How to apply Artificial Intelligence 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

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 the common elements of games as representation., interaction, conflict and safety.

Like anything else created by humans, it needs order and planning to make sense. 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.

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. Game designers must understand how AI methods 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 are explained and their practical application is 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).

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. Computer Vison collects data around the moving vehicle to detect obstacles. All systems must work with high precision and reliability to avoid accidents.

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).

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 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 deceives 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 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. In order that the Game can use pathfinding solutions, the game level must be represented in a readable data structure. In games, nodes (circles) represents a certain aera, such as a room, hallway, or a defined outdoor space. Connections (lines) 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.

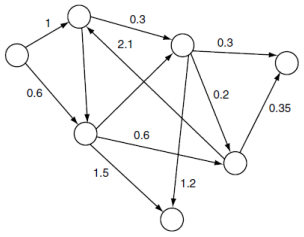


Figure 4. directed non-negative weighted graph. Millington and Funge (2009).

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. (Millington and Funge, 2009).

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. 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.

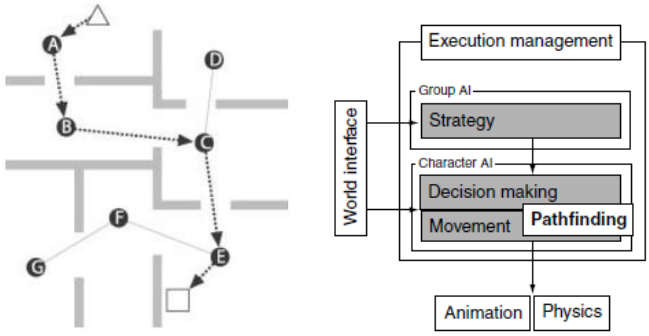


Figure 5. Waypoint based path. Figure 6. AI model. Millington and Funge (2009).

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. 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 and 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. (Millington and Funge 2009)

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 or position of other game characters. 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. This process is shown in Figure 7 (Millington and Funge, 2009).

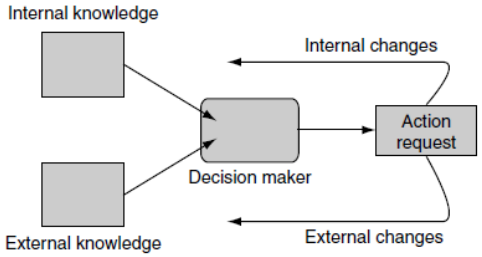


Figure 7. Decision making schematic AI model. Millington and Funge (2009).

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. 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. 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. Through training the AI should learn how to respond to a given situation based on the input values. (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 is chosen for to create the next generation. 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. 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. Composite tasks run child behavior and decide depending on the children's return code whether to stop or continue executing the task.

3. Artificial Intelligence Proposal

A 3D top-down game is built in Unity to demonstrate the practical application of a state machine, A*, path moving and flocking algorithm in game development. The player must free his companions, who are being held captive and guarded by patrolling drones to win the game; it must free one companion per level.

The player controls his/her aircraft with the W, A, S, D keys on the keyboard and can shoot with left mouse click. Right click activates a special attack that spawns wasps which attack the enemy automatically inside a certain attack radius. A state machine manages the different behaviours of the enemy drones. In its idle state, Patrol, it uses a path moving method so that the enemy drone moves along a predetermined path made by waypoints around the companions. It switches to the Chase state when the player enters its field of view and follows its position. If the opponent reached a certain distance to the aircraft, it switches to Attack state.

When visual contact is lost, the enemy drone enters the search state, where it stops at its current position for a few seconds and turns. Then it enters the Return state where it returns to the last waypoint in its patrol area and to find the return path while avoiding obstacles it uses an A* algorithm.

The goal is to free all companions and the game is lost when no more lives are left, as well as if any of the companions die. The companions are set free and start following the player when he/she destroys the lock of the prison. Figure 8 shows the basic Game setup and its AI methods.

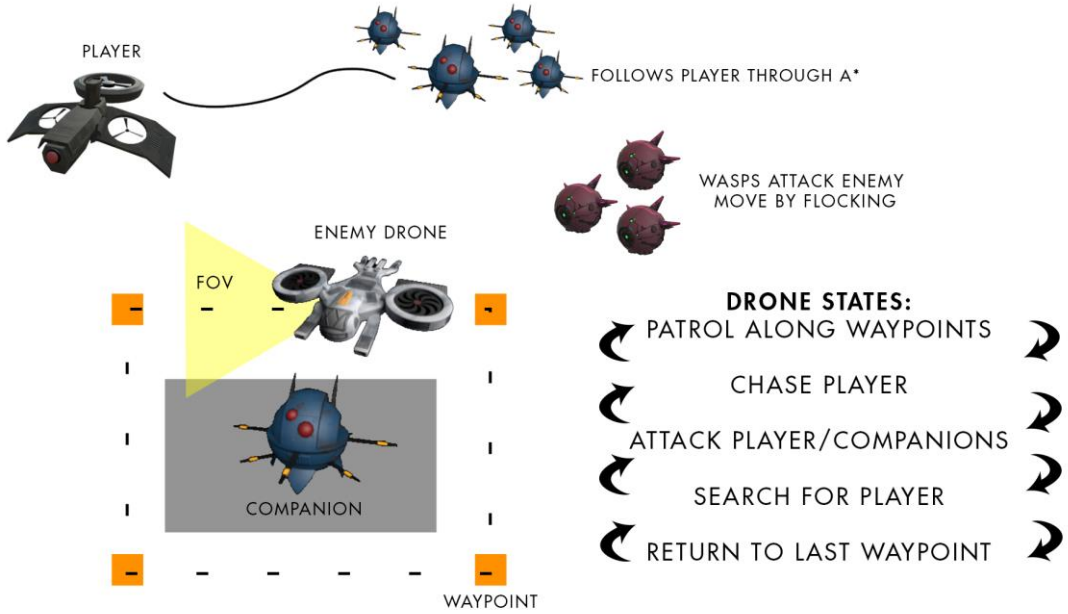


Figure 8. Schematic Game Setup

4. Development

This section describes how the IA techniques were implement in the game to create the gameplay behavior.

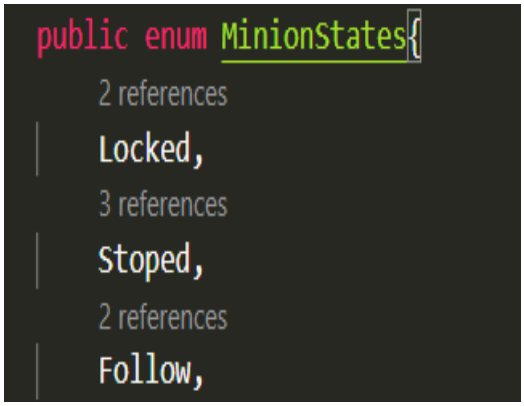
There are three IA agents which use different concepts to achieve their tasks: the Companions (or Minions, first agent), which follow the Player once he sets them free from the cage guarded by the Enemies (second agent), whose goal is to prevent the Player from freeing his Companions, actively chasing them and shooting them down by visual contact. The third agent is a wasp-like group of little individuals used by the Player to attack the Enemy, circling around it. Table 1 shows each agent's behavior system.

IA technique	Companions/Minions	Enemy	Wasps
Final State Machine	Apply states	Change and apply states	NONE
A* Pathfinding	Follows the player	Finds its way back to last patrol point	NONE
Waypoints movement	NONE	Patrols on an established path around the cage	NONE
Flocking	NONE	NONE	Move in a group and cause damage to the enemy that is close.

Table 1. Agents' IA techniques

The Companions use the state machine to switch between different states. Figure 9 shows the possible states for the Minion, like: Locked, Stopped and Follow which are checked to apply its equivalent actions inside the state machine itself, but are set through Unity Events: if they are set free from the cage, their state is changed from Locked to Follow.

The Enemy behavior system (Figure 10) follows the same approach but since their behavior is more complex, their states are both set and checked by its script. Both Enemy's and Minions' states are stored, for better handling, inside an Enum, as a collection of constants values.



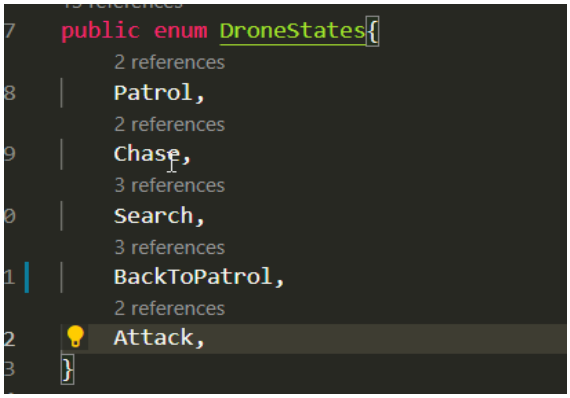


Figure 9. Companion Enum. **Figure 10. Drone Enum**

To switch between different states certain conditions must be set. These conditions are based on specific parameters of the game like distance between object, time constraints, visual contact between agents or location information.

They change dynamical during gameplay and are constantly updated so that states change accordingly. In the case of the Enemy – whose states are set inside its script – there is a first method used to check the Environment, a second one to set the state based on the information gotten from the first one, and a third method that applies the corresponding actions based on the current state. Figure 11 shows this Enemy's state machine logic inside its Update Method (which is called one per frame in Unity Engine).

```
private void Update(){
    LookForTargets();
    SetState();
    ExecuteState();
}
```

Figure 11. Enemy's State Machine Logic.

The Minions' state machine works differently. Although it checks its current state to apply the actions, it handles the changes of states inside Unity Events, that must be called, for example, into the script shown in Figure 13, that has a On Destroyed event (called once health reaches 0 or less). This script, which checks bullets collision, can be used by any object in game. For example, the Player's and Minion's destruction reloads the level. The Enemy's destruction applies a destruction effect on it. But it also is used in the cage locker, that once it is destroyed by the player's bullets, it changes the Minion's state.

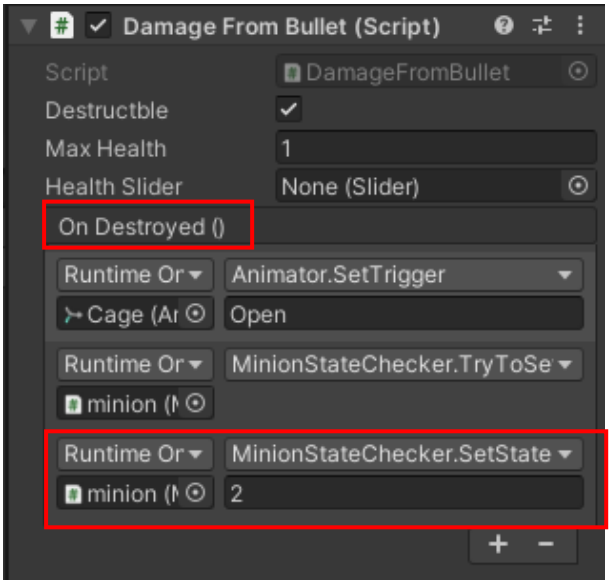


Figure 13. Changing the Minions' state when the locker of the cage is broken.

After the player released the companions, they enter the Follow Machine system – a hierarchical state machine system inside the normal state machine. It checks what

sort of Follow it must execute. Each Follow type has its own conditions to be called, such as distance from player and ground condition.

The Enemy's Patrol behavior uses a Waypoint system which defines an already predetermined path to follow. In Unity, a Waypoint is the position of the Transform Component (that stores not only the position, but also scale and rotation) of an empty GameObject (an Unity data type that stores basic information about an object in the game) which can be placed accordingly to the gameplay needs and adjusted dynamically in their quantity or location.

Each Waypoint has its unique index so that it is clear from which position to start and what direction to follow. The target waypoint is updated to the next one only when the drone reaches x distance from the current target.

When the drone chased the player and lost visual contact to him, it must get back to the last Waypoint based on the saved index to continue its Patrol state. The A* algorithm, used by the Enemy and the Minion, allows the agent to find its way and avoid obstacles on its path. The algorithm creates new Waypoints on the new paths vertices which are followed by a different basic waypoint algorithm (Enemy's return to patrol state and Minion's follow player state).

The Wasps agents are a cloud of small drones moving around inside a certain radius distance. They are spammed by a special attack from the Player, that causes damage to the Enemy inside its radius. Their movement is entirely controlled by the flocking algorithm which calculates the direction and respecting flocking concepts like alignment, cohesion, and separation.

Each rule calculates its own movement direction and sums them up to create the final direction. Alignment causes a particular agent to line up with agents close by. Cohesion alters the individual position so that it corresponds with the average alignment of the other nearby companions within a certain radius. To avoid collisions the agents, use the separation behavior to steer away from its neighbors. Figure 19 shows the composite of these concepts which create the final movement direction.

```
Vector3 move = Vector3.zero;
//passa pelos behaviours (align, cohesion, avoidance)
for (int i = 0; i < behaviors.Length; i++){
    Vector3 partialMove = behaviors[i].CalculateMove(agent, context, flock) * weights[i];
    if (partialMove != Vector3.zero){
        if (partialMove.sqrMagnitude > weights[i] * weights[i]){
            partialMove.Normalize();
            partialMove *= weights[i];
        }
        move += partialMove;
    }
}
return move;
```

Figure 19. Final Flocking Direction

5. Conclusion

All AI concepts could be implemented with success and create an interesting gameplay. Some concepts were devolved following online tutorials, others by us after working on the concepts in the classroom. The biggest challenge was the flocking behaviour since the game is in three-dimensional space the code had to be adapted.

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.