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

(Extended Abstract)

Steven de Jong*, Alex Klein, Ruben Smelik, and Freek van Wermeskerken
Department of Modeling, Simulation and Gaming, TNO, The Netherlands

ABSTRACT

Motivated by many potential applications within the domain of Defense, Safety & Security, we present an approach for the injection of *incidents* (e.g., an arrest following a shoplifting) in a running, large-scale urban simulation. Incidents are generated within desired user constraints such as what should happen, where, and when. We believe that this approach offers an intuitive and extendable way to create scenario-specific exceptions to daily behavior patterns.

Keywords

Run-Time Incidents, Scenario Generation, Urban Simulation

1. INTRODUCTION

Within the domain of Defense, Safety & Security, state-of-the-art simulation environments can help reduce the time and costs needed to organize large-scale training events or rehearsals. Moreover, they can provide scenarios and situations that are not easily experienced in a real-world setting. However, instructors often experience a lack of control over the flow of scenarios, especially since the behavior of both virtual and real humans involved in these scenarios can be highly unpredictable [3]. For example, a trainee may choose a different route than the one the instructor prepared for.

We present an approach that allows instructors to more directly control and adjust the flow of a running scenario by injecting *incidents* at run-time, to facilitate learning moments or cope with unforeseen behavior. Incidents in our approach specify the instructor's *intent*, i.e. what should happen, where, and when [2]. An efficient planning and sampling algorithm translates the intent to one or more parallel sequences of actions, to be performed by virtual humans (agents) at specific locations and moments in time. Actions, relations and constraints are represented in a modular library, allowing incidents to be automatically composed of building blocks.

To demonstrate our approach, we developed a prototype, which simulates the inhabitants and daily behavior in the Dutch municipality of Rijswijk, driven by GIS and census data. Daily behavior is simulated by relatively simple models, adapting to environmental conditions such as time of day or season. Within this urban setting, we are able to seamlessly inject elaborate incidents at run-time.

2. INJECTING INCIDENTS

To introduce a new incident in the simulation, users specify a small number of high-level parameter settings. 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 these 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.

As a running example in the text below, we will use an incident "Arrest an Offender", in which one or more offenders are arrested by one or more police officers at a specified location and time.

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. In this way, quite elaborate incidents can be automatically composed by the planner, without the need for extensive scripting.

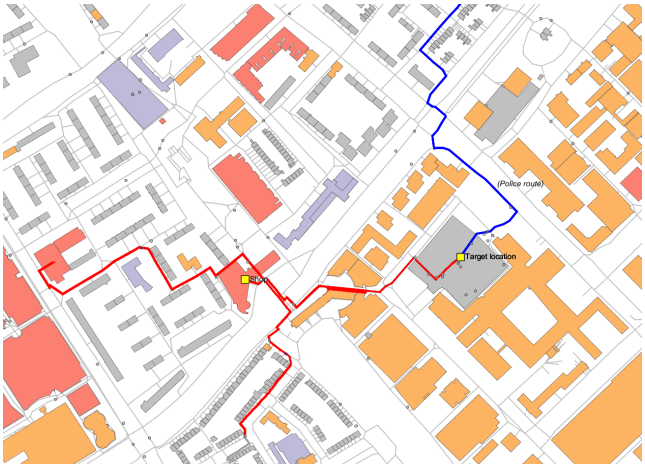
Actions are placeholders with a number of variables, i.e. an actor variable, an optional target variable, and a location/time variable, each with possible constraints. In the planning phase, these variables are not given a concrete value yet; instead, the planner makes a plan where the variables are 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 the behavior of the agents involved while the action is performed. We note that such a model may affect agents that are nearby via area triggers.

Initially, the planner investigates the enabling action of the incident. In the running example, the enabling action is Arrest, which shows one or more police officers arresting one or more offenders. The planner resolves any role preconditions that need to be satisfied to allow the enabling action. For the action Arrest, the actor must have the role Police Officer, while the target must have the role Offender. In case of a generic role, such as Offender, the planner also considers its specializations, such as Shoplifter. Searching for actions that give an actor or target the role Shoplifter, the action Shoplift might be found and added to the plan before the action Arrest; in this case, the actor variable of Shoplift is linked to the target variable of Arrest. Similarly, when resolving the role precondition concerning the Police Officer role, the planner may insert an action Activate Police before the action Arrest to create the required police officers. In this case, the actor variables of the two actions are linked.

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

* Corresponding author: steven.dejong@tno.nl.

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



(a)

(b)

Figure 1: Examples of run-time injected incidents: (a) an arrest of two shoplifters (red routes) by a police officer (blue route) occurs at the specified location, (b) an improvised protest march (green circles represent participants), which attracts some passers-by (gray circles) using an area trigger.

Movement between actions. Once a plan has been created that resolves all role preconditions, the planner enables agents to travel between locations. For this, the planner introduces movement actions, inserted before each action in the plan. The algorithm for inserting suitable moves takes into account two aspects. First, agents may have acquired a certain role as a result of actions they performed earlier (e.g., Police Officer or Shoplifter). Therefore, we need to select a type of movement that fits this role. The second aspect considered when inserting suitable move actions is the target of the action we move towards, e.g. a police officer should choose Pursuit over Flee or Walk for movement towards an offender. Once the most fitting move action is chosen, it is inserted in the plan.

Assigning agents and locations. The result of the planning algorithm described above is a sequence of actions (i.e., a plan), with correctly linked variables that can be instantiated by assigning agents and locations that are present in the environment. An algorithm based on sampling and iterative improvement now finds such 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).

Intuitively, the algorithm works by first sampling a random valid assignment for the locations and agent variables in the plan. A valid 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 action.

Then, while the plan still takes too long to execute, a number of improvement steps 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) one variable for a chosen action on the critical path will be assigned differently, in such a way that the plan is expected to take less time. 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 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.

3. DISCUSSION AND FUTURE WORK

Our contribution. 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. Examples are shown in Figure 1. Because incidents are planned on-the-fly based on a library of available actions and associated pre- and postconditions, new incidents can build upon content that is already present.

In addition to run-time incidents, the prototype system also implements a simulation of daily patterns-of-life, based on real-world census data. 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. 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.

Other work could focus e.g. on including other traffic modalities than pedestrians, or on allowing users to specify population behavior on the level of intent as well, e.g. by drawing a morning rush hour on a specific street, which influences population attributes such as residence and place of work to create the desired dense traffic.

Online repository. The current prototype system is available as open source [1].We encourage others to experiment with the framework and to extend it with novel models and algorithms.

References

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

2. 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.

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