In Search of Patterns: Disrupting RPG Classes through Procedural Content Generation

Alex Pantaleev
Department of Computer Science
SUNY Oswego
Oswego, NY 13126
alex@cs.oswego.edu

ABSTRACT

This paper presents a first attempt at exploring the search space of Role-Playing Game (RPG) skill systems, with the hope to find stable patterns outside of conventional RPG classes. At the foundation of the experiment is a small text-based game that allows human players to enter RPG combat and uses an evolutionary algorithm to generate and suggest new character abilities to them. The content representation was carefully chosen to be simultaneously simple and expressive. The evaluation function scores abilities based on their use by players. While the game is far from finished, preliminary test results are encouraging.

Categories and Subject Descriptors

K.8.0 [**Personal Computing**]: General—*Games*

General Terms

Design, Experimentation

Keywords

Games, procedural content generation, game design, role-playing games

1. INTRODUCTION AND BACKGROUND

Role-playing games (RPGs) have traditionally employed a class-based system, starting with the first edition of *Dungeons and Dragons* (TSR 1974), which set the staples of the genre. A character class aggregates several abilities or skills, and determines not only the role of the respective character in the game world, but also his or her potential capabilities. A class can, therefore, be perceived as a set of constraints on a character's development, confining the character's skills to a strict subset of the space of all skills. Despite its inherent limitations, the class-based RPG tradition is strong today, following the success of massively multiplayer online RPGs such as *World of Warcraft* (Blizzard Entertainment 2004).

A skill-based (or classless) RPG, on the other hand, does not artificially constrain the set of abilities that a character can acquire in the progress of the game. Users are free to mix and match all available skills and find their optimal point in the skill space of the game with no designer-imposed limitations. While there have been several highly successful commercial games to employ a skill-based system (notable examples include *Asheron's Call* (Turbine Entertainment Software 1999) as well as games developed by Bethesda Game Studios, such as *Fallout 3* (2008) or *The Elder Scrolls* series), professional designers prefer the tighter control over user experience that classes allow them [9].

In addition, RPG players usually build their characters according to certain templates even in skill-based games [1]. Such templates, while informal in nature, resemble design patterns as defined in architecture [2], software engineering [6], and game design [3]: they describe solutions to common problems that occur in the RPG combat domain, and form a language for designing character builds. While this language is a result of accumulated best practices for creating optimal character parties in RPGs, it is also a result of the culture of designer-imposed classes.

This paper describes an initial experiment to explore the search space of skill-based RPG systems, with the hope of providing insight into the templates that users prefer and finding character builds that do not conform to existing patterns. To this end I have built a small game that uses an evolutionary algorithm to search for optimal combinations of character abilities in a party in standard RPG combat. The content representation was chosen carefully to allow the modeling of important RPG tropes, such as health potions, purely on the basis of abilities in order to simplify the search. The interactive and implicit evaluation function is based on players' use of abilities while they play the game. The algorithm procedurally creates new abilities, which players can swap with an existing ability of a character in their party. Some of the characters' starting abilities are crafted to conform to well-known patterns, which, however, do not contribute to evolution. Hence, users are encouraged to explore unfamiliar combinations of abilities, rather than strive towards known character builds.

1.1 Situating the Algorithm

The game generates character abilities procedurally using an evolutionary search technique. Hence, it can be categorized within the field of search-based procedural content generation, of which Togelius et al. [22] recently published an ex-

PCG 2012 Raleigh, North Carolina, USA

tensive taxonomy and survey. According to the distinctions the taxonomy makes, the game presented here uses a direct encoding in the genotype-to-phenotype mapping, since the genome variables (the genotype) map to attributes of the character abilities (the phenotype). In addition, the algorithm ranks abilities on the basis of their usage by players, which makes its evaluation function interactive and implicit. The abilities themselves are optional content, because players can choose to avoid a certain ability. Finally, the content generation algorithm is online, since it is performed during the runtime of the game.

1.2 Search-Based PCG in Games

The game described in this paper allows players to continuously replace the abilities of their characters with procedurally generated ones. While modeling weapons was explicitly avoided, this concept is nonetheless similar to games in which players are allowed to exchange their weapons with procedurally generated ones. The most prominent example of this kind is *Galactic Arms Race*, a multiplayer game by Hastings et al. [8]. In this game players can choose to exchange any of the limited number of weapons of their spaceship with procedurally generated ones found in the game world. The fitness of a particular weapon is a function of its usage by all players logged into the game server. The game uses cgNEAT, a search-based algorithm, to evolve the connection topology of a neural network for each weapon's particle system [7].

Borderlands (Gearbox Software 2009) is a commercial first-person shooter (FPS) in which weapons are also procedurally generated. While both *Galactic Arms Race* and the experiment presented here use an evolutionary algorithm to produce the weapons and abilities, respectively, in *Borderlands* weapons are randomly created, and then balanced with respect to the player's level.

Search-based PCG for games is a new and promising field. Successful applications include generating tracks and levels [12, 15, 16, 19], maps [20], buildings [14], camera control [5], unit types [13], and rule systems and mechanics for games [4, 10, 17, 21].

1.3 Design Patterns

This experiment aims to disrupt and add to existing *RPG character build* patterns, which have arisen naturally and informally among the player community, partly through the proliferation of designer-imposed class systems. One potential future development is the identification and formalization of new patterns in this domain. This is similar to efforts to identify and enumerate game design patterns [3], quest and level design patterns in RPGs [18], and FPS level design patterns [11].

2. EXPERIMENTAL SETUP

The game allows human players to choose abilities for their party of four characters. Their party is then matched against another human-controlled party of four characters in standard RPG combat. For reasons of simplicity the game does not represent two-dimensional space; every character is reachable from every other character for both combat and skill application purposes. The game combat thus resembles that of room-based Multi-User Dungeons (MUDs) or classic Japanese RPGs.

The game implicitly keeps a fitness score of each ability, which is based solely on how often an ability is used. The fitness evaluation function does not attempt to measure how *effective* an ability is, because it is assumed that players will naturally use the most effective ability in a given situation. Every character has four abilities. A character can only use one ability every ten ticks. Hence, players need to choose the best ability a character can apply at a certain moment, because that same character will only be able to apply one of his or her four abilities again after some time has passed.

After a battle is over, the game procedurally generates new abilities. The players are allowed to choose any of those and substitute them for any abilities that characters in their party already have. Once players are satisfied with their characters, another round of combat commences.

2.1 Content Representation

Character abilities are a central concept to the experiment. A character in the game has a set of parameters, all of which are important for some aspect of combat. The parameters are modeled around standard RPG tropes, and will be discussed in detail later. An ability is a means to influence a character's parameter in either a positive or a negative direction over a period of game time; such influences are called effects.

An effect has four attributes: a parameter to influence, a value to influence it by (positive or negative), a duration in terms of game time, and an allowed target: either a friend or an enemy to the player. When an effect is applied to an allowed target character, the value of the effect is added to the target character's parameter as specified by the effect. After the effect's duration has expired, the applied effect disappears from the target character, meaning that the effect's value is subtracted from the respective parameter. This model enables the expression of many RPG tropes. A health potion, for example, can be modeled by an effect that can be applied to the health regeneration parameter of a target character, with a relatively large value and a duration of 1. During the next game tick the health points of the respective character will be increased by the health regeneration value, and the health regeneration parameter itself will be reset to its old value, because the effect will expire.

An effect is strictly defined as either a benefit or a cost to the character that applies it. All character parameters in the game are beneficial to the respective character when their values increase, and detrimental when their values decrease. Respectively, an effect is a benefit if its value is positive and its target is friendly, and a cost if its value is negative and its target its friendly; the cost/benefit determination is inverted for enemy targets.

Every ability has exactly two effects, which are applied together when the ability is used: one benefit, and one cost. The two effects together have a total of eight attributes. These eight attributes form the genome that the genetic algorithm uses to evolve new abilities.

In addition, all effects have an integer-valued worth in the game. When an ability is created, it compares the worths of its benefit and cost and adjusts their value attributes, if necessary, to prevent the players from climbing the hill of infinite benefit at zero (or negative) cost. The balancing function impacts the value attributes (hence, the genome) directly, and is the last step the evolutionary algorithm takes before it offers the evolved abilities to the players. That impact is significant for evolutionary purposes if the balanced ability is selected for evolution later.

2.2 Game Mechanics

A character in the game has health points, magic points, and eight parameters that can be influenced by abilities. Those are:

- Maximum health points. This parameter sets the upper boundary of a character's health points.
- Maximum magic points. As above, for magic.
- Health regeneration. The value of this parameter is added to a character's health points each tick.
- Magic regeneration. As above, for magic.
- Damage. The damage that a character deals when it attacks.
- Armor. The protection a character has from damage when attacked.
- Dodge. The chance of evasion a character has when attacked.
- Speed. How soon the character can attack again.

Each parameter has a minimum and a maximum value. Abilities cannot push parameter values outside of their allowed ranges. All characters start with the same initial values of all parameters. Hence, any differences among characters will be a result purely of effects applied to those characters.

Whenever a character's *health* points drop to zero, the character is incapacitated and cannot enter the current battle any more. Whenever a character's *magic* points drop to zero, all effects that are a result of the character's abilities are erased.

A character can only attack once every n game ticks, where n is determined by the character's speed. A character can attack any character of the opposing team. The target of the attack has a chance to dodge that depends on its Dodge parameter, and if unsuccessful, its health points decrease by an amount determined by the attacking character's Damage and the defending character's Armor.

A character can only use an ability once every ten game ticks. This number cannot be influenced in any way during the course of a battle. When an ability is used, the player has to choose a target for the beneficial effect of the ability and a target for the detrimental effect of the ability. Those targets can be any characters in the battle, as long as they conform to the respective effect restrictions (friend or enemy).

The representation and mechanics described above are capable of modeling a large array of standard RPG tropes, and were specifically chosen to be able to express most of the character build templates inspired by RPG classes. The starter parties of four characters have one ability per character that is crafted after these templates. For example, one character has a "damage dealer" ability. This is a pattern that arose from the fighter class, and the respective ability has applied beneficial effects to increase its damage parameter at a magic point cost. The other three characters have a "nuker" (mage), a "healer" (cleric), and a "fragile speedster" (rogue) ability, respectively.

2.3 Evolving Character Abilities

To ensure diversity, the game keeps a population of abilities above a certain fitness value. While only two players can currently participate in a game, an archive of evolved abilities is kept between game instances. Characters start with a mixture of random abilities and pre-crafted abilities, as described previously. The latter do not contribute to evolution.

Once a battle ends, the game ranks all eligible characters' abilities according to their accumulated fitness, and uses a roulette-wheel selection scheme to select a population of parents. Since the variables in a chromosome are of different types (some are enumerated, some are integer-valued), the algorithm then performs discrete recombination, followed by a low-probability mutation, to produce the offspring. There is also a small chance that a random member of the archive will be added to the offspring.

The abilities evolved in this way are then presented to the players. The players can attach any ability from the offspring to any of their characters, discarding a previous ability (which, however, may already have been placed in the archive based on its fitness). Any abilities that were not chosen are discarded.

3. RESULTS AND DISCUSSION

The first tests of the game resulted in the need to tweak the balancing function due to the strategy to gather abilities that attack directly the enemy health point regeneration value at any non-health-related cost, following the "nuker" character build. While this strategy was not completely dominant, it still dominated a large number of other strategies. After the balancing function was reevaluated and adjusted, four different users played a total of six games, with an average of approximately fourteen battles per game. Several interesting patterns emerged.

One of those concerns the durations of effects. A character can only use one ability every ten ticks, but there is no restriction on the duration of an effect, as long as the duration is positive. It is quite possible, with the correct combination of abilities, to have applied effects that amplify with time. Hence, one emergent strategy was to somehow survive the first few ticks of battle, continuously applying low-value long-term benefits (either positive effects on friends or negative ones on enemies), and win the end game. This often required the sacrifice of two or more characters from a party.

Another interesting pattern, closely related to the previous one, exploits a game mechanic: when a character's health reaches zero the character does not die (and is not removed from the game), and is instead incapacitated. This mechanic

exists to explain the fact that all characters are available in consequent battles. An incapacitated character, while incapable of affecting the game in any way after incapacitation, will not have any previously applied effects erased from the game, either. Hence, another interesting pattern was characters applying long-term effects and then becoming incapacitated, in effect sacrificing themselves. An extreme version of this pattern is an ability that applies a long-term effect as its benefit, and reduces the health of the character that uses it to zero as its cost.

A third pattern emerged that directly counters the previous one. Magic points were initially included in the game to be used as costs to counter beneficial effects, according to RPG tropes, and every character's magic point regeneration parameter is initially set to a positive value. However, the game mechanic that erases all previously applied effects by a character whose magic points drop to zero, in combination with the previous two patterns, converted magic points to a valuable commodity. Abilities that attacked or defended magic point regeneration became highly sought after.

Some of the standard patterns, such as healing, also arose during these preliminary tests. Since the initial healing ability did not participate in fitness calculations, some character parties kept it throughout the tests, while others developed separate healing abilities, with larger benefits (and costs), and applying the costs to parameters other than magic points.

The emergence of the new patterns described in this section is very encouraging at this early stage. With the further development of the game and the participation of more players, providing fresh perspectives, more patterns that disrupt the traditional character builds are likely to arise. While it can be argued that these patterns arose as a result of the mechanics of this specific game, they can nevertheless inform the design of future games.

4. CONCLUSIONS AND FUTURE WORK

This work described an initial experiment to explore the search space of skill-based RPG systems through procedural content generation. To that end, a small game was developed that allowed players to participate in RPG combat. Preliminary results are promising, and several interesting patterns were identified.

An obvious avenue for improvements to the game is making it more visually attractive in order to increase the number of testers. As an example, the interface can be improved by the addition of graphics, since it is currently text-based after the MUD genre: users type commands in a terminal window to control their characters.

A more interesting improvement is to develop a continuously running server for the game and allow an unlimited number of players to participate. Currently, the largest possible parent population size is thirty-two (two character parties, four characters with four abilities in each one). More simultaneous players, with a server aware of them, will increase the parent population and produce interesting abilities reflecting combinations of player styles.

Finally, the game mechanics could be expanded to cover a much larger subset of RPG tropes such as sneaking, traps, ranged attacks, etc. This will also entail converting the combat from room-based to spatial, which will in turn allow the emergence of new ability patterns to reflect tactics on a two-dimensional map.

5. REFERENCES

[1] Examples of archetypal class roles. `http://tvtropes.org/pmwiki/pmwiki.php/Main/AnAdventurerIsYou`.

[2] C. Alexander, S. Ishikawa, and M. Silverstein. *A pattern language*. Oxford University Press, 1977.

[3] S. Björk and J. Holopainen. *Patterns in game design*. Cengage Learning, 2005.

[4] C. Browne. *Automatic generation and evaluation of recombination games*. PhD thesis, Queensland University of Technology, 2008.

[5] P. Burelli and G. Yannakakis. Combining local and global optimisation for virtual camera control. In *IEEE Symp. Computational Intelligence and Games*, 2010.

[6] E. Gamma, R. Helm, R. Johnson, and J. Vlissides. *Design Patterns*. Addison-Wesley, 1995.

[7] E. Hastings, R. Guha, and K. Stanley. Neat particles: Design, representation, and animation of particle system effects. In *IEEE Symp. Computational Intelligence and Games*, 2007.

[8] E. Hastings, R. Guha, and K. Stanley. Automatic content generation in the galactic arms race video game. *IEEE Transactions on Computational Intelligence and AI in Games*, 1(4):245–263, 2009.

[9] E. Heimburg. Classes vs. open skill systems. `http://www.eldergame.com/2011/01/classes-vs-open-skill-systems/`.

[10] V. Hom and J. Marks. Automatic design of balanced board games. In *AAAI Conf. Artificial Intelligence and Interactive Digital Entertainment*, 2007.

[11] K. Hullett and J. Whitehead. Design patterns in fps levels. In *ACM Int. Conf. Foundations of Digital Games*, 2010.

[12] D. Loiacono, L. Cardamone, and P. Lanzi. Automatic track generation for high-end racing games using evolutionary computation. *IEEE Transactions on Computational Intelligence and AI in Games*, 3(3):245–259, 2011.

[13] T. Mahlmann, J. Togelius, and G. Yannakakis. Towards procedural strategy game generation: Evolving complementary unit types. *Applications of Evolutionary Computation*, pages 93–102, 2011.

[14] A. Martin, A. Lim, S. Colton, and C. Browne. Evolving 3d buildings for the prototype video game subversion. *Applications of Evolutionary Computation*, pages 111–120, 2010.

[15] C. Pedersen, J. Togelius, and G. Yannakakis. Modeling player experience for content creation. *IEEE Transactions on Computational Intelligence and AI in Games*, 2(1):54–67, 2010.

[16] N. Shaker, J. Togelius, G. Yannakakis, et al. The 2010 mario ai championship: Level generation track. *IEEE Transactions on Computational Intelligence and AI in Games*, 3(4):332–347, 2011.

[17] A. Smith and M. Mateas. Variations forever: Flexibly generating rulesets from a sculptable design space of mini-games. In *IEEE Symp. Computational Intelligence and Games*, 2010.

[18] G. Smith, R. Anderson, B. Kopleck, Z. Lindblad, L. Scott, A. Wardell, J. Whitehead, and M. Mateas. Situating quests: Design patterns for quest and level design in role-playing games. In *Int. Conf. Interactive Digital Storytelling*, 2011.

[19] N. Sorenson, P. Pasquier, and S. DiPaola. A generic approach to challenge modeling for the procedural creation of video game levels. *IEEE Transactions on Computational Intelligence and AI in Games*, 3(3):229–244, 2011.

[20] J. Togelius, M. Preuss, N. Beume, S. Wessing, J. Hagelback, and G. Yannakakis. Multiobjective exploration of the starcraft map space. In *IEEE Symp. Computational Intelligence and Games*, 2010.

[21] J. Togelius and J. Schmidhuber. An experiment in automatic game design. In *IEEE Symp. Computational Intelligence and Games*, 2008.

[22] J. Togelius, G. Yannakakis, K. Stanley, and C. Browne. Search-based procedural content generation: A taxonomy and survey. *IEEE Transactions on Computational Intelligence and AI in Games*, 3(3):172–186, 2011.