Scripting Versus Emergence: Issues for Game Developers and Players in Game Environment Design

Penelope Sweetser and Janet Wiles
School of Information Technology and Electrical Engineering
The University of Queensland
St Lucia, Queensland,
Australia
E-mail: penny@itee.uq.edu.au

KEYWORDS
Emergence, Scripting, Game Environments, User-Centered, Game Design.

ABSTRACT

This paper defines and discusses two contrasting approaches to designing game environments. The first, referred to as scripting, requires developers to anticipate, hand-craft and script specific game objects, events and player interactions. The second, known as emergence, involves defining general, global rules that interact to give rise to emergent gameplay. Each of these approaches is defined, discussed and analyzed with respect to the considerations and affects for game developers and game players. Subsequently, various techniques for implementing these design approaches are identified and discussed. It is concluded that scripting and emergence are two extremes of the same continuum, neither of which are ideal for game development. Rather, there needs to be a compromise in which the boundaries of action (such as story and game objectives) can be hard-coded and non-scripted behaviors (such as interactions and strategies) are able to emerge within these boundaries.

INTRODUCTION

The approach that is used to develop game worlds holds considerations for game developers and players. The current approach to developing game worlds is a scripted approach. Scripting involves a specific, low-level, entities-based approach to developing game worlds. The considerations of the scripted approach for game players include inconsistencies in the game world, unintuitive interactions, a slow learning curve, limited freedom for the player and no possibility of emergent gameplay. For game developers, developing scripted game worlds involves substantial effort in planning, implementing and testing, difficulties in extending and modifying, and issues with quality assurance due to inconsistencies. However, the current scripted approach does afford developers full creative control, no uncertainty in how the game system will behave and ease of giving feedback and direction to players. The current proliferation of the scripted approach is partly due to these reasons and partly due to the widespread use of scripted and static software techniques, such as scripting and finite state machines.

One possible alternative to the current scripted approach is an emergent approach to developing game worlds. Emergence involves a top-down, systems-based approach to developing game worlds. Emergence has been integrated to a limited degree in previous games to allow emergent gameplay or emergent narrative. Considerations of an emergent approach for game developers include significant planning and tuning in development, a loss of creative control, difficulties in giving feedback and direction to players and uncertainty in how the game will respond to the player. However, emergent systems are easier to modify and extend and the uncertainty gives the possibility for emergent gameplay. Emergent systems can potentially improve player experience as they are inherently consistent, interactions can be more intuitive, the players' learning curve can be reduced, and emergent systems allow far more freedom for players and the possibility of emergent gameplay. Techniques that can potentially be used to facilitate emergence in games include flocking, neural networks, cellular automata and evolutionary algorithms.

This paper presents the issues associated with these different design approaches for game developers and players. It then discusses the methods that are currently used for hand-crafting game worlds and identifies and explores the techniques that have the potential to facilitate emergence in games.

SCRIPTING AND EMERGENCE

The majority of current games are developed with a scripted approach, which involves the game developer predefining specific paths and interactions that the player will take throughout the game. Scripted game design is the creation of gameplay out of the ideas of a particular designer, as needed for a specific, localised occurrence in the game. Scripted design involves limited awareness of global game patterns and relies on a given designer's ideas of what is consistent and fun (Smith, 2002). The environments, objects and agents in these games are limited to the narrow and static behaviour that the developer has predefined. As a result, the players' possible interactions with these game elements and resulting gameplay is confined, inflexible and lifeless. These scripted systems have also been referred to as "emulations" (Church, 2002) and "specific" systems (Smith, 2002).

A possible alternative to the current scripted approach to game design is to design general, rule-based systems that allow the creation of gameplay out of combinations of existing game elements with globally defined, consistent characteristics and behaviour. This emergent approach to game design is also referred to as "simulation" (Church, 2002) and "systemic" system design (Smith, 2002) in the game development literature. An emergent approach to game design requires a globally designed game system that provides rules and boundaries for player interactions, rather than prescripted paths.

CONSIDERATIONS FOR GAME DEVELOPERS

Different approaches need to be taken to develop games that are emergent versus scripted. They also offer different advantages and disadvantages for the development team. In developing scripted games, the development team needs to design specific game elements, and implement and test them individually, which can be costly in time and effort. However, the designers are empowered to create a specific narrative and flow for the game and there are no nasty surprises. For emergence, the development team needs to design types of objects and has the convenience of dropping a type of object into a certain level. This approach gives rise to greater efficiency in implementation and testing. However, there are potential problems with uncertainty and loss of control for the designers.

There are five central issues in the game development literature that are important to consider when designing game systems. These issues are (1) effort in designing, implementing and testing, (2) effort in modifying and extending, (3) level of creative control for game developers, (4) uncertainty and quality assurance, and (5) ease of feedback and direction to players. Each of these issues is described in this section and discussed with respect to scripted and emergent games.

Developer Considerations for Scripted Systems

Effort in Designing, Implementing and Testing

In developing scripted games, specific interactions need to be planned by the game designers (Church, 2002) and the possible courses of action that the players can take need to be manually setup by the developers (Smith, 2001). Scripting requires a "look and feel" approach to the placement of units, weapons, tools, resources, and specific puzzles or scripted sequences. Scripted games require a considerable time and effort by the designers, as well as vigilant manual effort to ensure consistency in the game world (Smith, 2002).

Effort in Modifying and Extending

Scripted systems scale poorly and do not lend themselves to extensibility (Church, 2002). The properties and parameters of objects in scripted systems are different for each instance. Also, objects must have explicit relationships with other game elements for interactions to occur. For example, for a bullet from a gun to break a window, there needs to be a direct relationship between the gun entity and the window entity (Smith, 2001). The gun class would need to contain code listing all the things it could affect. Consequently, any changes that need to be made to the system require revision of any aspect of the game that is affected by the change (Church, 2002). Also, fixing bugs in the system requires each instance of a game element to be visited and reconfigured manually (Smith, 2002).

Level of Creative Control

As game developers manually plan and set up specific situations, interactions and events in scripted games, the game designers have full creative control over the game. The designers are empowered to create a specific narrative flow for the game, by defining the order and nature of the players' actions and encounters in the game.

Uncertainty and Quality Assurance

Similarly, nothing occurs in the game that was not intended or planned by the game developer. Consequently, there is no uncertainty or unexpected events in the game. The player plays the game in the exact way that the developer had intended. However, due to the inconsistencies that can exist in scripted games, quality assurance requires extensive testing of each game element, interaction and event. The scripted approach is effective for developing simple systems or specific complex behaviour, but can be difficult to manage on a larger scale.

Ease of Feedback and Direction

As with creative control, giving feedback and direction to players is simple in scripted systems as the developer knows when and how the player will interact with various game elements. As the desired outcome is known, it is straightforward to give players feedback on their success at performing actions or fulfilling goals.

Developer Considerations for Emergent Systems

Effort in Designing, Implementing and Testing

Creating emergent games involves designing types of objects and interactions, rather than specific ones (Church, 2002), which can give rise to greater efficiency in development and testing. The properties and parameters reside at a higher level (Smith, 2002). Rather than having a specific gun able to break a specific window, there is an additional layer of abstraction that allows a gun to break anything made of glass. For example, the gun would project a bullet entity that has certain properties (e.g. ballistic damage, heat or electricity) and the glass is a stimulus-receiving entity (Smith, 2001). The system would have a set of rules about the relationship between the entities' general-case properties and when the bullet meets the glass, the game's object-property system looks up the effect of the bullet's properties on the glass entity. Therefore, the gun will work on any window (or any other stimulus-receiving object), rather than only the specified windows.

Emergent systems often require considerable initial effort in planning and building, as the rules and properties need to be defined in advance. Additionally, the system can require a lot of tuning to get the rules and properties to function correctly. However, development can be more efficient as programmers can build tools that allow designers to "drop" objects into levels, with the properties and behaviour of the object already defined. Designers can also create new objects and attribute properties to the objects using the tools (Smith, 2002).

Effort in Modifying and Extending

Once an emergent system is built successfully, the design scales well (i.e. increases in size easily, maintaining robustness and manageability) and is easily extended (Church, 2002). Making changes to the system (e.g. fixing bugs) has the potential to be more efficient as changes can be made to object types, rather than each particular instance of an object than needs to be changed (Smith, 2002).

Level of Creative Control

The use of emergence in games could result in a possible loss of creative control for the game designer. Using emergence involves defining types of interactions and behaviours, which makes it is more difficult to set up specific narrative and sequences. Consequently, controlling the flow of game and telling a specific story is not as straightforward in an emergent system.

Uncertainty and Quality Assurance

Emergence also introduce uncertainty, which means that the game can behave in ways that the developers had not anticipated. Although this uncertainty can give rise to desirable, emergent gameplay, it can also be undesirable if the system allows behaviour that is detrimental to the game (Church, 2002). Extensive testing is required to ensure that the game does not allow detrimental behaviour. However, the emergent events can be too numerous or subtle for the development team to predict or detect during testing (Smith, 2002).

Ease of Feedback and Direction
Players have a greater need for feedback on the outcome and success of their actions in emergent systems, as the openness of the game world gives rise to more possibilities for action (Smith, 2001). Consequently, the players need more feedback to know that they are on the right track and that their actions are successful.

CONSIDERATIONS FOR GAME PLAYERS

As well as having significantly different development approaches, scripting and emergence also give rise to different methods of playing the game. Some issues that need to be considered include the ability of the game to uphold the player's suspension of disbelief, consistency in the game world, the intuitiveness of the environment, player expectation and learning, and how well the game facilitates player expression and emergent gameplay. This section discusses the impact that scripting and emergence have on each of these issues.

Consistency and Immersion

Game worlds that behave consistently and in ways that the player understands enable the player to become immersed in the environment and suspend disbelief (Smith, 2001). Conversely, inconsistencies in games remind that player that it is just a game, breaking their suspension of disbelief. For example, if the player becomes stuck in a wall when adventuring in a dungeon (Hecker, 2000) or a monster attacks them through the wall then inconsistencies occur with the fantasy that the game has created. Similarly, if a boom microphone appears in an emotional scene in a movie, the immersion the viewer feels – their suspension of disbelief – is instantly broken (Hecker, 2000). The viewer of the movie or the player of the game is transported back to the real world, reminded and disappointed that their experience was fake. Scripted game systems inherently break the player's immersion, as their specific interactions and situations give rise to many inconsistencies.

Emergent systems have the potential to be used to create more consistent game worlds (Smith, 2001). The game worlds in emergent systems are inherently consistent as the rules and properties are defined globally, for types of objects, rather than locally for each specific object. For example, the player knows that bullets affect everything that is damageable, such as windows, vases and chairs, rather than some windows and no vases. Furthermore, the player can deduce that if they can move objects and put objects on top of one another then they can stack crates. Games that obey a consistent set of physical laws allow the player to stay immersed in the game, sparing them from unpleasant surprises (Hecker, 2000).

Intuitiveness and Learning

Another important aspect of player interaction with the game environment is intuitiveness and player expectation. Casual game player or non-game players can be baffled by the physics in game worlds (Smith, 2001). In some game worlds, only "explosive" barrels burn, some pieces of light furniture cannot be moved, the player's character might not be able to climb onto a desk and sometimes glass does not break. In order to be able to play computer games, it is necessary to relearn the physics of the world like a child (Smith, 2001). These types of problems arise in scripted games because the possible interactions that the player can have with the game environment are not intuitive and they do not meet player expectation.

The intuitiveness of interactions in game worlds can be partly attributed to how the interactions correspond to interactions with the same objects in the real world. Game worlds are populated with objects that are visually similar to objects that we use every day, but that are functionally different. Not only can these interactions be counter-intuitive for the player, but they can often confuse and frustrate the player (Hecker, 2000). It is natural for a player to expect that they will be able to pick up a phone, kick a chair and break a window, as they have learned these actions are possible throughout their whole life. However, in scripted games, these actions are only possible if the developer has specifically coded them for each game object. Consequently, it is likely that many intuitive and seemingly logical actions will not be possible.

Game worlds that work in a way that reflect players' lifelong experiences (in the real world) are more intuitive and easier to understand for the average person, even in fantasy realms and alien dimensions (Smith, 2001). Emergent games are more likely to be intuitive to the average person as it is easier to create objects that behave and interact in more natural ways, with a wider variety of interactions. The objects in emergent games are not limited to specific interactions that have been hard-coded. Instead, they interact in ways that are conducive to their properties and rules for interaction.

An important benefit of making game worlds more intuitive is that they become easier to learn. The player is more likely to develop an intuitive understanding of the game elements if they are consistent with real world elements (Smith, 2002). For example, if fire in the game behaves like fire in the real world then the player will have an inherent understanding of how the fire works, without needing to be retaught the rules of fire within the game (Smith, 2001). With the use of intuitive game elements, the player is more likely to understand the elements, even when encountering them for the first time. As a result, the learning curve of the player is substantially decreased, which means that the player spends less time learning and more time playing the game (Smith, 2002).

Emergent Gameplay and Player Expression

The final issue identified in the game development literature is the degree of freedom of player expression and the possibility of emergent gameplay that is supported by the game system. In scripting, the designers manually define a number of outcomes or interactions and allow the player to pick one. The result is a handful of canned solutions to each particular problem (Smith, 2001), which makes the game linear (i.e. only one path through the game). The player is given a choice of a small number of static courses of action to take, which have been predefined by the game designers. The game is played in the exact way it was specified, which might not accommodate player creativity (Church, 2002).

In contrast to scripted systems, emergent systems define global possibilities for actions the player can perform, which can be applied in more open ways in specific situations. Players have more freedom to express their creativity and gameplay can occur that wasn't anticipated by the designers. Emergent gameplay allows players to solve game problems by using strategies that were not envisaged by the designers (Smith, 2001; Garneau, 2002). Emergent gameplay occurs when a player's actions result in a second order of consequence that the development team did not predict and the game behaves in a rational but unplanned way (McLean, 2002; Smith, 2002). For example, in the game Deus Ex, players used proximity mines to create ladders up walls to climb off the map, a possibility that was not foreseen by the developers.

Emergent games empower the player by putting them centre stage (Church, 2002), giving them freedom to experiment, greater control, a sense of agency, and less of a feeling of uncovering a path set for them by the designers (Smith, 2002). Consequently, the game can be more satisfying and interesting for the player. Game worlds that are not full of prescripted one-to-one interactions are empowering to the player as the gameplay becomes largely about exploring the possibility space and the game experiences become richer (McLean, 2002). Emergent games also have high replayability as each time the player plays the game they make different decisions, which change the game as a whole and result in different possibilities for action (Garneau, 2002).

The major difference between scripting and emergence is that emergence focuses on what the player wants to do, whereas scripting focuses on what the designer wants the player to do (Smith, 2001). However, it is important to realise that emergence alone isn't a game (Church, 2002). Emergence in games needs to be used to improve gameplay, not simply for its own sake.

TECHNIQUES FOR SCRIPTING AND EMERGENCE IN GAMES

The techniques that are used to implement the game environments, objects and agents define whether the system will be static and scripted or dynamic and emergent. Techniques that require everything to be built into the system in development, with no room for adaptation or unexpected behavior, can only facilitate a system that behaves as it is told to behave. On the other hand, techniques that are given the boundaries for behavior (rather than the script) or are able to grow and change have the potential to give rise to behavior that may not have been foreseen (or expected) by the developers. This section describes several techniques that have the potential to be used in games for implementing scripted or emergent games, or aspects of games, with the considerations for using each technique.

Techniques for Scripting Game Worlds

Scripted systems are custom coded for specific reactions to complex inputs for various localized situations in a game. The majority of current games are designed with this approach and there are two main techniques that are used for implementation, scripting and finite state machines. Almost every commercial computer game uses scripting or state machines for some, if not all, of the game system.

Finite State Machines

A finite state machine (FSM) is a device that consists of a set of states, a set of input events, a set of output events and a state transition function, which takes the current state and an input event and returns the new set of output events and the next state. The purpose of an FSM is to divide a game object's behaviour into logical states so that the object has one state for each different type of behaviour it exhibits (Rabin, 2000).

FSMs are by far the most popular technique in modern games, as they are simple to program, easy to understand and debug, and general enough to be used for any problem (Rabin, 2002). FSMs are amongst the simplest computational devices and provide a large amount of power relative to their complexity. Consequently, FSMs are ideal for the conditions of game development, which involves limited computational resources, as well as limited development and testing time. Some problems with using FSMs are that they tend to be poorly structured with poor scaling, so that they increase in size uncontrollably as the development cycle progresses. As a result, FSM maintenance can be very difficult and game FSMs that are not well planned and structured can grow out-of-hand quickly.

Scripting Languages

Scripting languages are designed to simplify some set of tasks for a game and hide many complicated aspects (Berger, 2002), thus allowing non-programmers, such as designers and artists, to write script for the game. Scripting languages for games, such as Quake's QuakeC or Unreal's UnrealScript, allow game code to be programmed in a high-level, English-like language (LaMothe, 1999), which is used to control the game engine from the outside. The scope of a scripting language can vary significantly depending on the problems it is designed to solve, ranging from a simple configuration script to a full-blown runtime interpreted language (Poiker, 2002).

Scripting languages are ideal for games as they are suitable for non-programmers, such as designers, artists and end users. During development, the designers use scripting to implement stories (Poiker, 2002), while artists use scripting to automate repetitious tasks, do things that the computer can do better than humans and add new functionality (Stripinis, 2001). After the game is shipped, "mod" groups and hobbyists write scripts if the scripting system has been exposed to the public (Poiker, 2002). Also, scripting languages are generally separate from the game's data structures and codebase and thus provide a safe environment for non-programmers and end users to make changes to the game, so that bugs in the script will not cause the game to crash. However, as with FSMs, scripting languages are deterministic and they require the game developer to hard-code character behaviour and game scenarios. Therefore, the developer must anticipate and hard-code each of the player's possible situations, making the game predictable and linear.

Techniques for Emergence in Game Worlds

Emergent behavior occurs when simple, independent rules interact to give rise to behavior that wasn't specifically programmed into the system (Rabin 2004). Techniques that can be used to facilitate emergence come from complex systems, machine learning and artificial life. Some examples of these techniques that can and have been used in games are flocking, cellular automata, neural networks and evolutionary algorithms.

Flocking
Flocking is a technique for simulating natural behaviours for a group of entities, such as a herd of sheep or a school of fish (Grub, 2003). Flocking was devised as an alternative to scripting the paths of each entity individually, which was tedious, error-prone and hard to edit, especially for a large number of objects. Flocking assumes that a flock is simply the result of the interaction between the behaviours of individual birds. In flocking, the generic simulated flocking creatures are called boids. The basic flocking model consists of three simple steering behaviours, separation, alignment and cohesion, which describe how an individual boid manoeuvres based on the positions and velocities of its nearby flockmates. Separation enables the boid to steer to avoid crowding local flockmates, alignment allows the boid to steer towards the average heading of local flockmates and cohesion makes the boid steer to move toward the average position of local flockmates (Reynolds, 2003). Each member in the flock revaluates its environment at every update cycle, which reduces the memory requirements and allows the flock to be purely reactive, responding to the changing environment in real time.

Flocking has been successfully used in various commercial games, including Half-life, Unreal, Theme Hospital and Enemy Nations, as it provides a powerful tool for unit movement (Johnson & Wiles, 2001) and for creating realistic environments the player can explore (Woodcock, 2003). It is a relatively simple algorithm and only composes a small component of a game engine. However, flocking makes a significant contribution to games by making an attack by a group of monsters or marines realistic and coordinated. It therefore adds to the suspension of disbelief of the game and is ideal for real-time strategy or first-person shooter games that include flocks, swarms or herds.

Cellular Automata
Cellular automata (CA) are widely-used techniques in the field of complex systems, which studies agents and their interactions. A traditional CA is a spatial, discrete time model in which space is represented as a uniform grid (Bar-Yam, 1997). Each cell in the grid has a state, typically chosen from a finite set. In a CA, time advances in discrete steps. At each time step, each cell changes its state according to a set of rules that represent the allowable physics of the model. The new state of a cell is a function of the previous state of the cell and the states of its neighbouring cells. A CA can be represented in one, two or more dimensions. A one-dimensional CA consists of a single line of cells, where the new state of each cell depends on its own state and the state of the cells to its left and right. In a two-dimensional CA, each cell can have four or eight neighbours, depending on whether cells diagonally adjacent to a cell are considered neighbours. CA have been proposed as a solution to the static environments that are prevalent in current computer games (Forsyth, 2002). The use of CA could lead to more dynamic and realistic behaviour of many game elements that are currently scripted, such as fire, water, explosions, smoke and heat.

A variation of CA, influence mapping, is a method for representing the distribution of power within a game world in a two-dimensional grid (Rabin, 2004). Influence maps are commonly used for strategic assessment and decision-making in games (Sweetser, 2004a), but were also used in the game SimCity to model the influence of various social entities, such as police and fire stations around the city (Rabin, 2004).

Neural Networks
Neural networks are machine learning techniques inspired by the human brain. Neural networks are comprised of artificial neurons, called units, and artificial synapses, called weights. In a neural network, knowledge is acquired from the environment through a learning process and stored in the network's connection weights (Haykin, 1994). The network learns from a training set of

data by iteratively adjusting its weights until each weight correctly reflects the relative influence that each unit has on the output. After training is complete, the network is ready to be used for prediction, classification or decision-making.

Considerations when developing neural networks for games include which variables from the game world will be used as input, the design of the structure of the network, what type of learning will be used, and whether learning will be conducted in-game or during development (Sweetser, 2004b). If the neural network is allowed to learn during the game then it will be able to dynamically build up a set of experiences and adapt to new situations and the human player as the game progresses. Alternatively, training the neural network during development will produce a network that will behave within expectations and require minimal resources. Overall, advantages of neural networks include their flexibility for different applications, their ability to adapt when trained in-game and the efficiency of their evaluation once trained. However, neural networks can also consume a lot of resources when training, can require substantial tuning to produce optimal results and can learn unpredictable or inaccurate information if trained incorrectly.

Evolutionary Algorithms

An evolutionary algorithm (EA) is a technique for optimization and search, which evolves a solution to a problem in a similar way to natural selection and evolution. An EA's similarities to nature include the use of a population of possible solutions to a problem, referred to as chromosomes, as well as processes that evaluate each chromosome's fitness and select which chromosomes will become parents. Additionally, the chromosomes that are selected to be parents take part in a process similar to reproduction in which they generate new offspring by exchanging genes. The new offspring also have a chance that they will mutate, similar to natural mutation. As the cycle continues over time, more effective solutions to the problem are evolved.

Considerations that need to be made when designing an EA for a game include the many parameters that need to be tuned, such as choice of a suitable representation, population size, number of generations, choice of a fitness function and selection function, and mutation and crossover parameters (Sweetser, 2004c). There are many advantages to using an EA, as they are a robust search method for large, complex or poorly-understood search spaces and non-linear problems. An EA is useful and efficient when domain knowledge is limited or expert knowledge is difficult to encode as they require little information to search effectively. Also, they are useful when traditional mathematical and search methods fail. On the down side, an EA is computationally expensive and requires a lot of tuning to work effectively. In general, the more resources they can access the better, with larger populations and generations giving better solutions. However, an EA can be used offline, either during development or between games on the user's computer, rather than consuming valuable in-game resources.

CONCLUSIONS

The two extreme approaches to game design discussed in this paper ranged from hand-crafted, hard-coded, scripted environments to rule-based, general, emergent environments. An emergent approach to game design is significantly different from the current scripted approach to game design, in terms of modelling techniques, as well as the implications for developers and players. However, the two approaches are not mutually exclusive. Rather, scripting and emergence can be seen as two extremes of a continuum (Church, 2002; Smith, 2002).

Both extremes hold benefits and drawbacks for game developers, as well as consequences for the game players. At the specific, scripted end of the continuum, the developers must hand-craft, implement and test every aspect of the game individually but are able to keep full creative control and rest assured that the game won't break after release. With the scripted extreme, the players are often locked into playing the game in a predefined way, unable to express their own creativity and may encounter inconsistencies in the game world. At the other end of the continuum are emergent game worlds that simply contain general rules for how the environment, objects and agents will interact, and the specific behaviours and events emerge from the interactions of the general rules. However, emergence can be a disconcerting prospect for developers, who cannot be sure how the game will actually behave after it is released, and is a sandbox type environment even a game? The emergent extreme does, however, hold the potential for players to express their own creativity and for intuitive and consistent interactions to take place in the game world.

It seems that it is somewhere between these two extremes that the future of game development lies; that there needs to be the right combination of scripted, narrated gameplay and freedom to interact within the world. There needs to be some way to define the boundaries of action, moving the story forwards, but still letting the player do their own thing along the way. We suggest that a game world that facilitates emergent interactions, based on a technique such as cellular automata, can be used in conjunction with other more conventional techniques for gameplay, such as scripting, to allow the player sandbox-style interaction within the boundaries of a predefined story and game objectives. The ongoing research that we are conducting is aimed at developing such a game environment (Sweetser, 2005).

REFERENCES

Bar-Yam, Y. 1997. *Dynamics of Complex Systems*. Addison Wesley, Reading, MA.

Berger, L. 2002. "Scripting: Overview and Code-Generation." In *AI Game Programming Wisdom*, S. Rabin, ed. Charles River Media, Inc, Hingham, MA.

Church, D. 2002. "Simulation, Emulation, and the Game Design/Development Process." Presented at *Australian Game Developers Conference*, Melbourne, Australia, 6-8 December.

Forsyth, T. 2002. "Cellular Automata for Physical Modelling." In *Game Programming Gems 3*, D. Treglia, ed. Charles River Media, Inc, Hingham, MA

Garneau, P. 2002. *Emergence: Making Games Deeper*. Available online at http://www.pagtech.com/ Articles/Emergence.html.

Grub, T. 2003. *Flocking*. Available online at http://www.riversoftavg.com/flocking.htm.

Haykin, S. 1994. *Neural Networks: A Comprehensive Foundation*. Maxwell Macmillan International.

Hecker, C. 2000. "Physics in Computer Games". In *Communications of the ACM* 43, no. 7: 34-37.

Johnson, D. and Wiles, J. 2001. "Computer Games with Intelligence." In Proceedings of the 10th IEEE International Conference on Fuzzy Systems.

LaMothe, A. 1999. *Tricks of the Windows Game Programming Gurus*. SAMS.

McLean, J. 2002. *Conversations from GDC Europe: Bill Fulton, Zeno Colaco, Harvey Smith*. Available online at http://www.gamasutra.com/features/20020911/ mclean_01.htm.

Poiker, F. 2002. "Creating Scripting Languages for Nonprogrammers." In *AI Game Programming Wisdom*, S. Rabin, ed. Charles River Media, Inc, Hingham, MA.

Rabin, S. 2000. "Designing a General Robust AI Engine." In *Game Programming Gems*, M. DeLoura, ed. Charles River Media, Inc, Hingham, MA.

Rabin, S. 2002. "Implementing a State Machine Language." In *AI Game Programming Wisdom*, S. Rabin, ed. Charles River Media, Inc, Hingham, MA.

Rabin, S. 2004 "Common Game AI Techniques." In *AI Game Programming Wisdom 2*, S. Rabin, ed. Charles River Media, Inc, Hingham, MA.

Reynolds, C. 2003. *Boids*. Available online at http://www.red3d.com/cwr.

Smith, H. 2001. *The Future of Game Design: Moving Beyond Deus Ex and Other Dated Paradigms*. Available online at http://www.planetdeusex.com/witchboy/articles/ thefuture.shtml.

Smith, H. 2002. "Systemic Level Design." Presented at *Game Developers Conference*, San Jose, CA, March 21-23.

Stripinis, D. 2001. "The (Not So) Dark Art of Scripting for Artists." In Game Developer Magazine: 40-45.

Sweetser, P. 2005. PhD Thesis: "An Emergent Approach to Game Design – Development and Play". Available online, 15 May, at http://www.itee.uq.edu.au/~penny/publications.htm

Sweetser, P. 2004a. "Strategic Decision-Making with Neural Networks and Influence Maps." In *AI Game Programming Wisdom 2*, S. Rabin, ed. Charles River Media, Inc, Hingham, MA.

Sweetser, P. 2004b. "How to Build Neural Networks for Games." In *AI Game Programming Wisdom 2*, S. Rabin, ed. Charles River Media, Inc, Hingham, MA.

Sweetser, P. 2004c. "How to Build Evolutionary Algorithms for Games." In *AI Game Programming Wisdom 2*, S. Rabin ed.. Charles River Media, Inc, Hingham, MA.

Woodcock, S. 2003. *Games Making Interesting Use of Artificial Intelligence Techniques*. Available online at http://www.gameai.com.

BIOGRAPHY

Penelope Sweetser is a game designer at The Creative Assembly, Brisbane, Australia. She is also completing her PhD in emergence in games and lecturing game design at The University of Queensland, Australia. Her research interests include artificial intelligence, emergence and user-centred design in games.

Janet Wiles is Associate Professor in the Division of Complex and Intelligent Systems Research in ITEE at the University of Queensland. She studies complex systems with particular applications in biology, neuroscience and cognition. Insights from such systems contribute to game design by showing how local interactions in such systems can give rise to system-wide properties.