PAStE: A Platform for Adaptive Storytelling with Events

Alexander Shoulson, Mubbasir Kapadia, and Norman I. Badler
Department of Computer and Information Science
University of Pennsylvania
Philadelphia, PA 19104-6389, USA
{*shoulson, mubbasir, badler*}*@seas.upenn.edu*

Abstract

We present ongoing work on PAStE, a system that supports a narrative director by describing a virtual world and managing the behavior of its characters. Rather than operating in the action domain of each individual actor, PAStE works at the scope of events – pre-authored multi-actor sequences describing interactions between groups of actors and props. A director can select which event to perform, and populate that event with characters in the virtual world. Once an event is chosen, PAStE handles the event's execution with support for high-fidelity character animation on fully-featured 3D virtual humans. Events are an accessible tool for assembling narrative arcs, invoking contextual activities, and conceptualizing user intervention. We explore the possibilities presented by an event-centric behavior environment for conducting stories, and address some of the potential limitations.

Introduction

Complex virtual worlds with sophisticated artificial actors present a unique opportunity for telling immersive interactive stories. With a rich behavioral repertoire, virtual characters can perform narrative roles, represent personalities, and convey information or emotion to a user. For high-fidelity expression, need characters with responsive controllers to react to intervention from human users, and adapt the structure of the narrative to incorporate external input. As virtual characters and the worlds they inhabit grow in complexity, so too do the decision structures needed to dictate which actions a character should take, at what time, and in what manner.

Traditionally, an increase in the number of actions a character can perform has an exponential effect on the computational cost of a decision-making engine. This imposes a challenge on our efforts to increase characters' expressiveness and add features like gesturing, locomotion, reaching, gaze tracking, facial animation, and other tasks suitable for a virtual human in a fully-realized 3D space. To achieve a large repertoire of character actions while mitigating the computational impact on those characters' controllers, we are developing a system that encapsulates behavior into structures with less granularity and more semantic meaning. Rather than exposing to a controller an intractably large list of actions that may hold little narrative significance on their own, PAStE uses the following two techniques:

Smart Objects. Using Smart Objects (Kallmann and Thalmann 1999), the way an actor can use an object, be it a static prop or another character, is encoded in the target object itself as a list of *affordances*. This makes it possible to add objects to the world without affecting any other behavior logic already in place, and to personalize each object's use to its unique geometry. With Smart Objects, a controller can prune actions with little contextual significance.

Event-Centric Behavior. Events are authored interactions between groups of agents and objects, often exhibiting cooperative or competitive behaviors. They are not merely action decompositions, but rather autonomous coroutines that temporarily control multiple actors as if they were limbs of some central agent. The behavior logic of an event comprises numerous atomic actions with control structures to enable synchronization, nondeterministic behavior, and permanent state changes in the objects involved, with an emphasis on enabling rich 3D character animation.

This system is not a narrative director. Instead, our environment is intended to *support* a director by exposing a library of events, giving that director the tools it needs for selecting which events are most relevant, and managing the execution of each event invoked. For this reason, events are designed to describe interactions with narrative value – the contribution of one gesture animation to the story is difficult to quantify, but a series of gestures representing an argument between actors can make a distinct impact on the plot. Rather than describing the world's action domain as all possible combinations of the characters' atomic actions, we expose externally the list of possible events and the actors and objects available for participation. This dramatically reduces the granularity of a narrative action sequence into more conceptually relevant atoms, and each event can be analyzed for narrative cost and effect before being made available. Because events represent temporary interactions, and many events can operate simultaneously in a world, they occupy a theoretical ground between atomic character actions, and

full scenes or encounters (Thue et al. 2007).

Our events, which are loosely based off of Smart Events (Stocker et al. 2010), represent compound behaviors spanning multiple participating actors and objects, and serve as more complex transitions in the control domain than atomic actions. Because of this, the event centric authoring paradigm presents two major challenges:

Authoring Burden. While events can be described with preconditions and effects like atomic actions, that state transition data may be more complicated than it would be for an atomic action. This information can be explicitly annotated in the event creation process, but doing so requires a significant manual effort. We can address this burden by performing an offline exploration of the Smart Object affordance domain in what we call the "Smart Object Laboratory", followed by an abridged simulation of the system's events. Doing so allows us to automatically infer the preconditions and effects of an event without any explicit author annotation at the event level and store them in a model for fast retrieval.

Computational Cost. Since events take a number of participating actors and objects to function, a controller deciding which event to execute needs to consider not only which event to select, but also which actors and objects to nominate as participants. The search for the correct event to execute can grow combinatorially unless mitigated. To address this complexity, we introduce a *salience* metric, which prioritizes candidates based on their previous participation in events, and has the side effect of creating "main characters" for a particular story.

The contributions in this paper lay the foundation for realizing an automated story director where a skilled author creates and populates the world with reusable smart objects, and end users can design events and higher-level story goals to create narratives in fully-realized 3D virtual worlds. The entire system is fully generalizable and reusable to create multiple dynamic narratives.

Related Work

Since our work is concerned with both interactive narrative and controlling fully-articulated 3D characters in a virtual world, the PAStE system draws from Crowd Simulation research and applies it to the domain of Interactive Narrative to bridge the gap between narrative and simulation.

Crowd Simulation. In addition to telling a story, we are also interested in creating a realistic, immersive environment with interesting ambient activity. Traditional approaches (Pelechano, Allbeck, and Badler 2008) for large-scale crowd simulation incorporate social force models (Helbing and Molnar 1995), reactive behaviors (Reynolds 1999), or hybrid solutions (Singh et al. 2011) to handle the characters' navigation and decision processes. Typically, characters in a crowd are simulated with low behavioral fidelity, driven by simple goals (Shao and Terzopoulos 2007) or heavy scripting (Massive Software Inc. 2010) that do not incorporate narrative objectives. Our work builds on traditional crowd simulation approaches by developing a narrative on the backdrop of a virtual populace. Characters in PAStE can move fluidly in and out of focus, participating in the story when needed for an event, and returning to simple autonomy when not involved in the narrative (Shoulson and Badler 2011). Using this hybrid approach, we can tell stories in "living" worlds populated by functional, purposeful agents.

Interactive Narrative. The field of interactive narrative has produced the concept of a virtual director or drama manager (Magerko et al. 2004). Virtual director systems are driven by a virtual agent responsible for steering the agents in the world towards predetermined narrative goals (Weyhrauch 1997). This area has been well studied, and a number of techniques exist for designing effective directors. Facade (Mateas and Stern 2003) executes authored beats to manage the intensity of the story, Mimesis (Riedl, Saretto, and Young 2003) employs narrative planning (Li and Riedl 2011) with atomic agent actions, Thespian (Si, Marsella, and Pynadath 2005) uses decision-theoretic agents to create actors with social awareness, while PaSSAGE (Thue et al. 2007) and the Automated Story Director (Riedl et al. 2008) monitor a user's experience through the story to choose between scenes and character behavior. Riedl and Bulitko provide a more detailed survey (Riedl and Bulitko 2013) of the current work in interactive narrative.

The bulk of research in interactive narrative focuses on the virtual director, which represents only part of the interactive narrative problem. Most of this work is concerned with the ordered selection of abstract, story-relevant action sequences to produce a narrative that responds to the actions of a user. PAStE is designed to support this focus. Our virtual world manages the ambient activity of a pool of "primordial" characters with little initial personality, and exposes to the director an interface for involving those characters in events with narrative significance. All of this activity is performed in a fully-realized 3D virtual environment that converts high-fidelity animations and procedural character controllers on fully articulated models into a series of abstract narrative events that a virtual director can comprehend.

Like PAStE's events, A Behavior Language (ABL) (Mateas and Stern 2004) provides a generalized scripting language for single- or multi-character actions based on manually authored preconditions for successful action execution. Multi-actor behaviors in ABL use a role-based negotiation process to determine factors like leader/follower relationships. PAStE events are based on Parameterized Behavior Trees (PBTs) (Shoulson et al. 2011), which share some similarities with ABL. Both ABL and PBTs allow for sequential and parallel control structures for synchronous actions. However, PAStE's PBT events differ from ABL in two key ways:

Character Interactions. ABL has support for character interactions, but its joint behaviors are still agent-centric and rely on an agent's autonomy. An ABL "follow the leader" event has two different versions, one for the leader, and one for the follower, that both agents execute individually. PAStE's PBT event version of "follow the leader" contains only one event behavior structure that suspends the autonomy of both the leader and follower,

and controls both exclusively for the duration of the event as limbs of a single entity. This centralizes the authoring process for complex synchronous interactions.

Behavior Metadata. ABL and PAStE events both require precondition and effect information in order to to be used effectively. In ABL, these are hand-authored in the behavior script, while PAStE learns this descriptive information without manual annotation. After experimenting with the world's objects in the Smart Object Laboratory, PAStE can simulate its events using combinations of arbitrary participant descriptions to determine whether an event will succeed or fail under certain conditions.

Problem Definition

We define our problem domain as $\Sigma = \langle \mathbf{S}, \mathbf{A} \rangle$, where $\mathbf{S}$ is the state domain, and $\mathbf{A}$ is the action domain. We define a single problem instance as $\mathbf{P} = \langle \Sigma, S_{start}, S_{goal} \rangle$ where $S_{start}, S_{goal} \in \mathbf{S}$ are the start and goal states. Note that having start and goal states makes the assumption that the virtual director using PAStE will employ some form of search for event selection.

State Domain

Each object (actors are objects with autonomy) in the world $\mathbf{W}$ is described as $o = \langle c, s, n, A \rangle \in \mathbf{W}$ where c is a controller, s is the object's individual state represented as a binary vector, A is a set of smart object affordances, and n is a set of pairs of type (r, o') where $r \in R$ is a relationship and $o \neq o' \in \mathbf{W}$. Individual state represents flags affecting only the owning object (`Sleeping`, `HasKey`, etc.) while relation entries represent relationships with other objects (`Friend(x, y)`, `Owns(x, y)`, etc.). Object type information is also encoded in its state, so that the individual state flags could contain entries like `IsActor`, `IsChair`, `IsTable`, and so on. We distinguish between s and n because n changes for a given object in the contexts of its interactions with other objects, while s does not.

Globally, the world state can be described as $S = \{(s_1, n_1), \ldots, (s_n, n_n)\}$, but a global controller will only rarely examine that world state. Instead, the story controller will generally examine subdomains of the world containing the states of the objects in a particular event, so that for an event e, $S_e = \{(s_1, n_1), \ldots, (s_m, n_m)\}$ where each (s_i, n_i) belongs to an object o_i participating in e.

Action Domain

Affordance Domain A smart object affordance for an object o is a function $a(o, o_u) : ((s_o, n_o), (s_{o_u}, n_{o_u})) \rightarrow ((s'_o, n'_o), (s'_{o_u}, n'_{o_u}))$ that takes in the object itself and another object o_u, the "user" of the affordance (not to be confused with a human user) and uses their controllers to modify their states and relations. The affordance can only add or remove relations under one of the following conditions: either the relation is of the form $(r, o) \in n_{o_u}$, or it is of the form $(r, o_u) \in n_o$ for some r. That is, during the affordance, an object can only add or remove a relation in the other object, and only referring to itself. Note that the activation of an affordance can persist over a period of time, and both the affordance user and the object being used have their autonomy suspended for the duration. An affordance can also fail during its execution, such as if the user or used object do not match certain state criteria.

Affordances represent the use or activation of an object. For instance, a chair might have a "sit" affordance that, when used by a character, directs the character to approach that chair and sit on it, writing to that character an `SittingOn` relationship. By preventing the affordance from modifying or writing references to any objects aside from itself and its current user, we limit the portion of the global state domain affected by the affordance and simplify the transition function created by using it.

Affordance State Encoding When evaluated during an affordance activation, the state of an object o is encoded as a binary vector with two regions. The individual state of an object, s_o, is already stored in binary and fills the first region of the bit vector, while the second region of the vector is generated dynamically based on the object's relations n_o. We represent the total binary encoding for an object o, relative to another object o' as $d_o(o') = [s_o | \delta(n_o, o')]$.

The region $\delta(n_o, o') = [e_0 \ldots e_{|R|}]$ is generated for o relative to another object o' in a given affordance. Specifically, let o be the owner of the affordance, and let o_u be the user of the affordance. Let $\delta(n_o, o_u) = [a_0 \ldots a_{|R|}]$ and $\delta(n_{o_u}, o) = [b_0 \ldots b_{|R|}]$. Then for each relation $r_i \in R$, $a_i = 1$ if and only if $(r_i, o_u) \in n_o$, and $b_i = 1$ if and only if $(r_i, o) \in n_{o_u}$. That is, each object's entry for each relationship is set to 1 if and only if that object contains an entry for that relationship referring to the other object involved in the affordance.

For example, suppose we have four individual state flags in our simulation: `IsActor`, `IsChair`, `Empty`, and `HasKey`, and the following relationships: `SittingOn`, and `Friend`. Let us define two characters a and b, and a chair c. Character object a has the following state information: `IsActor`, `HasKey`, and `Friend(a, b)`. Character object b has `IsActor`, `Friend(b, a)`, and `SittingOn(b, c)`. Chair object c only has `IsChair`, since b is sitting on it and it is not currently empty. We can produce the following encoded states:

$$d_a(b) = [1\,0\,0\,1 \mid 0\,1], \;\; d_a(c) = [1\,0\,0\,1 \mid 0\,0]$$
$$d_b(a) = [1\,0\,0\,0 \mid 0\,1], \;\; d_b(c) = [1\,0\,0\,0 \mid 1\,0]$$
$$d_c(a) = [0\,1\,0\,0 \mid 0\,0], \;\; d_c(b) = [0\,1\,0\,0 \mid 0\,0]$$

By encoding relationships in this way, we reduce the impact a single affordance can have on the world state, compartmentalizing the affordance domain into manageable regions. Rather than considering a large set of relationships for each object with every other object, we restrict the information that is encoded and made available to both the affordance itself and any higher-level controller in charge of activating the affordance. Since affordances are unable to write to or reference any objects other than the two immediate participants, the scope of their possible effect on the state of the world is very limited.

It is important to note that the high-level state of an object (i.e., s and n) is very much an abstraction of its actual state

in the world. Objects in our virtual world contain a wealth of information pertaining to factors like animation, inverse kinematics, and geometry. The state domain for our problem ignores most of these details, so the affordance functions themselves are responsible for making sure that an underlying state change in the character is reflected with a change in the high-level state of that character object. We treat two characters with the same high-level state (i.e., their encoded state vectors are equal) in the same affordance context as functionally identical. That is, if objects u and v both use affordance a of object o, and $(s_u, \delta(n_u, o)) = (s_v, \delta(n_v, o))$, then $u \equiv v$ relative to affordance a. As far as affordance a is concerned, u and v are interchangeable and should produce the same result upon activation with o, even if the details of their condition within the virtual world differ at a lower level.

Event Domain The top-level action domain in PAStE is based on the event domain. Behaviorally, an event is a process that temporarily suspends the autonomy of any participating objects, carries them through an interaction, and then restores those objects' autonomy once the event completes. Events are manually authored and stored in an event library. In PAStE, the event's behavior logic is encoded as a PBT controlling a sequence of affordance activations. Note that behavior trees ultimately report success or failure, and an event's tree may fail independently of the precondition function (though ideally this would not occur).

In the problem domain, each event is defined as

$$e = \langle t, c, \phi : \mathbf{W}^n \rightarrow \{0, 1\}, \Delta : S_e \rightarrow S'_e \rangle$$

where the t contains the event behavior, c is the event's cost, the precondition function ϕ transforms a selection of n objects from the world into a true or false value, and the postcondition function Δ transforms the event state subdomain as a result of the event. The transition information for an example event that instructs an actor to unlock a door would take two objects, have preconditions such as "Object 1 is a character", "Object 2 is a door", "Door is closed", and "Door is locked", and effects such as "Door is unlocked". Our goal is to produce a system where an author designs only t and part of c, while the rest of an event's information is discerned from pre-processing.

The final action domain, however, has one more complication. Rather than only considering which event to execute, a controller must also select participants. Once a candidate event is selected from an authored library of events, the event must also be populated with n participating objects. The n objects must be selected from the world, and the precondition function ϕ must be evaluated on the selection. The precondition function factors in both an objects's individual state, and its relationships to the other candidate objects. In the earlier example with characters a and b, and chair c, the total encoded state of the objects (a, b, c) passed to an event would be a concatenation of their three individual states, followed by their six relative relationship states, as follows:

$$[1\ 0\ 0\ 1][1\ 0\ 0\ 0][0\ 1\ 0\ 0][0\ 1][0\ 0][0\ 1][1\ 0][0\ 0][0\ 0]$$

Order matters in this encoding, and the input would differ if the objects were given as (b, a, c). Because of this, the precondition function needs to be aware not only that an object has been nominated, but which role in the event the object would take on. As a result, our worst case for picking n objects for one event would be $\frac{|\mathbf{W}|!}{(|\mathbf{W}|-n)!}$. We will introduce some ways to avoid this combinatorial growth.

Goals The goal of PAStE in its current form is not to satisfy story goals, but to expose the environment in a serviceable form to an attached virtual director that can. Ultimately it is the director's responsibility to select events and participants, but PAStE can help in this process. Rather than specifying a desired value for the composite world state, goals can be predicated on the existence of an object with a given state. For instance, we can specify that there exists an object in the world with a certain set of flags, or that for a specific object in the world, certain conditions hold on the state of that object. A story goal could be for a character to be holding a certain prop, or for two characters to gain an `Friend` relation. All of this information is made readily available by the PAStE system.

Exploring the Affordance Domain

By definition, we encapsulate all character actions into affordance activations, where a virtual actor can activate affordances on itself, other actors, or non-autonomous props in the environment. An affordance has only two participants, the activator and the object being activated, and generally comprises multiple mechanical tasks (navigation, reaching, gazing, gestures, etc.) to accomplish one objective (such as coming to sit on a chair, or picking up an object from a table). The affordances of a smart object are manually authored by an expert user, and given handwritten preconditions and effects. An event can have an arbitrary number of participants, and generally expresses complex behavioral phenomena (such as a group conversation or a riot). All higher level behavior in an event is authored as a series of affordance activations with additional control structures for decision-making, synchronization, and so on. However, since events involve more participants, and represent more complex behavior than an affordance, their preconditions and effects are likely to be more complex and difficult to manually author. Fortunately, if the system understands the preconditions and effects of each affordance, and events are presented as sequential or simultaneous affordance activations, then the preconditions and effects for an event can be learned. This reduces authorial burden and allows less trained authors to create events.

Once a world is designed with character and object archetypes, our first task is to run a series of simulations to exhaustively explore the affordance domain for the objects designed by the author. This exploration task operates on a reduced world called the Smart Object Laboratory (SOL). The reduced world does not require complete functionality emulating the full simulation space, as the goal of the SOL is only to learn the behavior of each object. In practice, we expect a simulation space to simply be a line-up of each object archetype (including other actors in different configurations), with one or more exemplar character(s) to experiment with each object. Creating the laboratory would be very simple to make alongside the intended full simulation

environment, requiring only the placement of each object in a position that can be reached. The character attempts all sequences of interactions with all of the available objects, until the behavior of each object (carrying state changes from one object to the next) until all new discoveries are exhausted.

The process for exploring the SOL is illustrated in Algorithm 1. The algorithm walks through the affordance domain, branching whenever it encounters previously unseen combinations of object, affordance, and state. Whenever the algorithm encounters a situation identical to a seen example (defined by equality over d_o and d_{o_s}), it ceases that branch. Recall that we consider two objects to be functionally identical if their encoded states match – this is the property that allows our search to terminate. The final result of the simulation is a set T of transition records $\{(o, a, (d_o, d_{o_u}) \rightarrow (d'_o, d'_{o_u})\}$ over all affordances a belonging to all object archetypes o, over all possible state encodings d_o and d_{o_u} for o and candidate user object o_u. Note that the state serialization is an optimization that is not required for the algorithm to function properly.

Data: simulation world object list $\mathbf{W}'$
Data: a sampling object o_s
Result: transition record T
create sets O, C;
serialize current world state S;
foreach $o \in \mathbf{W}'$ **do**
  foreach $a \in A_o$ **do**
    $O = O \cup \{(S, o, a)\}$;
    $C = C \cup \{(o, a, d_o, d_{o_s})\}$;
while $|O| > 0$ **do**
  select (S, o, a) from O ;
  set current world state to S ;
  record $t_{in} = (d_o, d_{o_s})$;
  execute $a(o, o_s)$;
  serialize current world state S';
  if a *is successful* **then**
    record $t_{out} = (d'_o, d'_{o_s})$;
    foreach $o' \in \mathbf{W}'$ **do**
      foreach $a' \in A_{o'}$ **do**
        let c = (o', a', d'_o, d'_{o_s});
        if $c \notin C$ **then**
          $O = O \cup \{(S', o', a')\}$;
          $C = C \cup \{c\}$;
    $T = T \cup \{(o, a, t_{in}, t_{out})\}$;

Algorithm 1: Exploring the affordance domain.

As long as a full representative set of examplar objects, with all of their starting configurations, is tested on all starting character configurations, this will generate an exhaustive coverage of the affordance domain. If a state transition is not present in the final database, then it cannot be achieved from the starting configuration of the world, irrespective of user input (which is also bound to the affordance domain). The process, as it appears in our engine, is displayed in Figure 1.

Discovering the Event Domain

An understanding of the affordance domain allows us to approximate the transition function for each event. To do so, we create replace each event's affordance activations with the modeled transition functions found in the SOL. Instead of fully executing the affordance in the virtual world, these "proxy" events transform the states of their participants according to how each affordance would. If an object's state cannot be found as a valid input for that affordance's transition function model, then the event is treated as a failure. If the event terminates successfully, having executed all of its affordances and transformed the states of its participants, we store the input and output states of the participants into a new table to produce a model of the event's transition function, like we do with affordances.

Evaluating all permutations of input states as input would be prohitively expensive. Suppose we have an event taking a objects with N_t total unary state flags and N_r total relation flags in their encoding. Each of the a objects has $a - 1$ possible encoded relationship values, one relative to each of the other objects. The number of possible encoded inputs to the event then is $2^{(aN_t)+(a(a-1)N_r)}$. This grows too fast to exhaustively enumerate, so we perform static analysis on the event tree and detect all of the possible first affordances each object could be instructed to activate. If the event is nondeterministic, an object could have multiple first affordances. For all of those affordances, we collect their valid input states, and evaluate the event on that collection. This number will be much smaller than the worst case estimate, as each event will only have a small number of starting affordances, and those affordances will have a small number of valid inputs. This ensures coverage of all the ways an event could possibly succeed, without wasting time on object configurations that will fail on the first affordance invocation.

Creating a model for an event's transition function is still a storage and retrieval challenge. Because only a certain number of input configurations will allow an event to execute, the transition function's input and output values will be large and sparse. This is the subject of ongoing work, exploring two techniques. First, we can keep track of which flags in each object's state are actually read or written by an affordance or an event, which would allow us to mask out and ignore data that is never used. Second, we intend on producing an approximation based on model reduction methods, using a technique like principal component analysis rather than just a lookup table.

Reaching a Goal

Once we have a transition function for the event domain, we can project the outcome of the event and the new states that those objects will take on. We can also predict if a set of candidate objects is a valid selection for an event, since we know under what state conditions the event will succeed and fail. These are the main tools that PAStE exposes to an attached virtual director. The process of achieving a narrative goal is to execute successive events until the result of one of those events places a number of objects in a desired configuration. With events, however, the action domain is more

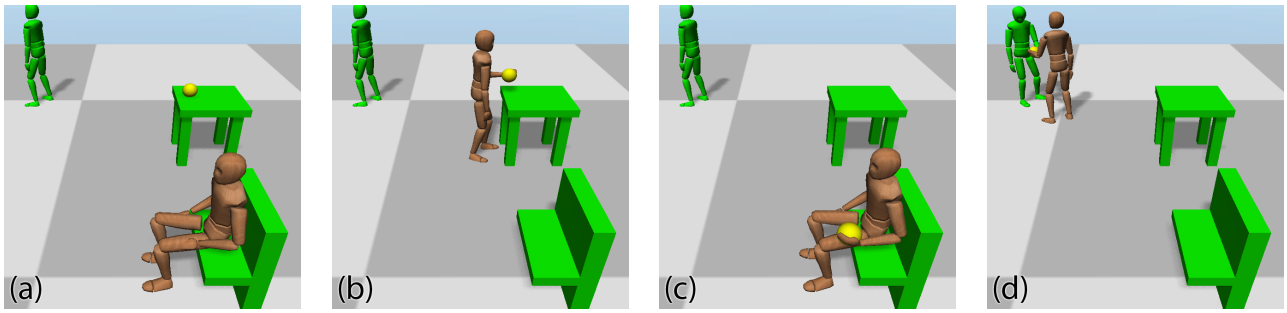

Figure 1: Affordance domain exploration. The actor (a) sits, then (b) picks up an object and tries both (c) sitting with that object and (d) handing that object to another dummy actor. We use Unity and the ADAPT platform (Shoulson et al. 2013) for visualization.

than just a selection of which events to perform, but also which objects in the world are selected to participate in those events. The director could naively select all combinations of objects from the world to explore the action domain for all events, but this is intractable for rich worlds with numerous props and actors. One way of reducing this complexity is to divide objects into roles, and author role requirements into an event's parameter list. As a pre-processing step before runtime, we can divide all of the world objects into bins by archetype, creating lists of objects each filling a particular role in the narrative. An event would then specify that its first object must be of a certain type, its second object of the same or another type, and so on. For an event e with n participating objects, this reduces the possible combinations of valid participants from $|\mathbf{W}|^n$ to $r_1 \cdot r_2 \cdots r_n$, where r_i is the number of objects matching the i^{th} role required by e. Thus, the number of candidates would be small for a diverse environment.

One reason that a large number of candidates is undesirable is that all candidate objects are equal in value to the controller. The controller has no knowledge of what the state of an object means, other than seeing a binary vector encoding. To reduce the uniformity of the objects in the world, we introduce a metric called *salience*. As objects (actors or props) participate in events, they are given a growing salience weight. A controller searching through the action domain is then highly incentivized to use objects and characters with higher salience values. This metric can be easily integrated into exsiting control frameworks, for example: as a reward function in Markov decision processes, or as a heuristic to guide the search for automated planners. Salience has two parts: a method of events leaving residual information in their participants, and a search heuristic that weights higher actors and objects that have already been featured in events. This creates a phenomenon we call *progressive differentiation* (Shoulson, Garcia, and Badler 2011), where characters begin as primordial actors with little individual qualities, and through involvement in events, pick up traits and qualities that contribute to their involvement in subsequent events. Salience does not mitigate preconditions – a character still requires a key to unlock a door, but by incentivizing the use of salient characters, our hope is that this would drive a controller to select a "main character" when it needs an actor to retrieve a key and use it. We expect a single character participating in three narrative events is to matter more to the user than three characters participating in one event each. There are other ways we can limit the branching factor induced by the number of possible candidates to an event, including priming and restricting selection to a geometric radius (Stocker et al. 2010).

Conclusions

PAStE is designed to take an environment, populate it with interesting props and a virtual populace of functional actors, and cultivate a fertile environment for telling interesting stories and develop compelling characters. However, PAStE is just one part of the interactive narrative ecosystem, and we would like to explore ways to create a virtual director that takes full advantage of the PAStE toolkit. As we do so, we have a number of open questions to address going forward:

How do we continue to control the growth of the action domain? Given the combinatoric effect of event parameters, we will need to limit the growth of the event action domain. We can control this in several ways, including participant selection restrictions, and dividing the problem domain into subdomains (Kapadia et al. 2011).

What is the likelihood of an event leading us to the goal state? To reach a goal state, a controller must apply successive events to the virtual populace, until one or more objects exhibit(s) a desired object configuration. How do we compute the effectiveness of a given event on a given subset of the population?

How do we handle non-determinism? Up until this point we have assumed that the transition functions for affordances and events are deterministic. However, virtual worlds are chaotic systems, where navigation and procedural controllers can fail, and events themselves can make stochastic choices.

Answers to these questions and others will inform the development of a more comprehensive event-centric narrative controller capable of telling stories in a dynamic, interactive, and living virtual world.

References

Helbing, D., and Molnar, P. 1995. Social force model for pedestrian dynamics. *PHYSICAL REVIEW E* 51:42–82.

Kallmann, M., and Thalmann, D. 1999. Modeling objects for interaction tasks. In Arnaldi, B., and Hgron, G., eds., *Computer Animation and Simulation '98*, Eurographics. Springer Vienna. 73–86.

Kapadia, M.; Singh, S.; Reinman, G.; and Faloutsos, P. 2011. A behavior-authoring framework for multiactor simulations. *IEE CGA* 31(6):45 –55.

Li, B., and Riedl, M. 2011. *Creating Customized Game Experiences by Leveraging Human Creative Effort: A Planning Approach*. Springer. 99–116.

Magerko, B.; Laird, J. E.; Assanie, M.; Kerfoot, A.; and Stokes, D. 2004. AI Characters and Directors for Interactive Computer Games. *Artificial Intelligence* 1001:877–883.

Massive Software Inc. 2010. Massive: Simulating life. www.massivesofware.com.

Mateas, M., and Stern, A. 2003. Integrating plot , character and natural language processing in the interactive drama facade. In *Proceedings of the 1st International Conference on Technologies for Interactive Digital Storytelling and Entertainment TIDSE03*, volume 2.

Mateas, M., and Stern, A. 2004. A behavior language: Joint action and behavioral idioms. In *Life-Like Characters*. Springer. 135–161.

Pelechano, N.; Allbeck, J. M.; and Badler, N. I. 2008. *Virtual Crowds: Methods, Simulation, and Control*. Synthesis Lectures on Computer Graphics and Animation. Morgan & Claypool Publishers.

Reynolds, C. 1999. Steering behaviors for autonomous characters. In *Game Developers Conference 1999*.

Riedl, M. O., and Bulitko, V. 2013. Interactive narrative: An intelligent systems approach. *AI Magazine* 34(1):67–77.

Riedl, M. O.; Stern, A.; Dini, D.; and Alderman, J. 2008. Dynamic experience management in virtual worlds for entertainment, education, and training. *International Transactions on Systems Science and Applications, Special Issue on Agent Based Systems for Human Learning* 4(2):23–42.

Riedl, M. O.; Saretto, C. J.; and Young, R. M. 2003. *Managing interaction between users and agents in a multi-agent storytelling environment*, volume 34. ACM Press. 186–193.

Shao, W., and Terzopoulos, D. 2007. Autonomous pedestrians. *Graph. Models* 69:246–274.

Shoulson, A., and Badler, N. I. 2011. Event-centric control for background agents. In *ICIDS*, 193–198.

Shoulson, A.; Garcia, F.; Jones, M.; Mead, R.; and Badler, N. 2011. Parameterizing behavior trees. In *MIG*, volume 7060. Springer. 144–155.

Shoulson, A.; Marshak, N.; Kapadia, M.; and Badler, N. I. 2013. Adapt: the agent development and prototyping testbed. In *Proceedings of the ACM SIGGRAPH Symposium on Interactive 3D Graphics and Games*, I3D '13, 9–18. New York, NY, USA: ACM.

Shoulson, A.; Garcia, D.; and Badler, N. 2011. Selecting agents for narrative roles. In *INT4*.

Si, M.; Marsella, S. C.; and Pynadath, D. V. 2005. Thespian: An architecture for interactive pedagogical drama. In *Proceeding of the 2005 Conference on Artificial Intelligence in Education*. 595–602.

Singh, S.; Kapadia, M.; Hewlett, B.; Reinman, G.; and Faloutsos, P. 2011. A modular framework for adaptive agent-based steering. In *ACM SIGGRAPH I3D*, 141–150.

Stocker, C.; Sun, L.; Huang, P.; Qin, W.; Allbeck, J. M.; and Badler, N. I. 2010. Smart events and primed agents. In *IVA*, 15–27.

Thue, D.; Bulitko, V.; Spetch, M.; and Wasylishen, E. 2007. Interactive storytelling: A player modelling approach. In *AIIDE*.

Weyhrauch, P. W. 1997. *Guiding interactive drama*. Ph.D. Dissertation, Pittsburgh, PA, USA. AAI9802566.