20

Hierarchical Architecture for Group Navigation Behaviors

Clodéric Mars and Jérémy Chanut

20.1 Introduction

It is now fairly common to find autonomous human-like characters that are able to navigate in 3D environments, finding paths and avoiding collisions while exhibiting convincing navigation behavior. In the past few years, several major publications have been applied successfully to games: we now have well-tested recipes to generate navigation meshes (nav meshes), compute paths, have pedestrians follow them, and avoid collisions in a convincing way.

However, we still fall short when it comes to group navigation. Like real groups, we want virtual humans to be able to walk down roads with their group of friends. Like real ones, virtual soldiers should be able to patrol while staying in formation. And like real ones, virtual tourists should be able to enjoy a tour of the Mont Saint-Michel following their guide's umbrella.

The aim of this chapter is to provide a base recipe to implement a group navigation system. The first two sections form an introduction, presenting the different kinds of group navigation and the basics of navigation behaviors. The next section presents our proposed hierarchical architecture, and the following sections present different aspects of its design.

20.2 Group Navigation

Taxonomy can be a daunting word, but classification can help establish a common understanding. Reading the navigation simulation literature, three main categories of approaches can be found: flocks, formations, and small "social" groups.

20.2.1 Flocks

A flock is, by definition, a group of birds traveling together. Flocking strategies for navigation can be applied for other animal species as well as humans (e.g., school children crossing the street to the swimming pool).

Entities in a flock travel at roughly the same speed and form a cohesive group without strict arrangement. Figure 20.1 showcases such a flock; you can notice that entities are not facing in the same direction and are not evenly distributed. Generally, an entity in a flock will follow independent local rules to stay in the group. While the term is primarily associated with a large number of entities, the same kind of strategy can be used for groups of only a few members.

Reynolds popularized flocking simulation in what must be the two most cited articles in the field, making their implementation a well-known subject [Reynolds 87, Reynolds 99].

20.2.2 Formations

While flocks emerge from a set of individual rules enforcing the general cohesion of the group, formations are a kind of group arrangement where members enforce a set of strict top-down rules. The first and most important one is the formation's spatial arrangement, that is, the relative positions of members; it is designed for tactical, aesthetic, or other specific purposes. Most of the time, a formation gets much of its usefulness from allocated fields of fire and sight, which is why orientation is also enforced [Dawson 02].

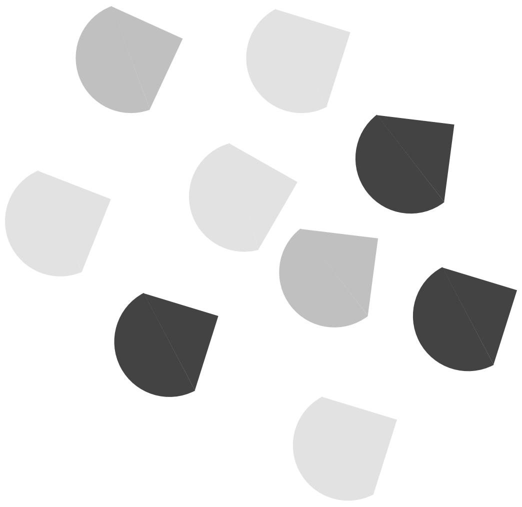

Figure 20.1

A flock of navigating entities.

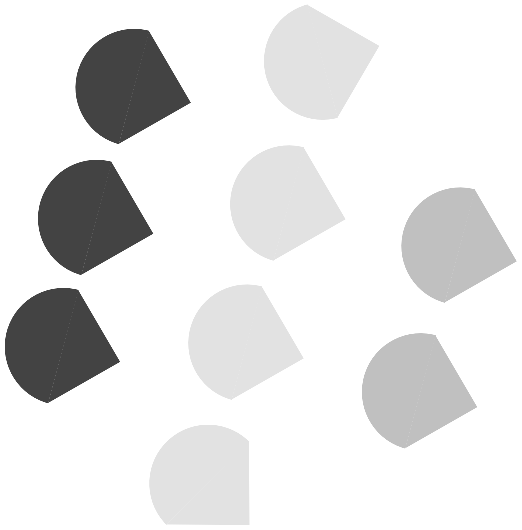

Figure 20.2

A formation of navigating entities.

The last rule is to assign entities having the right role to the right slot: archers at the back, foot soldiers facing the enemy.

Figure 20.2 showcases a formation of nine entities in three layers dedicated each to a specific role, represented by the entities' colors. As formations are important for real-time strategy games, interesting and working solutions have been known for some time: Dave Pottinger, who worked on the *Age of Empire* series, presented his in a Game Developer Magazine article, which is now available for free at Gamasutra.com [Pottinger 99].

20.2.3 Social Groups

Beyond amorphous flocks and rigid formations, groups that are more common in our everyday lives are small and their spatial configuration is the result of social factors and crowd density.

In two different surveys focusing on those small social groups, the authors showed that there are more groups than single pedestrians in urban crowds and that groups of more than four are very rare [Peters 09, Moussaïd 10].

Furthermore, it appears that the formation assumed by the observed groups is influenced both by the lateral clearance to nearby obstacles and by the need of social interaction between members of the group.

These two surveys show that social groups tend to follow three preferred formations depending on the density of the crowd. When motion is not constrained (i.e., when obstacles are far and the crowd density is low), a group tends to adopt an abreast formation that facilitates dialog between its members (leftmost formation on Figure 20.3).

When facing navigation constraints, the group compacts the formation to reduce its frontal width. And, when the lateral space between members becomes too thin, that is, when members are shoulder to shoulder, the formation is staggered. The bending of the

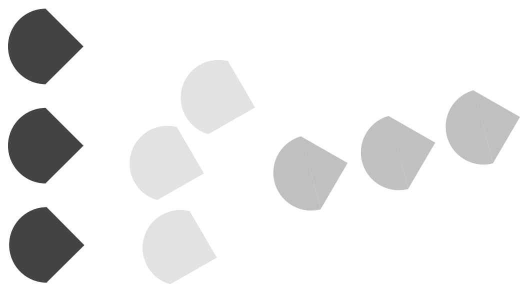

Figure 20.3

Social navigation formations, from left to right: abreast, V-like, lane.

group is, most of the time, forward (V-like formation—in the middle in Figure 20.3) to maintain good communication. A backward bending (inverted-V-like or wedge formation) would be more flexible moving against an opposite flow but seems to be less usual. As the crowd density increases, groups tend to form a tight lane (rightmost formation of Figure 20.3).

Another observation found in these studies is that groups tend to avoid collisions with other pedestrians or with obstacles while remaining together, but if needed, they are able to split and merge back afterward.

In the following section, we introduce a way to efficiently include the group navigation process into a software architecture.

20.3 Navigation Pipeline Architecture

Before delving into topics specific to group behaviors, in this section, we will give a quick overview of what we call *navigation behavior* and the *navigation pipeline* that makes it possible to combine them.

20.3.1 Navigation Behaviors

Typically, a navigation behavior is responsible for computing velocity changes from

- Higher-level individual orders
- Other entities (e.g., neighbors to take into account for collision avoidance)
- And, generally, the state of the world (nav mesh, scene geometry, etc.)

As illustrated in Figure 20.4, this input is usually a path to follow. Paths are computed to reach a target, which is selected by some decision-making code. It then outputs orders driving a locomotion engine that actually makes the entity move.

This architecture supports partial updates. For example, the navigation behavior and the following components can be updated on their own by reusing the previous navigation orders. This allows a compromise between costly components that do not require high reactivity (such as decision making or path finding) and cheaper ones that benefit from a high update frequency (e.g., physics or animation) [Mononen 10, RecastDetour 14].

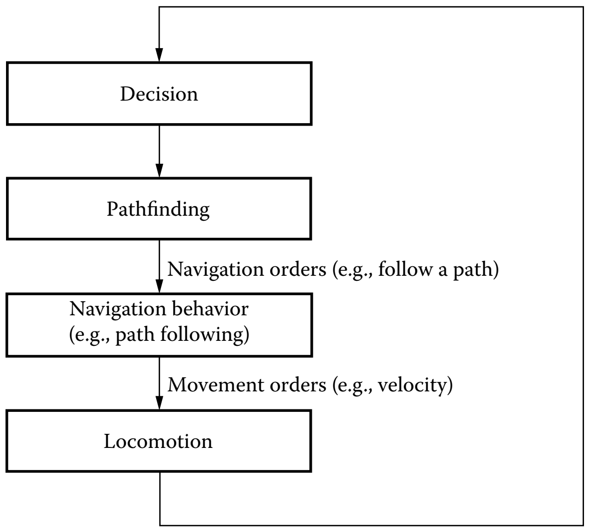
Decision
Pathfinding
Navigation orders (e.g., follow a path)
Navigation behavior (e.g., path following)
Movement orders (e.g., velocity)
Locomotion

Figure 20.4

Typical character update loop involving navigation.

20.3.2 Navigation Pipeline

In real-life scenarios, entities exhibit different navigation properties and are able to handle several types of orders and constraints:

- An entity can reach a target
- Wounded and thus not walking straight
- While it is avoiding obstacles

In order to model this kind of complex behavior, we use a *navigation pipeline*: a sequence of navigation behaviors.

At runtime, the behaviors are updated sequentially, each considering the state of the entity as well as the orders output by its predecessor in the pipeline. In practice, each behavior "corrects" the orders of the previous one.

Consider the "wounded" behavior in the pipeline of Figure 20.5. The previous behavior computes a velocity that makes the entity follow a path. The "wounded" behavior will use this desired velocity as an input and compute a new one that is close to it by applying some noise function. In turn, the "collision avoidance" behavior will correct the orders to avoid future collisions. As the last behavior in the pipeline, it has the last word on the actual decision.

This architecture comes with two great benefits: modularity and reusability. In the case of groups, member entities behaviors need to take into account both the collective goals, for example, flock or stay in formation, and the individual goals, for example, avoid collisions early or minimize deviation from initial trajectory. Modeling these as navigation behaviors and using the navigation pipeline architecture gives us a flexible framework to fulfill these requirements.

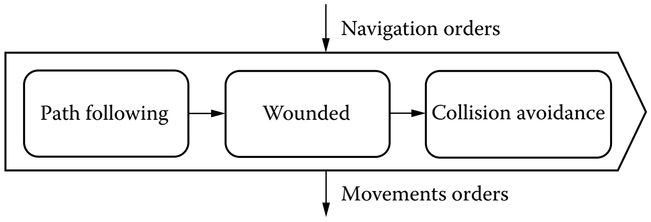

Figure 20.5

A navigation pipeline using three navigation behaviors.

In the following sections, we'll see how the navigation pipeline can be used to model the described group navigation behaviors.

20.4 Group to Members Relationship Model

While some behaviors might be decentralized, in order to manage groups in a context where we need to make them go from A to B, top-down decision making is needed [Musse 01]. A group-level process will be able to make the group move while each of its members follows. Two very different approaches can be used:

1. Make one of the group members the leader.
2. Introduce a virtual entity representing the group itself.

The two following sections will describe the two approaches through their use in the technical literature; the third will describe how we propose to implement an entity hierarchy.

20.4.1 Leader

When trying to enforce a strict equivalence between simulated entities and actual characters, many approaches rely on a leader–followers approach. With such an approach, one member of the group is the leader and the others are the followers. The leader takes responsibility for the whole group's navigation [Loscos 03, Qiu 10].
Implementation using a navigation engine for independent entities is straightforward:

- The leader is similar to any entity.
- The followers maintain a reference to their leader and follow its decisions.

However, the leader cannot reuse the exact same navigation process as an independent entity. Its navigation must take into account the bulk of the whole group as well as the different locomotion constraints of its followers. It is also better to differentiate the leader's own attributes (position, orientation and velocity) from the group's [Millington 06]. Taking all these constraints into account makes the decision-making process of the leader very different from those of the other members.

20.4.2 Virtual Group Entity

Noting that the leader-based approach has several flaws, a growing proportion of architectures chose to move the group "anchor" from the leader to a virtual group entity [Karamouzas 10, Schuerman 10, Silveira 08]. This virtual entity is similar to any other simulated entity but does not have a visual or physical representation. In such an architecture, the group members are identical to one another. The group entity creates a one-level-deep hierarchy of entities. This approach can be taken a step further to create groups of groups and so on [Millington 06, Schuerman 10], allowing a more structured crowd.

Such hierarchical separation of responsibility leads to a cleaner software architecture as well as arguably simpler behaviors, but it is also slightly more complex to implement. In the following section, we'll describe the design choices we made when doing this.

20.4.3 Hierarchical Entity Architecture

In our architecture, we build upon the virtual group entity approach to create a hierarchy of entities (see Figure 20.6). Everything is an entity and is handled in the same way in our navigation loop; groups are composites of entities.

This hierarchy allows us to differentiate the group from the individual. An individual is the most basic entity we can have in our simulation. Groups, on the other hand, are entities containing other entities. It is a fairly standard implementation of a composite pattern.

Navigation behavior algorithms need information about the entity they are working on (position, velocity, orientation, etc.). They could take these from the entity, but the situation is more complicated when working with groups, because a group's properties depend on its entities. The way to define this relationship can be tricky to get right; here are the key ideas:

- The group's position can be computed from the members as their barycenter.
- Its bulk can also be computed either as a radius or as an oriented bounding box.
- Its orientation is tricky to define from the members; the best course of action is to tie it to the group's velocity or to have specific navigation behaviors handle the group's rotation [Millington 06].
- Its maximum speed, acceleration, and other movement limits need to be computed from the entities so that they are able to follow the group. For instance, the maximum speed of the group should be below the smallest of the members' maximum speeds. It is also important to consider that the maximum rotation rate of the group needs to take into account the maximum speed of its members and the width of the group.
- Finally, its velocity is independent, as we want the entities to "follow" the group.

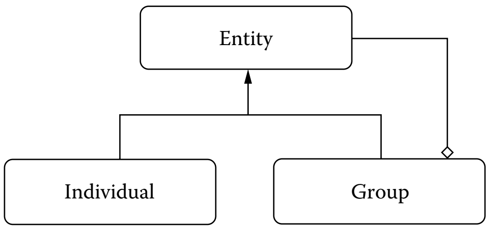

Figure 20.6

Entity hierarchy.

As we mentioned, a navigation behavior only relies on its "orders," the state of the world, and the state of the following entities:

- The one it is working on
- Its parent and/or children, used, for example, by formation slots assignment (discussed later)
- Its geometrical neighbors, used, for example, by collision avoidance

This means that it is easy to make the entities' navigation processes completely independent from one another by keeping the previous simulation update state as a read-only input. Thus, allowing easy multithreading.

One drawback is that the entity hierarchy has to be static from the point of view of the navigation behaviors. In other words, a navigation behavior cannot split or merge groups. The preferred approach to control groups' creation and membership changes is to treat the group hierarchy as an external parameter akin to a path planning target. A higher-level control layer is in charge of organizing groups; the navigation behavior should be resilient to these changes when they occur between updates.

This architecture can be used to create hierarchies with more than one level. This allows a complex structure and choreography for groups of entities with no actual additional cost.

One pitfall can be observed in deep hierarchies, however. Group members only take into account orders computed by the group during the previous simulation update, thus introducing a tiny delay. When adding layers of hierarchy, the delay grows linearly with its depth. We believe that this is not a real-world problem as a deep hierarchy does not have many use cases.

20.5 Pathfinding

One of the reasons to introduce group navigation is to factorize a costly aspect of navigation: pathfinding. As the members of a group are expected to follow the same high-level path through the environment, a single query should be sufficient for the whole group.

The most important aspect of group-level path planning is to choose how to take the bulk of the group into account. Contrary to a single entity where its bulk is static and thus is a hard constraint, a group may be able to reconfigure itself in order to pass through narrower corridors.

Therefore, the query has to be tuned in order to

- Prefer paths on which the group, in its current spatial configuration, can navigate
- Allow the use of narrower passages, for which the group can be reconfigured, if necessary

This means that the cost of falling back to a narrower spatial configuration needs to be comparable to the cost of taking a longer path [Bayazit 03, Kamphuis 04, Pottinger 99].

Once the path is computed, the path-following process provides local steering orders resulting in the entity following the path. In some works, the group-level path-following computation is also responsible for environment-aware formation adaptation, allowing the formation to change when the clearance to obstacles changes [Bayazit 03, Pottinger 99].

20.6 Emergent Group Structure

In most modern navigation engines, the simulated entities are autonomous, with their behavior relying on local "perception" to take action, not on an external choreographer. With this approach in mind, it is possible to design decentralized navigation behaviors to comply with group constraints.

20.6.1 Boids and Derivatives

At the core of Reynolds' work [Reynolds 87, Reynolds 99], three steering forces allow entities to flock. For a given entity in the group, separation makes it move away from close neighbors, alignment makes it go in the same direction as other members, and cohesion makes it move toward the group's anchor. The combination of these simple forces allows the emergence of a simple flocking behavior.

Given the nature of this model, it is simple to add new forces or to change the relative importance of forces (e.g., more or less repulsion) to better control the structure of the group. One example of such adaptation is the addition of a force modeling the desire for members of small social groups to keep all group members in their field of view for communication purposes [Moussaïd 10]. Another example is the modulation of members' attractivity to better take into account social relations [Qiu 10].

20.6.2 "Local" Formations

With the same strictly decentralized approach and by taking inspiration from molecular crystals, some formation control can be applied using an attachment site method. Each entity defines several attachment sites indicating, relatively, where its neighbors are supposed to be. When navigating, group members locate the nearest available site among their neighbors' and steer toward it.

The resulting formation arrangement is a direct result of the attachment sites position and it can scale to any number of group members. But, as the attachment rules are local, no control on the overall shape is possible; it is a good fit, though, for modeling social groups [Balch 00].

20.6.3 Implementing an Emergent Group

To get an idea of how such an emergent group structure can be implemented using our hierarchical architecture (see Figure 20.7), let us consider Boids' flocking behavior. In the by-the-book approach, given an initial velocity, the group will move cohesively in some direction. But, an adaptation is needed to control the group's movement.

Usually, a special entity is added: orders are given (e.g., a path to follow) to this leader, who "drags" the rest of the group around. Using our approach, no physical leader is needed. The group entity is the high-level order recipient and executor, and the group members use its position and velocity as an input for their cohesion and alignment behaviors.

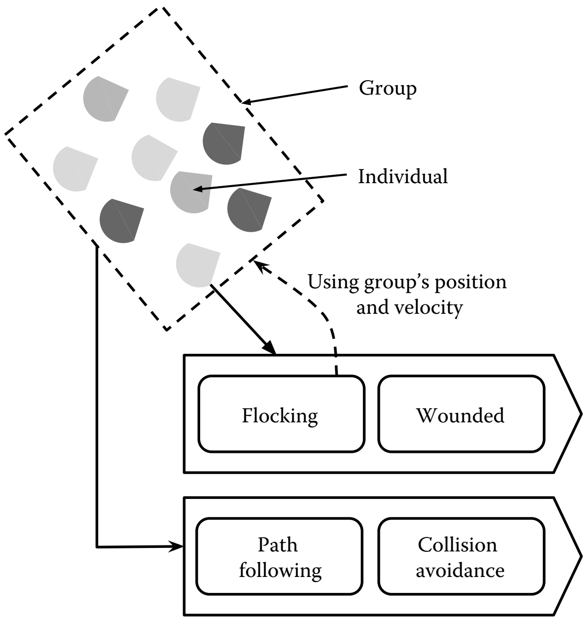

Figure 20.7

Flock architecture in our hierarchical group architecture.

The algorithm unfolds as follows during each update (the order of these is not important):

- The group updates its position and current velocity based on its members and then computes a new velocity based on a given path.
- Group members compute their new velocity based on the group's position (cohesion), the group's velocity (alignment), and the other members' relative positions (separation).

20.7 Choreographed Formations

While groups whose members are implementing local rules can exhibit convincing behavior, they cannot take into account the group as a whole and thus are not fully controllable. If exact group arrangement is needed, some of the behavior must be delegated to a higher level of control [Musse 01]. In this section, we will study the three steps needed to make a group stay in a given formation: formation design, slot assignment, and formation following.

20.7.1 Formation Design

In the context of navigation, a formation is the specification of the spatial arrangement of the members of a group. As we focus on pedestrians walking on the ground, each slot of the specification has a 2D position; two properties might be added, an orientation and a role (i.e., which kind of entity should be assigned to each slot). The slots are defined relative to the group's own position and orientation. The slots specification can come from different sources for different use cases, such as military doctrine, artistic choices, or even survey results.

The number of slots should match the number of entities in the group. If not, simple techniques can be used to select the used slots or create needed slots [Silveira 08].

20.7.2 Slots Assignment

Before our entities can navigate as a group, each of them must be assigned slot. This might seem trivial but should be implemented with care to avoid congestion between members of the same group; this will affect the credibility of the simulation. The greedy approach of each member being assigned the closest slot doesn't always work: the entities might have to cross each other's paths and the last entities might have to circle around the group to get to their slots [Dawson 02, Millington 06].

The best solution would be to globally minimize the distance the entities are covering to get to their slots but its implementation would lead to an $O(n!)$ complexity as every permutation would have to be tested.

One solution works well when no specialized slots are defined: The general idea is to sort the slots spatially then sort the members in the same way and assign the ith entity to the ith slot [Mars 14].

20.7.3 "Blind" Formation Following

The most basic approach to formation following is to have members no longer be responsible for their steering: members are placed on relative coordinates around the group's position [Pottinger 99]. This solution is fine if the group steering is robust.

Implementing this approach using our architecture is straightforward:

- The group updates its position and current velocity based on its members and then computes a new velocity based, for example, on a given path. Finally, it assigns a slot to each group member. It is also possible and often desirable to extend the group's navigation behavior with collision avoidance.
- Group members retrieve their slots and set their position accordingly.

One potential evolution of this approach is to assign group members a simple behavior that can compute and apply the necessary velocities for reaching their slot's position.

This makes it possible to customize the velocity application phase, taking into account characteristics such as maximum speed or acceleration or delegating it to an external system (e.g., locomotion).

When using this strategy, it is important to extrapolate the slot position to make it nonreachable in a single simulation update. This will contribute to avoid motion jolts [Karamouzas 10, Schuerman 10]. In practice, a simple extrapolation of the slot position using the group velocity over a time period greater than the frame duration is enough. This computation also handles gracefully nonmoving groups, as their velocity is null.

Additionally, the extrapolation time period can be controlled to define the "cohesion" of the formation, a small value for a tight formation a larger one for a very "loose" formation. The farther the target is, the less it will impact the member velocity.

20.7.4 Autonomous Formation Following

In most instances, members of a formation do not follow orders blindly. Instead, they have an autonomous strategy to stay in formation. This is especially true when simulating small social groups, where the formation is more of an emergent feature than a strict rule. Furthermore, it allows entities to break formation to pass through tight corridors and around small obstacles [Silveira 08].

This use case is where our architecture shines. The same strategy as before can be applied and, to enhance the individuality of the members, their behavior can be extended with (as shown in Figure 20.8)

- Collision avoidance, so that groups do not have to micromanage everything to avoid collisions between their members
- Specific behaviors, allowing entities to have "subgoals," for example, attraction to store fronts
- Specific velocity noise functions, to give them "personality"

While the same collision avoidance behaviors can be used by the entities whether they are part of a group or not, they must be adapted. As a matter of fact, collision avoidance algorithms, such as Reciprocal Velocity Obstacle [van den Berg 08], try to enforce a safe distance to obstacles and other entities that might forbid close formations [Schuerman 10].

To mitigate this issue, a member's behavior needs to either differentiate between its peers (other members of the group) and the other entities or to be adapted when it is part of a group by, for example, only considering imminent collisions.

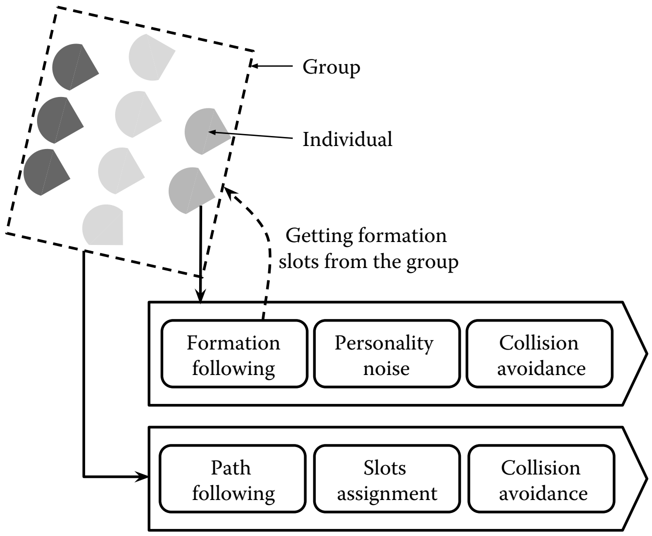

Figure 20.8

Autonomous formation following.

20.8 Group Collision Avoidance

In real life, groups tend to stay coherent when navigating between obstacles and among other pedestrians, which is why it is interesting to use group-level collision avoidance. Many existing algorithms for entities can be applied directly or adapted for group-level collision avoidance. As we noted earlier, the main difference between groups and single entities is that their bulk is not a hard constraint. The spatial configuration of a group can be adapted to occupy less frontal space, less longitudinal space, or both.

20.8.1 Velocity Correction

Existing collision algorithms such as RVO can be applied by considering the bulk of the group as a disc. The resulting collision avoidance is very conservative as the disc is, most of time, greatly overestimating the real footprint of the group [Schuerman 10, van den Berg 08].

To get better results, navigation behaviors of this family can be adapted to reason on the group's oriented bounding box [Karamouzas 04, Karamouzas 10, Peters 09].

20.8.2 Formation Adaptation

As discussed for path following, groups can spatially reconfigure themselves to change their bulk; this idea can be applied for better collision avoidance.

Consider a couple, walking side by side in a corridor: a simple formation. When another pedestrian arrives in the opposite direction, the couple will form a lane, reducing their frontal bulk, allowing the crossing. This is an instance of formation adaptation.

In RVO-like collision avoidance algorithms, several candidate velocities are computed around the current velocity, the ones leading to future collisions are pruned and the remaining one closest to the desired velocity is kept. The same approach can be used for formation adaptation [Karamouzas 10]:

- Define a set of formations the group can use and its preferred one (cf. Section 20.2.3 for social groups).
- At each time step, interpolate a number of candidate formations from the group's current state to the formations of the initial set.
- For each candidate formation, do an RVO-like evaluation outputting its "best" velocity and time to collision.
- Compute a cost for each candidate that take into account those values and the distance to the preferred formation.
- Take the lowest cost.

It is important to limit the number of candidate formations to preserve the performance of the algorithm. The original work uses a set of five formations and interpolates three candidates to each one of them, thus evaluating 15 in total.

Those group-level navigation methods allow the group to take responsibility for a part of the collision avoidance and more easily preserve the group cohesion. They can be easily implemented in our architecture as group behaviors and combined with finer granularity entity level steering.

20.9 Conclusion

In this chapter, we introduced a hierarchical architecture for group navigation, and we have shown how it can be used to fulfill different use cases, flocks, formations, and social groups, leveraging existing work. We proposed a generic framework to design and implement group navigation. A similar architecture was already implemented as a part of the Golaem SDK [GolaemSDK 14] and it is our plan to implement it in the open source navigation engine Recast/Detour [RecastDetour 14].

Externalizing some of the tricky collaborative decision making to a virtual group entity is one of the major design choices we made. Such "choreographer" entities are also a good pattern to apply when a high degree of control is needed over a group of individuals: traffic management around a door, group discussions, tactical synchronization, combat pacing, etc. Moreover, as we have shown in the context of navigation, this centralized decision-making method does not come at the cost of the individuality of each entity's behaviors.

References

[Balch 00] Balch, T. and Hybinette, M. 2000. Social potentials for scalable multi-robot formations. In *IEEE International Conference on Robotics and Automation*, San Francisco, CA, pp. 73–80.

[Bayazit 03] Bayazit, O., Lien, J., and Amato, N. 2003. Better group behaviors in complex environments using global roadmaps. In *Eighth International Conference on Artificial Life*, Department of Computer Science, Texas A&M University, College Station, TX, pp. 362–370.

[Dawson 02] Dawson, C. 2002. Formations. In *AI Game Programming Wisdom*, ed. Rabin, S., pp. 272–282. Charles River Media, Hingham, MA.

[GolaemSDK 14] Golaem SDK. 2014. Available from: http://golaem.com/ (accessed July 10, 2014).

[Kamphuis 04] Kamphuis, A. and Overmars, M.H. 2004. Finding paths for coherent groups using clearance. In *ACM SIGGRAPH/Eurographics Symposium on Computer Animation*, Copenhagen, Denmark, pp. 19–28.

[Karamouzas 04] Karamouzas, I. and Overmars, M. 2004. Simulating human collision avoidance using a velocity-based approach. In *VRI-PHYS 10: Seventh Workshop on Virtual Reality Interactions and Physical Simulations*, Eurographics Association, Copenhagen, Denmark, pp. 125–134.

[Karamouzas 10] Karamouzas, I. and Overmars, M. 2010. Simulating the local behaviour of small pedestrian groups. In *17th ACM Symposium on Virtual Reality Software and Technology*, Hong Kong, China. Center for Advanced Gaming and Simulation, Utrecht University, Utrecht, the Netherlands, pp. 183–190.

[Loscos 03] Loscos, C., Marchal, D., and Meyer, A. 2003. Intuitive crowd behaviour in dense urban environments using local laws. In *Proceedings of the Theory and Practice of Computer Graphics*, Manchester, U.K., p. 122.

[Mars 14] Mars, C. 2014. Simple formation assignment. *GDC 2014 AI Summit,* San Francisco, CA.

[Millington 06] Millington, I. 2006. *Artificial Intelligence for Games*, pp. 41–202. Morgan Kaufmann, San Francisco, CA.

[Mononen 10] Mononen, M. 2010. Navigation loop. In *Paris Game/AI Conference 2010*, Paris, France.

[Moussaïd 10] Moussaïd, M., Perozo, N., Garnier, S., Helbing, D., and Theraulaz, G. April 2010. The walking behaviour of pedestrian social groups and its impact on crowd dynamics. *PLoS ONE*, 5(4):e10047.

[Musse 01] Musse, S. and Thalmann, D. 2001. Hierarchical model for real time simulation of virtual human crowds. *Transactions on Visualization and Computer Graphics*, 7(2):152–164.

[Peters 09] Peters, C., Ennis, C., and O'Sullivan, C. 2009. Modeling groups of plausible virtual pedestrians. *IEEE Computer Graphics and Applications*, 29(4):54–63.

[Pottinger 99] Pottinger, D. January 1999. Implementing coordinated movement. Available from: http://www.gamasutra.com/view/feature/3314/implementing_coordinated_movement.php?print=1 (accessed May 21, 2014).

[Qiu 10] Qiu, F. and Hu, X. 2010. Modeling dynamic groups for agent-based pedestrian crowd simulations. In *IEEE/WIC/ACM International Conference on Web Intelligence and Intelligent Agent Technology*, Toronto, Canada, pp. 461–464.

[RecastDetour 14] Recast/Detour. 2014. Available from: https://github.com/memononen/recastnavigation (accessed July 10, 2014) and https://github.com/masagroup/recastdetour (accessed July 10, 2014).

[Reynolds 87] Reynolds, C. 1987. Flocks, herds and schools: A distributed behavioral model. In *ACM SIGGRAPH '87 Conference Proceedings,* Anaheim, CA, pp. 25–34.

[Reynolds 99] Reynolds, C. 1999. Steering behaviors for autonomous characters. In *Proceedings of Game Developers Conference.* Miller Freeman Game Group, San Francisco, CA, pp. 763–782.

[Schuerman 10] Schuerman, M., Singh, S., Kapadia, M., and Faloutsos, P. 2010. Situation agents: Agent-based externalized steering logic. In *International Conference on Computer Animation and Social Agents*, University of California, Los Angeles, CA.

[Silveira 08] Silveira, R., Prestes, E., and Nedel, L. 2008. Managing coherent groups. *Computer Animation and Virtual Worlds*, 19(3–4):295–305.

[van den Berg 08] van den Berg, J., Lin, M., and Manocha, D. 2008. Reciprocal velocity obstacles for real-time multi-agent navigation. In *International Conference on Robotics and Automation*, Pasadena, CA, pp. 1928–1935.