Thomas Tregel, Christian Reuter, Stefan Göbel, Ralf Steinmetz:
Generating Multiplayer Games for Interaction Learning using Game Design Patterns. In: Thomas Connolly, Liz Boyle: Proceedings of The 10th European Conference on Games Based Learning, p. 686 - 695, Academic Conferences and Publishing International Limited, October 2016. ISBN 978-1-911218-09-8.

Generating Multiplayer Games for Interaction Learning using Game Design Patterns

Thomas Tregel, Christian Reuter, Stefan Göbel, Ralf Steinmetz
TU Darmstadt, Darmstadt, Germany
thomas.tregel@kom.tu-darmstadt.de
christian.reuter@kom.tu-darmstadt.de
stefan.goebel@kom.tu-darmstadt.de
ralf.steinmetz@kom.tu-darmstadt.de

Abstract: Practicing team communication and player coordination in games requires a dedicated multiplayer setup especially tailored towards the given requirements. Common AAA-titles rather focus on a positive game experience than a beneficial training effect. Most Serious Games rely on singleplayer gameplay to try to achieve their intended behavior change. Based on Game Design Patterns for Collaborative Player Interactions and the usage of Procedural Content Generation, this paper explores the automatic creation of interaction sequences, usable for player interaction training.

Additionally, E-learning pattern aspects are integrated to support the team's learning progress. The design focuses on non-linear sequences to create a network of player interactions. Each player interaction represents an individual challenge within a game world that needs to be approached by players in a coordinated fashion. The non-linearity promotes a decision-making process, whenever the team has to make a choice concerning their next steps. While it might be possible to create individual interaction networks by hand, Procedural Content Generation allows us to diversify the expected result, based on the given parameters. Due to its abstract representation, different playstyles, user groups and communication requirements can be targeted. Furthermore, different valid ways leading through the given challenges let players choose their path to take in a group effort, which increases the game's variety and adaptability.

This paper presents a research concept and analyses the potential use and pitfalls when using highly abstract representation of content to use in a Procedural Content Generation approach. Although the generation algorithm is implemented for scene-based games with our *StoryTec* authoring environment, the concept can be used to generate pattern networks for most game genres, independent from their game speed or game flow.

Keywords / Key Phrases: multiplayer, game design patterns, interaction learning, procedural content generation

1. Introduction

Collaboration elements have lately risen in popularity in multiplayer games. These games encourage players to team up and work together towards the set goal. One distinguishing factor lies in the emphasis of the group's interest compared to personal objectives (Beznosyk et al, 2012). The game's goal is thereby different from other competitive games as it is not all about to win as an individual player, but as a whole team (El-Nasr et al, 2010). This adds additional aspects to the game that need to be considered. In a collaborative environment, players are often required to communicate with their teammates in order to successfully complete different types of challenges. These challenges can for example centre on parallelization, synchronization, time constraints or communication of distributed information (Reuter et al, 2014). Collaborative games encourage players to team up and work together, creating many opportunities for them to interact with each other.

Design patterns were introduced in 1993 as a new mechanism to describe design structures in software engineering (Gamma et al, 1993). They provided vocabulary for designers to communicate an abstract view on the given problem and describe the core of a solution for that problem in software design. Game Design Patterns are an adaptation of regular design patterns to game design with similar goals (Björk & Holopainen, 2004). As game design is more of a creative process, strict problem-oriented thinking cannot always be applied (Reuter et al, 2014).

Procedural Content Generation (PCG) refers to the automatic creation of game content with the use of algorithms and is used in many different genres. It can address scalability issues, diversify the content it is generating or create a dynamic gameplay layer (Smith et al, 2011). Due to its broad application range it can be implemented in many forms and has been used in a multitude of games with different requirements and constraints. The differences between the available algorithms and their respective characteristics have been examined to a large extent (Togelius et al, 2010).

This paper is structured as follows: First we described the characteristics of game design patterns for collaborative player interactions. This is followed by a concept to (non-)linearly combine them into interaction network. Afterwards we present approaches to measure the interaction network's quality and to iteratively improve it. In the last sections we discuss limitations of the concept regarding the pattern collection and give our conclusion and future work on the topic.

2. Related Work

For collaborative multiplayer games Rocha et al. (2008) created six basic game design patterns for cooperative games which cover a broad range of multiplayer scenarios and describe the underlying game system on a high abstraction level. Based on his work El-Nasr et al. (2010) created seven additional patterns focusing on two different types of cooperative games: The first category is played sharing a computer or screen while the second one relies on distributed collaboration using multiple machines.

These patterns build the foundation for over 20 collaborative player interaction patterns (Reuter et al. 2014). For this collection the focus lies on the interaction characteristic of each pattern, as each described action is directly performed by a player and is either directed upon another player or the game world in general, which then influences other players directly. For each interaction the level of contribution can vary between the participating players. This can be due to players with unequal skill levels or different game roles. Due to their limited and well-defined area of application and interaction length the patterns are suitable for game and interaction design as they can be combined by concatenating them into an interaction chain.

Collaborative player interactions relevant for our later model are classified along the four dimensions of space, time, player characteristics and functional constraints, which sum up to eight parameters describing the interaction's nature (Reuter et al. 2014). The available parameter values are shown in Table 1 and display the patterns' characteristics based on the given dimensions. As collaborative player interactions are the core matter of the previously described patterns, they become the defining factor applicable in the game-based learning domain in terms of interaction learning.

3. Generating multiplayer games using collaborative player interaction patterns

Our goal is to create a model that allows us to generate a network of collaborative player interaction patterns that uses the patterns' provided information in order to combine them procedurally. We subsequently modify the generated network according to metrics that ensure the network's quality and rule out less optimal choices.

The presented generation approach does not include asset generation and focuses on the interaction sequence generation. Due to the patterns' abstraction level no explicit game world or storyline is generated, leaving room for game designers to create their game idea around the generated interaction network. The focus on player interactions allows for a game mainly designed for communication and interaction training applicable for E-learning scenarios. The difference to other comparable approaches lies within the primary focus on interactions and the generation algorithm optimizing for smooth interaction learning.

3.1 Using PCG algorithms

In order to choose the proper PCG algorithm we use the presented classification guidelines (Togelius et al, 2010, Togelius et al, 2011).

- As online content generation needs to be performed during a game's runtime and adapts to player actions during the game it is not applicable for our approach. Because game assets and mechanics are still missing during the generation process, we choose to use offline generation.
- The generated content cannot be directly classified into optional and necessary content. Because of the network's abstract representation of player interactions the need for solvability cannot be determined for an individual interaction. For that reason we decide to only allow necessary content in the context of the generation algorithm.
- For offline PCG techniques a parameter is best suited for the generation process, because in our case game developers still have an influence on the actual game during the implementation process.
- The previous case leads to the demand for deterministic generation with a specifiable random seed parameter.

- Generate-and-test algorithms are more flexible than constructive algorithms in general, as the final result can be created over multiple iterations. Because the game itself cannot be tested after each iteration, we choose to use practices from constructive algorithms to ensure the content's quality.

3.2 Generating non-linear interaction networks

With the generated network, interaction paths are described that contain all required information about the interaction parameters and their effect. In order to not limit the model to a linear sequence of interactions we incorporate non-linear pattern sequences which increase variety, increase interaction dynamics through changes in group constellations and give game designers an option to place optional content.

The player separation process to create non-linear pattern sequences uses branches. From now on, the term 'branch' is used for a pattern that has at least two emerging paths in the network. A 'path' is a connection between two patterns that controls the team's game flow. Thereby branches are patterns that have more than one following pattern option. Regarding player pathing we identify two different branch types which depending on the preceding pattern can have different causes and meaning:

- Split Branch: A branch following an interaction splits the team and forces them to go on separated.
- United Branch: Following an interaction the team goes on together, but has to decide which path it uses.

There is an additional case where players choose individually which path to take, which can lead to a split up team. This case can be represented by a combination of the two identified branch types as seen in Figure 1 and is not our main focus, as it does not contain any cooperation effort in the decision-making process.

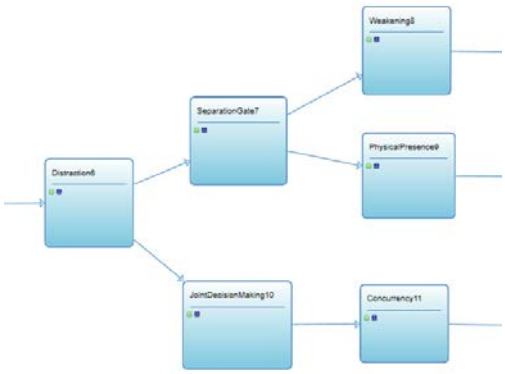

Figure 1: StoryTec representation for a branch with individual player choice

3.3 Pattern duration

Because the patterns are combined in such a way that connected patterns are direct predecessors or successors of each other, their duration dictates the temporal course of events and thereby is an impactful parameter for the generation process. When facing a strictly linear sequence of patterns the duration is summed up and can be seen as an estimate of time needed to perform all interactions once in the given order.

After a separating event like the 'Separation Gate' used after split branches, multiple paths run in parallel. Depending on each individual path's duration sum, major differences can occur which leads to delay at the respective point where the paths are merged again. As described in the 'Gathering Gate' pattern, waiting times are possible, but they can cause faster players to get annoyed while waiting. Game designers want to reduce possibilities for negative game experiences as much as possible. Depending on the pattern, different requirements arise for the involved players.

3.4 Nesting patterns

When investigating the spatial location property three pattern types are possible: patterns are either defined as always happening in a pervasive location, always happening in a specific location, or both

options are possible. The special case where patterns always happen in a pervasive location can be used for a pattern sequencing characteristic we call ‘nesting’. Nested patterns have a mutual superordinate pattern. The actual interpretation of this structure is as follows. A superordinated pattern always has the pervasive location property assigned to it. It can be used whenever players are in the process of solving the pattern’s subordinated, nested pattern challenges. It gives players the option to use that interaction whenever they choose to. Additionally it can be used multiple times as long as possible pattern conditions are met. The division into regular and nestable patterns can be seen in Table 1 along with all structural parameter values for each pattern.

Table 1: Overview over the four non-interaction based parameters for all patterns. Greyed out parameter values are unavailable.

	Name	spatial relation		spatial location		player freedom		role flexibility	
		collecting	separating	specific	pervasive	obligatory	voluntary	fixed	free
regular	Concurrency								
	Parallelization								
	Dependency								
	Joint Decision Making								
	Perspectives								
	Guide								
	Distraction								
	Weakening								
	Transport								
	Moving In Formation								
	Physical Presence								
	Hot Potato								
	Contest								
	Traitor								
nestable	Strengthening								
	Resupply								
	Protector								
	Savior								
	Tactical Guidance								
	Trade								
	Shared Resources								
gates	Separation Gate								
	Gathering Gate								

3.5 Interaction Difficulty

The main parameter used during the generation process is the ‘Interaction Level’. Depending on a pattern’s timing and communication requirement the amount of interaction needed to overcome its challenge can be determined. Each of these two pattern parameters has different consequences on the interactions taking place. A high communication requirement forces players to communicate in order to determine a correct answer, the best approach to continue, or a decision regarding the current task. The kind of communication taking place depends on several external parameters and

can thereby not be part of the generation process. Only the estimated amount of communication needed can be used.

External parameters for communication identified by Nasr et al. can be a deciding factor for successful player interaction, but could not be integrated in our model, because of the early generation point in time.

We construct the 'Interaction Level' as a scaling difficulty parameter in order to increasingly challenge and train a team's interaction capabilities. By combining both the patterns' timing and communication requirement parameter, nine possible parameter value combinations are available – the interaction values. In interaction-focused games a gradient curve can provide another dimension of difficulty, beneath a game's mechanical difficulty level. All available interaction values can be seen in Table 2.

Table 2: Different available interaction values for each respective pattern.

	Name	Interaction values
regular	Concurrency	{5}
	Parallelization	{1}
	Dependency	{0}
	Joint Decision Making	{4}
	Perspectives	{3.5}
	Guide	{5, 7}
	Distraction	{2}
	Weakening	{4}
	Transport	{1}
	Moving In Formation	{7}
	Physical Presence	{1, 2, 3, 4, 5, 7}
	Hot Potato	{7}
	Contest	{1}
	Traitor	{0}
nestable	Strengthening	{1}
	Resupply	{3}
	Protector	{4}
	Savior	{5}
	Tactical Guidance	{3, 5}
	Trade	{2}
	Shared Resources	{1}
gates	Separation Gate	{1}
	Gathering Gate	{1}

4. Quality assessment and personalization

As stated before the current generation process does not generate assets nor does it create a playable game. In a game with a high number of optional paths or alternate story lines, these game elements have to be created by the game developers. Thereby an extensive use of multiple possible paths, in a world of limited development time, has the direct consequence of decreased game length.

For branches we limit both the number of possible emerging paths at one branch, and the number of possible layers, in order to not run into a too high number of alternate paths. Both parameters are meant to reduce the networks breadth in order to confine the chance for a pattern interaction element to not be experienced in a play-through.

An additional option to control the model's branching behaviour is to force the generation algorithm to unite all existing paths into a single path. Henceforward the algorithm can continue to generate the following content and split the team up again. The advantage of this technique is that a main path is generated that needs to be passed where important new game elements can be placed or the game's core story can be further developed. In addition to that, this option works against a possible outcome, where a team is split up at the very beginning and progresses further in two or more separated teams until they are reunited at the very end. While this outcome may be a viable outcome it is undesirable because of its low interaction possibilities and the reduced amount of generation options due to limited group sizes.

4.1 Pattern selection

Choosing which pattern to use next is another non-trivial task and is a deciding factor for the network's quality. One possible approach is to randomly select one pattern from the pattern collection without restricting the selection choices. This method causes an about equal distribution of pattern usage, with no big outliers who are either picked frequently or infrequently. The downside here lies within the aspect that the algorithm has no influence on the selection process and cannot incorporate any knowledge or metrics that improve the selection in terms of resulting content quality. By choosing a random pattern from the collection, the amount of possible parameter values is restricted to the values each respective pattern allows. Thereby the pattern selection process would already bias the outcome, since some parameter values can no longer be chosen in that situation.

Since we only want to limit the randomness to confine impractical generation results, we use a direct fitness function known from search-based PCG. Depending on the required parameters and the disallowed ones, a fitness value is calculated for each pattern. The pattern choice for the remaining patterns is chosen based on other metrics that predict the content quality given the respective pattern is selected. All pattern parameters not enforced by the chosen pattern or the filtering process are randomized.

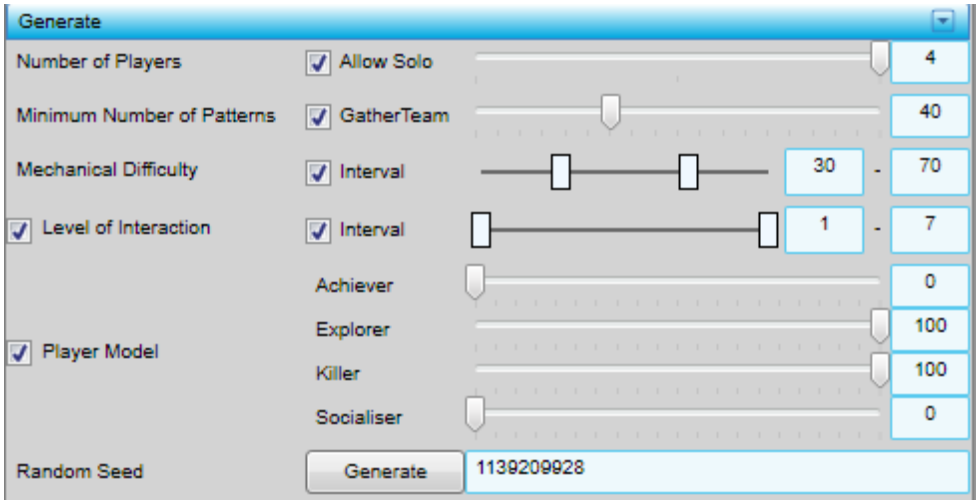

Figure 2: Generation dialogue used to set parameter vector.

4.2 Player type adaptation

We chose to annotate all collaborative interaction patterns with player model data using Bartle's model (Bartle, 1996). For some patterns we could not assign a specific player type, due to their abstract design, resulting in neutral player model values.

During the pattern selection process the respective player model is a determining factor as it is included in the fitness function described before. This results in interaction paths inside the generated networks tailored either towards specific individual player types or towards a certain combination of player types. Depending on the player type composition individual interaction paths are generated for the present player types. For that generation step either a global player model or a model for each expected player can be set.

Because player type information collected during gameplay cannot be used for this process, game developers need to specify a targeted player model for the game segment they are generating. The

determined player types during runtime can then influence the game segment selection based on the team constellation.

4.3. Repeating patterns

One metric we use to ensure a high diversity is to check for repeating patterns in the network flow. Whenever similar patterns are located close to each other on the same path, actions that are required by players may feel repetitive or boring. When enough time lies between two interactions of the same type, it may even be beneficial to reuse the pattern for two reasons.

First, the amount of available patterns is limited. When using a pattern that has not been used on the current network path for a longer time, it ensures a high pattern variety, because it creates more space for the following patterns regarding their previous duplicates. Second, a previously used pattern is likely to have a significantly lower difficulty value assigned to it.

Now assuming the basic actions, required to successfully complete the given challenge, are comparable over all difficulty levels. Players encountering a harder version of a previously encountered challenge can make use of their previous experience and knowledge to approach the new challenge. This lets players recall their previous actions, and adapt them according to the higher difficulty. When neither of the two aspects is applicable, we do not reuse a pattern in the given situation. Both patterns within the ‘Progression Gates’ category are excluded from this rule as they are used to create different network branching characteristics.

4.4 Path duration balancing

Another metric we use is the path duration balancing. Its purpose is to use the patterns’ duration parameter and to analyze branches and alternate paths in terms of similar duration.

Whenever a branch occurs in the network, all possible paths towards a mutual merging pattern are used to calculate the duration differences between these paths. Since every pattern on a path has its own duration, these values can be summed up to calculate the respective path’s duration value. This however only works under the assumption that transitions between these patterns do not use a significant amount of time compared to the interactions represented by them. In a system where transition times matter, this metric could be easily enhanced by the respective estimated transition durations. An overall path duration would then be calculated by adding together both the pattern duration sum and the pattern transition sum.

Different path durations can have different implications for the generated content. When a split branch is generated, each of the possible paths should have similar lengths duration-wise. The reason for that is that players using the shortest path would encounter significant waiting times when they reach the position in the game where the previously split paths merge together. To balance out the different paths, multiple options exist. First, patterns on a path could be modified by setting different duration parameter values. This however is unlikely to achieve a balanced state, because only the ‘Concurrency’ and the ‘Shared Resources’ pattern allow more than one duration value. A second option is to insert an additional pattern, which has the respective duration assigned to it. It is important to note, that this additional pattern should only be inserted using a linear connection between two patterns that are neither a branch nor a merge point.

When a united branch is generated, path durations can differ without much further impact. Since no waiting times occur because players do not experience the merging interaction as a situation where different paths merge together, paths do not need to be balanced. In this situation, other effects for the paths are conceivable. The shorter a path is, duration-wise, the harder in terms of game difficulty he could become. This behaviour would simulate the concept of having a short, but dangerous or risky path and having a longer, but safer path.

5. Parsing into StoryTec

After the generation process has terminated and all post processing has been finalized, we start parsing the generated structure into a StoryTec story. Thereby the parsing process can be treated modularly as it does not change any pattern information. We start representing all generated patterns as single StoryTec scenes. For each scene a pattern object is added that contains all pattern parameters with their respective chosen parameter values. Each connection between two patterns is

represented by a transition in StoryTec. The transition does not contain any information, and models the connection between the two patterns as well as its direction, since they are all unidirectional.

In order to better visualize the final result we choose to rearrange all scenes in a more clear fashion. Parallel paths are positioned one below another to save space in the horizontal dimension and to better demonstrate the characteristic of branches as seen in Figure 3.

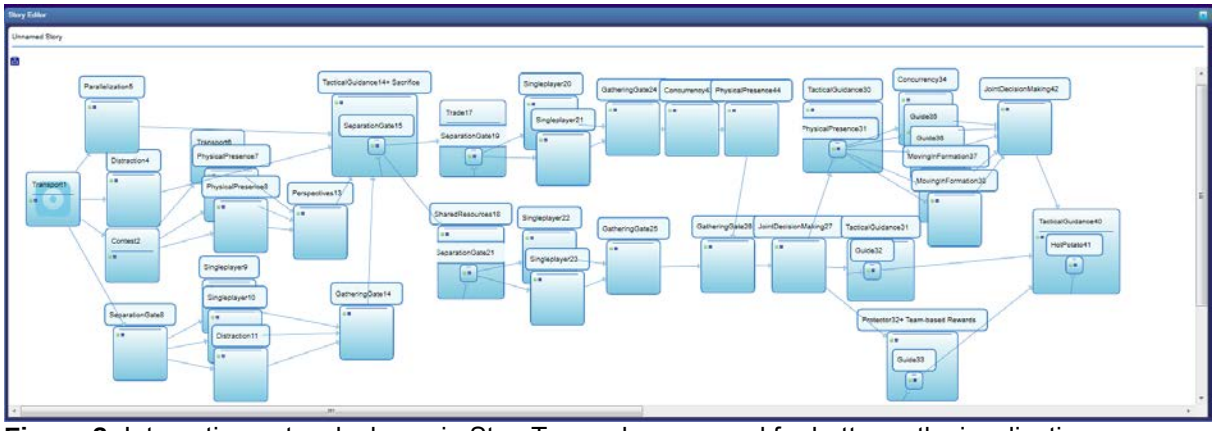

Figure 3: Interaction network shown in StoryTec and rearranged for better path visualization.

6. Evaluation

The generated pattern networks cannot be evaluated in a conventional way, because of the patterns' high abstraction level and the associated result, which cannot experienced yet in a game environment. We choose to analyze the pattern distribution depending on the interaction level, which is the major characteristic of these patterns. It is important to note that all pattern networks have been generated using static parameters, not present within the main parameter vector. These parameters control the amount of variety introduced into different aspects of the network, like the chance for branches or the variety in pattern selection. During the evaluation process neither of those parameters have been changed and remained on their static predefined values.

6.1 Interaction level

In Figures 4 and 5 we see the average chosen interaction values, depending on the patterns' position within the network. The seven shown datasets are based on the seven different static interaction. For the regular patterns all graphs, except the one for interaction level 3, start at their respective interaction value. While the graphs for the values between 3 and 5 have a stable characteristic around the targeted interaction value, the outer two graphs for each side behave different. For the two uppermost values, both graphs drop significantly in their mean value as seen in Table 3. The graph for the interaction value of 7 drops to an arithmetic mean of 6.410, and the graph for the value of 6 stays comparatively higher at a mean value of 5.555. The greater drop in mean value for the higher interaction level can be explained through the greater interaction level distance, when a pattern with deviating interaction value is chosen. The interaction graph for the value of two has a well-fitting mean value but also a high standard deviation (SD). Its implications are shown at the end of the respective graph, with a spike above the interaction value of 3. The static interaction level of 1 suffers from similar problems as its counterparts at the interval's other side. The graph has a much higher arithmetic mean of 1.426 than intended, coupled with the second highest standard deviation.

For the nestable patterns another problem appears, due to the lacking choice of high interaction patterns. Starting from an interaction level of 4, the average interaction value for these patterns only rises marginally with an increasing interaction level. Additionally, the standard deviation is much higher for most of the data. The first two parameter values are the only cases where more stable data can be observed. Here the parameter is sufficiently mapped. One special problem, not emerging in these two settings, is the parameter coverage of the 'Player Support' patterns. Because the 'Shared Resources' pattern is the only pattern containing the 'separating' spatial attribute, it can become the only possible choice. Due to its interaction value of 1, the average value for high interaction level networks decreases.

We conclude that networks for the medium targeted interaction values can be created with high precision. For the lowest value, a slightly higher resulting interaction value needs to be accepted in

order to introduce variety. For the two highest values the targeted values cannot be met properly because of missing patterns, defined for the required parameter values. Analogue to the case of interaction level 1, the interaction level 5 creates a network with a slightly lower resulting interaction value than specified.

Table 3: Statistical assessment for different interaction values dependent on pattern type.

Interaction level		μ (mean)	σ (SD)
1	regular	1.426	0.232
	nestable	1.271	0.173
2	regular	1.995	0.292
	nestable	1.977	0.232
3	regular	3.245	0.130
	nestable	2.956	0.269
4	regular	4.045	0.114
	nestable	3.428	0.211
5	regular	4.788	0.117
	nestable	3.503	0.306
6	regular	5.555	0.167
	nestable	3.600	0.347
7	regular	6.410	0.207
	nestable	3.647	0.299

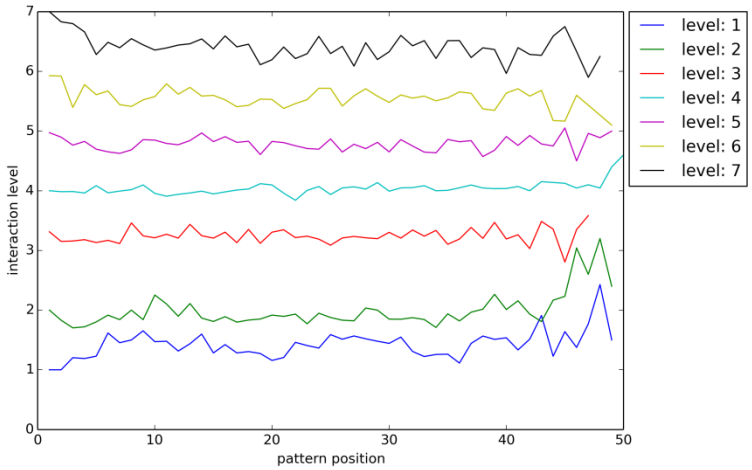

Figure 4: Interaction values for regular patterns.

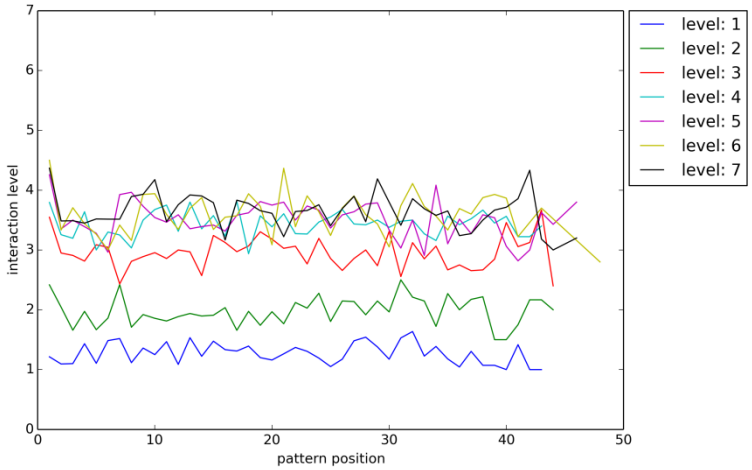

Figure 5: Interaction values for nestable patterns.

7. Conclusion

In this paper, we have developed a concept to procedurally generate multiplayer interaction networks based upon collaborative multiplayer interaction patterns. The model is parameterized by player parameters, a targeted interaction value or interval and an optional estimated player model based upon Bartle. All pattern properties described within the original collection are incorporated and an interaction level for each pattern is calculated and incorporated into the pattern selection process. To simulate an increase in interaction requirement, the parameter can be set to dynamically adapt to the network length. We applied this model using the rapid-prototyping environment StoryTec, and adapted it according to its technical limitations. The system shows the combinability of abstract interaction patterns to create an interaction network usable in digital games focusing on player cooperation and interaction learning.

Additional patterns, without any particularly discriminate characteristics or consequences, can be added without problems to ensure expandability. Our evaluation showed gaps for specific parameters and interaction levels. This leads to a situation were no additional generation rules or concepts can safely be introduced without risking situations, where no more valid patterns can be found. Thereby the pattern collection is not complete and is subject to change as more patterns are introduced to fill definition gaps in interaction value.

As a next step regarding player adaptation an experimental study is needed to cover the attraction of individual patterns, when being faced by teams with a predetermined player type or special combinations of player types, allowing us to further determine targeted player types for each pattern.

To enhance our model to be applicable for full online generation an algorithm is needed to transcribe the abstract pattern parameters into playable game segments. For scene-based adventure games this could be done by developing a handful of parameterized challenges for each pattern and to represent each pattern as an independent challenge in its own scene. The complexity here is increased by the availability of nestable patterns and the need to incorporate their actions into each pattern's challenge.

References

Bartle, Richard. (1996) Hearts, Clubs, Diamonds, Spades: Players who suit MUDs. *Journal of MUD research*, 1(1).

Beznosyk, A., Quax, P., Coninx, K., and Lamotte, W. (2012) The influence of cooperative game design patterns for remote play on player experience. *Proceedings of the 10th asia pacific conference on Computer human interaction - APCHI '12*, pages 11–20.

Björk, Staffan and Holopainen, Jussi. (2004) Patterns in Game Design (Game Development Series). *Charles River Media, 1 edition.*

El-Nasr, M.S., Aghabeigi, B., Milam, D., Erfani, M., Lameman, B., Maygoli, H., and Mah, S. (2010) Understanding and Evaluating Cooperative Games. *Proceedings of the SIGCHI Conference on Human Factors in Computing Systems - CHI '10*, pages 253–262.

Gamma, E., Helm, R., Johnson, R., and Vlissides, J. (1993) Design Patterns: Abstraction and Reuse of Object-Oriented Design. *Proceedings of the 7th European Conference on Object-Oriented Programming - ECOOP '93*, pages 406–431.

Reuter, C., Wendel, V., Göbel, S., and Steinmetz, R. (2014) Game Design Patterns for Collaborative Player Interactions. *DiGRA 2014*.

Reuter, C., Göbel, S., and Steinmetz, R. (2014) A Collection of Collaborative Player Interaction Patterns. Available at: ftp://ftp.kom.tu-darmstadt.de/TR/KOM-TR-2014-01.pdf.

Rocha, J.B., Mascarenhas, S., and Prada, R. (2008) Game Mechanics for Cooperative Games. *ZON Digital Games 2008*, pages 72–80.

Smith, G., Gan, E., Othenin-Girard, A., and Whitehead, J. (2011) PCG-based game design: enabling new play experiences through procedural content generation. *Proceedings of the 2nd International Workshop on Procedural Content Generation in Games - PCGames '11*, pages 5–8.

Togelius, J., Yannakakis, G.N., Stanley, K.O., and Browne, C. (2010) Search-based Procedural Content Generation. *Springer.*

Togelius, J., Kastbjerg, E., Schedl, D., and Yannakakis, G.N. (2011) What is Procedural Content Generation?: Mario on the borderline. *Proceedings of the 2nd International Workshop on Procedural Content Generation in Games - PCGames '11*.