Situation Agents: Agent-based Externalized Steering Logic

Matthew Schuerman Shawn Singh Mubbasir Kapadia Petros Faloutsos

{mschuerm,shawnsin,mubbasir,pfal}@cs.ucla.edu

University of California, Los Angeles

Abstract

We present a simple and intuitive method for encapsulating part of agents' steering and coordinating abilities into a new class of agents, called *situation agents*. Situation agents have all the abilities of typical agents. In addition, they can influence the steering decisions of any agent, including other situation agents, within their sphere of influence. Encapsulating steering logic into moving agents is a powerful abstraction which provides more flexibility and efficiency than traditional informed environment approaches, and works with many of the current steering methodologies. We demonstrate our proposed approach in a number of challenging scenarios.

Keywords: crowd simulation, steering, group behaviors, situation agents

Introduction

Crowd simulation has become a topic of increasing interest recently. Aside from the obvious entertainment applications, it has implications for fields such as urban planning, architecture, and law enforcement. One of the most challenging aspects of crowd simulations is obtaining the performance and simplicity required to make them a practical tool, which often involves trading agent sophistication for speed.

Many agent-based steering methods exhibit good performance and realistic agent behavior in crowds of reasonable density in open areas, but begin to have trouble in situations where body language or social customs become the dominant factor in human steering decisions. In such cases, many steering methods are prone to exhibit deadlocks or unnatural behaviors. To address this, agents are often given enough sophistication to be able to recognize such situations (or annotations in the environment) and attempt to resolve them. This can either be done using very robust planning techniques or by adding specialized logic to the agent's basic steering algorithm – neither of which is an ideal solution. More robust steering techniques often reduce performance, especially when used by many agents. Similarly, augmenting the agent's standard steering methods to deal with specialized situations can lead to extremely complex steering algorithms which attempt to cope with a wide variety of special cases. Both approaches result in polluting agents with additional computation and storage costs in order to deal with situations they might not even encounter.

A more effective solution is to externalize the specialized steering logic that deals with

troubling cases to some third party and allow agents to use the most fundamental form of their steering algorithm for their navigational needs. The key idea of this paper is to use a new class of agents, called *situation agents*, as this third party. Situations agents are similar to standard agents in many regards: they have a defined location, size, velocity, and goal. They can utilize steering strategies to interact with other situation agents. However, they differ from standard agents in that they exist solely to supervise the actions of other agents in localized scenarios that their typical steering algorithms might not support.

Situation agents provide all the advantages of traditional informed environment approaches [1, 2, 3], as well as further benefits. Since specialized logic is being removed from standard agents, modularity and performance are enhanced using either approach. However, using *agents* as couriers for specialized logic provides several advantages over traditional informed environment approaches:

1. For reasons previously stated, environmental annotation does not always significantly reduce the standard agent's complexity. Situation agents encapsulate the logic for both the detection and resolution of difficult situations in a different, and far less numerous, population of agents. Also, when influencing multiple agents, situation agents can implement such logic with a broader view than individual agents.

2. Smart objects and areas [1] are better at segmenting complexity than environmental annotations, but like annotations, are often static. Situation agents give specialized steering behaviors intuitive ways to change the area they affect through agent re-sizing

and movement, which can become important in heterogeneous or dynamic environments.

3. Encapsulating behaviors into agents provides a powerful interface for making hierarchical decisions. Broadly scoped situation agents can choose the correct aggregate action for many agents, and then individuals revise those suggestions as required. Since situation agents are agents themselves, arbitrarily many layers of them can be used in this fashion. This provides a natural paradigm for behaviors such as steering, where groups often attempt to steer within larger groups. Also, because of this hierarchy, macroscopic crowd behavior becomes a controllable option, not an emergent property of an algorithm. This is key to the results presented in this paper.

Contribution: This paper proposes the concept of situation agents, which allow specialized steering algorithms to be externalized from standard agents to a separate class of agents. Such separation allows us to enhance agent-based steering algorithms to exhibit correct behavior in challenging situations with minimal additional processing complexity. Situation agents are also highly customizable and localized, and offer a simple, intuitive, and efficient way for animators and programmers to orchestrate crowd behaviors on a fine scale. We illustrate this with three specific examples: deadlock agents, groups agents, and formation agents. Our results show agent cooperation to resolve difficult scenarios, an important and challenging aspect of realistic agent behavior.

The rest of this paper is organized as follows: Section 2 gives a brief summary of related

literature, Section 3 explains some example situation agents in depth, Section 4 outlines our results, and finally Section 5 concludes and discusses future plans.

Related Work

Since Reynolds' pioneering work [4], crowd simulation has become a field of active and diverse research. The reader is referred to [5] for an extensive summary of this work. These techniques draw inspiration from a wide variety of sources including fluid dynamics [6], physics [7], robotics [8], motion capture technologies [9], and artificial life [10] and thus vary significantly.

However, common themes do exist in all crowd simulation frameworks. The problem of maintaining realistic behavior in all cases while minimizing the processing per agent has always been a paramount concern. Externalizing portions of agents' complexity is one solution to this problem, and it is the one we will focus on here. Several frameworks [1, 2] include both smart areas and smart objects which can dictate how an agent can interact with them. The work in [3] extends this notion to allow the environment to add or remove behaviors from agents. However, we are not aware of any previous attempts to externalize steering behaviors to a distinct set of *agents*. Composite agents [11] did allow some agents to instantiate and re-size a set of passive agents –proxy agents– to act as obstacles to aid in navigation. However, proxy agents do not contain any specialized logic or information, making that methodology quite different from the one presented here.

Situation Agents

Situation agents coordinate the behaviors of other agents by modifying those agent's parameters. In the their most general form they can modify any parameter within any agent, including other situation agents. Fully leveraged, this allows them to produce arbitrarily complex hierarchies and behaviors from relatively simple and modular components. However, for demonstration purposes the situation agents presented in this paper modify only two internal parameters, *boldness* and *preferred velocity,* and typically use only a single layer of hierarchy. This compact interface makes them compatible with a wide variety of simulation frameworks. We will discuss how these parameters are implemented in the context of the continuum crowds [6] (CC) and reciprocal velocity obstacles [12] (RVO) frameworks. These popular frameworks employ quite different methodologies, and using them as examples demonstrates the portability of our approach.

Situation agents make their changes *before* the agents they influence compute their actions. This allows the influenced agents to use their own steering or behavior logic to carry out the suggestions of the situation agents and creates the hierarchy which is one of the primary advantages of this technique. For the CC framework these modifications would take place as the unit cost field is being calculated. In RVO frameworks, situation agents would process their actions first and during that time they would modify the parameters of the agents they influence.

Boldness is how likely an agent is to yield to other agents during steering. An agent with

boldness parameters set high will exert less effort to avoid other agents, while an agent with low boldness parameters will act more timidly. By modifying boldness parameters, situation agents can order some agents to yield and allow others to pass. In the CC framework, boldness is varied by changing the length and intensity of the discomfort field projected in front of each agent: large and more intense discomfort fields cause other agents to move out that agent's way. As a result that agent is not required to as actively steer around others, effectively increasing their boldness. In RVO frameworks, boldness is implemented by modifying the safety factor and α parameter. Decreasing an agent's safety factor makes it less afraid of collisions, while reducing its α parameter decreases its responsibility for movement as two agents reciprocally avoid each other. Tuning these two parameters changes the boldness of individual RVO agents.

All the steering frameworks we are aware of calculate either a best path or best velocity for each agent which will most directly guide them toward their goal during each iteration. This is an agent's *preferred velocity*. Situation agents modify this to herd other agents when required. At each iteration the CC framework computes a best path based on the unit cost field. Adding a local perturbation to the path and speed fields effectively changes the best path and speed toward the goal for a CC agent. In the RVO framework the preferred velocity is directly calculated as a vector, typically the direction is toward the next way-point on a pre-computed path and the speed is the desired speed of the RVO agent.

Situation agents modify the preferred velocities of other agents, $\vec{v}_a^{pf}$, by replacing them with $\vec{v}_a^{pf\,new}$, a weighted average of a velocity of their own calculation, the correction veloc-

ity $\vec{v}_{corr}$, and the agent's original preferred velocity, as seen in Equation 1. For best path formulations there is an analogous method which involves adding a local perturbation to the distance field. In Equation 1, and for the rest of this paper, the subscript a implies an agent's parameter and the subscript sa implies a parameter associated with a situation agent.

$$\vec{v}_a^{\,pf\,new} = w_{sa}\vec{v}_{corr} + w_a\vec{v}_a^{\,pf}. \tag{1}$$

The sum of the weights w_{sa} and w_a used in the above linear combination should be 1. Using these weights, situation agents can choose how strongly to affect other agents' behavior.

Varying the steering behavior of situation agents, the weights in Equation 1, and the boldness parameters of individual agents can produce a wide variety of behaviors capable of addressing challenging crowd simulation scenarios. Specifically, we will examine how varying these options can make situation agents effective in the cases of deadlocks (no steering), coherent group behaviors (moderate weighting), and marching formations (extreme weighting).

Example: Deadlock Agents

Consider the case of two groups of agents moving in opposite directions arriving at a narrow hallway at the same time. The hallway is so narrow that only a single agent can pass through it at once. Clearly in such a case one of the two groups of agents must wait while the other one passes or a deadlock will arise. However, this can be hard to anticipate and coordinate among independent agents. In some frameworks cooperative planning has worked,

but this approach is computationally expensive and hard to scale up to more than a handful of individuals. To handle this coordination problem we introduce the deadlock situation agent. This type of agent can modify the preferred velocities of all the nearby agents in a centralized and efficient way to produce the required waiting behavior.

Deadlock agents can be placed dynamically by the user or statically by the framework. The environment is analyzed before the simulation begins to find regions so narrow that only a single agent can pass through at once. Deadlock agents are placed in the center of such regions. They are stationary and circular in shape with initial radii proportional to the longest side of the narrow region. Agents within these radii are influenced by the deadlock agents.

At each computation iteration deadlock agents perform the tasks outlined in Algorithm 1. They dynamically expand their size by looking slightly past their current radius ($r_{sa} + \epsilon$) and then updating their radius to encompass the furthest possible agent within a predefined limit. This allows them to grow to influence –and prevent deadlocks in– entire groups approaching a passage. They can also contract by a fixed ratio, λ, during each iteration if all nearby agents are already within their radius.

Deadlock agents operate by instructing agents not moving in a particular direction, the *favored direction*, to yield. Moving with or against the favored direction is determined by using γ as a threshold on the normalized dot product of an agent's preferred velocity and the favored direction. The favored direction is first chosen to be the preferred velocity of the first agent to arrive. So long as agents are currently moving in the direction it remains

Algorithm 1 Psuedocode for Deadlock Agents

$r_{detect} \leftarrow r_{sa} + \epsilon$

$r'_{sa} \leftarrow max(\lambda * r_{sa}, r_{sa}^{min})$

for each agent a within r_{detect} **do**

// p's represent the positions of the agents

$\vec{d}_a \leftarrow \vec{p}_a - \vec{p}_{sa}$

$r'_{sa} \leftarrow max(r'_{sa}, \|\vec{d}_a\|)$

if $(\vec{v}_a^{pf} \cdot \vec{v}_{fav})/(\|\vec{v}_a^{pf}\|\|\vec{v}_{fav}\|) \leq \gamma$ **then**

decrease boldness(a)

save possible favored direction$(\vec{v}_a^{pf})$

$\vec{v}_a^{pf} \leftarrow \vec{v}_a^{pf\,new}$

end if

end for

$r_{sa} \leftarrow min(r'_{sa}, r_{sa}^{max})$

update favored velocity$()$

unchanged. However, when the last such agent leaves, the preferred direction becomes the preferred velocity of the agent closest to the center of the deadlock agent.

Influenced agents are instructed to yield by reducing their boldness and modifying their preferred velocities using Equation 1 with the following correction velocity:

$$\vec{v}_{corr} = \|\vec{v}_a^{pf}\|(\cos\theta, \sin\theta), \tag{2}$$

where $\theta = \arctan(y_a - y_{sa}, x_a - x_{sa}) \pm 90°$. This causes the agents not moving in the favored direction to move aside so others can pass. The result of these modifications is that, as groups of agents arrive at a narrow doorway, one is allowed to pass while others are forced to step aside and wait. This continues until all the groups have passed. In the rare event that multiple traffic agents attempt to influence a single agent, only the correction velocity from the nearest traffic is used.

Example: Group Agents

Often a desirable property of crowd simulations is that groups of agents steer around each other coherently as a whole. Group situation agents are designed to implement such behavior. Once associated with a group of agents, group situation agents steer to avoid other group agents with different goals and linearly combine the resulting velocity with the preferred velocities of the group's members. As a result steering becomes hierarchical for the influenced agents: groups first attempt to avoid each other and then individual agents refine that group steering decision to suit their needs. This produces the desired cohesion without inhibiting

individual steering capabilities or adding additional complexity to standard agents.

Group situation agents should be instantiated around groups of agents moving toward the same goal at similar speeds. They are capable of dynamically re-sizing themselves in exactly the same manner as deadlock agents. However, they only expand to influence agents moving to the same goal as the rest of the agents in the group. Once instantiated, they act as agents moving toward the same goal as the agents in their group. They can use any steering algorithm, but we have found using a modified version of the standard agent's steering algorithm is simplest to implement. The required modification is to allow group agents to selectively ignore both standard agents and other group agents moving toward the same goal while steering. Note that this effectively allows group agents moving toward the same goal to overlap, which is required to effectively cover irregularly shaped groups of many individuals. This modification is compatible with all steering frameworks which we are aware of. The result of this steering, $\vec{v}_{sa}$, becomes $\vec{v}_{corr}$ in Equation 1, which dictates the new preferred velocities of the group members. Varying w_{sa} controls the degree of synchronization in the movements of the group members.

Group agent placement and membership can be the responsibility of the user, or it can be done in various automated ways. In our examples, we use the following greedy algorithm, which provides group coherence and effective lane formation. At every update each agent and its neighbors are examined to determine how many of them are moving toward similar locations in the near future, and that count is stored (this is already done by agents in most frameworks, so these counts are often already available). At the beginning of each update

those stored numbers are sorted, and group agents are created around the center agents of the largest potential groups. They begin to expand to encompass the group at the next iteration of computation. Group situation agents are also automatically removed when the group reaches a goal or if they are not being used. Specifically, if a group situation agent does not steer around other group agents for some set period of time or moves too far off the center of its group, it is removed.

Example: Formation Agents

Formation situation agents are used to force groups of agents to march in formation through crowds or open spaces. In many regards they are a special case of group agents: they utilize steering, but w_{sa} (Eq. 1) becomes 1, implying $w_a = 0$, and making the velocity correction a substitution for the agent's preferred velocity. Formation agents also provide additional functionality such as re-forming and rotation. These abilities are controlled by a set of parameters including shape, rigidity, and agent participation which are defined by the user.

In order to enforce their group's shape the boldness parameters of all the group members in a formation agent are increased. In crowds, this encourages other agents to avoid the agents maintaining the formation. This tendency is further enhanced by group situation agents steering to avoid formation agents as though they were extremely bold group agents. In return, formation agents treat group agents as timid formation agents. So on both large and fine scales, agents are encouraged to let formations pass with relative ease. The

correction velocity (Eq. 1) formation agents apply to their members is the sum of two terms,

$$\vec{v}_{corr} = \vec{v}_{sa} + \beta(\vec{p}_{sa} + \vec{\delta_a} - \vec{p}_a), \tag{3}$$

where δ_a is the offset from the center of the formation agent to the agent's position in the formation. As the orientation of the formation agent rotates, this offset is rotated as well, which allows formations to rotate as they turn. However, if static turning is desired, this rotation can be omitted. β is a scalar between $(0, 1]$ which controls how rigidly each member of the group holds their position. When $\beta = 1$ group members always attempt to reach their correct formation position exactly. When β is a lesser value members only attempt to move a fraction of the way to their formation position. Lowering this parameter essentially allows agents to take their time when restoring the formation. As in Algorithm 1, p_{sa} and p_a represent the positions of the situation agent and agent respectively.

Formation situation agents have the ability to break apart and re-form their formations. To break the formation the formation agent simply stops modifying the boldness and preferred velocities of its members. At that point the members behave as standard agents. However, re-forming is a more intricate process. To account for the possibility that the formation members have drastically re-ordered themselves, the formation agent reassigns the positions of the formation. This is done by rotating the positions of the members into a coordinate system aligned with the orientation of the formation agent and centered at its position. Both the transformed member positions and the offsets are then sorted by y-value (the x-value is used to break ties) and paired by order. The result is that nearby member agents are assigned

positions near the front of the formation and further agents are given positions in the back. This re-ordering takes place exactly once when a formation agent musters its members. After that the individual members use their standard behaviors to reach their positions within the offset. To facilitate this, upon re-forming the formation agent slows down until all its members have gotten sufficiently close to their positions within the formation. After the formation has re-formed the formation situation agent returns to normal speed. This is how formation agents move through a narrow passage: they break the formation when entering it, both them and their member agents pass through, and then they re-form when exiting the passage.

Results

Although the notion of situation agents is compatible with many crowd simulation frameworks, for this paper we focus on its application to RVO-based steering. Recent studies [13] have shown that variants of this method of agent steering can operate at high frame rates for large numbers of agents. However, at present the RVO framework has no way for agents to coordinate specialized group interactions. Steering behavior is calculated independently for each agent, which can lead to undesired behaviors in certain scenarios. Situation agents are a well-suited solution to this coordination issue, making RVO an ideal steering algorithm to test them with.

The following sections outline the results of the three types of situation agents presented

previously as they apply to RVO steering. During testing we based our code on RVO library 1.1 [14] and used the A* algorithm [15] as our path planner. Steersuite 1.02 [16] was used for visualization and as an inspiration for test cases. The examples presented in this section are not exhaustive. Additional scenarios are included in the supplementary video.

Deadlocks

Figure 1 shows a narrow doorway with two sets of agents arriving at similar times. Using standard RVO agents a deadlock occurs. By placing a deadlock situation agent at the center of the narrow doorway the deadlock can be avoided. The boldness parameters and preferred velocities of the agents moving right are modified to encourage them to move to the side while the agents moving left pass. Once the agents pass the original parameters of the remaining agents are restored.

Using deadlock agents eliminates the threat of deadlocks in many situations. They also give the artist or programmer flexibility: agents can line up on either side of the wall and their aggressiveness can be calibrated. Also, because this behavior is simple and predefined, it does not significantly reduce performance as cooperative planning schemes might.

Group Behaviors

RVO agents do produce lanes in the scenario of two small groups passing through each other shown in top sequence of images in Figure 3. However, these lanes of agents are

rather thin and interleaved. Often it would be desirable for the groups to remain coherent and steer around each other completely. By applying group agents using the automated process described earlier we see this result in the lower sequence of images in Figure 3. The same behavior is evident when many group agents (the large agents in blue) are assigned to large groups as shown in Figure 2a.

Group agents can resolve deadlocks as well. Consider the case of two groups of 2 agents walking toward opposite ends of a hallway that is wide enough for 2 agents to walk abreast. If a group agent is placed on each pair it encourages the agents to form lanes which allows the groups to cross as shown in Figure 4. However, without such encouragement agents can end up deadlocked, as we have observed using RVO agents. Our technique works for larger groups of agents in the same hallway scenario as well.

Formations

Figure 5 illustrates a user-specified triangle formation. As the formation moves through a large group, other group agents (in blue) steer away from the formation agent (in green). On a finer level the individual RVO agents also steer away from the agents within the formation. As a result the formation can maintain its shape. Exactly how well it maintains its shape can be controlled by parameters. If we reduce β to 0.5 the group becomes less rigid and the sides begin to tuck behind the main portion of the triangle as seen in Figure 2b.

If the formation encounters a space too narrow, it must detect that and act accordingly.

Once the formation agent detects an obstacle within its range (Figure 6) it allows the member agents to break form. The agents move through the passage individually and re-form once the formation agent no longer intersects with the walls of the passage.

Formation agents are particularly useful for RVO steering because RVOs have no built-in mechanism for enforcing rigid group movement. But even in frameworks with such features, formation agents are useful for their ability to determine when marching in formation is feasible. They are also useful because they allow all the agents within the formation to be treated as a single agent. This allows for hierarchical structures such as the formation of formations shown in Figure 2c. Note that hierarchy is possible *between* agent classes as well: a traffic agent influencing formation agents would be an equally valid example.

Conclusions and Future Work

We have presented an approach for augmenting steering frameworks with externalized hierarchical steering logic. Our approach reduces the complexity of standard agents by externalizing the specialized steering logic required for some situations to a new class of agents called situation agents. These agents exist solely to influence the agents within their vicinity by modifying their preferred velocities and boldness parameters (other parameters can be modified as well in the general case). Through this influence situation agents orchestrate other agents to successfully navigate difficult situations. Additionally, the hierarchy that situation agents provide often enhances the elegance and simplicity of the solutions they

facilitate. Examples of this were given for scenarios involving possible deadlocks, coherent group behavior, and enforcing marching formations.

However, the true strength of situation agents is not the specific behaviors demonstrated in this paper, but the modularity they provide. Since the logic to deal with challenging scenarios has been externalized, standard agents can use much simpler steering algorithms, enhancing both performance and maintainability. The specialized steering algorithms also benefit from this separation. Encapsulating them into individual situation agents provides a clean interface for integrating them into frameworks and provides the separation required for parallel development. Also, once developed, artists have an intuitive way to select when, where, and which algorithm to use. For example [17, 18, 19] provide more intricate methods of group management than the current group agent. Situation agents could be made to implement each of those algorithms and artists could use them as appropriate. None of the existing standard or situation agents would require modification, the only thing required would be new situation agents. This would not be feasible with traditional informed environment approaches. It is only because of the additional flexibility of allowing specialized behaviors to *move* as agents that such encapsulation and separation is possible.

For future work we would like to more thoroughly test the concept of situation agents using steering frameworks other than RVO. Also, more types of situation agents should be explored. In the long term, this method could be added to game engines and animation suites to aid artists and developers in producing realistic crowds.

References

[1] N. Farenc, S. Raupp Musse, E. Schweiss, M. Kallmann, O. Aune, R. Boulic, and D. Thalmann. One step towards virtual human management for urban environment simulation. In *Proceedings of the ECAI Workshop on Intelligent User Interfaces*, 1998.

[2] Franco Tecchia, Cèline Loscos, Ruth Conroy, and Yiorgos Chrysanthou. Agent behaviour simulator (abs): A platform for urban behaviour development. In *GTEC*, pages 17–21, 2001.

[3] Mankyu Sung, Michael Gleicher, and Stephen Chenney. Scalable behaviors for crowd simulation. *Computer Graphics Forum*, 23:519–528(10), Sept 2004.

[4] Craig W. Reynolds. Flocks, herds and schools: A distributed behavioral model. In *SIGGRAPH*, pages 25–34, 1987.

[5] Norman Badler. *Virtual Crowds: Methods, Simulation, and Control (Synthesis Lectures on Computer Graphics and Animation)*. Morgan and Claypool Publishers, 2008.

[6] Adrien Treuille, Seth Cooper, and Zoran Popović. Continuum crowds. *ACM Trans. Graph.*, 25(3):1160–1168, 2006.

[7] Dirk Helbing and Peter Molnar. Social force model for pedestrian dynamics. *Physical Review E*, 51:4282, 1995.

[8] Paolo Fiorini and Zvi Shillert. Motion planning in dynamic environments using velocity obstacles. *International Journal of Robotics Research*, 17:760–772, 1998.

[9] S. Paris, J. Pettre, and S. Donikian. Pedestrian reactive navigation for crowd simulation: a predictive approach. In *EUROGRAPHICS*, pages 665–674, 2007.

[10] Wei Shao and Demetri Terzopoulos. Autonomous pedestrians. In *SCA*, pages 19–28, 2005.

[11] Hengchin Yeh, Sean Curtis, Sachin Patil, Jur van den Berg, Dinesh Manocha, and Ming Lin. Composite agents. In *SCA*, pages 39–48, 2008.

[12] J. van den Berg, Ming Lin, and D. Manocha. Reciprocal velocity obstacles for real-time multi-agent navigation. In *ICRA*, pages 1928–1935, 2008.

[13] Stephen J. Guy, Jatin Chhugani, Changkyu Kim, Nadathur Satish, Ming Lin, Dinesh Manocha, and Pradeep Dubey. Clearpath: highly parallel collision avoidance for multi-agent simulation. In *SCA*, pages 177–187, 2009.

[14] Jur van den Berg. RVO library 1.1, Sept 2009. www.cs.unc.edu/∼geom/RVO/Library/.

[15] P. E. Hart, N. J. Nilsson, and B. Raphael. A formal basis for the heuristic determination of minimum cost paths. *IEE Transactions on Systems Science and Cybernetics*, 4(2):100–107, 1968.

[16] Shawn Singh. Steersuite 1.02, Sept 2009. www.magix.ucla.edu/steersuite/.

[17] Taesoo Kwon, Kang Hoon Lee, Jehee Lee, and Shigeo Takahashi. Group motion editing. *ACM Trans. Graph.*, 27(3):1–8, 2008.

[18] Yu-Chi Lai, Stephen Chenney, and ShaoHua Fan. Group motion graphs. In *SCA*, pages 281–290, 2005.

[19] Arno Kamphuis and Mark H. Overmars. Motion planning for coherent groups of entities. In *ICRA*, pages 3815–3822. Press, 2003.

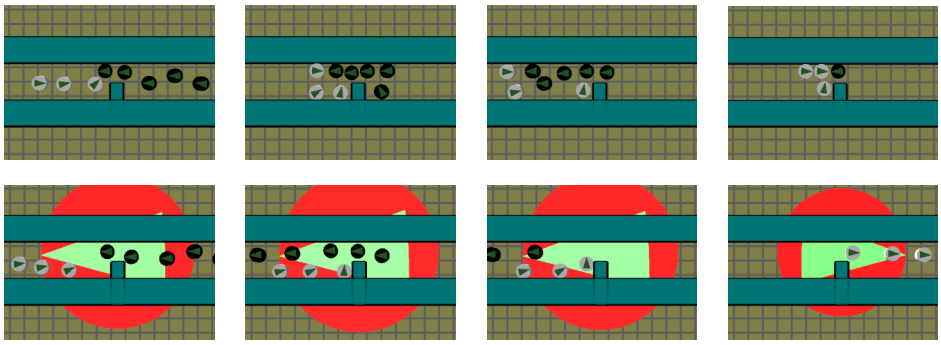
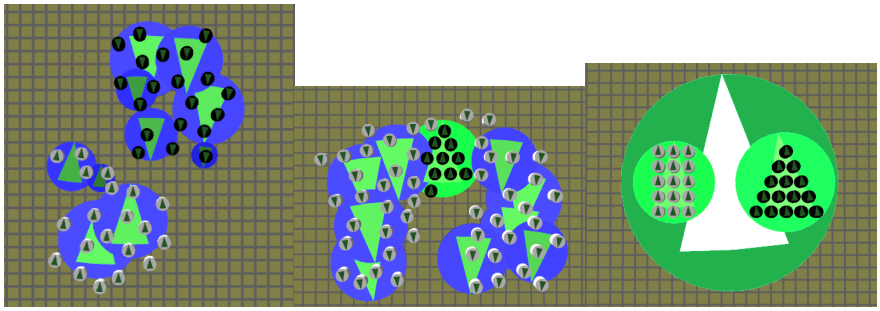


Figure 1: (Left to right) Agents are placed in random locations on both sides of a narrow doorway. Their goals are to cross to the other side, as initially indicated by the arrow on each agent. The top sequence shows RVO agents attempting this and the resulting deadlock. In the bottom sequence a deadlock agent (in red) has been placed at the doorway in the same scenario and deadlocks are avoided. The upper sequence spans a period four times longer than the lower sequence due to agent shoving, which deadlock agents also eliminate.



(a) (b) (c)

Figure 2: Example situation agents

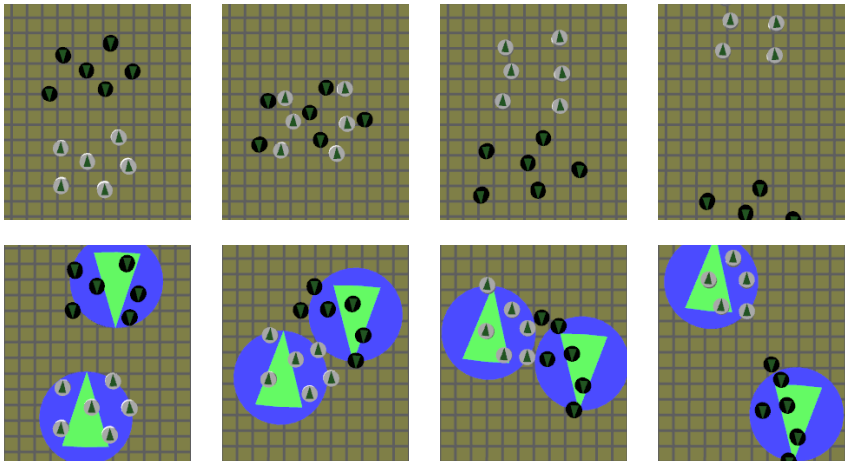


Figure 3: (Left to right) Two groups of 6 agents each are initially placed opposite each other with opposing trajectories. The top sequence shows the behavior of typical RVO agents. The bottom sequence shows the same groups of agents, but each has been assigned a group agent (in blue) using our automatic placement algorithm. The groups now steer as aggregates around each other.

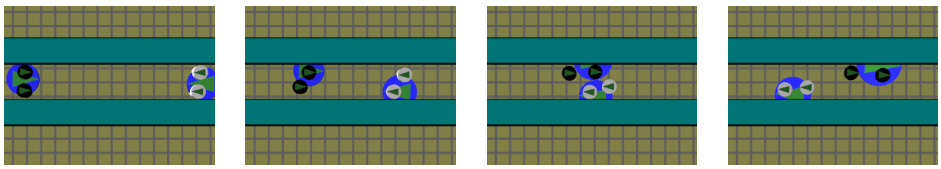


Figure 4: (Left to right) Two groups of 2 agents walking toward each other abreast in a narrow hallway. The group agents (in blue) assigned to each pair steer around each other and as a result encourage lane formation. This allows the pairs to pass each other smoothly.

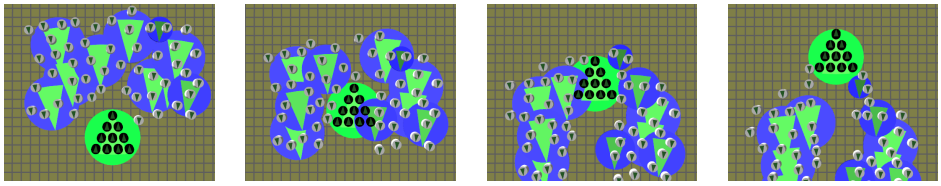


Figure 5: (Left to right) A formation agent (in green) regulates a group of agents moving through a crowd. The group agents (in blue) steer away from the formation agent, while individual agents steer away from the individual agents of the formation on a finer level. Due to this hierarchy, collisions between group agents and the formation agent do not disturb the formation.

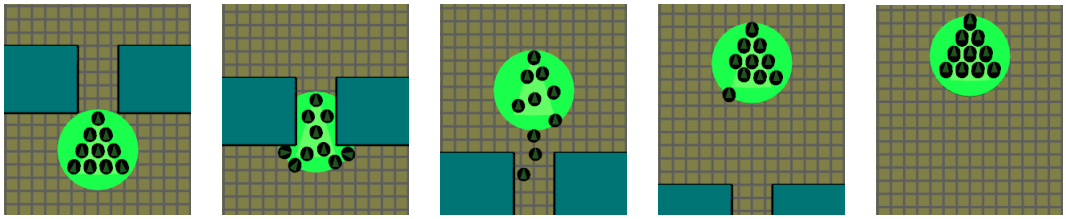


Figure 6: (Left to right) The formation agent (in green) detects that it intersects an obstacle and stops enforcing the formation. Both it and the agents in its formation proceed through the narrow passage. Once the formation agent no longer detects an intersection with an obstacle, it begins rebuilding the formation.