Complex Game Design Modeling

Viknashvaran Narayanasamy, Kok Wai Wong, Shri Rai, Andrew Chiou

Complex Game Design Modeling

Viknashvaran Narayanasamy [1], Kok Wai Wong[1], Shri Rai[1], Andrew Chiou[2],

[1] Murdoch University, School of Information Technology,
South St, Murdoch, Western Australia
viknash@hobbiz.com, k.wong@murdoch.edu.au, s.rai@murdoch.edu.au
[2] CQUniversity Australia, School of Computing Sciences,
Rockhampton, Queensland, Australia
a.chiou@cqu.edu.au

Abstract. This paper looks at the game design and engineering approach to model the game design. The game modeling framework discussed in this paper could be a systematic alternative for implementing in the game engine architecture. The suggested game modeling framework incorporates structural game component, temporal game component and boundary game component frameworks. It is suitable to model most complex games and game engines.

Keywords: game modeling framework, complex system, game component, game design and engineering approach

1 Introduction

Many fields of computer science and engineering have been studied as complex system and thus modelling is an important area of research in many applications. In computer games, applications of complex system modelling has been utilised to facilitate the modelling of Artificial Intelligence in agent-agent and human-agent interaction research [1]. Recent computer games have often made use of multi-agent technologies for modelling cooperation, teamwork and character behaviour as well [2] in agents. Models are idealisations of a system in which certain aspects of the system are captured and other aspects are ignored. As a model should not be as complicated as the phenomenon it purports to describe the best models are usually the simplest possible model of the system [3]. Models that increase the complexity of a system and models that need to be updated constantly are not desirable. The main difficulty in constructing a model is identifying the important aspects of the system without compromising on features that are required to regenerate the actual system with sufficient detail. Since all models are abstractions of the modelled entity, information will be lost when the modelling is performed. Thus a variety of models representing different views of the system are often needed. These types of models used are driven by what aspects of the system the definers of the model want to expose [4].

In the context of the Game Engine Architecture, three modelling frameworks are proposed. Each of these frameworks models Game Design from a different point of

view. The complete models can then be mapped onto elements in the Game Engine Architecture and be implemented easily.

2 Game Design and Engineering

This section introduces the Game Modelling and Designing and Engineering approach that is needed to design a game model that can be applied to the Game Engine Architecture so that a game can be built.

In the interest of clarity, this approach is best described from left to right. The High-Level Architecture (HLA) consists of the Game Modelling Framework and the Game Engine Architecture. The Game Modelling Framework itself consists of three Game Component Frameworks. The Game Modelling Framework is used to refine the above mentioned player-driven [5] Game Design into a more concrete and formalised model for applying a Game Design.

The complete Game Model would then be the result of numerous iterations and proto¬typing efforts, made with inputs from player-driven requirements and Game Design goals. The Game Architecture, which is described in this section and in the completed Game Model, is used to systematically develop the Game Engine in the Game Engineering phase. The high-level architecture is the result of the application of the Game Modelling Framework (consisting of the Game Design and the Game Model) and the Game Architecture. From then on the Game Engine can be configured and extended for different games by using a data-driven approach. Both the player-driven and data-driven approaches offer a greater level of flexibility to the Game Designer.

3 Modelling the Game as a Complex System

In [6], the authors define a system as a set of parts which are in a relationship to each other to create a complex whole. In that sense a system would fundamentally consist of four components, namely, objects, attributes, internal relationships and the environment. The game itself will be represented by a set of states, for which transition functions are required to move from one state to another. [7] describes games as a container of objects which change their state during the play, where the evolution of their state is governed by the rules and influenced by the players or other objects. These two definitions provide an abstract yet accurate representation of a simple game.

In this abstract representation, games consist of Game Objects that are part of a system that must contain attributes to define the properties of the objects; internal relationships to define the behaviour of objects and between objects; and the environment to govern objects with rules. Each game system can embody a control system. Each system can be a subsystem of a larger system. This will require hierarchical ordering of a number of state machines within game systems as well as composition of a number of control systems. The composition of a number of control

systems will unintentionally turn into a complex system problem, as more and more Game Objects are added.

It is also important to note that games are not collections where objects can be added, removed or modified without affecting the relationships of other objects or the behaviour of the whole system [8]. As the majority of objects in an artificial environment respond better to discrete input it may be good to model games as discrete control systems.

Discrete control systems can be defined by making use of finite state automata, which can be represented by a triple (Q, Σ, δ) where [9]:-

Q – defines a finite sets of states
Σ - defines a finite set of input symbols
δ - defines a mapping function that maps the current state to the state given a finite set of input symbols.
i.e. $\delta : Q \times E \to Q'$

For each $q \in Q$, there exists a set of choices, such that elements $\sigma \in \Sigma$ are defined for $\delta(q, \sigma)$. As games tend to be mechanically deterministic and have a deterministic number of states [10], it is only necessary to model the controlled evolution of the state transitions in the game with the partial function δ [11].

It is also beneficial to look at this finite state automata representation from a different perspective that involves representing the list of states or inputs as strings. The string representation of sets of states simplifies the process of defining the allowable set of combinatorial actions on sets of states.

Let Σ^* denote the set of all finite strings obtained by concatenating elements in Σ, including the empty string ε.

As the concatenation of strings operation can be considered a map from $\Sigma^* \times \Sigma^*$ to Σ^*, Σ^* can be considered a monoid.

As the concatenation of strings is an associative operative operation, ε can be considered the monoid identity since it satisfies $s \cdot \varepsilon = \varepsilon \cdot s = s$ for any $s \in \Sigma^*$.

From basic automata theory, δ defines a unique partial map $Q \times E^* \to Q$, with the following properties:

$$\delta^*(q, \varepsilon) \to q$$
$$\delta^*(q, \sigma_1\sigma_2) \to \delta^*(\delta^*(q, \sigma_1)\sigma_2)$$

From this, it can be concluded that δ^* is a partially defined action of the monoid Σ^* on the set Q.

Σ^* can also be represented as a disjoint union as follows:

$$\Sigma^* = \coprod_{n \in N_o} \Sigma^n$$

Given a set of n elements, $\{1,2,3,.....,n\}$, the map u : $\{1,2,3,.....,n\} \to \Sigma$ defined by u(i) = σ_i for i=1,2,3,…n, will be ($\sigma_1, \sigma_2, \sigma_3, ...\sigma_n \in \Sigma^n$.

The concatenation of strings and maps are related as follows:

$$\Sigma^n \times \Sigma^m \to \Sigma^{n+m}$$

$$(u(i), v(i)) \text{ a } (u.v)(i) = \begin{cases} u(i) & if \quad 1 \le i \le n \\ v(i-n) & if \; n+1 \le i \le n+m \end{cases}$$

However, the above operation is only well-defined in discrete systems where n is finite. This is because the concatenation operation n+m will not be possible if n is not finite. Also, since $u \cdot v \in \Sigma^{n+m}$, n+m and consequently m has to be finite.

In reality, however, control systems can either be discrete or continuous. The modelling of continuous control systems is similar to that of discrete systems, as they can be both modelled after monoids. However, the state space and input space of continuous control systems are modelled using smooth functions instead of discrete values [11]. Continuous control systems can also work with variables that are not finite.

It has been assumed that games have an unknown but finite number of states. The introduction of continuous input variables can introduce useful emergent behaviour. Continuous input variables can be introduced by making use of higher-level AI implementations such as Fuzzy Finite State Machines and Neural Networks. To allow for the emergence of emergent behaviour the game is best modelled after hybrid control systems that handle both discrete and continuous input.

The first step in modelling the game as a complex control system involves using a higher-level abstraction of control systems that is common to both discrete and continuous control systems. This is required as the composition of a number of discrete control systems can cause the control system of the game to exhibit properties that more closely resemble continuous control systems, especially when the game rules change with respect to time. Furthermore, a higher-level abstraction of a control system such as that based on abstract control systems can also be the basis upon which hybrid control systems can be built upon in future [11]. An abstract control system can be represented as follows:

A game utilising an abstract control system can be modelled as a triple (S, M, Φ).

S – is a Set representing the state space of the game, which is a collection of the states of all game objects.

M – is a monoid that represents a set of inputs.

Φ – is the abstract control system defines a map, that is, an action or partially defined action of the monoid M on the set S.

i.e. $\Phi : S \times M \to S$

Identity Property: ε represents a neutral element. So, $\Phi(s, \varepsilon) = s$

Semi-Groups: Given two consecutive inputs m_1 followed by m_2, and then applying the inputs in that particular order is the same as applying input m_2 to the resultant state after m_1 has been applied.

$\Phi(s, m_1 m_2) = \Phi(\Phi(s, m_1), m_2)$, where $s \in S$ and $m_1 \,\&\, m_2 \in M$

The evolution or state transition of the abstract control system can be defined as follows:

$$s \xrightarrow{m} s'$$

The formal method of using an abstract control system to define systems in games is useful in defining the states of Game Objects and the interconnection between state

spaces of Game Objects. However, this technique does not address the complexity issues involved in interconnecting various systems and subsystems of Game Objects.

4 The Game State

Playing a game can be described as making changes in quantitative game states, where each state is a collection of all values of all Game Elements and the relationships between them [12]. Game play can then be described as an action by which quantitative changes can be made to game states. Players would then be influencing the game state by performing actions on Game Elements to achieve goals in the game.

More importantly, the game state is not necessarily the same as the state of the running game program. The state of a program can include the state of all hardware register values, the state of data structures, the state of the rendering subsystem, etc. The states of these items and many other items are crucial for the proper functioning of the Game Engine and consequently the game while it is running. However, in a high-level architecture, the implementation details and the states of low-level platform-dependent features are irrelevant in analysing the state of the game.

The game state in this context will refer to the states in finite state machines (FSMs) and their variants in each object. It will also refer to the state of data in components of data-driven architectures; the running state of processes, threads and micro-threads used by a game; the state of responses to high-level function calls and events; and the mode of the game. Generally, this can be termed the game play state of a game.

The game when viewed as a system is made up of objects. Objects are made up of attributes and their functionality is described by behaviours. Attributes can be used to determine the result of behaviours while the relationships between objects can be determined by their behaviours [13]. There are three fundamental parts to defining a valid and functional Game Element. They include the definition of the Game Object; the presence of at least one control system to control the behaviour of an object; and progress conditions that can introduce cumulative emergent behaviour.

5 The Game Modelling Framework

The Game Component Framework as proposed by [12] consists of four possible categories, by which the activity of playing a game can be viewed. These categories are referred to as components and from the players' perspective represent the action of playing a game. [12] mentions that all four component frameworks are essential for describing the game. A thorough analysis of these component frameworks was done and their relation to the proposed architecture was mapped. It was found that the fourth and final component framework, i.e. the Holistic Component, was not useful as a tool to map informal techniques in Game Design to a formalised Game Architecture as it contrasts the activity of game play with other external activities, which is not relevant considering the design of a Game Engine Architecture.

The Game Modelling Framework was derived from the Structural, Temporal and Boundary Game Component Frameworks. As such, the following sections will describe three approaches for which elements in a game can be modelled and translated to a form that can be implemented in the Game Engine Architecture. The Structural Game Component Framework presents an opportunity to model Game Design Elements using a control view from a Complex Systems perspective. The Temporal Game Component Framework, on the other hand, enables Game Design Elements to be modelled and implemented using a bottom-up approach where the individual components are modelled before their higher-level structures. The Boundary Game Component Framework, in contrast to this however, models the rules and constraints of actions in the environment.

6 The Structural Game Component Framework Mapping

The Structural Game Component Framework consists of five essential components: Game Elements, Game Time, Players, Interface and the Facilitator. Games Design can be broken down into the following components and the components can be implemented by mapping the functionality of the components to their counterparts in the Game Engine Architecture:

Game Elements – In the Structural Game Component Framework, Game Elements are the physical and logical components that contain the game state, which are an abstract representation of the actual Game Object in the human domain. Game Elements will contain both attributes and actions. Attributes define the object types of Game Elements and the Ownership of a Game Element. In the Framework this would refer to the controller of a Game Element. In the Game Engine Architecture this will refer to the Game Object that has privileged and/or exclusive rights to the Game Object that implements the Game Element. Attributes can also be numerical attributes that are used in algorithms to determine the outcome of actions. The Environment in the Game Engine Architecture is a special Game Element that defines the spatial world in which other Game Elements reside.

Game Time – Game Time determines how changes in the game state (the progress of game) relate to real time. A change can be discrete or continuous and can be different for different modes of play. In the Game Architecture, the Game Objects are event-driven and react to messages passed to them by the scheduler or other Game Objects. Ultimately, all Game Objects rely on the clock module in the scheduler to generate periodic or time-based offset events. True representations of time can be simulated using the Scheduler. To make use of the Structural Game Component Framework, a time-dependent order of allowable actions has to be modelled so that it can be translated to a series of periodic events and implemented in the Scheduler. If strict time-based ordering is not required, then event-based actions can be modelled directly into the state machine representation using the temporal Game Component Framework.

Interface – The interface provides information to the players regarding the game state. It does this by providing an access point where players can query and perform available actions. The interface can also help to express the theme of the game, or

rather define a protocol for interaction between heterogenous Game Elements. External interfaces are input devices like the keyboard or the mouse. However, their internal representations as input events are of more importance in this framework. An example of an interface in a game is the play grid in a turn-based strategy game. The Game Objects on the grid would then partially represent the Game State.

Players – Players can be defined as representation of entities that are trying to achieve the goals in the game. Players change the game state through actions. Players can either be human-controlled Game Entities or computer-controlled agents. They can either compete or cooperate with other players or agents.

Facilitator – The facilitator takes care of setting up the game and synchronises the game state and maintains the game time. Facilitators can also be ultimate arbitrating entities between the players and the game system. In the Game Architecture, facilitators are available to synchronise the Game States of client and server machines in MMP Games and all client nodes in a DIS. Facilitators in the form of Overseers are also available to arbitrate the allocation of system and Game Resources.

7 The Temporal Game Component Framework

The Temporal Game Component Framework was designed so that the flow of the game can be easily represented. The causality and the action-consequence behaviour of the game are of particular interest in this framework. This framework de-emphasises rule making and does not take into consideration the reasons for players' actions. In the Game Engine Architecture, the Temporal Game Framework most closely represents the microscopic views of state machines that exist for each and every Game Object in the Game Environment.

Actions will define the actions that are provided and the actions that can possibly be taken at particular states in the game. Actions in the context of this framework would mean through which players can make changes to the Game State. Actions can either be implicit or explicit and are associated with the interface provided for each player. Implicit actions are actions taken to find the actions that are possible. Actions in the Temporal Game Component Framework map onto the actions that are performed by a Game Object in response to an input event.

In contrast to Actions, Events are means by which the players can be informed of the consequence of their actions. In a way, events are the output of Game Objects and form part of the Game States perceivable by the player. Additionally, events can show the events generated by other players' actions. Events can also be generated by randomness, algorithms, etc. In the Game Engine Architecture, Events would translate to the Event-Based Messages generated after a successful change in state due to Actions taken by the Game Object.

Closures – Closures are quantifiable meaningful player experiences usually associated with changes in game state. Closures occur at points in time when a goal is met or when an end state is no longer reachable. Closures are not always associated with a milestone change in game state or significant progress. Closures can form on players' subjective experiences. Game state closures are usually non-reversible.

End Conditions – End Conditions define the set of conditions that need to be met for either a switch in mode of play, the completion of a closure, the end of a game instance, the end of a game or the end of a play session.

Evaluation Functions – Evaluation Functions are algorithms used to determine the outcome of end conditions or closures. Deterministic evaluation functions are desirable as they lead to consistent results.

By streamlining the list and sequence of allowable actions for each Game Element, a well-defined temporal structure can be defined in the Temporal Game Component Framework. This structure can then be translated and implemented in the event-based state machine in the Game Engine Architecture.

8 The Boundary Game Component Framework

The Boundary Game Component Framework was designed to implement constraints in the game to limit the activities that can be performed in a game. This can be done in several ways. It can be done by allowing only certain actions (i.e. by negative constraints) or it can be done by making certain activities more rewarding (i.e. by positive constraints). It can also be done by establishing social contracts between the players which have to be satisfied while playing through a set of limitations or by making use of rules of play [6], where boundaries between allowed and forbidden actions are clearly defined and enforced.

Rules – Rules are used extensively in this framework. They govern how components interact and determine the actions that are allowed, and the order in which they are allowed. They can also determine the choice and order of actions as they can limit a player's range of actions. Rules are also used to define and describe boundaries and govern how all other components are instantiated.

Although rules are necessary to define the Game Environment, their extensive use is undesirable in successful emergent systems. Rules can be classified into three categories [6]: Operational Rules are rules that define how the game is played by players. They also define how the game corresponds to explicitly written rules by players and define the physical implementation of constitutive rules. Constitutive Rules are more abstract and define the underlying formal structures that exist beneath operational rules. The last classification of rules is Implicit Rules. As Implicit Rules deal with the human and social aspects of playing the game, they will not be included in the design of the framework. These rules include Etiquette, Sportsmanship, etc. In Multi-tiered AI Frameworks different rule types are used for different tiers in the AI Architecture.

There are other rules that differ between game instances and these are usually optional. House rules are rules agreed to by players before the start of the game (e.g. Game Play Options for multiplayer maps in Real-Time Strategy Games, such as "Auto respawn", are one such example). These rules balance the game and introduce variance.

Non-Static Rules in games allow rules to change. Such rules allow for games where the game play is about changing rules. Such rules require high-order rules that are rules created to govern non-static rules.

Goals – Goals set predefine tasks for players to achieve. In the Game Engine Architecture goals are represented as favourable game states set for individual players to achieve. Goals give players motivation for their actions. Sub-goals can be introduced, too, if the process of achieving a goal is too lengthy. Diagrammatically, this can be represented by Composite and Sub-State machine. In some games the process of identifying goals can be part of a game, and in such games identifying a goal can in itself be a goal.

Modes of Play – Modes of play refers to sections of the game where perceivably different types of activities take place. For example, this can be in the form of having sub-games within games. These are considered sub-modes of play. Different players can have different action sets and each action set is associated with different modes of play. Role reversals can also be considered as a different mode of play.

Table 1: Rule types in a Multi-tiered Framework

	Operational Rules	Constitutive Rules	Implicit Rules
Strategic Intelligence		✓	✓
Operational Intelligence	✓		×
Tactical Intelligence	✓		×
Individual Unit Intelligence	✓		×

Creating a game by making use of the Game Modelling Framework for the Game Engine Architecture requires the modelling of the Game using the three Game Component Frameworks. Firstly, structural modelling will have to be done on the individual Game Element using the Structural Game Component Modelling Framework and this will have to be implemented as a Game Object in the Game Architecture. Secondly, the state machine, (i.e. the states of the game) will have to be accurately modelled using the Temporal Game Modelling Framework and the abstract control system model for representing the game. The Boundary Game Component Framework can then be used to design the rules needed to govern the behaviour of Game Entities. This can be done by implementing the Boundary Game Components as Overseers in the Overseer Tier or other higher-level AI mechanisms. . A high fidelity prototype based on a sub-set of the framework presented in this paper has been developed by Chiou [14].

9 Conclusion

This paper has introduced the Game Design and Engineering approach to model a Game Design and implement it in the Game Engine Architecture. It has also shown how the Game can be modelled as a system of systems within the Game Component Framework. The Game State was modelled using Finite State Automata. This

formulation can prove useful in consolidating and validating the list of possible Game States.

The Game Component Framework has also shown how a Complex system of systems can be achieved by mapping the various objects and associated game states by means of an abstract control system. If successfully applied, this methodology will be able to simplify the content generation task by allowing designers to define simple game objects and the rules to create more complex objects from simple game objects. On the other hand the Game Component Framework has shown how complex rules can be broken down into simpler rules that are manageable by entities in the game.

The suggested Game Modelling Framework incorporates the Structural Game Component, Temporal Game Component and Boundary Game Component Frameworks. By making use of a combination of these three frameworks, the Game Design can be formally captured into a format that can be easily translated to be implemented in the Game Engine Architecture.

References

1. Bezek, A.: Modelling Multiagent Games Using Action Graphs, www.cs.biu.ac.il/~galk/moo2004/proceedings/poster/1.pdf (2004)
2. Orkin, J.: Constraining Autonomous Character Behavior with Human Concepts. In AI Programming Wisdom 2, Charles River Media. (2005)
3. Wolfram, S.: A New Kind of Science. Wolfram Media, Champaign, IL, USA. (2002)
4. Little, R.: Architectures for Distributed Interactive Simulation, Software Engineering Institute, http://www.sei.cmu.edu/architecture/Architectures_for_DIS.html
5. Gaash, A.: Asian Game Markets and Game Development - Mass Market for MMP Games. In Massively Multiplayer Game Development 2, T. Alexander ed., Charles River Media, Hingham, MA, 481-494 (2005)
6. Salen, K. and Zimmerman, E.: Rules of Play - Game Design Fundamentals. The MIT Press, Cambridge, MA, USA (2004)
7. Grunvogel, S.: Formal Models and Game Design. Game Studies : The international journal of computer game research 5 (1) (2005)
8. Fullerton, T., Swain, C. and Hoffman, S.: Game Design Workshop - Designing, Prototyping, and Playtesting Games, CMP Books (2004)
9. Cassandras, C. and Lafortune, S.: Introduction to Discrete Event Systems. Kluwer Academic Publishers, Boston, MA, USA (1999)
10. Crawford, C.: The Art of Computer Game Design. McGraw-Hill Osborne Media.
11. Tabuada, P., Pappas, G. and Lima, P.: Compositional abstractions of hybrid control systems. In Proceedings of the 40th IEEE Conference on Decision and Control, Orlando, Florida (2001)
12. Bjork, S. and Halopainen, J.: An Activity-Based Framework for Describing Games. In Patterns in Game Design, Charles River Media, Hingham, MA, USA (2004)
13. Barry, I.: Game Design. In Introduction to Game Development, S. Rabin ed., Charles River Media, Hingham, MA, USA (2005)
14. Chiou, A.: A game AI production shell framework: generating AI opponents for geomorphic-isometric strategy games via modeling of expert player intuition, Australian Journal of Intelligent Information Processing Systems, vol. 9, no. 4, pp. 50-57 (2008).