Proceedings of the Fourteenth Artificial Intelligence and
Interactive Digital Entertainment Conference (AIIDE 2018)

Playable Experiences at AIIDE 2018

Ben Samuel
University of New Orleans
bsamuel@cs.uno.edu

Aaron Reed, Emily Short
"Escape Plan" and "At the Bar":
Dynamic Characters Driven by Spirit AI Character Engine
Spirit AI
{aaron, emily}@spiritai.com

Samantha Heck, Barrie Robison, Landon Wright, Terence Soule
Project Hastur: An Evolutionary Tower Defense Game
University of Idaho
{heck9873, wrig8396}@vandals.uidaho.edu, {brobison, tsoule}@uidaho.edu

Mike Treanor, Joshua McCoy, Anne Sullivan
Vox Populi: The Ustradian Games
American University, University of California, Davis, Georgia Institute of Technology
treanor@american.edu, jamccoy@ucdavis.edu, anne@play-crafts.com

Alireza Shirvani, Edward Garcia, Rachelyn Farrell, Stephen Ware
Camelot: An Interactive Narrative Sandbox Environment
University of New Orleans
{ashirvan, etgarci1, rfarrell, sgware}@uno.edu

Katherine Compton
Bottery
University of California, Santa Cruz
galaxykate@gmail.com

Abstract

This paper describes the accepted entries to the sixth Playable Experiences track to be held at the AIIDE conference. The Playable Experiences track showcases innovative complete works that are informed, inspired, or otherwise enabled by artificial intelligence.

Introduction

The AIIDE Playable Experiences track offers creators a platform to showcase work that is informed, inspired, or otherwise enabled by artificial intelligence. While other tracks at the conference, including the main track, the demo track, and artifact-evaluation, provide opportunities for creators to supplement their research contributions with interactive demonstrations, accepted entries in the Playable Experiences track are complete experiences in and of themselves. As such, though the pieces presented here are diverse in their approaches and applications of artificial intelligence, they all share two commonalities: they are vehicles for both ground-breaking artificial intelligence research and artistic achievement.

Five entries were accepted to the 2018 AIIDE Playable Experiences track (chaired by Ben Samuel). These pieces demonstrate a range of technological innovations and artificial intelligence techniques, applied towards a variety of genres both existing and novel. The accepted entries are:

- **"Escape Plan" and "At the Bar": Dynamic Characters Driven by Spirit AI Character Engine** is a linked pair of playable experiences, representing two conversations with AI-driven characters in a shared fictional universe.

- **Project Hastur: An Evolutionary Tower Defense Game** features procedurally generated enemies that evolve based on the automated defenses erected by the player.
- **Vox Populi: The Ustradian Games** is a game-based assessment of cross-cultural competency and metacognition, that integrates complex simulation technologies with gameplay.
- **Camelot: An Interactive Narrative Sandbox Environment** is a modular and customizable interactive narrative environment, which provides a sandbox to visualize and test different narrative systems.
- **Bottery** is a combination of a simulator, language, parser, and IDE for finite state machine chatbots.

The rest of this paper consists of each of the creators discussing their works, influences, and approaches towards applying artificial intelligence to create novel playable experiences.

Accepted Submissions

"Escape Plan" and "At the Bar": Dynamic Characters Driven by Spirit AI Character Engine

We present a linked pair of playable experiences, "Escape Plan" and "At the Bar," representing two conversations with AI-driven characters in a shared fictional universe. The experiences are driven by Spirit AI Character Engine, a new commercial platform for developing dynamic character interaction. "Escape Plan" uses free text natural language input, while "At the Bar" uses a menu-based input system with contextually-selected choices. Both experiences use Character Engine to determine likely player actions, reason over character reactions, and generate a procedurally-assembled response. A diagnostic display shows the logic behind the procedural text system in real-time. Third-party facial animation, lip-sync, and text-to-speech technology are used to round out the demos.

These demos have been created by Character Engine team members to showcase the engine's capabilities. Our team includes several people in key technical and creative positions with a deep background of building interactive stories and characters within academia and as working game designers. Our motivation for creating this platform is to assemble (in a single, authorable package) a range of technologies that together make more dynamic game characters realizable for commercial game companies: machine-learning trained natural language classification, human-level text-to-speech and speech-to-text, and years of design wisdom about procedural text generation and social simulation. Our system designs have been driven by our practical experiences designing previous games and engines concerned with dynamic and authorable interactive stories [Short, 2011; Plotkin, 2011; McCoy *et al.*, 2014; Evans and Short, 2014; Reed, 2014] and we aim to both iterate and improve on these prior systems, and to make this design space accessible to mainstream game designers.

The primary innovation these demos showcase is an end-to-end system for authoring richly interactive game characters. Using an authoring tool and SDK, designers can build reactive characters with capabilities not yet seen in AAA games, including responding to natural language, expressing multiple aspects of game state in a single procedurally assembled response, and shaping a conversation to meet design goals. For instance, in "Escape Plan," the NPC follows an author-defined spine where she attempts to first decide whether the player is trustworthy, and if so, find out whether they're willing to perform a mission for her, all the while allowing the player to improvise around this spine by asking follow-up questions and bringing up other topics. At a high level, the NPC responds naturally and then redirects the conversation back to the authorial goals, an approach enabled by a system of social practices that drive which behaviors the NPC wants to enact. At a low-level, the particulars of each NPC response are shaped by variable character traits to produce transcripts that differ significantly from one playthrough to the next. In “At the Bar,” the NPC’s personality is randomly reset each playthrough, leading to different levels of politeness, extroversion, and other traits visible in the top center of the interface.

Video clips of the demos can be seen at the link below. The first two clips show two different playthroughs of “Escape Plan,” highlighting different player strategies. The third clip showcases “At the Bar.”

http://aaronareed.net/aiide-playable-18/

Project Hastur: An Evolutionary Tower Defense Game

Project Hastur is a tower defense game that features evolving, procedurally generated enemies. These enemies, called the Protean Swarm, are defined by a digital genome of real numbers that defines all of their game traits. The Proteans’ traits evolve over time, allowing the Proteans to adapt to the player’s strategy and the environment. The traits include morphology (size, limb types, color, texture, and presence of swappable parts), behavior (avoidance and preference for humans and towers), and performance (resistances, sensory abilities, health, and movement speeds). Variation in the enemy traits are created by mutations of the genome and crossover between parent creatures. The trait expression is fully integrated into the Unity game engine, allowing the morphology, behavior, and performance

of each creature to be dynamically generated during runtime.

Figure 1: Screenshot from Project Hastur showing some of the possible variety of evolving opponents.

In a typical tower defense game, the enemies are encoded as scripted waves. In Project Hastur, each wave is a discrete generation that is the result of sexual reproduction of the preceding generation. The Proteans are encoded as diploid, sexually reproducing hermaphrodites. During a generation, an individual Proteans's genetically encoded traits work in combination to help it earn fitness by either damaging the player's defensive towers or approaching and damaging the player's base tower. At the end of each generation, Proteans are selected to be parents based on these fitness values. Parents are chosen in pairs using tournament selection and each parent contributes one mutated chromosome to create a novel, diploid offspring. This process is repeated until the fixed population size is reached. As such, a small sub-set of the most fit Proteans win parenthood and pass their advantageous traits to their offspring. We call this game mechanic evolutionary procedural generation (EPG). In Project Hastur, EPG produces game outcomes that are responsive to differences in player strategy – the enemies literally adapt to the player.

Figure 2:Screenshot from Project Hastur showing the parameters available for experiment mode.

The potential applications of Evolutionary Procedural Generation of game enemies are numerous. The most obvious application is the customization of player experience without expensive and time consuming generation of game content. For example, rather than carefully scripting waves of enemies, EPG automatically produces generations of enemies that become progressively more difficult. More importantly, the increasing difficulty occurs in response to player decisions. If the player focuses on building flamethrowers the Proteans are very likely to evolve fire resistance. This potentially increases replayability because every time the player tries a new strategy they are likely to see a different opponent response. Additionally, even when the player makes the exact same strategic decisions, random mutations can produce different but equally effective enemy populations that will ultimately crush the player.

Project Hastur also includes an experiment mode (Figure 2) that allows the player to control the parameters of evolution: mutation rates, selection pressure, population sizes, etc. The complete genome of every Protean in every generation can be saved to an external file for analysis, and there are tools for automating gameplay that allow for the easy collection of data. Thus, Project Hastur can also be used as a teaching and research tool, allowing players to explore the evolutionary processes and perform experiments.

A playable demo of Project Hastur can be found at: http://polymorphicgames.com/DEMO

Vox Populi: The Ustradian Games

Vox Populi: The Ustradian Games is a game-based assessment of cross-cultural competency and meta-cognition, that integrates complex simulation technologies with gameplay. In the game, players seek to achieve objectives through having conversations with the people of an unknown artificial culture. The player's dialogue options and the non-player character responses are determined by Ensemble [Samuel *et al.,* 2015] and the Social Practice engine [Treanor *et al.*, 2016]. The game roughly conforms to the "Visual Novel" game genre, but rather than simply navigating pre-scripted dialogue trees, the responses are dynamically selected based on the social state and history of player choices. Because the results of conversations are based on a rich model of the non-player character's culture, we believe that players will need to focus on gaining an understanding of the culture, rather than memorizing, or enumerating all paths through dialogue trees.

The story of the game is about the Ustradian people, and a tournament that they hold to determine the new "cultural minister" of their planet, Ustrad. The winner of the tournament is the person who is able to complete three quests, and then correctly demonstrate their understanding of the culture through a quiz. Example quests involve learning about the occupation of, or getting an item that belongs to,

the Ustradians. In order to successfully achieve these goals, the player must learn about the society's hierarchical structures, and the collectivistic or individualistic nature each individual (among other cultural factors). Part of the game involves players inputting their current understanding about the culture into a game interface. Through empirical studies with two different versions of the game, where one had this interface and the other didn't, it was found that presenting the game interface resulted in better performance at the end of game quiz (thus a better understanding of the simulated artificial culture). We believe that this demonstrates that interrelating the gameplay and simulation elements can result in deeper engagement with experimental simulation games, and can point the way for new genres of games that promote system level understandings of complex subjects.

As mentioned, central to Vox Populi is the concept of a "social practice". In the context of the Social Practice engine, a social practice is a normative pattern of social interaction that captures the nuances that result from the individual agent's situation in the social state. In Ensemble, the particular path through an instance of a social practice is not strongly pre-authored; instead the path is generated. Rather than adhering to a static branching tree structure, or a state machine with transition rules, a sophisticated selection mechanism (a modified version of Ensemble) is used to determine an agent's response to the previous action and current social state. This mechanism is driven both by the structure of the practice as well as a model of socio-cultural norms of the story world.

Vox Populi is the first completed game to make use of the Social Practice engine. Building from this work, the Social Practice engine and Ensemble are being used for an in-development role-playing game where social history and a dynamic social state will shape the game's narrative.

Camelot: An Interactive Narrative Sandbox Environment

Purpose

Camelot is a modular and customizable interactive narrative environment, which provides a sandbox to visualize and test different narrative systems. This environment is a real-time 3D third-person over-the-shoulder game that takes place in a medieval fantasy setting and includes customizable components, including characters, places, and items, implemented in Unity3D engine.

Architecture

Camelot can be considered the presentation layer to a narrative system that serves as the logic layer. These two layers can communicate through standard input messages to create and customize a virtual world, as well as the interactions of its virtual characters, and allow a "player" to interact with it.

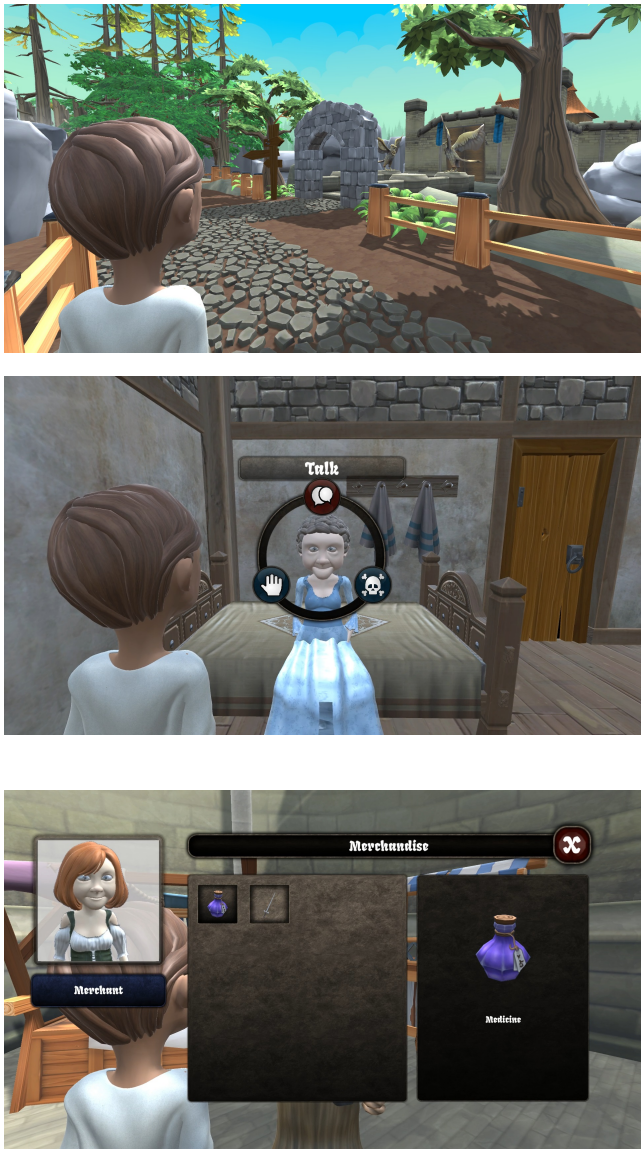

Figure 3: Camelot presentation layer.

More specifically, *Camelot* listens for messages in the "start some task" format and in turn, performs that task if possible, and sends back messages about the life cycle of that task, e.g. started, succeeded, or failed. Moreover, it also processes player input to move the player character and sends appropriate input messages as the player interacts with the world, for instance "input Player arrived at a position," "input selected an option," etc. The logic layer can use these messages to properly respond to player input and execute arbitrary sequences of tasks on the presentation layer.

The presentation layer has several main components. Characters can be created using the "Create Character" command with the options to set the gender, age, hair style,

skin color, eye color, and clothing. Places represent different architectural and natural locations with different gateways, e.g. doors, gates, paths, etc. Each place has a default spawn position, as well as different positions, which can be used to put the characters or trigger events, when a character enters or leaves them. Finally, as their name suggests, items are various accessories, which can be held by characters or be placed in their inventory. Examples of items include swords, coins, potions, etc.

Another key task type that allows the player character to interact with different world components is *EnableIcon/DisableIcon*. These two tasks respectively enable/disable an interaction option on another character, door, or object in the world. The corresponding component is highlighted when hovered by the mouse pointer if it has at least one enabled option. The player can interact with different objects by either clicking on them to choose the default enabled option or right click to open a radial menu showing all currently enabled options. Upon selecting an option, a message is sent to the logic layer to execute the corresponding task sequence.

Example

The logic layer packaged with this playable experience is a plotgraph experience manager, which uses an intelligently pruned plotgraph to enable/disable player actions and order NPCs to take actions at each state of the graph. The experience manager uses a domain with 4 locations, 4 characters, and 7 actions. The actions include walking from a place to another, taking items out of the bandit's chest, buying items from the merchant for one coin each, stealing an item from an unarmed character, attacking and killing another character, and looting items from slain characters. Finally, a character who knows the bandit's location can report him to the town guard. The experience manager maps different templates of these actions to w sequences of tasks on the presentation layer.

In our experiment, the player character begins in their house, where they learn their grandmother is sick. She gives them a coin that can be used to buy medicine. The game features three NPCs, a merchant trading goods for coins, the town guard who attacks criminals, and a bandit waiting to steal items. The places include the house, the cottage, the market, the camp, and the crossroads which connects the last three places. Each character has a sword (except the player), the merchant also has the medicine, and there is another coin in the bandit's chest in the camp. The game ends when the player either dies or returns home carrying the medicine. Despite its small size and simplicity, this domain yields a surprising number of interesting ways the player can accomplish their goal or die in the attempt. We asked 34 participants to play two versions of the game, one with a randomly generated and the other with the intelligently pruned plotgraph. The results indicated that when players noticed a difference between the two versions, they significantly preferred the intelligent graph. Otherwise, the participants playing the random version first significantly preferred the intelligent one.

In conclusion, we hope other researchers could utilize this system to visualize their experiments and evaluate their story generation systems.

Acknowledgements

This research was supported by NSF awards IIS-1647427 and IIS-1464127.

Bottery

Bottery is a combination of a simulator, language, parser, and IDE for finite state machine (FSM) chatbots. It was developed at Google from April 2016 to February 2017, then open-sourced in October 2017.

Bottery is based on lessons from my previous language-engine-and-IDE library, Tracery [Compton *et al*., 2015]. Tracery runs at least 11,000 active Twitterbots, and created a Cambrian explosion of new bot forms and unexpected bot poetics [Compton, 2017]. It succeeded (rather accidentally) by combining a novice-friendly online editor with an on-page simulator and a programming language whose programs could be expressed as a JSON file (and safely run on a 3rd party site like CheapBotsDoneQuick).

FSM libraries already exist, of course, but with Bottery, the goal is for users to be able to describe interesting, complex, and generative bot conversations, in a JSON format. These bots can be standalone experiments, or hosted on a similar site to CheapBotsDoneQuick for conversational agents that could converse with users on Twitter, smart speakers, or even be casually embedded as a part of webpages or Unity games, as Tracery was.

Bottery programs are maps describing finite state machines. Each map has a dictionary of states. Each state has a list of exits with preconditions and post actions, and actions that it can fire on entering, exiting, etc. When giving bottery tutorials, I describe maps as boardgame boards with individual pointers moving around them and changing scores. Boardgame players know that moving between states can mean spatial movement, but also temporal (the Game of Life) or metaphorical/emotional/moral (Chutes and Ladders) movement, a connection which helps new users understand how to represent conversational spaces as FSMs.

Bottery is built to use Tracery for generative text as well as parsing. Bottery pointers can change the blackboard (to set a user's name or hitpoints), and actions that output text can call on Tracery to expand statements like "#/userName# has #/data/points#, #congratulations#."

We can also use Tracery grammars in reverse . Why? It is common for conversational bot authors to want to respond to prompts that the user may phrase in different

ways (yeah, yes, no, yeah totally). Machine-learning can help with this, but isn't easily controlled by an author. Here, the author writes a grammar capable of generating input values. The exit condition for a state might be #yes#, which matches any generation of that Tracery rule. Conveniently, this also gives us a way to simulate the users of the system, either autonomously for testing, or to give suggested inputs.

Bottery is a system under continuing development, but it's been able to make a number of interesting bot prototypes very quickly. The original version is on GitHub (https://github.com/google/bottery). A playable version is at https://rawgit.com/google/bottery/master/index.html.

Conclusion

The Playable Experiences track at AIIDE presents five unique works that leverage artificial intelligence to create novel, polished gameplay experiences. These pieces are technical contributions and are valuable resources for the future of the field. With an emphasis on playability and audiences, these works have the potential to inspire future game designers and researchers to further chart unexplored design space through innovative artificial intelligence approaches.

References

Compton, K.; Kybartas, B.; and Mateas, M. 2015. Tracery: an author-focused generative text tool. *International Conference on Interactive Digital Storytelling:* Springer, Cham.

Compton, K. 2017. Bot Poetics. *Talk at Interrupt.XYZ.* Providence, RI (https://vimeo.com/225566776).

McCoy, J.; Treanor, M.; Samuel, B.; Reed, A.; Wardrip-Fruin, N.; Mateas, M. 2014. Social Story Worlds with Comme il Faut. *IEEE Transactions on Computational intelligence and AI in Games 6.2*:97-112.

Plotkin, A. 2011. Characterizing, if not defining, interactive fiction. *IF Theory Reader*:59-66. TRANSCRIPT ON Press, Boston.

Reed, A. A. 2014. Ice-Bound: Combining richly-realized story with expressive gameplay. *Foundations of Digital Games*.

Samuel, B.; Reed, A. A.; Maddaloni, P.; Mateas, M.; Wardrip-Fruin, N. 2015. The Ensemble Engine: Next-generation Social Physics. *Proceedings of the Tenth International Conference on the Foundations of Digital Games (FDG 2015).*

Short, E. 2011. NPC Conversation Systems. *IF Theory Reader (2011):* pages 331-358. TRANSCRIPT ON Press, Boston.

Treanor, M.; McCoy, J.; Sullivan, A. 2016. A Framework for Playable Social Dialogue. *In the Proceedings of the AI and Interactive Digital Entertainment Conference (AIIDE 2016).*