Towards a resource-based model of strategy to help designing opponent AI in RTS games

Juliette Lemaitre, Domitile Lourdeaux, Caroline Chopinaud

Towards a resource-based model of strategy to help designing opponent AI in RTS games

Juliette Lemaitre[1,2], Domitile Lourdeaux[1] and Caroline Chopinaud[2]
[1]*Heudiasyc - UMR CNRS 7253, Université de Technologie de Compiègne, 60200 Compiègne, France*
[2]*MASA Group, Paris, France*
{*juliette.lemaitre, domitile.lourdeaux*}*@hds.utc.fr,* {*juliette.lemaitre, caroline.chopinaud*}*@masagroup.net*

Keywords: Strategy, Behavior Modeling, Game Design, Player Experience, Adaptation.

Abstract: The artificial intelligence used for opponent non-player characters in commercial real-time strategy games is often criticized by players. It is used to discover the game but soon becomes too easy and too predictable. Yet, a lot of research has been done on the subject, and successful complex behaviors have been created, but the systems used are too complicated to be used by the video games industry, as they would need time for the game designer to learn how they function, which ultimately proves prohibitive. Moreover these systems often lack control for the game designer to be adapted to the desired behavior. To address the issue, we propose an accessible strategy model that can adapt itself to the player and can be easily created and modified by the game designer.

1 INTRODUCTION

The Artificial Intelligence (AI) used for non-player characters (NPC) in commercial video games does not take full advantage of the academic research that exists on the subject. Some games have used planning systems (Orkin, 2004) or even learning algorithms, but they remains exceptions. The main reason lies in a clear difference in goals and constraints that causes the solutions developed by the research community to be inadequate for the industry.

Indeed, during game development, the design of an AI for the NPCs is commonly a low priority task. Other aspects, like 3D graphics and animation, account for most of the time and money available for game design. The lack of time also leads to the creation of AI solutions from scratch, specific to a single game, producing underexploited and non-reusable AI. Hence, introducing a new AI system into the design process requires proposing a reusable and easy to understand and use solution.

Some video games, for which AI is at the heart of the scenario, need to provide interesting and complex AI. This is the case with real-time strategy (RTS) games: their environments present multiple challenges, widely studied by the research community. But even if the required complex behaviors constitute a necessarily important part of the game design, simple but ill-adapted solutions like finite state machines (FSM) and behavior trees (BT) are used to design NPCs' behaviors rather than more advanced technologies like genetic algorithms or planification. Thus, simple and accessible solutions to design AI are preferred, but this simplicity is not the only requirement for an AI system.

Indeed, BT and FSM not only offer easier comprehension but also control and reliability. Most of the solutions provided by the research community lack one of these criteria: sometimes they do not provide a solution, making the NPC inanimate; when they do provide a solution, it is often difficult, even impossible, to customize it to obtain the desired behavior. Indeed, while in research most of the studies aim to improve the results of the AI (faster or better behaviors than the opponent), the goal of a video game is also and for the most part to entertain the players. If the two goals are not completely incompatible, they are rarely satisfied by the same solutions. In a game an NPC can have several roles: opponent, partner, assistant, populating the environment... Each one can bring fun to the game in different ways, which is the difficult part because there is no consensus on the explanation of fun; but game research and the industry have tried to understand it and several definitions have appeared in the last few years.

In short, AI in games has to be simple to design, accessible for the game designer, reusable and suitable for designing attractive scenarios. These criteria

mean that the design process for behaviors of NPCs in a game have to be easy for designers to test, control and customize to achieve the desired behaviors. In order to address this issue, we propose a solution to conceive and efficiently test collective behaviors, commonly called strategy in RTS games. In this paper we present a model to design strategies accessible to the game designers. The proposed model aims to be used to design behaviors that are adaptable, reusable and reliable. To address the specific issue of entertainment, a semantic layer is added to the behavior model to clarify the impact of the behavior on the game experience. After presenting the related work and its limits, we will present our behavior model and then discuss the perspectives we foresee for this work.

2 RELATED WORK

2.1 RTS Research

(Ontanon et al., 2013) has gathered most of the work related to RTS games AI, separating the work from Starcraft AI competitions in the CIG and AAAI conferences, and the work done for research purposes.

Both round-robin and single-elimination tournaments proposed during the CIG and AAAI conferences focus on the performance of the competitors. The AI system of each participant competes against the others and the only parameter that will determine the winner is the percentage of winning games. Moreover, the architectures created for the events are adapted specifically for Starcraft, broken down into several parallel and hierarchical modules (Ontanon et al., 2013). It shows the importance of decomposition of the decision task, but the specialization of the modules makes it difficult to reuse them in another environment. With our model, we aim to provide a strategy structure that can be used in diverse environments.

The RTS environment has become widely used as a testbed in game research because of the multiple challenges that emerge from it (Buro, 2003). Multiple IA techniques have been tested for their effectiveness but often lack usability required for commercial use. For example (Dereszynski et al., 2011) uses sets of game logs from Starcraft to produce hidden Markov models. The resulting behaviors depend entirely on probability which makes it unpredictable and limits the game designers control. Case-based planning in (Ontanon et al., 2007), studied more extensively in (Palma et al., 2011), also fails to provide the necessary control over obtained behaviors and requires experts to create example libraries. Furthermore, both use a learning process which does not integrate well in the creation process of a video game. Indeed, a learning process can only be performed properly on a completed game. If an incomplete version is used, it is faster to restart the learning process from scratch than to adapt the previous result to the final version. Automated planning has also been used in game research, (Churchill and Buro, 2011) uses it to optimize build order in Starcraft. The computing time and the vast search spaces in RTS games prevent it from being used for the entire decision mechanism. We would like to provide a reusable solution where the game designer has control over the AI produced and can understand the resulting behaviors.

2.2 Defining Fun

A lot of studies have tried to explain the meaning of fun and how it can be triggered. The most studied aspect of fun is the level of difficulty, which needs to be challenging, neither too easy, nor too hard, to stay between anxiety and boredom as defined in the theory of flow (Nakamura and Csikszentmihalyi, 2002). Most of the studies resulted in classifications of kinds of fun, (Malone, 1980; Lazzaro, 2004). (Read et al., 2002) defined 3 dimensions of fun: endurability, engagement and expectations. Other studies focus more on the player, (Bartle, 1996; Bateman and Boon, 2006). When comparing these classifications, we can find some similarities but no consensus has been made on which one is the most accurate and their use in the creation of behaviors is still unexploited even though fun is at the heart of game design. In our solution, we want to allow the designer to create a behavior that reacts and adapts to the player, so that it provides a fun and interesting experience.

3 PROPOSITION

The goal of our work is to provide an accessible strategy model in order to simplify the design of complex behaviors. A strategy is defined as the decision-making process of the allocation of available resources, such as agents or objects, to sub-tasks in the pursuit of an overall goal. Our model aims to facilitate the designing of reusable, reliable and easily extendible collective behaviors through the description of strategies. It will therefore fit into the creation process of a new video game, during which several modifications of the game mechanisms are applied and require the adaptation of every element, including the AI of the NPC. We want the model to be apprehensible, to the extent that the reason for the occurrence

of an unwanted behavior can be spotted easily and solved in order to allow designers to manage the coherence, credibility and fun of the designed behaviors.

3.1 The Strategy Model

Our model aims to facilitate the creation of collective behaviors in RTS, and more generally the behaviors of several agents which need to be coordinated. To construct the overall behavior, that we call strategy, we use a hierarchical structure to allow decomposition into simpler sub-behaviors and thus handle scalability. A behavior is then composed of a set of sub-behaviors and the strategy can therefore be represented by a Directed Acyclic Graph (DAG) with a unique root node as shown in figure 1. All vertices without outgoing edges are primitive tasks, meaning that it can be directly applied in the virtual world.

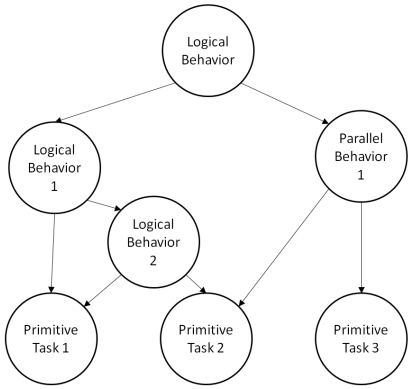

Figure 1: Strategy.

A behavior is defined by a set of behaviors or primitive tasks, its type, and the resources it needs. The type of a behavior can either be logical or parallel. The parallel type allows the expression of collective behavior while the logical type allows the AI to choose a behavior according to the state of the game. To avoid complexity, it has been decided to clearly separate them by defining a behavior as either logical or parallel rather than permitting a mixture of both in a single behavior. The parallel type requires additional information about the distribution of resources between its different sub-behaviors. The logical one needs information about the decision-making process to choose only one of its different sub-behaviors at each evaluation. They are detailed in 3.1.2 and 3.1.3.

3.1.1 Primitive Tasks

Primitive tasks are the smallest part of the decomposition of a behavior. They cannot be decomposed, but each one is linked to a piece of code that allows it to run in the virtual environment. They are defined by a set of resources that are required for it to work. Each resource is defined by a tuple *<R, Min, Max>* where R is the type of resource defined with the resource model presented in 3.2, Min is the minimum amount required, and Max is the maximum amount possible and can be set to -1 if there is no maximum. This offers more flexibility than a fixed number and is more suited to evolving numbers of resources which is the case in RTS games. For example, to represent the task of construction, a primitive task can be created with the following resources:

```
<<worker, 1, -1>, <ground, 1, 1>>
```

In this way, several workers can work on a construction that requires a unit of ground. To handle failure, a status of ending, success or failure, is returned to the parent behavior so it can adapt accordingly.

3.1.2 Logical Behaviors

A logical behavior represents the logic of selection of a sub-behavior from several. Its execution consists of selecting the sub-behavior to be executed, according to the previous one executed and the current state of the virtual world. It is represented by a tuple *<B, M, SB, CB>* where:

```
B is the set of sub-behaviors
M is the set of triggers <OB, T, DB>
SB is the starting sub-behavior
CB is the current sub-behavior
```

A trigger is composed of a transition T that triggers the selection of the sub-behavior DB if OB is the current sub-behavior. The new sub-behavior DB can be one of the predefined sub-behaviors *Success* or *Failure*, in which case the transition causes the end of the logical behavior and a signal corresponding to the selected status is sent to the parent behavior.

The transitions can be internal or external signals, or information requests: internal signals are the success or failure feedback from the current sub-behavior, external signals come from modules dedicated to the game, and information requests are represented by predicates. For example, an external signal can come from a module that manages the state of the world if the condition is a partial world state, or it can come from a module of player modeling if it depends on the player state. In the case of an information request, the transition is activated if the predicate is true. Signals are used for punctual events whereas information requests are used to check more static conditions.

Consider an example consisting of 3 sub-behaviors: Explore, Fight, and Gather. We define Explore as the initial sub-behavior, then if enemies are encountered or food is found during the exploration, the sub-behaviors Fight or Gather, respectively, are

selected. When fighting of the enemy or gathering of food terminates successfully, the Explore sub-behavior is again selected. In case enemies are spotted during the food gathering, the fight behavior is the preferred selection. This example can be expressed as follows. Note that here *Gather* is the currently selected sub-behavior.

```
B = {Explore, Fight, Gather}
M = { <Explore, SpottedEnemy, Fight>,
      <Explore, SpottedFood, Gather>,
      <Fight, Success, Explore>,
      <Gather, Success, Explore>,
      <Gather, SpottedEnemy, Fight> }
SB = Explore
CB = Gather
```

A logical behavior can thus be represented as a directed graph that can be cyclic, the sub-behaviors being the nodes and M being the edges with OB as the origin and DB as the destination. A graph illustrating the example above is shown in figure 2.

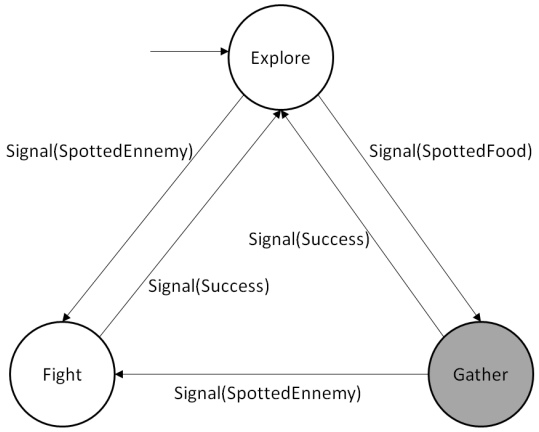

Figure 2: Simple example of logical behavior.

3.1.3 Parallel Behaviors

A parallel behavior handles the distribution of resources between its sub-behaviors. It is composed of priority levels that are represented by a tuple $<M, C>$ where M is the maximum number of times the priority can be executed in parallel and C is a set of components. Sometimes there is no need for a maximum value for M, especially when assigning default tasks to all the remaining resources with the lowest priority levels. In this case by convention we use the value -1 for M, simply to indicate that no limit is required. A component is described as a tuple $<B, W>$ where B is a sub-behavior and W is a weight. The resources are allocated to the first levels as long as possible and until the number M is reached. The weights W are used during the distribution of resources among the sub-behaviors of the priority level, but are not useful when there is a unique sub-behavior. A sub-behavior can appear in several priority levels, but it cannot appear more than once in a priority level. The priority and weight mechanisms allow the description of the distribution of resources without knowing the exact amount of available resources.

Table 1: Simple example of parallel behavior.

Maximum iterations	Behavior	Weight
2	Fight	1
	Cover	2
1	Alert	
-1	Explore	

Considering table 1 as the selected behavior and that each of its sub-behaviors need only 1 soldier, if we suppose that 10 soldiers are available, the resource allocator will proceed as follows: first it will use 3 soldiers for the first priority level, 1 for fighting, and 2 for covering. As the number associated (2) is not attained, the resource allocator will continue with the first priority level and allocate 3 soldiers again, 1 for fighting and 2 for covering. The first priority level has reached its maximum number of executions, so the resource allocator goes to the second priority level and assigns a soldier to alert the others. The maximum number of executions is also reached, so the third priority level is selected. As it does not have a number associated, the remaining 3 soldiers are assigned to the task Explore.

3.2 The Resource Model

A resource can be an object, an agent, but also a location, or a skill. A hierarchy of resources is defined by the game designer. Only the leaves can be used when creating a resource, to define its type, but all the types can be used to define the requirements of a primitive task, meaning that when a resource is required to execute a behavior or a task, the type can be more or less specific. For instance, figure 3 represents a hierarchy of types of resources. An agent of the game can be a *Soldier* or a *Worker*, and the environment also includes resources of type *Sword* and *Shovel*. This means that a primitive task that requires 1 resource of type *Agent* can be performed by a *Soldier* or a *Worker*. More advanced options allow the game designer to indicate if a task requires 2 resources of type *Agent* that are of the same subtype, or of a different subtype, or that it does not matter.

A resource is defined by the following characteristics: its type cannot change during its lifetime; it can be destroyed or created by a primitive task; it can be

modified through destruction then creation of a new resource.

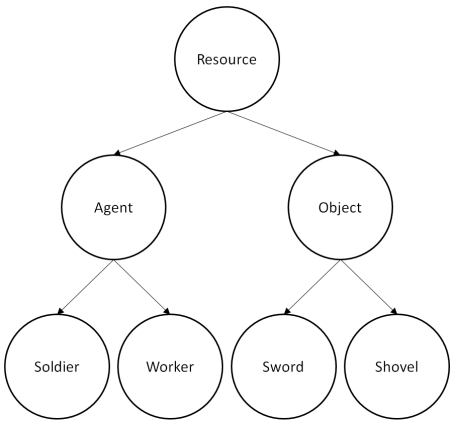

Figure 3: Resources hierarchy.

3.3 Semantics

A semantic layer will be applied to the behavior to clarify for the game designer the impact of the behavior on the game experience. To do that, a typology of feelings resulting from the experience of play will allow the tagging of the behaviors, and the game designer will be able to create the desired experience in a logic behavior by selecting sub-behaviors tagged with the appropriate feelings. For example an external module can spot that the player plays a decreasing number of actions per unit of times, deducing that the player is becoming less involved, and send a signal that will trigger an aggressive behavior, causing a battle requiring the player to be more active.

4 A USE CASE

The following example utilizes our model to build a basic behavior in an environment which employs the most common mechanisms found in RTS games: a search is performed for mineral resources which can then be extracted for the creation of buildings and units (workers or soldiers). The buildings upgrade knowledge and allow the construction of more efficient soldiers. To simplify the understanding of the proposed behavior, we will only provide a detailed description of a small section of the overall strategy.

The overall strategy itself is shown in figure 4: the strategy begins by creating buildings until it is considered necessary to start preparing an army by creating soldiers, this eventually leads to an attack on the enemy. Several reasons can be found to start building an army: it can be a length of time, the number of buildings created, but it can also be triggered by intelligence, for example if the enemy is also preparing an army. If it is too late to prepare an army because the enemy has managed to surprise us, combat mode is engaged directly.

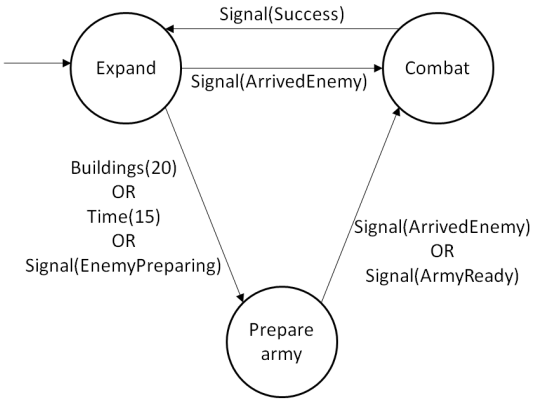

Figure 4: Overall strategy of the example.

The Combat sub-behavior can be described as a parallel behavior, as represented in table 2. The soldiers engage in combat but the workers will continue to collect and transform resources to create more soldiers. Here, every soldier that attacks an enemy is covered by another soldier. The Collect sub-behavior is the default behavior so it is placed in the last position. If there are not yet enough resources to create a soldier, the sub-behavior Create soldier will not be assigned to workers and they will all collect resources. As soon as there are enough resources to create a soldier, a worker will be assigned to the task.

Table 2: Combat parallel behavior.

Maximum iterations	Behavior	Weight
-1	Fight	1
	Cover	1
-1	Create soldier	
-1	Collect	

The sub-behaviors Fight and Cover are detailed in Figure 5 and Figure 6, respectively. The Fight behavior is relatively simple: shoot at the weakest enemy, and if no enemy is in range, move toward the nearest enemy soldier. The Cover behavior is a little more complex and requires external signals and information requests concerning the environment. It consists of finding a hiding place and placing the soldier behind it, ducking if necessary, then shooting at visible enemies, and finally moving on if the covered soldier has moved and the hiding place is no longer suited.

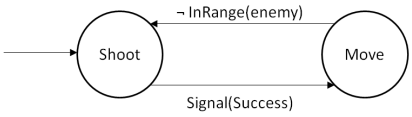

Figure 5: Fight logical behavior.

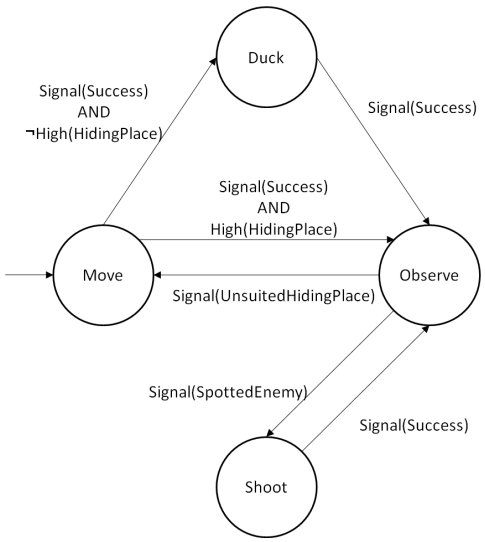

Figure 6: Cover logical behavior.

5 PERSPECTIVES AND FUTURE WORK

We have presented a model that makes building collective behaviour easy. The first step of our future work is to extend expressivity by adding functionality while retaining usability. One way to achieve this objective is to present the model to game designers, with a view to making improvements following the reception of feedback that describes their needs. Indeed this is the principle purpose of this model. The second step will focus on the semantic layer; its purpose is to highlight the player's experience in the conception of AI behaviors. The associated vocabulary needs to be defined, based on the work presented in section 2. The goal is to be able to automatically compute the type of experience that a behavior can provide, so part of the work will be to match experience types with the characteristics of behaviors that cause them.

6 CONCLUSION

In this paper we presented a model for the definition of strategic behavior for RTS games, whose specificity is to combine ease-of-use with the ability to produce high-performing strategic behaviors. The model uses hierarchy and parallelism to be easily understandable. The agents are used as resources, using proportion and priority to separate them on possible tasks, making it adaptable for variable resources. A strategy can easily be modified and sub-behaviors can be extracted and reused. The model is part of a bigger project which aims to provide a behavior generator, meaning that the system could give to the game designer a first strategy to work with, simply by defining the primitive tasks. Future work will focus on making it more accessible by adding a semantic layer.

REFERENCES

Bartle, R. (1996). Hearts, clubs, diamonds, spades: Players who suit MUDs. *Journal of MUD research*, 1(1):19.

Bateman, C. and Boon, R. (2006). *21st Century Game Design*. Charles river media edition.

Buro, M. (2003). Real-time strategy games: A new AI research challenge. In *IJCAI*, pages 1534–1535.

Churchill, D. and Buro, M. (2011). Build order optimization in StarCraft. In *AIIDE*.

Dereszynski, E. W., Hostetler, J., Fern, A., Dietterich, T. G., Hoang, T.-T., and Udarbe, M. (2011). Learning probabilistic behavior models in real-time strategy games. In *AIIDE*.

Lazzaro, N. (2004). Why we play games: Four keys to more emotion without story.

Malone, T. W. (1980). What makes things fun to learn? heuristics for designing instructional computer games. In *Proceedings of the 3rd ACM SIGSMALL symposium and the first SIGPC symposium on Small systems*, pages 162–169. ACM.

Nakamura, J. and Csikszentmihalyi, M. (2002). The concept of flow. In *Handbook of positive psychology*, pages 89–105.

Ontanon, S., Mishra, K., Sugandh, N., and Ram, A. (2007). Case-based planning and execution for real-time strategy games. In *Case-Based Reasoning Research and Development*, pages 164–178. Springer.

Ontanon, S., Synnaeve, G., Uriarte, A., Richoux, F., Churchill, D., and Preuss, M. (2013). A survey of real-time strategy game AI research and competition in StarCraft. *Computational Intelligence and AI in Games, IEEE Transactions on*, 5(4):293–311.

Orkin, J. (2004). Applying goal-oriented action planning to games. In *AI Game Programming Wisdom*.

Palma, R., Gonzlez-Calero, P. A., Gmez-Martn, M. A., and Gmez-Martn, P. P. (2011). Extending case-based planning with behavior trees. In *FLAIRS Conference*.

Read, J. C., MacFarlane, S. J., and Casey, C. (2002). Endurability, engagement and expectations: Measuring children's fun. In *Interaction design and children*, volume 2, pages 1–23. Shaker Publishing Eindhoven.