UDC 004.42,94

K. Hazin, I. Stetsenko

LIGHTWEIGHT AGENT-BASED GAME AI ARCHITECTURE

Abstract: The article is devoted to the research game Artificial Intelligence (AI) and architectural solutions for its development. Game AI is one of the most complicated parts of game development, and it needs good tools to reduce complexity and speed up the development. However, there is a lack of lightweight solutions, which can be easily implemented, providing the desired flexibility and reduced complexity. A comparison of game and academic AI is presented, and it is also explained why standard academic AI techniques can't be broadly applied to game development. Considerable attention is paid to examine existing alternatives such as GAIA, SOAR and AI.Implant, their advantages and disadvantages. The proposed solution for a lightweight agent-based game AI architecture is described in detail with examples. In addition, the solution provides a space for improvement and extension, which can be useful for more complicated cases than described in the article.

Keywords: game AI, academic AI, software architecture, agent.

Introduction

Nowadays the gaming industry is the largest category in the entertainment industry. This year, the gaming industry is expected to be worth more than $170 billion in global revenues, five times greater than global movie box office revenues [1]. There is a growing tendency of releasing new games on Steam, which can be seen on the graphs. In 2022 the number reached 12879 releases (Fig.1). That is 35.3 games a day in only one digital store for only personal computers.

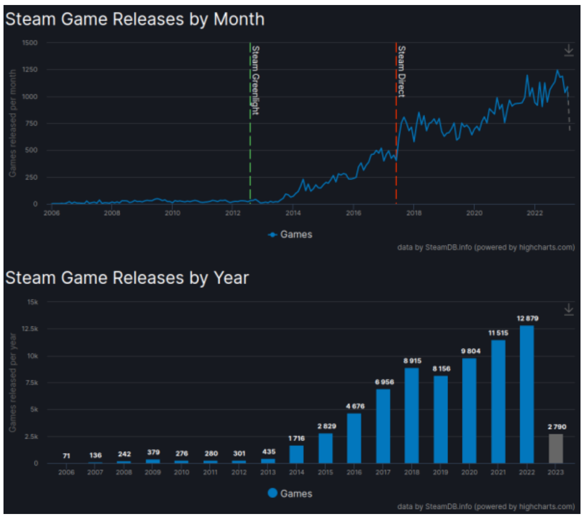

Figure 1. Steam Game Releases

Problem tasks

It is very important to have reliable instruments, robust and flexible code base to iterate more quickly, speed up the development process and reduce the number of bugs in this rapidly developing and high grossing media[3]. Game artificial intelligence (AI) often is one of the most complicated parts of the code base, so by standardizing its architecture and dividing it into smaller more manageable pieces we reduce its complexity and solve part of this problem.

There are many game AI architectures which are either very hard to implement, or provide not enough context to fully solve the problem and focus only on one or two aspects of the SENSE-THINK-ACT AI loop. And this research will try to fill in this niche of lightweight solutions.

In this research only agent-based AI will be considered. It is a bottom up design where each agent has its own AI [4].

Game AI and academic AI

First, we need to set the difference between academic AI and game AI. The goals of game AI differ from the goals of academic AI. The main goal of academic AI is to design and develop optimal agents which will try to reach the most optimal solution for the problem and have large-scale autonomy. Whereas the game AI goal is to entertain, seem intelligent to the player and have a fast iterative process. The illusion of intelligence is more important than the real level of intelligence. Game AI needs some degree of autonomy, to make every interaction feel natural, but also needs some degree of authorial control to direct its actions to deliver the intended player experience [5].

The key metrics of game AI are more subjective because the main goal of games is to deliver desired experience to the end player. It is very subjective and can not be measured by classic AI metrics such as precision, accuracy, F1 score and others. The main metric is player engagement, which can be measured only through manual play testing. And because of this the only way to fine-tune the AI is to iterate over and over until it is engaging and delivers the desired experience, not necessarily fun [6].

That is why the techniques for game AI development differ. Learning algorithms such as Machine learning (ML) and Reinforcement learning (RL) are very rare because they need a lot of data to perform better results and, in most cases, they reach for optimal solutions which are not necessary. They are very hard to tune, too complex for a desired task and take too much time. In game AI development heuristics and ad hoc solutions are very common. Because you do not need optimal solutions, and you need a high performance and high speed of development to iterate and fine-tune the AI. The difference between Academic AI and game AI is represented in Table 1.

Table 1.

The difference between Academic AI and game AI

	Academic AI	**Game AI**
Goals	Optimal and intelligent agents	Entertaining, seemingly intelligent agents which are fast to develop and iterate over
Key metrics	Precision, accuracy, execution speed, objective metrics	Iteration speed, perceived intelligence of agents, player engagement, execution speed, subjective metrics
Common techniques	ML, RL, non-learning algorithms	Heuristics, ad-hoc solutions(hacks), non-learning algorithms

Existing alternatives

GAIA. Game AI Architecture [5, 8]. Written by Kevin Dill and Lockheed Martin Advanced Simulation Center (Fig.2).

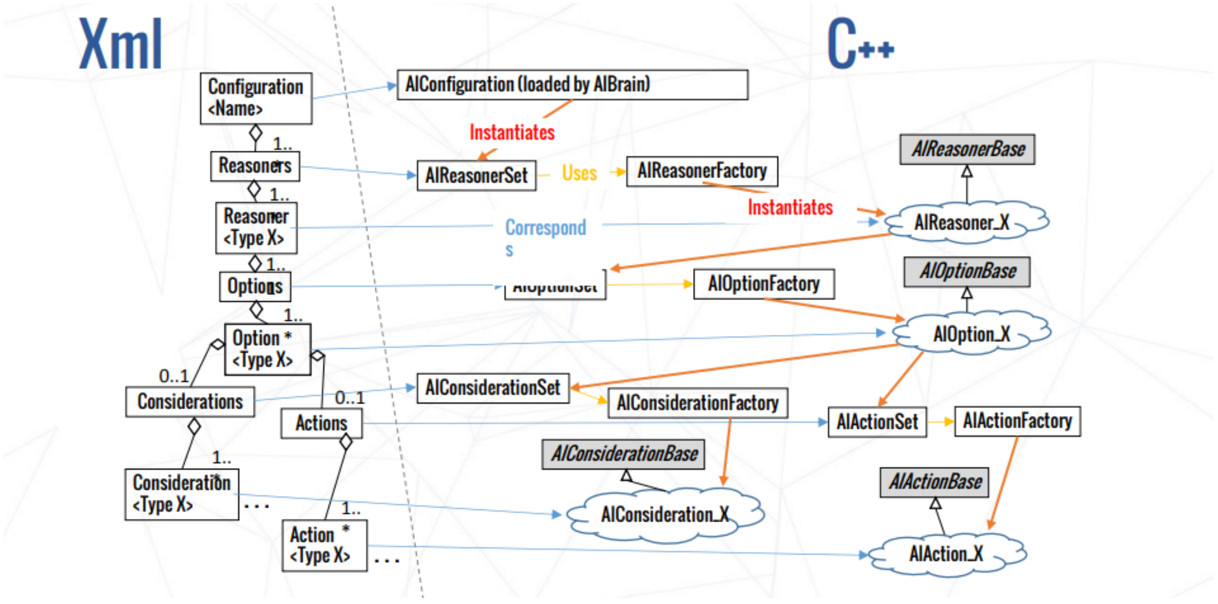

Figure 2. GAIA Architecture

The following advantages could be mentioned:
- very flexible,
- allows to isolate complexity,
- provides great reusability of its components,
- easy to scale,
- data driven,

- makes easier AI configuration,
- integration with different game and simulation engines.

The most noticeable disadvantages are:

- difficulty and time-consuming to implement even the most basic version
- too complicated for small to medium projects with simple agents

In summary, fits best for medium to large projects and medium to big development teams with highly competent staff.

SOAR. Maintained and developed by University of Michigan [9, 10].

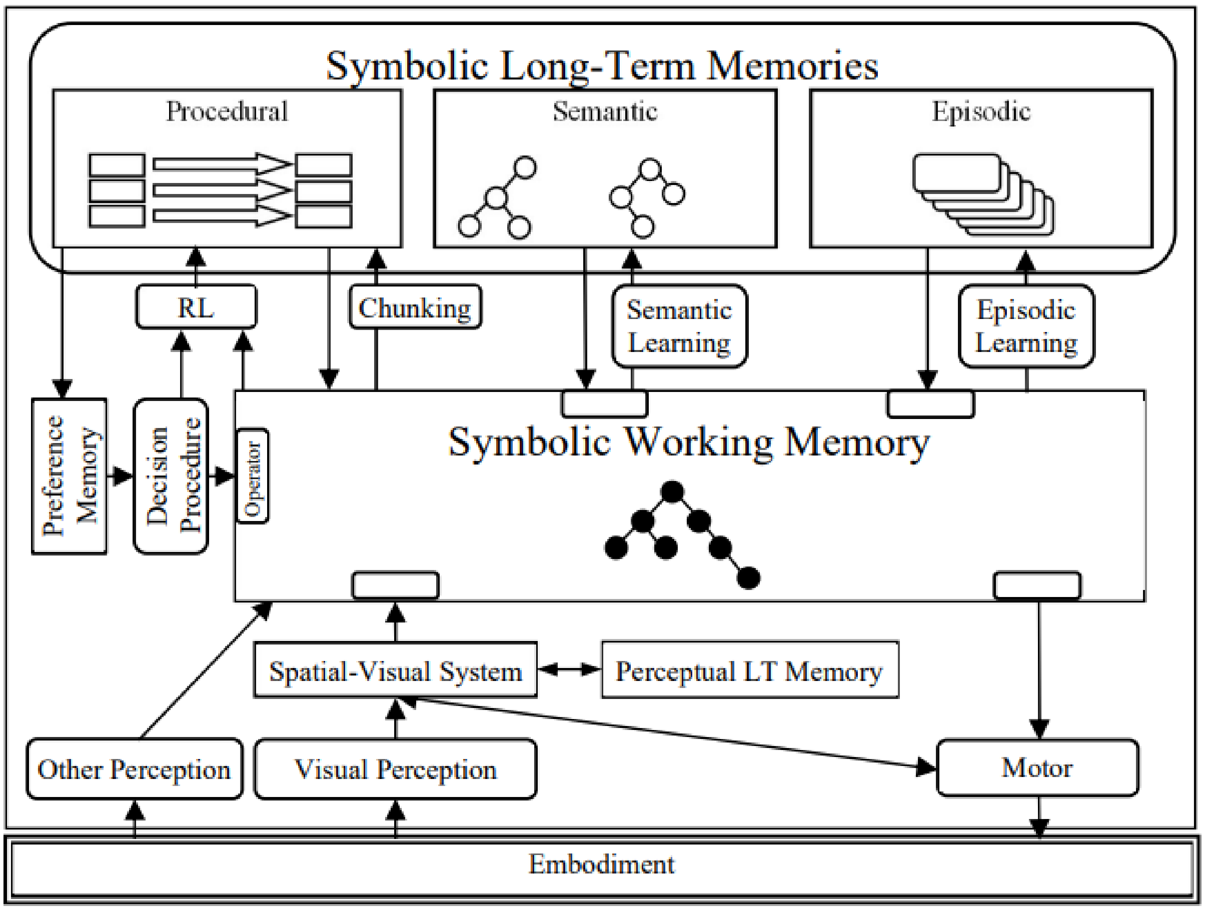

Figure 3. Structure of Soar memories, processing modules, learning modules and their connections

SOAR has such advantages:

- flexible,
- adapts to environment,
- can be used to combine real-time decision-making, planning, natural language understanding, metacognition, theory of mind, mental imagery, and multiple forms of learning,
- frequently updated.

But can nott be broadly used in game development because of these disadvantages:
- enormous complexity
- has learning under the hood
- more of an academic, than game AI

In summary, it is a powerful tool for academic AI, but hard to adapt it to game AI.

AI.Implant. Made and maintained by Presagis [11].

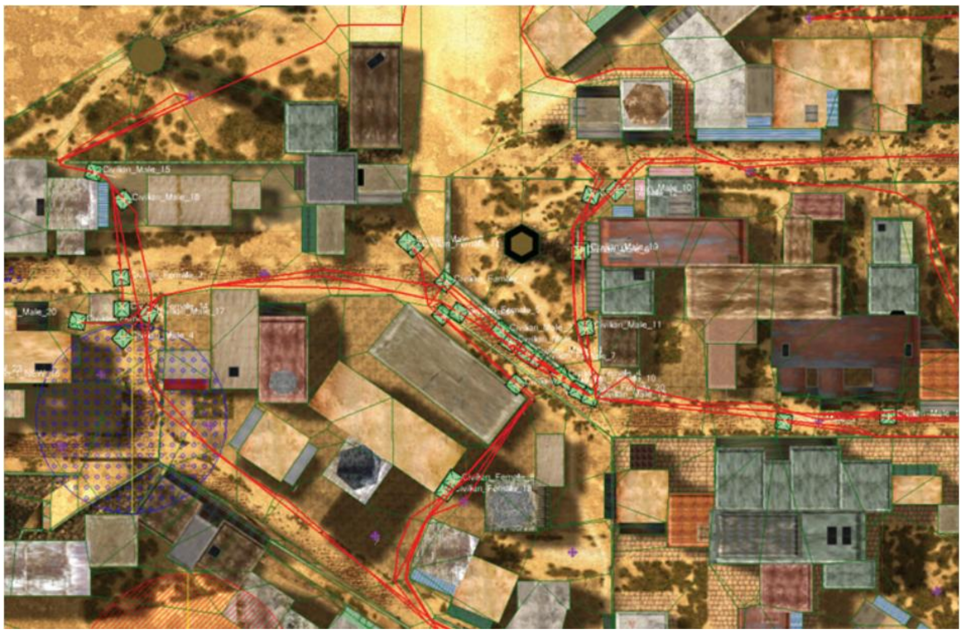

Figure 4. Scheme of AI.Implant

Advantages of AI.Implant are:
- integrations with different engines and platforms
- plugins for development software
- visual tools
- crowd behaviour
- extensible api

The AI.Implant also has some crucial disadvantages:
- outdated and has no updates for several years
- has barely any documentation

In summary, used to be a good tool, but now not lacks support to be competitive.

Agent-based game AI Architecture

Most game engines use the component approach. So the most universal way to implement AI is to make a distinct component for it. We offer an architectural approach of an AI Component consisting of 3 layers: Perception, Decision Making, Action Execution (Fig. 5). They follow the AI principle SENSE-THINK-ACT.

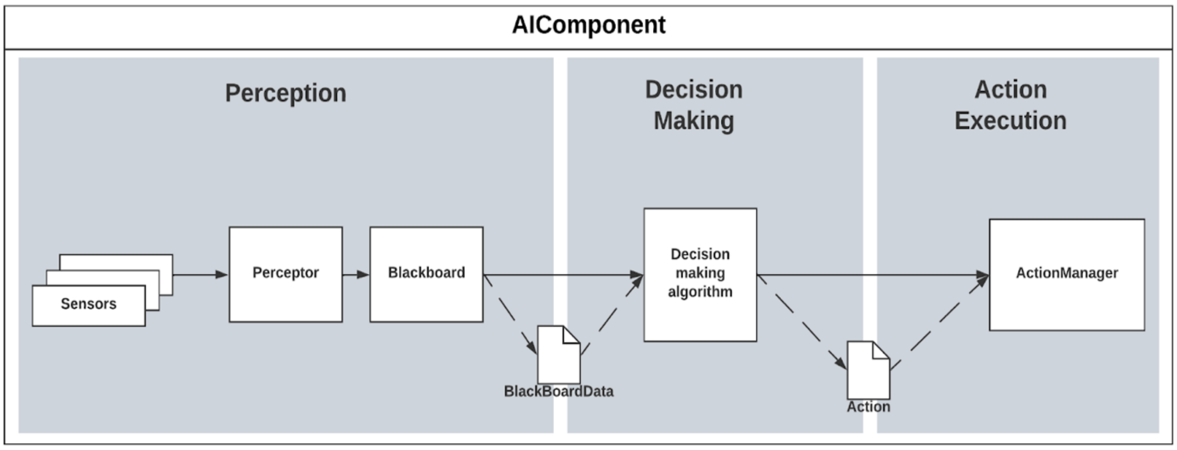

Figure 5. Data flow in AI Component

The first layer, Perception consists of 3 components: Sensors, Perceptor and a Blackboard. Sensors gather information about the world, namely all the external and internal state that influences decision making. Sensors can be direct calculations of the environment or indirect. Direct calculations are raycasts, measuring distance to target, pathfinding, etc. Indirect is looking up to the global Blackboard and retrieve data from there or other available source. Also sensors can be implemented in two ways: polling and events. During polling sensors directly query the environment to get data. Events on the other hand are triggered by changes in the environment.

The next element is optional. With simpler implementations Sensors can write straight to the Blackboard. Perceptor modifies information gathered through sensors, depending on an agent, without need to change sensors themselves and writes results to the Blackboard. So sensors can be universal to all agents, but information gathered from them can be tweaked depending on the agent and circumstances.

Blackboard is a connection between Perceptor and Decision Making to decouple them. Also, it can be used as a memory of our agent if we are no longer getting fresh data from corresponding sensors and Perceptor. If the desired agent is simple, it is enough to use a hashtable under the hood and simply put, read data from it.

However, to use Blackboard as a more complicated system we must wrap our data in a container — BlackboardData, which stores additional metadata about information received from Perception such as time of retrieval, source of information, etc. More complicated Blackboards can be used as a memory system for our agent, emulation of fuzzy information and fuzzy decision-making if we can not be sure of the given information.

Decision-making has just one element – decision-making algorithm. It can vary in complexity: from something simple as a rule-based system with couple if-else to complicated hierarchical reasoning systems combining different decision making algorithms. However, regardless of the implementation the interface must be the same. It must return an Action, which can be later used by an Action Execution layer.

Action is a piece of game logic, which represents the result of a decision process. For Behavior Tree these are leaf nodes, for Goal Oriented Action Planning — actions, for Hierarchical Task Network — tasks, for Finite State Machine — logic for a state and logic for a transition [8]. Actions also can be compound and sequential. Compound action has sub actions that execute simultaneously. Sequential actions are more of a scripted behavior, all sub actions are executed in sequence.

Action execution consists of Action Manager. Action Manager is a scheduler for Actions, which makes Action Execution centralized and easy to manage because all actions, which agent performs are in one place. It runs the active action, handles the queue of next actions and provides feedback to Decision Making: whether the action is active, finished, or something else. It can be implemented using priority queues, interruptive, expiring actions and many other techniques.

Many common techniques for game AI fit in the described architecture: hierarchical reasoning, option stacks, intelligent objects [7] and considerations [5, 8].

Use case

As an example, we can consider two AI agents for a shooter game: a sniper and a regular infantryman. This example will demonstrate every part of an architecture and its flexibility.

Both agents can see and detect enemies (sniper detects enemies faster), cover points, and can be blinded by a flashbang (sniper recovers from a flashbang faster). They can decide what action to do next based on this information (the sniper is smarter): take cover, shoot enemies or just stand still. Let’s break it down into layers we discussed earlier.

Perception Layer. The agents have 2 sensors: a vision sensor to see enemies through raycasts and a well-being sensor which handles buffs, debuffs and health. Sensors are identical. Information from sensors goes to the Perceptor. Both agents can detect enemies if they are too long in the agents' vision area. The threshold detection for a sniper is smaller, so he can detect enemies faster. After the detection, Perceptor writes information about enemies to the Blackboard to decide what to do. If Perceptor recognizes a flashbang debuff from a well-being sensor, for a period of time, Perceptor doesn’t write any information to the Blackboard, which comes from the vision sensor. The period of blindness is shorter for a sniper. When Blackboard stops receiving data from Perceptor, it still holds previous data about enemies and covers. Blackboard data has time stamps for each entry, so can be used as a memory tool.

Decision making layer reads data from Blackboard: enemies, cover points, well-being. An infantryman can have FSM under the hood and just stand still, because he can’t see and he can’t decide what to do. But a smarter sniper can use a Blackboard as a memory. For example he can have a HTN under the hood, and when flashed decide to take the closest cover by memory, which protects from the last known location of enemies. Actions have priority and can be interruptive.

The Action Manager receives actions from the decision-making layer, queues them. For example our sniper was flashed, decided to go to cover, because he can not continue shooting. Going to cover queues up. It is an important action, so it immediately replaces the current action and starts executing.

Conclusion

The proposed agent-based game AI architecture covers a whole SENSE-THINK-ACT loop and helps to decouple code and make it more flexible and reusable. The architecture is also lightweight. It can be easily implemented with little effort, focusing on implementation of game logic. Also, architecture itself can be broadly interpreted because it provides a lot of flexibility with few constraints, yet helps to decouple and organize code. For example GAIA loosely fits in our model of architecture. Sensors are considerations, reasoners are a complicated decision making algorithm. But it also offers a more specific solution for game AI architecture, leaving less space for misinterpretation and bad architectural decisions and heavily enforcing code reusability.

The proposed architecture is a solid foundation of game AI. For a simple AI these loose guidelines would be more than enough to handle code base. For more complex cases it can become a good temporary solution. Complex architectures like GAIA can be built atop our solution. Their high cost implementations can be partitioned in time by gradually transitioning from our non-constrained easy-to-implement solution to more sophisticated and complex. So it can greatly benefit projects of all sizes.

REFERENCES

1. Federal trade commission USA Microsoft/Activision: Administrative Part 3 Complaint (Public) URL: https://www.ftc.gov/system/files/ftc_gov/pdf/ D09412MicrosoftActivisionAdministrativeComplaintPublicVersionFinal.pdf

2. SteamDB service. URL: https://steamdb.info/stats/releases/ (last accessed: 22.03.2023)

3. Schreier J. (2017) Blood, Sweat, and Pixels: The Triumphant, Turbulent Stories Behind How Video Games Are Made. HarperCollins. 304 p.

4. Millington I. (2019) AI for Games. CRC Press; Third Edition. 1030 p.

5. GAIA URL: https://www.sisostds.org/DesktopModules/Bring2mind/DMX/API/Entries/Download?Command=Core_Download&EntryId=35466&PortalId=0&TabId=105 (last accessed: 22.03.2023)

6. Koster R. (2013) Theory of fun for game design. O'Reilly Media; Second edition. 300 p.

7. Game AI Pro URL: http://www.gameaipro.com/GameAIPro/GameAIPro_Chapter05_Structural_Architecture_Common_Tricks_of_the_Trade.pdf (last accessed: 22.03.2023)

8. GDC 2016 AI Summit. Kevin Dill, Christopher Dragert, Troy Humphreys https://www.gdcvault.com/play/1023092/Nuts-and-Bolts-Modular-AI (last accessed: 22.03.2023)

9. Introduction to the Soar Cognitive Architecture. John E. Laird 2022 URL: https://arxiv.org/pdf/2205.03854.pdf (last accessed: 22.03.2023)

10. 10. SOAR home page. University of Michigan https://soar.eecs.umich.edu (last accessed: 22.03.2023)

11. 11.AI.Implant Presagis Brochure URL: https://www.loyola.com/partners/presagis/pdf/2011_04_DS_SIM_AIimplant_web.pdf (last accessed: 22.03.2023)