindarajulu, 2018)

3 Artificial Intelligence as a Part of Video Games

AI in video games is something that is often overlooked when discussing the development of characters in games, even though it is one of the most important steps when it comes to creating a character that feels like it makes intelligent decisions. This is important for the sake of immersing the player in media. Immersion is one of the biggest reasons for one to play video games (Michailidis et al. 2018). Immersion is not the only reason to have good AI, as it can also be used to manage the difficulty of the game. Enemy NPCs can be made harder or easier depending on the difficulty selected by the player. This is especially relevant in games where the player fights NPC characters, such as FPS games. In FPS games, the developers are always able to change the accuracy or damage output of NPCs, and this will affect how difficult they are for the player to beat (Hubble et al. 2021).

3.1 Non-Playable Character

Non-playable character (NPC, sometimes also known as non-player character) in video games is a character operated by a computer. These NPCs vary vastly in behaviour depending on genre of games they are in and often even games of same genre will have big differences between their NPCs. For example, in most first-person shooters (FPS) NPCs are programmed to emulate real players and mimic their behaviour to either battle with or against the player. Or in massively multiplayer online role-playing games (MMORPG) NPCs are more often very static and usually much more dumbed down compared to FPS or other action type genres. Friendly NPCs in these games mostly exist for the player to interact with and be given dialogue or a quest. Unfriendly NPCs in MMORPGs however are often referred to as mobs (mobile objects) and like their friendly counterparts are also very simple in behaviour. The difference usually being that mobs usually roam around a certain area and are aggressive to the player.

Even though the term NPC is commonly associated with video games in modern day, the term actually predates video games and originates from traditional tabletop role-playing games (RPG) such as Dungeons & Dragons. In these kind of tabletop RPGs NPCs are characters portrayed by the gamemaster (GM) and are there to interact with players participating in the game. Despite the switch to a more modern medium, the term NPC still holds all the same characteristics and depending on the game takes on variety of different sub terms. Since NPCs can differ vastly from game to game and even media to media, there is always an ongoing debate on what really constitutes as an NPC. For the purpose of this text, it will not be discussed whether something should or should not be considered an NPC. This allows for a common term for all entities that will be discussed in this thesis, without having to go into several sub genres of NPCs.

Even though the purpose of this thesis is not to discuss what does and does not constitute as an NPC, there needs to be some kind of definition established for what will be considered as an NPC during this analysis. For the purposes of this text for a character to be considered an NPC the character must meet all the following three criteria:

- Entity must be rational.
- Entity must act with intention.
- Entity must not be controlled by the player.

The entity must act in a way that it can be considered to have intended to act with a consciousness of some sort and that it seems to act out its actions with purpose and intention. The intended consciousness by the game developers is a key part of my definition of an NPC in this analysis, since it makes sure that a vending machine for example couldn’t be considered an NPC, even though it might be a non-controllable entity that has purpose when a player interacts with it. But when this same purpose is filled with a merchant, who gives the player dialogue and some sort of theatrics when the player interacts with it, that merchant will be considered an NPC.

For the purpose of this text the entity must also not be controlled by the player at any point during the game. There are many games in which the player controls certain characters only for a portion of the game, such as many turn-based RPGs, even if outside of those select portions, they could be considered NPCs. For this text, only entities that are completely controlled by the game will be considered NPCs.

This will serve as the most basic definition of an NPC for this text, however, as will become evident in later sections, the complexity of NPCs varies vastly from a simple roaming monster to a much more advanced sidekick character whose actions vary greatly depending on the situation.

NPC's complexity as a character can often have a direct link on how complex the decision making or AI of said NPC is. While not true one hundred percent of the time, it is a good rule of thumb and since the purpose of this text is to analyse artificial decision making and not characters as a whole, it won't be an issue.

Aarseth (2012) presents a theory, that in narrative video games characters can be classified into three different categories based on their depth or the lack of it:

- Deep characters
- Shallow characters
- Bots

Deep characters (such as Handsome Jack from Borderlands 2) are characters with clear individual identities, and they evolve along with the player as they progress through the story of the game. Shallow characters (such as Nomadic Merchants from Elden Ring) are characters with individual names and appearance that is distinct enough to identify, but little personality and a shallow backstory. Bots are fodder enemies or allies that have no clear individual identity and only exist to offer a simple interaction with the player (such as practice range robots in Valorant).

3.2 Different Types of AI in Video Games

Purpose of this thesis is to research and find out what different types of solutions game developers have implemented in order to create an artificial intelligence that suits their vision for specific NPCs. To achieve this, conferences, developer logs, and articles were compiled into a spreadsheet (Table 1). These entries were then looked through and the data extracted from that was the basis of this research. Main source of this research was the Game Developers Conference (GDC) and their YouTube channel. GDC is a conference that is held annually and hosts a multitude of lectures held by various game development industry's professionals and a lot of these lectures have been recorded and uploaded to their YouTube channel. While GDC is not the only source, it is by far the biggest, since they host massive amounts of professionals, with credentials to back their work.

	A	B	C	D	E
1	**Link**	**Name of the game**	**Main focus**	**Where is it used in game**	**Genre**
2	https://www.youtube.com/watch?v=fQb0Iaj50bU	./ThisIsTheGame	GOAP	All friendly NPCs	Metanarrative/Metafiction
3	https://www.youtube.com/watch?v=LHEcpy4DjNc	Watch Dogs 2	Anecdote Factory	Non-hostile, non-police civilian NPCs	Action/Adventure
4	https://www.youtube.com/watch?v=yqZE5O8VPAU	Death Stranding	A* search algorithm & Navigation Mesh	Hostile enemy NPCs	Story-rich/Action
5	https://www.aiandgames.com/p/the-ai-of-dark-souls	Dark Souls	Unnamed AI system (Derived from GOAP and Hierarchical Goal Network)	Boss NPCs	ARPG/Soulsborne
6	https://www.youtube.com/watch?v=LxWq65CZBU8	Marvel's Spiderman	HFSM	Enemy NPCs	Action/Adventure
7	https://www.youtube.com/watch?v=tB88gTpdk48	Assassin's Creed Syndicate	Navigation mesh	NPC Vehicles	Stealth/Adventure
8	https://www.youtube.com/watch?v=e7AWHT7j3V4	Sid Meier's Civilization III & IV	Soft-Coded AI	Bot players	Turn Based Strategy/4X
9	https://www.youtube.com/watch?v=wusK-mciCVc	BioShock Infinite	Navigation mesh & "Smart Terrain"	Partner NPC Elizabeth	Adventure/FPS
10	https://www.youtube.com/watch?v=VoXSJBVqdek	Dishonored 2	Navigation mesh & Behaviour trees	Hostile NPCs and Civilian NPCs	Stealth/Adventure
11	https://www.youtube.com/watch?v=dnGzEn6swqo	The Last Of Us	"Buddy AI"	Partner NPC Ellie	Action/Adventure
12	https://www.youtube.com/watch?v=RFWrKHM0vAg	Splinter Cell: Blacklist	Navigation mesh, Artificial perception and awareness	Enemy NPCs	Stealth/Shooter
13	https://www.youtube.com/watch?v=PLchlSvrSpc	Wisplight	Vision cones, pathfinding & agression	Enemy NPCs	Action/RPG
14	https://www.aiandgames.com/p/the-ai-of-horizon-zero-dawn	Horizon Zero Dawn	Hierarchical Task Network Planner	Hostile Monster NPCs	Action/Adventure
15	https://www.aiandgames.com/p/how-commander-ai-works-in-halo-wars	Halo Wars	Data driven AI	Bot players	RTS

Table 1: Survey of different solutions used in video game AI development. Links of the sources found in appendix 1.

3.3 Goal-Oriented Action Planning

Goal Oriented Action Planning (GOAP) is a complex AI technique that is regarded as one of the best AI systems to use in video games. GOAP was developed by Jeff Orkin in the early 2000s for the 2005 video game F.E.A.R.

The way GOAP works is that it breaks down complex tasks or goals into series of smaller and simpler actions that the agent can perform. In this case ‘agent’ refers to the NPC that is being controlled by GOAP-system. Once the actions are broken down, they are organized into a sequence of actions that create a path towards the goal of the agent.

After a path to the objective is set, the agent begins by evaluating its current state and the goal state. The agent then goes through a library of possible steps to take that will push it towards the state of the goal. Each action has a set of conditions that must be met for that action to be executed and after each action that is completed, the agent re-evaluates its state and all the remaining actions left for it to reach its goal state. If the agent encounters a change in the environment or another kind of obstacle, it can dynamically adjust its sequence of actions to find a new path of actions that will lead it to its goal state.

GOAP is a powerful technique that is particularly useful in games that have a complex environment and multiple different goals for its NPCs, as it allows them to adapt to an everchanging situations and make their own unique decisions based on their current state and their goal state. This allows for the creation of multiple unique feeling NPCs with differing personalities and behaviours by weighting the priority of different actions and goals.

3.3.1 Towards Real-Time Planning

Jeff Orkin, the creator of GOAP, in his paper Three States and a Plan: The A.I of F.E.A.R (2006) states that the complexity of AAA games in the early 2000's was getting unmanageable. Especially AIs for games were getting exponentially more complex, not because of any one specific action, but because the number of different combinations and interactions between several different AI behaviours wouldn't be realistically manageable. According to Orkin, introducing real time planning of actions instead of having everything work within predetermined finite state machines (FSM), was their attempt to solve this problem of complexity. Orkin states that big motivator for this solution was also the fact that they only had one AI programmer, and they wanted to move some of the workload from the single programmer to the AI themselves. Real time planning allows the AI to make decisions depending on what is happening around it, instead of following a predetermined path set by the state of FSM. This not only helps massively with managing the ever-growing complexity of AI, but also creates a more unique experience for the player, as the AI does not just follow the same script and become predictable.

3.4 Hierarchical Finite-State Machine

Hierarchical finite-state machine (HFSM) is achieved when regular finite-state machines (FSMs) are nested in a way that certain states of a state machine are implemented as state machines themselves. Finite-state machine is a model of computation comprised of one or more states. Only one state of an FSM may be active at any given time, and it can change from one state to another depending on inputs that it is given. These FSMs become hierarchical when they are nested inside of each other. HFSM has at least one or more so-called sub state machines that themselves can also have state machines inside of them. In game development this technique is often used when several states have a lot of overlapping code and can be inherited to sub states from a parent state to avoid having to write the same code over and over again. An example how a simple HFSM might work in video game could look something like a cook NPC being controlled by an FSM that has the sates “Sleep”, “Work”, “Talking”, and “In combat”. Inside the “Work” state can be another FSM that has the states “Gather flour”, “Gather eggs”, and “Use stove”. The state “Use stove” can be another FSM with states “Light the stove”, “Combine ingredients”, and “Cook” and so on.

3.4.1 Hybridization of Two Known Methods

In Marvel's Spiderman (Table 1, row 6), developers ran into a problem in which their behaviour trees for enemy AIs were becoming way too complicated and there were just too many enemy types to cover. They stated that the game has 64 unique AI classes just for enemies alone. Having a unique behaviour tree for each one was just not feasible, since the trees had become so complex. Debugging such massive trees was also a nightmare for developers.

The system that the developers created for Marvel's Spiderman makes it so, that all enemies are derived from a single parent behaviour tree and use data driven methods, in this case FSMs, to handle most of the complicated stuff like what combinations of attacks will the AI use or when to run for cover. This is used in conjunction with regular behaviour trees, so that things like animation transitions are handled purely by code.

3.5 Navigation Mesh

Navigation mesh (Navmesh) is a data structure that is used to represent the accessibility of a 3D space or area by fracturing said area or space into a bunch of polygons. Navmesh is one of the most common tools for game developers is almost always relevant in some capacity in most 3D games since NPCs' navigation is such an important part of creating an AI that looks and feels like it is making intelligent decisions based on the environment around it. NPC navigation is especially important to the immersion of the game since when it goes wrong it is very easy to notice it as a player, for example NPCs getting stuck on corners and just running in place or straight up walking through what should be an impassable surface.

Navmesh stores data about given space or area and this data is then used to create available navigation options for the AI. Three main types of information that navmesh stores are accessibility, connections, and values. Accessibility refers to what parts of any given area are accessible to the AI. Areas (commonly also referred to as "maps") often have obstacles such as pitfalls, furniture, doors, and even other NPCs and these have to be accounted for in navigation. Navmesh accomplishes this by dividing the already divided are into even more polygons for the obstacles. How these different polygons are divided and connected to each other is the "connections" part of the data stored inside the navmesh. This data is most important for artistic reasons as the makers of the map won't just create one big chunk of an area, but instead map usually consists of many smaller segments and navmesh has the data that tells AI how these chunks are connected. These different chunks or smaller segments of the map are often connected "links" that dictate who can get from one segment to another and how they need to do it. For player characters this might be something like a door or a tight squeeze between some tight shelves and large enemies not being able to follow them. Or for NPCs this might look like soldiers climbing a ladder to gain advantage from higher ground. Moving between areas through these links are often tied to some kind of criteria and usually require some sort of animation.

These criteria are usually some state that the NPC needs to be in for the criteria to be met. States could be something like fear, chasing the player or relocating. The third type of information that navmesh stores is different values assigned to different surfaces on the map. These values are determined by the developers and are used to make the AI prioritise surfaces with higher values to make AI feel like it is making an actual decision to go to these surfaces while making it so that it is not impossible for it to enter lower valued surfaces. These values can be visualized as different surfaces each giving a certain number of points if entered by an AI. This system can also be used to make sure enemies avoid certain surfaces at any cost, without making any changes to the surface itself that could affect anyone who is allowed to enter the area. For example, the developer could allow the player to enter water sections and swim in there while also making sure no NPCs follow the player into water.

3.5.1 Navigating Using Surface Values

Navigation mesh is one of the most essential tools in video game development when it comes to creating an immersive experience with the games AI. The sheer versatility of navmesh allows it to be used to create so many different experiences regarding NPC movement while always still essentially doing the same thing further proves why it is so essential to game developers. Even with limited number of examples gathered in Table 1 it can be seen that navmesh is used widely in different ways in different genres as shown on rows 4, 7, 9, and 12.

In BioShock Infinite (Table 1, row 9) navmesh is in a massive role in helping create an “alive” feeling NPC partner called Elizabeth, who accompanies the player throughout the whole game. Biggest goal that the developers had was to create a character that feels like she belongs in the vast open world and interacts with objects scattered around the world. Scripted and hardcoded encounters with Elizabeth wouldn’t really work that well since the game is open

world and the philosophy of the game wasn't really one that strictly tells the player where to go and when.

They also didn't want Elizabeth to just aimlessly follow the player everywhere they went. Using navigation mesh's value setting property and an algorithm that would try to predict players movement they created a system that would make Elizabeth move with the player instead of just following the player. They used this in conjunction with a system they coined "smart terrain" that places interaction triggers to inanimate objects that Elizabeth could interact with. Combining these two systems allowed to create a pathfinding AI for Elizabeth that felt natural the way she moved around the player and interacted with objects around her.

In Assassin's Creed Syndicate (Table 1, row 7), the developers needed to create an AI system that allowed NPCs to operate horse carriages and would be able to chase the player throughout the city. This is nothing too complicated, as it is mostly just basic pathfinding to follow the player, but the biggest challenges came from the shape of the carriage. The carriage is made of two separate objects, the carriage and the horse which are connected by some slims objects like wooden beam. This would cause the carriage to easily get stuck on corners. Developers had to create a unique navmesh layout specifically for when an NPC is operating a carriage so that they would take wide enough lines through the corners and avoid getting stuck.

3.6 A*

A* (also sometimes written as A-star) is a graph pathfinding and traversal algorithm that was originally created as pathfinding solution in 1968 for a robot as a part of the Shakey project. It became quickly evident that A* could be used for much more and would be guaranteed to find the optimal path for any problem, as long as certain conditions are met. (Edelkamp et al. 2005)

Nowadays A* is used in a variety of fields due to its efficiency and simplicity, one of the biggest uses being in video game development. A*'s biggest strength and draw is its optimality and efficiency. The way A* traverses a graph is by starting from a specified starting node and trying to find a path to a specific goal node, by minimizing the cost of the path. The cost in this case can be anything like distance or time. In each iteration of a loop that the algorithm runs, it determines which path is should extend by finding which of the next nodes would have the cheapest cost. To be more precise, A* determines the path by selecting the path that returns the lowest cost for $f(n) = g(n) + h(n)$, where n is the next node on the path. $g(n)$ is the cost of travelling from the start node to the node n and $h(n)$ is the estimated cost from the node n to the goal node. Example of the A* algorithm and how it calculates the shortest path from node A to node H, can be seen in Figure 1. In the example, costs between nodes are represented by black numbers and heuristic values of nodes are represented by red numbers. In A* the function $h(n)$ is a heuristic function, meaning that is almost never fully optimal, but works well enough to justify not using resources to fully optimize it and compromise the efficiency of the algorithm. This function's optimality differs from problem to problem.

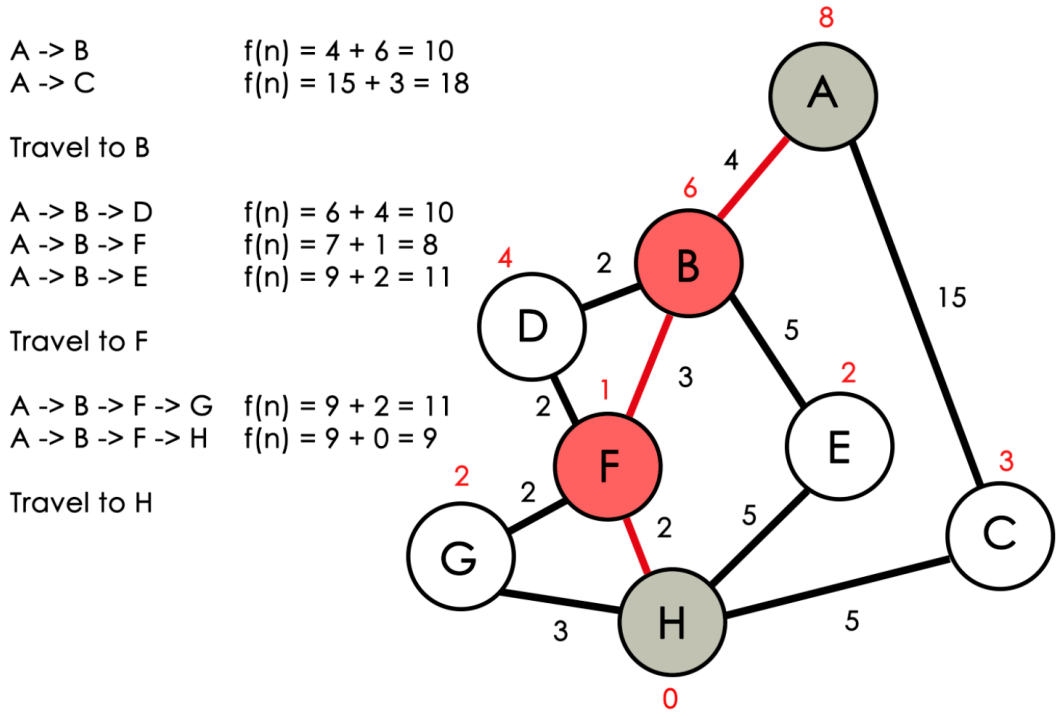

Figure 1: Example of A* finding the shortest path from node A to node H.

A* is favoured in game development due to its efficiency, since pathfinding often needs to happen in real time and there can be hundreds of calculations that need to be done fast and without compromising performance. A* is often used in tandem with Navigation mesh, since navmesh tells the AI what the area around the NPC is like and A* tells the AI how to navigate said area. Navmesh's ability to give certain areas different values can be used together with A*'s cost-based system to give different surfaces multipliers that can be used to give off the illusion that the NPC is actually making a decision.

For example, developers may want to keep their NPC's away from watery surfaces for the main part, but may allow it if A* calculates that, even with water having a high-cost set by the navmesh, it would be faster to cross the water than to go around it.

3.6.1 Heuristic

Heuristic (sometimes referred to as heuristic technique) is a way to help solve a problem by using a method that is considered suboptimal, but close enough as an estimation. Heuristic solution trades optimality and precision for massive gains in speed and efficiency and is useful tool when finding an exact solution is slow, impractical or straight up impossible. While heuristics are employed in many fields such as psychology, law, and philosophy, they are also utilized massively in mathematics and computer science.

In search algorithms, such as A*, heuristic function greatly improves the algorithms efficiency by evaluating each available branch and at first it tries each option at every node. However, it immediately stops exploring branches that are valued as worse options than the current best solution that has been found. This ensures that algorithm like A* always returns the shortest possible path, while improving its speed greatly, as long as the heuristic function $h(n)$ is admissible. Russel (2010) states that an admissible heuristic will never overestimate the cost of reaching the goal. Cost in this case can be distance, time, value, etc.

3.6.2 Overcoming Complicated Paths

Death Stranding (Table 1, row 4) features a complex environment and because of that the navigation mesh made for the area around the player is made of smaller and abnormal shaped polygons. Because of its efficiency, the developers used A* as the pathfinding solution for NPCs to navigate these polygons. But because of their size and odd shapes dues to the environment being very uneven, even the efficiency of A* was not enough in some cases, were the path needed to be reiterated sometimes hundreds or thousands of times over. To save computing power and not compromise the performance of the game, the developers put a 500-iteration cap on their pathfinding AI. This

worked well when the area wasn’t that complex, and the AI was able to accurately find optimal paths even when distance between start and goal would span over entire settlements. To solve this problem developers used something called navmesh bubble. Effectively this just means that to improve performance, navmesh is only generated inside of invisible sphere that is centred on the player or an NPC. This made it so that A* didn’t need to find path over massive distances, but instead would find multiple shorter paths.

4 Conclusion

The objective of this thesis was to seek out and study common ways that video game developers use to create AIs that make NPCs feel like they are making smart decisions and like they belong in the world that they inhabit. To achieve this objective, data was extracted mostly from conferences held by Game Developers Conference, where developers share information about the industry and the projects that they have worked on.

The data gathered in this thesis shows that the four base methods that came up the most commonly were GOAP, HSFM, navigation mesh, and A*. It also showed that these solutions were often used to derive a method to solve some specific issues, and in many cases, there was also overlap, meaning that these solutions were used in conjunction with each other. In this thesis the best example of this would be Death Stranding (Table 1, row 4), in which the developers used navigation mesh to create an invisible corridor for the AI and then used A* to find the optimal path through that corridor. This goes to show that even many in-house systems have their roots in the basics and a good developers should always be familiar with the fundamentals.

In this thesis, it is shown that even though the number of different NPCs is nearly limitless, many aspects of their AI, when broken down, consist of few basic systems that can be modified or combined to suit any developer's needs. Looking ahead, this research could be massively expanded upon by including a larger sample size and diving deeper into the derivations of these methods and studying how these common solutions are modified in order to achieve the needed end product. The video game industry is constantly evolving and even though the methods have taken their place as a standard, there will eventually be a new standard set in the future.

References

Aarseth, E. 2012. *A narrative theory of games.* Association for Computing Machinery. In: Proceedings of the International Conference on the Foundations of Digital Games (FDG '12). Association for Computing Machinery, New York, NY, USA, 129–133.

[Online]

Available at: https://doi.org/10.1145/2282338.228236

[Accessed 10.05.2024]

Bringsjord, S., Govindarajulu, N.S. 2018. *Artificial Intelligence*. Metaphysics Research Lab, Standford University. [Online]

Available at: https://plato.stanford.edu/entries/artificial-intelligence/

[Accessed 19.05.2024]

Edelkamp, S., Jabbar, S., Lluch-Lafuente, A. 2005. *Cost-Algebraic Heuristic Search.* In: AAAI'05: Proceedings of the 20th national conference on Artificial intelligence - Volume 3. AAAI Press. [Online]

Available at: https://cdn.aaai.org/AAAI/2005/AAAI05-216.pdf

[Accessed 16.11.2024]

Hubble, A., Moorin, J., Khuman, A.S., 2021. Artificial Intelligence in FPS Games: NPC Difficulty Effects on Gameplay. In: *Fuzzy Logic*. Springer, Cham. [Online]

Available at: https://doi.org/10.1007/978-3-030-66474-9_11

[Accessed 10.12.2024]

Kaul, V., Enslin, S., Gross, S.A., 2020. *History of artificial intelligence in medicine*. In: Gastrointestinal Endoscopy, Volume 92, Issue 4. [Online]

Available at: https://doi.org/10.1016/j.gie.2020.06.040

[Accessed 03.06.2024]

Michailidis, L., Balaguer-Ballester, E., He, X. 2018. *Flow and Immersion in Video Games: The Aftermath of a Conceptual Challenge.* Frontiers in Psychology. [Online]

Available at: https://doi.org/10.3389/fpsyg.2018.01682

[Accessed 05.12.2024]

Orkin, J. 2006. *Three States and a Plan: The A.I of F.E.A.R.* [Online]

Available at: https://www.gamedevs.org/uploads/three-states-plan-ai-of-fear.pdf

[Accessed 31.10.2024]

Rockwell, A., 2017. *The History of Artificial Intelligence*. [Online]

Available at: https://sitn.hms.harvard.edu/flash/2017/history-artificial-intelligence/

[Accessed 19.05.2024]

Russel, S.J., Norvig, P. 2009. *Artificial Intelligence: A Modern Approach* (3rd ed.). Prentice Hall.

Schuchmann, S., 2019. *History of the Second AI Winter*. [Online]

Available at: https://towardsdatascience.com/history-of-the-second-ai-winter-406f18789d45

[Accessed 29.05.2024]

Turing, A., 1950. *Computing Machinery and Intelligence.* Mind.

Links of Table 1

Row 2: https://www.youtube.com/watch?v=fQb0Iaj50bU

Row 3: https://www.youtube.com/watch?v=LHEcpy4DjNc

Row 4: https://www.youtube.com/watch?v=yqZE5O8VPAU

Row 5: https://www.aiandgames.com/p/the-ai-of-dark-souls

Row 6: https://www.youtube.com/watch?v=LxWq65CZBU8

Row 7: https://www.youtube.com/watch?v=tB88gTpdk48

Row 8: https://www.youtube.com/watch?v=e7AWHT7j3V4

Row 9: https://www.youtube.com/watch?v=wusK-mciCVc

Row 10: https://www.youtube.com/watch?v=VoXSJBVqdek

Row 11: https://www.youtube.com/watch?v=dnGzEn6swqo

Row 12: https://www.youtube.com/watch?v=RFWrKHM0vAg

Row 13: https://www.youtube.com/watch?v=PLchlSvrSpc

Row 14: https://www.aiandgames.com/p/the-ai-of-horizon-zero-dawn

Row 15: https://www.aiandgames.com/p/how-commander-ai-works-in-halo-wars