FORMAL MODELS AND GAME DESIGN

STEFAN M. GRÜNVOGEL

Abstract. In this article results from mathematics are used to create a formalism for games. Games are considered as systems and the design of games as the creation of models for games. By abstract control systems, a formalism for describing models of games is introduced. Methods to create new models from given ones are described. To handle complexity problems in game design, simulations of models by other models are explained.

1. Introduction

The design of computer games is a highly artistic undertaking, which needs a lot of experience from the designer. Game design as a craft has created a vast diversity of methodologies to balance interaction, game mechanics and audio-visual presentation for different game genres and players. But there are only very few attempts to support this process by using formal methods. Recently techniques from object oriented design are reported to be used in interaction design (cf. [Bethke, 2003]). Semi-formal approaches as the 400 design rules project [Falstein, 2002], formal abstract design tools [Curch, 1999] or game design patterns [Björk et al., 2003] try to support the creation of games. In the area of computer science, a recent approach was reported in [Natkin and Vega, 2003] where Petri-Nets are used to describe quests in a combined formal and graphical way.

Why should one look at game design with formal methods? There can be stated several reasons for this. Formal models can be used to create a language for certain aspects of the design process, which is mathematical precise. Mathematical precision is not important for its own sake, but can be used to detect connections between game elements. Similarities between different games can be made apparent with the right tool and be used to build an asset of recurring patterns.

In the following, we will introduce the concept of formal models by using abstract control theory to describe games and we will state the benefits and the limitations of this formalism.

2. Games are systems

Although there are a great number of different definitions of games (cf. [Juul, 2003] or [Salen and Zimmerman, 2004]), most of these definitions have one thing in common: they consider games as a kind of *system*. The problem with these definitions is, that mostly they do not define the term system any further. An exception to this practice can be found in [Salen and Zimmerman, 2004], where a system is defined as a set of parts which are in relationship to each other to create a complex whole. A system consists of four components, namely objects, attributes, internal relationships and the environment.

Date: *April 16, 2004.*

Most of the definitions of games also include the concepts of *rules* (in fact, a game definition *without* including rules can be found in [Costikyan, 1994]). The rules define the systems and therefore the game. Because rules are so important for games, further classifications of rules are crucial. In [Salen and Zimmerman, 2004] rules are subdivided into three groups: constituative, operational and implicit rules. Implicit rules are the unwritten rules of social behaviour (netiquette) between players. We will not consider implicit rules in this article. The constituative rules are the abstract, core mathematical rules of the game. The operational rules are the rules of play, they direct the players' behaviour and interaction with the game.

Another view on game rules can be found in [Järvinen, 2003], where a typology of games is given. He identifies five basic elements which belong to games: components, procedures, environment, themes and interface. Rules are categorized into rules for these elements, i.e. component rules, procedure rules etc. The relation of the player to the game is also divided into four categories (core, spatial, rhetorical and physical) and the rule categories are assigned to these layers.

3. Games represented as abstract control systems

As we have seen before, it makes sense to look at games as systems. What are the mathematical instruments to represent games and rules? A classical approach is by mathematical game theory, founded by John v. Neumann and Oskar Morgenstern (cf. [Neummann and Morgenstern, 1944]). This formalism has a long tradition and holds many approaches to understand games. But we will choose a different more general approach to look at games in a formal way.

If one forgets for a moment about the social, psychological and cultural aspects of a game, then it consist 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. Thus, games can be described as *abstract control systems*, defined like in [Tabuada et al., 2002]:

Definition 1. *A* game *is a triple* $(S, \mathcal{M}, F)$*, where* S *is a set,* $\mathcal{M}$ *is a monoid and* F *is a (possibly partially defined) action of the monoid* $\mathcal{M}$ *on the set* S*, that is, a map* $F : S \times \mathcal{M} \to S$ *satisfying:*

- Identity: $F(s, \varepsilon) = s$ *for all* $s \in S$ *and* ε *the neutral element of the monoid* $\mathcal{M}$*.*
- Semi-group: $F(s, mn) = F(F(s, m), n)$ *for all* $s \in S$ *and* $m, n \in \mathcal{M}$*.*

We usually denote an abstract control system as F *or* F^S *if we wish to emphasize the set* S*. We also represent by* $s \xrightarrow{m} s'$ *the evolution of the state* s *under the action* m*.*

We will not go deeper into the explanation of the mathematical terms, but assume that everybody has some basic understanding, what sets and maps between sets are. The monoid $\mathcal{M}$ represents the input of the system, e.g. it can be used to model the input of the players. Bear in mind that we do not explicitly define a player, this may be a part of a concrete example for a game. The player may also be modelled as part of the state space S. The set S is in fact the state space and represents the states of the different game objects. The rules are defined by the action F which govern the states of the objects under the current state and the input. Note that we define the action F as time-independent, that means we assume here that the game rules do not change during the play. It is an easy task to

generalise the above definition to include time-dependent game rules but we will not pursue this path in this paper. In the *Identity* property the element ε is the neutral element of the monoid. The neutral element can be interpreted as empty input. The semi-group property means: Suppose our game is in the state s. Suppose that we can break an input p into two consecutive inputs m followed by n. If we apply the input p, then the resulting state of the system is the same, as if we would have applied the input n to the system in state $F(s, m)$.

Example 1. *Consider the following two-player game. There are two characters on a flat game field. Each player can direct the character to walk in a certain direction by joystick. There are barrels with toxic waste scattered around the field, and because the characters lover their environment, the task is to collect all the barrels. If all barrels are collected, the player with the highest number of barrels has won. But because the barrels contain toxic waste, the characters get weaker for each barrel they collect. Thus, if the two character are near enough, and one character has collected less barrels, it is stronger then the other and kills him - Game over. If the character meet and have the same strength, they start discussing about the weather, and the game is over with no winner.*

We now define the corresponding game $A = (S, \mathcal{M}, F)$. The state space S can be modelled as a set consisting of

- *the position of player one Pos_1 and player two Pos_2 on the field*
- *the number of barrels each player has collected Col_1, Col_2.*
- *the strength of each character Str_1, Str_2.*
- *the set of positions of the barrels on the field Bar.*
- *the set $Win = \{Player1, Player2, none, ongoing\}$ which indicates if a player has won the game (if any) or if the game is still in progression.*

The input $\mathcal{M}$ consists of the evolution of the directions of the joysticks over a fixed period of time. Note that we do not work here with discrete time, but instead with continuous time. The function F now maps a given state of the game (at a certain point of time) and the evolution of the joystick directions over a fixed period of time t to a new state of the game, i.e to the game state we reach if the two player play the games t seconds and apply the corresponding joystick directions during this time.

This also explains the identity *property of Definition 1: In our example the neutral element ε means the evolution of the joystick directions in zero seconds. This does not change the state of the game. Imagine, that we could freeze the time for a moment, then every manipulation of the joystick direction has no influence on the game. But as soon as the time goes forward, we actually get an evolution of the joystick directions, which changes the game state.*

So, can this be of any help to design new games and is this formalism actually strong enough to represent a given game? In fact, describing a game with this formalism seems to be a cumbersome task. In principle, computer games *could* be presented with this formalism. But even if one tries to formalise very simple games like Tick-Tack-Toe one encounters various difficulties. The top most difficulty is - complexity! It is a very hard task to detect all game objects, their state space and the interconnection between the state spaces (namely the rules) of a current computer game and write them down in a human readable format. Currently this is done in a semi-formal way by game design documents, which exist in various

forms and describe certain aspects of games. But they are at most *semi*-formal, i.e. the game developers have to create the code and programs from these descriptions. The interplay between game designer and developer assures, that game behaviour resulting from the programs and algorithms match the imagination of the designer. The resulting game engines and the programs are in fact one instance of the game the designer had in mind. The program *is* a model of the game. But this model is also very complex, (consisting of state machines, algorithms for artificial intelligence etc.). The source code itself represents the game at the very lowest level, but on the other hand it is too complex to reason about the game just by looking at the code. There are also other representations (or models) of the game engine (e.g. using UML diagrams) which leave out details and concentrate on certain aspects of the game. As a consequence one tends to say, that the actual implementation of a game is *one* instance of the game, and there exist certain kinds of representation describing games at different levels. But a different implementation of a game may result in the same game, i.e. one can exchange certain parts of the game engine, without affecting the resulting game at a macroscopic level. This leads to the question: what model is suitable for a game? But this is the *wrong* question!

In general there is *no* model of a game which is capable of representing every aspect of the game - because it *is* a model and has to leave out certain aspects of the game. Hence instead of asking for the ultimate model of a game, we are content with having one model which simulates certain aspects of a game.

4. Simulations of Games

What does it mean to simulate a game? One way of investigating models (and especially the ones given by Definition 1) and simulations is by using category theory (cf. [Joyal et al., 1994]). The idea is that simulations between two systems are homomorphism between the systems, preserving their structure. We will not go into technical details here, but the idea is the following: Every model of our game can be described as a system. Suppose we have two models of a game, given by the abstract systems $A_1 = (S_1, \mathcal{M}_1, F_1)$ and $A_2 = (S_2, \mathcal{M}_2, F_2)$. For A_2 being a *simulation* of system A_1 we need two maps ϕ and ω which together specify the simulation.

The map ϕ maps elements of the state space S_1 to the state space S_2. The map ϕ can be understood as a map which specifies, which state information is propagated from the original system A_1 to the simulation A_2. The other map ω maps pairs of states in S_1 and inputs $\mathcal{M}_1$ to elements of inputs $\mathcal{M}_2$. This means, that for each state-input pair (s_1, m_1) we have corresponding state-input pair (s_2, m_2) given by $(s_2, m_2) = (\phi(s_1), \omega(s_1, m_1))$. The map also has to respect the semi-group property of Definition 1. A_2 is a simulation of A_1, if the state of the system A_2 is the same if

- we play a game in system A_1 and map the resulting state to the simulation A_2 by ϕ, or
- we map the current state and input (s_1, m_1) of A_1 to a state-input pair (s_2, m_2) of A_2 and play the game A_2 with this pair.

Example 2. *We introduce now a simulation of Example 1. First we rename our system $A = (S, \mathcal{M}, F)$ with $A_1 = (S_1, \mathcal{M}_1, F_1)$. Next we have to define the components of our simulation $A_2 = (S_2, \mathcal{M}_2, F_2)$.*

We start by defining the state space S_2. Because the characters in the game have the same speed if they walk, we do not consider the game field as continuous area, but instead represented as set of connected squares. For simplification, we even assume, that all these squares together form a rectangle. Each square contains either the number 1 *or the number* 0*, indicating, if there is a barrel in this square or not. Thus the field can be represented as matrix $Field$ with n columns and m rows and elements* 0 *and* 1*, like e.g.*

$$Field = \begin{bmatrix} 0 & 0 & 1 & 0 & 0 \\ 1 & 1 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 \\ 0 & 1 & 0 & 0 & 0 \end{bmatrix}.$$

Thus we have combined a representation of the game field with the positions of the barrels. The positions of the character are represented by Ind_1 and Ind_2, where Ind_1 and Ind_2 are pairs like $[1, 4]$ indicating, that the character is placed in row 1 *and column* 4 *on the field. We overtake the number of barrels Col_1 and Col_2 from the model A_1 and do not measure the strength of the characters any more. We also do not model who has won the game and the strength of the characters.*

We also discretise the input of the joystick and the time. Instead of measuring the direction of a joystick in degrees, we assume, that we only can steer up and down, left or right, or diagonal. The direction Dir can thus be represented as a tuple $[vertical, horizontal]$, where $vertical$ and $horizontal$ can have the values $-1, 0, 1$. The input $\mathcal{M}_2$ consists of sequences of tuples of this form for each player. Thus e.g. the input $[-1, 1]$ means the character is moved one row down and on column left on the field.

Instead of defining the function F_2 for arbitrary sequences, we just show what happens to the state in one step, i.e. we define the rules. We assume that player one creates the input P_1 and player two P_2.

1. *If the matrix $Field$ consists only of* 0*, then nothing happens.*
2. *If $Ind_1 = Ind_2$, then nothing happens.*
3. *If $Ind_1 \neq Ind_2$, then $Ind_1 = Ind_1 + P_1$, $Ind_2 = Ind_2 + P_2$, (where we assume, that we can not leave the field). Furthermore, if at the new position of the characters the $Field$ has value* 1*, then the number of barrels of the character is increased by 1.*

Thus we have created a simplified game, which we actually could play by paper and pencil. We do not indicate any more, who has won the game. Also, the end of the game is only given implicitly be (1) and (2) - nothing happens any more on the field. In fact, the game can be played an unbounded period of time without changing the game state. This reflects the situation in the underlying system A_1, where also nothing happens any more as soon as the game is over.

So what can we do with this definition? As stated above, models' main purposes are to leave out certain aspects of complex systems to facilitate the study of these systems. Thus if we simulate a model A_1 of a game by model A_2, this allows us to study properties of the model A_1 by studying the properties in system A_2. Playing a game in our abstract sense means, that we consider the evolution of the states of game objects in model A_1 starting at some initial game state under the control of some input. If we map this evolution of the states (the *orbit*) into the simulation A_2 (i.e. we collect the information of the played game in terms of A_2), then the

resulting game states in A_1 are a part of the game states we would get if we played the game in A_2 starting at the corresponding game state. Thus playing games in the simulation A_2 will forfeit some information of the simulated game in A_1, but on the other hand may give some insight because of the simpler structure in A_2.

5. Designing Models

If one models a system (with whatever formalism or tool one uses) it is important to construct the *simplest* possible model for each system. Models are idealisations of a system, in which certain aspects of the system are captured and other aspects are ignored. In [Wolfram, 2002] basic principles for modelling are given, where the role of complexity for systems and models is stressed. For Wolfram it is typically not a good sign for a model, if it is almost as complicated as the phenomenon it purports to describe. Even worse is a model which needs to be updated constantly if the system reveals new aspects by observation. Good models are those who are simple yet still manage to reproduce even quite roughly a large number of features of a particular system. Translated into our objective, the main difficulty to construct a model is to identify the important aspects of the system.

We will not go into methodologies of game design e.g. the top-down approach which is reflected in different kinds of design documents [Laramée, 2002], or iterative approaches, or some other methods and approaches (like the The 400 design rules project [Falstein, 2002] , formal abstract design tools [Curch, 1999], game patterns [Björk et al., 2003]) which are actually used to create a *good* game with great balanced gameplay and forth. In fact, the above theoretical formalism inhabits no model for fun and does not prevent anybody from describing very bad games. But it can actually be used to create new games.

One of the simplest ideas to create a new game is to take two given games and play them in parallel, which we call the *product* of two games. There is no interaction between the games and every game can evolve independently. In the sense of Definition 1 this can be defined as the product of two abstract control systems. Given two models of games $A_1 = (S_1, \mathcal{M}_1, F_1)$ and $A_2 = (S_2, \mathcal{M}_2, F_2)$, the product of the two is defined as the system $A_1 \times A_2 = (S_1 \times S_2, \mathcal{M}_1 \times \mathcal{M}_2, F_1 \times F_2)$. For each game, every game state can be reached, as if we would play the game separately. But the games do not influence each other, and thus this does not seem to create really new games let alone interesting ones.

Example 3. *Consider the game $A = (S, \mathcal{M}, F)$ from the initial Example 1. The product of this game with itself $A \times A = (S \times S, \mathcal{M} \times \mathcal{M}, F \times F)$ is now a game which can be played by four players. It consists of two game fields, where on each game field, two players play against each other. A state of the game $A \times A$ is just the collection of individual states of each game. The input is the evolution of directions of the four joysticks over a time period.*

To get a fruitful basis for new games we introduce another very simple concept of game design, which starts from a given game. The idea is to create a new model of a game, by restricting the game states and the set of admissible inputs of a given model. Mathematically, this can be expressed by choosing a subset $L \subset S \times \mathcal{M}$. But it is not enough just to restrict the state spaces or the inputs of a given model to get a new model of a game. Generally the resulting system will not be a model of a game. In fact, to fulfil the properties of the Definition 1, the restricted state and input set L has to meet certain properties.

Suppose that we take a certain start state s and an input m from the restricted model. First of all we demand, that if we play the game with these two elements, the resulting game state will also be an element of the restricted system. Furthermore if we take any input m' which can be prolonged to the input m such that input (s, m') is also an element of the restricted system, the following holds: if we play the unrestricted game with this pair, the resulting game state is an element of the restricted system.

Example 4. *We consider the game A_2 from Example 2, and restrict the positions a character can move to. If the movement of a character is restricted to one row in the field, it can only move left or right on the game field. This is actually a correct restriction of the game as one can easily check.*

As we have seen, building the product of two models of games is a very simple concept to create new games, because the given games do not interact with each other. On the other hand, the resulting product model is not very interesting. But with the technique of restriction, we have a formal tool to express interactions between the two models by means of synchronisation. Just take the product of two models A_1 and A_2 (which is actually a model again) and restrict it, i.e. choose a subset of the products $(S_1 \times S_2) \times (\mathcal{M}_1 \times \mathcal{M}_2)$ and restrict the product system to this subset. The resulting system is a qualitatively new game, because by the restriction, playing this game is not only the parallel execution of the underlying games A_1 and A_2 but also has to respect the constraints of the restriction. We call this the *parallel composition with synchronization* according to the definition given in [Tabuada et al., 2002]. Thus with these approaches, new games can be constructed, by taking given ones and combine them in a meaningful way. This can be interpreted as a kind of bottom-up construction of games. Given some rules acting on some state spaces, we can now construct new rules acting on the combined state spaces. The interesting part is, that by parallel composition with synchronisation of 'simple' games, possible new and complex behaviour of the resulting game may emerge. Thus we do not have the problem from common top-down approaches to identify the state spaces of every game object and all admissible inputs beforehand, but instead can start from a small and manageable set of rules.

Example 5. *We construct a collaborative game for four players by taking the game A of Example 1. First we build the product game $A \times A$, which is not collaborative, because we have the situation that the two games are played in parallel without any interconnection between the players. Player 1 and 2 play the first game, and player 3 and 4 play the second game.*

Next we construct a new game out of this product game by restriction. We want, that player 1 and player 3 collaborate with each other, and that player 2 and player 4 collaborate with each other. Because the game fields of the first and the second game are identical, we can compare the position of the players between these fields. Thus we demand, that the distance between player 1 and player 3 on the field is not greater then some given distance D. We also demand, that the distance between player 2 and 4 also fulfil the same restriction. Thus as the characters 1 and 3 move on the field, they are chained together, and can not move independently. The same holds for the characters of player 2 and 4. We restrict the product state space $S \times S$ to those states, where these conditions holds.

But if we would allow every possible input of joystick movements, this constraint is easily broken, because nothing prevents player 1 to move his character as far away as he wants from the character of player 3. Thus, this is not *a correct restriction of the product game.*

To actually assure that the restricted system is a game in the sense of Definition 1, we also have to restrict the input $\mathcal{M} \times \mathcal{M}$*. Thus we choose the subset* $L \subset (S \times S, \mathcal{M} \times \mathcal{M})$ *such that the characters actually fulfil the distance constraints (it is possible to choose such a set). This means, that under this restriction not all joystick movements are allowed by the players. Of course, in reality one would not restrict the movements of the joystick, but instead modify the handling of the output of the joysticks.*

The techniques developed here also can be applied for the top-down analysis of games. Without going into detail, we state the basic idea. The task is here to divide a given model into different models, which together form by parallel composition the given model. Let A_1 and A_2 be the two submodels, which by parallel composition over synchronization set L form the model $A_1 \| _L A_2$. Suppose furthermore, that for the models A_1 and A_2 there exist two simulations B_1 and B_2. We then can construct an simulation Z from the composed system $A_1 \| _L A_2$ with these simulations (and the corresponding maps). But we also can view the simulations B_1 and B_2 separately and build an abstraction of the composed system $A_1 \| _L A_2$ by restricting the product system of B_1 and B_2.

$$\begin{array}{ccccc} B_1 & \times & B_2 & \longrightarrow & Z \\ \uparrow & & \uparrow & & \uparrow \\ A_1 & \times & A_2 & \rightarrow & A_1 \| _L A_2 \end{array} \tag{1}$$

Thus we can use this to construct models of games in the following way. We construct a new model (denoted by $A_1 \| _L A_2$ above) by composition of two given models A_1 and A_2. If for the two last models we also have simulations B_1 and B_2, we can construct a simulation Z of the composed model easily by using the simulations B_1 and B_2 separately and then restrict the resulting product system to Z. This is mirrored in the above diagram , as it shows the two ways from $A_1 \times A_2$ to Z.

6. The bridge between theory and practice

Is the sketched approach usable in practice? In fact the formalism developed above can be used to create new games from given rules. The abstract view on games by simulation enables the designer to produce different presentations of a game. The creation of complex games by composition can be facilitated by simulating the components separately. This all could help to formalize certain parts of game design.

Because it is a formalism, it is more precise to describe rules and games then natural spoken language. But there also lies the general drawbacks with formalisms. As they are not natural spoken language, they have to be learned, which assumes that a game designer is actually willing to spend some time learning this new formalism. The other drawback is, that it is *precise*, i.e. it forces the designer to be precise, too. But precision and design processes seem to be orthogonal concepts, as anybody knows who has ever programmed. Thus as a consequence, it is unlikely

that this concepts will be used directly by game designers to start to design a game from scratch (everybody is invited to do it anyway). But it could be used during different levels at the design process by creating new rules or by providing simulations of games. It also may more likely serve as a theoretical foundation for game and rule design, or be used to conceive creativity tools.

References

[Bethke, 2003] Bethke, E. (2003). Structuring game design elements. *Game Developer Magazine*. www.gamasutra.com.

[Björk et al., 2003] Björk, S., Lundgren, S., and Holopainen, J. (2003). Game design patterns. In *Level Up: Digital Games Research Conference*, pages 180–193.

[Costikyan, 1994] Costikyan, G. (1994). I have no words and I must design. *Interactive Fantasy*, 2. www.costik.com/nowords.html.

[Curch, 1999] Curch, D. (1999). Formal abstract design tools. *Game Developer Magazine*. www.gamasutra.com.

[Falstein, 2002] Falstein, N. (2002). Better by design: The 400 project. *Game Developer Magazine*, 9(3):26.

[Järvinen, 2003] Järvinen, A. (2003). Making and breaking games: A typology of rules. In Copier, M. and Raessens, J., editors, *Level Up: Digital Games Research Conference*, pages 68 – 79.

[Joyal et al., 1994] Joyal, A., Nielsen, M., and Winskel, G. (1994). Bisimulation from open maps. Technical report, BRICS.

[Juul, 2003] Juul, J. (2003). The game, the player, the world: Looking for a heart of gameness. In Copier, M. and Raessens, J., editors, *Level Up: Digital Games Research Conference*, pages 30–45.

[Laramée, 2002] Laramée, F. D. (2002). *Game Design Perspectives*. Charles River Media, Inc.

[Natkin and Vega, 2003] Natkin, S. and Vega, L. (2003). Petri net modeling for the analysis of the ordering of actions in computer games. In Mehdi, Q. and Gough, N., editors, *GAME-ON 2003, 4th International Conference on Intelligent Games and Simulation*, pages 82–89.

[Neummann and Morgenstern, 1944] Neummann, J. V. and Morgenstern, O. (1944). *Theory of Games and Economic Behaviour*. Princeton University Press.

[Salen and Zimmerman, 2004] Salen, K. and Zimmerman, E. (2004). *Rules of Play: Game Design Fundamentals*. Massachusetts Institute of Technology.

[Tabuada et al., 2002] Tabuada, P., Pappas, G. J., and Lima, P. (2002). Composing abstractions of hybrid systems. *Lecture Notes in Computer Science*, 2289:436–450.

[Wolfram, 2002] Wolfram, S. (2002). *A New Kind of Science*. Wolfram Media, Inc.

NOMADS Lab, Piusstr. 40, D-50823 Cologne, Germany, E-mail: gruenvogel@nomadslab.org