Running head: VIZUALIZING GAME DYNAMICS

Visualizing Game Dynamics and Emergent Gameplay

Joris Dormans, MA
School of Design and Communication,
Hogeschool van Amsterdam

Abstract

This paper aims to explore a method of visual notation based on UML (Unified Modelling Language) to help the game designer understand the dynamics of his or her game. This method is intended to extend and refine the iterative process of designing games. Board games are used as a case study because emergence in board games is often easier to study than in computer games. In order to understand emergence in games, some concepts from the science of complexity are discussed and applied to games. From this discussion a number of structures that contribute to emergence are used to inform the design of the UML for game design.

Introduction

There is beauty in games. For some the beauty of games is directly related to dazzling visuals or emotional emergence. Personally, I am fascinated by the rich gameplay so many games offer using only a handful of rules. The age-long tradition of *Go* is testimony to power and beauty of this richness. Every session of play becomes a performance; a ritualized dance that is focused and confined by the game's rules and premise, but which is never the same twice. Mine is an aesthetic appreciation of the freedom and possibilities set up within the logical game space within the magic circle of the game (*cf.* Huizinga 1997: 25).

Unfortunately this type of beauty is very hard to create. The richness and complexity of gameplay does not directly translate to detailed and complex rules. On the contrary, games that offer the richest experience often consist of the simplest rules. The gameplay potential offered by a classic game such as *Checkers*, or a simple computer game such as *Boulder Dash,* does not become immediately apparent from a study of their rules. On similar ground, how does one explain the enormous difference in gameplay potential of *Tic-Tact-Toe* and *Connect Four* when their rules do not seem to be so very different at all?

These days, many game designers agree that, at least in certain types of games, gameplay is an emergent quality of the game system (Bateman & Boon 2006, Adams & Rollings 2007, Fullerton 2008). Emergence is a term taken from the science of

complexity and used to indicate apparently non-deterministic behavior of a system consisting of many parts that cannot be directly related to the deterministic rules that govern the system. Classic examples include John Conway's 'Game of Life', a cellular automaton with only three rules that can produces chaotic, sometimes life-like, behavior (Ball 2004: 125-126). Emergent behavior is notoriously difficult to predict and can uncomfortably feel like magic (Bedau 1997: 378). A defining characteristic of emergence is that the most convenient way to find out what happens in an emergent system is to simulate it, or have the system run its course (*ibid.* 379). This explains why using iterative process with many prototypes is the most effective way to design games.

This paper aims to explore a method of visual notation based on UML (Unified Modelling Language) to help the game designer understand the emergent dynamics of his or her game. This method is not intended to replace the iterative design process. Rather it is intended to possibly add an extra quick iteration and to improve the accuracy of the information gained from each iteration, and thereby to increase the effectiveness of interventions of the game designer between iterations. Board games are used as a case study because emergence in board games is often easier to study than in other types of games. The complexity of the rules in board games is simply more limited than what is possible in a computer game. In addition, the rules of a board game are always explicit; nothing is hidden behind layers of code. Both these aspects facilitate the study of successful examples of game design to identify patterns that contribute to their success and quick experimentation with the structural features that contribute to the emergent gameplay.

Related Research

This is not the first attempt to get to grips with the elusive nature of gameplay using diagrams and patterns. The work of Staffan Björk and Jussi Holopainen (2005) has a very similar aim. They have identified and described a large number of patterns in game design. These are intended to help the designer analyze successful games, understand his or her own game, and to inspire new designs. However, their focus is on the activities of the player, whereas my focus is on structural aspects of the game system. As a result the approach presented here intends to focus more on game mechanics. My aim is to find the

'atomic particles' that can be used to create and describe many different patterns. A step which seem absent from Björk and Holopainen's work. What is more, the UML I propose is a visual notation. It provides the game designer with a gestalt of his design that is less abstract than the purely verbal patterns of Björk and Holopainen's work. In the end, I think that their game design patterns and my UML notation complete each other. Expressing their patterns in game design UML would be a great test for the comprehensiveness of the latter.

This paper has more in common with Raph Koster's research in game diagrams. His focus is also on atomic particles that make up the game experience; on what he calls the 'ludemes' and devising a graphical notation for these (Koster 2005). However, there are two important differences between his approach and mine. First, again, I focus less on the game experience and more on the system that facilitates this experience. Second, Koster's main method is that of reverse engineering (*ibid*): he looks at games and tries to find patterns, whereas I work from the hypotheses that emergence has a number of key, structural features which the notation must be able to express first and foremost. Although I will complement and test the results with the study (or reverse engineering) of a number successful games.

Neither am I the first to use UML for the design of games. UML is a modeling language designed for software engineers, and as many games are computer programs, UML has been used to design the software of games. However, my prime interest is not in the software, but in the conceptual complexity of games. M.J. Taylor et al. (2006) extend UML use-case diagrams to describe game-flow, the experience of the player. Their focus is on the play session and the progression of the player through the game. As a result their approach is very different from the approach presented in this paper. Their diagrams represent the linear, or branching, experience of play, not the structural complexity of game systems.

Finally, I did come across an example of UML describing games. Perdita Stevens and Rob Pooley use class diagrams, collaboration diagrams and state machines diagrams (three different types of UML diagrams) in their educational case study of modeling the structure of *Chess* and *Tic-Tac-Toe* with standard UML (1999: 178-189). However, their intention is to explain UML to students, not to study games. And though their approach is

valuable, I do not think it goes far enough to be of any real use for game design. Standard UML diagrams are not always suited to express the particularities of games, and are not really designed to visualize emerging properties of complex system.

The Structure of Emergence

This paper departs from the premise that the notion of emergent behavior in complex systems is a suitable framework to study games. The relation between games and emergence has been discussed by many game designers and researchers (see for examples: Smith 2001, Salen & Zimmerman 2003, Juul 2005, Sweetser 2006, Adams & Rollings 2007). With both types of systems the whole is more than the sum of their parts: it is hard (if possible at all) to predict the behavior of the whole by just looking at the behavior of the parts. While, the active agents or active elements in a complex system can be quite sophisticated in themselves, they are usually represented by rather simple models. Even when the study is about the flow of pedestrians in different environments, great result have been achieved by simulating them with only a few behavioral rules and goals (Ball 2004: 131-147). Similarly, the elements that make up games are can be a lot more complex than the elements of a typical system studied by the science of complexity, but some games (such as *Go* and *Chess*) are famous for generating enormous depth of play with relative simple elements and rules. The active substance of these games is not the complexity of individual parts, but the complexity that is the result of their many interactions. The main assumption of this paper is that the particular configurations of elements into complex systems that contribute to emergence are also potential valuable sources of gameplay. For a game designer this means that understanding the structural characteristics of emergent systems in general, and in his or her games in particular, is essential knowledge.

One of the simplest systems that show emergent behavior is the particular class of cellular automata studied by Stephen Wolfram. His extensive study has revealed three critical qualities of such systems: 1) They must consist of simple cells which rules are defined locally, 2) the system must allow for long-range communication, and 3) the level

of activity of the cells is a good indicator for the complexity of the behavior of the system.

The cells of cellular automata are relative simple machines that abide only to local rules. The algorithm that defines their behavior is not complicated and takes input only from its immediate surroundings. The easiest way to create a cellular automate is to design a simple state-machine that takes into account the states of its immediate neighbors. Without such input all cells would behave individually, and system-wide behavior would not be possible. When a cell takes input from other cells that are beyond its immediate surroundings, the behavior quickly becomes chaotic (a state beyond complex behaviour that is generally undesirable).

Despite locally defined and operating rules communication between cells must also facilitate long-range communication in the system. This means that long-range communication cannot be direct and takes time to spread through the system. Systems that show pockets of communication with little or no communication between the pockets will show less complex behavior than systems in which such pockets do not occur or are less frequent (Wolfram 2002: 252). Connectivity is a good indicator of long-range communication in the system. A special case of long-range communication is feedback, in which a cell or group of cells produce signals that ultimately feed back into its own state. Long range communication travel over long distances through the system or, alternatively, through time and produce delayed effects. As we shall see below feedback is very important in games.

The number of cells that are active (cells that change their state) is important for the behavior of the system as a whole. Complex behavior occurs mostly in systems with a lot of active cells (*ibid.* 76).

Cellular automata show us that the threshold for complexity is surprisingly low. Relative simple rules can give rise to complex behavior. Once this threshold is passed introducing extra rules does not affect the complex behavior as much (Wolfram 2002: 106).

In another study of emergence, Jochen Fromm builds a taxonomy of emergence that consists of four types of emergence. These types can be distinguished by the nature of communication, or feedback, within the system (Fromm 2005).

In the simplest form of emergence, *nominal or intentional emergence* (type I), there is either no feedback or only feedback between agents on the same level of organization. Examples of such systems include most man-made machinery where the function of the machine is an intentional (and designed) emergent property of its components. The behavior of machines that exhibit intentional emergence is deterministic and predictable, but lacks flexibility or adaptability.

Fromm's second type of emergence, *weak emergence* (type II), introduces top-down feedback between different levels within the system. Flocking is an example he uses to illustrate this type of behavior. A flock-member reacts to vicinity of other flock-members (agent-to-agent feedback) and at the same time perceives the flock as a group (group-to-agent feedback); a flock-member perceives and reacts to two different scales with in the system. When the top-down feedback within the weak emergent systems is negative the emergent behavior is stable. When the top-down feedback is positive the emergent behavior is instable.

One step up the complexity ladder from weak emergent systems we find systems that exhibit *multiple emergence* (type III). In these systems multiple feedback traverses the different levels of organization. Fromm illustrates this category by explaining how many interesting emergence can be found in systems that have short-range positive feedback and long-range negative feedback. It propels the appearance stripes and spots in the coat of animals and the fluctuation of the stock-market. The Game of Life is also an example of this type of emergence. The Game of Life can easily be shown to include both positive feedback (the rule that governs the birth of cells) and negative feedback (the rules that governs the death of cells). The Game of Life also shows different scales of organization: at the lowest end there is the scale of the individual cells, on a higher level of organization we can recognize persistent patterns and behaviors such as gliders and glider-guns.

Fromm's last category is *strong emergence* (type IV). His two main examples are life as an emergent property of the genetic system and culture as the emergent property of

language and writing, although one could question whether in both case one follows from the other, or whether these have evolved in unison and emergence might not be the best way to describe their mysteries. In any case, strong emergence is attributed to the large difference between the scales on which the emergence operates and the existence of intermediate scales within the system. Strong emergence is multi-level emergence in which the outcome of the emergent behavior on the highest level can be separated from the agents on the lowest level in the system: a Turing Machine can be build from the Game of Life, but also in other systems. The causal dependence between emergent behavior of a Turing Machine in the Game of Life and the game of life is minimal.

From this brief discussion a number of important observations on the nature and structure of emergence come forward. Emergent systems must consist of many elements that act more or less independently. A sufficient level of activity is required; a system with only few active elements tends to be too stable and predictable. Communication (or interaction) must exist between these elements at a local scale and this local communication must indirectly enable long range communication. Feedback, a form of communication where information and actions are fed back to the source, play a crucial role in emergent behavior, without feedback there is no emergence and stronger types of emergence only exists in systems with multiple feedback. Finally, emergent systems often show different scales of organization, with communication and feedback traversing between these scales. A UML notation for games that aims to represent the emergence must be able to visualize these important elements.

UML for Games

UML describes a collection of different types of diagrams. Of these, class diagrams, collaboration diagrams and state machines diagrams seem to be to most relevant for games. Perdita Stevens and Rob Pooley use these in their educational case study of modeling *Chess* and *Tic-Tac-Toe* with standard UML (1999: 178-189). State machines diagrams are very good at describing the inner workings of game objects or classes, just as state machines are a good technique to implement most game objects. Class diagrams are good tools to describe game elements, however, the complexity of a game ideally

does not come from its number of classes but from the number of ways their instances can be combined in a complex system. After all, most cellular automata describe by Wolfram (see above) consist of only one class of object. Collaboration diagrams show the interaction between objects, but are generally not specific enough to show the exact nature of communication.

To resolve this problem I chose to follow the lead of Bran Selic and Jim Rumbaugh (1998) who extended the UML collaboration diagrams by introducing capsules, ports and connectors. In their methodology, capsules are complex objects that interact with their surroundings through ports. Ports are connected by connectors that relay information between the capsules. Capsules are the central construct in their approach. The inner structure of a simple capsule is represented by state machine that communicates through the capsule's ports. More complex capsules are represented by subcapsules that are nested within the structure. Figure 1 shows a simple UML diagram representing a typical cellular automate in full detail (on the left) and in a collaborative structure (on the right).

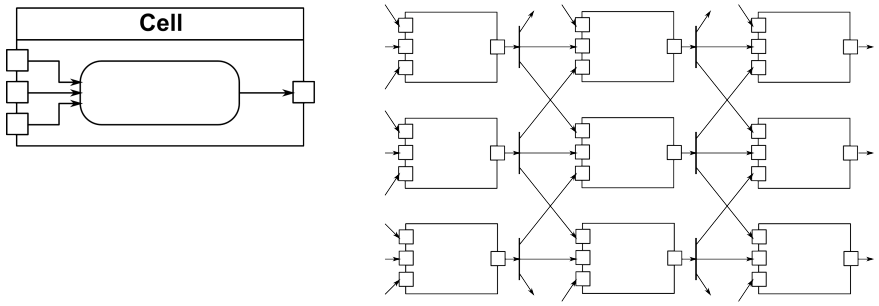

Figure 1 – UML for Wolfram's cellular automata

This approach forms a good basis to model games. It has the advantage of showing types of objects, showing interactions between objects (or their instances), and allows us to depict different scales (capsules and subcapsules). However it lacks in the representation of the nature of communication and feedback within the system. In games communication between elements does not only indicate flow of information, but also flow of resources. In addition, games work with very narrowly defined actions. The ways a player or an element can act upon another is limited and governed by the game's rules.

Finally, the difference between positive and negative feedback should be stressed, as these have a different impact on the behavior of the game. Below, I will discuss these three points in more detail.

Most games have an internal economy of resources. A lot of games (from board games such as *Monopoly* to simulation games such as *SimCity*, and from strategic games such as *Warcraft* to MMORPGs such as *World of Craftcraft*) incorporate a mechanism for money and other economic resources. But even in shooter in which money plays no role, concepts such as ammunition, energy and health-points can be, and often are, considered as vital resources. In these games it is important to identify where resources flow into and out of the game, and how elements can exchange resources between them (*cf.* Adams & Rollings 2007: 331-340). Technically speaking, the flow of resources is just another form of communication within a system and therefore can be represented by normal connectors. Yet, resources play often such a vital role in balancing a game that it is best to distinguish resources from normal information, in some cases it helps to differentiate between the different type of resources in the game.

Another argument for distinguishing resources from information is that is some ways resources act more like real-life objects than pure information does. Resources can be pooled and are not as easily duplicated. When resources flow from one capsule to another, they actually disappear from the fist object and arrive at the second. By using a special notation to capture these effects of communication with resources, it is no longer necessary design other structures to model the same behavior (such as tokens in Petri-nets). Distinguishing between information and resources in this way makes the model much clearer and less complex.

In the UML I propose I distinguish between the flow in information from flow of resource by using white ports and dotted lines to indicate the first and black ports and continuous lines to indicate the latter. In addition pools of resources, which can be thought of as special instances of state machines, are represented by black rounded rectangles (see figure 2).

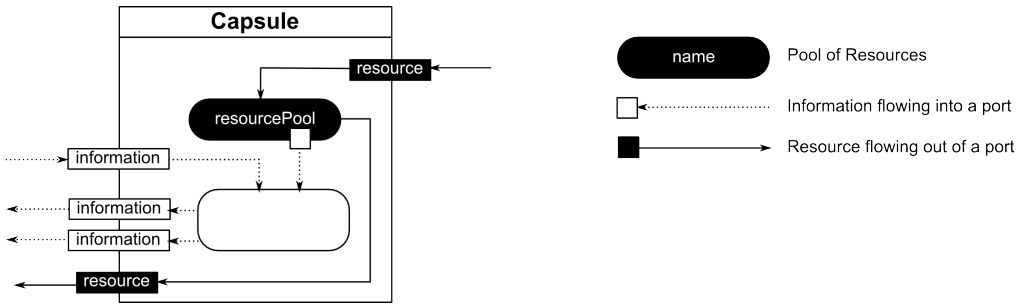

Figure 2 – UML to distinguish between information and resources, showing a capsule with a resource pool and (unspecified) information about the pool feeding into the capsule's state machine

Actions are an important formal aspect of games. Actions determine what how objects, and by extension players, can act and interact. Most game objects have only a limited set of actions that guides the players' behavior. The challenge of most games comes from achieving their goals with the operations allowed by its procedural rules; moving a ball past a goal area is easy, using only your feet makes it challenging (*cf.* Fullerton 2008: 25-26).

Actions are represented by rounded rectangles and distinguished from state machines by a name followed by parentheses that denote their similarity to functions or methods in computer code. Some actions tie into game's internal economy or have the power to create or destroy new game objects. As this has a clear effect on the dynamic of the system I have come up with a set of symbols and notions to denote these special instances (see figure 3). The concepts of sources, drains, converters and traders are taken from Adams and Rollings (2003: 332-334). A source produces a particular resource, a drain destroys a resources, a converter converts one resource into another, while a trader simply facilitates the exchange resources. The difference between a trader and a converter is that the total number of resources does not change as a result of a trader, while with a converter X resources are destroyed to produce Y resources of another type, where X does not have to equal Y.

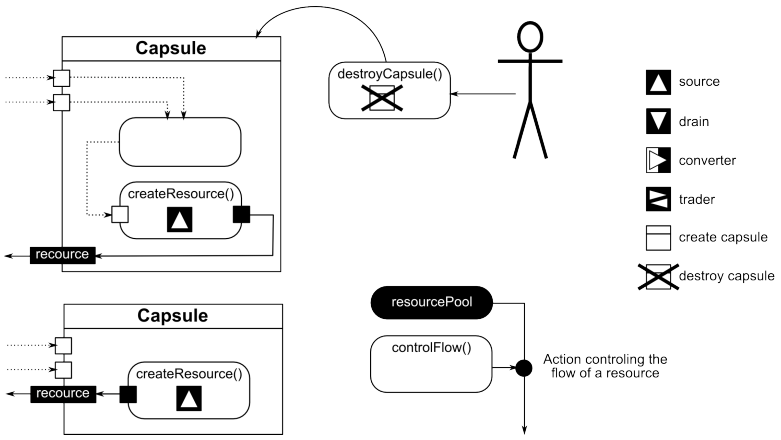

Figure 3 – UML for internal and external actions, showing a capsule that can generate a resource and for which a player action to destroy the capsule is defined. Curved lines imply hierarchy or action that works upon an object. The same capsule is shown twice (the shorter nation on the bottom implies the more explicit notation on top).

As we have seen feedback plays a vital role in emergence. Many games have multiple positive and negative feedback mechanisms that are responsible for the way competition between players or the players and the game evolves. When analyzing games it is necessary to be able identify positive and negative feedback. In the UML we use feedback becomes apparent through actual loops in the diagrams. In order to make feedback loops more visible I use plus and minus symbols to mark opportune connectors (see figure 4).

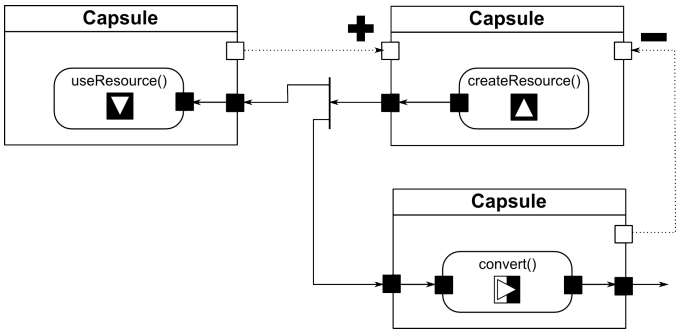

Figure 4 – UML showing positive and negative feedback

To illustrate the use of UML for games I have drawn up a diagram for *Tic-Tac-Toe* and on for *Connect Four* (see figure 5, the game state capsule evaluates whether or not a

player has won). As already mentioned, the rules of both games are quite similar, but there is a world of difference between the quality and depth of their gameplay. *Tic-Tac-Toe* is easy and deterministic, once both players have figured out the game they will always tie. On the other hand, the gameplay offered by *Connect Four* is considerably richer. Playing this game is not as straight forward as players can employ different strategies to trick their opponents. The diagrams of these games differ at two important points: *Connect Four* has a layered structure with rows consisting of squares, and where the communication in *Tic-Tac-Toe* flows in only one direction, the communication in *Connect Four* includes a feedback loop (the information of which squares have been occupied feeds back into the action of occupying the next square of a row). As we have seen, both of these structural characteristic play an important role in emergence.

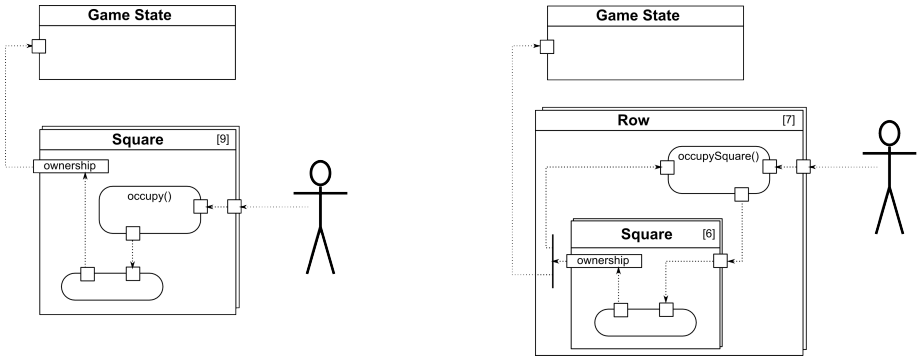

Figure 5 – UML for Tic-Tac-Toe (left) and Connect Four (right).

Case Study: Power Grid

Power Grid is a popular game that has been well received by a critical audience of board game enthusiasts. The current success and popularity of *Power Grid* can be largely understood as a direct consequence of its clever game design, in which most of the most important game elements seem to follow rules defined locally, but allow for many interactions; it fits the view games as complex systems particularly well.

In *Power Grid* the players make money by supplying electricity to cities in Germany or the United State (depending on which side of the board is used). The players need to balance three important aspects: investing in their power grid to connect more cities on the board, investing in power plants to increase their power production, and

buying fuel from a dynamic market (see figure 6). *Power Grid* uses almost no chance mechanisms. There is some randomness in the initial conditions. But most rules are designed to reduce randomness and to even out the effects of the initial condition. Still, *Power Grid* is a very dynamic game: two sessions with the same players can produce distinctly different effects. In one session a particular type of fuel might be in available in abundance while those same resources are strained in the next session.

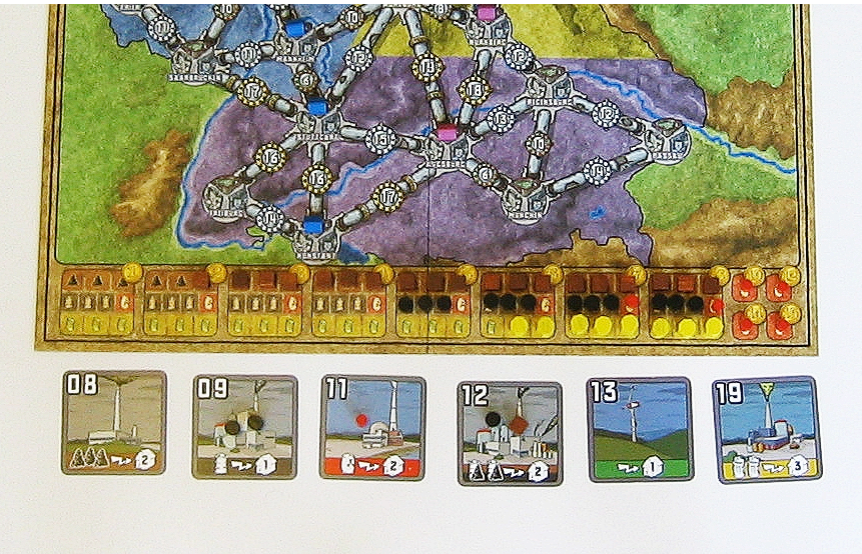

Figure 6 – Power Grid's power plants and board with fuel market

Negative feedback plays an important role in *Power Grid.* At the start of every round, the turn-order is established by the relative progression of the players. The player who has connected the most cities is first, while the one with the least is last. Frequently, two or more players have the same number of players, in which case the single highest base value of their power plants determines who is first and last. Being in the lead is no advantage. The leading player will need to bid on weaker plants first, buy resources last, and expand his grid last. The effects of this negative feedback are so great that in general it is best not to go far ahead of the pack during the beginning and middle phases of the game.

Closer inspection of *Power Grid* reveals that most of the game's rules operate on a very local level. For example, all the power-plants have a base value, a formula that dictates how much fuel it can convert in how much power during a turn (expressed in the

number of cities it can provide power to), and storage capacity that is directly derived from this formula (a plant can store twice the fuel it can convert in a single turn). The rules of the power plants interact with other elements: the number of cities it can supply is relevant for, but not directly connected to, the number of cities the player has connected to his grid. A plant's storage capacity can be used to influence the fuel market by creating strategic reserves and driving up the price for other players. The inclusion or exclusion of certain plants or certain type of plants (defined by the type of fuel they process) has an interesting effect of the game in its whole. The relative value of plant that runs on coals is determined by the available coal on the market, its absolute capacity, the influence it has on the turn order, and distribution of other coals plants among the rest of the players. It is fairly easy to understand the elements that make up this equation, yet hard to estimate its outcome. Just as it is fairly easy to understand how the power plants work, yet it takes some experience to master the dynamism a plant might cause. Even elements of *Power Grid*, such as the turn-order, which at a first glance seem to dictate the game's dynamic from the top down, have rules that mostly operate on a local level and are not connected to many other elements.

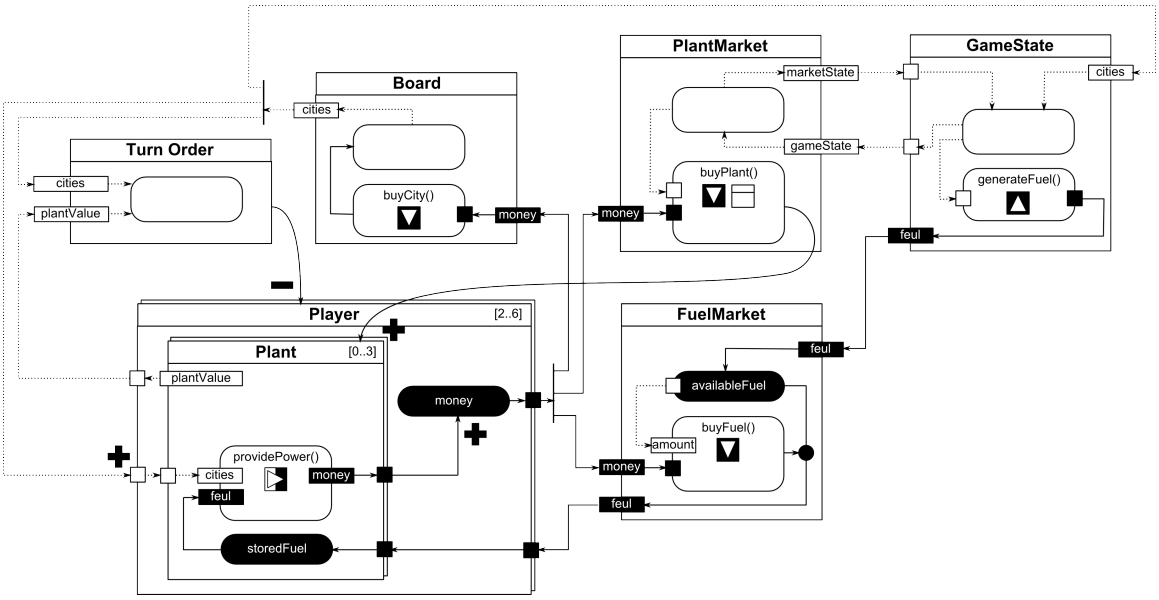

Figure 7 – UML for Power Grid

Figure 7 is a schematic representation of Power Grid using game UML. Figure 7 is not complete; it leaves out considerable detail. It represents the board, the mechanisms that govern the market for new power plants, and the mechanisms that determine the game's progression through three different phases as state machines without elaborating further. However, all the mechanisms of these elements have no direct impact on other elements of the games. Most of the possible lines of communication in the system are made visible in the schema, and are sufficient to explain the game's characteristic dynamism.

What stands out from figure 7 is that the relative low number of connections between the elements. Even players are not connected directly with one and another. Competition is only possible because the players all interact with the same board and the same markets. Yet, as became apparent from the description above, the system has enough connectedness and communication for all individual elements to influence each other indirectly. In short *Power Grid* is an excellent example of a system with mostly local rules that still facilitate long-range communication. At the same time there is considerable activity within the system. The number of cells that are active (each player has up to three power plants) is relatively high, matching another important design criteria for complex behavior described by Stephen Wolfram (see above).

From figure 7 the delicate feedback structure of *Power Grid* also comes forward. The game is driven by a main loop of positive feedback in which money is used to buy fuel that is in turn converted in more money (if the player plans correctly). This loop ensures that as the game progresses the players gain more money and resources. Its efficiency determines how well the player does in the game. Then there are two positive feedback loops that improve the efficiency of the main loop, but which also feed into the game's main negative feedback mechanism: turn order. Connecting more cities allows the player to make more money, but also put him ahead of other players. This means that her fuel and new connections will be more expensive, and good plants become less accessible to her. Buying more efficient power plants ensures that more cities can be supplied with less fuel, but also puts the player ahead of other players with the same number of cities; having the best (most valuable) power plant can occasionally be very inconvenient. This particular configuration of positive and negative feedback loops, ensures resources pile up and the game is driven towards its conclusion, yet it makes it

hard for a player to get to far ahead of the others without interfering too much with the main cycle (i.e. the game never drags on). If *Power Grid* has one problem it is that is feedback mechanisms are rather slow (one must invest in a system that feeds back into the main loop, to make the latter more effective). This makes it hard for an inexperienced player to assess the effectiveness of her actions over time. In turn, this makes the game somewhat difficult to learn, or, worse, can falsely lead players to believe that their actions are trivial.

Structural Game Design Patterns

The detail shown in figure 7 is not always necessary to analyze games. For quick analysis a short hand version of the same diagram can be sufficient. Figure 8 shows a diagram of *Power Grid* that shows less detail and brings forward the game's internal economy. In this diagram objects are still shown, but procedures are only implied by their symbols and the flow of different resources is color-coded.

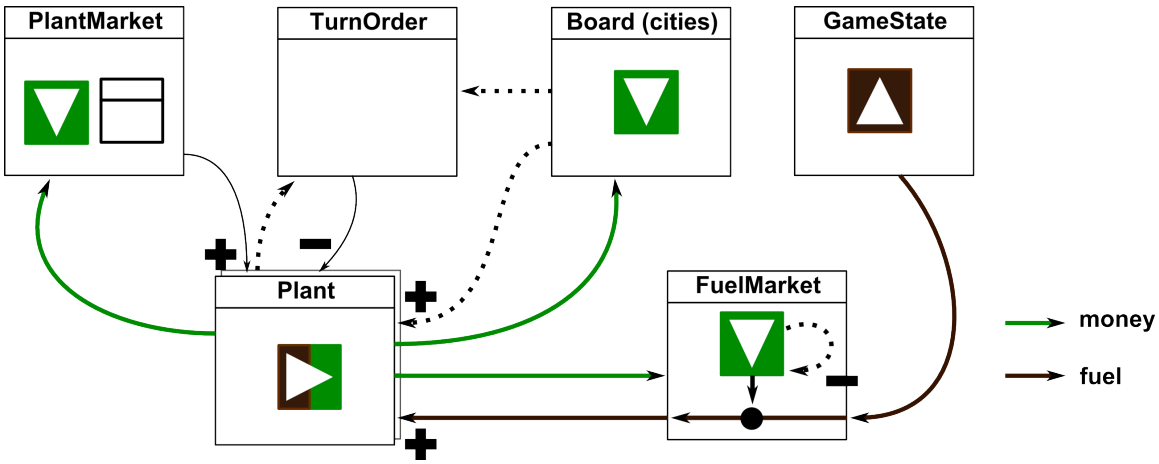

Figure 8 – Power Grid in short hand UML

We can go even further. Figure 9 shows only the feedback mechanisms of *Power Grid.* This diagram reveals game design patterns that can be generalized from the game. These patterns are shown in Figure 10 and discussed below.

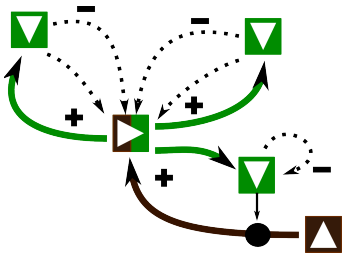

Figure 9 – Feedback structure in Power Grid

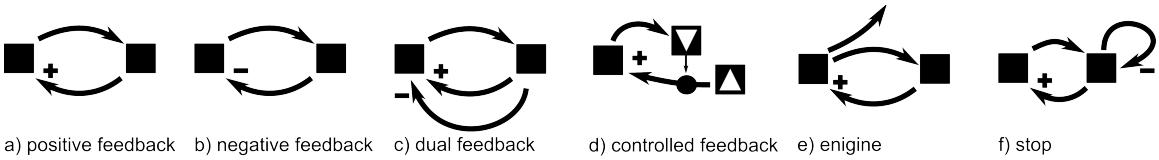

Figure 10 – Game design patterns distilled from Power Grid

At the heart of *Power Grid* is an engine (pattern e). An engine is a particular kind of feedback loop that creates a surplus of resources. In *Power Grid* the engine consists of the feedback loop between the fuel market and a player's power plants. The extra money the engine creates can be reinvested to increase the efficiency of the engine by connecting more cities or by buying better plants. But both these feedback loops are dual feedback loops (pattern c): they produce both negative and positive feedback. Finally the engine uses a controlled feedback loop (pattern d): investing more money does not necessarily lead to more resources as the amount of available resources is dictated by the game state, not by the amount of money the player has. In addition buying resources drives up the price for resources: it is a natural stop (pattern f). It dampens the effect of the positive feedback in the engine, and grows stronger when the strength of the positive feedback increases. The crucial difference between a stop (pattern f) and dual feedback (pattern c) is that a stop can only dampen and ultimately stop the positive feedback whereas with dual feedback the negative feedback can grow stronger than the positive feedback. It is also my assessment that for players the effect of a stop is in general easier to predict than the effect of dual feedback.

The possible patterns are be no means all identified by our analysis of *Power Grid*. Although the size patterns identified can be used to build many constructions, I

expect there to be plenty more. To me it is no surprise that in their research Staffan Björk and Jussi Holopainen (2004) have identified several hundred of patterns, and even their list is not complete. In fact, I doubt that a definite list could ever be produced, as the possible combination of functions and connectors is probably infinite, even though many would just be variations on basic themes and consist of the same atomic elements. Still the identification and study of these patterns is well worth the trouble. UML notation for games constitutes a visual language in itself. The expressive power of this language needs to be vast compared to the number of elements (or 'words') in order for the language to be effective at all. In the end, UML for games is supposed to be an expressive tool for game designers, not limit their options. It is my intention that UML for games fulfills a similar role as the diagrams and models of structuralist linguistics such as A.J. Greimas actantial narrative schema (see Martin & Ringham 2000: 19).

Looking Ahead (and Sideways)

One case study and a handful of examples is not enough to validate a new type of UML notation. Although my initial explorations are promising, a lot of work remains to be done. Currently, I am actively testing and refining the UML using a large number of board games. I have students using the method when designing and analyzing board games. So far the results have been mixed. Obviously, the UML as presented here is very well suited to a particular type of game: those that rely on a dynamic internal economy. Although, according to some, *every* game has an internal economy, it is obviously more important to some games than to others.

Looking sideways at computer games, emergence does not always seem to be as important to the gameplay as it does for board games. Jesper Juul distinguishes between two categories of games: games of emergence and games of progression. The latter category, which he associates with computer games, relies more on a carefully designed series of events than on emergent behavior, not unlike an interactive movie experience (Juul 2005: 5). However, the scale of modern games and player expectations regarding their freedom to explore, dictate that for computer games, too, emergence is of growing importance (*cf.* Smith 2001, Sweetser 2006). It is my opinion that computer games do not make enough use of emergence. Hopefully, being able to visualize the structures that

contribute to emergence will inform and inspire future designs of computer games. How such games should be modeled using a similar scheme is not immediately apparent, it is one focus of my future research.

One promising result is the insights a detailed study of a game through UML yields. It is my, and my students, experience that UML helps identify successful structures and patterns in existing games and helps identify potential weaknesses or opportunities when evaluating prototypes. It helps them to adjust a design more effectively because feedback structure (among other things) becomes very tangible using UML. What is more, game design UML, and especially, the patterns they describe can be a valuable creative tool used for brainstorming. In this respect it has much in common with the Björk and Holopainen's game design patterns (2004: 44-47).

The UML for game design is a work in progress that I will evolve through my work as a lecturer and researcher at the Hogeschool van Amsterdam. Using UML to study games presupposes a particular view on games that frames game design as an object-oriented design practice. This view ultimately treats all games as 'games of emergence' in which many parts contribute to the behavior of the whole. Although the work is still in its initial stages, and despite the mixed results that came out of student work using UML to analyzing and designing board games, I have high hopes UML for game design. In the end, I hope it will provide game designers with an effective tool that is intuitive to use, expressive and articulate in its representation, and precise and accurate enough to facilitate understanding of games as dynamic systems.

References

Adams, E., & Rollings, A. (2007). *Fundamentals of Game Design*. Upper Saddle River: Pearson Education, Inc.

Ball, P. (2004) *Critical Mass: How One Thing Leads To Another*. New York: Farrar, Straus and Giroux.

Bateman, C., & Boon, R. (2006). *21st Century Game Design*. Boston: Charles River Media.

Bedau, M.A. (1997) Weak Emergence in J. Tomberlin (ed.) *Philosophical Perspectives: Mind Causation, and World*, Vol 11, 375-399.

Björk, S. & Holopainen, J. (2004) *Patterns in Game Design.* Boston: Charles River Media.

Fromm, J. (2005) Types and Forms of Emergence. Retrieved September 8, 2008, from http://arxiv.org/abs/nlin.AO/0506028

Fullerton, T. (2008). *Game Design Workshop: A Playcentric Approach to Creating Innovative Games, 2nd Edition.* Burlington: Morgan Kaufman.

Huizinga, J. (1997/1938) *Homo Ludens: Proeve Ener Bepaling Van Het Spelelement Der Cultuur*. Amsterdam: Pandora.

Juul, J. (2005) *Half-Real, Video Games between Real Rules and Fictional Worlds*. Cambridge: The MIT Press.

Koster, R. (2005) A Grammar of Gameplay: game atoms: can games be diagrammed? Presentation at the Game Developers Congres 2005. Retrieved September 8, 2008, from http://www.theoryoffun.com/grammar/gdc2005.htm

Martin, B. & Ringham, F. (2000) *Dictionary of Semiotics*. London: Cassell.

Salen, K.& Zimmerman. E. (2003) *Rules of Play: Game Design Fundamentals*. Cambridge: The MIT Press.

Selic, B. & Rumbaugh, J. (1998) Using UML for Modeling Complex Real-Time Systems. Retrieved September 8, 2008, from IBM.com site: http://www.ibm.com/developerworks/rational/library/content/03July/1000/1155/1155_umlmodeling.pdf

Smith, H. (2001) The Future of Game Design: Moving Beyond Deus Ex and Other Dated Paradigms. Retrieved September 8, 2008 from Igda.org site: http://www.igda.org/articles/hsmith_future.php

Stevens, P. & Pooley, R. (1999) *Using UML: Software engineering with objects and components, updated edition*. Harlow: Addison Wesley Longman Ltd.

Sweetser, P. (2006.) *An Emergent Approach to Game Design - Development and Play.* PhD thesis, The Univerisity of Queensland

Taylor, M. J., Gresty, D., & Baskett, M. (2006) Computer Game-Flow Design in *ACM Computers in Entertainment*, Vol. 4 No. 1, article 3A.

Wolfram S. (2002) *A New Kind of Science*. Champaign: Wolfram Media Inc.