Towards Game Generation

Michael Mateas (UC Santa Cruz), Mark Nelson (Georgia Tech)

Research in game generation seeks to build AI systems that make design decisions with respect to the game rules, as well as the visual realization of the rules. Where procedural content generation focuses on the generation of assets such as textures, meshes, animations and sounds, and the physical layout of levels, game generation focuses on generating the game rules themselves, the game mechanics that describe how the game state evolves over time, and how player action influences the game state. Procedural content generation is then used to realize the abstract game dynamics. This paper provides a brief work-in-progress report on game generation working taking place in the Expressive Intelligence Studio at UC Santa Cruz.

Why Game Generation

The goal of our game generation research is not to replace human designers, but rather:

- to facilitate formal game analysis through the computational expression of game rules, mechanics, and representations,
- to enable new game mechanics and game genres where the game dynamically morphs and changes as a function of player interaction,
- to move human design up the abstraction hierarchy to the meta-authorship of generative processes that generate games.

Both game designers and game scholars have discussed the need for a game design language, noting that designers currently lack a unified vocabulary for describing existing games and thinking through the design of new ones. Many of the proposed design language approaches focus on offering aid to the designer, either in the form of design patterns, which name and describe design elements, or in the closely-related notion of design rules, which offer advice and guidelines for specific design situations. Other analyses draw methods and terminology from various humanistic disciplines – for example, games have been analyzed in terms of their use of space, as semiotic systems, as narrative forms, in terms of the temporal relationships between actions and events, and in terms of sets of features in a taxonomic space, using clusters in this space to identify genres.

The Game Ontology Project (GOP) (www.gameontology.org), is developing a game ontology that identifies the important structural elements of games and the relationships between them, organizing them hierarchically [1, 2]. Our use of the term *ontology* is borrowed from computer science, and refers to the identification and (oftentimes formal) description of entities within a domain. Often, the elements are derived from common game terminology (e.g. level and boss) and then refined by both abstracting more general concepts and by identifying more precise or specific concepts. An ontology is different than a game taxonomy in that, rather than organizing games by their characteristics or elements, it is the elements themselves that are organized. The GOP provides a framework for exploring, dissecting and understanding the relationships between different game elements. A few examples of research questions we have already begun to explore include: “How can we understand interactivity in games?”, “How is gameplay regulated over the progress of a game?”, and “What roles does space play within games?”

Where the GOP attempts to achieve broad coverage at a fairly high level of abstraction, producing a semi-formal analysis in which ontological elements are describe in natural language (English) and intended to facilitate human game analysis, the game generation project formalizes ontological elements at a high level of detail. The ontological structure must be formal enough that it supports machine reasoning. Thus the game generator becomes a highly detailed theory of both game structure and game design expressed operationally *as a program*; in the same way that AI-based story generators have, over the years, served as operational models of both narrative and the story generation process, and thus served to expose the strengths and weaknesses of different models of narrative, so too can game generation facilitate formal game analysis. Given the formality and level of detail required of the game model, we’re starting by formalizing small subsets of the Game Ontology Project, focusing on simple arcade games.

In addition to facilitating game analysis, game generation enables new game mechanics and game genres where the game dynamically changes (or is dynamically generated from scratch) as a function of either

player input or other exogenous events. Newsgames are one such category of game – a newsgame is a micro-game that provides commentary on a news item, much in the same way a political cartoon provides commentary. But unlike a political cartoon, newsgames provide their commentary through gameplay; only through interaction on the part of the player is the point made. Some well known newsgames include *September 12* [3], critiquing the war on terror, *Madrid* [4], a memorial game released shortly after the Madrid train bombings on March 11, 2004, and *Bush Backrub* [5], poking fun at the impromptu back rub President Bush gave German Chancellor Angela Merkel at a G-8 meeting in July 2006. Newsgames have a number of properties that make them amenable to game generation. First, in order to function as effective commentary, newsgames must be timely, appearing within a day or two of the news item. Yet even microgames can be difficult to complete in only a day; *Madrid*, for example, took a team of several people two days of intense effort to complete. Thus, automatic generation facilitates the rapid construction necessary for timely commentary. Second, newsgames tend to be small microgames exhibiting simple play mechanics, making automated generation tractable, even in the short term. Game generation can thus enable newsgames as an effective form of commentary, allowing many newsgames to appear on a daily basis in response to news events.

With Ian Bogost, we are exploring the use of game generation in the area of political games. Currently, political games tend to hardcode their rhetoric either into the visual representation (for example, visually representing invaders as tax bills that must be shot down so as to express an anti-tax viewpoint), or into the simulation rules of simple simulation games (for example, tuning an economic simulation so as to make a specific point about the relationship between tax rates and business growth). As an alternative to coding a single rhetoric into the visual representation and simulation rules, we're creating a framework in which the system dynamically generates a series of mini-games that takes the player on a tour through the ideological space associated with a political issue. The specific games generated are a function of how the player has played previous mini-games in the sequence. While each individual mini-game represents a single, hard-coded rhetoric, the design process for conveying a rhetoric through a game has been pushed into the game itself, allowing this process to dynamically respond to the gameplay. A computational model of ideology, borrowing techniques from *Terminal Time* [6, 7], will guide the game generator described in this paper.

Service Games

We are beginning our game generation work with simple arcade games, specifically starting with a game style we call *service* games. In a service game, the player engages in frenetic time management in the context of satisfying "customer" requests. In order to understand the design space of service games, it is useful to look at a number of such games in order to pull out common elements and understand the space of variation. The political game generation framework mentioned above will need to be able to generate games in many game styles. However, we're currently focusing on a single style as a way to make progress on our architecture and knowledge representation formalisms.

The canonical service game is *Tapper* [8], a 1983 arcade game from Bally Midway in which the player serves beers to customers (customers must be served before they reach the end of the bar) and collects tips and empty glasses (see figure 1). As levels advance, the player is placed under increasing time pressure as the numbers of customers increase and they move with increasing speed up the bar. Beers are served by filling glasses from the tap and releasing them down the bar. When a customer receives a glass of beer, they drain it and send the empty glass sliding back up the bar; the player must catch the empty glass before it reaches the end of the bar. Occasionally a customer will leave a tip. If the player collects the tip, she earns bonus points; additionally, dancers appear for several seconds after the tip is collected, distracting some of the customers, temporarily halting their advance up the bar.

Pressure Cooker [9] is a 1983 Atari 2600 cartridge from Activision in which the player is a short order cook at hamburger stand who assembles hamburgers to order without letting the hamburgers or ingredients fall on the floor (see figure 2). Unlike *Tapper*, in which beer orders must be delivered but do not require assembly, hamburger orders in Pressure Cooker require the player to put together multiple ingredients (e.g. cheese, lettuce, tomato). Customers are not explicitly represented; rather, the hamburger orders are represented in a grid at the bottom of the screen. The player stacks completed hamburgers in bins. Rather than the time limit for fulfilling an order being associated directly with an order, instead time limits are associated with the movement of partially-assembled hamburgers on a conveyor belt. If the hamburger is

not properly assembled before it reaches the end of the conveyor belt, it falls on the floor and the player looses points.

Figure 1: *Tapper*

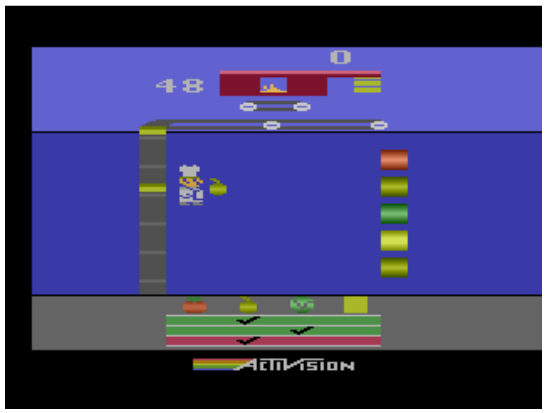

Figure 2: *Pressure Cooker*

Figure 3: *Disaffected!*

Figure 4: *Bush Backrub*

Dissaffected! [10], an "anti-advergame" from Persuasive Games released in January of 2006, provides a contemporary example of a political game making use of the service mechanic (see figure 3). *Dissaffected!* is a videogame parody of a Kinko's copy store. As dissatisfied customers line up in the store, the player confronts obstacles such as fellow employees who refuse to work, or who indiscriminately move customer orders around. As is characteristic of service games, each customer has a time limit within which the player must satisfy their order. If too many customers angrily leave the store with their orders unfilled, the game ends.

The last example service game we will present in this paper is *Bush Backrub*, a newsgame poking fun at the impromptu back rub President Bush gave German Chancellor Angela Merkel at a G-8 meeting in July 2006. In *Bush Backrub*, the player, as President Bush, runs back and forth between world leaders giving backrubs. Happiness meters under each leader continuously drop; a backrub recharges the meter. If the happiness meter under any leader drops to 0, the game is over. As a service game, it has the interesting property of service requests being continuous rather than discrete. Backrub requests are constantly active, with backrubs recharging an analog value rather than fulfilling discrete requests.

Knowledge Representation for Game Generation

A game generator requires formally represented knowledge of game design. We've currently identified four different knowledge sources: abstract game mechanics, game representation, thematic world, and player input. By separating game design knowledge into four independent knowledge sources, we hope to support the mixing and matching of knowledge sources in order to define different design spaces.

The game mechanics knowledge source describes the elements of the abstract game state, and how that game state evolves over time, both autonomously and in response to player interaction. In our game generation project, we are starting with the generation of abstract mechanics; thus, the ontology of this knowledge source has been worked out in the greatest detail. The game mechanics knowledge source helps define the game type currently being explored in the design space. For service games, the game mechanics knowledge source contains knowledge about the following objects:

- Service requests: An open request for a specific service or object. In *Tapper*, customers who aren't currently drinking a beer have an open request for a beer. Open service requests have a timer associated with them; if the service request isn't satisfied within the time period, the player is penalized in some way (e.g. in *Tapper* losing a life, in Pressure Cooker loosing "performance points"). Service requests can chain, with the satisfaction of an outstanding service request resulting in the spawning of an additional request, often of a different type. For example, in *Tapper*, satisfying an outstanding beer request generates a request to pick up the drained beer mug. The customer throws the beer mug down the bar; the movement of the mug down the bar represents the timer associated with the request. If the glass reaches the end of the bar without the player picking it up, it falls off the end of the bar and breaks, resulting in the player losing a life. Occasionally, satisfying a drink request in *Tapper* results in a tip being left on the bar. If the player picks up the tip before it disappears, she gains bonus points, in addition to causing dancers to appear who may temporarily distract customers. The tip is modeled as a service request that is randomly spawned, with low probability, after the satisfaction of an outstanding beer request.
- Service request sources: The game entities that issue service requests. In *Tapper*, the bar customers are service request sources. In *Pressure Cooker*, the abstract request board at the bottom of the screen lists the ingredients of the currently requested hamburgers without anthropomorphically representing the service request sources as customers. Services request sources can have state associated with them representing the request source's (customer's) attitude towards their associated service request(s). For example, in *Tapper*, customers can be in the state of being satisfied (while they possess and drain their beer), unhappy (while they have an outstanding beer request, during which time they repeatedly pound the bar and slowly move up the bar, indicating the countdown of the timer associated with their current request), and distracted (when the dancers appear after the player has picked up a tip, some of the customers turn towards the dancers – during this period they have no outstanding service request). In *Dissaffected!*, service request sources (customers) can become exasperated, at which point the timer of the service request associated with the customer counts down twice as quickly.
- Requested service: a specific service associated with a service request issued by a service request source. In *Tapper*, the requested service is a non-compound object (a beer), in *Pressure Cooker* a compound object (a hamburger composed of specific ingredients), in *Bush Backrub* an action (a backrub). In the later case, a backrub increases a continuous "happiness" value rather than satisfying a service request – so the service requests associated with the three world leaders (the service request sources) are continuously outstanding, rather than coming into being, being discretely satisfied, and disappearing. One can imagine a version of *Bush Backrub* (perhaps called *Bush Spa*) with multiple requested service types (backrub, pedicure, facial, etc.) where the player must run back and forth satisfying the correct request for each world leader. In this game, service requests would be more discrete, with, for example, a request for a pedicure coming into being once the backrub request has been satisfied.
- Service source: a source for the requested service. In *Tapper*, the service source is the tap, where players can retrieve beers (requested service) to deliver to customers (service request sources), in order to fulfill an outstanding service request (a request for a beer). In many service request games, much of the gameplay revolves around the player moving back and forth between service request sources (customers) and service sources; the skill in the game lies in time management, fulfilling

open service requests in an order that minimizes the number of services requests that time out. In *Pressure Cooker*, the service source is compound, consisting of both the conveyor belt carrying partially constructed hamburgers, as well as ingredients bins (colored blocks). Ingredients fly out of the bins towards the player; the player must catch the appropriate ingredients and apply them to the hamburgers under construction. In *Bush Backrub*, the service source is implicit and is attached to the player avatar. Providing a service merely requires moving to the service request source (world leader), rather than coordinating movement between two locations (request and service source). In *Dissaffected!*, the service sources are the piles of completed work (e.g. copy jobs) scattered throughout the workspace. Co-workers randomly move completed jobs between piles, forcing the player to frantically search through different service sources (piles) for the requested service.

In addition to objects, the game mechanics knowledge source has an ontology of events. Events reify state changes associated with objects as well as provide the representational hooks for the audio-visual representation of state changes. For service games, event types include:

- Service fulfillment event: an open service request is fulfilled.
- Service acquisition event: an attempt to acquire a service (object) or the ingredients for a compound service (object). The attempt may succeed in fail. In *Tapper*, the player must hold down the joystick button (corresponding to pulling the lever on the keg) long enough to fill the glass. If the button is not held down long enough, the glass is only partially filled and can't be sent down the bar to fulfill an open service request. In the visual representation *Tapper* represents the service acquisition event by depicting the glass filling; a failed acquisition attempt is represented by a partially full glass.
- Service request source spawn event: the appearance of a service request source. In *Dissaffected!*, customer spawning is visually represented by having a customer walk through the front door. In *Bush Backrub*, all service request sources (the world leaders) appear at the beginning of the game and never disappear.
- Service request source removal event: the removal of a service request source. In *Tapper*, customers disappear (are pushed off the end of the bar) if they are served a beer when they are close to the far end of the bar (the timer of their associated service request has not counted down very far).

At the abstract mechanics level, a game is represented by a collection of predicate calculus assertions written using the above ontology. The assertions capture the evolution of game state over time. A design-in-progress is a collection of uncompleted assertions which are incrementally modified by the design process (our architecture for modeling the game design process is described in more detail below).

The other three knowledge sources have not yet been developed in as much detail as the game mechanics knowledge source. Here we provide brief descriptions of the knowledge we intend to include in each source.

The game representation knowledge source captures knowledge about how to provide an audio-visual representation of the underlying game state. This knowledge source includes both spatial layout knowledge as well as well as interface design knowledge for representing non-spatial game states. For example, in representing a countdown timer, it might be represented through the movement of objects towards a goal location (in *Tapper*, the countdown of a timer associated with an open beer request is represented through the movement of the customer towards the close end of the bar), a sweeping analog clock (in *Dissaffected!*, timers associated with service requests appear as analog clocks floating above the heads of customers), or a slowly emptying bar graph (in *Bush Backrub*, the timer is represented by a bar under each world leader). The knowledge for these different strategies for representing a number (the timer is just a number counting down) live in the representation knowledge source. Different representation knowledge sources correspond to different representational styles. We're currently defining a representation knowledge source for 2D game representations. A different representation knowledge source might describe representational strategies for 3D first-person displays. The goal is to be able to independently swap out the representation knowledge source, so that representational choices such as 2D top-down vs. 3D first-person can be varied independently of the abstract game mechanics.

The thematic knowledge source captures knowledge about the theme (real-world references) expressed by the game. In *Tapper*, for example, the theme is a bar. In order to generate service games set in a bar, our system requires information about bars such as the typical spatial layouts of bars, the types of roles people play in bars (e.g. the bartender behind the bar, the cocktail server, the bouncer, a customer), the types of activities in bars (e.g. drinking, watching TV, talking, fights…), and the objects found in bars (e.g. taps, glasses, mixed drinks, beer…). At the simplest level, the thematic knowledge source can be used to "skin" the abstract game mechanics after the mechanics have been generated, determining for example that a service request should be for a beer, that the player should be a bartender or a cocktail server, etc. In a pipelined architecture in which the abstract game mechanics are completely generated prior to applying thematic knowledge, the same underlying game could be mapped to different themes such as bar, fast food restaurant, or copy store by dropping in different thematic knowledge sources. Ultimately, however, we're less interested in such a pipelined approach than in exploring the ways in which thematic and game mechanics knowledge are applied simultaneously and interact richly during the game generation process. Thematic knowledge should suggest gameplay opportunities and constrain the abstract mechanics, just as the abstract mechanics constrain thematic mappings. For example, if the bar theme is active, thematic knowledge might reason that as people drink they become drunk, and drunk people move erratically, suggesting that drunk customers might serve as obstacles for the player as they serve drinks, but only if the player is a cocktail server moving between tables. This movement from theme to mechanics is particularly important for the generation of political games, where much of the rhetorical force of the game depends on the appropriate incorporation of thematic elements into the gameplay.

The last knowledge source, control mappings, captures knowledge about the mapping between the physical player input and abstract player actions in the gameworld. This knowledge source provides analogous knowledge to the game representation knowledge source, but on the input side rather than output side. For example, in mapping the player's acquisition of a service during a service acquisition event (e.g. filling a beer in *Tapper*) to the game controls, it is this knowledge source that recommends different control mapping strategies (e.g. button tap, holding the button for a certain length of time, holding the button while pumping the joystick, etc.). Different control mapping knowledge sources correspond to different mapping strategies, usually for physically different controllers. So you may have a game control knowledge source for button and joystick controls, another for 3D gestural controls (ala the Nintendo Wii), and a third for dancepad controls. Again, we aim for this knowledge source to be changeable independently of the other sources, allowing the formation of design spaces offering different combinations of game mechanics, state representation, theme and control mappings.

Architecture

A game design is represented through a collection of logical formula describing the game mechanics and associated thematic mappings, state representation, and input mappings employed by the game. The predicates and functions of these assertions are taken from the ontologies of the active knowledge sources, which, collectively, define the active design space. The knowledge of the active knowledge sources is put into action by a rule-based system whose rules match on the structure of the design-in-progress, as well as knowledge in the active knowledge sources, to incrementally modify the design-in-progress. The left hand side of the rules is not limited to matching on the structure of ground terms, but can perform arbitrary inference, allowing rules to match on inferred properties of the design (inference rules live in the knowledge sources, along with the taxonomic assertions of terms defined in the ontologies).

Rules are divided into three different categories, namely local and global critics and local proposers. Local proposers and critics are the most frequently run rules. Local proposers are specialists on specific subparts of the design, focusing on, for example, the conditions under which a service request source should be removed, or the visual representation of a timer. As the name implies, local proposers propose changes to the design, corresponding to asserting or retracting logical assertions in the current design. Local proposers tend to propose promiscuously, suggesting many possible design changes; as specialists, the proposed design changes don't take global design constraints into account. Local critics look at the suggestions made by local proposers and assign them numeric quality ratings. The actual design proposal implemented in the evolving design is chosen probabilistically based on the quality ratings (a proposal with a high quality rating will be chosen more often). Together, the local proposers and critics serve a stochastic, bottom-up

brainstorming function, trying out many different design proposals without worrying too much about how they fit together to form a unified design.

Global critics take into account global design constraints, and are thus responsible for ensuring that all the individual design decisions made by the local proposers form a coherent design. Global critics turn different collections of local proposers on and off, as well as tune parameters in local proposers. For example, a global critic may notice that the current design requires different controller actions to be performed by the player for acquiring different services of the same general type (e.g. different types of beer in a *Tapper* variant with multiple beer types) and activate and tune local proposers and critics such that it becomes likely that either the same controller action will be chosen for the different services or that the services are differentiated enough that different controller actions make sense (e.g. pushing the joystick button to acquire a beer vs. moving the joystick back and forth rapidly to blend a martini). Global critics thus provide top-down guidance to the bottom up brainstorming, focusing brainstorming on specific design problems.

Once a design has been generated as a collection of assertions, it must be turned into running code that instantiates a playable game. We are currently using simple parameterized code techniques in which we provide the system with carefully parameterized building blocks for service games (code objects corresponding to generic service request sources, requested services, etc.), and fill in the parameters to instantiate a specific game by querying the formally represented design. Eventually we want to move to a richer code generation approach in which the code is generated by reasoning about the current design, without the need for the author to provide the system with large blocks of pre-written code as is required by the parameterized code library approach. We currently handle art assets by mapping thematic elements to canned assets. If a design refers to a concrete service source such as a beer tap or stack of paper, or a physical prop such as a counter, the appropriate art asset is retrieved from an indexed library. Eventually we would like to procedurally generate art assets based on assertions about the properties of the desired asset, though this art generation problem is, in its full generality, as big as the rest of the game generation problem.

Conclusion

As a research agenda, game generation facilitates formal game analysis, enables new game mechanics and game genres, and moves human game authorship up the abstraction hierarchy from individual games to potential game spaces. In this paper we've presented our work-in-progress towards creating a game generator.

Bibliography

1. Zagal, J., Mateas, M., Frenadez-Vara, C., Hochhalter, B., and Lichti, N. Towards an Ontological Language for Game Analysis. In *Proceedings of the Digital Interactive Games Research Association Conference* (DiGRA 2005), Vancouver, B.C., June 2005.
2. Zagal, J., Fernandez-Vara, C., and Mateas, M. Gameplay Segmentation in Vintage Arcade Games. Forthcoming in Bogost, I., and Bittanti, M. (Eds), *Ludologica Retro Volume 1: Vintage Arcade (1971-1984)*.
3. *September 12*. Montevideo, Uruguay, newsgaming.com, 2001.
4. *Madrid*. Montevideo, Uruguay, newsgaming.com, 2004.
5. *Bush Backrub*. Addicting Games, 2006.
6. Domike, S., Vanouse, P. and Mateas, M. The Recombinant History Apparatus Presents: Terminal Time. In Mateas, M. and Sengers, P. eds. *Narrative Intelligence*, 2003, 155-173.
7. Mateas, M., Vanouse, P. and Domike, S. Generation of Ideologically-Biased Historical Documentaries. In *Seventeenth National Conference on Artificial Intelligence (AAAI 2000)*, Austin TX, 2000, AAAI Press, 36-42.
8. *Tapper*. Bally Midway, 1983.
9. *Pressure Cooker*. Activision, 1983.
10. *Disaffected!*. Persuasive Games, 2006.