Developing an Automated Planning Tool for Non-Player Character Behavior

Diego Romero, Mario Sánchez, José Manuel Sierra, Maximiliano Miranda, and Federico Peinado

Departamento de Ingeniería del Software e Inteligencia Artificial
Universidad Complutense de Madrid
c/ Profesor José García Santesmases 9, 28040 Madrid (Spain)
diegorom@ucm.es - marios20@ucm.es - josemsie@ucm.es - m.miranda@ucm.es - email@federicopeinado.com
www.narratech.com

Abstract. Artificial intelligence is one of the fundamental pillars on which the video game development is settled. For this reason, there are tools used in the production of a video game specifically designed for the simulation of intelligence, in order to improve the behavior of non-player characters. However, only few companies have enough resources for dealing with innovation in the field of artificial intelligence. This causes that small companies or independent developers often have to rely on well-known techniques that are available by default in game engines or asset stores. Unfortunately, there is a lack of quality resources related to classic techniques of artificial intelligence. To solve this problem, we have develop a tool for Unreal Engine that allows automated planning for non-player characters in a simple way, using a Goal-Oriented Action Planning architecture. This tool has been developed as a code plugin, allowing it to be easily included in any project, and it has been published as a free asset, to make it more accessible to researchers and developers.

Keywords: Artificial Intelligence · Goal-Oriented Action Planning · Software Engineering · Video Game Tools · Interactive Entertainment

1 Introduction

Since the beginning of the video game industry in the 1970s, the business of video game development, distribution, promotion and sale has been in continuous growth. Video Game Industry has become a great economic engine that generates billions of dollars annually [10]. From its origins, this phenomenon has continued to expand and evolve, limited only by the progress of technological evolution [7].

One of the technological pillars of the progress of the video game industry is the field of Artificial Intelligence (AI) [14]. Thanks to the research and development of techniques to simulate intelligence, it has been possible to achieve credible behaviors in Non-Player Characters (NPCs) [3] and improve player modeling [20], not only for academics but for professional developers as well.

For this reason, a considerable part of the toolkits used in the creation of a video game are focused on the development of AI. However, not all studios have the means to meet the costs associated with such development [18]. Small companies and indie developers have to rely on obtaining standard resources through content stores and game engines, but it is difficult to find innovative resources related to artificial intelligence there.

In this situation, we planned to build a tool that helps to define the behavior of characters in a flexible way that can be used easily. Based on that idea, we define three objectives that serve to specify the purpose of our work.

- Firstly, to identify a current major shortage in the AI tools for NPCs available to *indie* developers. We will investigate the most widely used AI models, and analyze the current availability of resources in the video game market.
- Secondly to develop a tool that meets the needs identified above. We will carry out an appropriate engineering process that includes specification, design, implementation and testing.
- Finally, to solve this deficiency by launching the tool on the market as a free asset. With the feedback received from the community, we will continue to improve it in future iterations.

The rest of the paper is structured as follows: the next section introduces the concepts used in our research and summarizes the related work in the field. Section 3 describes the contribution of our research, and the explanation of the development process of our tool and how it works. Next, Section 4 explains the evaluation with real users and discuss their impressions and thoughts. Finally, we close the paper with some conclusions and future lines of research.

2 Character's Behavior Models

Talking about how to generate intelligent behavior in NPCs, we have focused particularly on the different "planning models" that have been used on the video game industry. For that, we take as reference the planning perspectives according to the classification made by Champandard, so we can differentiate between reactive and automated planning [4].

Reactive planning focuses on "how to do something". This type of planning relies on the implementation of a decision system based on explaining the behavior that the character must perform depending on the stimulus he receives from the environment [1]. The most used decision-making techniques for this planning are Finite State Machines (FSMs) and Behavior Trees (BTs).

FSM is one of the oldest decision-making techniques [2], and video games as famous as Pac-Manor Half Lifeuse it. Its design is very simple and intuitive, but its complexity increases greatly when trying to implement more advanced behaviors [5].

On the other hand, BTs emerged as a mix between hierarchical FSM and hierarchical task network planners and became an evolutionary advance to alleviate the weaknesses of the FSM. One of the first video games to use this model

was Halo 2 , and since then it has become the most expanded decision-making technique in the game industry.

Automated planning focuses on "what the character can do" [6], and in contrast to reactive planning, the solution to an artificial intelligence problem is obtained at runtime, that is, when the transitions between actions are dynamically established [19].

Goal-Oriented Action Planning (GOAP) is the most representative model of automated planning [11], since it was first used in the F.E.A.R. videogame The GOAP model is based on the dynamic management of the set of available actions of a character in order to achieve specific objectives [13]. This greatly facilitates the changes that can be made to the behavior of the characters, since there is no need to implement transitions between actions as in reactive planning [9].

Regarding the existence of tools for using GOAP for building NPCs behaviors, we have analyzed the ones related to character behavior planning that are available for the two most used game engines in the industry: Unity [1] and Unreal [2], and more specifically in their official resource stores.

Although there are a lot of resources related to reactive planning in both stores, there is a shortage of quality automated planning tools, especially if we focus on the GOAP architecture. In the Unity Asset Store[3], we found only two tools that use GOAP. While in the Unreal Marketplace[4], the situation is even worse, having found just one tool that mentions the use of GOAP, but not really applying GOAP, but using some GOAP-inspired logic for BTs[5].

This is in contrast to the number of recent and successful games that use GOAP, such as *Fallout 3*, *Deux Ex: Human Revolution*, *Tomb Raider* or *Middle-Earth: Shadow of Mordor*.

3 Automated Planning for Unreal Engine

After identifying the lack of AI tools related to planning models in the game engines markets, we proceeded to carry out the development of our tool following an appropriate engineering process.

3.1 Planning Perspectives

When comparing both planning perspectives, it is important to mention that there is no perfect planning model. The choice between one model or another will largely depend on the specific needs of each video game in relation to the

[1] Unity; Unity Technologies; https://unity.com/es
[2] Unreal Engine; Epic Games; https://www.unrealengine.com/en-US/
[3] https://assetstore.unity.com/
[4] https://www.unrealengine.com/marketplace/
[5] https://www.unrealengine.com/marketplace/en-US/product/visai-an-advanced-modular-ai-system

behavior of its characters. However, there are certain factors that allow us to differentiate one way of planning from another.

In reactive planning, as transitions between actions are predefined, we have a better control of the expected result of the character behavior, but also it is more difficult to make changes to already created plans. In the case of automated planning, by not creating the plan until runtime, it becomes more difficult to control the result, but the problem of expanding or modifying the character behavior is almost completely avoided [12].

Also, automated planning gives us more realism about how the character deals with a problem [8]. The AI of reactive planning is limited to following a predefined script [15], while in automated planning the character has more freedom and it may seem independent [17].

3.2 Development Process

Before the designing process of the tool, we made a prototype that would help us to become familiar with the development in Unreal Engine. In this prototype we tried to perform a simple implementation of an AI that controlled a character to carry out a series of actions.

Then we designed the tool based on the class structure proposed by Jeff Orkin [11], adapting it to the needs of the Unreal Engine environment. We carried out the implementation of the system taking into account that the objective of the first iteration was to meet the functional requirements. That is, we had get the tool to actually carry out automated planning under a GOAP model of character behavior.

Once this iteration was completed, we went on to perform the modular adaptation of the tool to make it easy to use and applicable to any Unreal Engine project. For this, we prepared certain functionalities so that they could be inherited through Blueprints, which allows the user to interact directly with the classes and methods that we had developed.

The modularity of the tool meant having to convert it into a plugin. These types of tools can be included into the engine as extensions that can be easily enabled or disabled without installation. Thus, the classes and methods developed, are perfectly integrated with the ones of the environment.

Furthermore the contents of the tool can be edited in a visual and simple way, facilitating its use to any type of developer and in any type of project.

3.3 Software Architecture

The implementation of our tool consist on five C ++ classes and an auxiliary struct. Below we detail the most relevant functionality of each of these classes.

Action *Action* is the class that contains the attributes and functionality of an action. Each element of type *Action* has an attribute *name* that allows it to be identified and distinguished from the rest.

As we have already mentioned, this class includes as attributes the list of preconditions that must be met in the current world in order to carry out the action itself. In the same way, the class also includes the list of effects it causes on the current world when the action is completed. Both preconditions and effects are expressed through attributes of type *WorldState*.

On the other hand, the class carries an attribute that indicates the type of objective of the action. This attribute is used to define the type of the actor on which the action will be performed. The objective of this attribute is to avoid the duplication of actions that perform the same behavior on different actors.

This class is *Blueprintable*, which means that from the editor of Unreal Engine developers can generate *Blueprints* that inherit from it. Also, it has two functions (*doAction* and *checkProceduralPrecondition*) that are implemented directly through *Blueprints* by the developer.

WorldState *WorldState* represents the state of the world and is composed of *atoms*, which are predicates which represent the characteristics that define the world. Predicates are key-value pairs of type *String* and *Boolean*, which are stored in a list of type *map*.

This class includes methods that facilitate the checks between different elements of type *WorldState*, as well as a method to add and modify the state of the world applying the logic of the GOAP model.

Planner *Planner* is the core of GOAP as it contains and manages the logic of the action planner. This class receives a list of actions of type *Action*, as well as the states of the initial and meta world of type *WorldState*.

Planner implements the necessary methods to be able to generate the least expensive action plan using the A* algorithm in a GOAP architecture framework. Using this class, the node trees are generated with the solutions to planning problems, in which each node is represented by the *Node* class.

The initial node is the current state of the world, while the final node is the desired state of the world. The intermediate nodes are states of possible worlds, while the edges are the available actions.

As support elements in the development of the A* algorithm, this class contains two lists of nodes: *openList* and *closedList*. In the open list are nodes that can be accessed, but have not yet been explored; while the closed list contains the nodes that have already been visited.

Node *Node* is a helper class to represent the nodes within the scheduler algorithm A*. As a representative of a node, it contains the state of the current

world of type *WorldState*, as well as the action of type *Action* that has been performed to reach it.

This class also contains the information of the cost *G*, which is the accumulated from the initial node to reaching it , and the cost *H*, which is the heuristic cost, equivalent to the number of different predicates between the state of the world of the current node and that of the final node. Both costs added together give the value of *F*, which is the evaluation function, by which they will be selected when they are in the open list, in increasing order.

Controller *Controller* inherits from *AIController* class, and is in charge of managing the available actions, the state of the current world and the state of the target world, as well as the calls to the scheduler. *Controller* represents the AI of the agent, since this class is responsible for decision-making, based on the information it has.

This is also a *Blueprintable* class and, as with *Action*, it is prepared for the developer to generate *Blueprints* that inherit from it. This allows the developer to adapt the AI to his liking according to the needs of each project.

4 Experimental Validation

We performed a series of tests with 16 real users in order to evaluate the tool and to receive some feedback about their experience with it. We prepared a questionnaire that consisted of conducting a guided test of our code plugin. In this test, the respondent was asked to develop an environment in which it was necessary to create an AI planner to solve a specific problem.

After completing the test, the respondent was asked to answer a series of questions related to the test. Taking advantage of the responses received, we have been able to carry out an analysis on positive and negative aspects of our tool that helps us to improve the quality of the product, in view of the official publication in the Unreal Marketplace.

One of the most outstanding conclusions we have drawn from the questionnaire is that, although most respondents were unaware of the existences of GOAP, more than 75% of respondents have found our tool to be “easy” to use, and more than 60% consider that there is “enough documentation” for it. This allow us to be reasonably optimistic with the objective of making an application adapted to any type of developer, not only to the most experienced and knowledgeable of automated planning techniques.

In addition, 7 out of 10 respondents were confident that they would use the code plugin in their Unreal Engine projects, and 9 out of 10 “will recommend” the plugin to their colleagues, pointing to a very positive marked acceptance.

The constant evolution of the video game industry requires continuous adaptation to new technologies and emerging needs. This fact means that any tool available in a resource store has to be maintained over time in order to remain

useful and efficient for developers, e.g. in order to adapt it to future versions of Unreal Engine.

Indeed, thanks to the evaluations and suggestions made by users about our application, we have obtained information about possible improvements that could be carried out. These improvements include the development of other algorithmic models in the planning heuristics. Although the GOAP architecture is based on the use of the A* algorithm, it is true that the tool could be extended to allow the use of other search algorithms, or different heuristics.

Another extension that could be carried out in the future is the integration of the Environment Query System (EQS) in our tool. This system is a complement to Unreal Engine that is in charge of collecting information from the environment in order to facilitate decision-making in specific situations proposed by the developer. Currently, our tool can be used in conjunction with EQS, but would require the user to be responsible for implementing communication between one system and another. Thus, the extension that could be carried out would be to directly integrate the use of EQS into the logic of the scheduler.

5 Conclusions

In the research we carried out on the evolution of video game AI and the current state of it, we concluded that there were great differences between some planning models and others. Based on these differences, we compared planning perspectives evaluating the strengths and weaknesses of each model. We also carry out an analysis of the current state of the video game market regarding AI tools for video game development by small studios or independent developers. This revision allowed us to conclude that there was an evident lack of AI resources for indie video game developers, which was especially noticeable in the Unreal Marketplace, where there was no AI tool that used the well-known GOAP architecture for automated planning of NPCs behavior.

Thus, we set ourselves the objective of developing a tool that would cover this need. After making a prototype in which we tested the Unreal Engine functionalities related to NPCs behavior, we carried out the design of our automated planning tool under a GOAP architecture. The tool was implemented in C++, and in this first approach, the goal was to meet the functionality of the application, that is, to ensure that it was actually planning automatically at runtime.

After developing the tool, we tested the practical aspect of the application. This meant that we had to make the tool easily accessible by any developer and addable to any Unreal Engine project. For this reason, we carried out a second implementation of the tool as a code plugin. This type of implementation allowed us to develop the functionalities of the application in native code, but preparing certain classes to be inherited through Blueprints. In this way, any user could adapt the GOAP architecture created to the specific needs of their project. Full technical details of this work can be found in [16].

Finally, after having carried out an experimental validation with users, we published the tool in the Unreal Marketplace[6] as a free asset (with more than 53,000 downloads during the first trimester), marking the final milestone of the project, since we had managed to meet all the objectives that we established initially. In addition, thanks to the positive feedback received by our users, we are verifying that the tool is fulfilling its purpose.

References

1. Brom, C.: Hierarchical reactive planning: Where is its limit? (2005)
2. Buttice, C.: Finite State Machine: How It Has Affected Your Gaming For Over 40 Years (2019)
3. Carryer, S.: The Brains in Games: Video Game AI (2019), https://towardsdatascience.com/the-brains-in-games-video-game-ai-d0f601ccdf46
4. Champandard, A.J.: AI Game Development: Synthetic Creatures with Learning and Reactive Behaviors. New Riders (2003)
5. Champandard, A.J.: 10 reasons the age of finite state machines is over (2007), http://aigamedev.com/open/article/fsm-age-is-over/, [Online; accessed 17-July-2019]
6. Ghallab, M., Nau, D., Traverso, P.: Automated Planning and Acting. Cambridge University Press (2016)
7. Holdings, I.T.: 2019 GDC State of the Game Industry (2019)
8. Horti, S.: Why F.E.A.R.'s AI is still the best in first-person shooters (2017)
9. Long, E.: Enhanced NPC Behaviour using Goal Oriented Action Planning (University of Abertay Dundee). Master's thesis (2007)
10. Naramura, Y.: Peak Video Game? Top Analyst Sees Industry Slumping in 2019 (2019)
11. Orkin, J.: Symbolic Representation of Game World State: Toward Real-Time Planning in Games (2004)
12. Orkin, J.: Agent Architecture Considerations for Real-Time Planning in Games (2005)
13. Orkin, J.: Three States and a Plan: The A.I. of F.E.A.R. (2006)
14. Pascual, J.A.: Así está cambiando los videojuegos la inteligencia artificial (2019)
15. Rasmussen, J.: Are Behavior Trees a Thing of the Past? (2016)
16. Romero, D., Sánchez, M., Sierra, J.M., Peinado, F.: Automatic Planning for Video Game Characters Behavior as Unreal Engine Plugin (Final Degree Project, UCM) (2020)
17. Statt, N.: How Artificial Intelligence will revolutionize the way videogames are developed and played (2019)
18. Toftedahl, M., Engström, H.: A Taxonomy of Game Engines and the Tools that Drive the Industry (2019)
19. Vassos, S.: Introduction to STRIPS Planning and Applications in Video-games (2012)
20. Yannakakis, G.N., Maragoudakis, M.: Player modeling impact on player's entertainment in computer games. In: User Modeling 2005, 10th International Conference, UM 2005, Edinburgh, Scotland, UK, July 24-29, 2005, Proceedings (2005)

[6] GOAP NPC, https://www.unrealengine.com/marketplace/en-US/product/goap-npc-goal-oriented-action-planning-for-non-player-characters