How to apply AI Technics in Game Development

Diego Tubino Gomes Hartmann, Philipp Altendorf

Design de Jogos e Entretenimento Digital – Univali – Brazil
Escola de Artes, Comunicação e Hospitalidade

dtghartmann@edu.univali.br, philipp.altendorf@edu.univali.br

1. Introduction

The history of humanity and games go hand in hand, going back to our origins themselves. To these days, people seek fun and pleasure in many different activities that can be called games (Huizinga, 1949). Games include simple playful activities such as hide and seek, complex board games such as Go (500 BC), card games, athletic games, and with the advancement of modern technology, video games (1950) are becoming more popular. Although they seem very different, they have similarities that are the basic common elements of games.

A game can be described as a structured system in which the participant or so-called player accepts a predetermined set of rules that apply only in this artificially created space in order to achieve a certain goal. Depending on the game, these rules or game mechanics interact in a complex manner. Crawford (1982) describes one of these common elements as representation.

Games are often played with other people, and this interaction creates an emotional dynamic that is attractive to many players and distinguishes games from other static media or activities. Playing against a human opponent provides a new interpersonal challenge to solve each time. Interaction is the second element of games. To prevent the player from immediately reaching the goal and thus winning, the game needs conflict. This conflict arises from the interaction with another player or obstacles, which can be direct or indirect. The last characteristic described by Crawford (1982) is safety. The game must provide a safe space in which the player can accept a challenge and try different approaches to emerge from the conflict victorious or defeated but without consequences.

Representation, interaction, conflict and security alone do not make a game. Like anything else created by humans, it needs order and planning to make sense. Design is essential to create a coherent product. Design is the act of planning objectively to give a product its logical structure and therefore its form depending on the needs to function (Munari, 1966). Game Design, then, is the task of creating an experience that meets the expectations of the player. Since a meaningful experience cannot be delivered directly, the game designer must create a set of game mechanics that guide the player through the game and ultimately make the experience possible (Schell, 2015).

As mentioned earlier, one characteristic of games is interaction. When it comes to digital games, players do not necessarily interact with other humans, especially at the beginning of the video game era when multiplayer games were not so common. Therefore, the implementation of Artificial Intelligence (AI) systems are fundamental to

video game development. The classical academic view describes AI as a multidisciplinary field with the goal of building intelligent entities. These autonomous and adaptive systems analyze their environment and make decisions based on it. Game AI, however, has a different goal and is not concerned with creating true intelligence. All the different parts that make up a video game, such as graphics, sound, gameplay, and game AI, serve only the main goal of creating a special experience for the player. Therefore, game AI is only about creating the illusion of intelligence maximizing the player's enjoyment and participation during long periods of gambling (Mass and Luc, 2019).

With the increased complexity of modern video games and the high expectations of players, the design and implementation of Artificial Intelligence systems is still a challenge. Several AI methods already exist that are used to solve specific problems in video games. Game designers must understand how they work to use them in the best possible way. In this paper, the most common AI methods used in video games for movement and decision making will be explained and their practical application will be demonstrated with a game.

The article describes the relation of Artificial Intelligence and Digital Games (2), what movement techniques exist for Games (2.1) and popular decision-making techniques (2.2) used by Game AI. The article continues with the proposal of AI methods for a game (3), the development (4) and an evaluation of the results (6).

2. Artificial Intelligence and Digital Games

Because AI encompasses many different disciplines and technologies, it is constantly being redefined as new topics are introduced, and others are classified as non-AI, there is no precise definition of AI. It can be described by their characteristics, which primarily relate to natural language processing, knowledge representation, automated reasoning, machine learning, computer vision, and robotics (Russel and Norvig, 2010).

To successfully process these massive streams of data, AI systems rely on logic, computation, and probability. Algorithms are limited by what is computable, and thus their functionality to produce unambiguous results. Because of the non-computable inputs, algorithms can run endlessly in a loop. Machine learning, deep learning, data science, and robotics are subfields of or related to AI and are influencing the development of AI science.

The development of self-driving cars requires a combination of AI techniques from different fields. Search algorithms must navigate through complex environments and analyze different ways to solve the problem. Depending on time, cost, effort, or other criteria, different search techniques lead to different solutions. In a constantly changing and dynamic environment, decisions must be made quickly and lead to the desired consequences. Computer Vison collects data around the moving vehicle to detect obstacles. All systems must work with high precision and reliability to avoid accidents. Other areas where AI is implemented are Image Recognition (Google and IBM), Speech Recognition (Siri and Alexa) and Translation (GNMT).

When using AI systems in games, the purpose is to give the appearance of intelligent behavior. The best AI system is the one that provides the best player experience. Players often interact with AI systems through Non-Player Characters (NPC), an autonomous agent capable of making decisions. The AI communicates with

the player through animation and audio what their intentions, opinions, or feelings are (Rabin, 2013).

One of the first games which features an AI system was Pac-Man. NPCs (enemy ghosts who follow Pac-Man) seem to consciously conspire against the player, being moved by the AI technique called State Machine (Millington and Funge, 2009). The State Machine consists of a predefined state for - in the case of a game - the NPC, establishing actions (such as, Stopped, Walking and Shooting) that are triggered under certain conditions. For better understanding, Figure 1 illustrates a Lamp State Machine, which has two states: On and Off, which are changed according to the established conditions.

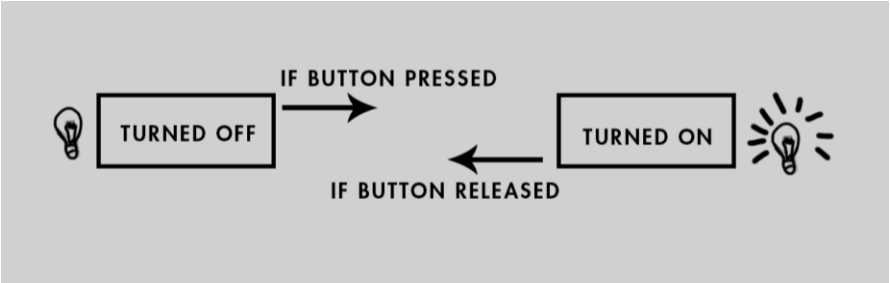

Figure 1. Lamp State Machine.

Figure 2 illustrates a State Machine from one of Pac-Man's NPCs, with Scatter, Hunt and Flee states.

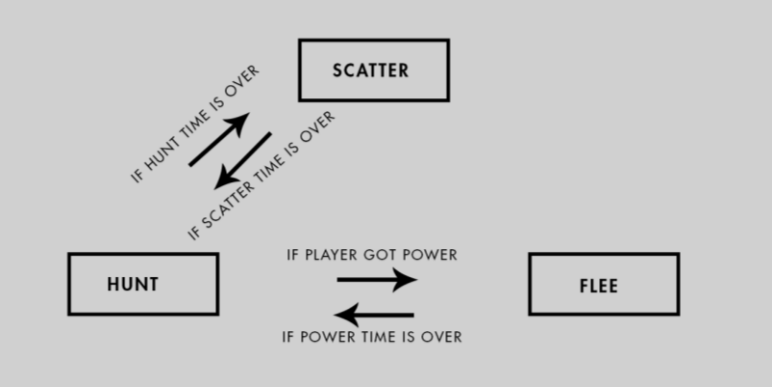

Figure 2. Pac-Man´s state machine.

The First-Person Shooter (FPS) F.E.A.R, released by Monolith Productions in 2005, used communication through audio to create a believable AI system that was new to the gaming industry in this way. The usual gameplay behavior in first-person shooters is based on animations and superficial comments, for example, the player fires at an enemy and the enemy responds with a painful animation and grumbling. When multiple enemies are in combat with the player, it becomes more obvious that they are talking to themselves. Each AI acts individually and says phrases like "Where is he?", "I'm hit!", "I can't see him.". The AI's responses seem out of context or don't explain the situation to the player.

The AI in F.E.A.R. uses a dialogue system to explain mental state or intentions to the player. One enemy may yell "What's your status?" and another may reply "I'm hit" or "I'm fine!". This system successfully communicates to the player that they have hit and injured an enemy and, more importantly, creates the illusion of human-like teamwork. The effective use of language fools (??? engana) the player into thinking that the AI is communicating, thinking, planning, and coordinating attacks against the player (Rabin, 2015).

2.2 Movement techniques for Games

Moving agents in games involves finding paths form one point to another through a (complex) digital world space. Game AI uses pathfinding techniques to find the shortest or quickest route from point A to point B respecting time and distance constraints. One of the first solutions was the Dijkstra algorithm from 1959. Even though pathfinding algorithms have improved over time, complex 3D environments are still a challenge for pathfinding algorithms to find not only the shortest path, but the best path. In order that the Game can use pathfinding solutions, the game level must be represented in a readable data structure.

In a graph, the essential structure of the problem is represented, freed from insignificant side aspects. Figure 3 shows a general graph made of nodes (circles) and connections (lines).

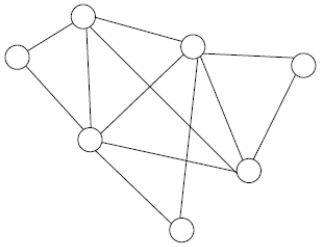

Figure 3. A general graph. Millington, I and Funge, J. (2009).

In game application nodes represents a certain aera, such as a room, hallway, or a defined outdoor space. Connections show the relations between the areas. These connections dictated how to go from node X to node Y. A commonly used type of graph for pathfinding algorithms is called the directed non-negative weighted graph, see Figure 4.

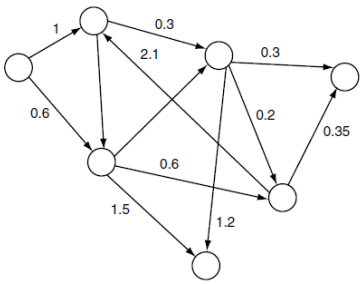

Figure 4. directed non-negative weighted graph. Millington, I and Funge, J. (2009).

The numerical value represents the weight or cost, in games mostly related to time or distance. Each connection has only one possible direction, indicated by the arrowhead. In a game this would be represented through obstacles or level changes which does not allow to go the same path back. In order that pathfinding algorithms can provide solutions they need access to nodes, connections and their corresponding cost and direction (Millington and Funge, 2009).

The performance of a pathfinding algorithm is based on its pathfinding list, the graph, and the heuristic. The heuristic evaluates the cost of the optimal path from node A to node B. Each heuristic is more suited to a specific problem and can reduce costs and time. Straight-line (Euclidean) distance, octile distance, which only allows 45° and 90° angles and Manhattan (city-block) distance must be chosen depending on the given problem (Rabin, 2013).

For game designers, it is important not only to create an interesting opponent, but also a predictable one. A guard will always follow its designed path and the player is rewarded for observing the pattern and identifying the enemy's weakness. Complex enemy movement can be frustrating for players in certain situations. In this cases Waypoints help the designers to create a more pleasing experience. A waypoint is a single position (node) in the virtual game world. Each node is placed in the line of sight of the other node so that a game-controlled character always can reach the next node by using a simple line-of-sight algorithm. Figure 5. Shows a map with connected nodes that build the pre-computed path.

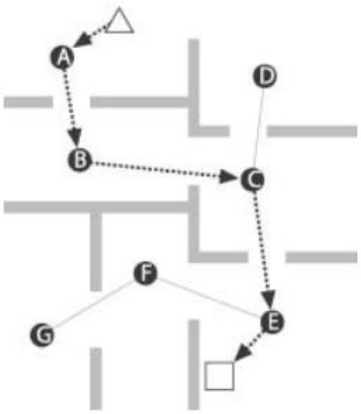

Figure 5. Waypoint based path Millington, I and Funge, J. (2009).

These waypoints can also mark a fixed safe location or positions with tactical features. When the player engages with the enemy, he will move to the closest waypoint to have a strategic benefit. The player will perceive this as intelligent behavior (Bourg and Seeman, 2004).

Most games use an algorithm called A* that calculates a suitable route through the game level so that a character can move to its goal as sensible and rapid as possible. Pathfinding sits in between the movement AI system and decision-making AI system, shown in Figure 6.

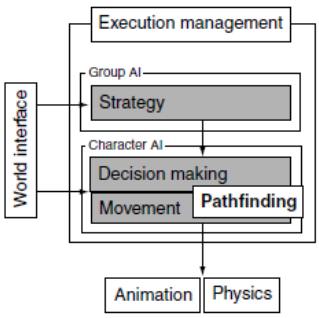

Figure 6. AI model Millington, I and Funge, J. (2009).

The movement AI system is responsible for calculating how the character can move in the Game and the decision-making AI dictates the where to go. A* is a point-to-point pathfinding algorithm which is simple to implement, efficient and offers room for optimization (Millington and Funge 2009). A* chooses the path with the, through a heuristic established, most likely shortest path instead of the lowest cost path. If the quality of the heuristic is accurate, A* performs well.

When characters, animals or other entities must move in cohesive groups rather than independently, steering behaviors are used to simulate a more believable movement. The flocking algorithms presented by Craig Reynolds (1987) introduced a coordinated group movement with the illusion that all individual members have the same purpose. This behavior is achieved through matching the position of the unit with the average position of the target. The unit is aligned to match the orientation of the target and separation behavior avoids collisions with other units (Bourg and Seemann 2004).

2.2 Decision-making techniques for Games

Besides Movement techniques another big field of AI in Games are decision-making techniques. An entity analyzes a data set to execute a desired action. The information needed to generate this action can come from internal and external knowledge. External knowledge provides the AI with information about the game environment around it like the geometry of the level, position of other game characters or changing states in the environment. The internal knowledge is about the internal state or thought process of the character which includes, its health, objectives and performed actions in the past. Based on this knowledge the AI decides what to do what is represented by internal or external changes. What the player perceives are mostly external changes. This process is shown in Figure 7 (Millington and Funge, 2009).

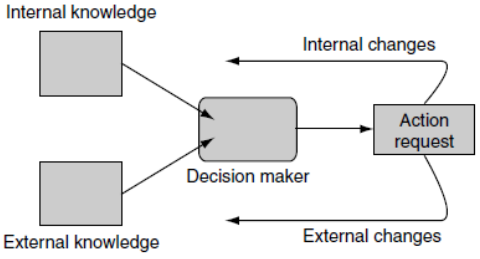

Figure 7. Decision making schematic AI model. Millington, I and Funge, J. (2009).

To create more believable character behaviors that don´t just switch between two distinct states or so-called classical sets e.g. confident or cautious, game developers use Fuzzy Logic to create a gradation in states. Fuzzy Logic adds values to the sets so that a character can be confident with a value of 0.4. The sets are called fuzzy sets and the value is the degree of membership. When something is completely in the fuzzy set the degree of membership receives a value of 1. Something completely outside the fuzzy set on the other hand receives a value o 0. In traditional logic a character can`t be healthy and hurt at the same time because these sets are mutually exclusive. However, this is possible with Fuzzy Logic, through the different degrees of membership for each set. A character can be 0.2 hurt and 0.8 healthy. But it is necessary that the membership degrees sum to 1. An invalid set would be 0.5 hurt and 0.7 healthy. This technique creates more a believable game character behavior since also humans show a large spectrum of in between states in their behavior (Millington and Funge, 2009).

Another approach to AI behaviors is the implementation of machine learning techniques. One of the oldest and widely used are Neural Networks. There are many types of neural networks but the most common used in games are multilayer perceptrons (MLPs). These MLPs are typically made of three layers of nodes. These layers are the input layer, the hidden layer, and the output layer. Each of these nodes is connect and communicates with a layer of nodes before and after. The flow of information in only in one direction, starting form in input to the hidden layer and then to the output. This pattern is the so call topology of the neural network. Every node also has a value, the weight, associated, in a range from zero to one. Figure 8 shows a multilayer perceptron network.

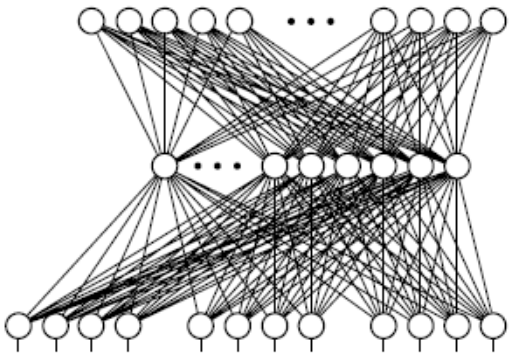

Figure 8. Multilayer perceptrons (MLPs). AI model Millington, I and Funge, J. (2009).

The behavior of an MLP is dictated by the value of its weights and the process of training. The training involves many iterating and the performance is based on the quantity of nodes. Through the training the AI should learn how to respond to a given situation based on the input values. This saves lot of programming time and tweaking of the right values for each situation. But for every change needed in the AI behavior the training process must start over again. Neural Networks have as all tools strengths and weaknesses. MLPs are useful for games with well-defined set of actions or response which don´t require a lot of control through the designers (Millington and Funge, 2009).

Genetic Algorithms belong also in the group learning techniques inspired by nature. They start with a set of solutions, choose the best ones, and evolve them to find an optimal solution for the given problem. Much like the Darwinian principle, survival

of the fittest. The algorithm is divided into 5 steps and starts of by randomly generated states, the so-called population creation. This pool of possible solutions gets evaluated in the fitness function. The more suitable the solution, the higher the score. In the selection phase the two highest scored solutions are chosen for to create the next generation. These act as parents to generate offspring which will naturally inherit the strongest features. The other solutions are dropped. The crossover stage creates the new generation based selected parents. Through random mutation more possible solutions are created. The new population is then used in the next iteration of the algorithm. This evolution process can be time consuming to and does not guarantee an optimal solution (Russel and Norvig, 2010).

Game Engines like Unity or Unreal Engine provide the game developer with a popular AI tool named Behavior Trees. Through their graphical user interface (GUI) they can be used by developers form different areas. Figure 9 shows a behavior tree example in Unreal Engine 4.

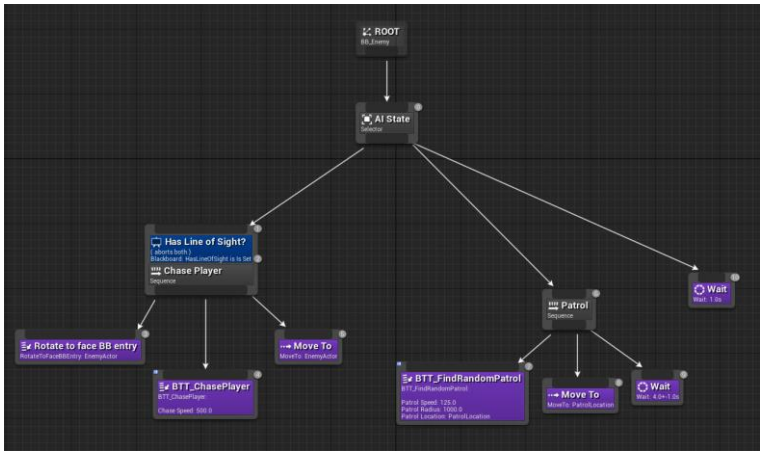

Figure 9. GUI of a Behavior Tree in Unreal Engine 4

Behavior Trees seem like State Machines, but their core function is built on executing tasks. Tasks are divided in Conditions, Actions and Composites. Each task can have a sub-tree and this composability creates complex character behaviors which is one of the reasons that behavior trees are frequently used in modern games. The Condition tasks evaluates properties of the game, like the position of character or gameplay elements or the state of objects and characters. Each task has a limited CPU time to executed and then they return a Boolean value, success, or failure. Based on this evaluation the task executes actions which can be for animations, play audio or change the internal state of the character.

Composite tasks run child behavior and decide depending on the children's return code whether to stop or continue executing the task. Two types of Composite tasks are used in game development. The selector returns immediately a success status code when on of its children runs successfully. In case the children are failing, it will keep trying. Just when there are no children available, it will return a failure code. The Sequence returns immediately a failure status code when one of its children fails. When the children are succeeding it will go on and when there are no children available it returns a success code.

3. Artificial Intelligence Proposal

A 3D top-down game is built in Unity to demonstrate some of the AI methods mentioned in a practical application. The game consists of the player, his companions and enemy drones. The player must free his companions, who are being held captive and guarded by patrolling drones to win the game.

The player can start the game via menu. The player controls his avatar with the W, A, S, D keys on the keyboard and can shoot with left mouse click. The drone's behaviour is controlled by AI methods so that it patrols a predetermined path and follows the player if he enters the drone's field of view. If the drone loses visual contact with the player, the drone searches for the player for a few seconds before resuming its patrolling behaviour. When the player gets hit by the bullets of the enemy drone, his live bar reduces. The player's goal is to free all his companions and he loses if he has no more lives, as well as if any one of his companions die. The companions follow the player when he destroys the lock of the prison. The player returns to the menu after winning or losing.

The game uses a state machine, A* and a flocking algorithm to demonstrate the application of AI methods in games. To manage the different behaviours of the enemy drone the game uses a state machine. The idle state is the Patrol State. The drone follows the determined path on its waypoints and moves from one waypoint to the next. The drone changes its state when visual contact with the player is given and enters the Chase State – which makes the other drones to enter the Search State. Now the drone moves and rotates towards the player and follows his position. If the drone reached a certain distance to the player, the drone opens fire and goes into the Attack state. The player can use his higher speed benefit and environmental objects in the scene to escape the drone. The enemy drone goes in Search state when losing sight if the player. In Search State, it stops pursuing the player and keeps rotating (looking around) for a few seconds on its current position. Then the drone goes in Back to Patrol State and continues form the last waypoint where it left, through the A* pathfinding algorithm – so it does not go through obstacles and walls like a ghost. Once reaching this waypoint, the drone continues its Patrol State. If the player destroys the lock of the space where his companions are held captive, he can free them and they follow the player based on a flocking algorithm, as well as a real time path search (of a A* algorithm) to know how to reach the player avoiding obstacles. Figure 8 shows the basic Game setup and its AI methods.

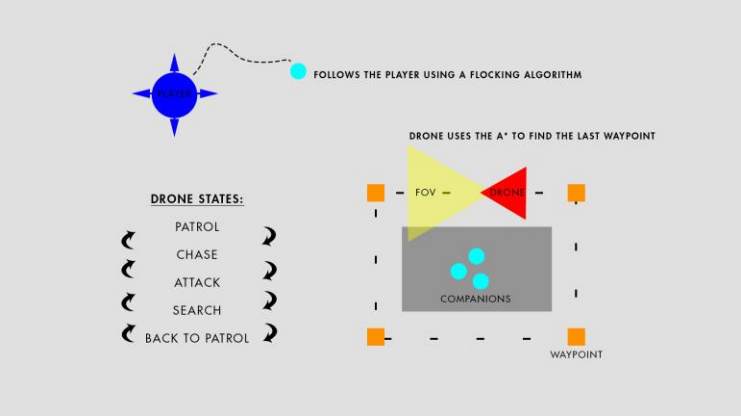

Figure 8. Schematic Game Setup

References

Huizinga, J. (1949). Homo Ludens, Rowohlt, 23rd edition.

Crawford, C. (1982). The Art of Computer Game Design.

Munari, B. (1966). Design as Art. Penguin Group.

Scheel, J (2015). The Art of Game Design, A Book of Lenses. CRC Press, 2nd edition.

Delloite. Artificial Intelligence. [S. l.], 2018. Available in: https://www2.deloitte.com/content/dam/Deloitte/nl/Documents/deloitte-analytics/deloitte-nl-data-analytics-artificial-intelligence-whitepaper-eng.pdf. Access: 17 mar. 2021.

Russel, S and Norvig, P. (2010). Artificial Intelligence, A Modern Approach, Pearson Education, 3rd edition.

Rabin, S. (2013). Game AI Pro: Collected Wisdom of Game AI Professionals. A K Peters/CRC Press, 1st edition.

Rabin, S. (2015). Game AI Pro 2: Collected Wisdom of Game AI Professionals. A K Peters/CRC Press, 1st edition.

Millington, I and Funge, J. (2009). Artificial Intelligence for Games. Morgan Kaufmann Publishers. 2nd edition.

Bourg, D and Seemann, G. (2004) AI for Game Developers: Creating Intelligent Behavior in Games. O'Reilly Media, Inc.

Mass, Laura E. Shummon Maass; Luc, Andy. Artificial Intelligence in Video Games. Available in: https://towardsdatascience.com/artificial-intelligence-in-video-games-3e2566d59c22. Accessed: 16 March 2021.