Integrating Run-Time Incidents in a Large-Scale Simulated Urban Environment

Steven de Jong*, Alex Klein, Ruben Smelik, and Freek van Wermeskerken
Department of Modeling, Simulation and Gaming, TNO, The Netherlands
* Corresponding author: steven.dejong@tno.nl

ABSTRACT

The domain of Defense, Safety & Security offers interesting applications and research challenges for agent-based simulation of human behavior in urban environments. For example, virtual training of military personnel often requires complex urban scenarios to align with the current nature of operations.

Previous research has focused on the simulation of a large population of civilians within an urban environment, introducing techniques such as pattern-of-life and activity generation. We have used this research as a basis for an approach that allows the injection of *incidents* (e.g., a crowd gathering, a shoplifting with a subsequent arrest) in a running, large-scale urban simulation. Such incidents can be used by an instructor during a training or exercise to teach a trainee to deal with unforeseen circumstances. The incidents are not explicitly scripted, but are generated by a planning algorithm within desired constraints such as what should happen, where, and when. We believe that this approach offers an intuitive and efficient way of specifying scenario-specific exceptions to daily behavior patterns in a representative urban setting.

We describe an extendable *prototype system* aimed at demonstrating the approach. A necessary basis for the prototype system is the generation and simulation of credible daily behavior, driven by relatively simple models. In the prototype, we simulate daily behavior in the Dutch municipality of Rijswijk, which houses approximately $30,000$ inhabitants. In this context, we are able to inject several different types of incidents that interrupt daily behavior. We aim to share this prototype system to encourage collaboration on this topic, with a particular focus on the challenges posed by applications within the Defense, Safety & Security domain.

Categories and Subject Descriptors

I.2 [**Distributed Artificial Intelligence**]: Multi-agent Systems

General Terms

Algorithms, Design, Experimentation, Human Factors

Keywords

Run-Time Incidents, Scenario Generation, Population Modeling, Urban Simulation

Appears in: *Proceedings of the 15th International Conference on Autonomous Agents and Multiagent Systems (AAMAS 2016), John Thangarajah, Karl Tuyls, Stacy Marsella, Catholijn Jonker (eds.), May 9–13, 2016, Singapore.*

1. INTRODUCTION

Within the domain of Defense, Safety & Security, simulation is increasingly applied, both as part of the education and training curriculum, as well as to facilitate experimentation with new capabilities or operational procedures. State-of-the-art simulation environments can help reduce the time and costs needed to organize large-scale training events or rehearsals, and, more importantly, can provide scenarios, situations and events that are not easily experienced in traditional training. However, a lack of realistic simulated human behavior is perceived as a limiting factor for wider application of simulation as a training tool [43].

Many scenarios take place in dense urban environments, where personnel has diverse tasks to perform, such as meeting with local authorities, providing security at checkpoints, assisting local first responders, controlling crowds and protests, and identifying insurgents hiding among the population. To prepare personnel for such diverse and non-traditional tasks using cost-effective virtual training puts high demands on the complexity of scenarios and the capabilities of simulators. In addition to simulating a number of key agents with a high level of detail, a simulator must be capable of simulating thousands of civilian agents, many of which are not directly involved in the scenario. Imagine, for example, a surveillance operation, in which a specific key agent needs to be identified and tracked through a busy city center. This operation would not be properly simulated if civilian agents display behavior that is not sufficiently plausible, making the target clearly stand out against the background agents.

To facilitate simulating scenarios with a large number of virtual human agents in real-time and without the need to explicitly script behaviors for each agent, behavior needs to be modeled on a relatively high level of abstraction. A number of methods have been devised for efficiently populating virtual environments with simulated human agents. The level of fidelity of the proposed methods differs, from generated crowds that wander the streets without any clear purpose, to more believable approaches that simulate typical patterns-of-life [e.g. 11, 19, 20, 22, 34]. For the application domain, an interesting addition to simulating population behavior at a relatively high level of abstraction is the ability to control the flow of events during a training or exercise to facilitate learning moments or to cope with unforeseen trainee behavior.

Our contribution. We present an approach that facilitates instructors to control the flow of events by injecting *incidents* in a running urban simulation. In our approach, such incidents only specify the instructor's *intent* [33], i.e. what should happen, where, and when. An efficient planning and sampling algorithm translates this specification of intent to one or more parallel sequences of actions, represented in an extendable modular library, to be performed by concrete agents at specific locations and moments in time.

As an example of this approach, imagine a trainee playing as

a convoy commander, leading a convoy of trucks through a city. During the simulation, s/he may be confronted with an unexpected road block that delays the convoy's progress. The trainee will need to learn how to deal with such a delay. To facilitate a learning moment, the instructor places an incident called "Road block" on the road the trainee is heading for (which is not known beforehand), and indicates the incident should occur 2 minutes from now. Internally, the incident specification maps to a non-executable action called e.g. 'create obstruction'. The system now queries the action library for executable actions that implement 'create obstruction' and finds 'arrest offender' as a possible executable action to create an obstruction. To execute this action, it needs an agent to perform the role of offender. By applying backwards reasoning, it identifies that the action 'street theft' can turn an ordinary civilian into an offender. Thus, an agent in the vicinity is selected to perform a street theft and to move to the location of the arrest, within the 2 minute time constraint. Similarly, a team of agents are selected to interrupt their patrol (or exit a nearby police station as police officers), and arrive at the location of the arrest at exactly the same time as the offender.[1]

Given this extensible approach with a library of reusable actions and role definitions, introducing new types of incidents is relatively straightforward. For example, a new incident "First-responders help victim" can reuse actions detailed above within the context of a "Road block" incident: the 'street theft' action assigns a victim role to the robbed agent as a side effect, and thus can be used to plan this new type of incident that requires a victim.

To validate and demonstrate our approach, we developed a prototype urban simulation system. Since we aim to demonstrate how incidents can influence daily civilian behavior, we devoted attention to the simulation of such behavior as well. We developed a toolchain that allows scenario builders to create rich urban scenarios, including a geo-specific environment and a population with credible daily behavior, with little implementation effort. We applied this toolchain to simulating inhabitants and daily behavior in the Dutch municipality of Rijswijk (faster than real-time), which covers approximately 10 square kilometers and has around 30, 000 inhabitants. The properties of the environment and the simulated human agents are based on real-world census data [as in, e.g. 3, 5, 10]. Daily behavior is driven by relatively simple models, and adapts to environmental conditions such as the time of day, day of the week, or season.

2. RELATED WORK

Given the broad applicability of algorithms for population generation, activity generation, and real-time planning, it is no surprise that there is a similarly broad array of related work. Below, we give a short overview. We note that it was not our intention to include all state-of-the-art algorithms for population and activity generation in our prototype system; instead, we aimed to develop a pragmatic prototype that is capable of simulating relatively simple behaviors with a minimum of effort required from a scenario builder, and to add incidents on top of those behaviors.

Modeling a population. Population simulations are a form of large-scale multi-agent systems [14, 34, 40] and as such can benefit from many existing scalable architectures. An application field different from virtual training and scenario generation, where agent-based population generation has seen a lot of attention, is the large research field of urban traffic modeling and simulation [see, e.g. 1, 3–5, 19, 22, 24, 25]. The population generation algorithms in our prototype system build upon and can be extended with new developments in pattern-of-life or activity generation [e.g. 11, 17, 20, 35]. In numerous research papers, joint agent activities are considered in addition to individual activities [see, e.g. 13].

Using census data. Many researchers model populations based on real-world census data [see, e.g. 5, 10]. Two algorithms to fit simulated data with real-world data are widely used, i.e. Synthetic Reconstruction and Combinatorial Optimization [16, 17]. Synthetic Reconstruction combines various data sources into one data set. Given this data set, households can be sampled using e.g. Monte-Carlo techniques. Combinatorial Optimization generates households based on a global data set. It optimizes the fit of generated and expected household statistics by generating too many households and iteratively improving the selection of households that are placed. Some researchers suggest that Monte-Carlo Markov Chains yield a better fit [10]. In our prototype, we use a straightforward Monte-Carlo method that generates too many households and iteratively improves the sample, as this provides us with a sufficiently close fit with real-world data.

Modeling individuals. For simulation of individual agent movement and other behaviors, techniques such as path planning and character animation are continuously improving in performance and realism [e.g. 21, 38, 39]. When going from individual agents to groups, crowd navigation methods are able to achieve plausible pedestrian steering and avoidance patterns at e.g. busy intersections [e.g. 32, 41]. On an even deeper level of detail, we find cognitive and emotional modeling of agents and (cultural) agent interaction [see, e.g. 9]. In the current prototype system, we deliberately limited the level of detail, to ensure that all agents can be simulated efficiently while displaying sufficiently credible perceived behavior (in our case, the only perceived behavior is movement).

Planning and generating stories in run-time. A great body of work exists in the planning literature on realizing run-time plans. Especially the domain of games yields interesting work, such as investigations into the required world state for creating plans [29, 30], and the merits of techniques such as dynamic programming [2], Monte-Carlo sampling [8], hierarchical planning [15], and case-based planning [27]. What this work has in common is an agent-centered perspective on solving the planning problem at hand.

In our work, we aim to focus on the actions that lead to a certain incident. In the literature, the associated field of story generation is an active topic, that generally dives quite deep into aspects such as believability of individual characters. Early work with this focus aimed to generate stories based on a model of the human mind [26]. Later work focused on aspects such as plot coherence [33], providing tools to authors without much knowledge of programming [36], or adapting stories according to authors' preferences [42]. Moreover, instead of generating goal-driven stories, researchers aim to generate stories based on analogies [44]. Topics such as procedural storytelling and game quest generation are also relevant to discuss here [see, e.g. 18, 23, 33]. They share similarities with our approach in that they often plan a story or quest based on a library of elements, such as events and actions. A major difference is that these methods often generate a complete scenario and environment, while in our approach, we blend incidents into a running scenario.

3. APPROACH

Given that we intend to investigate and demonstrate how run-time incidents can be injected into a running simulation of daily behavior, our approach requires attention to three other aspects that together form the basis of the urban simulation: (i) modeling the environment, (ii) synthesizing a matching population, and (iii) simulating its daily behavior. One of the main requirements for these aspects was that they

[1] Obviously, this example illustrates just one of many valid manners to realize the requested incident; the road is effectively blocked while the police officers are performing an arrest on the street thief.

could be implemented and extended in a relatively straightforward manner. Before moving to run-time incidents in Section 3.4, we first discuss these three aspects in some detail.

3.1 Modeling the environment

The environment model, which contains buildings, infrastructure, and outdoor areas such as parks and water, is derived from geo-specific open data sources, such as OpenStreetMap [28], and includes a functional description of each building or public area, which is represented in a hierarchy that features multiple inheritance. For example, a shop is inside, is a place where people work, and is a place where people can buy products.

Each building or area also includes relevant environmental properties such as housing capacity, taken from the Dutch building register. In cases where data is insufficiently detailed, incomplete, or inconsistent, it is estimated based on heuristics (such as the size of a building), or, in some cases, manually provided.

3.2 Generating a population

Our population model defines a number of basic properties for simulated human agents (e.g., gender, age, household type, role in the household). The values of these properties for each agent are sampled from openly available census data provided by the Dutch Central Bureau of Statistics [7]. The data provides us with various independent statistics concerning the neighborhoods in the area of interest, such as the number of inhabitants, number of men and women, number of people per age category, number of people with a certain civil status (single, couple, couple with children, single parent), number of households, and average household size. Due to privacy concerns, conditional statistics are provided only on the national level (e.g., how many married men are in their thirties).

We generate agents and households using the aforementioned independent statistics on neighborhood level and the conditional statistics on national level. To generate households, we first sample an adult reference agent, including properties such as gender, age and household type. If the household type is consistent with having children, we sample the number of children, given the gender, age and household type of the reference agent. We then sample properties for each child. If the household type indicates that the reference agent has a partner, we also sample a partner (and all properties of that partner) according to the statistics. Subsequently, we assign households to houses. The surface area of the house creates an obvious constraint on the size of the household assigned to it.

Given that we use national conditional data to generate households, we may observe a mismatch between actual and observed data on the level of neighborhoods once we assigned all households to houses. For example, a certain neighborhood may have far more single-parent households than the country as a whole. To correct this, we use a simple algorithm [17]. We generate too many households, fill houses in each neighborhood with a random sample from these households, and iteratively improve the observed statistics for this sample by swapping in unassigned households, until observed statistics on the neighborhood level match sufficiently with expected statistics. We found that this simple algorithm can find a nearly perfect fit for our area of interest, with $30,000$ inhabitants, within a few seconds.

From the basic properties assigned to agents, such as age and gender, additional useful properties, such as an agent's workplace (if any), preferred recreational locations, and typical walking speed [6], are derived. Moreover, we generate each agent's social network to allow for behavior that requires others, such as having dinner with friends. After explicitly adding all members of a household to each others' social network, we extend the network with an algorithm from the literature [37].

3.3 Daily behavior

Daily behavior for each agent is represented in the form of an *agenda*. The activities in an agent's agenda depend on the properties of that agent, and on environmental properties, such as the day of the week and the season. Clearly, the activities in the agenda should not overlap in time, and travel times between activities should be taken into account. In order to generate all agendas, our agenda planner uses a Monte-Carlo activity generation algorithm, while considering time constraints. In our prototype system, a simulation lasts for a maximum of 24 simulated hours. Therefore, we do not (need to) incorporate concepts such as multi-day agendas, future planning, or utility-based activity generation. Furthermore, in the current system, dynamic changes in agendas are not implemented.

Agendas are created by first calculating agent-specific probabilities for each activity. The probability that a certain activity appears in the agenda of a specific agent is determined by a number of factors. First, each activity has a baseline probability to be added to an agenda.

Second, to facilitate a compact and extensible notation for the relation between activities and individual or environmental properties, the baseline probability can be altered using *multipliers*. For example, let us assume that the baseline probability for "going to school" is set to 0.5. Obviously, the probability of "going to school" depends strongly on an agent's age; therefore, we might introduce a multiplier of $\times 3$ for this activity and agents below 16 years of age. Similarly, an agent over 21 can have a multiplier of $\times 0.1$ for this activity. The probability of "going to school" does not only depend on the age of the agent; for example, Sunday, or the Summer holiday, are rather unlikely moments for this activity, which can be denoted with a multiplier of $\times 0$ for this activity on Sunday or in the time of the year corresponding to Summer holiday. All relevant multipliers are applied to the baseline probabilities of all activities. The probabilities are then normalized.

The resulting agent-specific probabilities for each activity are used to sample activities to be planned for each agent. To make sure selected activities do not overlap in the agent's agenda, we use a simple iterative algorithm. This algorithm keeps a list of available time frames for each activity that might still be planned. New activities are placed in an available time frame in the agent's agenda, if possible. We note that, given the nonlinear order in which activities are planned, there may already be activities before and after the activity that is currently being placed in the agenda.

When an activity is placed in an agent's agenda, a number of properties of the activity are computed, as follows.

Start and end time. The concrete start and end time of the activity within the time frame indicated by the agenda are picked. Preferences for starting times are represented probabilistically in each activity. For example, the agenda may plan the activity "going to school" to start anywhere between 8am and 10am, and the activity picks a starting time of 8.30am.

Location. As explained above, locations in the environment are annotated with functional descriptions, and activities specify the functional descriptions they require. For example, "going to school" has to take place in a school location. From the set of locations in the environment that match the functional description required (or a more specific one), a number of random samples are taken. A sampled location is chosen when it allows an agent to move towards and from it within the available time frame, while also performing the activity for at least a (specified) minimal duration. Once all activities for the day have been planned, additional "travel time" activities are inserted to move between locations. Agents always start and end their day at home.

Contacts involved. We need to take into account which, if any, other agents from the agent's social network are involved in an activity. In some cases, multiple agents share a high-priority activity, which therefore has to be planned very early in the process; an example is bringing your child to school. In other cases, shared activities have a low importance and are planned relatively late; an example is having a drink with friends. In the latter cases, it may very well happen that friends' agendas do not leave room for the activity and that the activity has to be canceled.

Simulating daily behavior. Once the agenda planning algorithm is finished, each agent ends up with a sequence of activities that need to be performed. Coupled with each activity is a *behavior model* that specifies what the agent must do to perform the activity. For example, an activity such as "walking to work" is coupled with a movement model, whereas "relaxing in the park" is coupled with a model that makes the agent wander around the park for a bit and perhaps sit down now and then. Within each agent, models are kept in a *stack*, and each agent executes only the model at the top of the stack, once per time step. Models are removed from the stack when finished (e.g., a movement model terminates once the agent arrives). Models may be shared by multiple agents to facilitate coordination and joint behavior. For example, the model that controls parents bringing their child to school also controls the child.

3.4 Generating run-time incidents

The key element in our approach is that users can change the behavior of the population during a simulation run. To this end, the user can make specific incidents happen at a desired moment, for instance to create a challenge or new learning experience for a trainee.

Users can specify a small number of high-level parameter settings for incidents. Some of these are defined explicitly, e.g., an incident is to take place at a certain location and time. Others are automatically derived based on incidents' definitions, e.g., which types of agents are required to participate in the incident.

Based on the chosen incident and parameter settings, we derive what is needed to make the given incident happen at the desired time, in several phases, detailed below. Instead of directly generating a plan that includes actions, agents and locations, we restrict the size of the planning problem to the number of actions, by creating an abstract plan with a number of unassigned variables, instead of concrete agents and locations. Then, we instantiate the abstract plan by assigning agents and locations to the variables. We adopted a phased approach to overcome the issue that planning problems are PSPACE-complete in the size of the input [12].

Roles and actions. In the first phase, we create an *action plan* to cause the incident. We prepare agents involved for their *roles* in the incident by giving them a suitable sequence of *actions*. Actions and associated concepts, detailed below, are all defined in a modular library that supports multiple inheritance. In this way, quite elaborate incidents can be automatically composed by the planner, without the need for extensive scripting.

Actions are placeholders with a number of variables, i.e. an actor variable, an optional target variable, and a location/time variable, each with possible constraints. In the sequel, we use the notation NameOfAction(a, t, l) for actions; actions without target will be denoted as NameOfAction$(a, \cdot, l)$. In the planning phase, the variables a, t and l are not given a concrete value yet; instead, the planner makes a plan where the variables are correctly linked. In addition to variables, any action may define preconditions concerning roles that are required for actor or target agents, and postconditions specifying the roles the actor or target agents will have after the action is performed. Finally, any action has an underlying model, which controls

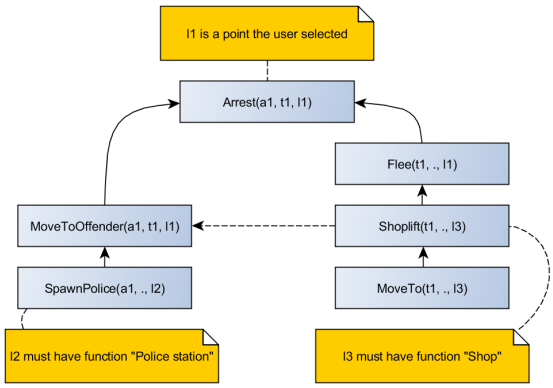

Figure 1: Example of a plan to arrest a shoplifter.

the behavior of the agents involved while the action is performed. We note that such a model may affect nearby agents via area triggers.

Pseudo-code for the planner is provided in Algorithm 1. References to line numbers in the sequel refer to this pseudo-code. We see that a plan is first initialized with the enabling action of the incident, i.e. the scene the user would want to see (line 1–3). For example, the incident "Arrest an Offender" is caused by one or more agents a performing the action Arrest(a, t, l) on one or more target agents t. Second, the planner resolves any role preconditions that need to be satisfied to allow this enabling action (line 4). In the example of the action Arrest(a, t, l), the actor must have the role "Police Officer", while the target must have the role "Offender". Since the incident allows the user to specify a more specific role than a generic offender

Algorithm 1 Algorithm for planning run-time incidents.

```
 1: function CREATE_PLAN(event e)
 2:     plan p ← EMPTY_PLAN(e)
 3:     action a ← ENABLING_ACTION(e)
 4:     RESOLVE_ROLE_PRECONDITIONS(p, a)
 5:     INSERT_MOVE_ACTIONS(p, a)
 6: function RESOLVE_ROLE_PRECONDITIONS(plan p, action a)
 7:     SATISFY_ROLE(p, a, ACTOR(a))
 8:     SATISFY_ROLE(p, a, TARGET(a))
 9: function SATISFY_ROLE(plan p, action a, group g)
10:     role r ← MOST_SPECIFIC_ROLE_REQUIREMENT(p, g)
11:     actions eas ←ACTIONS_ENABLING_ROLE(r)
12:     action ea ← SELECT_RANDOM(eas)
13:     INSERT_BEFORE(p, ea, a)
14:     if TARGET(ea) = ACTOR(a) || TARGET(ea) = TARGET(a) then
15:         ADD_TARGET_DEPENDENCY(p, a, ea)
16:     RESOLVE_ROLE_PRECONDITIONS(p, ea)
17: function INSERT_MOVE_ACTIONS(plan p, action a)
18:     for all action i ← ACTIONS_BEFORE(p, a) do
19:         role r ← ROLE_AFTER(i)
20:         group t ← TARGET(i)
21:         actions ms ← ACTIONS_FOR_ROLE(r)
22:         if t ≠ nil then
23:             action c ← SELECT_BEST_FIT(ms, t)
24:         else
25:             action c ← SELECT_RANDOM(ms)
26:         INSERT_BETWEEN(p, c, i, a)
27:         if HAS_TARGET_DEPENDENCY(a) then
28:             MOVE_TARGET_DEPENDENCY(p, a, c)
```

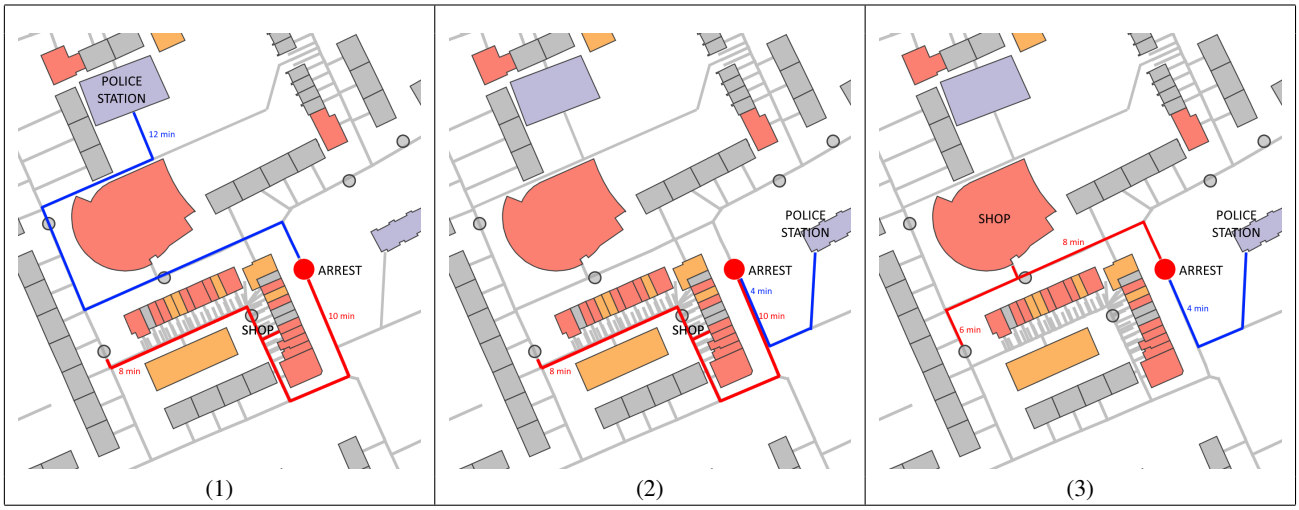

Figure 2: Example of three optimization steps in finding agents and locations for a plan to arrest a shoplifter within 15 minutes. (1) The shoplifter requires 8 minutes to reach the shop. Subsequently, s/he requires 10 minutes to arrive at the location of the arrest, while the police officer requires 12 minutes. The police officer has to wait for the actual shoplifting before s/he can start moving. The duration of the plan is therefore 20 minutes. The critical path improvement algorithm can now improve the location of the police station, the shop, and the agent that will be the shoplifter. (2) The algorithm chooses to find a different police station that is closer to the arrest location. As a result, the plan duration is 18 minutes, and the critical path is only concerned with the location of the shoplifter-to-be and the shop. (3) The algorithm chooses to find a different shop that is closer to the arrest location. Coincidentally, this shop is also closer to the shoplifter. The plan duration is now 14 minutes, which is within 15 minutes.

for the target (for the example, let us assume he chose "Shoplifter", which is a subclass of "Offender"), the planner looks for actions that give their actor or target the most specific role (line 10). For example, the action Shoplift$(a, \cdot, l)$ gives its actor a the role "Shoplifter".

Two actions will automatically be performed in the correct order if the actor is the same for both actions. We need to check, however, whether the target has performed preceding actions as well (line 14–15). For example, it would be quite unrealistic if police officers are arresting someone before s/he actually committed a crime. We call this dependency a *target dependency*, because the actor of an action has to wait for the target to do something.

Whenever new actions are added to the plan, the planner recursively resolves any role preconditions for these actions in the manner described above (line 16). This results in a plan in which all role preconditions for the enabling action of the incident are resolved.

Movements between actions. Once a plan has been created that resolves all role preconditions, the planner enables agents to move between action locations. For this, the planner introduces movement actions, inserted before each action in the plan (line 5). Although some actions might coincide at the same location, we assume that this is generally not the case. In the sampling process, when locations are selected, it may happen that two different location variables are assigned the same physical location.

The algorithm for inserting suitable moves before actions takes into account two aspects. First, agents may have acquired a certain role as a result of actions they performed earlier. For example, when we insert a move for the actor a involved in Arrest(a, t, l), we can derive, by traversing the action sequence for a top-down from Arrest(a, t, l), that this actor has the role "Police Officer" by the time they are to move towards Arrest(a, t, l). Therefore, we need to select a type of movement that fits this role (line 21). Otherwise, the planner might, for example, decide to let the police officers flee towards the arrest location. The second aspect considered when inserting suitable move actions is the target of the action we move towards (line 23). This is best illustrated by our running example. The police officers are moving towards the arrest of a shoplifter. The planner may be able to choose between a number of different movement actions to do this, for example a generic MoveTo$(a, \cdot, l)$, an action MoveToPersonInNeed(a, t, l) which has a role precondition for the target t ("PersonInNeed"), and an action MoveToOffender(a, t, l), which has a similar role precondition (t must be an "Offender"). Clearly, to achieve realistic movement towards the arrest location, the police officer(s) should move using the MoveToOffender(a, t, l) action. The generic MoveTo$(a, \cdot, l)$ is less suitable; the associated model will not take into account that, for example, the offender is to be chased when spotted. The MoveToPersonInNeed(a, t, l) is the least suitable, because the associated model will, for example, be very careful and respectful in approaching the target.

Once the most fitting move action is chosen, it is inserted in the plan just before the target action (line 26). Any existing target dependencies that realize certain roles, attached to the target action, need to be attached to this new move action instead (line 28). For example, it would be unrealistic for police officers to start running towards the location where an offender will be arrested, before this offender has actually committed a crime.

Assigning agents and locations. The result of the planning algorithm described above is a sequence of actions, with correctly linked variables that can be instantiated by assigning agents and locations that are present in the environment. An example is shown in Figure 1. An algorithm based on sampling and iterative improvement now finds suitable agents and locations, and ensures that the plan can be executed within the requested time constraints (i.e., the enabling action of the plan happens exactly when the user wants it to). An illustrative example for the plan in Figure 1 is given in Figure 2.

Intuitively, the algorithm works by first sampling a random valid assignment for the locations and agent variables in the plan. This assignment takes into account any constraints on the variables. For example, a shoplifting needs to take place in a shop, as represented in a constraint on the location/time variable of the Shoplift$(a, \cdot, l)$ action. Then, while the plan takes too long to execute, a number of

iterative improvements are performed, in which (1) the critical path in the plan is found, i.e. the sequence of actions that takes most time, and (2) along the critical path, one variable is assigned differently (but validly), such that the plan is expected to take less time.

Which variable needs to be assigned differently is determined by choosing an action on the critical path, with a probability equal to the time required to reach this action. In other words, actions that take more time on the critical path have a higher chance to be optimized. Given the chosen action, reducing plan duration works as follows. Whenever there is an action preceding the chosen action on the critical path, the current location of this preceding action is apparently too far away from the location of the chosen action. Therefore, the value of the location/time variable of this preceding action needs to be adjusted such that it refers to a valid location closer to the chosen action. If there is no preceding action, the chosen action involves one or more agents that are apparently too far away from the location the chosen action will take place at, and we need to find valid agents that are closer to this location.

If a fixed number of improvement iterations does not yield a plan that fits the time constraints, the algorithm is restarted with a new random valid assignment of the variables in the plan. After a number of failed restarts, the algorithm concludes that planning the incident within the required time is not feasible. The user is then presented with the best plan found so far, and can choose whether s/he wants to have the incident happen at the corresponding time instead.

Executing an incident. Once an incident is instantiated by creating a plan of actions and assigning agents and locations to these actions, it is immediately added to the simulation. The required behavior models are constructed with the proper constraints (e.g., some models should wait until others are finished) and are pushed onto the model stacks of the agents involved. Given how agents execute their model stacks (as described in the section about daily behaviors), this means that daily behaviors are temporarily interrupted, until the models related to the incident are finished. In addition to affecting the agents involved, incidents can also affect agents that happen to be nearby. For example, a procession typically attracts viewers. This is realized by allowing models to place area triggers in the environment. These triggers are automatically removed once a model finishes.

4. RESULTS

The approach described in the previous sections was implemented in a standalone prototype simulation environment. The prototype offers several tools to gain insight into the patterns-of-life generated on the basis of our population model, such as faster-than-real-time simulation, agent inspection and tracking, and jumping to a specific date and time. Furthermore, the tool allows one to inject an incident at any time during the simulation and immediately see its execution. Here, we discuss the results of the pattern-of-life simulation and run-time incident generation using a number of examples.[2]

Daily behavior. Figure 3 illustrates how the behavior of the local population is dependent on factors such as time-of-day, day of the week, season and location. The various subfigures show situation snapshots of three locations at different days and times. Figures 3 (a) and (b) concern a road on Monday; it is busy during morning rush hour (a), and much quieter during the day (b). Figures 3 (c) and (d) show the town square, where a market is held every Saturday morning (c); at other times, the only people on the square are bored youngsters (d). Finally, Figures 3 (e) and (f) are centered on a park, which is very busy on a day off in Summer (e), whereas hardly anyone goes there on a day off in Winter (f).

Run-time incidents. To demonstrate the capabilities of our incident planner, we defined a number of incidents that are quite diverse in nature. Figure 4 shows two example incident executions: (a) two thieves are arrested by police after shoplifting; and (b) an ad hoc protest rally of fifty people is injected into a busy street. We observed that these incidents are planned and start their execution in a running simulation within a second after being placed.

We conducted a more thorough experiment to establish the quality of the incident planner and sampler, and repeatedly placed 500 random incidents from our library at random positions in the environment, allowing for 3–7 minutes of simulation time to realize these incidents. On average, we found that a valid instantiation for an incident within our environment can be found within $0.22s$ ($\pm 0.11s$).[3] Moreover, for incidents that could not be planned, the best result was returned within $0.65s$ ($\pm 0.43s$). This was the case in approximately 25% of the requested random incidents. Exhaustive search in these cases did not yield a significantly lower failure rate. This shows that the failed incidents were in fact impossible to realize, simply because they were requested at a location that was not optimal (e.g. a shoplifter being arrested very far from any shops).

In general, results indicate that the time required to find a valid plan depends on three factors, viz. (1) the number of agents involved in the incident, (2), the length of the action sequences in the plan, and (3) the size of the environment. The first two factors are obvious. The third factor concerns the fact that we initially perform a random assignment and iteratively improve this assignment until the plan is realized within desired time constraints. Clearly, with a large environment, we should apply an initial assignment that is not completely random, but instead, we should sample from locations that are relatively close to the location of the incident. With an efficient environment data structure, such a proximity-biased location sampling mechanism would not be hard to implement.

5. DISCUSSION

Our approach intends to allow instructors to control the flow of events by adding run-time *incidents* to a running simulation of daily behavior. In this section, we discuss both the strengths of our approach and the capabilities of the prototype simulation, as well as the limitations in method and implementation.

Daily behavior. To obtain a credible simulation of daily behavior, the environment and population models are constructed upon real-world census data. Clearly, for this to work, data for the area of interest on e.g. the road network, shapes and functions of buildings, and the statistical properties of the population must be available. In the absence of such data, data synthesis may be required, potentially in combination with manual specification, which will increase the effort required to set up a population simulation in such a geographic setting. In our data, for a number of buildings and outside areas, detailed functional descriptions were missing and were heuristically completed, using properties such as location and surface area. Moreover, we added functions that were not in the census data at all, ranging from high-level functions such as “inside” or “outside”, to very detailed ones such as “playground” or “restaurant”.

[2]We note that the results presented here are not intended as a rigorous evaluation of the prototype system and the underlying algorithms. We solely intend to demonstrate the functionality and performance of the prototype system. To our knowledge, there are no benchmark or state-of-the-art systems with which we can directly compare the functionality and performance of our system, within the setting described (large-scale urban pedestrian activity).

[3]The prototype system is implemented in Java 8. All times reported were observed on average office hardware, i.e. an Intel Core i5 4310M at 2.7GHz, with no visual representation of the simulation.

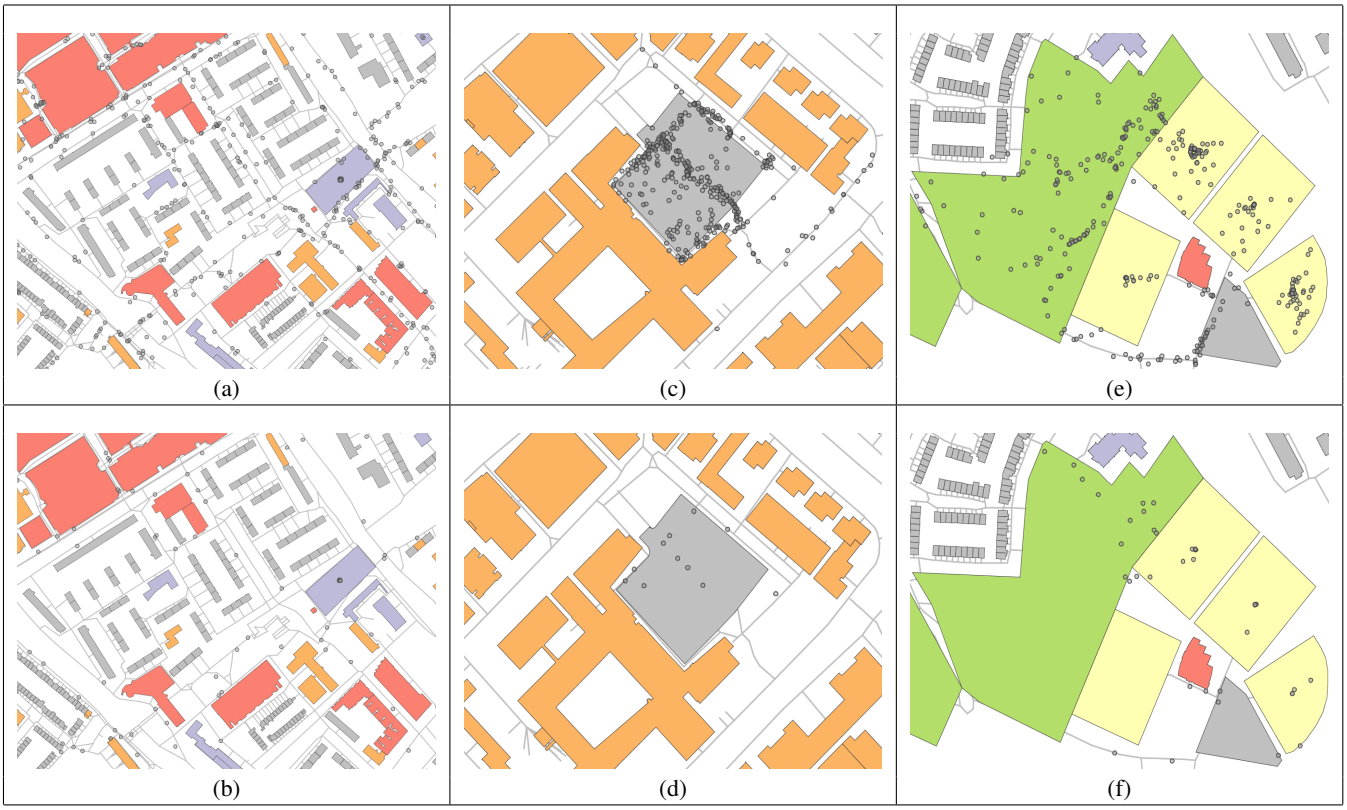

Figure 3: Variations in population behavior and densities as a result of switching day and time: (a) Monday morning rush (8:12), (b) Monday, late morning (11:00), (c) Saturday market on the town square (11:00), (d) the same square when there is no market (Monday, 11:00), (e) a recreational area on a Sunday in the middle of Summer (11:00), (f) the same area on a Sunday in Winter (11:00).

Figure 4: Two run-time injected incidents: (a) an arrest of two shoplifters (red routes) by a police officer (blue route), (b) an improvised protest march (blue route; green circles represent participants), which attracts passers-by (gray circles) using an area trigger.

Even in the presence of rich data sources, the current prototype implementation is limited to pedestrians and closed simulation (i.e. there are no agents coming from outside the area of interest, or leaving it). Obviously, creating a realistic simulation of daily behavior was not the focus of our work. On the other hand, the prototype could straightforwardly be extended to include multiple traffic modalities, or to support an open simulation. This extension would entail only engineering challenges. A more fundamental limitation of our approach is that agents' daily behaviors currently do not adapt to dynamic situations or changing circumstances. For example, agents will not decide to have an unplanned chat when encountering a friend on the way home. Unplanned, dynamic actions

can increase the realism and variety of agents' daily behavior.

Run-time incidents. The current implementation for changing behavior during a simulation run, i.e. run-time incident generation, works rather well and is fast and reliable. We did not discuss certain details here, however, that are worth improving upon.

First, similar to daily behaviors, incidents are planned once, but do not adapt to changing circumstances. A user might trigger this limitation by planning two or more incidents to occur at the same time, where one incident causes the other to fail. Moreover, in some cases, behavior models should take into account that agents may encounter each other earlier than explicitly intended. For example, in 2(b), police officers and shoplifters share part of their path towards the arrest location. Depending on the timing, we may observe a situation where police officers are walking in front of the shoplifters, with the shoplifters playing catch-up. In the current system, we do not solve this issue. It may be largely solved by introducing more constraints and mutual influence in behavior models. For example, the path and timing MoveToOffender(a, t, l) chooses should depend on the path and timing of the offender; ideally, the model could even change offender behavior instead of only police officer behavior.

Second, the current planner only takes into account pre- and post-conditions concerning roles and locations. For the currently defined incidents, these conditions are sufficiently expressive. More advanced incidents might require, for example, conditions concerning agents' possessions (e.g. a thief must have a stolen item) or conditions concerning attributes of locations (e.g. a shop must be open for a shoplifter to shoplift). Finally, the casting of agents to perform in the incident does not take into account all attributes of these agents, potentially leading to unrealistic situations, where a 93-year-old man outruns the police, because the incident planner ensures that the police do not overtake him.

6. CONCLUSIONS AND FUTURE WORK

Defense, Safety & Security personnel is increasingly often operating in complex urban environments, and simulation-based education & training scenarios need to represent these environments in a credible manner. Without a realistically behaving population, such urban scenarios do not provide the desired didactic value. Therefore, research topics such as population generation and daily activity generation provide many exciting opportunities for application in this domain. Of particular interest to us is the opportunity for instructors to influence running scenarios to optimally tailor the trainees' learning experience during the session.

Our contribution. In this paper, we present an approach for injecting incidents into an urban scenario at run-time, interrupting daily activities and temporarily promoting selected background agents to play a specific role in the sequence of events. Because incidents are planned on-the-fly based on a library of available actions and associated pre- and post-conditions, new incidents can build upon content that is already present. For example, it required very little effort to introduce an incident for helping the victim of a street theft, once we had created an incident concerning the arrest of a street thief. In addition to run-time incidents, the prototype system also implements a simulation of daily patterns-of-life, based on real-world census data. We are capable of generating a population that statistically matches the real population of a municipality with $30,000$ inhabitants. Individuals in the simulation are given daily activities (such as work, leisure, shopping, having dinner) that match their properties (such as age, gender, and household). We designed the prototype system such that it can be easily extended to include more daily activities, behavior models and different modalities.

Future work. The discussion above outlines a number of possible improvements to the system, e.g. supporting changes in plans, both for daily behavior as well as for incidents, and including other traffic modalities than pedestrians. Here, we outline some other ideas for future work. First, we see a great deal of potential in supporting models at various levels of detail. At the moment, the prototype system supports population models at a fixed level of detail. When, for example, trainee interaction with an individual population member is a scenario requirement, more detailed models are needed. Due to the computational complexity of such models, it is not feasible to have this level of detail for the entire population all the time. Instead, the system must be able to dynamically adapt to the level of detail required. Second, we feel that user accessibility is a very important property of any intent-driven approach. For the presented system, a clear improvement could be to allow users to specify not only incidents, but also population behavior on the level of intent, e.g. by drawing an intense morning rush hour on a specific street, which influences population attributes such as residence and place of work to ensure the desired dense traffic.

Online repository. The prototype system is released in a managed open source hosting environment, including documentation and demonstration material [31]. We encourage others to experiment with the system and to extend it with their own models and algorithms.

References

1. Q. Bao, B. Kochan, T. Bellemans, Y. Shen, L. Creemers, D. Janssens, and G. Wets. Travel demand forecasting using activity-based modeling framework feathers: An extension. *International Journal of Intelligent Systems*, 30(8): 948–962, 2015.

2. A. G. Barto, S. J. Bradtke, and S. P. Singh. Learning to act using real-time dynamic programming. *Artificial Intelligence*, 72(1–2):81 – 138, 1995. ISSN 0004-3702.

3. R. J. Beckman, K. A. Baggerly, and M. D. McKay. Creating synthetic baseline populations. *Transportation Research Part A: Policy and Practice*, 30(6):415 – 429, 1996. ISSN 0965-8564.

4. C. Bhat and F. Koppelman. A retrospective and prospective survey of time-use research. *Transportation*, 26(2):119–139, 1999.

5. C. Bhat and R. Misra. Discretionary activity time allocation of individuals between in-home and out-of-home and between weekdays and weekends. *Transportation*, 26(2):193–209, 1999.

6. R. W. Bohannon. Comfortable and maximum walking speed of adults aged 20-79 years: reference values and determinants. *Age and Ageing*, 26:15–19, 1997. School of Allied Hearth, University of Connecticut.

7. Centraal Bureau voor de Statistiek. URL `http://www.cbs.nl/`.

8. G. Chaslot, S. Bakkes, I. Szita, and P. Spronck. Monte-carlo tree search: A new framework for game ai. In *Proceedings of AIIDEC-08*, pages 216–217, 2008.

9. M. Dastani and J.-J. Meyer. Agents with emotions. *International Journal of Intelligent Systems*, 25(7):636–654, 2010.

10. B. Farooq, M. Bierlaire, R. Hurtubia, and G. Flötteröd. Simulation based population synthesis. *Transportation Research Part B: Methodological*, 58(C): 243–263, 2013.

11. J. Folsom-Kovarik, S. Schatz, R. Jones, K. Bartlett, and R. Wray. Scalable models for patterns of life. In *Innovative Applications of Artificial Intelligence*, 2013.

12. M. Ghallab, D. Nau, and P. Traverso. *Automated Planning: Theory & Practice*. The Morgan Kaufmann Series in Artificial Intelligence. Elsevier Science, 2004. ISBN 9780080490519.

13. J. Gliebe and F. Koppelman. A model of joint activity participation between household members. *Transportation*, 29(1):49–72, 2002.

14. V. Graudina and J. Grundspenkis. Technologies and multi-agent system architectures for transportation and logistics support: An overview. In *International Conference on Computer Systems and Technologies - CompSysTech*, 2005.

15. H. Hoang, S. Lee-Urban, and H. Munoz-Avila. Hierarchical plan representations for encoding strategic game ai. In *Artificial Intelligence and Interactive Digital Entertainment Conference*, pages 63–68, 2005.

16. Z. Huang and P. Williamson. A comparison of synthetic reconstruction and combinatorial optimisation approaches to the creation of small-area microdata. Department of Geography, University of Liverpool, 2001.

17. N. Huynh, M. Namazi-Rad, P. Perez, M. Berryman, and Q. Chen. Generating a synthetic population in support of agent-based modeling of transportation in sydney. *20th International Congress on Modelling and Simulation (MODSIM 2013)*, pages 1357–1363, 2013.

18. B. in het Veld, B. Kybartas, R. Bidarra, and J.-J. C. Meyer. Procedural generation of populations for storytelling. In *Proceedings of PCG 2015 - Workshop on Procedural Content Generation for Games, co-located with the Tenth International Conference on the Foundations of Digital Games*, 2015.

19. M. Jakob, Z. Moler, A. Komenda, Z. Yin, A. X. Jiang, M. P. Johnson, M. Pechoucek, and M. Tambe. Agentpolis: Towards a platform for fully agent-based modeling of multi-modal transportation (demonstration). In *12th International Conference on Autonomous Agents and Multiagent Systems*, 2012.

20. R. M. Jones, J. T. Folsom-Kovarik, P. McLaughlin, and R. Frederiksen. High-density pattern-of-life modeling. In *Proceedings on the International Conference on Artificial Intelligence (ICAI)*, pages 438–444, 2015.

21. I. Karamouzas, R. Geraerts, and M. Overmars. Indicative routes for path planning and crowd simulation. In *Proceedings of the 4th International Conference on Foundations of Digital Games*, pages 113–120. ACM, 2009.

22. R. Kitamura, C. Chen, R. M. Pendyala, and R. Narayanan. Micro-simulation of daily activity-travel patterns for travel demand forecasting. *Transportation*, 27: 25–51, 2000.

23. B. Kybartas, R. Bidarra, and E. Eisemann. Integrating semantics and narrative world generation. In *Proceedings of FDG 2014 - Ninth International Conference on the Foundations of Digital Games*, 2014.

24. Multi-Agent Transport Simulation (MATSim). URL `http://www.matsim.org/`.

25. Y. Nakajima, S. Yamane, and H. Hattori. Multi-model based simulation platform for urban traffic simulation. In *Proceedings of the 13th International Conference on Principles and Practice of Multi-Agent Systems*, PRIMA'10, pages 228–241. Springer-Verlag, 2012. ISBN 978-3-642-25919-7.

26. N. Okada and T. Endo. Story generation based on dynamics of the mind. *Computational Intelligence*, 8(1):123–160, 1992. ISSN 1467-8640.

27. S. Ontanon, K. Mishra, N. Sugandh, and A. Ram. Case-based planning and execution for real-time strategy games. In R. Weber and M. Richter, editors, *Case-Based Reasoning Research and Development*, volume 4626 of *Lecture Notes in Computer Science*, pages 164–178. Springer Berlin Heidelberg, 2007.

28. OpenStreetMap. URL `https://www.openstreetmap.org/`.

29. J. Orkin. Symbolic representation of game world state: Toward real-time planning in games. *Proceedings of the AAAI Workshop on Challenges in Game Artificial Intelligence*, 5, 2004.

30. J. Orkin. Agent architecture considerations for real-time planning in games. In *AIIDE*, pages 105–110, 2005.

31. Population Modelling with Run-Time Incidents. URL `https://github.com/TNOCS/idsa`.

32. F. Qiu and X. Hu. Modeling group structures in pedestrian crowd simulation. *Simulation Modelling Practice and Theory*, (18):190–205, 2009.

33. M. O. Riedl and R. M. Young. An intent-driven planner for multi-agent story generation. In *AAMAS 2004: The Third International Joint Conference on Autonomous Agents and Multi Agent Systems*, 2004.

34. D. Scerri, A. Drogoul, S. Hickmott, and L. Padgham. An architecture for modular distributed simulation with agent-based models. In *Proceedings of the 9th International Conference on Autonomous Agents and Multiagent Systems: Volume 1 - Volume 1*, AAMAS '10, pages 541–548. International Foundation for Autonomous Agents and Multiagent Systems, 2010. ISBN 978-0-9826571-1-9.

35. S. Schatz, J. Folsom-Kovarik, K. Bartlett, R. E. Wray, and D. Solina. Archetypal patterns of life for military training simulations. In *Proceedings of the Interservice/Industry Training, Simulation & Education Conference*, 2012.

36. J. Skorupski, L. Jayapalan, S. Marquez, and M. Mateas. Wide ruled: A friendly interface to author-goal based story generation. In M. Cavazza and S. Donikian, editors, *Virtual Storytelling. Using Virtual Reality Technologies for Storytelling*, volume 4871 of *Lecture Notes in Computer Science*, pages 26–37. Springer Berlin Heidelberg, 2007.

37. R. Toivonen, J.-P. Onnela, J. Saramäki, J. Hyvönen, and K. Kaski. A model for social networks. *Physica A: Statistical Mechanics and its Applications*, (371): 851–860, 2006. Helsinki University of Technology.

38. W. Van Toll, A. Cook IV, and R. Geraerts. A navigation mesh for dynamic environments. *Computer Animation and Virtual Worlds*, 23:535–546, 2012.

39. H. Van Welbergen, B. J. Van Basten, A. Egges, Z. M. Ruttkay, and M. H. Overmars. Real time animation of virtual humans: A trade-off between naturalness and control. In *Computer Graphics Forum*, volume 29, pages 2530–2554. Wiley Online Library, 2010.

40. M. Wooldridge and N. R. Jennings. Agent theories, architectures, and languages: A survey. In *Proceedings of the Workshop on Agent Theories, Architectures, and Languages on Intelligent Agents*, ECAI-94, pages 1–39. Springer-Verlag New York, Inc., 1995. ISBN 3-540-58855-8.

41. B. Yersin, J. Maïm, J. Pettré, and D. Thalmann. Crowd patches: populating large-scale virtual environments for real-time applications. In *Proceedings of the 2009 symposium on Interactive 3D graphics and games*, pages 207–214, 2009.

42. H. Yu and M. O. Riedl. A sequential recommendation approach for interactive personalized story generation. In *Proceedings of the 11th International Conference on Autonomous Agents and Multiagent Systems - Volume 1*, AAMAS '12, pages 71–78. International Foundation for Autonomous Agents and Multiagent Systems, 2012. ISBN 0-9817381-1-7, 978-0-9817381-1-6.

43. G. Zacharias, J. MacMillan, and S. Van Hemel, editors. *Behavioral modeling and simulation: from individuals to societies*. National Academies Press, 2008.

44. J. Zhu. Towards analogy-based story generation. In *In Proceedings of the First International Conference on Computational Creativity (ICCC-X*, 2010.