1-1-2015

Real-Time Storytelling with Events in Virtual Worlds

Alexander Shoulson
University of Pennsylvania, ashoulson@gmail.com

Real-Time Storytelling with Events in Virtual Worlds

Abstract

We present an accessible interactive narrative tool for creating stories among a virtual populace inhabiting a fully-realized 3D virtual world. Our system supports two modalities: assisted authoring where a human storyteller designs stories using a storyboard-like interface called CANVAS, and exploratory authoring where a human author experiences a story as it happens in real-time and makes on-the-fly narrative trajectory changes using a tool called Storycraft. In both cases, our system analyzes the semantic content of the world and the narrative being composed, and provides automated assistance such as completing partially-specified stories with causally complete sequences of intermediate actions. At its core, our system revolves around events -â?? pre-authored multi-actor task sequences describing interactions between groups of actors and props. These events integrate complex animation and interaction tasks with precision control and expose them as atoms of narrative significance to the story direction systems. Events are an accessible tool and conceptual metaphor for assembling narrative arcs, providing a tightly-coupled solution to the problem of converting author intent to real-time animation synthesis. Our system allows simple and straightforward macro- and microscopic control over large numbers of virtual characters with diverse and sophisticated behavior capabilities, and reduces the complicated action space of an interactive narrative by providing analysis and user assistance in the form of semi-automation and recommendation services.

Degree Type
Dissertation

Degree Name
Doctor of Philosophy (PhD)

Graduate Group
Computer and Information Science

First Advisor
Norman I. Badler

Keywords
Artificial Intelligence, Computer Graphics, Interactive Narrative, Storytelling, Virtual Worlds

Subject Categories
Computer Sciences

REAL-TIME STORYTELLING WITH EVENTS IN VIRTUAL WORLDS

Alexander Shoulson

A DISSERTATION

in

Computer and Information Science

Presented to the Faculties of the University of Pennsylvania

in

Partial Fulfillment of the Requirements for the

Degree of Doctor of Philosophy

2015

Acknowledgements

The use of “we” is not a figure of speech when describing this work. The work done here was made possible by numerous teams over several years on multiple individual projects. Norman I. Badler and Mubbasir Kapadia provided consistent mentoring, supervision, and co-authorship on nearly every publication from which this document draws. In addition, individual projects were made possible with the help and co-authorship of Francisco Garcia, Cyril Steimer, Samuel Oberholzer, Kai Ninomiya, Max Gilbert, Nathan Marshak, Robert Mead, Matthew Jones, and Daniel Garcia.

In addition, I would like to thank Ben Sunshine-Hill, Aline Normoyle, John Drake, Pengfei Huang, Joe Kider, Cory Boatright, and Tiantian Liu for help with advice, bouncing ideas around, and rehearsal presentations during the course of my time in the SIG Lab. Similarly I would like to thank Seth Frey, Fabio Zünd, Bob Sumner, Fabian Hahn, and Antoine Milliez and everyone else at Disney Research Zürich for our numerous discussions during my work there. Particularly, I would like to thank Mairuzio Nitti and Alessia Marra for their art assets used in the Storycraft and CANVAS projects.

All of this work is due in great part to the teachers and mentors that have brought me to this point. I would like to thank Stu Hirshfield, Alistair Campbell, and Rick Decker at Hamilton College for fostering my love for Math and Computer Science, and guiding me into this field in the first place. I am grateful to Larbi Mouedden at EA Tiburon and to Kevin Dill for introducing me to the video game industry and guiding me through the first steps. To my professors at Penn and elsewhere – Ani Nenkova, Ben Taskar, Sampath Kannan, Milo Martin, and Matt Blaze, and especially those willing to serve on my committee – Steve Lane, Ladislav Kavan, Lyle Ungar, and Mark Riedl from Georgia Tech. Finally, I would like to thank Mike Felker and Brittany Binler for greatly simplifying my life as a graduate student.

Most importantly, I owe a great debt of gratitude to my family, my parents, my grandparents, and my siblings for bringing me through this particular journey from start to typeset finish. I have been lucky to have you all with me in Philadelphia while working on my degree, and was much better off for it.

ABSTRACT

REAL-TIME STORYTELLING WITH EVENTS IN VIRTUAL WORLDS

Alexander Shoulson

Norman I. Badler

We present an accessible interactive narrative tool for creating stories among a virtual populace inhabiting a fully-realized 3D virtual world. Our system supports two modalities: assisted authoring where a human storyteller designs stories using a storyboard-like interface called CANVAS, and exploratory authoring where a human author experiences a story as it happens in real-time and makes on-the-fly narrative trajectory changes using a tool called Storycraft. In both cases, our system analyzes the semantic content of the world and the narrative being composed, and provides automated assistance such as completing partially-specified stories with causally complete sequences of intermediate actions. At its core, our system revolves around events – pre-authored multi-actor task sequences describing interactions between groups of actors and props. These events integrate complex animation and interaction tasks with precision control and expose them as atoms of narrative significance to the story direction systems. Events are an accessible tool and conceptual metaphor for assembling narrative arcs, providing a tightly-coupled solution to the problem of converting author intent to real-time animation synthesis. Our system allows simple and straightforward macro- and microscopic control over large numbers of virtual characters with diverse and sophisticated behavior capabilities, and reduces the complicated action space of an interactive narrative by providing analysis and user assistance in the form of semi-automation and recommendation services.

Contents

Acknowledgements **ii**

Contents **v**

List of Tables **ix**

List of Figures **x**

1 Introduction **1**

1.1 Story Participation 2

1.2 Story Control 5

1.3 Contributions 7

1.4 Applications. 8

2 Related Work **10**

3 Architecture **19**

3.1 Events 23

3.1.1 Characters Interacting with Each Other 24

3.1.2 Scheduling 25

3.1.3 Characters Interacting with the Environment 27

4 ADAPT: The Agent Development and Prototyping Testbed 29

4.1 Framework 31

4.2 Shadows in Full-Body Character Animation 33

4.3 Choreographers 34

4.4 The Coordinator 35

4.5 Using Choreographers and the Coordinator 38

4.6 Example Choreographers 39

4.7 Character Behavior 41

4.8 Body Capabilities 44

4.9 Computational Performance 45

5 PAStE: A Platform for Adaptive Storytelling with Events 47

5.1 Problem Definition 48

5.2 State Domain 48

5.2.1 Smart Object Affordances 49

5.2.2 Affordance State Encoding 50

5.2.3 Rules and Inference 52

5.3 Events and the Action Domain 53

5.3.1 Smart Object Groups 55

5.3.2 Event Instances, the Action Domain 56

5.3.3 Goals 56

5.4 Exploring the Affordance Domain 57

5.4.1 Discovering the Event Domain 60

5.4.2 Reaching a Goal 61

5.5 Progressive Differentiation 62

6 CANVAS: Computer-Assisted Narrative Animation Synthesis 64

6.1 Defining a CANVAS Story 65

6.1.1 Event Authoring and Story Structures 65

6.1.2 The Story Sequence 67

6.2 Handling Underspecified Stories 69

6.3 Parameter Filling for Incomplete Story Arcs 71

6.4 Story Inconsistency 72

6.4.1 Local Inconsistency Resolution 73

6.4.2 Global Inconsistency Resolution 75

7 Real-Time Story Exploration via Storycraft 81

7.1 Overview 81

7.1.1 Sentiment Tagging and Analysis. 82

7.1.2 Generative Storytelling. 83

7.1.3 Intervention Storytelling. 84

7.2 The Story Web 85

7.2.1 Pre-computation 86

7.3 Story Web Analysis 87

7.3.1 Low-level Features 88

7.3.2 Heuristics and High-level Features 89

7.3.3 Sentiment Tagging and Qualitative Analysis 91

7.4 Story Generation 91

7.4.1 Generative Storytelling 93

7.4.2 Intervention Storytelling 94

8 Results 95

8.1 Objects, Actors, and the Environment 95

8.2 Story Events 97

8.3 Scenario Creation Effort 98

8.4 Explicitly Authored Narratives 100

8.4.1 Fully Manual Authoring 100

8.4.2 Automated Parameter Selection 101

8.4.3 Global Inconsistency Resolution 101

8.4.4 Fully Automated Narrative Generation 102

8.5 Dynamically Exploring Narratives 103

8.5.1 Metrics 105

8.6 Independent Authors 106

9 Conclusions and Discussion **113**

9.1 Future Directions 115

Bibliography **118**

List of Tables

8.1 Metadata definition of some events used in the bank scenario. 109

List of Figures

2.1 An example deployment of a semi-automated forces (SAF) system. . . . 16

3.1 An illustration of the functionality of our system at runtime. 20

3.2 An illustration of the solution stack, using four discrete tools organized into three operational tiers. 21

3.3 A simple conversation PBT controlling two characters, `a1` and `a1`, with a `MeetingPoint` parameter. 25

3.4 The state machine for a chair smart object's "sit" affordance. 28

4.1 Overview of ADAPT framework. 30

4.2 Blending multiple character shadows to produce a final output skeleton pose. 34

4.3 A sample dataflow graph we designed for evaluating ADAPT. 37

4.4 The ADAPT Character Stack. 42

4.5 Update frequency for the character animation and navigation components in ADAPT. 45

5.1 Two sets of group activity. 55

5.2 Affordance domain exploration. 59

6.1 CANVAS Overview. 64

6.2 An illustration of the various arcs and vertices that describe a Story Sequence Diagram. 68

7.1 An illustration of the story web pre-computation process. 83

7.2 An illustration of the real-time storytelling process. 85

8.1 A sketch of the layout for the bank scenario. 108

8.2 A complex narrative authored using CANVAS. 110

8.3 Filling in an incomplete story specification. 111

8.4 A globally inconsistent story definition. 112

8.5 A screenshot of the virtual world and Storycraft interface overlay. 113

8.6 Close-up view of the storytelling dialog showing sentiment and event selection. 113

Chapter 1

Introduction

Interactive virtual worlds present a unique opportunity for simulation, training, and entertainment in settings that are otherwise prohibitive to recreate or control. One way to produce a more immersive experience is to populate these 3D environments with fully articulated virtual humans capable of exhibiting functional, purposeful behavior and rich interactions with other objects in the environment. Complex virtual worlds with sophisticated artificial actors present a unique opportunity for telling immersive interactive stories. With a large behavioral repertoire, virtual characters can perform narrative roles, represent personalities, and convey information or emotion to a human viewer. These types of settings create a fertile setting for dynamic storytelling, where sequences of occurrences in the environment are designed and carried out with overarching *narrative intent* rather than simply emerging from the individual motivations of the digital actors.

Purpose. This work is designed to facilitate the creation of complete interactive virtual worlds from the lowest-level animation controllers to the high-level abstract narrative intent defining a story scenario. While individual aspects of the virtual world experience (character animation, story generation, etc.) are well explored in isolation, there is a significant lack of research on systems capable of deeply assimilating these discrete disciplines into one cohesive experience that can fully exploit the state of the art in each field. Integration between bodies of research in these areas is sometimes considered an afterthought, while in reality the task of incorporating meaningful narrative and lifelike animation is non-trivial. Our work explores and addresses the problems that lie *within* and, perhaps more importantly, *between* the story control and animation synthesis tasks involved in creating a virtual world.

By far the most significant problem presented in authoring interactive narratives is in empowering an author to communicate narrative intent to the system in such a way that that abstract story can be converted into the moment-by-moment joint actuations on a virtual actor's body in real time. To give our authors the power to bring stories to life in a virtual world, we use a conceptual structure we call the *event*. Where the term "event" is typically used to described *observed occurrences*, our events are an accessible metaphor for describing *intended outcomes* (i.e., scripted interaction sequences) – authors dictate the events they expect to see in a scenario, and it becomes the responsibility of our authoring tool to ensure their appearance and animate their results.

1.1 Story Participation

A key goal of *interactive* narrative is to create environments where users can interact with the computer-controlled characters and act with agency in the world to change the story's outcome. However, because of the dynamic and mercurial nature of human interaction, the authoring process for these real-time stories can be prohibitively expensive in terms of both time and required expertise. The techniques by which human users can interact with these kinds of dynamic stories is increasingly well explored, within both academia and the entertainment industry, but the task of designing the stories themselves, particularly with rapid prototyping and feedback, remains elusively arduous. We are interested in designing a system where authors with minimal training can easily *create* interactive stories in large, continuous environments full of ambient human activity that unfold in real-time for the benefit of a human story participant. For high-fidelity expression, we need characters with responsive controllers to react to intervention from human users and adapt the structure of the narrative to incorporate this external input. Furthermore, fully simulating every virtual actor is computationally expensive, and wastes resources on characters that may be less important than others, so virtual characters from the ambient populace can be "promoted" in our system on an as-needed basis and augmented with richer behaviors that advance a story's plot. Our key goal is to allow an author to compose a compelling story using a cast of characters that all begin as simple pedestrians, but that can be upgraded at runtime to exhibit rich behaviors relevant to the narrative. This facilitates immersion by affording the author more complete ownership of his or her story based on which characters he or she chooses to include in the plot, and in what manner.

In our system we discuss two kinds of user – the *author*, upon whom our work focuses, and an end-user *participant* who interacts with the story in real-time after the author has crafted it. The system also relies on a third user, a *domain expert*, who designs the rules of the virtual world itself, but this domain expert is also not the focus of our work. In traditional media the participant user role would be represented by a reader or viewer, but interactive narrative blurs the lines between the creator/consumer roles by affording the consumer agency in and control over the creator's content. Similarly, the *author* most closely parallels the role of a screenwriter, and the domain expert could be considered the equivalent of a set designer or visual effects engineer.

This form of adaptive storytelling is specifically suited for *open world* or *free-roam* environments that place an emphasis on the participant's ability to explore and interact with a variety of world features. Our system cultivates a "primordial soup" of undifferentiated characters in the world that can transition from background ambience to pivotal actors in a long-form narrative. In so doing, we develop the ability for the author to design, and the participant to encounter compelling narratives at any point in the virtual world without the computational cost of fully simulating every character from the start. This draws inspiration from Alibi Generation (AG) [85], but where AG focuses on justifying a character's *past* observed behavior, we want to use a human-designed story as a blueprint for enriching *future* behavior by temporarily enhancing a character's complexity when needed.

There are a number of limitations that must be surmounted before a robust and fully realized open world interactive narrative system can be created. First, from a graphics perspective, virtual actors must be able to exhibit a rich repertoire of individual capabilities, with specialized controllers for navigation, locomotion, gaze tracking, reaching, gesture animation, and others. Second, specialized control structures are needed to facilitate both individual character autonomy and the coordination of complex multi-character interactions with a high degree of control fidelity. The author needs tools to control character coordination at varying levels of granularity, from two characters exchanging a prop to large groups of characters participating in a riot. Once the author crafts the story and conveys his or her intent to the system, the system must have the ability to carry out the author's story design by use of an automated virtual *director*. That director (sometimes called a drama manager) is a virtual agent that reasons about the objects and characters in the world, and the narrative impact of different character capabilities in order to produce animated action sequences that accomplish predetermined story goals and to decide which characters should participate in accordance with the author's instructions.

1.2 Story Control

Traditional interactive narrative approaches typically focus on using a small cast of characters to tell a single story. However, for large environments containing an entire populace of diverse and sophisticated characters, it is rare for a single narrative thread to continuously tie those characters into a cohesive story. Few traditional stories manage to contain an ensemble of dozens of characters that each significantly contribute a pivotal role in a single overarching plot. For these sorts of open-world environments, the narrative connections must consist of numerous interwoven story "threads" that combine to convey a sense of meaning, cohesion, and causality to a user's experience in the world. A single story "plan" or dictated story arc will not always suffice for this many potential principal characters in an interactive setting.

We address this problem in two ways. For offline story authoring, we use automation and computer assistance to mitigate the complexity of the virtual world and create richer stories with less effort. In addition, our solution also provides the opportunity involve the human author "in the loop" as part of a real-time authorial control process while the story progresses. This combined approach accomplishes two goals. First, since managing a populace of characters in a narrative setting is a demanding task, we can better leverage the unique creativity of a human storyteller rather than a completely automated system. This can help enrich the experience for the story participants and make for more variation between distinct storytelling sessions. Second, while automated storytellers are an important tool, they emphasize the merit of being *in* the story as a character participant, while discarding to an extent the value of the story*telling* experience. Our author-centric approach recognizes that telling stories is an important aspect of day-to-day human communication, and fully automated directors disconnect the creator of the story from the recipient by limiting the accessibility of story specification. In effect, we aim to democratize the digital storytelling experience with accessible tools for creating narrative.

Authoring stories in our system can be done in many different modalities. First, stories can be created offline using an intelligent storyboarding tool. This interface eases the authoring burden of creating complex stories in rich environments by allowing authors to significantly under-specify their desired narratives – allowing our system to fill in causal gaps with a series of planning tasks. In addition to this offline process, we provide a tool for real-time interactive story authoring in which the author manipulates the trajectory of a story as it progresses. In both cases, the author is presented with the world's virtual actors and a lexicon of events that those actors can perform with each other and the environment. The author composes the story by selecting sequences of events and their participants, and can immediately observe that story's execution in real-time by the virtual actors for instant feedback. This allows the author to make adjustments and see their effects with no computational downtime. Our system allows an author to explore a story space in real-time, observing events as they unfold for the characters, and making choices in the moment that affect the future trajectory of the story. With a human storyteller deciding the overall trajectory of the narrative, the computer is not responsible for directing the story entirely, but rather for facilitating the storyteller's vision of narrative-relevant events and interpreting the results of those event selections in a causally consistent manner.

This presents a daunting complexity problem, however. For example, in a well-authored story scenario, there may be hundreds of potential events that can occur in a world featuring any number of a virtual populace's several-dozen members. This is a prohibitively large action space to expose to a human user for real-time decision making, or even as an offline process in a rapid prototyping or previsualization environment. To solve this complexity problem, we introduce CANVAS, a system for Computer-Aided Narrative Animation Synthesis [29]. CANVAS understands the consequences of an event on the world and story, and can convert partially-specified stories by the author into complete and causally consistent narratives to execute on screen. An additional module, called Storycraft, can filter useful actions to produce a list containing the few most relevant, meaningful, and interesting events for the computer-aided human author to invoke in the world in real-time.

1.3 Contributions

The core contribution of our system is the event, a conceptual structure that forms an intermediary language between story and animation. Events represent atoms of narrative significance – interactions between characters and objects that drive the story forward in a meaningful way (recall that events are *intended outcomes*, not *observed occurrences* as the term is sometimes used to describe in other works). They are easy for a human author to select, describe, and arrange into a plot, and easy for our system to convert into high-fidelity animations on-screen once the story is decided. The rest of the system revolves around the creation, manipulation, understanding, and digestion of events. As such, CANVAS and Storycraft fill several roles in the authoring process, in terms of reducing the complexity and the burden on the author when designing stories using the event structure.

Understanding Ramifications. Both CANVAS and Storycraft can understand and communicate the effects of events to the human storyteller, such as the effect that a riot event will make its participating virtual characters angry, or that a conversation event between two characters will make them friends. This helps the human storyteller accomplish goals or produce interesting situations around which to revolve the narrative.

Populating Intermediate Events. CANVAS specifically can take *partially-authored* stories comprised of loosely related events and tie them together with reasonable causal associations by adding in the correct intermediary events. For example, if the human storyteller instructs a character to take her dog for a walk and then buy something at the store across town, CANVAS can fill in the intermediate events such as returning the dog to the house and the character using her car to drive to the store before shopping there. Furthermore, because CANVAS now has a horizon of future events that it has been told to enact, it can predictively interweave character actions such as the character picking up her car keys on the way to fetch her dog for the walk, removing that step later when she needs to enter her car. This kind of narrative understanding makes CANVAS a facilitator for the human storyteller to minimize authorial effort, focus on the creative aspects of storytelling, and create interesting scenarios for entertainment, education, or instruction without worrying about mechanical details.

Narrative Topology Analysis. Storycraft relies on the analysis and understanding of a complete Story Web generated as part of an offline process. Because Storycraft can examine a complete graph of the story space, it can extract topological domain-independent features from that graph and use these features to recommend choices to the user. This helps mitigate the effect of having an intimidating amount of choice and flexibility in the system, reducing a dauntingly large action space to a small but meaningfully diverse set of options.

1.4 Applications.

Our system in its current form is intended primarily for entertainment and education in virtual worlds. It is ideally situated for rapid prototyping and pre-visualization for film and other storyboard-based media, as an author can very quickly convert a narrative trajectory into a visual representation of actors in a scene. Additionally, because our simulation can accommodate interaction and generates simulations with real-time controllers at interactive frame rates, it could easily be used for more immersive applications. We envision using our tool both for open-world exploration experiences and proctored training scenarios where an educator or researcher (serving as the author) monitors the progress of their trainees or respondents and manipulates the parameters and story of the simulation in real-time to react to those participating users' actions.

Chapter 2

Related Work

This kind of interactive narrative system crosses many disciplines with respect to displaying sophisticated actors in a fully realized 3D virtual world, and managing them with a centralized system capable of driving an interactive narrative. There exists a wealth of research [58, 86, 26, 64] that separately addresses the problems of character animation, steering and path-finding, behavior authoring, and interactive narrative, with many open challenges [28] that remain to be addressed for the next generation of interactive narrative virtual worlds.

Character Animation. Data-driven approaches [35, 2] use motion-capture data to animate a virtual character. Motion clips can be manipulated and concatenated by using warping [93], blending [49, 50], layering [5], or planning [70, 18] to enforce parametric constraints on recorded actions. Interactive control of virtual characters can be achieved by searching through motion clip samples for desired motion as an unsupervised process [36], or by extracting descriptive parameters from motion data [21]. Procedural methods are used to solve specific tasks such as reaching, and can leverage empirical data [42], example motions [10], or hierarchical inverse kinematics [4] for more natural movement.

Physically-based approaches [8, 94] derive controllers to simulate character movement in a dynamic environment. We refer to Pettré et. al. [60] for a more extensive summary of work in these areas.

Steering and Path-finding. For navigation, the environment itself is often described and annotated as a reduction of the displayed geometry to be used in path planning. Probabilistic roadmaps superimpose a stochastic connectivity structure between nodes placed in the maneuverable space [33]. Navigation meshes [22] provide a triangulated surface upon which agents can freely maneuver. Potential Fields [90] generate a global field for the entire landscape where the potential gradient is contingent upon the presence of obstacles and distance to goal, but is prone to local minima. Dynamic potential fields [88] have been used to integrate global navigation with moving obstacles and people, efficiently solving the motion of large crowds without the need for explicit collision avoidance.

Steering techniques use reactive behaviors [62] or social force models [17, 57] to perform goal-directed collision avoidance in dynamic environments. Local perception fields [30, 31] and predictive approaches [55, 89] enable an agent to avoid others by anticipating their movements, while more complex scenarios such as group interactions and deadlocks are solved using hybrid techniques [77], space-time planning [78], or by externalizing steering logic [67]. Data-driven steering [38, 37] focuses on generating local-space samples from observations of real people which are used to create databases, or serve as training data to learn computational models which are queried to emulate real-human behavior. Recast [52] provides an open-source solution to generating navigation meshes from arbitrary world geometry by voxelizing the space, and the associated Detour library provides path planning and predictive steering on the produced mesh. Pelechano et. al. [58] provide a detailed review of additional work in this field.

Behavior Authoring. Animating virtual human characters has been addressed using multiple diverse approaches, particularly with respect to how behaviors themselves are designed. Early work focuses on imbuing characters with distinct, recognizable personalities using goals and priorities [43] along with scripted actions [59]. The problem of managing a character's behavior can be represented with decision networks [95], cognitive models [12], goal-oriented action planning [53, 25], or via learning [41]. Very simple agents can also be simulated on a massive scale using GPU processing [7]. Recent work [81, 71, 84] proposes an event-centric authoring paradigm to facilitate multi-actor interactions with contextual awareness based on agent type and event location. Our system makes use of Parameterized Behavior Trees (PBTs) [73] to coordinate interactions between multiple characters.

Our system is built on the PAStE platform (described in more detail in Section 5). Like PAStE's events, A Behavior Language (ABL) [47] 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. Since PAStE events are based on PBTs, they 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 can derive 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.

Multi-Solution Character Animation Platforms. End-to-end commercial solutions [45, 3] combine multiple diverse character control modules to accomplish simultaneous tasks on the same character, incorporting navigation, behavior, and/or robust character animation. SteerSuite [76] is an open-source platform for developing and evaluating steering algorithms. SmartBody [69] is an open-source system that combines steering, locomotion, gaze tracking, and reaching. These tasks are accomplished with 15 controllers working in unison to share control of parts of the body. SmartBody's controllers are hierarchically managed [23] where multiple animations, such as gestures, are displayed on a virtual character using a scheduler that divides actions into phases and blends those phases by interpolation. The controllers must either directly communicate and coordinate, or fix cases where their controlled regions of the body overlap and overwrite one another, making the addition of a new controller a process that affects several other software components. ADAPT, our animation platform (described in more detail in Section 4), shares some qualities with SmartBody, but also differs in several fundamental ways. While ADAPT does provide a number of character controllers for animating a virtual human, the platform focuses more on enabling high-level behavioral control of multiple interacting characters, the modularity of these character controllers, and the ease with which a user can introduce a new animation repertoire to the system without disturbing the other controllers already in place.

Crowd Simulation. In addition to telling a story, we are also interested in creating a realistic, immersive environment with interesting ambient activity. Traditional approaches [58] for large-scale crowd simulation incorporate social force models [17], reactive behaviors [62], or hybrid solutions [77] to handle the characters' navigation and decision processes. Typically, characters in a crowd are simulated with low behavioral fidelity, driven by simple goals [68] or heavy scripting [45] 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 [71]. Using this hybrid approach, we can tell stories in "living" worlds populated by functional, purposeful agents.

Visual Authoring. Visual languages have been successfully used in many applications [92] and are a natural metaphor for specifying narratives, particularly those taking place in a fully realized virtual environment. Game authoring systems [34, 66] facilitate the authoring of game logic by providing visual analogies to programming constructs, but typically operate at the level of an individual character's action space. There is a growing body of work in storyboard generation for interactive entertainment: Skorupski et al. [79] provide a storyboarding tool for novice users to author interactive comics. LongBoard [20] provides a hybrid sketch and scripting interface for authoring 2D storyboards which are used as visual constraints for rendering an animated 3D scene, while using a planner to fill in the unspecified shots. Hajarnis et al. [15] provide a graphical interface for authoring dialogue in generated cinematic scenes based on a case-based planning approach.

Automation. One of the earliest planning systems, the STRIPS planner [11], lays down the framework of a described world state with operators, preconditions, and effects manipulating that state. Subsequent developments to the pervasive STRIPS archetype include the planning domain definition language [48], cognitive-oriented planning [13, 61], hierarchical task networks [6], and planning with smart objects [1]. Planning has also been explored specifically in interactive narrative [74, 64]. Traditional planning approaches are limited by a lack of authorial control in the planning process and the need for detailed domain specification. They are typically restricted to simple problem domains (small state and action spaces) with a small number of agents and simplified representations [32, 20]. Our system can derive domain information from simple object affordances, while crowd-sourced [39] domain knowledge could also be used in narrative automation tasks.

To our knowledge no comparable system exists that allows an untrained author to sparsely specify a narrative in an accessible storyboard-like interface and use an automated planning framework to complete the narrative and display it in real-time in a fully-realized 3D virtual environment. Similarly, no system exists to our knowledge that (1) allows an author or explore and control a story in real-time as it happens with full authorial control over the environment and (2) uses a domain-independent topology-based recommendation system to nominate interesting story trajectories to the author while the story is executing.

Semi-Automated Forces (SAF). SAF systems are simulations in which one or more skilled operators control opposition forces in virtual military training exercises. Systems like ModSAF [51], OneSAF [83] exist in the sphere of military simulations, and similar systems have been applied to multi-player video games [19]. Applications of these systems are extremely complicated, and professional deployments require multiple skilled operators to manage the intricate detail of a military encounter with mission-critical realism, as illustrated in Figure 2.1. Additionally, SAF systems deployed both in video games and by the military perform little to no analysis of the scenario, relying instead on manual control from human authors and emergent behavior of autonomous virtual actors. In contrast, our system is designed to be far more accessible to untrained users, and is capable of assessing the narrative trajectories of the stories being created in order to provide recommendations to the human author and reduce the otherwise massive decision space.

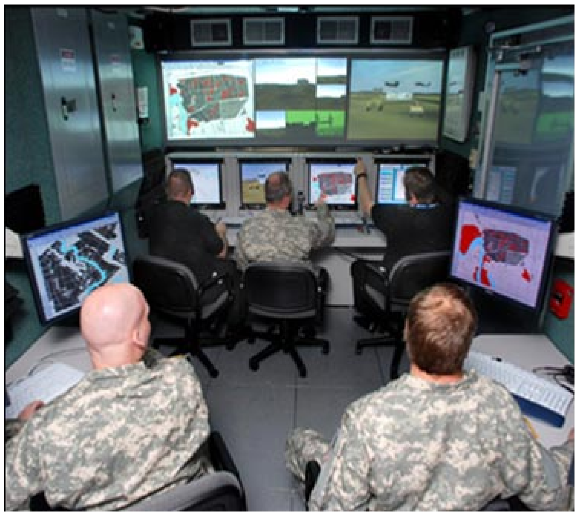

Figure 2.1: *An example deployment of a semi-automated forces (SAF) system.*

Interactive Narrative. The field of interactive narrative has produced the concept of a virtual director or drama manager [44]. Virtual director systems are driven by a virtual agent responsible for steering the agents in the world towards predetermined narrative goals [91]. This area has been well studied, and a number of techniques exist for designing effective directors. Facade [46] executes authored beats to manage the intensity of the story, Mimesis [63] employs narrative planning [40] with atomic agent actions, Thespian [75] uses decision-theoretic agents to create actors with social awareness, while PaSSAGE [87] and the Automated Story Director [65] monitor a user's experience through the story to choose between scenes and character behavior. Riedl and Bulitko provide a more detailed survey [64] 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 an automated system that performs an ordered selection of abstract, story-relevant action sequences to produce a narrative that responds to the actions of a user. Our approach is more author-centric than participant-centric. In our virtual world, a human author manages the ambient activity of a pool of "primordial" characters with little initial personality, and the system exposes to this human storyteller an accessible interface for involving those characters in events with narrative significance. The human storyteller is aided by a set of automated tools that reduce the complexity of the storytelling space and filters the information presented to the author. Specifically, our system differs from the traditional interactive narrative system in the following ways:

Control Granularity. Because our system is built on the PAStE platform, both the human storyteller and the automated system express their control over the virtual populace at the level of "events". Events represent compound interactions with narrative significance, rather than atomic actions such as opening a door or gazing at an object. Fine-grained details like these are necessary for proper character animation, but are also largely irrelevant to a human storyteller, and so the PBT-based event abstraction hides them.

Online Authoring. Typical interactive narrative systems involve human authoring of the story as an offline pre-process, where a human designer will specify a story prior to execution, and an automated virtual director is responsible for adhering to the story while also incorporating input from human participants. Our system allows an author to either (a) create stories in a distinct authoring phase and then immediately see the story in action in the virtual world with no compilation step, or (b) explore a story space in real-time to make top-level reactive decisions in the narrative as it happens. We focus on facilitating the human storyteller's decision-making process rather than completely automating the telling of the story.

Animation/Narrative Integration. While our system shares some similarities with the overall structure of other interactive narrative systems (like Automated Story Director [65] and Merchant of Venice [61]), these systems typically focus on the higher levels of narrative control – the arrangement and assembly of narrative atoms. Our work delves instead into the unsolved problems *between* the level of narrative atom, and the actual animations displayed on-screen during simulation. Events, as encoded in parameteized behavior trees, present contextually significant behavior control structures that are not only easy to reason about (as in a traditional interactive narrative), but also *directly translate* to complex coordinated multi-actor animated interactions with control down to the level of frame-by-frame joint orientations. Where other systems typically stop at the animation synthesis layer, our work examines the inherent and nontrivial issues of converting a generated or authored machine-readable story into a fully-realized real-time 3D simulation in rich virtual worlds using events as an intermediary language.

Chapter 3

Architecture

Figure 3.1 illustrates the manner in which our system will behave when simulating a narrative. Each story takes place in a fully realized 3D virtual world (1) containing virtual characters and objects embedded with narrative state information (2). Virtual human avatars controlled by human story participants can also negotiate and interact with the world and its characters (though this is not the focus of our work as presented) (3). Complex interactions between characters and other objects are accomplished using a pre-authored library of events (4). In order to decide on a course of action, the system draws candidate events from the event database (5), populates them with potential actor and object participants from the world, and analyzes the expected effects on the world should that event take place. These candidate events are passed to either CANVAS or Storycraft, each of which will filter them, add additional contextually useful information, and expose these events to a human storyteller (6). The storyteller decides on one or more events to execute, which are then instantiated into the event scheduler and begin dispatching commands to their participating actors and objects either in real-time (if using Storycraft), or after the story is completed and played (if using CANVAS) (7).

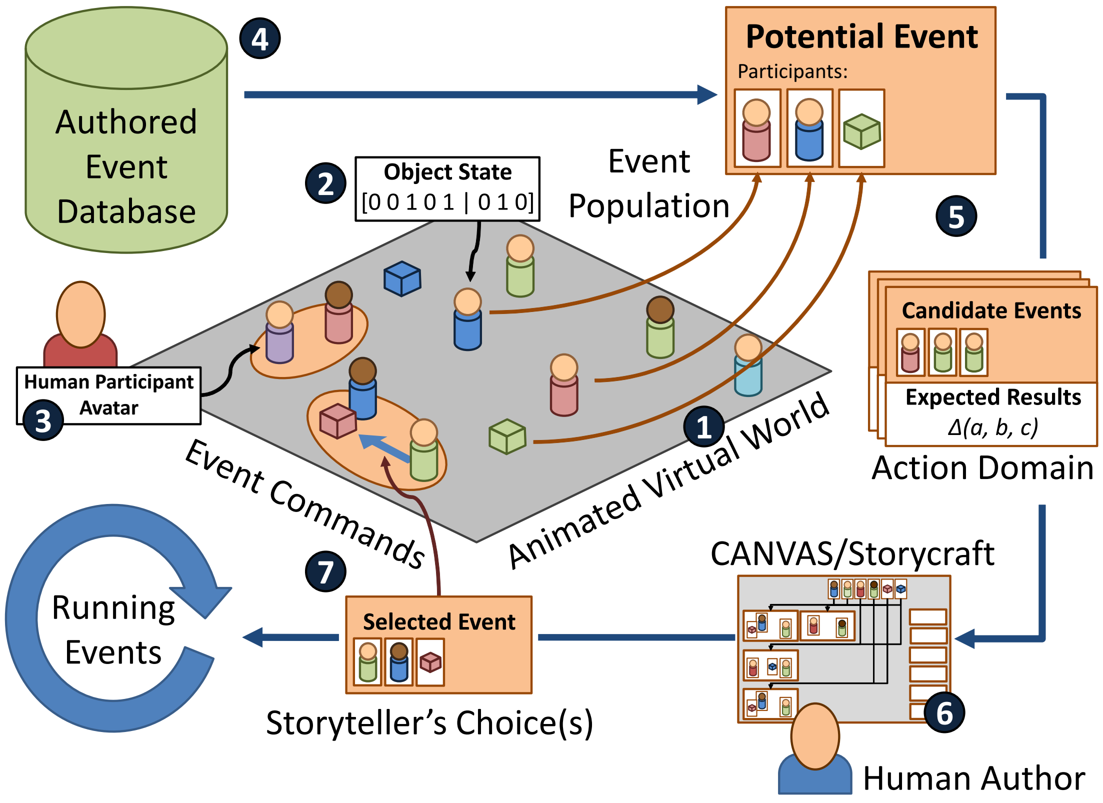

Figure 3.1: *An illustration of the functionality of our system at runtime.*

Because these stories take place in a fully-realized 3D virtual world, we have the added challenge (as opposed to text-based, 2D, or representational environments) of solving a problem that can be referred to, informally, as the "brains-to-bones" problem. That is, how can we take abstract narrative events and convert them into the frame-by-frame joint actuations on the rigged model skeletons of a populace of virtual characters. This is a significant conversion process that requires a complete solution stack of technologies and contributions in character animation, behavior authoring, and higher-level reasoning. We assume the availability of a graphics engine capable of sufficiently rendering rigged 3D models, but everything from character animation to interactive narrative generation remains a set of open problems to which our systems contribute.

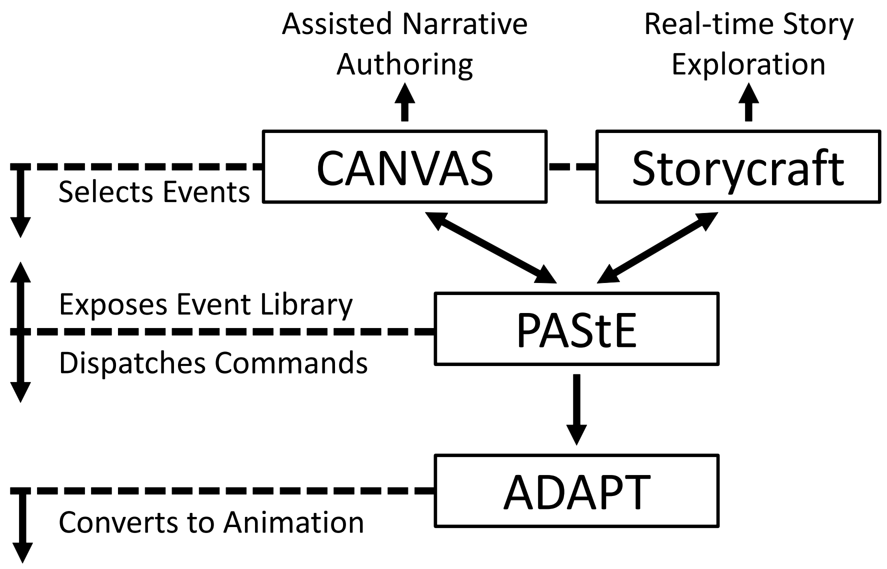

Figure 3.2: *An illustration of the solution stack, using four discrete tools organized into three operational tiers.*

The full system can be divided into three functional tiers with distinct responsibilities, as illustrated in Figure 3.2. These three tiers communicate with one another using the event, which is a dynamic encoding of multi-actor behavior used to represent narrative-relevant interactions. The lowest tier of the system, which sits right above the graphics engine, consists of the Agent Development and Prototyping Testbed (ADAPT) [27]. ADAPT is responsible for interpreting character behavior commands and events, using the information encoded in those behavior structures to pass messages to the various controllers acting on each character's skeleton, creating the final displayed animations for the story. ADAPT takes the skeleton of each virtual character's model rig and exposes simple commands like "ReachFor" and "LookAt" so that events can act upon the character's body. ADAPT is also the layer at which human story participants would use to interact with the system, using similar commands to control their virtual avatars and interact with the world. ADAPT receives events from the Platform for Adaptive Storytelling with Events (PAStE), which is responsible for managing a database of pre-authored events, assembling lists of what events can be enacted by which objects in the world, and analyzing the changes that will occur in the world state once those events are executed. PAStE exposes its library of events to both CANVAS and Storycraft. CANVAS and Storycraft occupy the human-facing top tier of the stack, and are the means through which the human author interacts with the system. Both tools filter the lists of candidate events from PAStE into a small set of the most narratively salient options, and both expose these to the human storyteller with additional information describing those events' ramifications. CANVAS exposes events and objects in such a way as to allow the human author to create long-form narratives in a distinct authoring phase, while Storycraft presents an interactive experience where the human user explores the branches of a story as it unfolds in real-time. In both cases the human storyteller is never concerned with narratively irrelevant details like making sure that characters open doors correctly, or how characters approach and orient towards one another before engaging in a conversation. Rather, the language of events focuses entirely on atoms of narrative significance and the communicative content of the story being told.

As an example, consider a story revolving around a bank robbery scenario. To tell this story, we need a virtual world presenting a 3D graphical representation of a bank, with rooms and various props. The bank is also inhabited by actors – fully articulated virtual humans serving roles such as those of the robbers, the guards, the bank staff, and customers. In our framework, the virtual human actors would respond to basic functional commands like "ReachFor" and "LookAt", but these more mechanical behavior commands do not represent narratively-significant actions. Series of actions on different characters can be combined into scripted sequences representing events. These events represent more contextual and meaningful interactions between characters and objects, such as a robber incapacitating a guard (involving, say, a "GoTo" command and an "Attack" animation command), or a bank staff member opening the vault. At its core, the means by which an author controls the world is by dictating (or partially specifying) the series of events that will take place in the world as well as indicating which characters should carry them out. From that point the system breaks down the event sequences into the more atomic functional commands that comprise them and uses those commands to animate the scene.

3.1 Events

The fundamental narrative atom of our system is called an *event*. An event is a centralized 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 using Parameterized Behavior Trees (PBTs). These are dynamic structures capable of dispatching commands to multiple characters from a central data structure, controlling them as limbs of the same entity. Because events have a single decision-making process, they are useful for coordinating carefully timed sequences of animations necessary for complex interactions between objects in the world. These can represent interactions between groups of characters like a conversation or riot, or interactions between characters and non-character objects using the smart object formalism. For very specific coordination of characters, this approach can be preferable over traditional behavior models where characters are authored in isolation and interactions between characters are designed in terms of stimuli, message-passing, and responses to triggers. When not involved in an event, characters continue to act autonomously with personal PBTs dictating their behavior, but this individual behavior is intentionally simple and intended only to occupy the character with ambient activity until the character is needed for its next event.

3.1.1 Characters Interacting with Each Other

Events are pre-authored by domain experts as part of the behavior design process. This is currently done using a simplified scripting syntax we have created, but the visual nature of PBTs also lends them well to authorship using a graphical user interface. Using PBTs, an event is designed to take one or more actors and props as parameters, and execute commands with direct control over those participating objects. The designer of an environment produces a number of PBT events that reside in an event library or lexicon during the simulation. At runtime, events are instantiated from the library and populated with objects from the world, at which point it is temporarily granted exclusive control over those participants. The autonomous behavior of those objects (if they have any) is suspended, and the event dispatches commands such as those instructing two actors to navigate towards and interact with one another. Once the event ends, control is returned to the characters' own individual decision processes, which re-evaluates the character's state after the event and decides on a new course of action. A conversation event can be authored as a simple sequential and/or stochastic sequence of commands directing agents to face one another and take turns playing gesture animations.

Figure 3.3 illustrates a sample PBT event conducting two characters through a conversation using our action repertoire. The characters, `a1` and `a2`, are passed as parameters to the tree, along with the meeting position (an invisible prop). The tree directs the two characters to approach one another at the specified point, face each other, and alternatively play randomly selected gesture animations. The gesturing phase lasts for an arbitrary duration determined by the configuration of the loop node in the tree. After the loop node terminates, the event ends, reporting success, and the two characters return to their autonomous behaviors. Note that this tree can be reused at any time for any two characters and any two locations in the environment in which to stand. This framework can be exploited

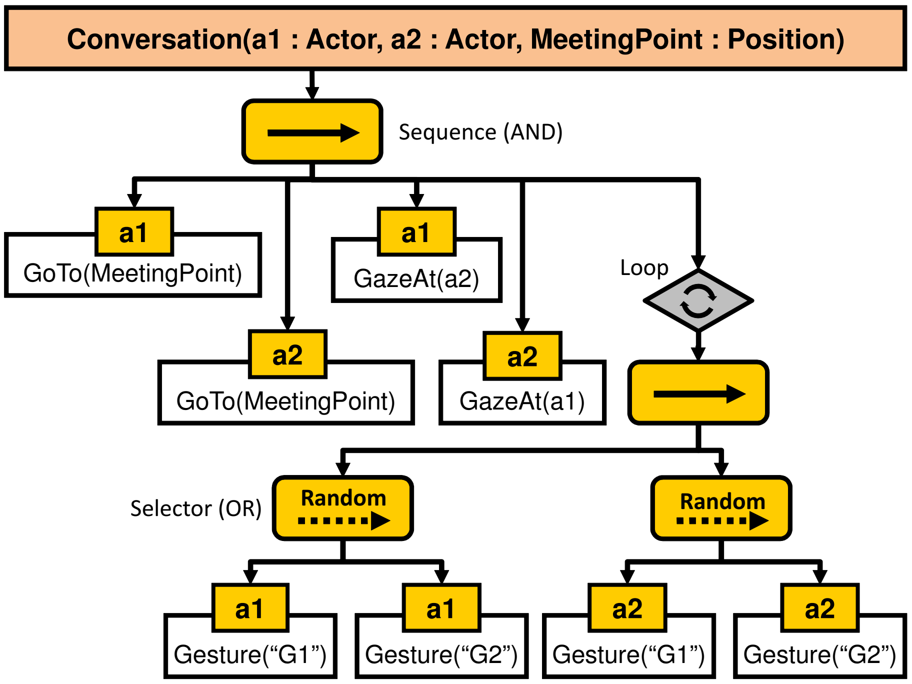

Figure 3.3: *A simple conversation PBT controlling two characters,* `a1` *and* `a1`*, with a* `MeetingPoint` *parameter.*

to create highly sophisticated interactions involving crowds of agents, and its graphical, hierarchical nature makes subtrees easier to describe and encapsulate for regular reuse.

3.1.2 Scheduling

Events are managed by a fully-featured scheduler responsible for updating both the personal behavior trees belonging to each character and higher-level event behavior trees encompassing multiple objects. Four basic principles in behavior design enable the scheduler to work effectively for orchestrating the behavior in a virtual world environment.

PBT Clock. PBTs operate on periodic clock ticks, where each tree refreshes itself, evaluates its current state, sends messages through the character's body controllers, and transitions to its next node if necessary. The scheduler keeps track of all of the active trees in the environment, both personal trees for individual characters, and event trees for multi-character interactions, and ticks them 30 times per second.

Character Suspension. Each autonomous character owns a “behavior process” object in its behavior controller, which maintains that character’s status with regards to autonomy. A character will not receive ticks to its personal tree if it has been suspended and placed under the control of a multi-character event tree.

Behavior Termination. PBTs are designed to respond to a termination signal. Termination signals can come at any time, and instruct a tree to interrupt its current action and end. The result of a termination signal can last multiple PBT clock ticks, so that poses such as reaching can be smoothly faded out before the tree reports that it has completed termination.

Event Priority. Multi-character event trees are assigned a priority value, where all character personal trees have minimal priority. When a character receives an event with a priority higher than its current tree, the scheduler terminates the current tree and suspends the character until all involved characters are ready. The event then begins ticking and dispatching commands to the new participant group. Once the event terminates, the characters return from being suspended. They may either continue their previous event that was interrupted, or return to their baseline autonomous behavior.

These four concepts allow very direct control over groups of characters in the environment, with smooth transitions between drastically different tasks. Since events can be cleanly terminated at any point in their execution, groups of characters involved in wandering or conversing with one another in an environment could very quickly activate a new, higher priority event to respond quickly to an event such as a loud noise or a fire. Each event tree operates as its own cooperatively multithreaded process with no direct communication between them.

3.1.3 Characters Interacting with the Environment

Using the same four principles for the behavior scheduler allows us to implement smart objects [24] into our virtual world for allowing characters to interact with the environment. A smart object's affordances can be encoded sequentially in a manner similar to that of a behavior tree or state machine. Smart objects receive ticks from the scheduler clock, and will block until reporting either success or failure. For example, when a character wants to sit in a chair, the behavior tree invokes the chair's "Sit" affordance with the character as a parameter. From that point, the tree will divert any ticks it receives from the scheduler clock to the smart object, which temporarily takes control of the character and directs it to approach and sit properly on the chair. Once the smart object chair determines that the character has succeeded in sitting down, it will report success to the behavior tree responsible for the character's behavior, so that the tree can move on to other actions. Parallel nodes in the tree can be used to synchronize actions, allowing a character to gesture or gaze at a target while still approaching and sitting. Smart objects represent the primary means of interaction with the environment, and are useful for a wide array of other interaction tasks. Figure 3.4 illustrates a simple state machine example for a chair smart object that instructs a character to approach and sit on it in the proper orientation. All objects in our virtual world are designed using affordances and the smart object formalism, including the virtual actors.

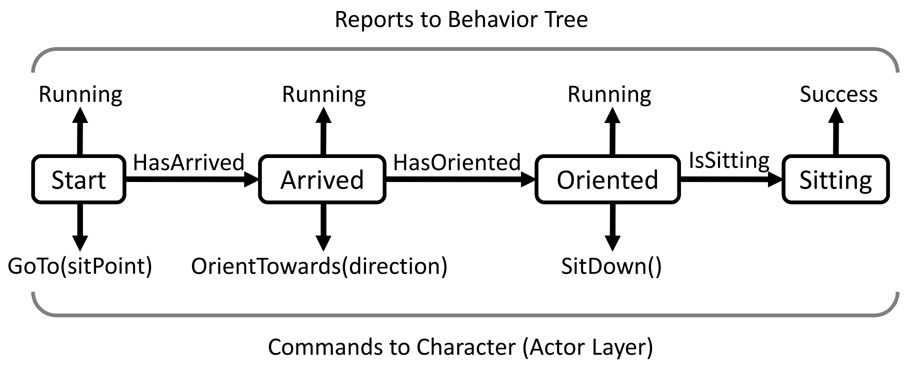

Figure 3.4: *The state machine for a chair smart object's "sit" affordance. The object transitions between states after evaluating predicates on the state of the character, periodically sending commands to the actor and reporting a status (running, success, or failed) to the behavior tree controlling the affordance.*

Chapter 4

ADAPT: The Agent Development and Prototyping Testbed

Executing events in our virtual world is dependent on fully-animated virtual human characters that can react in real-time to the world around them, and carry out complicated narrative-related interactions with one another in a 3D space. Animating these sophisticated virtual humans in real-time is a complex undertaking, requiring the solution to tightly coupled problems such as steering, path-finding, and full-body character animation (e.g., locomotion, gaze tracking, and reaching). This complexity is amplified as we increase the number and sophistication of characters in the environment. In order to provide this functionality, our system's lowest level consists of the ADAPT platform, a modular and extensible system that allows for the integration of multiple character animation controllers on the same model. ADAPT combines these animation controllers with an interface for path-finding and steering, as well as an integration of PBTs and events for single- and multi-character behaviors. ADAPT is highly extensible, allowing the addition of new character animations and capabilities for use in more sophisticated animations.

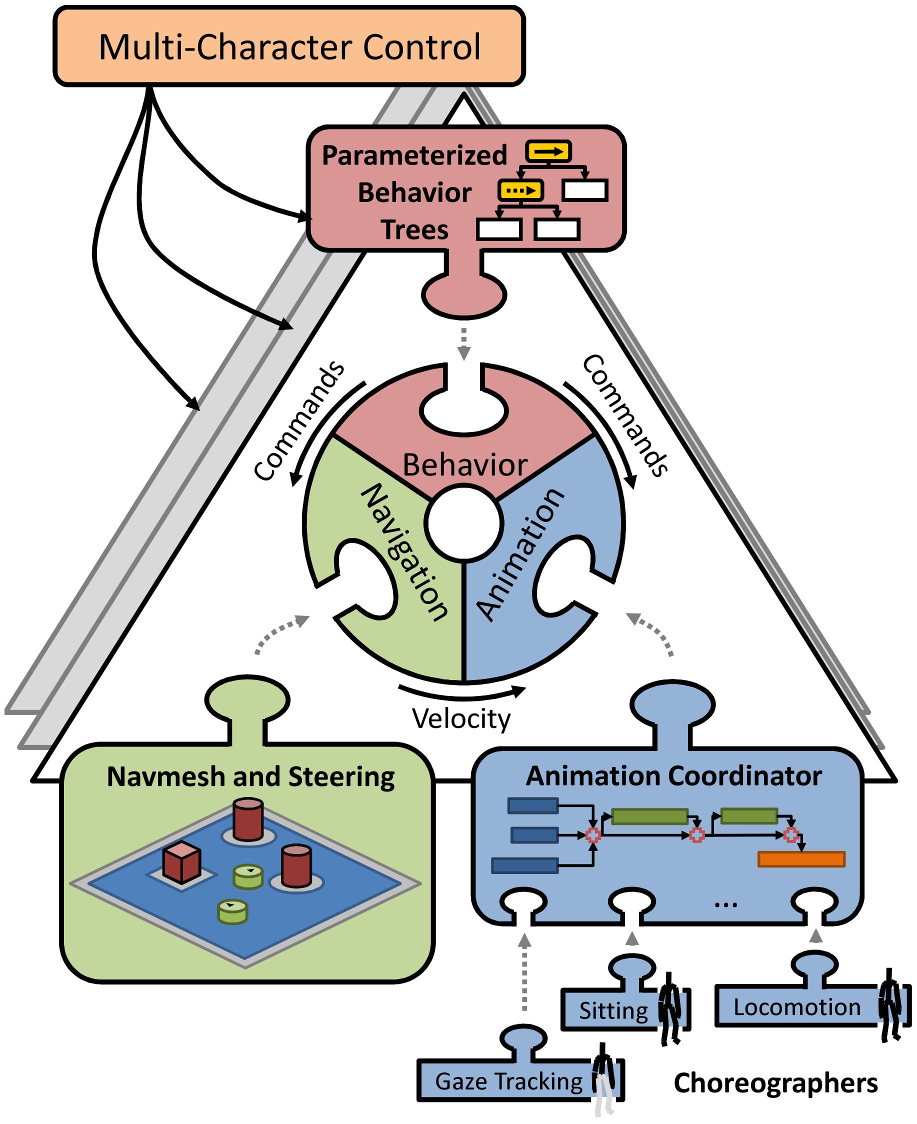

Figure 4.1: *Overview of ADAPT framework.*

4.1 Framework

The animation system performs control tasks such as locomotion, gaze tracking, and reaching as independent modules, called choreographers, that share parts of the same character's body and dictate joint movements on a frame-by-frame basis. These modules are managed by a coordinator, which acts as a central point of contact for manipulating the virtual character's pose in real-time. The navigation system performs path-finding with predictive steering and communicates directly with the rest of the framework. The behavior level is split into two tiers. Individual behaviors are attached to each character and manipulate that character using the behavior interface, while events are used to orchestrate the behavior of multiple interacting characters in real-time. The ultimate product of ADAPT is a pose for each character at an appropriate position in the environment, produced by the animation coordinator and applied to each rendered virtual character in the scene each frame. Figure 4.1 provides an illustration of the framework from an architectural integration perspective (the framework appears more hierarchical from a behavior control perspective, as illustrated in Figure 4.4).

We divide the problem of character animation into a series of isolated, modular components called *choreographers* attached to each character. Each choreographer operates on a *shadow*, which is an invisible clone of the character skeleton, and has unmitigated control to manipulate the skeletal joints of its shadow. Each frame, a choreographer produces an output pose consisting of a snapshot of the position and orientation of each of the joints in its private shadow. A *coordinator* receives the shadow poses from each choreographer and performs a weighted blend to produce a final pose that is applied to the display model for that frame. Since each choreographer has its own model to manipulate without interruption, choreographers do not need to communicate with one another in order to share control of the body or prevent overwriting one another. This allows a single structure, the coordinator, to manage the indirect interactions between choreographers using a simple, straightforward, and highly authorable process centered around blending the shadows produced by each choreographer. This system is discussed in more detail in Section 4.2.

We use a navigation mesh approach for steering and path-finding with dynamic obstacle avoidance. Each display model is controlled by a point-mass system, which sets the root positions (usually the hips) of the display model and each shadow every frame. Character choreographers do not directly communicate with the navigation layer. Instead, choreographers are made aware of the position and velocity of the character's root, and will react to that movement on a frame-by-frame basis. A character's orientation can follow several different rules, such as facing forward while walking, or facing in an arbitrary direction, and we handle this functionality outside of the navigation system itself.

To facilitate both rich character repertoires, and interesting narrative-relevant multi-actor events, ADAPT has a hierarchical behavior control system for its virtual characters. Each character has capabilities like `ReachFor`, `GoTo`, and `GazeAt` that take straightforward parameters like positions in space and send messages to that character's navigation and animation components. These are the capabilities that are invoked by PBTs representing a character's behavior, both in a character's automous decision-making process, and within the structure of an event. Having a single, flat interface for a character's action repertoire reduces the burden of behavior authoring with PBTs and the event paradigm, with well-described and defined tasks that a character can perform. The behavior system is discussed in more detail in Section 4.7.

4.2 Shadows in Full-Body Character Animation

General character controllers animate a virtual character using pre-recorded motions, or procedurally with physical models or inverse kinematics. We address the problem of coordination between these controllers by allocating each character controller its own private character model, a replica of the skeleton or a subset of the skeleton of the character being controlled. Our modular controllers, called choreographers, act exactly the same way as traditional character controllers, but do so on private copies of the actual rendered character model. These skeleton clones (shadows), match the skeletal hierarchy, bone lengths, and initial orientations of the final rendered character (display model), but have no visual component in the scene. Figure 4.2 illustrates a two-step blend process. First we combine the pose of the locomotion choreographer (green, full-body) during a walk cycle with the gesture choreographer (blue, upper-body) playing a waving animation, and then we apply the arm of the reaching choreographer (red, upper-body) full blend weight, safely overwriting the previous step for the joints of the left arm. The partial blend is represented with a mix of colors in the RGB space. Blend ordering is discussed in greater detail in Section 4.4.

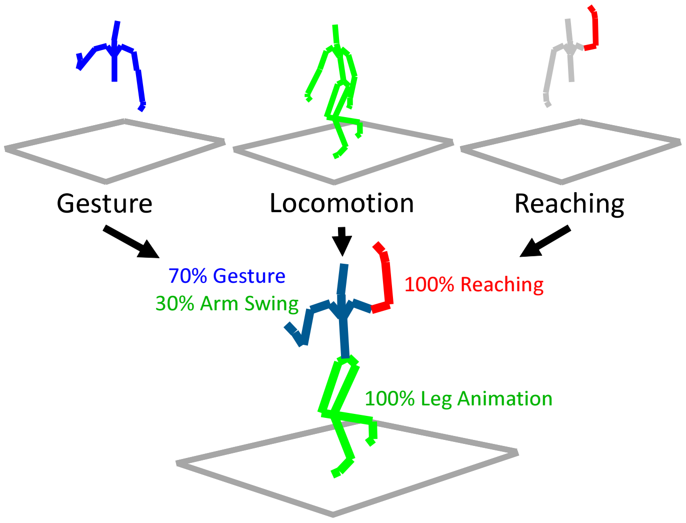

Figure 4.2: *Blending multiple character shadows to produce a final output skeleton pose.*

Character animation has two interleaving steps. First, each choreographer manipulates its personal shadow and outputs a snapshot (called a shadow pose) describing the position and orientation of that shadow's joints at that time step. Then, we use a centralized controller to blend the shadow pose snapshots into a final pose for the rendered character. For clarity, note that "shadow" refers to the invisible skeleton allocated to each choreographer to manipulate, while a "shadow pose" is a serialized snapshot containing the joint positions and orientations for a shadow at a particular point in time.

4.3 Choreographers

The shadow pose of a character at time t is given by $\mathbf{P}_t \in \mathbb{R}^{4\times|J|}$ where $\mathbf{P}_t^j$ is the configuration of the j^{th} joint at time t (expressed as a quaternion). A choreographer is a function $C(\mathbf{P}_t) \longrightarrow \mathbf{P}_{t+1}$ which produces the next pose by changing the configuration of the shadow joints for that time step. Using these definitions, we define two classes of choreographers:

Generators. Generating choreographers produce their own shadow pose each frame, requiring no external pose data to do so. Each frame, the input shadow pose $\mathbf{P}_t$ for a generator C is the pose $\mathbf{P}_{t-1}$ generated by that same choreographer in the previous frame. For example, a sitting choreographer requires no external input or data from other choreographers in order to play the animations for a character sitting and standing, and so its shadow's pose is left untouched between frames. This is the default configuration for a choreographer.

Transformers. Transforming choreographers expect an input shadow pose, to which they apply an offset. Each frame, the input shadow pose $\mathbf{P}_t$ to a transformer C is an external shadow pose $\mathbf{P}'_{t+1}$ from another choreographer C', computed for that frame. The coordinator sets its shadow's pose to $\mathbf{P}'_{t+1}$ and applies an offset to the given pose during its execution, to produce a new pose $\mathbf{P}_{t+1}$. For example, before executing, the reach choreographer's shadow is set to the output pose of a previously-updated choreographer's shadow (say, the locomotion choreographer with swinging arms and torso movement). The reach choreographer then solves the reach position from the base of the arm based on the torso position it was given, and overwrites its shadow's arm and wrist joints to produce a new pose. This allows the reach choreographer to accommodate multiple torso configurations without the choreographers directly communicating or even being fully aware of one another. A transforming choreographer can receive an input pose, or blend of input poses, from any other choreographer(s).

4.4 The Coordinator

During runtime, our system produces a pose for the display model each frame, given the character choreographers available. This is a task overseen by the coordinator. The coordinator is responsible for maintaining each choreographer, organizing the sequence in which each choreographer performs its computation each frame, and reconciling the shadow poses that each choreographer produces. The coordinator's final product each frame is a sequence of weighted blends of each active choreographer's shadow pose. We compute this product using the *pose dataflow graph*, which dictates the order of updates and the flow of shadow poses between choreographers. Generators pass data to transformers, which can then pass their data to other transformers, until a final shadow pose is produced, blended with others, and applied to the display model.

Blending is accomplished at certain points in the pose dataflow graph denoted by *blend nodes*, which take two or more input shadows and produce a weighted blend of their transforms. If the weights sum to a value greater than 1, they are automatically normalized.

$$B(\{(\mathbf{P}_i, w_i) : i = 1..n)\}) \longrightarrow \mathbf{P}' \tag{4.1}$$

Designing a dataflow graph is a straightforward process of dictating which nodes pass their output to which other nodes in the pipeline, and the graph can be modified with minimal effort. The dataflow graph for a character is specified by an engineer during the actor development process. The weights involved in blending are bound to edges in the graph and then controlled at runtime by commands from the behavior system. The order of the pose dataflow graph roughly dictates the priority of choreographers over one another. Choreographers closer to the final output node in the graph have the authority to overwrite poses produced earlier in the graph, unless bypassed by the blending system. We generally design the graph so that choreographers controlling more parts of the body precede those controlling fewer.

Blended poses are calculated on a per-joint basis using each joint's orientation quaternion. The blend function produces a new shadow pose that can be passed to other transformers, or applied to the display model's skeleton. Taking a linear weighted average of vectors is a solved problem, but such is not the case with the problem of quickly averaging $n > 2$ weighted quaternions. We discuss the techniques with which we experimented, and the final calculation method we decided to use in our related work on ADAPT [27]. In addition, Feng et. al. [9] provide a detailed review of more sophisticated motion blending techniques than our linear approach.

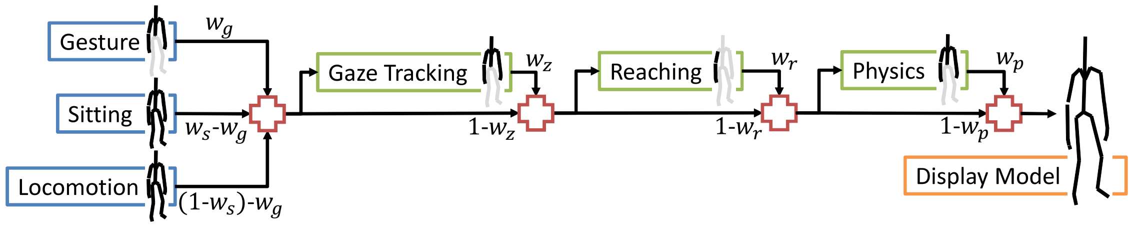

Figure 4.3: *A sample dataflow graph we designed for evaluating ADAPT. Generating choreographers appear in blue, transmuting choreographers appear in green, and blend nodes appear as red crosses. The final display model node is highlighted in orange. The sitting weight w_s, gesture weight w_g, gaze weight w_z, reach weight w_r, and physical reaction weight w_p are all values between some very small positive ϵ and $1 - \epsilon$.*

Figure 4.3 illustrates a sample dataflow graph. Three generating choreographers (blue) begin the pipeline. The gesture choreographer affects only the upper body, with no skeleton information for the lower body. Increasing the value of the gesture weight w_g places this choreographer in control of the torso, head, and arms. The sitting and locomotion choreographers can affect the entire body, and the user controls them by raising and lowering the sitting weight w_s. If w_g is set to $1 - \epsilon$, the upper body will be overridden by the gesture choreographer, but since the gesture choreographer's shadow has no legs, the lower body will still be controlled by either the sitting or locomotion choreographer as determined by the value of w_s. The first red blend node combines the three produced poses and sends the weighted average pose to the gaze tracker. The gaze tracking choreographer receives an input shadow pose, and applies an offset to the upper body to achieve a desired gaze target and produce a new shadow pose. The second blend node can bypass the gaze tracker if the gaze weight w_z is set to a low value (ϵ). The reach and physical reaction choreographers receive input and can be bypassed in a similar way. The final result is sent and applied to joints of the display model, and rendered on screen.

4.5 Using Choreographers and the Coordinator

The dataflow graph, once designed, does not need to be changed during runtime or to accommodate additional characters. Instead, the coordinator provides a simple interface comprising messages and exposed blend weights for character animation. Messages are commands (e.g., `SitDown`) relayed by the coordinator to its choreographers, making the coordinator a single point of contact for character control, as illustrated in Figure 4.1. In addition to messages, the weights used for blending the choreographers at each blend node in the dataflow graph are exposed, allowing external systems to dictate which choreographer is active and in control of the body (or a segment of the body) at a given point of time.

As an example, when the coordinator receives a gesture command, it raises w_g (in Figure 4.3), which takes control of the arms and torso away from both the locomotion and sitting choreographers and stops the walking animation's arm swing. Given sole control, the gesture choreographer plays an animation on the upper body, and then is faded back out to allow the walking arm-swing to resume. Since the gesture choreographer's shadow skeleton has no leg bones, it never overrides the sitting or locomotion choreographer, so the lower body will still be sitting or walking while the upper body gesture plays. All weight changes are smoothed over several frames to prevent jitter and transition artifacts. The division of roles between the coordinator and choreographers centralizes character control to a single externally-facing character interface, while leaving the details of character animation distributed across modular components are isolated from one another and can be easily updated or replaced.

4.6 Example Choreographers

ADAPT provides a number of diverse choreographers for animating a fully articulated, expressive virtual character. Some of these choreographers were developed specifically for ADAPT, while others were off-the-shelf solutions used to highlight the ease of integration with the shadow framework. ADAPT is designed to “trick” a well-behaved character control system into operating on a dedicated shadow model rather than the display model of the character, and so the process of converting an off-the-shelf character control library into a character choreographer is straightforward. Since shadows replicate the structure and functionality of a regular character model, no additional considerations are required once the choreographer has been allocated a shadow. Note that the choreographers presented here are largely baseline examples. The focus of ADAPT is to allow a user to add additional choreographers, experiment with new techniques, and easily exchange generic choreographers with more specialized alternatives.

Locomotion. ADAPT uses a semi-procedural motion-blending locomotion system for walking and running released as a C# library with the Unity3D engine [21]. The system takes in animation data, analyzes those animations, and procedurally blends them according to the velocity and orientation of the virtual character. We produced satisfactory results on our test model using five motion capture animation clips. The user annotates the character model to indicate the character’s legs and feet, which allows the locomotion library to use inverse kinematics for foot placement on uneven surfaces. We extended this library to work with the ADAPT shadow system, with some small improvements.

Gaze Tracking. We use a simple IK-based system for attention control. The user defines a subset of the upper body joint hierarchy which is controlled by the gaze tracker, and can additionally specify joint rotation constraints and delayed reaction speeds for more realistic results. These parameters can be defined as functions of the characters velocity or pose, to produce more varied results. For instance, a running character may not be permitted to rotate its torso as far as a character standing still.

Upper Body Gesture Animations. We dedicate a shadow with just the upper body skeleton to playing animations such as hand gestures. We can play motion clips on various parts of the body to blend animations with other procedural components.

Sitting and Standing. The sitting choreographer maintains a simple state machine for whether the character is sitting and standing, and plays the appropriate transition animations when it receives a command to change state. This choreographer acts as an alternative to the locomotion choreographer when operating on the lower body, but can be smoothly overridden by choreographers acting on the upper body, such as the gaze tracker.

Reaching. We implemented a simple reaching control system based on Cyclic Coordinate Descent (CCD). We extended the algorithm to dampen the maximum angular velocity per frame, include rotational constraints on the joints, and apply relaxation forces in the iteration step. During each iteration of CCD (100 per frame), we clamp the rotation angles to lie within the maximum extension range, and gently push the joints back towards a desired “comfortable” angle for the character’s physiology. These limitations and relaxation forces are based on an empirical model for reach control based on human muscle strength [80]. This produces more realistic reach poses than naïve CCD, and requires no input data animations. The character can reach for an arbitrary point in space, or will try to do so if the point is out of range.

Physical Reaction. By allocating an upper-body choreographer with a simple ragdoll, we can display physical reactions to external forces. Once an impact is detected, we apply the character's last pose to the shadow skeleton, and then release the ragdoll and allow it to buckle in response to the applied force. By quickly fading in and out of the reeling ragdoll, we can display a physically plausible response and create the illusion of recovery without requiring any springs or actuators on the ragdoll's joints.

4.7 Character Behavior

One of the most fundamental problems we needed to address in our system is converting simple commands like "reach for that object", or cooperative directives like "engage in a conversation" into a series of complicated joint actuations on one or more articulated bodies. ADAPT accomplishes this task with a hierarchy of abstractions known as the *ADAPT Character Stack*, illustrated in Figure 4.4. The stack is split into four main tiers: Behavior, Actor, Body, and Animation. The higher two levels ("Behavior" and "Actor") of the stack are designed for use by comparitively untrained authors, while lower levels levels offer more fine-grain control fidelity at the expense of simplicity, and can be accessed by expert authors to ensure very specific constraints on the character's movements.

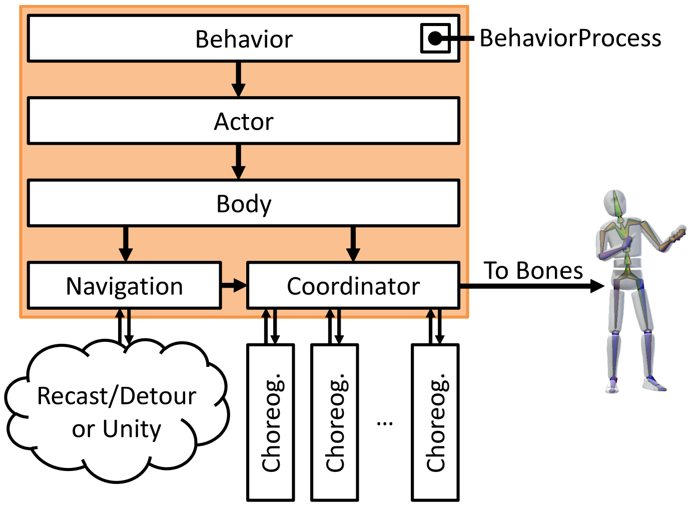

Figure 4.4: *The ADAPT Character Stack.*

Commands from each layer of the stack are filtered, converted, and distributed to subcomponents, starting as behavior invocations, translating to messages sent to the navigation or animation system, and finally converting into joint angles and blend weights used for posing the character on a frame-by-frame basis. Each layer in the character stack provides a different entry point for technical control over the character. The "Behavioral" layers offer interfaces for controlling the character at a high level, suitable for invocation by behavior trees and smart objects.

Animation (Navigation and Coordinator). This layer provides the lowest-level external access to the character's animation. A component accessing this part of the character stack is concerned with sending messages directly to choreographers (such as to change the reaching target position), or modifying blend weights to adjust the influence of a choreographer.

Body. This layer converts abstract commands like `ReachFor` into a series of messages passed to the reach choreographer and coordinator to set the reach target and raise the blend weight for the reaching pose. The Body layer is created to encapsulate the pose dataflow graph for a particular character, and assigns more semantic meaning to the blend weights for each blend node in the graph. An ADAPT character's list of capabilities is discussed in more detail in Section 4.8.

Actor. This layer abstracts commands in the Body layer. However, unlike the Body layer, the commands in the Actor layer will keep track of the duration of a task, and report success or failure. A call to `ReachFor` in the Body layer will return instantly and begin the reaching process, whereas a call to `ReachFor` in the Actor layer will begin the reach process and then block until the reach has succeeded or failed. Commands in the Actor layer are also designed to respond to a termination signal for scheduling, as described in Section 3.1.2. This layer of abstraction is necessary for controlling a character's behavior with behavior trees.

Behavior. This layer contains more sophisticated, contextual commands comprising multiple sequential calls to the Actor layer, such as playing a series of gestures to convey approval in a conversation. The Behavior layer also contains the character's personal behavior tree, and a BehaviorProcess node responsible for scheduling multi-character interactions using the behavior scheduler described in Section 3.1.2. Unless involved in a multi-actor event, the Behavior layer is responsible for directing the character's personal goals, and external calls to the Behavior layer are usually concerned with suspending or re-activating a character's autonomy.

4.8 Body Capabilities

The navigation and shadow-based character animation system provides a number of capabilities, enumerated below. Passing an empty target position will end that task, stopping the gaze, reach, or navigation. The locomotion choreographer will automatically react to the character's velocity, and move the legs and arms to compensate if the character should be turning, walking, side-stepping, backpedaling, or running. Note that only sitting and navigating are mutually exclusive. All other commands can be performed simultaneously without visual artifacts.

Commands	**Description**
`ReachFor(target)`	Activates the reaching choreographer, and reaches towards a position.
`GazeAt(target)`	Activates the gaze choreographer, and gazes at a position.
`GoTo(target)`	Begins navigating the character to a position.
`Gesture(name)`	Activates the gesture choreographer for the duration of an animation.
`SitDown()`	Activates the sitting choreographer and sits the character down.
`StandUp()`	Stands the character up and then disables the sitting choreographer.

Adding a New Body Capability. Adding a new behavior capability with a motion component, such as climbing or throwing an object, requires a choreographer capable of producing that motion. Since choreographers operate on their own private copies of the character's skeleton, they can be designed in isolation and integrated into the system separately.

Once the choreographer is developed, the process of adding a new behavior capability to take advantage of the choreographer requires two steps. First, the choreographer must be authored into the pose dataflow graph, either as a generating or transforming node, with appropriate connections to blend nodes and other choreographers. Next, the behavior interface can be extended with new functions that either modify the blend weights relevant to the new choreographer, and/or pass messages to that choreographer by relaying them through the coordinator. The sophistication of character choreographers varies, but ADAPT is specifically designed for integrating new choreographers into the character behavior and animation pipeline.

4.9 Computational Performance

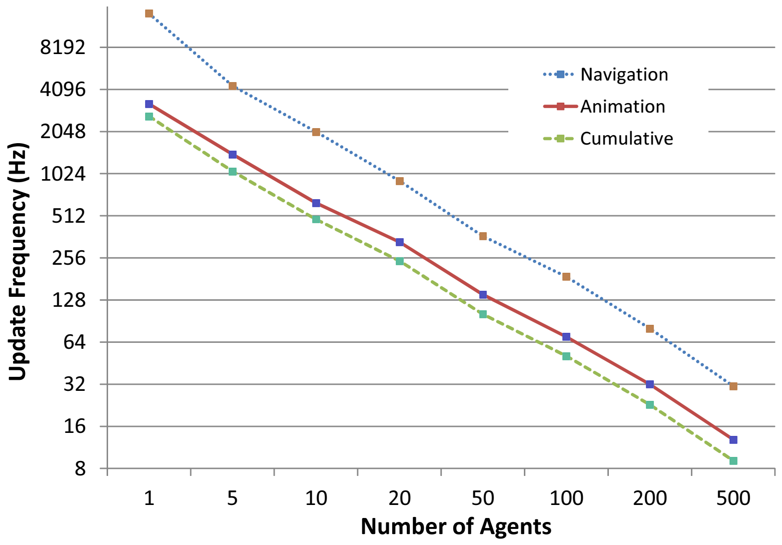

Figure 4.5: *Update frequency for the character animation and navigation components in ADAPT.*

ADAPT supports approximately 150 agents with full fidelity at interactive frame rates on commodity hardware. Figure 4.5 displays the update frequency for the animation and navigation system (for our scenes, the computational cost of behavior was negligible). This varies with the complexity of the choreographers active on each character. The ADAPT animation interface and the pose dataflow graph has little impact on performance, and the blend operation is linear in number of choreographers. Each joint in a shadow is serialized with 7 4-byte float values (a 3-vector and a quaternion), making each shadow 28 bytes per joint. For 26 bones, the shadow of a full-body character choreographer has a memory footprint of 728 bytes. For 200 characters, the maximum memory overhead due to shadows is less than 1 MB in the worst case. In practice, however, most choreographers use reduced skeletons with only a limb or just the upper body, making the actual footprint much lower for an average character.

Separating character animation into discrete modules and blending their produced poses as a post-processing effect also affords the system unique advantages with respect to dynamic level-of-detail (LOD) control. Since no choreographer is architecturally dependent on any other, controllers can be activated and deactivated arbitrarily. Deactivated controllers can be smoothly faded out of control at any time, and their nodes in the dataflow graph can be bypassed using the already-available blend weights. This drastically reduces the number of computed poses, and conserves processing resources needed for background characters that do not require a full repertoire of actions. The system retains the ability to re-activate those choreographers at any time if a specific complex action is suddenly required. Since choreographers are not tightly coupled, no choreographer needs to be made aware of the fact that any other choreographer has been disabled for LOD purposes.

Chapter 5

PAStE: A Platform for Adaptive Storytelling with Events

ADAPT is responsible for interpreting the commands sent to characters either from their own personal, autonomous PBTs, or from an event's PBT as it executes. Much of the rest of the system revolves around the selection and analysis of a library of created events for use on the virtual world's characters. PAStE sits directly above the ADAPT layer, and is responsible for three aspects of the system. First, PAStE manages the available library of authored events and provides the means for the rest of the system to access them. Second, PAStE handles the instantiation and execution of an event (adding it to the scheduler) once a higher-level directive (such as CANVAS or Storycraft) has selected an event to use. Finally, and most importantly, PAStE performs analysis on the events in the library, and provides information about what the pre- and post-conditions are for each event, which is necessary for a virtual director to understand the effects each event will have on the narrative. Fundamentally PAStE's role is to construct a more formal state and action space out of the virtual world and the events that can occur in it, allowing for higher-level reasoning about narrative trajectories.

5.1 Problem Definition

For a narrative involving events, we define our problem domain as $\Sigma = \langle \sigma, \alpha \rangle$, where σ is the state domain, and α is the action domain. We define a single problem instance as $\mathbf{P} = \langle \Sigma, S_{start}, S_{goal} \rangle$ where $S_{start}, S_{goal} \in \sigma$ are the start and goal states. Note that we are never committed to a single overarching problem instance, start state, or goal state, as our system revoles around producing numerous small interwoven narratives across a large populace. There may be many simultaneous problem instances at any given point during simulation.

5.2 State Domain

Each object (actors are objects with autonomy) in the world $\mathbf{W}$ is described as $w = \langle c, s, F \rangle \in \mathbf{W}$ where c is an ADAPT controller, s is the object's encoded state, and F is a set of smart object affordances. The object's state is defined as $s = \langle A, R \rangle$ where A comprises an object's individual attributes, and R is a sparse collection of pairwise relations with all other objects in $\mathbf{W}$. We define these two components of an object's state as follows.

Attributes. An attribute $a^i_{w_j} \in A_{w_j}$ is a binary flag that denotes the value of the i^{th} attribute for the object $w_j \in \mathbf{W}$. The vocabulary of attributes is predefined and global across all objects, so that this example each i^{th} attribute definition exists in a global attribute lexicon $\mathbf{A}$. These attributes consist of qualities affecting only only the owning object and can be immutable roles (such as `IsActor`, `IsChair`) or dynamic qualities (such as `IsSleeping`, `HasKey`, etc.). The symbol A_{w_j} denotes the compound value of all of object w_j's attributes, encoded as a vector of bits.

Relations. A relation $r^i_\cdot(\cdot)$ is a $|\mathbf{W}| \times |\mathbf{W}|$ matrix where $r^i_{w_n}(w_m) \in R_{w_n}$ maps to the true or false value of the i^{th} relation between objects w_n and w_m. Like attributes, the vocabulary of relations is predefined and global across all objects, so that in this example the relation key $i \in 1..|\mathbf{R}|$. Also like attributes, relations can be immutable (such as `Sibling(x, y)`) or dynamic (like `IsSittingOn(x, y)`). Note that relations are not necessarily symmetric in our system. The symbol $R_{w_n}(w_m)$ denotes the compound value of all of object w_n's relations relative to w_m, encoded as a vector of bits. Similarly, the symbol R_{w_n} denotes the compound value of all of object w_n's relations relative to all other objects in $\mathbf{W}$, encoded as a sparse triangular matrix of binary vectors[1].

Globally, the world state can be described as the compound state $S_\mathbf{W} = \{s_1, s_2, \ldots, s_{|\mathbf{W}|}\}$ of all objects $w \in \mathbf{W}$, but the processes responsible for selecting events will only rarely examine that complete world state. Instead, the world state is broken into subdomains containing the states of the objects in a particular event, so that for an event e, $S_e = \{s_1, \ldots, s_m\}$ where each s_i belongs to an object w_i participating in e.

5.2.1 Smart Object Affordances

A smart object affordance $f \in F_{w_o}$ for an object w_o is a function

$$f(w_o, w_u) : ((A_{w_o}, R_{w_o}(w_u)), (A_{w_u}, R_{w_u}(w_o)) \rightarrow ((A'_{w_o}, R_{w_o}(w_u)'), (A'_{w_u}, R_{w_u}(w_o)')$$

[1]These binary vector encodings allow us to store significant amounts of state data into a small number of 64-bit integers for each actor and object in the world, requiring a tiny memory footprint and allowing us to perform massive amounts of state evaluations with simple bitwise operations for maximum efficiency at scale.

that takes in the object w_o itself (the "owner" of the affordance) and another object w_u (the "user" of the affordance, not to be confused with a human user) and manipulates their controllers to modify their states and relations. The affordance can only modify relations under one of the following conditions: either the relation is of the form $r^i_{w_o}(w_u) \in R_{w_o}$, or it is of the form $r^i_{w_u}(w_o) \in R_{w_u}$ for some i. That is, during the affordance, an object can only add or remove a relation in the single other object involved in that affordance, and only relations that refer 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 (as described in Section 3.1.3). An affordance can also fail during its execution, such as if the user or used object do not match certain state criteria, in which case no change is made to any object's state.

Affordances represent the use or activation of an object. For instance, a chair has 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 `IsSittingOn` 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. This allows us to enforce an efficient and well-defined subdomain decomposition of an otherwise massive world space.

5.2.2 Affordance State Encoding

When evaluated during an affordance activation, the state s_{w_n} of an object w_n is encoded as a binary vector with two regions. The attributes of the object, A_{w_n}, is already stored in binary and fills the first region of the bit vector, while the second region of the vector is constructed dynamically based on the object's relations R_{w_n} relative to some other object w_m. We represent the total binary encoding for an object w_n, relative to w_m as $d_{w_n}(w_m) =$

$[A_{w_n}|\delta(R_{w_n}(w_m))]$. The binary vector region $\delta(R_{w_n}(w_m)) = [e_0 \ldots e_{|\mathbf{R}|}]$, where $e_i = r^i_{w_n}(w_m) \in \{0, 1\}$. That is, the bit entry e_i for each relationship $R_{w_n}(w_m)$ is set to 1 if and only if the object w_n contains a true value for the i^{th} relation in the relation vocabulary $\mathbf{R}$ when evaluated relative to w_m.

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:

$$d_a(b) = [1\ 0\ 0\ 1\ |\ 0\ 1], \ \ d_a(c) = [1\ 0\ 0\ 1\ |\ 0\ 0]$$

$$d_b(a) = [1\ 0\ 0\ 0\ |\ 0\ 1], \ \ d_b(c) = [1\ 0\ 0\ 0\ |\ 1\ 0]$$

$$d_c(a) = [0\ 1\ 0\ 0\ |\ 0\ 0], \ \ d_c(b) = [0\ 1\ 0\ 0\ |\ 0\ 0]$$

By encoding relationships in this way, we reduce the impact a single affordance can have on the world state, compartmentalizing the complete world affordance domain into manageable subdomains. This is important due to the fact that we may have multiple simultaneous problem instances at any given time. 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 limited and easier to reason about.

It is important to note that the high-level state of an object (i.e., A and R) 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 any details that are irrelevant to the narrative, 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 w_u and w_v both use some affordance f_{w_p} of object w_p, and $(d_{w_p}(w_u), d_{w_u}(w_p)) = (d_{w_p}(w_v), d_{w_v}(w_p))$, then $u \equiv v$ relative to affordance f_{w_p}. That is, as far as affordance f_{w_p} is concerned, u and v are interchangeable and should produce the same result upon activation with w_p, even if the details of their condition within the virtual world differ at a lower level (like being in different positions or poses).

5.2.3 Rules and Inference

Rules allow for logical inference on objects. A rule $\mathcal{R}(w_i, w_j)$ between two objects w_i, w_j is true or false, depending on the states and relationships of both objects. Rules are defined and solved using a declarative PROLOG-like interface[2] for logical programming where rule free variables are unified with the objects in the world that satisfy them. The following represents a small sample of the top-level rules used in our system according to their PROLOG representation:

```
% Can object U access object O?
RULE_can_access_object(U, O) :-
     % C: Object's current zone
     is_in_zone(O, C),
     RULE_can_access_zone(U, C).
```

[2]For efficiency, we do not use an actual PROLOG interpreter in real-time. Instead, we employ a customized tool capable of converting PROLOG statements into C# code during a precomputation step that can tie directly into our event evaluation framework.

```
% Can object U access zone Z?
RULE_can_access_zone(O, Z) :-
     % C: Object's current zone
     is_in_zone(O, C),
     RULE_path_exists_for_object(C, Z, O).

% A object can manipulate
% an object if it is unguarded ...
RULE_can_manipulate_object(U, O) :-
     is_guardable(O),
     is_unguarded(O).

% ...or if it is guarded
% by an allied character
RULE_can_manipulate_object(U, O) :-
     is_guarding(G, O),
     is_allied_with(G, U).
```

Rules are used for narrative-level reasoning and tasks such as evaluating whether a character can access a particular room, or manipulate another smart object based on the current world state.

5.3 Events and the Action Domain

In the problem domain, each event is defined as

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

where the t contains the event behavior (a PBT, as described in Section 3.1.1), c is the event's cost (either authored or derived based on its postconditions), 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 precondition function $\phi : \mathbf{W}^n \to \{0, 1\}$ is a conjunctive normal form (CNF) expression evaluated on the compound state S_e, composed of the states of the objects w_i attempting to participate in the event (these participating objects comprise the event's "candidacy set"). The CNF expression is composed of unary predicates evaluating individual objects' attributes, as well as relations and rules between the objects within that candidacy set. We also sometimes consider a reduction of the precondition function called the role evaluation function Γ, which more cheaply evaluates immutable aspects of an object's attributes (such as `IsActor`) to quickly identify the narrative role that object fills in the story.

Similarly, the postcondition function $\Delta : S_e \rightarrow S'_e$ evaluates the compound state of the participating objects in the candidacy set $\mathbf{w}_e \subset \mathbf{W}$, and produces a new compound state of attributes and relations to be applied to those objects (rules are never stored in memory, only inferred based on attributes and relations). If the precondition function is not satisfied, or the event fails during execution (either due to an error or by being interrupted), $S_e = S'_e$ – that is, no affect to the world state is recorded.

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". In practice, the author designs t and c, and can either manually author the pre- and postcondition functions, or automatically derive them (as explained in Section 5.4). In addition, the system is robust enough to infer parts of the participating object candidacy set in certain situations. For example, if a character is sitting on a chair, any subsequent event executions requiring the involvement of both that character, and the chair the character is sitting on (such as an event where the character wants to stand up), can automatically populate the chair as a participant based on the `IsSittingOn` relation between the two objects. This relieves author burden for enforcing consistency within the world.

5.3.1 Smart Object Groups

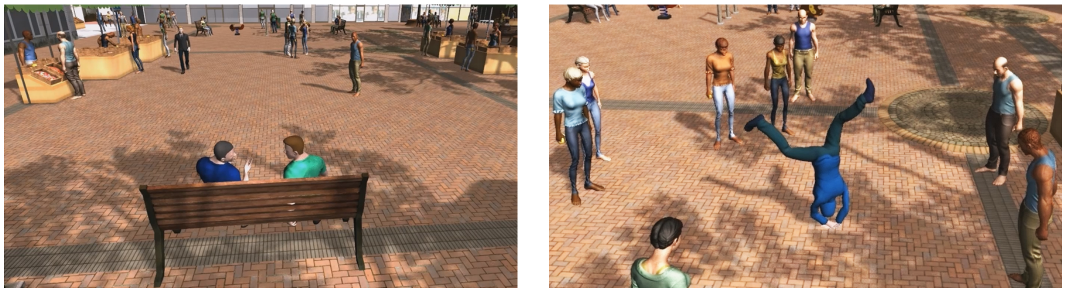

Figure 5.1: *Two sets of group activity. The left image displays groups of characters shopping in a street market and conversing with one another, while the right illustrates a coordinated group event of characters watching a break-dancer in the plaza.*

To more easily create ambient activity, individual smart objects can be grouped together to create groups that present a singular affordance interface and exhibit coordinated ambient activity. A group $w_g = \langle c_g, s, F, \mathbf{w} \rangle$ where c_g is a (non-ADAPT) group coordinator [72], s is the group object's state (treated as if it were a single object), F is the group's collective affordances, and $\mathbf{w}$ is the set of objects $\mathbf{w} \subset \mathbf{W}, w_g \notin \mathbf{w}$ contained in the group. The set $\mathbf{w}$ is malleable and objects can freely move in and out of groups based on contexts such as relationships or location in the virtual world.

The group coordinator operates on a group event lexicon $\mathbf{L}_g$ containing a library of group-oriented events that the coordinator can invoke and dispatch to its members to conduct their activity. The group events in $\mathbf{L}_g$ are non-transformative in that $\forall e \in \mathbf{L}_g, \Delta(S_e) = S'_e$. That is, the events in the group's event lexicon do not affect the states of the objects they execute on and are purely cosmetic. The group's ambient coordinator enforces manually authored distributions of events and dispatches events as needed to prevent objects in its group from appearing idle or inactive. These groups are effective for designating an ambient population to occupy and continually interact with a setting like a bank lobby, while also providing affordances for mass simultaneous activity like running from the sound of a gunshot. Where events are useful for singular cooperative activities between arbitrary objects, groups are designed for prolonged cosmetic interactions between actors and objects in the background of the scene. Since group membership is malleable, unneeded principal characters can temporarily join groups to stay visually busy and then be removed at any point when involved again in the story.

5.3.2 Event Instances, the Action Domain

An event instance $I_e = \langle e, \mathbf{w}_e \rangle$ is a tuple containing an event paired with a candidacy set $\mathbf{w}$. The instance is *valid* iff $\phi(\mathbf{s}_{\mathbf{w}_e}) = 1$. An event instance can be partially complete, in which case some of the members of $\mathbf{w}_e$ are undefined (i.e., $\exists w_i \in \mathbf{w}_e$ s.t. $w_i = \emptyset$). The final action domain for PAStE consists of all possible valid event instances given the set of objects in the world and the authored event lexicon. The act of generating a story within the PAStE framework consists of creating valid event instances and arranging them into a narrative order such that the postcondition function of one event leads to the validation of the precondition function of the next event(s) in the story sequence. The worst case growth for picking n objects for one event would be $\frac{|\mathbf{W}|!}{(|\mathbf{W}|-n)!}$, but we will introduce some segmentation, filtering, and sampling techniques to avoid this combinatorial growth.

5.3.3 Goals

PAStE alone is not responsible for satisfying story goals. Ultimately it is the human storyteller's responsibility to select events and participants with the help of CANVAS or Storycraft, 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.

5.4 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. In effect, this means that PBTs for events and autonomous behavior consist of three types of node: control nodes that affect the flow of tree ticks, assertion nodes that evaluate the state of the object(s) involved in the tree, and affordance nodes that activate affordances on participating objects. 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 can use these participants to express 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 derived automatically. This step does not learn anything "new" about the system or environment, but takes advantage of several assumptions about the nature of affordances to massively reduce authorial burden and allow 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. The laboratory is 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_{w_o} and d_{w_u}), it ceases that branch. Recall that we consider two objects to be functionally identical if their encoded states match in the context of a given affordance activation – this is the property that allows our search to terminate. The final result of the simulation is a set T of transition records $\{(w_o, f \in F_{w_o}, (d_{w_o}, d_{w_u}) \rightarrow (d'_{w_o}, d'_{w_u})\}$ over all affordances f belonging to all object archetypes w_o, over all possible state encodings d_{w_o} and d_{w_u} for affordance owner w_o and candidate user object w_u. Note that our binary state encoding is an optimization that is not required for the algorithm to function properly.

Algorithm 1: Exploring the affordance domain.

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

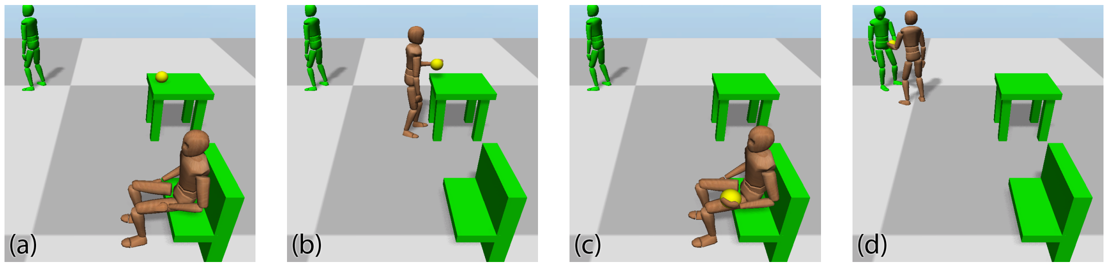

Figure 5.2: *Affordance domain exploration. The actor (a) sits, then (b) picks up an object and tries both (c) sitting with that object and (d) handing that object to another dummy actor.*

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 simulation engine, is displayed in Figure 5.2.

5.4.1 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 attribute 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 non-deterministic, an object could have multiple first affordances. For all of those affordances, we collect their valid input states (since we know under which conditions the affordance succeeded and failed in the SOL), 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.

5.4.2 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 CANVAS and Storycraft. 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 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 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 (such as `IsActor` or `IsChair`), 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 $r_1 \cdot r_2 \cdot \dots \cdot r_n$, where r_i is the number of objects matching the i^{th} role required by e. Thus, the number of candidates would be much smaller for a diverse environment.

5.5 Progressive Differentiation

One reason that a large number of candidates is undesirable is that all candidate objects are equal in value to the process responsible for selecting events. That process has no exposed 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. The event-selecting process searching through the action domain is then rewarded for using objects and characters with higher salience values. Salience has two parts: a way for events to leave 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* [72], 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 rewarding the use of salient characters, we can drive CANVAS and Storycraft to select "main characters" when it needs an actor to retrieve a key and use it. We expect a single character participating in three narrative events 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 character "priming" and restricting selection to a geometric radius [81]. We explored these options during development, but ultimately did not need to implement them.

Progressive differentiation is especially important to an event-driven narrative system. Individual characters contain very little decision-making capability by design – all complicated interactions are either stored in events, or in smart objects. Individual characters are not responsible for knowing how to perform tasks like holding a conversation, participating in a riot, or sitting in a chair. This makes each character essentially a "blank slate" onto which we can add information relevant to the narrative. Human story participants interacting with the system as character avatars initially know as little about the virtual actors as the system itself does. As a result, when a virtual actor participates in an event, two things happen. First, the human user avatar witnesses the event and ascribes narrative value to both the event and the personality of the participant (a character involved in a number of conversations can be perceived as friendly or talkative), and the event itself leaves residual information in the character (as a result of the conversation, the character is now "friends" with one or more other actors). Thus, the human participants understanding of the characters evolves with the system's own differentiation of those actors. Most actors are mechanically capable of performing most events from the very beginning, but it may not make narrative sense to do so. If two characters become friends, they may become eligible for more events such as playing a game of catch or buying one another a gift – actions that would otherwise be narratively inconsistent. The characters themselves don't become more sophisticated or capable, but information stored residually within them enables them to more plausibly play different roles in the stories that form.

Chapter 6

CANVAS: Computer-Assisted Narrative Animation Synthesis

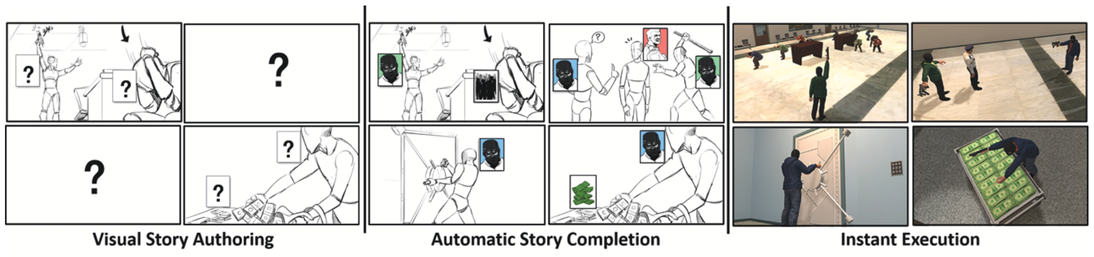

Figure 6.1: *CANVAS Overview. (a)* ***Visual Story Authoring****: Authors specify key plot points in the narrative using a graphical interface. (b)* ***Automatic Story Completion****: CANVAS automatically identifies and resolves incomplete stories by filling in missing participants and introducing new story elements to generate a consistent and cohesive narrative. (c)* ***Instant Execution****: The story is immediately displayed in real-time in the virtual world using the animated characters involved in the story.*

Sitting at the top of the ADAPT and PAStE platform stack is CANVAS, one of our two top-level interfaces (alongside Storycraft) that serves as the means for the human author to control the characters in the virtual world by selecting, populating, and dispatching events to execute. The event and action space described by PAStE is too large and complex for a human author to effectively constantly reason about in its entirety while creating long-form narratives. The principal role of the CANVAS, then, is to interpret the wealth of information provided by PAStE about the way events affect the narrative space of the world and use that information to fill gaps in the author's story specification. CANVAS is designed to receive a partially-defined trajectory of loosely affiliated choices (sequences of events that do not directly follow one another in causality), and through narrative interpolation insert the necessary connective events to produce a cohesive sequence of character actions. This improves the flexibility afforded to the author, and allows the author to create richer stories with less manual effort.

6.1 Defining a CANVAS Story

Like PAStE, CANVAS relies on its own set of abstractions to describe a story sequence as the author designs it. The fundamental building block for CANVAS is the event instance (as described in Section 5.3.2).

6.1.1 Event Authoring and Story Structures

Events themselves are authored as PBTs with parameter fields for their participating actors and objects. Because of their graphical nature, behavior trees are easy to author using a graphical user interface and accessible commercial products for doing so are widely available [14]. Because graphical authoring is an explored problem (while still taking considerable development resources to implement), for our purposes we author behavior trees using a structured code format in C#. An example PBT authored in C# appears as follows (with minor syntax simplifications):

```
public static Node Subtree_DistractAndIncapacitate(
    SmartCharacter distractor,
    SmartCharacter aggressor,
    SmartCharacter target)
{
    return Sequence(
        distractor.GoTo(target.position),
        ParallelSequence(
            Subtree_Distract(distractor, target),
            Sequence(
                Wait(17000),
                aggressor.GoTo(target.backWaypoint))),
        aggressor.UseAffordance(target, "Knockout"));
}

public static Node Subtree_Distract(
    SmartCharacter distractor,
    SmartCharacter target)
{
    return Sequence(
        distractor.Icon("speaking"),
        distractor.Gesture("callover", 3000),
        target.GoTo(distractor.transform.position),
        distractor.UseAffordance(target, "Talk"),
        distractor.Icon(null));
}
```

This code snippet presents most of the logic involved in the "Distract and Incapacitate" task, where two characters coordinate in such a way where one disractor calls the target over towards them, while the aggressor moves behind the target and incapacitates them. Some events have no animation component, and exist to silently change the state of the world (like BreakAlliance, which enables two previously allied characters to take aggressive actions against one another by changing their relationship flags). This system exposes the most power to advanced users, but could certainly be converted to a more accessible user interface for end-users with little conceptual difficulty. Once these events are authored, we organize them into the system with appropriate metadata and begin to assemble more complex narrative structures from them.

Story Beats. A Story Beat $\beta = \{I_1, \ldots, I_n\}$ is a collection of event instances for events occurring simultaneously at a particular juncture in the story. The preconditions of a beat $\beta = \{I_1 \ldots I_n\}$ are a conjunction of the CNF preconditions of all its event instances: $\phi_\beta = \phi_{e1} \wedge \phi_{e2}, \wedge \ldots \wedge \phi_{e_n}$. Multiple event instances within the same beat cannot share any object participant between them (as all of the events in that beat will begin in parallel). A beat β is valid if all instances $I \in \beta$ are valid. Recall that some of the instances of a beat may be partially defined, in that their candidate sets are missing members for a full successful execution of the event.

Story Arcs. A Story Arc $\alpha = (\beta_0, \beta_1, \ldots, \beta_m)$ is an ordered sequence of beats representing a story, where events can occur both sequentially and simultaneously throughout that story's execution. The beat β_0 is the initial configuration of the world as defined by the author, and operates on the initial authored world state $S_{\mathbf{W}}$.

6.1.2 The Story Sequence

Story arcs are authored using a structure called the Story Sequence Diagram (SSD) in the CANVAS graphical authoring interface. These diagrams chart the progression of the beats within the story arc, and illustrate which objects participate in each beat's event instances. An SSD is a directed acyclic graph $\mathbf{Q} = \langle \mathbf{V}, \mathbf{D} \rangle$.

The vertices of an SSD are divided into two classes $\mathbf{V} = \mathbf{V_W} \cup \mathbf{V}_I$. Object vertices $\mathbf{V_W}$ refer to objects from the world, i.e., $\mathbf{V_W} \approx \{w_1, \ldots, w_n\} \subseteq \mathbf{W}$, while event vertices refer to event instances, i.e., $\mathbf{V}_I \approx \{I_1, \ldots, I_m\} \subseteq \mathbf{I}$. An SSD's edges are divided into three classes $\mathbf{D} = \mathbf{D}_\pi \cup \mathbf{D}_\phi \cup \mathbf{D}_\tau$ that indicate three relationship types. Participation edges $\mathbf{D}_\pi \subseteq \mathbf{V_W} \times \mathbf{V}_I$ denote a "participates in" relationship between an object and an event instance as a participant. Precedent edges $\mathbf{D}_\phi \subseteq \mathbf{V}_I \times \mathbf{V}_I$ denote a "comes after" relationship between two event instances and are used to align events to different story beats. Termination edges $\mathbf{D}_\tau \subseteq \mathbf{V}_I \times \mathbf{V}_I$ denote a termination dependency, where an edge between (I_i, I_j) indicates that event instance I_j is terminated as soon as instance I_i finishes executing (typically used for repeating background activity). Note that I_i and I_j must be in the same story beat for this relationship to be created.

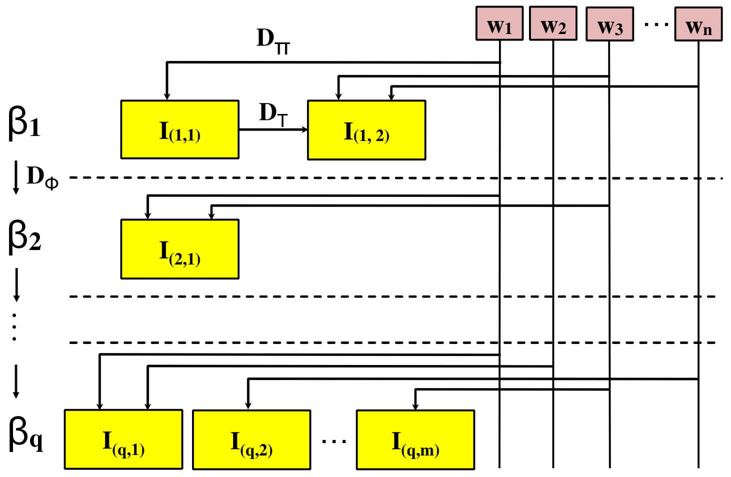

Figure 6.2: *An illustration of the various arcs and vertices that describe a Story Sequence Diagram.*

Precedent edges can be manually added by the author to define separate story beats. The CANVAS system also automatically detects when the a single object is used twice in the same beat, and will move one of the two involving event instants to a subsequent beat with a new precedent link between them. We display the SSD in such a way that each horizontal row of event instances delineates a beat, and the ordered sequence of beats represents the resulting story arc. Figure 6.2 illustrates a generic story arc represented as a story sequence diagram.

6.2 Handling Underspecified Stories

CANVAS is designed for authors to partially author stories where some events instances may have gaps in their participation sets, or may be omitted altogether in the causal sequence of narrative actions. The authoring environment is designed to accommodate these underspecified stores and perform inference to fill in the gaps in the author's desired narrative.

Automated Parameter Selection. Partially-specified story beats can contain event instances such as $I = \langle e, \mathbf{w} \rangle$ where $\mathbf{w} \subset \mathbf{W} \cup \{\emptyset\}$. That is, the participant objects of e may contain some entries that do not refer to an object in the world and are instead undefined. Note that objects in $\mathbf{w}$ are assigned to specific roles in order in the context of an event instance (i.e., e may be a "Sit on Chair" event, for which we use a PBT that takes an object to fill the role of the person sitting, and the chair, in that order), so we can consider $\mathbf{w}$ here both as an ordered vector and as a set. For every undefined participant object $\{w_j \in \mathbf{w} | w_j = \emptyset\}$, we consider the domain of possible values: $\mathbf{x}_j = \{x \mid x \in \mathbf{W}, \Gamma_e(x) = 1\}$ such that the candidate object x satisfies e's role evaluation function Γ_e for that particular role in the event. This produces a set of all possible combinations of objects $\mathcal{P}(I)$ that satisfy the role requirements of the event. Algorithm 2 details the process for computing $\mathcal{P}(I)$.

Once we generate $\mathcal{P}(I)$, we reduce the space complexity by filtering it according to three hard constraint criteria:

1. **Duplicate Elimination.** We filter duplicates in two ways. First, each candidate vector in $\mathcal{P}(I)$ must be unique relative to all other candidate vectors. Second, no object may appear more than once within the vector itself (i.e., an object can fill only one role in any given event instance).

Algorithm 2: Generating all possible parameter combinations for incomplete event instance I

```
    Data: $I = \langle e, \mathbf{w} \rangle$
    Result: $\mathcal{P}$, described as a set of vectors of length $|\mathbf{w}_e|$
1   $\mathcal{P} = \{()\}$ ($\mathcal{P}$ begins as a set of one empty vector)
2   foreach $j = 1..|\mathbf{w}|$ (for every role that needs to be filled in $e$) do
3       create set $\mathcal{P}' = \{\}$
4       foreach $v \in \mathcal{P}$ do
5           if $w_j = \emptyset$ (if the $j^{th}$ candidate is undefined) then
6               // append all role-satisfying objects to all the participant vectors
7               foreach $w' \in \mathbf{W}$ do
8                   if $\Gamma_e^j(w')$ (if $w$' satisfies the requirements for the $j^{th}$ role) then
9                       $\mathcal{P}' = \mathcal{P}' \cup \{(v\ w')\}$
10          else
11              // just add the object already selected by the author to all the vectors
12              $\mathcal{P}' = \mathcal{P}' \cup \{(v\ w_j)\}$
13      $\mathcal{P} = \mathcal{P}'$
```

2. **Precondition Satisfaction.** A candidate object vector $\mathbf{w}$ for an event e must satisfy the precondition function $\phi_e(\mathbf{s_w})$. This includes the evaluation of all attributes, relations, and rules within and between the objects in the candidate vector. The precondition function trivially includes an evaluation of the role evaluation function Γ_e, but is a more costly computation overall and so we wish to compute it as few times as possible. For events that appear in story beats after the first beat in a story arc, we predictively extrapolate the effects of all prior story beats to anticipate changes to the potential participants in the current candidate event instance.

3. **Unique Usage.** Objects are treated as unique resources in our story environment. No object can be used in more than one event at any given time during simulation. As such, when we evaluate story beats, we must ensure that no object is used simultaneously by two or more event instances in the same beat.

We can additionally weight event instance candidates by soft constraints to compute a score. We currently employ two such scoring strategies, which are mutually exclusive:

1. **Salience.** As described in Section 5.5, objects that have already participated in more events are given preferential treatment in selection for future events. This helps to establish “main characters” as recurring focal points in the story.

2. **Balanced Usage.** The inverse of salience, this criteria prefers a uniform distribution of all objects in the world, rather than focusing on recurring individuals.

Once the powerset $\mathcal{P}(I)$ is assembled, we compute the optimal object candidate vector $\mathbf{w}_I^o$ as

$$\mathbf{w}_I^o = \underset{\mathbf{w} \in \mathcal{P}(I)}{\text{argmax}}\ \textbf{score}(\mathbf{w})$$

such that $\neg\mathbf{dup}(\mathbf{w}) \wedge \phi_e(\mathbf{s}_\mathbf{w}) \wedge \mathbf{unique}(\mathbf{w}, \beta)$. This method can be extended to fill missing candidates in entire beats rather than individual event instances. To fill missing candidates across multiple instances in a story beat, we calculate $\mathcal{P}(I_i)$ for each $I_i \in \beta$ that does not contain duplicates, satisfies each respective event’s preconditions, and used each participating object only once. Of these, we pick a combination of candidate vectors for each event instance in β that maximizes **score**$(\mathbf{w})$ without violating the unique usage precondition between each object in each event instance in β.

6.3 Parameter Filling for Incomplete Story Arcs

For partially-defined story arcs where the author has omitted certain events in the expected causal trajectory of the story, there exist “narrative gaps” where the system must fill in entire events or beats and we cannot predict the state changes made to the world by the events in these gaps. In order to find valid participants for events to which we don’t immediately know every preceding event, we relax the precondition satisfaction requirement when filtering the power set $\mathcal{P}(I)$. Rather than enforcing that each candidate vector strictly satisfies $\phi_e(\mathbf{s})$ (here $\mathbf{s}$ represents the collective state of the objects $\mathbf{w}$), we introduce a second scoring term that evaluates the objects' likelihood of satisfying $\phi_e(\mathbf{s})$ (or the beat-wide precondition function $\phi_\beta(\{\mathbf{s}_{I_1}, \ldots, \mathbf{s}_{I_{|\beta|}}\})$. We estimate the world state before and after the execution of the current story beat β, $\mathbf{s}_{\beta-1}$ and $\mathbf{s}_\beta$. The hamming distance $dist(d(\mathbf{s}_{\beta-1}), d(\mathbf{s}_\beta))$ between the number of bits in the encoded state description $d(\mathbf{s}_{\beta-1})$ and description $d(\mathbf{s}_\beta)$ provides a good estimate of the likelihood of the parameters satisfying ϕ_β. (The intuition here stems from estimating the number of bits that have to be changed in some initial state in order to satisfy the desired state.) The function **RelaxedFill**(x, β_i) returns the x^{th} best valid population for β_i produced while optimizing $dist(d(\mathbf{s}_{\beta-1}), d(\mathbf{s}_\beta))$.

6.4 Story Inconsistency

Enabling the story author to be arbitrarily sparse in their story specification implies that authored stories may completely omit expected beats, or contain other inconsistencies that need to be resolved by the system in order to generate a consistent narrative. Two successive beats $\{(\beta_{i-1}, \beta_i) \mid \beta_{i-1}, \beta_i \in \alpha, 1 \leq i \leq |\alpha|\}$ in a story arc α are *locally inconsistent* if either of the following two conditions are violated: (1) There exists one or more event instances $I \in \beta_i$ that have unspecified participants. (2) The participant object states $\mathbf{s}_{i-1}$ reached by executing the postcondition functions of all of the event instances in all prior beats up to β_{i-1} do not satisfy the collective beat-wide precondition function $\phi_{\beta_i}(\{\mathbf{s}_{I_1}, \ldots, \mathbf{s}_{I_{|\beta_i|}}\})$ (where each $I_j \in \beta_i$). The story arc α is said to be *globally inconsistent* if it contains more than one local inconsistency.

6.4.1 Local Inconsistency Resolution

Local inconsistency between two successive beats is resolved by ensuring that all event instances contained in the two beats are fully populated (i.e., all of their participants are defined) and valid. This is achieved by populating the parameters of incomplete event specifications, and inserting new event instances (and possibly new beats) to satisfy the preconditions of all events. All human-authored events are treated as hard constraints and our system never generates a resolution strategy where authored events are removed or reordered relative to one another.

Consider a locally inconsistent beat pair: (β_q, β_r) in a story arc α such that the world state $\mathbf{s}_{\beta_q}$ after the execution of β_q violates β_r's precondition function ϕ_{β_r}. To resolve this inconsistency, we formulate a problem domain $\Sigma = \langle \mathbf{\Psi}, \mathbf{\Theta} \rangle$ where $\mathbf{\Psi}$ is the space of possible compound world states that the objects in the world can possibly take, and $\mathbf{\Theta}$ is the space of all possible valid event instances for every e contained in the event lexicon $\mathbf{E}$. The problem instance is defined as: $\mathbf{P} = \langle \Sigma, \mathbf{s}_{\beta_q}, \phi_{\beta_r} \rangle$. We use a heuristic planner [16] to generate a sequence of complete event instances $\mathbf{I} \in \mathbf{\Theta}$, starting from $\mathbf{s}_{\beta_q}$, and eventually producing a world state that satisfies ϕ_{β_r}. The resulting plan $\mathbf{\Pi}(\mathbf{s}_{\beta_q}, \phi_{\beta_r}) = \{I_1, \cdots, I_n\}$ is inserted either within or between between the two beats β_q and β_r to produce an ordered set of consistent story beats $\{\beta'_q, \cdots, \beta'_r)$ that may both modify the original beats (β_q, β_r) and potentially introduce new beats. While the plan $\mathbf{\Pi}$ returned by the planner is a sequence of event instances, we identify events that can execute in parallel and add them to the same beat, while events operating with overlapping participants need to be executed in different beats.

Heuristic Function. In order to focus the search expansion to most promising states, we define a heuristic estimate of distance to the desired goal state, calculated by hamming distance based on the number of bits that need to be changed in order to reach the goal state:

$$h(s_{\mathbf{W}}, s^r_{\mathbf{W}}) = ||s_{\mathbf{W}} \oplus s^r_{\mathbf{W}}|| \tag{6.1}$$

where $s_{\mathbf{W}}$ represents the initial state of the world and all of its objects, $s^r_{\mathbf{W}}$ represents a state of the world where the beat β_r can successfully execute, and $||s_{\mathbf{W}} \oplus s^r_{\mathbf{W}}||$ returns the number of $1's$ occurring in the binary encoding vector of $s_{\mathbf{W}}$ XOR $s^r_{\mathbf{W}}$. However, the desired goal state $s^r_{\mathbf{W}}$ is not uniquely defined in our problem definition, since our goal formulation is simply the satisfaction of ϕ_{β_r} on some set of objects in $\mathbf{W}$. We first generate a possible $s^r_{\mathbf{W}}$ by iteratively changing the bits of $s_{\mathbf{W}}$ until $\phi_{\beta_r}(s'_{\mathbf{W}})$ evaluates to true for some $s'_{\mathbf{W}}$. In order to only consider the bits that are relevant to ϕ_{β_r}, we introduce a bitmask M_{β_r} that filters out all irrelevant bits. We thus get a modified definition of the heuristic function:

$$h(s_{\mathbf{W}}, \beta_r) = ||(s_{\mathbf{W}} \oplus s^r_{\mathbf{W}}) \; \& \; M_{\beta_r}|| \tag{6.2}$$

where $\&$ represents bitwise AND. The branching factor when considering the space of all possible event instances $\mathbf{\Theta}$, even when considering a small library of events, grows exponentially large due the large number of unique possible combinations of participants that each event can take at any state (see Section 5.3.2). To optimize our planner, we use Weighted A* [56] by introducing an inflation factor $\epsilon > 1$ to bias the exploration towards states that are more likely to satisfy ϕ_{β_r}, as follows:

$$f(s_{\mathbf{W}}) = g(s^0_{\mathbf{W}}, s_{\mathbf{W}}) + \epsilon \cdot h(s_{\mathbf{W}}, \beta_r) \tag{6.3}$$

For the experiments and results described in our work, we found that $2 \leq \epsilon \leq 4$ worked well and produced a dramatic reduction in planning cost. Practitioners may set $\epsilon = 1$ if their application necessitates strict optimality guarantees.

6.4.2 Global Inconsistency Resolution

A story arc with more than one local inconsistency cannot be resolved by independently resolving each local inconsistency. In this case, the resolution of one set of preconditions may invalidate other preconditions, making existing consistent beats inconsistent and creating a cycle of invalidation. In situations of multiple local inconsistencies, there may be cases where a coordinated resolution strategy is needed across multiple inconsistent beats where the *optimal* solution for one inconsistency may invalidate the possibility of *any* resolution for another inconsistency.

Consider a simple scenario with a guard and robber. The robber has a weapon for coercion or incapacitation and the guard has the key to the manager room which has a button to open the bank vault. We author an incomplete story where in β_1, the robber will unlock the door to the manager room (which requires the key) and in β_2, the robber will coerce the guard into pressing the vault button. Resolving the local inconsistency of β_1 produces a narrative where the robber incapacitates the guard and takes the key, allowing him to open the door. However, this prevents any possible solution for β_2 since the guard is incapacitated and can no longer be coerced into pressing the vault button.

To address this chained inconsistency resolution problem, we introduce an algorithm to resolve globally inconsistent story arcs in an efficient way. The key intuition is to back-propagate the preconditions of a later beat $\phi_{\beta_{i+1}}$ when a plan between two locally inconsistent beats (β_i, β_{i+1}) fails. A new plan for the previous local inconsistency (β_{i-1}, β_i) is generated that satisfies the stricter preconditions $\phi_{\beta_i} \wedge \phi_{\beta_{i+1}}$. In the previous example, we propagate $\phi_{\beta_2} \leftarrow \neg\texttt{Incapacitated(guard)}$ backwards to produce a resolution for (β_0, β_1). In so doing, we ensure that the guard is not incapacitated in the process of the robber taking the key. This produces a globally consistent narrative where the robber instead coerces the guard to hand over the keys and later to press the button to open the vault door. Algorithm 3 details two variants of the back-propagated planning algorithm. The theoretically complete version uses the lines indexed using $\ominus$ while a more practical estimation-based algorithm is obtained by replacing them with the lines indexed using $\oplus$.

Complete Algorithm. The algorithm iterates through successive beats in the story arc α. When it identifies an inconsistent beat pair (β_{i-1}, β_i), it first populates β_i with the optimal set of event participants (Line 8) to generate an example populated beat β^* using relaxed-fill estimation. The simulation function **Simulate**$(\cdot, \cdot)$ is a PAStE-based function used to predict the "current" world state (i.e., the expected world state at the time of beat invocation) $s^c_{\mathbf{W}}$ of the objects by forward simulating the postconditions of the preceding beats (Line 6). A plan $\mathbf{\Pi}(s^c_{\mathbf{W}}, \phi\beta^*)$ is computed to satisfy the preconditions of this optimal beat ϕ_{β^*} and inserted into the story arc (Lines 13, 14). This process continues until all inconsistencies are resolved. If the planner is unsuccessful, we increment j and try the next valid beat population (Lines 33 – 36) for this particular inconsistency.

If the planner is unable to resolve the local inconsistency for any of the valid beat populations, it propagates the preconditions backwards to earlier beats in the story. It marks the previous beat pair $(\beta_{i-2}, \beta_{i-1})$ as inconsistent (Line 23), thus invalidating the previously computed plan, and generates the set of preconditions for β_i using each possible beat population, denoted as Φ (Line 25). Each precondition $\phi_{prop} \in \Phi$ is successively concatenated to ϕ_{β_i} (Line 26) to check if a resolution strategy can be computed for the previous inconsistency while accommodating the preconditions of the current inconsistency, for a particular beat population (Line 10). In this case, we store both the precondition function and the set of objects on which that precondition must hold (hence the pairs of precondition function and object candidacy vector in Line 27). We use this function-object pairing

Algorithm 3: Algorithm for finding a solution to an inconsistent story arc

```
Data: $\alpha$, a partially specified story arc containing some beats
Data: $s^0_{\mathbf{W}}$, the initial world state
Result: An in-place modification of story arc $\alpha$
$s^c_{\mathbf{W}} = s^0_{\mathbf{W}}$ (set the current world state to the initial world state)
$\Phi_{=}\emptyset$ ($\Phi$ is the set of preconditions)
$i \leftarrow 1; j \leftarrow 1; k \leftarrow 1$
while $i \leq |\alpha| \wedge i \neq 0$ do
    if $\texttt{incons}(\beta_{i-1}, \beta_i) = \texttt{TRUE}$ then
        $s^c_{\mathbf{W}} = \textbf{Simulate}(s^0_{\mathbf{W}}, \{\beta_0, \cdots, \beta_i\})$
        $\lambda \leftarrow |\textbf{RelaxedFill}(\cdots, \beta_i)|$ ($\lambda$ is the number of populations)
        $\beta^* \leftarrow \textbf{RelaxedFill}(j, \beta_i)$
        if $\Phi \neq \emptyset$ then
            $\phi_{prop} \leftarrow \Phi_k$                                              $\ominus$
            $\phi_{prop} \leftarrow \Phi_1$                                              $\oplus$
            $\phi_{\beta^*} \leftarrow \phi_{prop} \wedge \phi_{\beta_i}$
        $\mathbf{P} = \langle \Sigma, s^c_{\mathbf{W}}, \phi_{\beta^*} \rangle$
        $\mathbf{\Pi}(s^c_{\mathbf{W}}, \phi_{\beta^*}) \leftarrow \textbf{Plan}(\mathbf{P})$
    if $\mathbf{\Pi}(s^c_{\mathbf{W}}, \phi_{\beta^*}) \neq \emptyset \vee \neg\texttt{incons}(\beta_{i-1}, \beta_i)$ then
        $\alpha \leftarrow \alpha \cup \mathbf{\Pi} s^c_{\mathbf{W}} \phi_{\beta^*}$
        $i \leftarrow i + 1; j \leftarrow 1$
        $k \leftarrow 1$                                                                 $\ominus$
        $\Phi \leftarrow \emptyset$
    else
        if $j \geq \lambda \wedge k \geq |\Phi|$                                         $\ominus$
        if $j > \varepsilon$                                                             $\oplus$
            $\texttt{incons}(\beta_{i-2}, \beta_{i-1}) \leftarrow \texttt{TRUE}$
            if $\Phi = \emptyset$ then
                $\{\mathbf{w}^1_{\beta_i}, \ldots, \mathbf{w}^m_{\beta_i}\} = \textbf{RelaxedFill}(\cdots, \beta_i)$      $\ominus$
                foreach $\mathbf{w}^p_{\beta_i} \in \{\mathbf{w}^1_{\beta_i}, \ldots, \mathbf{w}^m_{\beta_i}\}$            $\ominus$
                    $\Phi \leftarrow \Phi \cup (\phi_{\beta_i}, \mathbf{w}^p_{\beta_i})$                              $\ominus$
                $\mathbf{w}^1_{\beta_i} = \textbf{RelaxedFill}(1, \beta_i)$                                          $\oplus$
                $\Phi \leftarrow \{(\phi_{\beta_i}, \mathbf{w}^1_{\beta_i})\}$                                      $\oplus$
            $j = 1; i = i - 1$
            $k = 1$                                                                      $\ominus$
        else
            if $j \leq \lambda$ then                                                     $\ominus$
                $j \leftarrow j + 1$                                                     $\ominus$
            else                                                                         $\ominus$
                $j \leftarrow 1; k \leftarrow k + 1$                                     $\ominus$
            $j = j + 1$                                                                  $\oplus$
return $\alpha$
```

to specify that any prior beats must resolve in such a way as to enable the precondition function for future beats to evaluate to true. The back-propagation of preconditions continues to earlier beats (Line 30) until either the inconsistency is resolved, or the start of the arc is reached when no further back-propagation is possible.

It is important to note that this is not a traditional backtracking algorithm. Rather than performing a search through strings of potential solutions, when the global planner in CANVAS encounters an inconsistency, it identifies conflicting constraints on *later* events in the authored sequence and *moves* those preconditions to earlier gaps in the story to resolve as a single search problem. The system back-propagates preconditions rather than executing sequences of search problems on a stack. The core algorithm works by expanding local search problems with constraints taken from global inconsistencies. In this way, solutions for the local problem (with the addition of the back-propagated consistency preconditions) are guaranteed to resolve the later inconsistencies that were previously encountered in the planning process. Of course, a solution for the expanded global-to-local planning process may not exist if the event lexicon is limited.

Practical Estimation. The theoretical algorithm may take minutes to find a solution due to the exponential number of parameter combinations that need to be considered when resolving inconsistencies across many beats in large worlds. To offset this computational overhead, we introduce certain practical estimations to dramatically reduce the computational complexity of our approach at the expense of completeness guarantees:

1. We introduce a threshold ε that limits the maximum number of beat populations that are investigated (Line 22).

2. The back-propagation of preconditions is done only for the single best beat population (Replace Lines 25 – 27 with Lines 28, 29), rather than iterating down the list of all potential objects that may fill each beat's required roles.

This algorithm works well in practice with real-time planning, and provides a responsive authoring experience where the planner generates possible solutions almost instantly for the author to refine and iterate. If the algorithm is unable to find a globally consistent solution, we have two options. First, we can choose to provide the partially consistent story arc to the author for reference and highlight the remaining inconsistent story beat(s) to help the author resolve the inconsistencies. Second, we can fall back on the more theoretically complete satisfaction process as an offline step with the author's approval at the cost of more computation time.

The overall computational complexity of this algorithm is dependent on the nature of the virtual environment and the story being authored. The biggest contributor to complexity is the expected number of inconsistencies that require global re-planning, and this is entirely dependent on environmental factors and the author's intent. With respect to the task of populating potential beats in the complete and practical versions of the satisfaction algorithm for a given inconsistency, the computational complexity is, in the worst case, bounded by the total number of potential beats. This number relies on the number of objects in the world n, the maximum number k of events per beat, and the maximum number of participants per event p, and the size of the event lexicon e, as $O(e^{k\binom{n}{kp}})$. This represents the number of potential beats that can be considered in the world for inconsistency resolution, based on permutations of the event lexicon and combinations of characters and objects chosen from the world to populate them. The practical estimation algorithm, in contrast, selects only one beat (the "best guess" beat it had at the time the inconsistency was discovered), and uses that when back-propagating preconditions and computing a new local resolution problem, effectively making the "beat population" problem $O(1)$. Of course, this comes at the cost of completeness, where the algorithm may fail if that particular greedily-selected beat is irreconcilable.

Event Failure. We experimented with using CANVAS's real-time planning capabilities to produce new, alternative plans in case of an event failing to execute (because of a bad navigation request, or some animation or IK malfunction). CANVAS is capable of finding alternative events by automatically creating planning subdomains between the current state of the world (during the event failure), and the next intended event in the story series. However, in practice, we found that this was unnecessary. Events themselves (and behavior trees in general) are designed with robustness in mind, and are capable themselves of accounting for failure in the event execution structure. If an event itself fails in our system, it represents a major error that must be fixed, and attempts by CANVAS to recover and re-plan around the failed event often still produce undesirable and unavoidable visual artifacts.

Chapter 7

Real-Time Story Exploration via Storycraft

In addition to the CANVAS system for authoring stories as a cohesive storyboard experience, our system supports an additional module for creating stories in real-time using an exploratory narrative generation process. Where traditional story authoring exists in two phases, one where the author defines the story, and one where a viewer or participant experiences it, the real-time Storycraft system merges both the act of authoring and experiencing a story into one interaction with the system. To do so, we rely on offline pre-computation of the story space described by PAStE and CANVAS, with both quantitative and user-driven qualitative analysis of the story space.

7.1 Overview

During an offline step, we pre-compute the space of all possible narrative trajectories that may ensue within a collection of characters in an environment, referred to as the "story web". By performing graph analysis of the story web, we can extract a variety of low-level features about the shape and topology of the graph, including: (1) measures of node popularity (PageRank), assessing how likely a random exploration of the story space would visit it, (2) inverse measures that identify rarely visited nodes which open up new possibilities in the story space, and (3) measures of the strength of connection between two arbitrary nodes in the graph (MinCut).

These low-level features can be combined to create high-level intuitive heuristics that estimate the "narrative value" of any path between two nodes in the story web. For example, *bottleneck identification* identifies clusters in the graph and loosely connected segments between them. This heuristic highlights "points of no return" where events in the story irreversibly change the state of the world and its characters.

7.1.1 Sentiment Tagging and Analysis.

Environment designers and content creators can semantically tag events with their associated sentiment (e.g., loyalty, betrayal, etc.). These analyses produce an annotated story web that estimates both the quantitative and qualitative value of a narrative, as well as the sentiment associated with any story traversal. They additionally open up numerous opportunities to enrich and streamline the process of experiencing and creating stories.

Figure 4 illustrates the offline pre-computation process. The virtual world is exported in its initial configuration with all objects and characters in their starting state, along with a lexicon of all possible events that can be executed in the world. That export is loaded into an offline story explorer that produces an exhaustive story web by considering all possible ways of executing each event from any reachable world state. The story web is then stored and serialized for real-time interactive storytelling.

During the real-time story interaction, we explore two new modalities for a author to interact with the virtual world and its inhabitant, that are made possible using story webs.

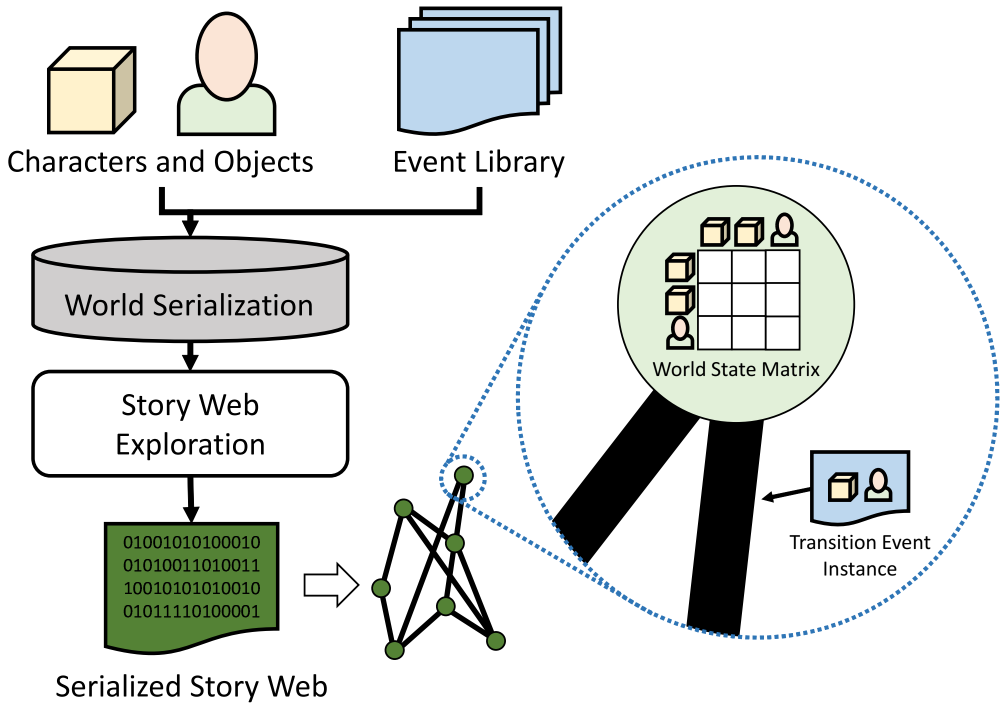

Figure 7.1: *An illustration of the story web pre-computation process.*

The first is a highly generative approach where a author dictates the progression of a story one step at a time, and the second is a more editorial modality where a author manipulates and diverts the course of complete generated story arcs as they unfold on-screen.

7.1.2 Generative Storytelling.

In generative storytelling, the author is presented with the virtual environment and given a list of potential events that can occur in the world involving different collections of characters and objects from the virtual world. As the author chooses events, those events execute in real-time (either sequentially or in parallel) and alter the state of the world. Because there may be a wide array of possible events for any particular story juncture, the system uses our state- and graph-analysis metrics on the complete generated story web to automatically suggest events that are more likely to take the story down a path that maximizes any author-specified criteria. These filters focus on identifying events that it believes the author would consider interesting. This allows the author to interactively explore the different ways in which characters and object may interact while progressively contributing to an overarching narrative.

7.1.3 Intervention Storytelling.

In intervention storytelling, our system selects and begins animating a default story using the characters and objects in the designed environment. At any point, the author can pause the story progression and alter its trajectory. This can be done either by modifying the sentiment of the story (i.e., changing a story about camaraderie to one based on betrayal), or by explicitly selecting the next event(s) for one or more story participants. After the author's intervention, the system continues autonomously along the new course of action determined by the author's perceived intent, until the author decides to intervene once again. As in the generative modality, the list of events presented to the author are filtered to remove meaningless and uninteresting options.

Figure 7.2 illustrates the online storytelling process (in the generative modality). The system loads in a serialized story web and plots candidate story trajectories based on the pre-computed graph topological features. The top trajectories are ranked and selected, and a series of choices are presented to the author for selecting the next event to execute. When the event is selected, it dispatches commands for the virtual actors and objects to carry out on screen. Afterwards, the change in world state is fed back to the storyteller and a new set of choices are selected for the author based on the story web.

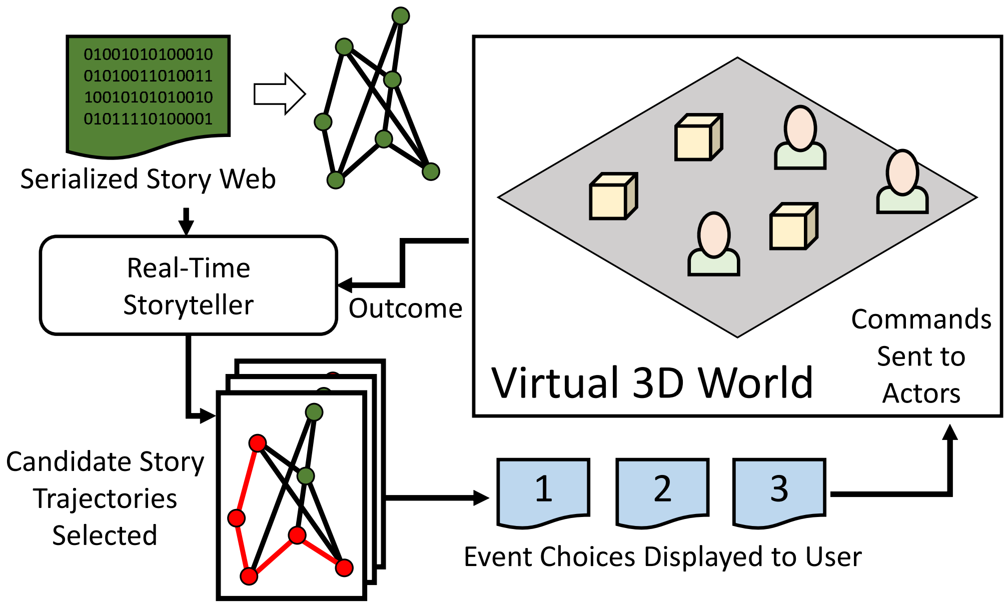

Figure 7.2: *An illustration of the real-time storytelling process.*

7.2 The Story Web

The Storycraft system operates on the same CANVAS/PAStE/ADAPT backbone, defining objects, their states, and the world's events in the same way. We depend on the story web as an additional data structure for the story exploration process. A story web $\mathbf{B} = (\mathbf{N}, \mathbf{A})$ is a graph with nodes $\mathbf{N}$ and edges $\mathbf{A}$. A node $N \in \mathbf{N}$ contains a complete representation $s_{\mathbf{W}}$ of the compound state of all of the smart objects in the virtual world $\mathbf{W}$, as well as containing space for storing scalar value features describing that world state configuration. As in PAStE, these compound world states are expressed as $|\mathbf{W}| \times |\mathbf{W}|$ matrices of encoded bit vectors containing both individual object attributes and pairwise relationships[1]. An edge $A \in \mathbf{A}$ is defined as a triple (N_i, I, N_j) containing a source node N_i, a target node N_j, and the event instance I that transforms the world state contained in N_i to that contained in N_j by modifying the attributes and relationships of some objects in

[1]Incidentally, each world state snapshot can be serialized very efficiently as a single delta-encoded matrix of fixed-size bit vectors where the asymmetric relations between objects form the two triangular sections, and individual character attributes occupy the diagonal.

$\mathbf{W}_{N_i}$. Paths in the story web can be converted to story arcs, each representing one possible traversal of the complete story space as well as the transformations of the world performed by the events in the web's edges (which are collapsed and converted to beats).

7.2.1 Pre-computation

The story web is created in several phases. First, the initial world state $s_{\mathbf{W}}$ is serialized and exported into a series of descriptive headers. These headers contain all of the story-relevant objects in the world with their initial state and relationships towards one another, as well as a complete listing of that world's event lexicon $\mathbf{E}$. The events are also tagged with one or more descriptive *sentiments* that align the event with a particular mood or theme such as "Brave" or "Non-violent". This world serialization is loaded into an external tool that will produce the story web.

Once the web explorer receives the tool, it performs a complete graph exploration of the web $\mathbf{B}$. For each node N in the graph, and for each event e in the complete event library, the explorer evaluates all possible event instances $I_e \in \mathbf{I}$ across all possible participant populations such that $\phi_e(\mathbf{w}_{I_e}) = 1$. For each such instance I, the explorer creates a new node and corresponding world state N', $s_{\mathbf{W}_{N'}}$ expressing the state created by modifying N by the postconditions Δ_e of the event e. A new edge A is created linking N and N' with the event instance I that precipitated their transition. The exploration terminates when all unique world state configurations have been found and any further event executions result in produced world states that have already been bound to nodes in the web. This process is illustrated in greater detail in Algorithm 4.

Pre-computation of the story web allows for later real-time analysis of the complete story space, including an understanding of the long-term trajectories of local event selection as performed by a user. Creating a full graph enables reasoning on the topological

Algorithm 4: Producing the story web through exhaustive story space exploration.

```
Data: an initial world configuration $\mathbf{W}_0$
Data: a library of events $\mathbf{E}$
Result: a story web $\mathbf{B} = (\mathbf{N}, \mathbf{A})$
1  $O = \{\mathbf{W}_0\}$ ;
2  $C = \{\mathbf{W}_0\}$ ;
3  $\mathbf{N} = \{\mathbf{W}_0\}$ ;
4  $\mathbf{A} = \{\}$ ;
5  while $|O| > 0$ do
6      let $n = \texttt{pop}(O)$ ;
7      for each $e \in \mathbf{E}$ do
8          for each permutation $\mathbf{w} \in \mathbf{W}_n^{|\mathbf{r}_e|}$ do
9              if $\phi_e(\mathbf{w}) == \mathit{TRUE}$ then
10                 let $n' = \texttt{transform(n, } e\texttt{)}$ ;
11                 if $n' \notin C$ then
12                     add $n'$ to $O$ ;
13                 add $n'$ to $C$ ;
14                 add $n'$ to $N$ ;
15                 add $(n, \langle e, \mathbf{w}\rangle, n')$ to $A$ ;
```

shape of the story itself, and can reveal qualities like climactic “points of no return” story moments without any domain-specific knowledge of the meaning of each world state element.

7.3 Story Web Analysis

After we have a complete graph containing connectivity information between world states, we pre-compute features and meta-data for use in story selection. The first analysis performed on the graph is a pre-computation of all of the shortest paths between nodes. This is a highly parallel process and we observed very promising results by computing this pathing step with a GPU-based “flood fill” algorithm.

7.3.1 Low-level Features

In addition to storing the shortest path between each node (and its length), we store three more heuristic features: Pagerank, Inverse Pagerank, and graph Min-Cut. Pagerank and Inverse Pagerank, are calculated using the techniques described by Page et al. [54] and their inverse, while Min-Cut is calculated according to a parallel version of the method described by Stoer et al. [82].

Pagerank. (Per node.) As a heuristic, Pagerank gives an estimate, for each node, of how likely a node is to be selected in a random exploration of the web. We use this heuristic to get an estimate of a node's "popularity", in terms of how much traffic can be directed to it in the graph.

Inverse Pagerank. (Per node.) In contrast, Inverse Pagerank gives an estimate of redundant or unnecessary nodes. That is, a node with a high Inverse Pagerank score is unlikely to be accessed, while also leading to many other nodes in the graph. We use this heuristic to identify "low-value" nodes that add little to the story beyond simply being another step in its trajectory.

Min-Cut. (Per node pair.) Min-Cut measures, between two nodes, the minimum number of edges that must be removed from the graph in order to sever all paths from the first node to the second. It gives a measure of the robustness of the graph's connectivity. We use min-cut to identify bottlenecks in our graph. Our intuition is that when two clusters of nodes are connected by a small bottleneck of one or two edges, then the events of those edges represent critical "climax" points of the story that drastically change the story along interesting points of no return. We also store the average min-cut in and out for each node.

Avg. Min-Cut In. (Per node.) Measures the average connective strength of other nodes in reaching this particular node. Nodes with high average min-cut in are very easy to reach and represent commonly reused story elements.

Avg. Min-Cut Out. (Per node.) Measures the average connective strength of this node in reaching other story nodes. Nodes with high average min-cut out represent the hubs of "story clusters", and can generally originate multiple diverse story trajectories.

These features are computed during the offline exploration process and serialized alongside the complete story web for use in real-time story exploration.

7.3.2 Heuristics and High-level Features

During the online story exploration process, we combine these low-level features into higher-level functions that assign a score to a potential story trajectory. Given a start state s, end state e, and path $p(s, e)$, we compute $cost(s, e, p(s, e))$. Note that $p(s, e)$ consists of a list of triples

$$p(s, e) = ((s, I_1, n_1), (n_1, I_2, n_2), \ldots, (n_{k-1}, I_k, n_k), (n_k, I_{k+1}, e))$$

where each triple (n_i, I_{i+1}, n_{i+1}) consists of a source node n_i, a target node n_{i+1}, and a connective event instance I_{i+1} that precipitates the transition between them. We have several different formulae for these cost functions, that can be changed based on story author preference. We prefer lower cost scores when selecting stories to execute. All values are assumed to be normalized both before and after computation.

Skinny-Pipe:

$$cost(s, e, p(s, e)) = mincut(s, e)/length(p(s, e))$$

Divides the min-cut from the first node to the second by the length of their path. This prefers longer stories with at least one major bottleneck.

Trap-Nodes:

$$cost(s, e, p(s, e)) = avgmincutout(e) * invpagerank(e)$$

Prioritizes story goal nodes with a low average page-rank out (i.e., low connectivity to the rest of the graph) and penalizes nodes with a high inverse pagerank (indicating low value). This finds nodes that represent drastically different states from their immediate peers, which would produce highly volatile chains of events that would cause major changes in the world when they execute.

Hub-Nodes:

$$cost(s, e, p(s, e)) = 1.0/(avgmincutout(e) * pagerank(e))$$

The opposite of trap-nodes, prioritizes "popular" nodes that can branch out to a wide array of other nodes, creating more possibilities for new and interesting story trajectories.

No-Return:

$$cost(s, e, p(s, e)) = mincut(e, s) - mincut(s, e)$$

Finds stories that are difficult to reverse, creating story trajectories along a "point of no return". This indicates transformative world events that modify the world state in drastic, meaningful, and irreversible ways. This value is normalized between 0 and 1, taking negative values into account.

Because these high-level heuristics are still rather simple, they can continue to be combined and weighted for a final story candidate analysis, taking additional factors like story path length into account.

7.3.3 Sentiment Tagging and Qualitative Analysis

In addition to the story feature heuristics, potential story start- and end-state pairs are weighted according to their involvement of events matching certain sentiments. When a cost score is computed for a start and end state, the scoring algorithm also traverses the path from start to end, and applies a bonus or penalty to the cost function depending on which sentiments appear in the path's transitions. The bonus or penalty is applied by multiplying or dividing the score by some small order of magnitude. For testing, our system's current sentiments are "Negligent", "Brave", "Violent", and "Nonviolent". These can be trivially expanded with additional themes and moods by tagging events in the event library. For a given start-end node pair, the score for that path is multiplied by a constant k for every occurrence of a desired sentiment, and divided by k for every occurrence of a sentiment that is not desired.

7.4 Story Generation

Story generation is dependent on the loaded story web and the score calculations based on sentiment and the cost function evaluation. Story generation proceeds in rounds, where the system identifies its current state and then evaluates potential candidates for the next event instance to execute. Given the current node c in the story web $\mathbf{B} = (\mathbf{N}, \mathbf{A})$, the system iterates over each potential goal node $g_i \in \mathbf{N}$ s.t. $\exists p(c, g_i)$ and computes $t_i = cost(c, g_i, p(c, g_i))$ (including sentiment bonuses or penalties). We create a list of scored goals $S = ((t_i, g_i, p(c, g_i)), \ldots)$ sorted in ascending order by t_i. Next, our goal is to produce k (a user-configurable constant) potential unique event instances to execute from the given state.

Algorithm 5: Producing the list of candidate events.

```
   Data: a constant $k$
   Data: a sorted list of scored goals $S = ((t_i, g_i, p(c, g_i)), \ldots)$
   Result: a sorted list of candidate instances and results $D = ((I_1, n_1), \ldots)$
1  $D = ()$ ;
2  $V = ()$ ;
3  for each $(t_i, g_i, p(c, g_i)) \in S$ do
4      expand $p(c, g_i) = ((c, I_1, n_1), \ldots, (n_k, I_{k+1}, g_i))$ ;
5      if $I_1 \notin V$ then
6          add $(I_1, n_1)$ to $D$ ;
7          add $I_1$ to $V$ ;
8      if $|D| == k$ then
9          break ;
```

For each tuple $(t_i, g_i, p(c, g_i))$, we extract the first event to be executed in the path. That is, if $p(c, g_i) = ((c, I_1, n_1), (n_1, I_2, n_2), \ldots, (n_k, I_{k+1}, g_i)))$, we extract I_1 and nominate that instance as a candidate for the next transition. We store the candidate instance I_1 and the next state in the path n_1 as a tuple (I_1, n_1) our candidate list. Note that while two different goals g_{i_1} and g_{i_2} may differ, the paths $p(c, g_{i_1})$ and $p(c, g_{i_2})$ may have the same first transition and resulting event instance, hence we filter for only presenting unique events by potentially skipping entire goals. Algorithm 5 illustrates this process in action.

Once a candidate (I_1, n_1) is selected from the list, the results of the event execution are displayed on-screen, and the current world state node becomes n_1. We then recalculate goals, and produce new candidates. For consistency, and to prevent vacillating goals in continuous stories, we always add the goal from the previously selected event candidate as the first goal in our new candidate list S for this round that we present to the user, so long as that goal is still reachable from our new state. This ensures that the first candidate option is to continue along the path towards the previously selected goal.

Ultimately, the process of story generation is to produce lists of candidates, select those candidates (either according to the user's wishes or automatically), wait for the selected event to play out on screen, transition to the new state, and then generate a new set of candidates. There are two control modalities that we can employ with respect to selecting events. These two modalities have no bearing on the manner that candidates are selected, but rather represent two different types of interaction with the user in terms of story control.

7.4.1 Generative Storytelling

Generative storytelling exposes the candidate events directly to the user at each round and waits for the user to pick each event from the top choices. During each round, the system pauses and the user is given an illustration of the top candidate events and which characters and objects those events involve. The user can also modify the story goal selection heuristics (sentiment and cost function) in real-time to generate new sets of candidates. When the user picks an event, the event executes on-screen and eventually terminates, at which point the system presents the user with new candidates. In this modality, the user is wholly responsible for generating the story based on the options available and picks events on a round-by-round basis.

7.4.2 Intervention Storytelling

Intervention storytelling instructs the system to automatically select the best candidate, but allows the user to pause the story at any time to change selection criteria or manually select an event as per generative storytelling. When left completely untouched, the automatic storyteller will play out and animate an entire story from start to finish. At any point, and in real time, the user can pause the story and either change goal selection criteria, or explicitly select the next event from a set of candidates. After this point, the user can continue to select events one after another as in generative storytelling, or unpause the automatic storyteller and continue until deciding to intervene instead. In this modality, the user is charged with selectively shaping the trajectory of the story, but is not necessarily concerned with every detail.

Chapter 8

Results

To demonstrate the capabilities of CANVAS and Storycraft, we designed a bank robbery scenario illustrated in Figure 8.1. Section 8.1 introduces the objects and actors used in this scenario, with their affordances and capabilities, and Section 8.2 describes the event lexicon exposed to the CANVAS and Storycraft authors to create the stories we demonstrate in this environment.

8.1 Objects, Actors, and the Environment

The bank robbery scenario includes 65 total objects. Of these, 19 of these objects are actors, split into four categories as follows:

Robbers. The robbers serve as the principal characters in the scene and are equipped with weapons that can be used to fire a warning shot (to frighten the customers), incapacitate other characters, or coerce them into opening doors and pressing buttons.

Guard. Guards keep the bank safe from robbers and are equipped a keycard to the locked doors in the scene. Some guards are also equipped with weapons.

Bank Teller and Manager. Tellers work at the bank counter to serve the customers while the manager oversees bank operations.

Customers. Bank customers provide ambient activity in the scene; wandering the bank, purchasing and consuming beverages from the drink dispenser, recycling used cans, and filling in forms. The customers are controlled by a group coordinator object (as described in Section 5.3.1), but they can be dynamically removed from their coordinator group. As such, the customers, bank tellers, and any other character in the scene can be promoted from a background role to playing a more significant part in the narrative depending on the author's intent.

In addition to the categories of actors, there are numerous other smart object props that are used for the actors to interact with in the scene, as follows:

Doors and Buttons. The doors to the teller room and vault entry area are locked by default and are additionally guarded by the security guards. There are two buttons located in the manager's office and teller room which must both be pressed (but not simultaneously) to unlock the vault door.

Bank Counters. The bank counters themselves have affordances for customers to fill out forms and present them to the bank tellers for service. These are used to present a more realistic set of baseline activities for the characters in the scene.

Ambient Props. Props such as chairs, a drink dispenser, and a trash can are added to the scene and can be interacted with. Characters can sit, purchase a drink, and dispose of it respectively using these props' affordances.

Note that small hand-held props themselves (like keys, weapons, and service forms) are not modelled themselves as smart objects. Rather, they are visual representations of character attributes such as `HasWeapon` or `HasKey`. The affordances of the smart objects read these character attributes and advertise additional affordances (such as a door's "Unlock" affordance if a character is holding a key).

8.2 Story Events

The bank scenario has an event lexicon containing 54 events. Of these, 11 are group events concerned with the background activity of characters controlled by a group coordinator object. A domain expert can easily add and remove events as needed, using a simple programming interface that directly links with the event lexicon. Even with a handful of events, we are able to author a variety of compelling narratives with vastly different outcomes. We outline a representative set of events and their metadata in Table 8.1, and we provide a brief high-level description as follows:

CoerceIntoUnlockDoor(Actor a_1, Actor a_2, Door d). Actor a_1 coerces a_2 into opening the door, d. In order for this event to be successful, a_1 must have a weapon, a_2 must have the keycard to open d, and must be able to access d.

IncapacitateStealthly(Actor a_1, Actor a_2). Actor a_1 sneaks up on a_2 and incapacitates him using his weapon. Actor a_1 must have a weapon and should be able to reach a_2 without being seen by him.

WarningShot(Actor a, Crowd: c). Actor a fires his weapon into the air, frightening the members of crowd c. The precondition for this event is that a must have a weapon.

TakeWeaponFromIncapacitated(Actor a_1, Actor a_2). Actor a_1 takes the weapon of a_2 who has been previously incapacitated. Actor a_2 must have a weapon and a_1 must be able to reach a_2.

DistractAndIncapacitate(Actor a_1, Actor a_2, Actor a_3). Actor a_1 distracts a_2 while a_3 sneaks up from behind to incapacitate a_2 using his weapon. For example, two robbers can cooperate to distract and incapacitate a guard using this event.

PressButton(Actor a, Button b). Actor a presses a button b which may have some effect elsewhere in the scene (e.g., unlocking the vault door). Actor a must have access to b in order to execute this event.

LockDoor(Actor a, Door d). Actor a locks the door d. In order to do this, he needs to have the keycard and should be able to access d. This event can be used to lock other characters in a room.

Flee(Crowd c). The members of c find the nearest exit and leave the bank. This event can be used to trigger the response of the crowd to the arrival of the robbers, for example.

8.3 Scenario Creation Effort

We estimate that within the framework of our system, the bank robbery scenario was created in a total of 50 man-hours by experienced programmers (not including planning and graphical asset creation). By far the most time consuming task was that of designing affordances for each smart object, and coordinating the animations used by the characters interacting with those objects (including IK targets, navigation waypoints, and trigger activation timing). Given a geometric level design and smart object affordances, however, the task of authoring behavior tree events was comparatively simple and iteration on the behavior trees themselves can be done with an immediate turn-around time. Note that this does not include the time spent on authoring the stories themselves, which can be completed in a matter of minutes by a skilled user of our CANVAS interface.

It is important to note that the addition of a single smart object to the environment does not automatically incur a performance or creative effort cost with respect to any of the other objects already in the world. Affordances are tied to existing user types (for example, a door expects a human character user). In the process of creating a new smart object, the object's designer decides which other types of objects in the world should interact with it (usually just a human), and creates affordances for those user object types. Likewise, if the new object's designer decides that that new object should be able to interact with pre-existing objects, then those pre-existing objects should be given new affordances to accommodate the new object. In practice, most smart objects fall into two categories: props (which are to be interacted with), and human actors (which perform the interactions). Almost every new smart object is usually just a prop designed to accommodate human interaction, and ultimately requires the addition of only a few new affordances.

Many of the narrative assets created in the bank robbery scenario could be reused in other settings. Interactions such as conversations, opening and closing doors, or exchanging props are highly portable and applicable to a large number of story domains. Smart objects themselves can be re-purposed for different environments and can typically perform their duties when placed in a variety of locations. Similarly, since behavior tree events rely only on the smart objects involved in them, they can be ported to different scenarios with minimal refactoring or adaptation effort. We expect that over time, scenario authors will develop a collection of smart objects and environments similar to a "back lot" of a movie studio. As in the real-world analogy, these object props can be periodically reused in other settings with minimal adaptation thanks to the nature of PBTs and events. The process of designing an environment, however, is a task that requires skilled programmers, especially if new smart objects must be created and used. End-users would not be capable of creating an entirely new story domain within the current iteration of our system.

8.4 Explicitly Authored Narratives

To demonstrate CANVAS specifically, we author a variety of narratives and execute them in the virtual scene. In this framework, the process of automatically generating globally consistent narratives, even for extremely sparse specifications where the planner had to introduce more than five beats into the story arc, takes only a few seconds on commodity hardware. We found that the practical variant of Algorithm 3 worked well in most cases and, when compared to the more theoretically complete algorithm, performed an order of magnitude faster while still providing useful results.

8.4.1 Fully Manual Authoring

Our manually-authored story begins with robbers r_1, r_2, and r_3 entering the bank from the bank's rear entrance. Robber r_1 incapacitates the guard g_1 who is protecting the back entrance. He then proceeds down the hallway to distract g_2. Meanwhile, r_2 sneaks up from behind and incapacitates g_2. Robber r_3 enters the main bank lobby and fires a warning shot at the ceiling of the lobby. Upon hearing the shot, the bank customers flee from the bank while r_1 takes the keys from the incapacitated guard g_2. He then proceeds to unlock the teller door, while r_2 and r_3 secretly scheme to betray r_1. Robber r_2 breaks his alliance with r_1 and stealthily incapacitates him from behind. At the same time, r_3 coerces the manager into unlocking the door to the vault entrance area. Robber r_2 enters the manager's office and presses the manager vault button while the manager and bank tellers escape. While r_2 is opening the vault, r_3 sneaks up and incapacitates him. Robber r_3 takes the money and escapes the bank alone with all of the money.

8.4.2 Automated Parameter Selection

Our first variant of the original narrative narrative outlines the sequence of events without specifying all of the event's participants. In this case, the author specifies the following story, which is missing explicit parameterization. Some three characters enter the bank and proceed to incapacitate the guards, as before. One character fires a warning shot which causes the customers to flee while another coerces someone to press the teller button. Some character then coerces the manager into unlocking his door and incapacitates him, while another unspecified character presses the manager button. Someone opens the vault and the three characters take all the money and escape. Figure 8.2 displays the sequence of animations synthesized by CANVAS in this case, and Figure 8.3 illustrates the authored narrative with the automatically populated event parameters. The CANVAS planner automatically selects a suitable combination of actors and objects (highlighted) to fill the roles of the story while ensuring the consistency of the story progression as partially dictated by the author. The user may choose between the two population heuristics described in Section 6.2: *salience* prefers characters who have already participated in events, while *balanced usage* prefers to distribute the events evenly among all valid actors in the scene.

8.4.3 Global Inconsistency Resolution

We created a second story variant using events that map only to climactic plot points in the story arc, without fully specifying the set of causally preceding events that lead up to them. This story, as authored, assumes there is only one guard in the bank, g_1. We specified that robber r_1 fires a warning shot and unlocks the teller door (which can not be done without obtaining a key). He then coerces g_1 into pressing the teller button and locks the guard in the teller room. Finally, we specify that some customer coerces r_1 into surrendering (which can not be done without a obtaining a weapon). Figure 8.4(a) illustrates the authored narrative with the highlighted nodes and parameters automatically inserted by CANVAS to create a globally consistent narrative. The planner inserts two beats at the beginning where r_1 first coerces g_1 into dropping his weapon, and takes his keys. This allows r_1 to unlock the teller door (as was authored) and to coerce g_1 into pressing the teller button. Finally, in order to satisfy the authored constraint where a customer must coerce the robber into surrendering, the planner inserts an event into the story where the customer picks up the weapon dropped by the guard at the beginning of the story, before proceeding with the rest of the author-specified resolution.

8.4.4 Fully Automated Narrative Generation

As a final variant of our story, we specified as little as possible to the CANVAS system, and let the planner generate its own story to satisfy our requirements. In this case, we specified only that a robber must take the money from the vault and escape. CANVAS in this case generates a complete story where the robber proceeds to coerce or incapacitate the guards to access the manager and teller rooms and press the buttons to open the vault door so he can successfully steal the money. After such a story is generated, users may edit the story to meet further requirements or choose from different possible narratives that suits their needs while relying on automation for story generation. This type of full narrative generation serves as a good starting point when creating a story, rather than requiring every user to begin each story from scratch. Figure 8.4(b) illustrates the minimally-specified desired outcome, and the CANVAS-generated story (highlighted).

8.5 Dynamically Exploring Narratives

In addition to the stories generated during an explicit authoring phase via CANVAS, we can use Storycraft to explore stories in the bank scenario in real-time using a precomputed story web. We additionally demonstrate three real-time authored narratives designed with both modalities of Storycraft. In Storycraft's real-time authoring mode, the user is presented with a simple dialog box to change the overall sentiment of the story (currently "Brave", "Violent", "Nonviolent", and "Negligent"), and to specifically intervene when desired to explicitly select events. The user can choose to "Select Events Automatically", which places the system into the intervention modality and will automatically generate a complete story from the user's chosen criteria. Alternatively, while the story is running, the user can select "Pause after Next Event", at which point, once the event terminates, the system will pause the story and switch to the generative modality. In the generative modality, the user is presented with three choices, each reflecting the next event in one potential story line for the virtual world and its characters. While paused, the user can also change the sentiment selections using a list of check boxes to change the story lines that the heuristic story explorer considers when generating event choices. Once the user clicks one of the three[1] event description buttons to choose the next event in the sequence, the event is executed by dispatching animation synthesis commands to the characters and objects in the virtual world, and then the system pauses and waits after presenting the next batch of choices to the user. At any point during the generative modality, the user can return to the intervention modality by instructing the system to begin automatically selecting events in sequence. Figure 8.5 displays a screenshot of the real-time storytelling

[1]This number is configurable.

interface, and Figure 8.6 shows a close-up view of the control dialog, including the button to switch modalities, the sentiment selection check boxes, and the event choice descriptor buttons.

Our first Storycraft narrative is created using the intervention modality with very little direct user input. The user selects only the “Nonviolent” sentiment and instructs the storyteller to proceed. The storyteller then generates a story where the robber coerces the guard to unlock a door and press a button to partially unlock the bank vault, then proceeds into the vault to steal its contents. Because of the “Nonviolent” sentiment selection, the robber does nothing to harm either the guard or the bank teller behind the counter.

Our second Storycraft narrative begins as the first, also using the intervention modality. However, in this case, just after the robber coerces the guard to press the bank vault button behind the teller counter, the user intervenes. The user pauses the event progression, which waits for the currently running event to terminate and then presents an event selection prompt in the control dialog (generative modality). In this case, the user opts to change the sentiment from “Nonviolent” to “Violent”, and explicitly selects one of the next events to execute. The user then returns the system to the intervention modality by instructing the storyteller to automatically select events. The story that is generated from this case differs significantly from the first, even though the two stories began the same way. In this “Violent”-sentiment story, the robber incapacitates both the guard and the teller, stealing a key from the guard and using it to proceed into the vault and empty its contents.

The third narrative is created entirely through the generative authoring modality. The user creates the entire story semi-manually by selecting an event, waiting for the event to terminate, and then making a choice about the next event to execute. In this case, the user begins with the “Violent” sentiment, and instructs the robber to incapacitate the guard and steal the guard’s keys. The user then replaces the “Violent“ sentiment with “Negligent”, which enables the option for the robber to drop his weapon on the floor. The robber then uses the guard's keys to begin unlocking doors and pressing the two necessary buttons to unlock the final vault door. However, during this process, the user once again replaces the "Negligent" sentiment with "Brave". This prioritizes a new story trajectory where the bank teller thwarts the robber's attempt to rob the bank. The "Brave" sentiment prioritizes an event where the teller picks up the robber's discarded weapon by the incapacitated guard. The user then selects the "Violent" sentiment (in addition to "Brave"), and is presented with the option for the teller to sneak up behind and incapacitate the robber with the weapon.

All three of these stories were created with minimal and/or intuitive control from the user, requiring little training or experience with the underlying story engine beyond the ability to select from the simple presented choices. The sentiment-based heuristic search produces interesting and varied story trajectories with drastically differing outcomes and methods, even in constrained scenarios such as a bank robbery. We expect that using a combination of these two real-time Storycraft modalities, casual users can easily exploit the event-driven system to create interesting and meaningful stories with minimal effort.

8.5.1 Metrics

The Storycraft offline exploration for the bank robbery scenario produced an upper limit of 12,992 story web nodes (variations during iteration and experimentation caused this number to fluctuate). We reduced the number of objects and events for this process because of a lack of early optimization. Subsequent optimizations and GPU processing for story web generation for this scenario demonstrate promising preliminary results, bringing computation time down from approximately 20 hours to an order of minutes. Depending on acceptable turnover time (we consider 8 hours an upper limit on a single machine), we expect that an optimized Storycraft system could accommodate the full bank robbery scenario (65 characters and objects, and a lexicon of 54 events). Given the highly parallel nature of the exploration algorithm, however, this computation could be scaled and distributed to accommodate much larger and richer environments and stories.

8.6 Independent Authors

In the process of evaluating CANVAS, we designed a simple preliminary experiment to evaluate developed competency with the system in untrained authors. We asked users to perform two tasks.

Task 1. Given a partially authored story (consisting of the robbers entering the bank and incapacitating guards), complete the narrative trajectory by having at least one robber successfully exit the bank with money from the vault.

Task 2. Taking the previously completed story, alter the ending so that the robbers are thwarted in their effort to rob the bank and leave (either through guard or civilian intervention).

These tasks were always presented in order after a five minute verbal orientation with the system. Both tasks required the addition, population, and/or manipulation of 2-3 events to accomplish, requiring approximately 7 interactions with the system's interface.

Ultimately we evaluated 12 users with moderate computer experience (but no prior experience with our system). The average time taken to accomplish task 1 was 6 minutes and 48 seconds, while the average time for task 2 was 4 minutes and 19 seconds. While not conclusive, we believe this decrease in time indicates acquired proficiency with the system. Feedback was generally positive, but users cited issues with clarity of the event library (events were difficult to distinguish, and in some cases the meaning of the event's name was unclear) and general user interface interaction issues (scrolling, dragging, zooming, etc.). While not a conclusive, formal user evaluation, we believe this test shows promise with respect to a naive user's ability to create coherent stories using our CANVAS system.

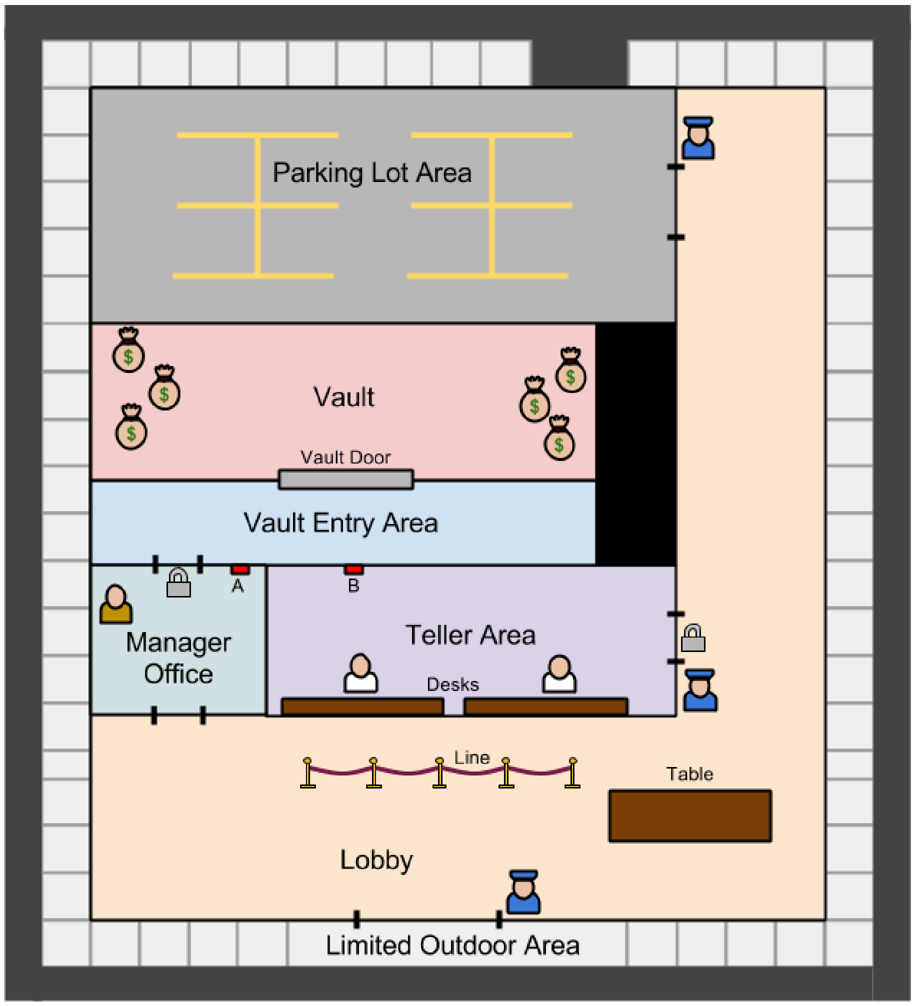

Figure 8.1: *A sketch of the layout for the bank scenario. A and B are buttons that need to be pressed in order to unlock the vault. The robbers begin the scene in the rear parking lot, and must deal with the guards (in blue), tellers (in white), and manager (in brown) to steal money from the vault.*

Event: *OpenVault*
Params: Actor: a_1, [*VaultDoor: d, Zone: z_1, z_2*]
Precondition ϕ:
$\neg\texttt{Incapacitated}(a_1) \wedge \texttt{BMPressed}(d)$
$\wedge\texttt{BTPressed}(d) \wedge \neg\texttt{Open}(d)$
$\wedge\texttt{InZone}(a_1, z_1) \wedge \texttt{FrontZone}(d, z_2)$
$\wedge\texttt{CanAccessZone}(a_1, z_2)$
Postcondition Δ:
$\texttt{Open}(d) \wedge \neg\texttt{InZone}(a_1, z_1)$
$\wedge\texttt{InZone}(a_1, z_2)$
Behavior Summmary:
a_1 opens the vault door

Event: *TakeMoney*
Params: Actor: a_1, Container: *c*, [*Zone: z_1, z_2*]
Precondition ϕ:
$\neg\texttt{Incapacitated}(a_1) \wedge \texttt{Occupied}(c)$
$\wedge\texttt{HasMoney}(c) \wedge \texttt{InZone}(c, z_2)$
$\wedge\texttt{InZone}(a_1, z_1) \wedge \texttt{CanAccessObj}(a_1, c)$
Postcondition Δ:
$\texttt{HasMoney}(a_1)\neg\texttt{Occupied}(c)$
$\wedge\neg\texttt{HasMoney}(c) \wedge \texttt{InZone}(a_1, z_2)$
$\wedge\neg\texttt{InZone}(a_1, z_1)$
Behavior Summmary:
a_1 takes the money out of container *c*

Event: *LockDoor*
Params: Actor: a_1, Door: *d*, [*Zone: z_1, z_2*]
Precondition ϕ:
$\texttt{HasKeys}(a_1) \wedge \neg\texttt{Incapacitated}(a_1)$
$\wedge\texttt{Unlocked}(d) \wedge \texttt{CanManipulate}(a_1, d)$
$\wedge\texttt{InZone}(a_1, z_1) \wedge \texttt{FrontZone}(d, z_2)$
$\wedge\texttt{CanAccessObj}(a_1, z_2)$
Postcondition Δ:
$\neg\texttt{Unlocked}(d) \wedge \neg\texttt{InZone}(a_1, z_1)$
$\wedge\texttt{InZone}(a_1, z_2)$
Behavior Summmary:
a_1 locks *d*

Event: *DistractAndIncapacitate*
Params: Actor: a_1, a_2, a_3, [*Zone: z_1, z_2, z_3*]
Precondition ϕ:
$\texttt{HasWeapon}(a_3) \wedge \neg\texttt{Incapacitated}(a_1)$
$\wedge\neg\texttt{Incapacitated}(a_2) \wedge \neg\texttt{Allied}(a_1, a_2)$
$\wedge\neg\texttt{Incapacitated}(a_3) \wedge \neg\texttt{Allied}(a_3, a_2)$
$\wedge\texttt{InZone}(a_1, z_1) \wedge \texttt{InZone}(a_2, z_2)$
$\wedge\texttt{InZone}(a_3, z_3) \wedge \texttt{CanAccessObj}(a_1, a_2)$
$\wedge\texttt{CanAccessObj}(a_3, a_2)$
Postcondition Δ:
$\texttt{Incapacitated}(a_2) \wedge \neg\texttt{InZone}(a_1, z_1)$
$\wedge\texttt{InZone}(a_1, z_2) \wedge \neg\texttt{InZone}(a_3, z_3)$
$\wedge\texttt{InZone}(a_3, z_2)$
Behavior Summmary:
a_1 distracts a_2, so a_3 can incapacitate a_2

Event: *IncapacitateStealthily*
Params: Actor: a_1, a_2, [*Zone: z_1, z_2*]
Precondition ϕ:
$\texttt{HasWeapon}(a_1) \wedge \neg\texttt{Incapacitated}(a_1)$
$\wedge\neg\texttt{Incapacitated}(a_2) \wedge \texttt{InZone}(a_1, z_1)$
$\wedge\neg\texttt{Allied}(a_1, a_2) \wedge \texttt{InZone}(a_2, z_2)$
$\wedge\texttt{CanAccessObj}(a_1, a_2)$
Postcondition Δ:
$\texttt{Incapacitated}(a_2) \wedge \neg\texttt{InZone}(a_1, z_1)$
$\wedge\texttt{InZone}(a_1, z_2)$
Behavior Summmary:
a_1 incapacitates a_2 from behind

Event: *TakeWeaponFromIncapacitated*
Params: Actor: a_1, a_2, [*Zone: z_1, z_2*]
Precondition ϕ:
$\neg\texttt{Incapacitated}(a_1) \wedge \texttt{HasWeapon}(a_2)$
$\wedge\texttt{Incapacitated}(a_2) \wedge \texttt{InZone}(a_1, z_1)$
$\wedge\texttt{InZone}(a_2, z_2) \wedge \texttt{CanAccessObj}(a_1, a_2)$
$\wedge\texttt{RHandEmpty}(a_1)$
Postcondition Δ:
$\texttt{HasWeapon}(a_1) \wedge \neg\texttt{InZone}(a_1, z_1)$
$\wedge\neg\texttt{HasKeys}(a_2) \wedge \texttt{InZone}(a_1, z_2)$
$\wedge\neg\texttt{RHandEmpty}(a_1)$
Behavior Summmary:
a_1 takes keys from the incapacitated a_2

Event: *PressButton*
Params: Actor: a_1, Button: *b*,
[*VaultDoor: d, Zone: z_1, z_2*]
Precondition ϕ:
$\neg\texttt{Incapacitated}(a_1) \wedge \texttt{InZone}(a_1, z_1)$
$\wedge\texttt{InZone}(b, z_2) \wedge \texttt{CanAccessObj}(a_1, b)$
$\wedge\texttt{CanManipulate}(a_1, b)$
Postcondition Δ:
$\texttt{BPressed}(b, d) \wedge \neg\texttt{InZone}(a_1, z_1)$
$\wedge\texttt{InZone}(a_1, z_2)$
Behavior Summmary:
a_1 presses the button *b*

Event: *CoerceIntoUnlockDoor*
Params: Actor: a_1, a_2, Door: *d*, [*Zone: z_1, z_2*]
Precondition ϕ:
$\texttt{HasWeapon}(a_1) \wedge \neg\texttt{Incapacitated}(a_1)$
$\wedge\neg\texttt{Incapacitated}(a_2) \wedge \texttt{HasKeys}(a_2)$
$\wedge\neg\texttt{Unlocked}(d) \wedge \texttt{InZone}(a_1, z_1)$
$\wedge\neg\texttt{Allied}(a_1, a_2) \wedge \texttt{InZone}(a_2, z_2)$
$\wedge\texttt{FrontZone}(d, z_2) \wedge \texttt{CanAccessObj}(a_1, a_2)$
$\wedge\texttt{CanManipulate}(a_2, d)$
Postcondition Δ:
$\texttt{Unlocked}(d) \wedge \neg\texttt{InZone}(a_1, z_1)$
$\wedge\texttt{InZone}(a_1, z_2)$
Behavior Summmary:
a_1 coerces a_2 into unlocking door *d*

Table 8.1: *Metadata definition of some events used in the bank scenario. We summarize the behavior for each event when executed, but in practice the logic and control structures for these events are authored as PBTs and are not shown here. Implicit parameters are italicized and indicated with brackets.*

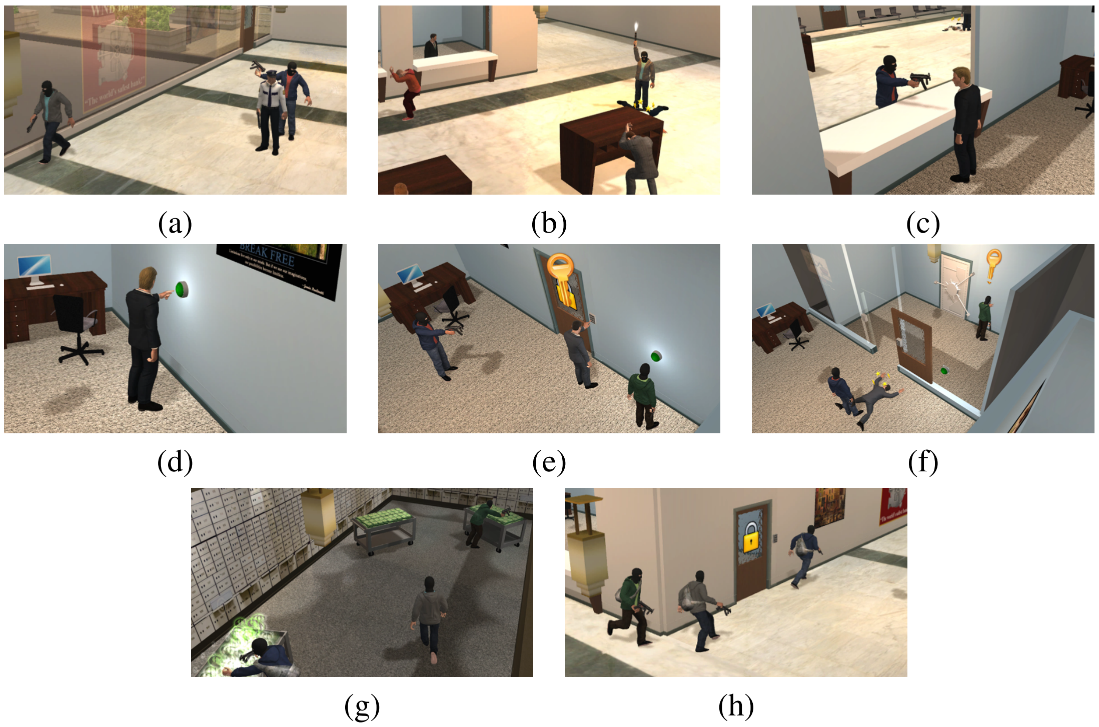

Figure 8.2: *A complex narrative authored using CANVAS. (a) Robbers enter the bank from the back door and begin incapacitating guards. (b) A robber fires a shot into the air to intimidate the crowd. (c) A second robber coerces the a teller to (d) press a button behind his desk to unlock the vault. (e) The robbers enter the manager's office and coerce the manager to unlock the door leading to the vault, while also pressing the second button needed to unlock the vault door. (f) The robbers incapacitate the manager and open the vault door. (g) The three robbers steal the money from the vault and (h) they escape by running out the back entrance.*

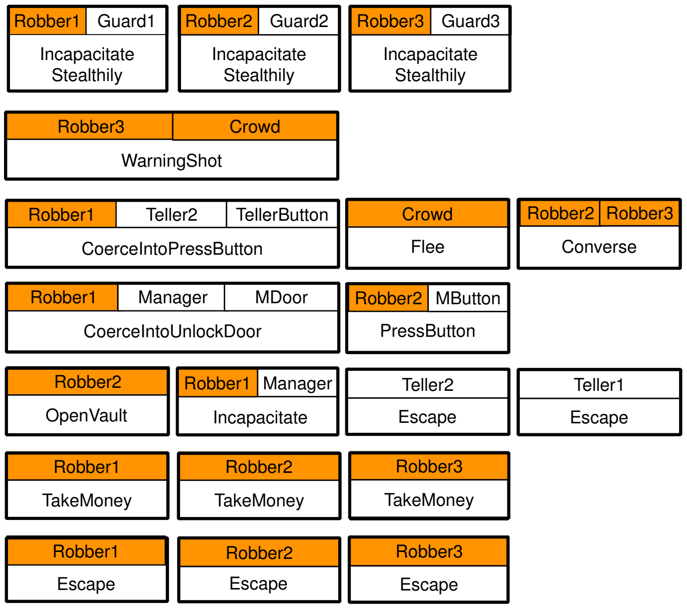

Figure 8.3: *Filling in an incomplete story specification. Automatically filled in parameters are highlighted in orange.*

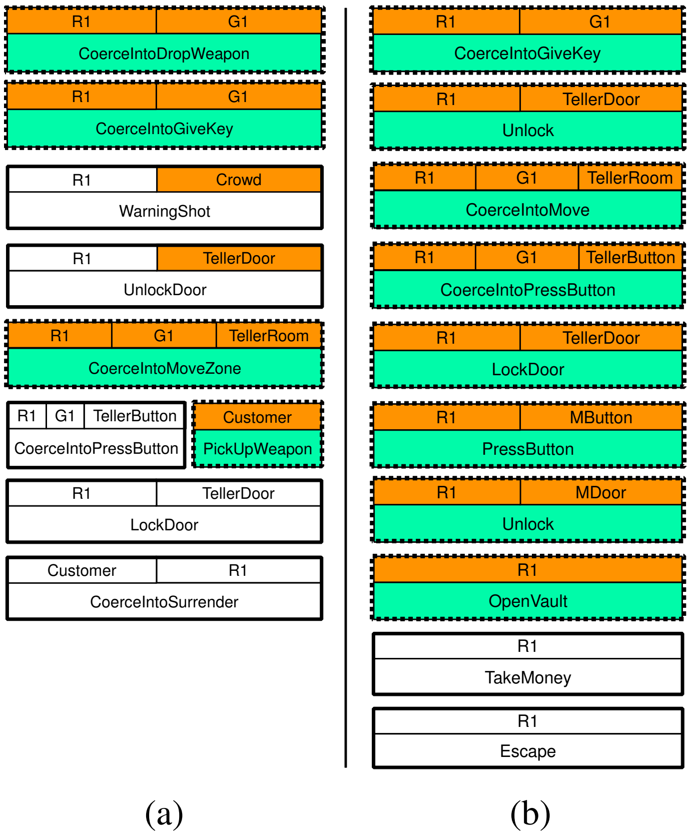

Figure 8.4: *(a) A globally inconsistent story definition. CANVAS fills in missing parameters (orange) and inserts new story events (green) to automatically complete the story. (b) CANVAS can automatically generate an entire story to satisfy a user-specified desired outcome.*

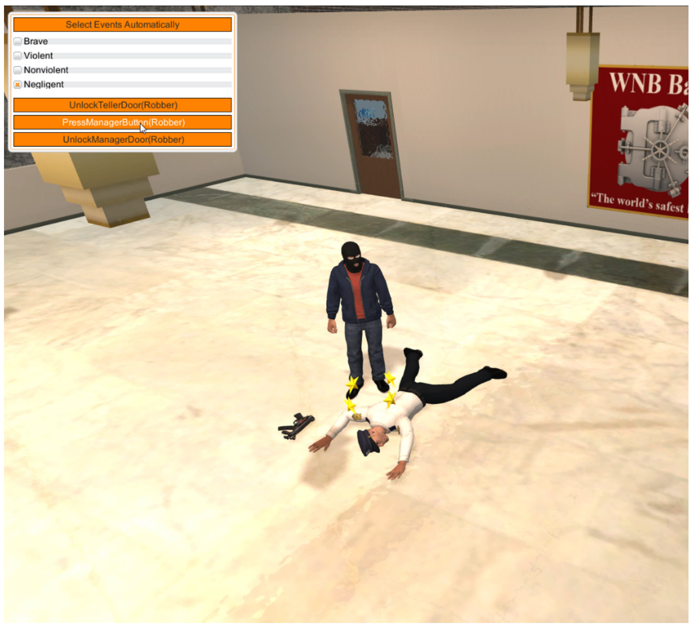

Figure 8.5: *A screenshot of the virtual world and Storycraft interface overlay.*

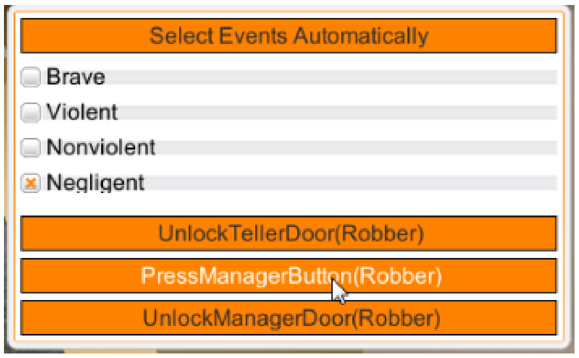
Select Events Automatically
Brave
Violent
Nonviolent
Negligent
UnlockTellerDoor(Robber)
PressManagerButton(Robber)
UnlockManagerDoor(Robber)

Figure 8.6: *Close-up view of the storytelling dialog from Figure 8.5 showing sentiment and event selection.*

Chapter 9

Conclusions and Discussion

This hierarchy of systems – ADAPT, PAStE, and CANVAS/Storycraft – provides a powerful, accessible, and extensible framework for creating and sharing stories taking place in rich virtual worlds full of active and dynamic characters and objects. We solve the myriad problems across the entire pipeline, from taking even the most vague story specifications and translating them through fully defined narratives down to the individual frame-by-frame joint angles and actuations of each individual virtual actor. Using this system, untrained authors can create narratives using a wide variety of different modalities, from a strictly specified, fully-authored set of events, to a partially specified story high concept, and even in an on-the-fly exploratory process requiring no prior designs or expectations. This allows us to fully explore the ramifications of what a virtual world can be used for, and how we can harness these rich digital spaces for meaningful communication of ideas.

One quality of this system worth emphasizing here is its focus on expansion and adaptability. The underlying animation platform, ADAPT, is purpose-built for adding new capabilities to the system to improve the physical repertoire of the characters and objects in the world. The PAStE system is designed to allow domain experts to easily expand the ways in which they describe their characters and their relationships with each other and the world. CANVAS and Storycraft hinge entirely on the ability to create new and interesting events for characters to exhibit intricate coordinated activity in interacting both with each other and with their environment. Though for brevity we focus only on a bank robbery scenario, this system or aspects of it have already been used to simulate city parks, a virtual middle eastern marketplace, and adversarial situations like an interactive "prison break" environment [74].

The core advantage of this system is the single architectural element around which every aspect revolves – the event. This simple notion of collecting multi-object behavior into a library of centralized, disposable control-flow data structures enables a number of otherwise impossible tasks:

1. Complex coordination of difficult actions between large groups of individuals is trivially reduced to a set of timed command dispatches. Rather than the traditional model of sensory perception and message passing between monolithic agents, events enable a simple and lightweight method to temporarily augment actors with all the behavior they need for a particular interaction on the fly.

2. Events present a new way of describing the world and what happens within it. Where a traditional simulation or planning model would focus on atomic actions within the repertoire of the characters (sit down, stand up, open the door), events serve as atoms of *narrative significance* with functional purpose for furthering the story (have a conversation, incapacitate a guard, break an alliance). Events exist at the ideal level of action-space granularity to provide accessible metaphors to untrained authors charged with deciding what should happen in a story.

3. The parametric structure of events (in that actor and object participants are filled in to an event's role slots) provides a simple and effective way for authors to intentionally under-specify stories without harming the system's ability to resolve them.

> Events allow the author to focus on *what* happens in their story, while allowing the system to decide *who* does it. In a traditional agent model, this would be far more difficult to communicate to an author, if not impossible. And yet, it is a powerful tool for planning flexibility both in rapid story prototyping, and in accommodating interactive narrative where plans can be invalidated and require a new solution.

Designing the system around events fundamentally changes the way characters and objects are controlled from more traditional models, and, as we have demonstrated, opens up new opportunities for creating compelling and purposeful stories.

9.1 Future Directions

Real-time User Participants. The single most important avenue of expansion for this system would be continued work exploring how a user participant can interact with the story once an author has created it. Though we did not focus on this aspect in this project, previous work [74] demonstrates a proof of concept where a user can interact with a real-time event planner to change the trajectory of a generated story. Our ideal scenario would be a multi-user real-time Storycraft environment, where a Storycraft author dynamically dispatches events for the characters and objects in the world, and one or more real-time user participants take part in those events and alter their outcome. We believe this collaborative storytelling experience could be a powerful tool in education, training, and entertainment.

Accomplishing this would be more of a logistical challenge (developing a networked multi-user synchronized story client) than a research problem. The real-time Storycraft scenario lends itself well to real-time story participant interaction. Where currently the human author of a Storycraft scenario selects events and simply waits for their execution, a multi-user Storycraft scenario would consist more of the scenario author picking events in *response* to the actions of the user participant. As the human participant interacts with the world, he or she moves the story state to different nodes in the story web, which alter the suggestions presented by Storycraft to the scenario controller. The main conceptual challenge would be that the participant's action space would be that of affordances, while the scenario author's would be that of events. In terms of research problems we would need to explore how to expand the story web exploration process to allow the human participant to activate arbitrary affordances on objects in the world through their avatar without moving the world state out of the precomputed story web.

Missing Events. Currently, the system is dependent on a pre-designed lexicon of existing events. CANVAS and Storycraft are both incapable of reaching certain world states if the required event transitions do not exist. While automatically generating narratively-contextualized world state transitions on the fly is well beyond the scope of our system, it is conceivable for the system to suggest areas of the world state space that are difficult to reach with the current lexicon of events. This would help guide domain experts towards producing a broad and balanced repertoire of character actions. Additionally, this kind of analysis could better inform situations where story planning fails if the system is able to identify areas where hypothetical event additions could help.

Exploration and Optimization. General exploration of the Storycraft framework would also greatly benefit this system. ADAPT, PAStE, and CANVAS are mature, fully-developed frameworks while Storycraft exists in a more prototypical state. More efficient algorithms for story web generation and exploration would allow Storycraft to accommodate bigger stories in richer worlds with more events, actors, and objects to involve. GPU-based parallel processing has been incredibly promising in this regard, but some work remains to be done in this endeavor. Additionally, a multi-author Storycraft system where different authors are responsible for different parts of the populace would additionally mitigate some accessibility and complexity issues.

Evaluation and Iteration. Finally, these systems rely on iteration based on use-case experience to reach their full potential. CANVAS and Storycraft are largely unique in their author-centric purpose and approach, making them difficult to evaluate or compare against any peer systems. However, evaluating users' experience authoring real-world narrative scenarios (outside of a purely academic environment) for education and training would be incredibly insightful and allow for significant improvements in both the way end-users interact with the system, and domain experts expand its capabilities. The complete system has shown promise in our use of it, would benefit greatly from real-world application and feedback.

Bibliography

[1] Tolga Abaci and Daniel Thalmann. Planning with smart objects. In *In The 13th International Conference in Central Europe on Computer Graphics, Visualization and Computer Vision: WSCG*, pages 25–28, 2005.

[2] Okan Arikan and D. A. Forsyth. Interactive motion generation from examples. *ACM TOG*, 21(3):483–490, July 2002.

[3] Autodesk, Inc. Autodesk gameware - artificial intelligence middleware for games, 2012.

[4] Paolo Baerlocher and Ronan Boulic. An inverse kinematics architecture enforcing an arbitrary number of strict priority levels. *Vis. Comput.*, 20(6):402–417, August 2004.

[5] Mira Dontcheva, Gary Yngve, and Zoran Popović. Layered acting for character animation. *ACM Trans. Graph.*, 22(3):409–416, July 2003.

[6] Kutluhan Erol, James Hendler, and Dana S. Nau. Htn planning: Complexity and expressivity. In *Proceedings of AAAI*, pages 1123–1128. AAAI Press, 1994.

[7] Ugo Erra, Bernardino Frola, and Vittorio Scarano. BehaveRT: a GPU-based library for autonomous characters. In *Motion in Games*, MIG'10, pages 194–205, 2010.

[8] Petros Faloutsos, Michiel van de Panne, and Demetri Terzopoulos. Composable controllers for physics-based character animation. ACM SIGGRAPH, pages 251–260, 2001.

[9] Andrew W. Feng, Yazhou Huang, Marcelo Kallmann, and Ari Shapiro. An analysis of motion blending techniques. In *MIG*, 2012.

[10] Andrew W. Feng, Yuyu Xu, and Ari Shapiro. An example-based motion synthesis technique for locomotion and object manipulation. I3D, pages 95–102, 2012.

[11] Richard E. Fikes and Nils J. Nilsson. Strips: A new approach to the application of theorem proving to problem solving. *Artificial Intelligence*, 1971.

[12] Michael Fleischman and Deb Roy. Representing intentions in a cognitive model of language acquisition: Effects of phrase structure on situated verb learning. In *AAAI '07*, pages 7–12.

[13] John Funge, Xiaoyuan Tu, and Demetri Terzopoulos. *Cognitive Modeling: Knowledge, Reasoning and Planning for Intelligent Characters*, pages 29–38. ACM Press/Addison-Wesley Publishing Co., 1999.

[14] Gavalakis Vaggelis. Nodecanvas, 2014. http://nodecanvas.com/.

[15] Sanjeet Hajarnis, Christina Leber, Hua Ai, Mark Riedl, and Ashwin Ram. A case base planning approach for dialogue generation in digital movie design. In Ashwin Ram and Nirmalie Wiratunga, editors, *Case-Based Reasoning Research and Development*, volume 6880 of *Lecture Notes in Computer Science*, pages 452–466. Springer Berlin Heidelberg, 2011.

[16] Peter E. Hart, Nils J. Nilsson, and Bertram Raphael. Correction to ”a formal basis for the heuristic determination of minimum cost paths”. *SIGART Bull.*, (37):28–29, 1972.

[17] Dirk Helbing and Peter Molnar. Social force model for pedestrian dynamics. *PHYSICAL REVIEW E*, 51:42–82, 1995.

[18] Yazhou Huang, Mentar Mahmudi, and Marcelo Kallmann. Planning humanlike actions in blending spaces. In *Intelligent Robots and Systems (IROS)*, 2011.

[19] Bohemia Interactive. Arma 3 dlc “zeus”, 2014. http://arma3.com/dlc/zeus.

[20] Arnav Jhala, Curtis Rawls, Samuel Munilla, and R. Michael Young. Longboard: A sketch based intelligent storyboarding tool for creating machinima. In *FLAIRS Conference*, pages 386–390. AAAI Press, 2008.

[21] Rune Skovbo Johansen. Automated semi-procedural animation for character locomotion. Master’s thesis, Aarhus University, 2009.

[22] Marcelo Kallmann. Shortest paths with arbitrary clearance from navigation meshes. In *Eurographics/SIGGRAPH SCA*, 2010.

[23] Marcelo Kallmann and Stacy Marsella. Lncs ’05. chapter Hierarchical motion controllers for real-time autonomous virtual humans, pages 253–265. 2005.

[24] Marcelo Kallmann and Daniel Thalmann. Direct 3d interaction with smart objects. In *ACM symposium on Virtual reality software and technology*, VRST ’99, pages 124–130, 1999.

[25] M. Kapadia, S. Singh, G. Reinman, and P. Faloutsos. A Behavior-Authoring Framework for Multiactor Simulations. *IEEE CGA*, 31(6):45 –55, 2011.

[26] Mubbasir Kapadia and Norman I. Badler. Navigation and steering for autonomous virtual humans. *Wiley Interdisciplinary Reviews: Cognitive Science*, 2013.

[27] Mubbasir Kapadia, Nathan Marshak, and Norman I. Badler. Adapt: The agent development and prototyping testbed. *IEEE Transactions on Visualization and Computer Graphics*, 99(PrePrints):1, 2014.

[28] Mubbasir Kapadia, Alexander Shoulson, Cory D. Boatright, Pengfei Huang, Funda Durupinar, and Norman I. Badler. What's next? the new era of autonomous virtual humans. In *MIG*, pages 170–181, 2012.

[29] Mubbasir Kapadia, Alexander Shoulson, Cyril Steimer, Samuel Oberholzer, and Robert Sumner. CANVAS: Computer-Assisted Narrative Animation Synthesis. *Eurographics*, 2015 (Under Review).

[30] Mubbasir Kapadia, Shawn Singh, William Hewlett, and Petros Faloutsos. Egocentric Affordance Fields in Pedestrian Steering. In *Interactive 3D graphics and games*, I3D '09, pages 215–223. ACM, 2009.

[31] Mubbasir Kapadia, Shawn Singh, William Hewlett, Glenn Reinman, and Petros Faloutsos. Parallelized egocentric fields for autonomous navigation. *The Visual Computer*, pages 1–19.

[32] Bilal Kartal, John Koenig, and Stephen J. Guy. User-driven narrative variation in large story domains using monte carlo tree search. In *Proceedings of the 2014 International Conference on Autonomous Agents and Multi-agent Systems*, AAMAS '14, pages 69–76, Richland, SC, 2014. International Foundation for Autonomous Agents and Multiagent Systems.

[33] L.E. Kavraki, P. Svestka, J.-C. Latombe, and M.H. Overmars. Probabilistic roadmaps for path planning in high-dimensional configuration spaces. *IEEE-RAS*, 12:566 – 580, 1996.

[34] Caitlin Kelleher, Randy Pausch, and Sara Kiesler. Storytelling alice motivates middle school girls to learn computer programming. In *Proceedings of the SIGCHI Conference on Human Factors in Computing Systems*, CHI '07, pages 1455–1464, New York, NY, USA, 2007. ACM.

[35] Lucas Kovar, Michael Gleicher, and Frédéric Pighin. Motion graphs. SIGGRAPH, pages 473–482, 2002.

[36] Jehee Lee, Jinxiang Chai, Paul S. A. Reitsma, Jessica K. Hodgins, and Nancy S. Pollard. Interactive control of avatars animated with human motion data. *ACM TOG*, 21(3):491–500, 2002.

[37] Kang Hoon Lee, Myung Geol Choi, Qyoun Hong, and Jehee Lee. Group behavior from video: a data-driven approach to crowd simulation. In *ACM SIGGRAPH/Eurographics SCA*, pages 109–118, 2007.

[38] Alon Lerner, Yiorgos Chrysanthou, and Dani Lischinski. Crowds by example. *CGF*, 26(3):655–664, 2007.

[39] Boyang Li, Stephen Lee-Urban, George Johnston, and Mark Riedl. Story generation with crowdsourced plot graphs. In Marie desJardins and Michael L. Littman, editors, *AAAI*. AAAI Press, 2013.

[40] Boyang Li and Mark O. Riedl. Creating customized game experiences by leveraging human creative effort: A planning approach. In *Agents for Games and Simulations II*, pages 99–116. Springer, 2010.

[41] Weizi Li and Jan M. Allbeck. The virtual apprentice. In *IVA*, pages 15–27, 2012.

[42] Ying Liu and Norman I. Badler. Real-time reach planning for animated characters using hardware acceleration. CASA, pages 86–93, 2003.

[43] A. Bryan Loyall. *Believable Agents: Building Interactive Personalities*. PhD thesis, Carnegie Mellon University, 1997.

[44] Brian Magerko, John E Laird, Mazin Assanie, Alex Kerfoot, and Devvan Stokes. AI Characters and Directors for Interactive Computer Games. *Artificial Intelligence*, 1001:877–883, 2004.

[45] Massive Software Inc. Massive: Simulating life, 2010. www.massivesofware.com.

[46] Michael Mateas and Andrew Stern. 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. 2003.

[47] Michael Mateas and Andrew Stern. A behavior language: Joint action and behavioral idioms. In *Life-Like Characters*, pages 135–161. Springer, 2004.

[48] D. Mcdermott, M. Ghallab, A. Howe, C. Knoblock, A. Ram, M. Veloso, D. Weld, and D. Wilkins. Pddl - the planning domain definition language. Technical Report TR-98-003, Yale Center for Computational Vision and Control,, 1998.

[49] Stephane Menardais, Franck Multon, Richard Kulpa, and Bruno Arnaldi. Motion blending for real-time animation while accounting for the environment. CGI, pages 156–159, 2004.

[50] Jianyuan Min and Jinxiang Chai. Motion graphs++: a compact generative model for semantic motion analysis and synthesis. *ACM Trans. Graph.*, 31(6):153:1–153:12, November 2012.

[51] Howard Lee Mohn. *Implementation of a Tactical Mission Planner for Command and Control of Computer Generated Forces in ModSAF.* PhD thesis, Naval Postgraduate School, 1994.

[52] Mikko Mononen. Recast/Detour navigation library, 2009.

[53] Jeff Orkin. Agent architecture considerations for real-time planning in games. In *Interactive Digital Entertainment Conference*, pages 105–110. AAAI Press, 2005.

[54] Lawrence Page, Sergey Brin, Rajeev Motwani, and Terry Winograd. The pagerank citation ranking: Bringing order to the web. Technical Report 1999-66, November 1999.

[55] Sbastien Paris, Julien Pettr, and Stphane Donikian. Pedestrian reactive navigation for crowd simulation: a predictive approach. *Computer Graphics Forum*, 26(3):665–674, 2007.

[56] Judea Pearl. *Heuristics: Intelligent Search Strategies for Computer Problem Solving*. Addison-Wesley Longman Publishing Co., Inc., Boston, MA, USA, 1984.

[57] N. Pelechano, J. M. Allbeck, and N. I. Badler. Controlling individual agents in high-density crowd simulation. In *ACM SIGGRAPH/Eurographics SCA*, pages 99–108, 2007.

[58] Nuria Pelechano, Jan M. Allbeck, and Norman I. Badler. *Virtual Crowds: Methods, Simulation, and Control.* Synthesis Lectures on Computer Graphics and Animation. Morgan & Claypool Publishers, 2008.

[59] Ken Perlin and Athomas Goldberg. Improv: a system for scripting interactive actors in virtual worlds. SIGGRAPH, pages 205–216, 1996.

[60] Julien Pettré, Marcelo Kallmann, and Ming C. Lin. Motion planning and autonomy for virtual humans. In *ACM SIGGRAPH classes*, pages 42:1–42:31, 2008.

[61] Julie Porteous, Marc Cavazza, and Fred Charles. Applying planning to interactive storytelling: Narrative control using state constraints. *ACM Trans. Intell. Syst. Technol.*, 1(2):10:1–10:21, December 2010.

[62] Craig Reynolds. Steering behaviors for autonomous characters, 1999.

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

[64] Mark O. Riedl and Vadim Bulitko. Interactive narrative: An intelligent systems approach. *AI Magazine*, 34(1):67–77, 2013.

[65] Mark O Riedl, Andrew Stern, Don Dini, and Jason Alderman. 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, 2008.

[66] Rodolfo Rosini. Storybricks. Namaste Entertainment Inc., 2014.

[67] Matthew Schuerman, Shawn Singh, Mubbasir Kapadia, and Petros Faloutsos. Situation agents: agent-based externalized steering logic. *Comput. Animat. Virtual Worlds*, 21:267–276, May 2010.

[68] Wei Shao and Demetri Terzopoulos. Autonomous pedestrians. *Graph. Models*, 69:246–274, September 2007.

[69] Ari Shapiro. Building a character animation system. MIG, pages 98–109, 2011.

[70] Ari Shapiro, Marcelo Kallmann, and Petros Faloutsos. Interactive motion correction and object manipulation. In *Interactive 3D graphics and games*, I3D '07, pages 137–144. ACM, 2007.

[71] Alexander Shoulson and Norman I. Badler. Event-centric control for background agents. In *International conference on Interactive Digital Storytelling*, ICIDS, pages 193–198, 2011.

[72] Alexander Shoulson, Daniel Garcia, and Norman Badler. Selecting agents for narrative roles. In *INT4*. 2011.

[73] Alexander Shoulson, Francisco Garcia, Matthew Jones, Robert Mead, and Norman I. Badler. Parameterizing Behavior Trees. In *Motion in Games*, pages 144–155. Springer, 2011.

[74] Alexander Shoulson, Max L. Gilbert, Mubbasir Kapadia, and Norman I. Badler. An event-centric planning approach for dynamic real-time narrative. In *Proceedings of Motion on Games*, MIG '13, pages 99:121–99:130, New York, NY, USA, 2013. ACM.

[75] Mei Si, Stacy C. Marsella, and David V. Pynadath. Thespian: An architecture for interactive pedagogical drama. In *Proceeding of the 2005 Conference on Artificial Intelligence in Education*, pages 595–602. 2005.

[76] Shawn Singh, Mubbasir Kapadia, Petros Faloutsos, and Glenn Reinman. An open framework for developing, evaluating, and sharing steering algorithms. In *MIG*, pages 158–169, 2009.

[77] Shawn Singh, Mubbasir Kapadia, Billy Hewlett, Glenn Reinman, and Petros Faloutsos. A modular framework for adaptive agent-based steering. In *ACM SIGGRAPH I3D*, pages 141–150, 2011.

[78] Shawn Singh, Mubbasir Kapadia, Glenn Reinman, and Petros Faloutsos. Footstep navigation for dynamic crowds. *Computer Animation and Virtual Worlds*, 22(2-3):151–158, 2011.

[79] James Skorupski and Michael Mateas. Novice-friendly authoring of plan-based interactive storyboards. In *Proceedings of the Sixth AAAI Conference on Artificial Intelligence and Interactive Digital Entertainment, AIIDE 2010, October 11-13, 2010, Stanford, California, USA*, 2010.

[80] Damian Slonneger, Matthew Croop, Jeremy Cytryn, Joseph T. Kider Jr., Richard Rabbitz, Eric Halpern, and Norman I. Badler. Human model reaching, grasping, looking and sitting using smart objects international symposium on digital human modeling. Proc. International Ergonomic Association Digital Human Modeling, 2011.

[81] Catherine Stocker, Libo Sun, Pengfei Huang, Wenhu Qin, Jan M. Allbeck, and Norman I. Badler. Smart events and primed agents. In *Proceedings of the 10th international conference on Intelligent virtual agents*, IVA'10, pages 15–27, Berlin, Heidelberg, 2010. Springer-Verlag.

[82] Mechthild Stoer and Frank Wagner. A simple min-cut algorithm. *J. ACM*, 44(4):585–591, July 1997.

[83] U.S. Army PEO STRI. Product manager one semi-automated forces (pdm onesaf), 2014. http://www.peostri.army.mil/PRODUCTS/ONESAF/.

[84] Libo Sun, Alexander Shoulson, Pengfei Huang, Nicole Nelson, Wenhu Qin, Ani Nenkova, and Norman I. Badler. Animating synthetic dyadic conversations with variations based on context and agent attributes. *Comput. Animat. Virtual Worlds*, 23(1):17–32, February 2012.

[85] Ben Sunshine-Hill and Norman I. Badler. Perceptually realistic behavior through alibi generation. In *AIIDE*. The AAAI Press, 2010.

[86] Daniel Thalmann and Soraia Raupp Musse. *Crowd Simulation, Second Edition*. Springer, 2013.

[87] David Thue, Vadim Bulitko, Marcia Spetch, and Eric Wasylishen. Interactive storytelling: A player modelling approach. In *AIIDE*, 2007.

[88] Adrien Treuille, Seth Cooper, and Zoran Popović. Continuum crowds. In *ACM SIGGRAPH*, SIGGRAPH '06, pages 1160–1168, 2006.

[89] Jur van den Berg, Ming C. Lin, and Dinesh Manocha. Reciprocal velocity obstacles for real-time multi-agent navigation. In *ICRA*, pages 1928–1935. IEEE, 2008.

[90] C.W. Warren. Global path planning using artificial potential fields. In *IEEE-RAS*, pages 316–321 vol.1, 1989.

[91] Peter William Weyhrauch. *Guiding interactive drama*. PhD thesis, Carnegie Mellon University, Pittsburgh, PA, USA, 1997.

[92] K. N. Whitley and Alan F. Blackwell. Visual programming: The outlook from academia and industry. In *Papers Presented at the Seventh Workshop on Empirical Studies of Programmers*, ESP '97, pages 180–208. ACM, 1997.

[93] Andrew Witkin and Zoran Popovic. Motion warping. SIGGRAPH, pages 105–108, 1995.

[94] KangKang Yin, Kevin Loken, and Michiel van de Panne. Simbicon: simple biped locomotion control. *ACM TOG*, 26(3), 2007.

[95] Qinxin Yu and Demetri Terzopoulos. A decision network framework for the behavioral animation of virtual humans. SCA, pages 119–128, 2007.