HOGESCHOOL VAN AMSTERDAM

PCG Quests

Project Game Technology

Rosa Corstjens, 500702627

February 2017

Glenn Hoff, Wouter van de Hauw, Michael Fuents Rodriguez, Rosa Corstjens, Guus van Gend, Kevin de Leeuw, Frederick van der Meulen & Jasper Meier

Project Game Technology
Team 5

Under the guidance of
Alexander Mulder, Product Owner
Anders Bouwer, Scrum coach
Eric Klok, Technisch consultant

PCG Quests

Project Game Technology

Rosa Corstjens, 500702627
Game Technology
Hogeschool van Amsterdam
February 2017

ABSTRACT
This research paper focuses on selecting a method for procedural quest generation that is suitable for implementation in a single player Adventure game. Methods, which have been proposed for implementing such a system, have been analyzed in a previous research by me (Corstjens, 2016). Based on that analysis, this research paper proposes a method that builds on these existing methods. Furthermore, a few complications and the implementation of this system in a game need to be addressed.

1. INTRODUCTION
A procedural content generation (PCG) system, that can generate quests in games, offers new opportunities for replay-ability and unique experiences for player and solves problems like lack of content and high costs for developers. Even though PCG quest systems have not been adopted in popular games and mainly has been a topic of research. In a previous research by me the potential of existing proposals for such a system have been already analyzed and it concluded that, while often not being complete and ready for commercial usage, important steps have already been made (Corstjens, 2016). Perhaps a combination of existing methods with some amendments would be ready for implementation in a game.

This research paper therefore proposes a method and framework that builds upon known proposals from other researchers. The goal of the proposal is to guide me and the other developers of my project team in their development of this quest system.

The previous research paper by me analyzed different methods and concluded what the responsibilities and parts of the system should be (Corstjens, 2016). The first part of this research will be roughly ordered based on those responsibilities and parts. For each of them, the different possibilities will be discussed and potential problems will be addressed.
The second part of this research focuses on the chosen methods and solutions by presenting a method for implementing a PCG quest system.

2. RELATED WORK
PCG quests is a subject that has been addressed by a multiple researchers (Doran & Parberry, 2010; Soares de Lima, Feijó & Furtago, 2014; Pita, Magerko & Brodie, 2007; Lee & Cho, 2012), showing clear interest in the possibilities these systems can offer in games. The biggest obstacle is, however, that these methods vary widely and that most of them are not ready for implementation, as my last research paper concluded (Corstjens, 2016). Even though they showed differences in implementation, most agree on the parts that make up the system. The previous research concluded that a system should consist of two main parts: a part generation quests and a part for managing quests and their system.

There are three important parts to the generation of the quest: it should have a source, a representation and there must be a generation algorithm.
Most research papers agree by stating that quests emerge from NPC's in the game world. Doran and Parberry propose the possibility of NPC's having needs that can be affected by events and interactions (2010). An unmet need would result in the start of a quest. Pita et al. suggest that NPC's could also have relations and memories (2007). A quest would start when the player prompts an NPC to give him a quest, based on recent memories.
The representation and generation of a quest are closely related, since some algorithms need a certain

data type as representation. Even though different data types for representation are used, it is agreed upon that quests are hierarchical (Soares de Lima et al., 2014). This doesn't imply that the data type used for representation should be hierarchical. Multiple method proposals use planning algorithms for generation in combination with a form of a finite state machine for representation. Planning algorithms take a random begin and end state and add action that change the current game state, until the end state is reached. The problem with this is the long search time in the fairly big search space, which makes it impractical at run time. Cho and Lee tackle this problem by using Petri nets for calculations with the game state (2012), while Soares de Lima et al. tackled it by generating only a part of the quest and continue generating when this was needed. Another option is to discard planning algorithms all the way, as proposed by Doran and Parberry (2010, 2011). They propose a tree structure as representation for a quest, which naturally support the hierarchical order of a quest. A tree is build based on a strategy (the root) consisting of a few actions to complete the strategy (root nodes). Iteratively, leaves are replaced with sub actions till the complete tree is created. This last method has been elaborated extensively, based on 3000 quests, and is implemented in a prototype, showing its potential. We prefer to build upon this last method over others due to its simplicity and applicability to many quests in known games.

Since only generating the quests won't enable a system that integrates quests in a game, managing components are needed. Multiple research papers argue about the different responsibilities these parts should have and what parts should be included. It's mostly agreed upon that a quest manager should provide an interface to the game, quests and entities in the world for communication purposes (Doran & Parberry, 2015; Soares de Lima et al., 2014). The manager should also be able to act autonomous, since it mustn't be a design tool (Doran & Parberry, 2011). This means that the manager must be able to detect the need of a new quest and, upon that moment, notice the quest generator. Once the quest generator has created a new quest the other needed content must be generated, such as items and dialogue. This can be done with another content generator, but the quest manager will communicate the need for this content. The problem of dialog generation has not yet been extensively researched and developed (Tomai, Rosendo & Salinas, 2012). Dialog is especially hard to generate in such a way that it feels relevant to the player and fits the story is Finally, the manager should be able to process changes in quests and interactions are important for quests (Soares de Lima et al., 2014), for example when a NPC dies, who was part of the quest. This includes monitoring active quests, optionally re-planning them when changes in the game world have occurred and analyzing information about the current game world for important changes. Possibly every quest instance would have its own monitor and planner that can communicate to the quest manager and vice versa (Soares de Lima et al., 2014). The quest manager should be able to communicate with the game manager, containing a representation of the current game world, to enable communicating important information about certain quests to the entities that need that information.

For the implementation of these responsibilities are multiple options. Some systems let the quest manager handle most of the responsibilities, while others divide them over different parts, like planners and monitors per quest instance. Dividing the responsibilities distributes the workload and circumvents over complication, therefor having our preference.

3. OVERVIEW PROPOSED SYSTEM

Based on my previous research and other related work, the following parts will be included in the system to support generation and managing of quests:

- **Quest manager**, a central communication and monitoring entity;
- **Quest generator**, for generating new quests;
- **Content generator(s)**, for generating needed content for the quest, such as dialogue and items.
- **Quest representation**, to have rules for constructing new quests and for monitoring or altering existing ones;
- **Database**, to store active quests;
- **Quest instance**, a concrete, active quest;
- **Player assistant**, an object keeping the player up-to-date about his objectives;
- **Source of a quest**, a source triggering the generation of a new quest.

The proposed system in this paper will consist of all these parts, implemented with knowledge and further supplemented methods from other research papers. The following chapters will discuss the detailed implementation for the proposed system, categorized under generation or managing functions.

4. QUEST GENERATION

The generation of a quest proposed in this paper is mostly based on the methods proposed by Doran and Parberry proposed in their papers from 2010, 2011 and 2015 with a few additions. To avoid referencing over and over again to these same references, when Doran and Parberry are named it implies a reference to one or more of their papers. This chapter will discuss the proposed method by discussing the source, the representation and the actual generation of a quest.

4.1. SOURCE OF A QUEST

NPC's are the source of a quest in the proposed system. This means that all quests result from unmet needs or problems with some NPC. The implementation is mostly inspired on Doran's and Parberry's system. They propose different motivations that result from needs of the NPC's. The needs of a NPC change as a reaction to events and interactions. When a need isn't met, the NPC messages the quest manager about this need, which is the motivation for the quest, so it knows a new quest should be generated, given some parameters. Based on the taxonomy by Doran and Parberry, a motivation communicates to one or more strategies and the strategy is the basis for a quest. Other parameters needed for the quest, like items, setting and other NPC's, can also be provided by the NPC in a later stadium of generating the quest.

A NPC must have a few attributes to support a system like this. To start with, they should implement the needs described by Doran and Parberry, see Appendix A for the motivations of a NPC. Since an unmet need results in a motivation, motivations and needs have the same categories. These needs must be able to chance by the cause of events or interactions. The NPC must be able to receive a message when his needs should change. The NPC itself decides how this event or interaction changes his needs. To further support parameters needed for the quest, we propose that NPC's also have the following attributes:

- Relations with other NPC's, like friends, family and enemies, to support the decision on the other NPC's needed for a quest;
- Profession or function to support a possible theme for the quest. A black smith, for example, could give weapons as rewards more often;
- Location or home city to support choosing locations for events in quests.

In all probability this list must be further extended while working on the implementation of this quest system. These attributes would give the possibility of further specification of the setting and theme of the quest, such as an alignment or the ability to store recent events to mimic short term memory.

The relations between NPC's can be implemented with a graph structure. This graph structure contains of nodes for NPC's and links for relations. Different types of links indicate the different kinds of relationships and each link has a weight, indicating the positive of negative feeling towards the other NPC. Between each two nodes can exist none, one or two relations. None relation would mean that the two NPC's don't know each other. One relation implies that one of them knows the other and two that they know each other. When the quest generator has to choose one or more NPC's to assign to the quest, it can search the graph to find the types of relations needed for the quest.

4.2. QUEST REPRESENTATION

Another key aspect is the representation of the quest for the algorithm to enable generating new quests in the same format and managing existing quests. All previous research suggests that quests are hierarchical, even though not all research uses a hierarchical data type to represent a quest. We prefer the simplistic and generic tree representation, again proposed by Doran and Parberry. This representation proved itself to be able to generate a wide range of different quests with context. Mostly because this method has already proven itself in a prototype, which many other methods have not yet, it seems to be more ready than the others for implementation in a complete game.

The tree representation would consist of one root node, the strategy of the quest. This strategy is assigned based on the motivation given as input for the quest; see Appendix A for the strategies with their corresponding motivations. The tree consists furthermore of node that represent abstract actions, also defined by Doran and Parberry and can be found in Appendix B. A strategy consists of a few abstract actions and every action itself can also be further divided in actions. The action <get>, for example, could be divided in <goto>, <get>. This enables the tree to consist of a set of actions that have to be executed in a certain order to complete the quest. Actions consist furthermore pre- and post-conditions, enabling monitors to monitor the status of a quest. Thanks to the way quest generation is done, the order of the actions can be found by reading the tree in preorder-traversal.

A tree consisting of abstract actions represents an abstract quest. This abstract quest needs to be 'filled in' with content to create a concrete quest, which is ready to be started by the player. To achieve this, for each abstract action, its references to NPC's, items, dialog and locations must be replaced with actual assets to convert the action to a concrete action. When the quest consists only of concrete actions, it represents a concrete quest.

4.3. QUEST GENERATION

The quest generation process will be triggered by the quest manager, upon an unmet need of a NPC. The generation algorithm takes as input the following parameters:

- Source NPC of the quest;
- Motivation for the quest;
- The player level to avoid selecting too hard content in the concretization process of the quest.

Based on the input, the quest will be generated. The exact implementation is left open for development, since it has to use other components of the game. The algorithm will follow along these lines:

1. Select randomly a strategy matching the motivation;
2. Instantiate the strategy as root node of the tree and its subordinate actions as its child nodes;
3. Select randomly a length for the quest (short, medium, long);
4. Select an amount of points based on the length of the quest and the player level;
5. Distribute the points randomly over the child nodes;
6. Iteratively replace random nodes with their own subordinate actions and redistribute the points, until there are no points to distribute left;
7. Assign NPC's to the references in the abstract actions, chosen from the relation network of the source NPC;
8. Assign locations to the references in the abstract actions, chosen from the locations where assigned NPC's live;
9. Assign entities, such as chests, to the references in the abstract actions, based on entities that reside in the given locations;
10. Prompt content generator to generate needed items;
11. Prompt content generator to generate needed dialog;
12. Write the concrete quest to the database and notice the quest manager.

4.4. GENERATION OF OTHER CONTENT

The quest generation process mostly relies on existing content, like NPC's and locations, and depends on them to ensure that different parts of the quest feel interconnected to the player himself. Other content, such as quest specific items and dialog, need to be generated.

The generation of this quest specific content will be handled by a separate system. The game, that implements this proposed quest system, already consists of an item generation system. It follows a Diablo 3-like approach in its generation, by randomly assigning a quality, a name and stats between ranges and constraints to items. This system can be altered to serve the quest system as well with the following changes:

- Quest flag, whether the item belongs to a quest;
- Unique flag, whether the item could be spawned again or only once;
- Slots for extra stats, to make the item more valuable to the player than a normal item.

The generation of dialog is not yet implemented in the game and it lies far outside of the scope of this research, since it concerns procedural storytelling and natural language processing. This problem will be addressed with the use of simple boilerplate dialog, in which each talk action will have a few possible template sentences to will in with the content assigned to the quest.

5. QUEST MANAGING

Next to generating new quest, existing quests must be managed. The central managing part is the quest manager, which satisfies the concluded responsibilities in related work. These responsibilities are formulated as following:

- Keep up which entities should receive messages about which quests and actions;
- Receive messages about the game state when changes occur;
- Receive messages from the planners and monitors of quest instances when changes occur;
- Send messages about the game state or quests when changes occur;
- Decide when a new quest should start;
- Prompt the quest generator to generate a new quest and receive a new quest;
- Prompt the content generator to generate content needed for a quest;
- Add a new quest to a source containing active quests;
- Access information of active quests when planners and monitors of quest instances need it;

- Alter information of active quests when planners and monitors of quest instances need it.

The quest manager is therefore a central interface for communication between the game world and quests. Other parts of the system communicate with the quest manager or serve as a source of information.

5.1. QUEST INSTANCE

When a quest is generated, it is stored in the database and the quest manager is noticed. The quest manager instantiates the quest and assigns a new planner instance to it. His responsibility is to adjust the quest when that is required based on events and interactions in the world. Notifications on events and interactions are received from the game manager, through the quest manager. Since the quest manager knows which resources are assigned to which quests, it only communicates information to the quest planners that need it. When, for example, an important NPC for that quest died or the player destroyed a quest item, the planner receives a notification and must determine the following:

- Have certain parts of the quest become impossible? If not, ignore the notification;
- Identify the actions that make certain parts of the quest impossible;
- Identify the resources of those actions that make the certain parts of the quest impossible;
- Are the resources commutable with other resource? If not, set the identified action(s) to complete and flag them as failed. This way, their post-conditions won't be of influence and the other parts of the quest can still be completed.

Resources are commutable if there is an alternative available in the game world or if it concerns a quest item that is not flagged as unique.

As soon as the player starts the quest, the quest manager assigns a monitor instance to the quest. Note that the planner is needed before the quest is actually started, since events and interactions can alter the quest before it started. The monitor keeps track of the progression of the quest. It communicates to the quest manager whether an action has been completed with success or failure, which in turn updates the quest information in the database and reacts on the information by generating resources or notifying the game manager about the post-conditions of the action. For example, a NPC may need to be in his home after a certain action is completed. The monitor instance also communicates this same information to the player assistant, discussed in chapter 5.4., to notify the player about his current goals and progression.

These two components, the quest planner and monitor, enable self-containing quests and reduce the work-load on the quest manager.

5.2. DATABASE

To enable storing, adjusting and managing quests, the quest system can access a database. Since the implementation of the database depends almost entirely on design choices, this chapter doesn't go in-depth about the database types or connection the developers should provide. It rather discusses attributes that must be stored and the flow of communication.

The quest manager is the only component that can store, alter and retrieve data from the database. This centralizes the checks that have to be done before information goes to of comes from the database. The quest manager implements methods, which enable other components of the system asking for information.

The database itself consists of multiple tables, containing information about all different resources needed for quests. This implies that information concerning the game, not the quest system, should also be stored in this database, since NPC's are needed for quest generation, but are not necessarily part of a quest. Therefore, the database design also depends solely on the game itself. To provide a stepping stone for designing the database itself, the following information should be included:

- Quests, with a title, tree representation consisting of concrete actions, start and finish conditions, rewards on completion;
- Motivations, with the corresponding strategies;
- Strategies, with the corresponding actions;
- Elementary actions, with the corresponding sub actions;
- Concrete actions, with a state, post- and preconditions, concrete assets;
- NPC's, with current needs, relations, function(s) and home town. Note that this list can be extended;
- Locations, with a name and position in the world. Optionally, information about past events in this location can be stored;
- Items, with an item type, a gold value, statistics, a quest and unique flag, and a name.

5.3. GAME REPRESENTATION

It is essential for the proposed quest system to read state the current game world and all entities it contains. Without, it wouldn't be possible to react on the game world and communicate information about

it effectively. It is recommended to use the database for the game representation. If the game manager is also allowed to access the database, it can alter the state of the world by for example adding items or changing information about NPC's. This way the database will always consist of the current game world representation. If the altered information concerns a quest resource, the quest manager must be noticed about the change.

5.4. PLAYER ASSISTANT

Without a component that communicates the current goals and states of quests to the player, the player can't know what to do. Therefore, one player assistant must be implemented. This assistant receives information from the quest manager and indirectly from the monitors to enable support for the player.

For each active quest the player assistant shows information, like the name, start location or NPC, rewards, current goals and completed goals. For each completed quest the same information is shown without current goals. The player assistant also informs the player about quests he can start. Quests that can be started should only be represented with a name and start location.

CONCLUSION

The proposed quest system will prove its efficiency and effectiveness in the mentioned game project at the Hogeschool van Amsterdam. While there are still design choices that have to be made for implementing this system, they're not likely to be big obstacles for the development team.

For further development of this quest system the problem of story-telling generation and dialog generation need to be tackled. These problems must be addressed in further research. But the existing proposal already promises for a more interactive and responsive quest system, which will enrich the gameplay of games consisting of quests.

REFERENCES

Corstjens, R. (2016). *PCG Quests, on a journey to discover the possibilities of a system procedurally generating quests for MMORPG's*. Student's research paper. Amsterdam: Author.

Doran, J. & Parberry, Ian. (2010). *Towards Procedural Quest Generation: A Structural Analysis of RPG Quests*. Retrieved from http://larc.unt.edu/techreports/LARC-2010-02.pdf.

Doran, J. & Parberry, Ian. (2011). *A Prototype Quest Generator Based on a Structural Analysis of Quests from Four MMORPGs*. Retrieved from https://larc.unt.edu/techreports/LARC-2011-02.pdf.

Doran, J. & Parberry, Ian. (2015). *A Server-Side Framework for the Execution of Procedurally Generated Quests in an MMORPG*. Retrieved from http://larc.unt.edu/techreports/LARC-2015-01.pdf.

Lee, Y. & Cho, S. (2012). *Dynamic quest plot generation using Petri net planning*. Retrieved from http://rps.hva.nl:2413/citation.cfm?id=2425304&CFID=565976299 &CFTOKEN=43410985.

Soares de Lima, E., Feijó, B. & Furtado, A. (2014). *Hierarchical generation of dynamic and nondeterministic quests in games*. Retrieved from http://rps.hva.nl:2413/citation.cfm?id=2663833&CFID=565976299&CFTOKEN=43410985.

Tomai, E., Rosendo, S. & Salinas, D. (2012). *Adaptive Quests for Dynamic World Change in 21 MMORPGs*. Retrieved from http://rps.hva.nl:2626/10.1145/2290000/2282402/p286-tomai.pdf.

Pita, J., Magerko, B. & Brodie, S. (2007). *True story: dynamically generated, contextually linked quests in persistent systems*. Retrieved from http://rps.hva.nl:2413/citation.cfm?id=1328228&CFID=565976299&CFTOKEN=43410985.

APPENDIX A, MOTIVATIONS AND CORRESPONDING STRATEGIES FOR QUESTS

Motivation	Description	Corresponding strategies	Sequence of actions
Knowledge	Information known to a character	Deliver item for study Spy Interview NPC Use an item in the field	<get><goto> give <spy> <goto> listen <goto> report <get> <goto> use <goto> <give>
Comfort	Physical comfort	Obtain luxuries Kill pests	<get> <goto> <give> <goto> damage <goto> report
Reputation	How others perceive a character	Obtain rare item Kill enemies Visit a dangerous place	<get> <goto> <give> <goto> <kill> <goto> report <goto> <goto> report
Serenity	Peace of mind	Revenge, Justice Capture criminal (1) Capture criminal(2) Check on NPC(1) Check on NPC(2) Recover lost/stolen item Rescue captured NPC	<goto> damage <get> <goto> use <goto> <give> <get> <goto> use capture <goto> <give> <goto> listen <goto> report <goto> take <goto> give <get> <goto> <give> <goto> damage escort <goto> report
Protection	Security against threats	Attack threating entities Treat or repair (1) Treat or repair (2) Create diversion (1) Create diversion (2) Recover lost/stolen item Rescue captured NPC	<goto> damage <goto> report <get> <goto> use <goto> repair <get> <goto> use <goto> damage <goto> repair <goto> defend
Conquest	Desire to prevail over enemies	Attack enemy Steal stuff	<goto> damage <goto> <steal> <goto> give
Wealth	Economic power	Gather raw materials Steal valuables for resale Make valuables for resale	<goto> <get> <goto> <steal> repair
Ability	Character skills	Assemble tool for new skill Obtain training materials Use existing tools Practice combat Practice skill Research a skill (1) Research a skill (2)	repair use <get> use use damage use <get> use <get> experiment
Equipment	Usable assets	Assemble Deliver supplies Steal supplies Trade for supplies	repair <get> <goto> <give> <steal> <goto> exchange

Based on the taxonomy of Doran and Parberry (2011).

APPENDIX B, ACTIONS

Action	Pre-condition	Post-condition
Capture	Somebody is there	They are your prisoner
Damage	Somebody or something is there	It is more damaged
Defend	Somebody or something is there	Attempts to damage it have failed
Escort	Somebody is there	They will now accompany you
Exchange	Somebody is there, they and you have something	You have theirs and they have yours
Experiment	Something is there	Perhaps you have learned what it is for
Explore	None	Wander around at random
Gather	Something is there	You have it
Give	Somebody is there, you have something	They have it, you don't
Goto	You know where to go and how to get there	You are there
Kill	Somebody is there	They're dead
Listen	Somebody is there	You have some of their information
Read	Something is there	You have information from it
Repair	Something is there	It is less damaged
Report	Somebody is there	They have information that you have
Spy	Somebody or something is there	You have information about it
Stealth	Somebody is there	Sneak up on them
Take	Somebody is there, they have something	You have it and they don't
Use	There is something there	It has affected characters or environment

Based on the taxonomy of Doran and Parberry (2011).