DOI: http://dx.doi.org/10.7551/978-0-262-31709-2-ch101

Follow the Leader: a Scalable Approach for Realistic Group Behavior of Roaming NPCs in MMO Games

Dario Maggiorini, Laura Anna Ripamonti, Samuele Panzeri

Department of Computer Science, University of Milano, Italy
ripamonti@di.unimi.it

Abstract

In modern Massively Multiplayer Online Games, Non Playing Characters (NPCs) moving on the battlefield play a key role in term of user experience: many time players are required – alone or in groups – to fight or avoid them in order to progress in experience. Unfortunately, standard NPCs behavior, i.e., patrolling between rally points, does not put a significant challenge to players once its deterministic movement pattern is discovered. This paper addresses the problem of defining a smart, more challenging, and natural movement model for NPCs. Getting inspiration from the kids' game "follow the leader" we adopt Artificial Intelligence techniques such as behavior trees and blackboards to provide NPCs with changing paths, dynamic aggregation in parties, and tactical decisions. Following this approach, players' experience will greatly improve thanks to an always-changing battlefield scenario.

1 Introduction

In these recent years, with the gain in popularity of online games, and massive online games in particular, we are witnessing a progressive increment in the number of Massively Multiplayer Online Games (MMOGs) available over the Internet, as reported by (MMORPG community). A MMOG is a multiplayer game where a huge number of users in a shared virtual environment can interact in real time and band together to achieve shared goal such as conquering a map or defeating a group of monsters. When designing the environment, Non Playing Characters (NPCs) are the most common interaction points between players and the game system. NPCs are autonomous entities in the virtual world in charge of assigning tasks (quests) to players or to oppose players when they are trying to fulfill quests. In particular, mobs (shorthand for "mobiles") are NPCs roaming the map and typically implement monsters to be defeated or avoided.

In standard implementations, due to technical and computational constraints, mobs (alone or in groups) patrol between pre-defined rally points on a map. This way, it is easy to keep mobs density uniform on the map and level designers may be reasonably sure about not having unguarded passages through a battlefield. Moreover, this standard approach is not very computational intensive and allows for a better scalability. Nevertheless, mobs moving between fixed rally points are not very entertaining for players (Koster, 2004): fixed paths can be easily guessed by experienced players and free passages in the battlefield will be discovered eventually. To address this issue, we believe that Artificial Intelligence (AI) techniques can be exploited in order to provide a better user experience. By using AI, it is possible to model every mob as an independent agent; each agent may be able to randomly choose its own path while coordinating with others to keep mobs density constant.

This paper focuses on proposing and implementing a novel movement model for mobs inside an MMOG. NPCs will benefit from changing paths, dynamic aggregation in parties, and tactical decisions. The proposed solution will make gameplay more challenging and entertaining for players by providing always-changing battlefield scenarios.

The rest of this paper is organized as follows: first we discuss issues related to implementation of smart NPCs in Sec. 2; then, in Sec. 3, related work in literature is presented. The proposed solution is introduced and discussed in Sec. 4 while Sec. 5 provides details about our prototype implementation. Finally, Sec. 6 concludes the paper.

2 Artificial Intelligence and MMOGs

Artificial Intelligence has been present in videogames starting from the late 70s. Among the first examples we can find Qwak (Atari, 1974) and Pursuit (Atari, 1980). Pac-Man (Midwest Games, 1980) is the first notable example where the opponents, controlled by AI, follow distinct - and personalized - behaviors. Starting from Pac-Man, it become commonplace to include an AI subsystem in every game engine.

Videogame implementations up to a few years ago saw every agent as an independent entity taking autonomous decisions. This design was mainly due to the limited computational power available, constraining agents to very simple behaviors. In the majority of cases, NPCs were designed to just go toward or away from the player. Nevertheless, thanks to a new technology wave in the past decade, complex decision making techniques have been introduced in videogames. These decision making techniques allow for tactical coordination between agents with a significant increase in realism. Unfortunately, this same evolution is yet to come in MMOGs where the extremely high number of players and NPCs poses serious scalability limitations; World of Warcraft (Blizzard, 2004), as an example, reports 11.1 millions active users as of today.

When implementing an online game, one of the most important factors for player experience is the frame rate (Frame Per Second, FPS), which is the rate used to update the

game status on client side. A good, and constant, FPS value (typically between 30 and 60) provides a reactive game with smooth visuals. A reduction of FPS due to workload will lead to variable interaction times and unpredictable lags in the world evolution. In an MMOG, the update of the virtual world status is performed in a centralized way: a game server will take care to compute the new world configuration and distribute it trough the network to all clients. To reach 30 FPS on every client, we are required to solve all interactions between users (and NPCs) at least once every 33 ms at server side. As a result, the number of NPCs and the complexity of their behavior is technically limited by the server computational power. Since the number of agents to simulate in an MMOG is at least two orders of magnitude greater than a single player game, AI solutions commonly used in standalone games becomes technically useless; game designers are then required to choose between few smart NPCs (in a small world) or many dumb ones (on a larger map).

Due to the above limitations, mobs' behavior in a MMOG follows a very simple model: every agents can assume two states: idle and in combat. When in idle state, the mod patrols between pre-defined rally points. If the mob is in combat state, it moves toward the player in a straight line and, when it gets close enough, starts attacking. The transition between these two states is triggered by the player's avatar being in or out the field of view of the mob. Usually, there is very little (if any) coordination between NPCs converging on the same player.

3 Related Work

Despite the fact that the problem we are addressing in this paper is of obvious interest for MMOG service providers, it seems, to the best of our knowledge, that limited efforts have been devoted to this topic from the scientific community. We may hint to two reasons for this situation: the tremendous expansion of online gaming is a relatively recent phenomenon, and commercial MMOG engines are usually secretly kept by the running companies making it very difficult for researchers to perform experiments. We believe that in the future, with the increasing popularity of open-engine MMOGs such as PlaneShift (Atomic Blue, 2000), the situation will positively change.

Most probably, the first work on the topic is (Reynolds, 1987). In this paper, the author proposes a computer model for a coordinated animal motion such as bird flocks and fish schools. These generic simulated flocking creatures have been baptized *boids*. While boids are actually providing a realistic behavior, they are missing any sort of group management: all agents in the simulation always move as a single flock.

In some more recent studies, such as (Synnaeve, 2010) and (Rhujittawiwat, 2006), AI techniques are applied to NPCs mainly to create a more believable behavior during combat. In (Synnaeve, 2010) authors use Bayesian programming to select actions and pick targets in a battlefield where both allies and foes are present while in (Rhujittawiwat, 2006) a genetic algorithm is used as a learning method to train an NPC about how to assist real players. None of these contributions are set on the specific goal to bound the computational workload on servers and improve scalability. Moreover, both papers use simulated environments for performance analysis instead of real game engines.

Other works, such as (Combs, 2005) and (Fairclough, 2003), focus on using AI to improve storytelling. In particular, in (Fairclough, 2003) a character director system, which dynamically generates and controls a story, is proposed. In both cases, authors focus on storyline quality and completeness; limited attention is devoted to real-time interaction and system performances.

Most probably, the most interesting contribution on the topic comes from Zyda et all (Zyda, 2010a; Zyda 2010b). In these works, authors propose Cosmopolis: a free MMOG for larger-scale social modeling. In this virtual world, AI-driven NPC communities featuring customized cultural models are used as a means of researching interactions between individuals and societies. Despite some similarity in the goals, it is difficult to compare Cosmopolis and our work. This is because Cosmopolis is strictly focused on interaction and designed specifically as a research testbed for social and behavioral models while, in this paper, we propose an approach to be implemented in existing game engines to reduce computational workload on servers.

4 The Proposed Solution

In this section we are going to describe our solution to make mobs' movement and behavior more realistic in MMOGs.

To design our model we took inspiration from the kids' game "follow the leader". In this game a kid is elected as the leader and is free to move and behave as she likes; other participants are supposed to follow and mimic her, anyone who doesn't follow exactly is out of the game. Kids, during play, may also decide to join as followers or leave the group as they like.

We envision a MMOG where mobs move using a random path selection and are able to band together and perform coordinated actions. Every mob will wander randomly until another mob – or a group of mobs – will be in its field of view; when this happens, the wandering mob can decide to join to the mob/group and become a follower. In those cases where two mobs will band together, one of them will be elected leader and made responsible for group steering. Periodically, every member of the group may decide to leave and go back wandering alone; when the leader leaves the group another one will be elected.

From a technical point of view, the group may be considered as a stand-alone agent whose code sits on the leader; the group will be identified by means of a reference point (group position) and an orientation (see Fig. 1).

Following this approach, we can use a smart – and complex – behavior to manage the whole group while keeping the followers from using excessive computational resources. Resources request from the followers may get even lower than the standard approach thanks to the reduced number of decisions they will be required to take. Player experience, on the other hand, will benefit from many improvements:

- leaders (groups) paths may now be selected randomly to raise the bar in term of offered challenge;

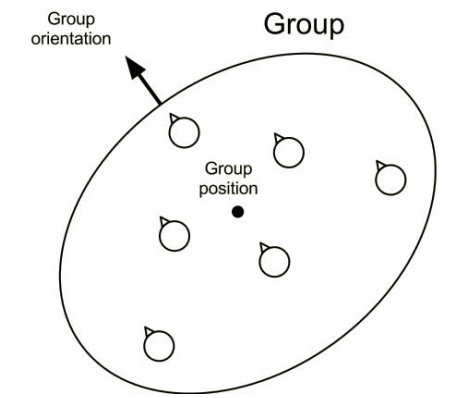

Figure 1: A group seen as a single agent with a leader and followers inside.

- players will feel the mobs as more realistic by seeing them moving in platoons;
- a group will be able to take tactical decisions based on a strategic assessment of player capabilities (i.e., to flee or to fight).

4.1 Movement

The group movement, i.e., picking direction and speed for the group, will be in charge to the leader.

Many strategies are possible for path selection; e.g., we can choose a random rally point, pick a steering angle every frame, or head toward (or away from) other mobs or a point on the map. The implementation of these movement algorithms strictly depends on the kind of mobs we are going to model; as said, doing this at group level allows us to push on complexity without overloading the system. As long as the paths are not pre-determined our model will work and we are not going to discuss algorithms for path selection here.

The first real issue we are going to address is how to make all mobs move in a coordinated formation. As already explained, the leader will set a pace and all followers will do the same while trying to keep a certain distribution pattern. How the pattern is selected is also up to the kind of mobs we are going to model; as an example, if mobs are soldiers an ordered formation is preferred while if they are wild animals picking random positions around a center point might be a good solution.

A more important decision than selecting a distribution pattern is about choosing how group movement should be implemented. We have two options: one- and two-level formation steering. With one-level steering the pattern is fixed and the whole group moves as a single unit keeping rigid distances between followers. This solution is easier to implement but fitting the pattern in the environment (e.g., a pack of wolves traversing a forest) may become a problem. With two levels we have to provide a formation manager to process the pattern and assign destinations to followers (i.e., wolves will swerve around trees and go slightly out of formation but will also try to stay close to the ordered position). This second option is preferable because mobs will blend more nicely within the environment. Moreover, the additional computational requirement for individual movements is, in total, usually less than what is required to fit the formation pattern in the environment.

To implement the two-level group movement we define a center point for the group (which might not be the physical location of the leader). Group collisions and line of sight may be calculated starting from this center point; this way, computation will not be required for every group member and having a global line of sight/hitbox will still be valid approximation. Mobs take relative positions around the center point following the required pattern and, on every frame calculation, a proper destination is assigned to each follower. All computation required from follower agents is a linear movement to location that is “close enough” to the current destination while avoiding environment obstacles.

4.2 Group Management

The group management deals with members joining and leaving the group and how single actions are coordinated between group members. The most viable way to perform these operations is by means of a decision making process.

Implementing a decision making process in every group member is not an easy task. A first option could be to use a Finiste-State Machine (FSM). A FSM is a mathematical model of computation used to design systems that can be in one of a finite number of states. The machine is in only one state at a time and can change from one state to another by a triggering event or condition. A FSM-based approach could be too demanding in term of memory and will require state changes depending on the state of other members, with severe scalability reduction. A secondo option could be to adopt Hierarchical FSM (HFSM), where the states can be ordinary states or super-states, which are FSM themselves. HFSM may be a better solution but they are not always granted to provide satisfactory performance. For the above reasons, we decided to adopt Behavior Trees (BTs) for group management. BTs are a formal, graphical modelling language used primarily in systems and software engineering; they employ a well-defined notation to unambiguously represent a large number of natural

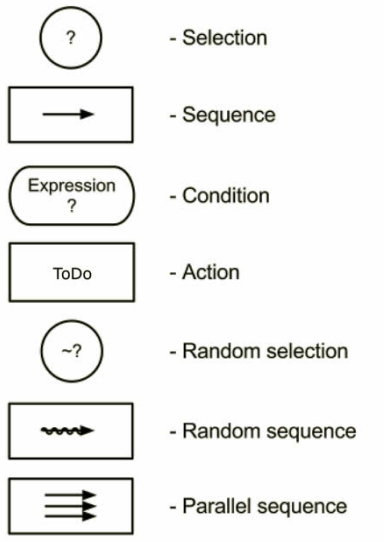

Figure 2: Symbols used in Behavior Trees.

language requirements to express the stakeholder needs for a software-integrated system. Thanks to their structure, BTs also allow to hide a subset of the information shared between group members in action and condition nodes; this way, we can also limit the complexity added by followers dynamically joining and leaving the group. Figure 2 reports the symbols we will be using in our BT schemes.

Communication and coordination between group members is performed using a blackboard system. A blackboard system is not a decision making tool in its own rather than a mechanism for coordinating the actions of several decision makers. The basic structure of a blackboard system has three parts: a set of different decision makers (called experts), a shared memory area, and an arbiter. Any expert may use the shared memory area to read system status and write suggestions about actions to be undertaken. Permission to access the shared memory must be granted by the arbiter. In our specific case, the experts are the mobs and the group agent.

Since, as already discussed before, we can see the whole group as a single agent we need two coordination systems: one between mobs and one between each mob and the group agent. To achieve this we extended the general concept of blackboard to a multi-level blackboard. A multi-level blackboard is a blackboard with two memory areas, which will host both of the above coordination systems (see Fig. 3). The first memory area is used for coordination between mobs; this area allows coordinating and prioritizing actions of single group members, e.g., first attack using magic, then with long-range weapons, and then using melee combat. The group agent uses the second memory area to suggest decisions to all mobs based on group status or environment; e.g., if one of the members is under attack someone must cure it and others should fight back.

An open issue with this blackboard-based approach is how to implement an arbiter that is able to satisfy the timing requirements of an MMOG. To speed-up execution, it is possible, when the number of experts is low, to remove the arbiter and use BTs where one or more nodes are tagged using states and priority values. During execution, first the group agent and then the other mob agents run their decision process. Once the group agent has run to completion it will write on the blackboard the selected action with its associated state and priority. After the group agent is done, all mob

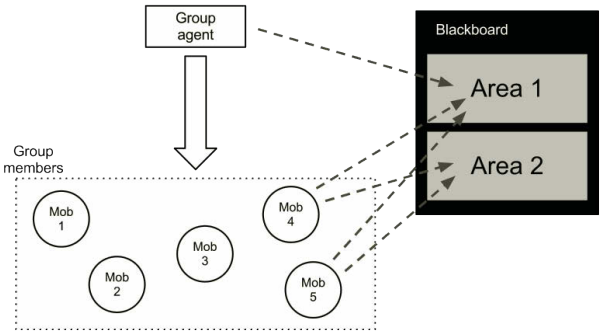

Figure 3: Scheme of a multi-level blackboard.

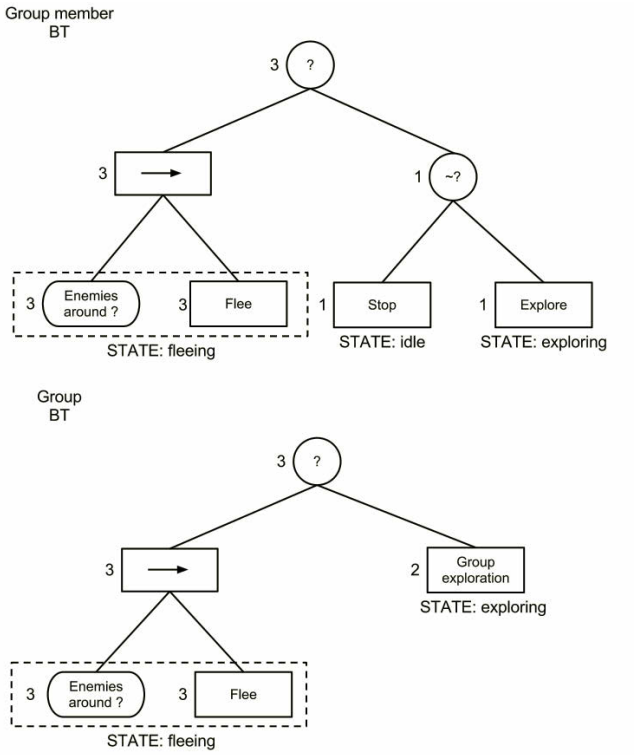

Figure 4: Sample Behavior Trees for an exploration group.

agents can start executing their own BTs. During execution, branches with a lower priority value than the one reported on the blackboard will not be evaluated. If, at the end of the execution a group member reaches a state with a higher priority than the one on the blackboard it will perform that action autonomously, otherwise (i.e., an untagged node has been reached or there is no execution with nodes on higher priority) it will perform the action selected by the group. In Fig. 4 a couple of sample BTs for an exploration group are shown. In the picture we can observe that explorers my act alone or in a group. In the case enemies are around the explorer will always flee. In the case no one is around it may decide to stay idle or to explore the surrounding, unless it is in a group; otherwise, it will be always required to explore with the other mobs (priority 2 on node "Group exploration").

As we can see, following this approach it is easy to model complex behaviors for partially independent agents inside a group. Moreover, a single BT is able to describe both group and solo behavior for all mobs.

5 Prototype Implementation

In this section we are going to describe a prototype implementation of the system we presented in the previous section. This test application has been implemented using C++ and OpenGL.

In our prototype we define a square battlefield where a variable number of mobs and environment obstacles are randomly placed; see Fig. 5 for a screenshot. In the figure, the player's avatar is located in the lower right corner and the user can change its position using the keyboard. All other agents

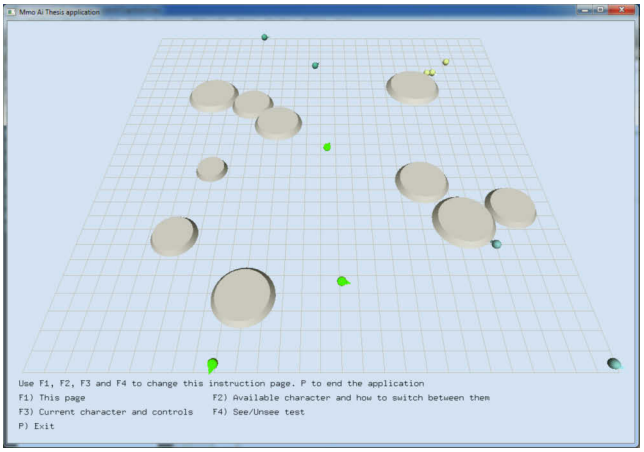

Figure 5: A screenshot of the prototype application.

are calculated in real time by our engine. To mimic an online game, the application implements two independent software modules (back-end and front-end) exchanging information via socket. The back-end works as a server taking care of AI execution for all agents and sending virtual world updates to the frontend. The front-end draws the virtual world on screen and implements the user interface. Using the front-end module the user can manipulate simulation parameters and move her avatar.

Mobs can be assigned to different groups (indicated by their color in the screenshot); only mobs belonging to the same group can band together. These groups are modeling different kind of monsters sharing the battlefield. Mobs will not attack each other.

When a group encounters the user's avatar a tactical decision is taken by the leader based on the group strength (the number of members): if the group size is below a certain threshold (i.e., it is weaker than the user) it will start to flee, otherwise it will start an attack. Since the combat system is not a key point at this stage of the project, attackers will just go toward the target and the defender is supposed to flee. During pursuit, the number of group members may change, because mobs can still join, leave, or are just left behind. On a group change, the tactical decision is re-taken and an attacking group may decide to just leave the player alone. If the player manages to leave behind a pursuing group, the mobs will start moving randomly again.

The purpose of environment obstacles is mainly to check the two-level implementation of group movement. They are currently implemented as circular "hills" to achieve easier collision detection based on distance.

5.1 Movement

Every group (and single mob) moves at a constant speed along its orientation. On every frame calculation a (small) steering angle is randomly selected taking collisions with hills into account. The group will move using an arrowhead formation (as in Fig. 1).

5.2 NPCs Behavior Tree

The BT used in mobs is depicted in Fig. 6. As we can see, the first branch is selected when the mob senses an enemy (or is under attack) or there is a help request from another mob nearby. If the mob is inside a group it will enter the "Notifying" state and write on the blackboard that an enemy is

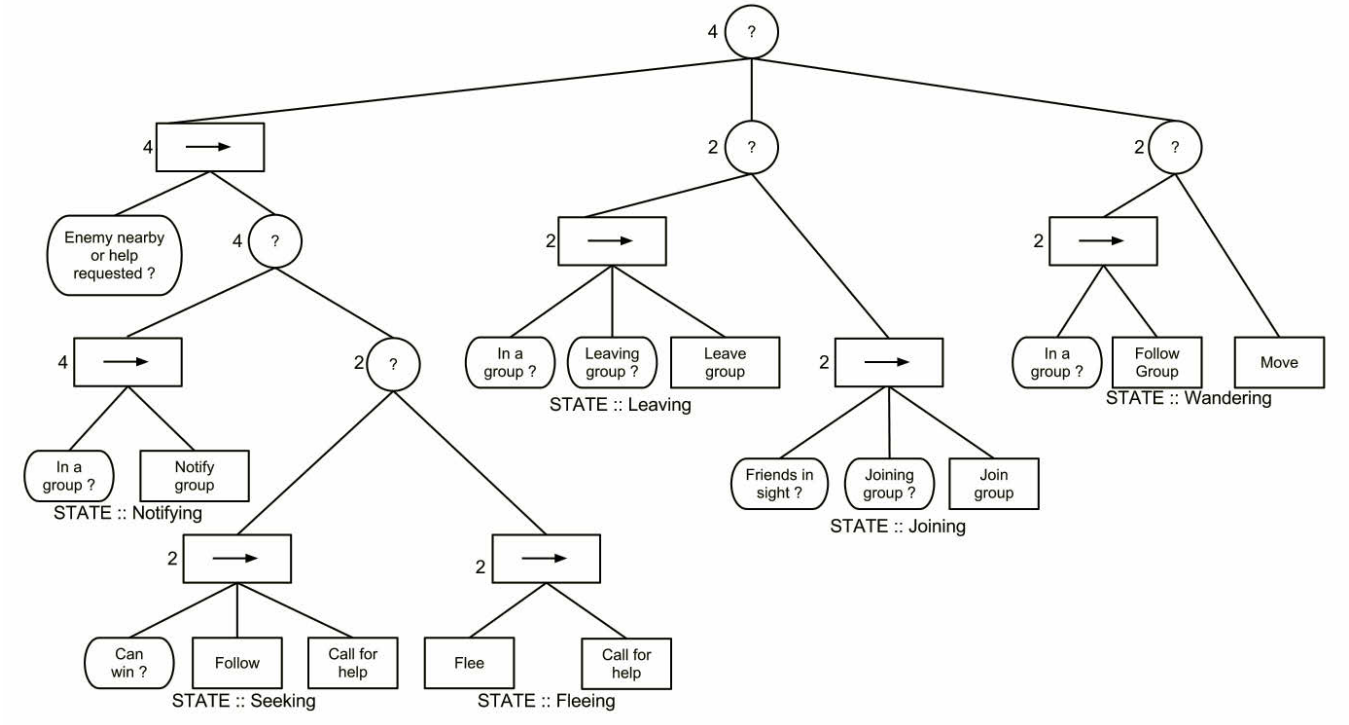

Figure 6: NPCs Behavior Tree in the prototype.

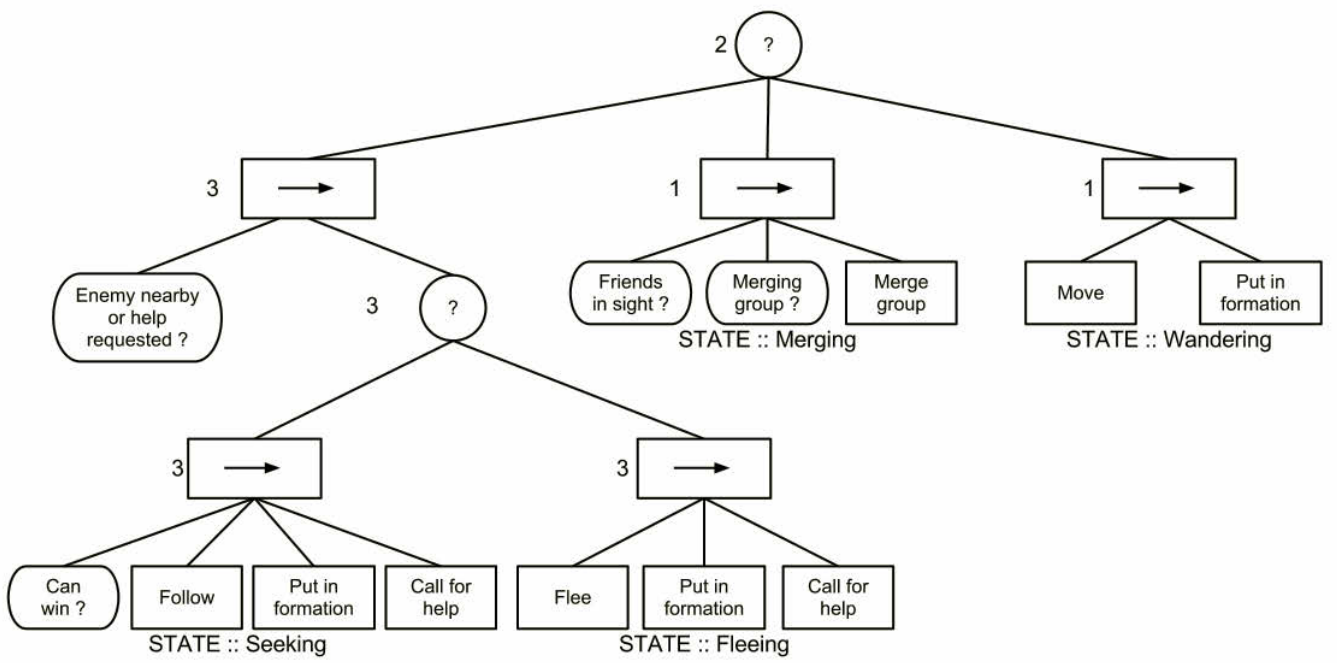

Figure 7: Group Behavior Tree in the prototype.

nearby and submit to the tactical decision that will be taken from the leader. Posting the message on the blackboard will also allow other group members to converge on the enemy even if it is not yet in their field of view. Otherwise, if the mob is not in a group, it must take its own tactical decision: fight or flee. If the mob evaluates it can win the battle it will start in pursuit ("Seeking" state) of the enemy, calling also for reinforcements. Otherwise, if the enemy is stronger, it will start to escape ("Fleeing" state) also calling for reinforcement and, eventually, to become stronger and fight back. The branch we just discussed will be evaluated also in the case the mob will hear a help request coming from another group (if it is from the same group, the blackboard takes precedence). When called for assistance, the mob will still need to take a tactical decision and evaluate if it is better to give support or flee.

The second branch of the behavior tree deals with group management. If the mob is in a group and wants to leave the state is set to "Leaving"; then, the mob will become an independent agent. Otherwise, if there are friendly units nearby and the mob wants to join the group, the state is set to "Joining". The willingness of a mob to join or leave a group is evaluated with a random variable following a Bernoulli distribution where p can be set from the front-end interface (by default p=0.1). In order to limit group dynamics we also set a minimum time t for the mob to stay in a group before leaving: only if enough time has passed the random variable can be evaluated. As for p, t is set from the interface with a default value of 5 seconds.

The last branch is the one managing movement when no other mob is around ("Wandering" state). If the mob is part of a group it will just follow the leader (reading from the blackboard); otherwise, it will pick its own random steering angle and proceed along its orientation.

5.3 Group Behavior Tree

The BT used in groups is presented in Fig. 7. The first branch in the figure, as for the mobs, is selected when there are enemies around. Differently from the previous case, the group will not post on the blackboard but, rather, force a (global) strategic decision on all group members with priority 3. Single mobs will still notify other group members (priority 4) but will follow the strategic decision taken from the group agent, because that specific sub-branch has priority 2. When the tactical decision is taken from the group agent, the "Seeking" and "Fleeing" states also provide an action labeled "put in formation". This action will make all group members rearrange around the group position; the actual location may change depending on the mob stance: attacking, fleeing, or just moving around.

The second branch manages group merging: if two groups are close to each other, one of the group agents may decide to merge and become one with the other. This state has no correspondence with the "Joining" and "Leaving" states of an NPC; since it has a lower priority compared to the second branch in Fig. 6, it is possible for a mob to leave a group independently from an ongoing merging action.

The last branch of the BT takes care of the group movement when no one is around: it sets the new group position (using the blackboard) and puts all participants into formation.

5.4 Testing the Architecture

During experiments the user moves on the battlefield and try to escape from the mobs after attracting their attention. Subjective evaluation of groups in pursuit gave positive results in term of movement patterns: formation seems to

adapt smoothly enough to the obstacles and server workload is reasonable. As a matter of fact, CPU usage never rose over 70%.

Increasing the number of mobs on the map requires more than 500 agents to drop performance below 30 FPS on a middle-end PC (Intel dual core at 2.5 GHz with 4 GB of main memory). Despite the fact that 500 agents are much less than the total population of an MMOG (which is in the order of millions, as claimed in Sec. 2), we have to remember that: (*i*) agents are managing NPC, which are outnumbered by players with at least two order of magnitude and (*ii*) the complete infrastructure of an MMOG provides a virtual world replicated on a number of independent clusters (realms) where each server in a cluster is in charge of a subsection of the local map. With the above considerations in mind, an acceptable workload with 500 agents is also an encouraging result. Nevertheless, the scenario is still hard to evaluate without an implementation inside an actual game engine.

6 Conclusion and Future Work

In this work we addressed the problem of realistic movement for Non Playing Character in Massively Multiplayer Online Games. Realistic behavior can be achieved adopting Artificial Intelligence techniques but, despite tremendous technical improvements in the last decade, application to massive environments is still critical due to performance constraints. We proposed a solution based on a hierarchical approach: NPC can aggregate in groups to be managed as a single agent; a group agent will impose general decisions (such as movement) to all members while single NPCs will manage simpler decisions (such as joining or leaving a group). The proposed solution has been used to implement a prototype. In our prototype NPCs populate a battlefield with environment obstacles; they roam the battlefield and eventually aggregate in groups to attack (or escape from) the user. First results are encouraging: NPCs behavior feels quite natural while server workload is bound to an acceptable level.

Future evolutions for this work include cooperation between heterogeneous NPCs, and the implementation inside a real game engine such as unity3d (Unity Technologies, 2005) or smartfox (gotoAndPlay(), 2004).

References

Atari (1974) Qwak!. http://en.wikipedia.org/wiki/Qwak!_(arcade_game). Last visited Apr. 13, 2013.

Atari (1980) Pursuit. http://en.wikipedia.org/wiki/Pursuit_(video_game). Last visited Apr. 13, 2013.

Atomic Blue (2000). Planeshift. http://www.planeshift.it/. Last visited Apr. 13, 2013.

Blizzard (2004) World of Warcraft. http://www.worldofwarcraft.com/. Last visited Apr. 13, 2013.

Combs, N. (2005). The Intelligence in the MMOG: from scripts to stories to directorial AI. In Proc. of Other Players Conference, IT University, Copenhagen.

Fairclough, C. and Cunningham, P. (2003). A Multiplayer Case Based Story Engine. Dublin, Trinity College Dublin, Department of Computer Science, TCD-CS-2003-43.

gotoAndPlay() (2004). Smartfox server. http://www.smartfoxserver.com/. Last visited Apr. 13, 2013.

Koster, R. (2004). A Theory of Fun for Game Design. Paraglyph Press.

Midwest Games (1980) Pac-Man. http://en.wikipedia.org/wiki/Pac_man. Last visited Apr. 13, 2013.

MMORPG Community. http://www.mmorpg.com/. Last visited Apr. 13, 2013.

Reynolds, C. W. (1987). Flocks, herds and schools: A distributed behavioral model. In Proc. 14th annual conference on Computer graphics and interactive techniques, Anaheim, California.

Rhujittawiwat, T. and Kotrajaras, V. (2006). Learnable Buddy: Learnable Supportive AI in Commercial MMORPG. In Proc. 9th International Conference on Computer Games: AI, Animation, Mobile, Educational & Serious Games, Dublin, Ireland.

Synnaeve, G. and Bessière, P. (2010). Bayesian Modeling of an Human MMORPG Player. In Proc. 30th International Workshop on Bayesian Inference and Maximum Entropy Methods in Science and Engineering, Chamonix, France.

Unity Technologies (2005). Unity3D game engine. http://unity3d.com/. Last visited Apr. 13, 2013.

Zyda, M., Spraragen, M., Ranganathan, B., Arnason, B. and Landwehr P. (2010a). Designing a Massively Multiplayer Online Game/Research Testbed Featuring AI-Driven NPC Communities. In Proc. of 6th AAAI Conference on Artificial Intelligence and Interactive Digital Entertainment, California.

Zyda, M., Spraragen, M., Ranganathan, B., Arnason, B. and Liu, H. (2010b). Information Channels in MMOGs: Implementation and Effects. In Proc. 1st International Conference on Human Factors and Ergonomics in Healthcare, Florida.