Proceedings of the Seventeenth AAAI Conference on Artificial Intelligence and Interactive Digital Entertainment
(AIIDE 2021)

# Multi-Agent Cooperation in Games with Goal Oriented Action Planner: Use Case in WONDER Prototype Project

**Gautier Boeda**
Advanced Technology Division
Square Enix Co., Ltd., Tokyo, Japan
boedagau@square-enix.com

### Abstract

Multi-agent cooperation systems often rely on an external commander-like entity that will plan for all the agents it manages. When it comes to games development, this entity can be hidden from the player, which makes the player believe the characters are actually taking their own decisions. However, it has limitations. We believe that to achieve truly believable character-interactions between non-playable characters, real(non-scripted) communication between the agents is the key. We will introduce our multi-agent cooperation system where each AI-agent thinks for itself and communicates with the other agents to cooperate and achieves complex goals. The system will be demonstrated with a use case in our WONDER prototype project.

## Introduction

As of today, multi-agent cooperation systems often rely on an external commander-like entity that will plan for all the agents it manages. Outside of the game industry, this system is often used to organize secretary tasks on a network(Cao, Bian, and Hartvigsen 1997) by planning and dispatching the actions to the agents. When it comes to the game development industry, this commander-like entity(Gehlin 2014), also named AI Director since the development of Left 4 Dead (Booth 2005), does not need to be visible to the player to exist. If it is invisible, we can compare it to a god-like entity watching and controlling its entities, and if it is visible, to a coach coaching its team of people. While the goal of this commander-like entity is to make smarter and believable AI-driven agents, the player can feel cheated or can think these agents are unnatural as they are using knowledge they could not have gotten by themselves. It is particularly true when it comes to achieve goals requiring more than one agent. These knowledge come from a 3rd-person entity that is able to watch everything from above and gives appropriate orders to the entities it manages. Following our research on character interaction in games(Boeda 2019b,a, 2018), we believe that to achieve truly believable and lively non-playable characters, real communications between the agents is the key to share knowledge and organize themselves naturally and smartly. In these research, we highlighted progress in character interaction around emotion and mood expression, speech recognition, precise location base information understanding and independent behaviors. However, some of these interactions still felt too one sided (controller - responder) due to a lack of communication, dialogue from the responder despite coherent emotional reactions.



Figure 1: WONDER prototype project example: Cooperation scenario where the character has to ask for help to escape from the prison

## Method

Bringing real(non-scripted) communications between the agents would make the game world feel more alive and emergent behaviors could happen thanks to the different characteristics and state of mind the different characters. In order to achieve such experiences, we designed a multi-agent cooperation system based on the Goal Oriented Action Planning (GOAP)(Orkin 2005) and a simple messaging system similar to the multi agent planning language (Brenner 2003) but adapted to our experience.

### Cooperation-Based GOAP

**Decision-Making Flow** The decision making system of our AI-driven characters is composed of two essentials pieces. The first one being the goal manager which is used to select an appropriate goal to achieve at any given time. Once a goal has been selected, our 2nd essential piece comes into place: our GOAP planner will try to find a suitable plan to achieve this goal. This plan will be composed of GOAP

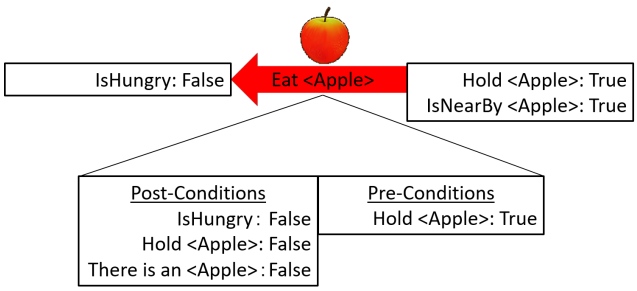


Figure 2: GOAP Planner: GOAP action effect from node game state

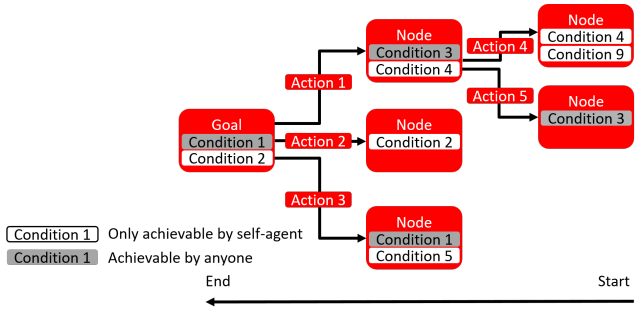


Figure 3: Example of GOAP Planning with condition-tagging highlighted

actions which will be translated into gameplay actions before the plan is executed by the agent. A gameplay action is composed of an action identifier, its execution code, and necessary information about objects, characters, locations it needs to interact with during the execution. In comparison, a GOAP action does not have an execution code, it is only composed of the necessary information for the GOAP planner: the action identifier, a way to compute the cost of the action (the higher the cost, the longer or more difficult the action is to execute), a list of pre-conditions, a list of post-conditions and a list of variables. A pre-condition is a condition that needs to be valid on the preceding node game state to allow this action to be executed from this game state. A post-condition is a change on the game state this action brings if executed. An example of a GOAP action applied to a game state is shown in figure 2. Variables are dynamic parameters that can be set to conditions to specify them. For instance, `Hold <Apple>: True` seen in figure 2 has a variable `<Apple>` applied to the condition `Hold`. A game-state is a list of conditions representing the state of the game-world based on the knowledge of the AI-driven character. As the AI-driven characters do not share knowledge to each other, they can only know what they see or hear during the play. As such, when one character will try to find a plan using GOAP for a said-objective that requires multiple agents to achieve, this agent cannot plan for the other agents fully. It can only determine where in a partial plan another agent can help. The two following sections will detail this process in details.

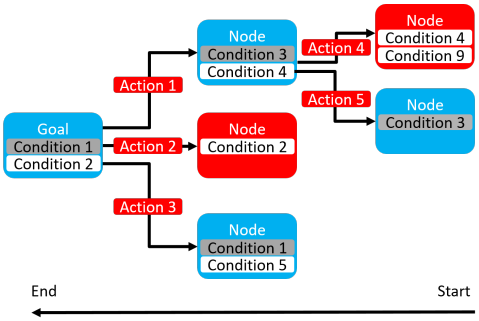


Figure 4: Detection of node game-state for cooperation (highlighted in blue)

**Condition-Tagging** First, an agent has to understand it needs to cooperate with another agent to achieve a specific goal. It is done thanks to two key features of our system. The first being condition-tagging. As said earlier, in GOAP planning, to go from a one game state to another, we use GOAP actions which have pre and post conditions. In our experience, our GOAP planner is a backward planner (from goal game state to initial game state), the post-conditions needs to satisfy the goal game state in order to validate this action. Then we apply the pre-conditions to the goal game state to create the following node game state. To distinguish conditions that can be validated by any agent to the conditions that can only be validated by the agent executing the GOAP planning, conditions can be tagged with a `achievable_by_any_agent` flag. For instance, the condition `DOOR_IS_OPENED:true` will be tagged with this flag as this condition could be satisfied by any AI-driven character living in this game world. However, `IS_HUNGRY:false` is a condition that depends on the agent itself. The only agent that can execute an action validating this condition can only be the agent executing the GOAP planning. An example of GOAP execution with condition-tagging is shown in figure 3. Thanks to these conditions-tagging, the agent executing the GOAP planner can now understand from which node game state it is possible to ask another agent to find a solution if the agent itself is not able to find a complete plan. The second key feature will explain this detection process.

**Detect Node World-State for Cooperation** The second key feature of our cooperation-based GOAP system is to detect node game state that could be achieved by other agent through cooperation. To be cooperation-friendly, these nodes game state needs to have at least one condition achievable by other characters. The detection is done as follows: after the GOAP planner execution, if no plan was found, the cooperation-based GOAP planner will create a list of the cooperable node game states by going through the whole GOAP tree search. From the example we have in figure 3, the algorithm will find the node highlighted in blue in figure 4.

The next step is to validate one condition achievable by another character by adding a ”cooperation action” and see if it can then find a plan from each of the cooperable node

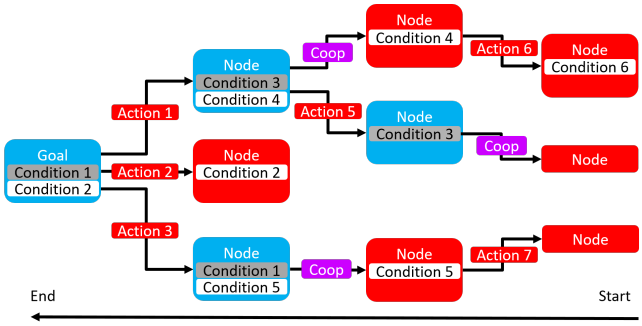


Figure 5: Planning after adding a cooperation action to the cooperable node

game state. The meaning of this method is as follows: if we can find an agent that can do the said-condition, can we find a plan that satisfy the remaining conditions? At the end of this search problem, the cooperation-based GOAP planning will have a list of valid plans that will have at least one cooperation action, as shown in figure 5. Thanks to this cooperation-based GOAP system, our agents are now able to find plans of actions that may rely on cooperation with other agents. However, even if we found valid plans, it does not mean we will be able to find an agent that can achieve the condition achievable by another character we validated with a cooperation action in each of these plans.

## Messaging Component

As shown previously, this GOAP cooperation system alone is not enough to allow cooperation to happen. The AI-driven characters still need a communication system to find characters, request cooperation, and adapt their plan depending on the reply they will get. To achieve this behavior, each character has a messaging component. When a character tries to execute a plan that contains a cooperation action, it will try to find other agents to discuss with. As each agent has a perception system that keeps track in the memory of all objects, characters, locations it has seen during the play, it can try to initiate a discussion with any character it has encountered until now. This character will send a `RequestTalk` message to one of them, and wait for a reply. Depending on the other character level of busyness or like dislike toward the requester character, it may refuse or accept. If the other character refuses, the agent will have to try to initiate a talk with another character. In case there are no other character, the goal is considered unachievable and the agent will cancel the chosen goal. On the other hand, if the other character accept the talk, the agent will send a second message, this time to ask for a cooperation.

When the other character receives this message, two checks are done before giving a reply. The first one checks if the agent is available to execute a cooperation goal or if it is too busy to execute such goal. The second one checks if the agent is actually able to satisfy the goal conditions list given by the `RequestCooperation` message shown in the listing 1. While the first check is quite simple, the second one implies the character to execute the GOAP planner

```
Message:
{
    type: RequestCooperation;
    goal: {
        condition1,
        ...
    }
}
```

Listing 1: Request Cooperation Message


Figure 6: WONDER prototype project example: GOAP tree search for a plan to escape from the prison.

to try to find a suitable plan. If found, it will send a positive reply message to the character asking for cooperation. This character will then send an `EndTalk` message and start to execute its part of the plan. Similarly, when the other agent will receive the `EndTalk` message, it will execute its part of the plan. While both characters are now executing their plan, there is still one important point the agent asking for cooperation has to take into account. When this agent will arrive at the execution of the cooperation action that should be executed by the cooperative character, the character will wait and analyze the game state to see if the conditions the other character has to validate are validated. This analysis can be done in two ways: using its perception component, the character can see the game state changing by the influence of the other character executing its part of the plan. Or, if the two characters cannot see each other, or the conditions cannot be checked by the perception component, the character will wait a `CooperationResult` message before continuing its own plan until its completion.

## Experiments/Results

Thanks to these two systems working together, we are able to achieve multi-agent cooperation without commander-like entity, where each agent has its own abilities, memory, and decision making systems. To illustrate this multi-agent cooperation, we will take an example of our WONDER prototype project where quests can be solved through cooperation. One of the quest of this game consists of getting out of a prison by the help of other characters. A very simple


Figure 7: WONDER prototype project example: Successful cooperation leading to the escape of the main character.

setup can be seen on figure 1. The main character is currently locked on the left side of the figure behind the prison bars. Another character can be seen on the right side with a pressure plate that can open the prison. The goal of the main character is to escape from the prison, while the other character is just idle for the demonstration. The figure 6 shows the plan found by the main character after discussion with the other character. We can see the node game state with their list of conditions. The name of the condition displayed is formatted so that the first later indicates if it is `[T]RUE` or `[F]ALSE`, then the name of the condition is appended. If a condition is grayed out, it means that it has been validated by the previous action (Reminder: as it is a backward planner, we go from goal to initial state). If a red conditions has a validation mark, it means that this condition is already valid in the initial state. The remaining red conditions are conditions that still need to be validated by other actions later in the plan. Between two nodes, we can see a box with two icons representing one variable (objects in this case) with the action chosen by the GOAP planner. For instance, if we follow the red line from the `Goal` node state, the action between the `Goal` node and the following node game state indicates that this action is the action `MOVE`. Also, if the action is of color blue, it means it is executed by the main character. If the action is dark yellow, it means that the other character is executing this action. The plan of both agent has been merged on the display UI, however, they are actually two separate plans, planned by each character. If we follow the red line, we can see that the last node on the right has all its conditions validated, which means that this node is fully compatible with the initial state. As such, the red line represents a valid plan. By looking in details at the plan found, we can see that (from right to left) the other character moves to the pressure plate. The pressure plate activates the prison bars and it opens them. Then the main character moves out of the prison. When executed by both characters, the main character is able to escape successfully from the prison, as shown in figure 7.

## Conclusion

Thanks to our GOAP cooperation system with our messaging system, our characters are able to cooperate together to achieve complex goals. Despite not having much knowledge about what can do the other characters, they are able to plan for a potential cooperation plan. Then they are able to explore the world and communicate with the different characters to find a character that can do and accept the cooperation request. Together, they know when to wait for the other character to finish its tasks, and how to find out if it was successful and can continue its plan to completion. We demonstrated our systems in our WONDER prototype project through multiple figures (1, 6, 7). However, such multi-agent technologies are more costly than traditional AI Director due to the lack of knowledge about the other agents. Many failures during planning and communication can happen before finding a satisfiable plan to achieve their goals, or even finding a character that accepts the request. Nevertheless, we believe it is the cost to pay to achieve further autonomous and believable character-interactions to the player.

## Acknowledgments

We would like to thank our colleagues from the Advanced Technology Division of Square Enix Co., Ltd. who provided insight and expertise, but also helped on the creation of many assets for this WONDER prototype project.

## References

Boeda, G. 2018. An architecture for immersive interactions with an emotional character AI in VR. https://dl.acm.org/doi/10.1145/3289160.3289165. Accessed: 2021-07-21.

Boeda, G. 2019a. Enhanced Immersivity: Using Speech Recognition for More Natural Player AI Interactions. https://www.gdcvault.com/play/1026476/Enhanced-Immersivity-Using-Speech-Recognition. Accessed: 2021-07-21.

Boeda, G. 2019b. NPCs Have Feelings Too: Verbal Interactions with Emotional Character AI. https://www.gdcvault.com/play/1026254/NPCs-Have-Feelings-Too-Verbal. Accessed: 2021-07-21.

Booth, M. 2005. From COUNTER-STRIKE to LEFT 4 DEAD: Creating Replayable Cooperative Experiences. https://www.gdcvault.com/play/1422/From-COUNTER-STRIKE-to-LEFT. Accessed: 2021-07-21.

Brenner, M. 2003. A Multiagent Planning Language. In *Proceedings of ICAPS'03 Workshop on PDDL*.

Cao, W.; Bian, C.-G.; and Hartvigsen, G. 1997. Achieving efficient cooperation in a multi-agent system: the twin-base modeling. In Kandzia, P.; and Klusch, M., eds., *Cooperative Information Agents*, 210–221. Berlin, Heidelberg: Springer Berlin Heidelberg. ISBN 978-3-540-68321-6.

Gehlin, R. 2014. Action Planning and Cooperation (APAC) between multiple AI-agents. *Institute of Technology* 55.

Orkin, J. 2005. Agent Architecture Considerations for Real-Time Planning in Games. In *Proceedings of the First AAAI Conference on Artificial Intelligence and Interactive Digital Entertainment*, AIIDE'05, 105–110. AAAI Press.