Polymorph: A Model for Dynamic Level Generation

Martin Jennings-Teats Gillian Smith Noah Wardrip-Fruin

Expressive Intelligence Studio
University of California Santa Cruz
1156 High Street, Santa Cruz, CA 95064
{mjennin1, gsmith, nwf}@soe.ucsc.edu

Abstract

Players begin games at different skill levels and develop their skill at different rates—so that even the best-designed games are uninterestingly easy for some players and frustratingly difficult for others. A proposed answer to this challenge is Dynamic Difficulty Adjustment (DDA), a general category of approaches that alter games during play, in response to player performance. However, nearly all these techniques are focused on basic parameter tweaking, while the difficulty of many games is connected to aspects that are more challenging to adjust dynamically, such as level design. Further, most DDA techniques are based on designer intuition, which may not reflect actual play patterns. Responding to these challenges, we have created Polymorph, which employs techniques from level generation and machine learning to understand level difficulty and player skill, dynamically constructing levels for a 2D platformer game with continually-appropriate challenge. We present the results of the user study on which Polymorph's model of level difficulty is based, as well as a discussion of the unique features of the model. We believe Polymorph creates a play experience that is unique because the changes are both personalized and structural, while also providing an example of a new application of machine learning to aid game design.

Introduction

The classic 2D side-scrolling platformer is a genre of games that focuses on jumping dexterity and precise timing to get past obstacles in fairly linear levels; for example, Super Mario Bros (Nintendo, 1985). The game levels are designed to be difficult and unforgiving, so the player is only able to complete a level after playing it partway through multiple times to learn the exact necessary pattern of actions. This genre of game has been very popular, but it cannot be said to cater to every player's experience and abilities. This is one example of the types of problems that can be addressed with Dynamic Difficulty Adjustment (DDA).

Polymorph is a 2D platformer game that generates and adjusts its levels during play as a means of DDA. Specifically, rather than being authored by hand, game levels are procedurally generated as the player moves through the level, one chunk at a time as needed. The generation of these chunks is customized to create a difficulty curve that matches the player's performance, so that each player is presented with a level that provides a challenge appropriate to their skill. This is not to say that the player will never die in a tough section or breeze through an easy section, but Polymorph corrects for this in the next section, in an attempt to avoid difficulty-related player frustration and boredom.

We tackle the DDA problem by creating a machine-learned model of difficulty in 2D platformer levels along with a model of the player's current skill level. These models are gleaned with a Multilayer Perceptron from play traces. The play traces were collected with a web-based tool that asks users to complete multiple short level segments and rate them by difficulty. Polymorph uses the difficulty models to select the appropriate level segment for a player's current performance. The level segments are generated automatically using a variation on the work of Smith et al. (2009), which is described in more detail below.

This paper shows how a game can be designed to accommodate the skill and experience of every individual player by incorporating machine learning techniques and dynamic level generation. This is an advance on prior work in dynamic difficulty adjustment, which has for the most part avoided adaptive level design, and in procedural level generation, which has mainly focused on creating full levels for replayability. Polymorph consists of a data collection tool, a level generator, a game engine, and a learned model of level difficulty.

Related Work

Dynamic Difficulty Adjustment

Game designers nearly always strive to create games in which the difficulty of the obstacles presented to the player is appropriate for the player's skill level. As a player's skill improves through practice, a well designed game will present more formidable difficulties so that the player is never bored by overly easy gameplay or frustrated by overly difficult gameplay. Both game designers and scholars have referenced Csikszentmihalyi's concept of "flow" to describe the player state in this approach to games (Csikszentmihalyi, 1990; Chen, 2007; Fullerton et al., 2004; Juul, 2009). This in itself is a very challenging design task however, and game designers spend great effort making sure that their game is well balanced so that challenges will be appropriate for players' abilities. Even so, designers are usually not able to accommodate every player's skill level, and frustrating mismatches between a player's skill and a game's difficulty are common (Hunicke, 2005; Phillips, 2009).

As described briefly above, Dynamic Difficulty Adjustment (DDA) is a term for techniques in which games automatically alter themselves in some way to better fit the skill levels of the players. This is common in the genre of racing games with the practice of "rubber banding", wherein players in last place are granted an increased maximum speed (Hunicke, 2005). DDA is rarely used in other game genres, but there are some notable exceptions. One of the most complex examples of DDA in a commercial game is the first-person shooter SiN Episodes (Ritual Entertainment, 1998). It uses a statistical model of player performance with "adviser" sub-systems that adjust attributes such as the number of concurrent enemies, the damage and accuracy of the enemies' weapons, and the enemies' tendency toward throwing grenades (Kazemi, 2008). Hunicke (2005) created the Hamlet system, which uses sets of probabilities to determine the appropriate time to intervene in a first-person shooter by giving the player more ammo or a health boost. Also, Olesen et al. (2008) created an AI opponent for a real-time strategy game, using an evolutionary algorithm to alter the opponent's strategy. What these DDA approaches all have in common is that the intervention into the game is primarily through a numeric attribute adjustment. In contrast, the dynamic changes made by Polymorph are structural rather than numeric in nature.

An alternative method of intervention, on which this paper is focused, is the modification of the level design. For example, Left 4 Dead (Valve Software, 2008) changes the location and frequency of spawn points for enemies and items based on player performance, which can have a significant impact on player experience but is not a substantial change in the structural design of the levels (Booth, 2009). Pedersen et al. (2009) created a version of Infinite Super Mario Bros (Persson) from which they derived a statistical model of player challenge and frustration, among other emotional states. Using their evolutionary algorithms, this model could be used to generate levels for a particular level of player challenge, which is the closest work (of which we are aware) to the approach of Polymorph. However, it is not designed to be dynamic during play, unlike Polymorph, which generates sections of a level ahead of the player's movement, allowing a level to change in difficulty from start to finish in response to changes in the player's performance.

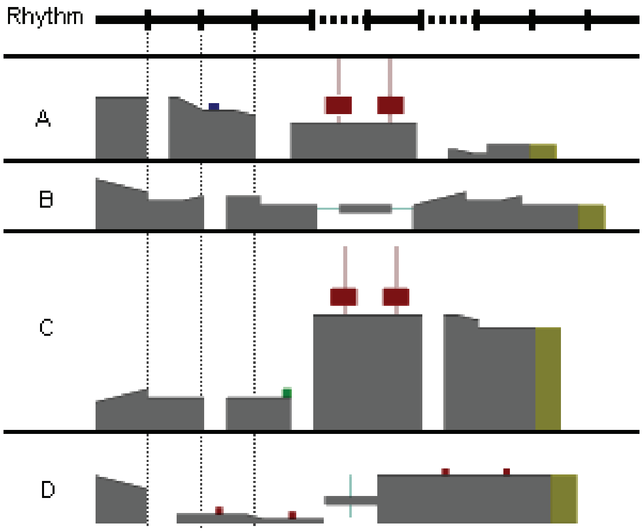

Figure 1 – Several possible mappings of level geometry onto a particular rhythm of player action.

Procedural Level Generation

Procedural level generation has been used in games for decades, with popular RPGs such as Rogue (Toy and Wichman, 1980) and Diablo (Blizzard Entertainment, 1997), as well as some 2D platformers such as Spelunky (Yu, 2009). These games typically work by fitting together hand-authored level chunks into random combinations. Pedersen et al.'s (2009) variation on Infinite Super Mario Bros also works on this principle, combining level chunks to create a level that is measured according to a statistical model of the emotions it would evoke in a player. This work, along with Togelius et al.'s (2007) work on generating tracks for racing games, uses an evolutionary algorithm to iteratively create similar levels with slight modifications. These approaches to level generation differ from Polymorph by generating geometry in larger granularity—as well as differing from Polymorph's unique learning features, described later.

Smith et al. (2009) created a generator for 2D platformer levels based on a model of player action-rhythm, which is the basis for the level generation done in this project. Their approach starts with a rhythm of desired player actions, such as run, jump or wait. This generated rhythm is that which the player feels with her hands while playing and describes the pacing of the level. The generator then chooses from sets of geometry that can fulfill each of these actions—the same starting rhythm can produce many distinct level designs depending on the geometry selected for each action, as shown in Figure 1. Because the generation is based on player actions and their associated level geometry, it has a finer granularity of control over the level design than the previously mentioned techniques which use hand-authored level chunks. All of these strategies for level generation have been offline full-level generation techniques, meaning that they create an entire playable level as a whole—usually ahead of time, rather than generating parts of the level during play (Togelius et al., 2010). This is because a primary motivation for procedural level generation has been to create improved replayability for a game, which can be accomplished by giving the player a new level each time through. This is an effective strategy for creating engaging game experiences, but online level generation, in which the player's behavior alters the level as they play, is a much more dynamic approach to the core challenge to which Polymorph responds: difficulty adjustment.

The game Charbitat is an example of online, real-time level generation, where the player's preference for interacting with certain elements will alter the game world to increase the prevalence of that element (Nitsche et al., 2006). This focus on the world's elements differs substantially from Polymorph's focus on difficulty.

Data Collection

In order to generate parts of a level to match a player's skill level, Polymorph uses both a model of difficulty in our domain of 2D platformer levels and a dynamic model of the player's current performance. To answer these two questions—what makes a 2D platformer level difficult or easy, and how do we determine if a player is struggling or needs more of a challenge—we turn to a strategy of mass data collection and statistical machine learning. We created a data collection tool that asks a human player to play a short (approximately 10 seconds) level segment, collecting data on the level and the player's behavior along the way. The collected data and its use as machine learning features are discussed in more depth later. After the player completes the level or their character dies, they are asked to label the level segment by answering the multiple choice question: how difficult was this level segment? The label choices presented to the player are 1-Easy through 6-Hard. Only data from players completing multiple levels is considered, and the first attempt by each player is ignored since it is assumed that the player is still learning the mechanics.

The level segments are generated by an adaptation of the action rhythm-based generator from Smith et al. (2009), described in brief previously. The generator was modified to create the shorter segments rather than full levels. We propose that difficulty in interesting 2D platformer levels comes largely from the combination of adjacent components and not just from the presence or absence of a particular component or from the width and frequency of gaps—which has been the focus of past related work. This claim has been supported by guides for level design (Nicollet, 2004) and it is the reason we limit the level segments to such a short length. With short level segments, which don't contain too many level components, we can control which independent variables (in this case level component interactions) might be resulting in the difficulty label the player assigned to the segment. It would be impossible to measure the difficulty contributed by a single pair of components (such as a gap followed by an enemy) by using a large-scale, full-level difficulty label in which the challenge is not pinpointed to a particular segment of the level. We recognize that not all aspects of level challenge are captured by these short segments, but we are focusing on the micro level of component combinations rather than level-wide patterns or the introduction of new mechanics.

Another reason for using these short level segments is that they seem an ideal granularity for custom, player performance-based, generated level chunks. As the player progresses through a Polymorph level, each time they successfully pass through a segment of this length (or die), another segment of the same length is generated and placed in front of them, extending the level.

One potential drawback to this method of data collection is that players have differing levels of mastery over the game mechanics and an inexperienced player might therefore spend the first level segment learning to play. To compensate for this initial player disorientation, we discard each player's first level from the data analysis. After learning the game mechanics and controls in the first level, the differing skill of the players allows Polymorph to model difficulty for an average player based on the wide range of data.

The data collection tool, along with the Polymorph game proper, is Flash-based so that it can be easily distributed and used through most web-browsers by many simultaneous players. Using this tool we collected data from 211 unique players completing 2258 level segment playthroughs. Data collection is ongoing as of the writing of this paper.

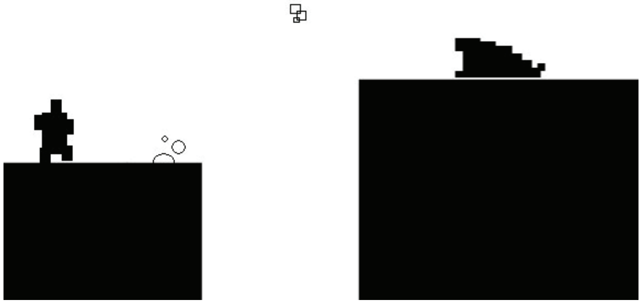

Figure 2 – Part of a level segment in Polymorph's data collection tool with the player character on the left. The segment includes a jump up over a gap, a coin and an enemy that moves left and right along the platform.

Learning Features

The first statistical model that Polymorph has learned from the collected data is a ranking of level segments according to their difficulty. As mentioned previously, we claim that difficulty in 2D platformer levels is related to the combinations of adjacent level components more than to the presence of a particular level component. Using the example shown in Figure 2, a gap by itself is easy to overcome and a slow plodding enemy is not much of a difficulty, but by placing the enemy on the landing platform of the gap, the level designer has created a much larger challenge for the player, requiring more exact timing and prediction of the enemy's movements. Therefore we have included as learning features not only the number of occurrences of a particular level component, but also the occurrences of any two-component adjacency in the level segment. Using the example depicted in Figure 2 once again, the feature regarding the number of upward-rising gaps in the segment is incremented, as is the feature regarding the number of gap-enemy adjacencies. Other level segment-related features of interest include the average gap width, the total change in altitude of the platforms in the level and the width of the largest and smallest platforms.

Polymorph also collected data for a model of the player's current performance. The data collection tool keeps track of features representing the player's behavior while playing. Some of the more interesting features are the amount of time the player spends standing still or moving backwards, the total completion time of the level segment, the number of coins collected, and whether the player died or completed the segment. The data collection tool does not ask the player how well they think they were performing, but we assume that this is implicit in their answer to how difficult they think the level segment was.

Given the level-descriptive features, we have applied a Multilayer Perceptron algorithm to rank all of the generated level segments as a model of difficulty, a process described in more detail in the following section. During play, Polymorph evaluates player behavior features on the model trained from the collected data. Then, before the player progresses into the next segment of the level a new segment is chosen that is ranked either higher or lower as necessary for the player's behavior. This way, as the player learns to play the game better and improves their personal skill, the level increases in difficulty to compensate and maintains an appropriate challenge. Alternatively, if it becomes clear that the player is struggling, the next segment of the level is chosen to reduce challenge.

Results

The data instances received from the level segments with the data collection tool are labeled by the individual player according to how difficult they deem the segment to be. Coming from over 200 different players, this difficulty rating is of course subjective. Not all players will use the same range of ratings or the same average rating, even given the same levels. A conservative player might only rate a particularly hard level as four out of six, while a more liberal player might rate a difficult level as six out of six. We used Gaussian normalization (Jin and Si 2004) to convert the player-specific ratings to a usable, comparable distribution.

The collected data was used to train and test a Multilayer Perceptron algorithm with 10-fold cross validation. Many classifiers were tested, but the best performance was achieved by the Multilayer Perceptron, a feedforward neural network that builds a model of the training data for instance evaluation. The model is intended to be able to order level segments by difficulty, so it was tested by taking all pairs of instances (level segments) in the test set, classifying them and ordering the pair by difficulty. The model ordered 66.4% of the instance pairs correctly (the same as the player ratings). Pedersen et al. (2009) were able to order pairs of levels by "challenge" more accurately (77%) in their work on Infinite Mario Bros. by focusing their model on features related to gap width and frequency. However, the levels generated by Polymorph are classified by a model of how different combinations of components affect the level's difficulty, which allows us to examine these combinations as aspects of difficulty in themselves.

Feature	Correlation coefficient
Gap_Kill	0.7749
JumpUp	0.7095
Gap_Gap	0.6765
Gap_Avoid	0.6177
FlatGap	–0.5316
KillEnemy	–0.5222
Avoid	0.3818
JumpDownGap	0.3219
JumpUpGap	–0.0842
Avoid_Thwomp	0.0709

Table 1 – Ten most highly correlated features. Underscores indicate adjacent components.

We can see from Table 1 that several of the component combination features are significantly correlated with the difficulty of the generated level segments. Gap_Kill, the most highly correlated feature, is a gap followed by a moving enemy on the landing platform, such as the level segment shown in Figure 2. Similarly, Gap_Gap is the feature for two adjacent gaps with a short platform in between and Gap_Avoid is a gap followed by spikes on the landing platform. This seems to indicate the intuitive level design notion that a good way of increasing the difficulty is by placing another level component immediately after a gap, requiring better jumping precision and timing from the player. On the other hand, the negative entries in Table 1 show that a lone gap or killable enemy is easy to overcome on its own. One surprising result from Table 1 is the correlation of a JumpUp component with difficulty. Certainly we would expect a JumpUp component to be more difficult than a JumpDown component, but we also suspect that this is in part an artifact of Polymorph's game mechanics, which allow the player character to jump farther the longer the jump button is pressed. This is common to a lesser extent in many classic 2D platformers, but it still may be a source of difficulty for some players when attempting a long upward jump.

The player behavior related features were less strongly correlated with the difficulty of the level segment. Only the player character's death had a very strong (positive) correlation with difficulty, though the number of coins collected also had a weak (negative) correlation. Given this revelation, until a further user study is performed to determine more player behavior features that are strongly correlated with level difficulty, Polymorph is using a somewhat simplified model of player performance. Possibly a larger game with more player-actions would be necessary for a better player performance model. When the player character dies it is taken as an indication that the player is struggling, more strongly if they die twice within a short time period. The number of coins collected is also taken as a weak indicator of performance: the higher percentage of available coins that the player collects, the better they are doing.

It is a well known game design principle that players should face increasing difficulty as they advance through the game, gradually honing their mastery of the game mechanics (Juul, 2009). Falstein (2005) has added that game difficulty should increase irregularly and that some amount of unpredictable challenge enhances player enjoyment. As such, Polymorph attempts to gradually increase the difficulty of the level design as the player progresses further into the level, varying the amount of increase for an irregular difficulty curve. Polymorph only halts the difficulty increase or starts a difficulty decrease when the player's performance has drastically declined, indicated by the death of the player character.

When Polymorph determines that the difficulty of the next level segment should be increased or decreased, the level generator creates a list of level segments, which are evaluated on the Multilayer Perceptron model. These segments are then ordered by difficulty in relation to the current 'in play' segment. Then a segment with the appropriate difficulty relation to the 'in play' segment, easier if the player is dying or harder according to the gradual difficulty curve, is selected and added onto the end of the current level. Thus Polymorph's levels adjust themselves dynamically to fit an individualized difficulty level for the player.

In addition to personalized difficulty, Polymorph's dynamic level generation also achieves the traditional procedural content generation goal of replayability. Two Polymorph players performing identically will still have a different experience as the level unfolds differently before each of them.

Example Segments

Figure 3 shows two of the level segments that were created by Polymorph's level generator, played by a human player with the web-based data collection tool and rated according to difficulty. The first segment was given a 1 (easy) player rating, while the second was rated 6 (hard). The first segment includes the JumpUp and Gap_Kill features, and the second includes the JumpUp, Gap_Avoid and Avoid_Thwomp features. In this case, the addition of the thwomp component on the final platform seems to have substantially added to the challenge of the player's experience. This is the kind of level design knowledge that Polymorph encodes in its learned model of component difficulty.

Conclusion & Future Work

This paper has described the implementation of the 2D platformer Polymorph. The game's goal is to create a unique player experience in which Dynamic Difficulty Adjustment is performed on the structural design of the game levels, giving players the appropriate level of challenge while trying to minimize difficulty-related frustration or boredom. This is achieved through a model of level difficulty learned from a web-based data collection user study, wherein over 200 players rated generated level segments according to difficulty. Our results show correlation between many of the features and the player rating of difficulty for the level segment. The features of this level-difficulty model are unique because they take into account the combinations of level components confronting the player. This model is used dynamically in Polymorph to maintain a gradual difficulty curve until the player shows signs of struggling, when the difficulty of the level is reduced to compensate.

This paper's contributions are the description of a new, machine learning-based strategy for dynamic level generation and the creation of a unique gameplay experience in Polymorph, which gives the player individualized game level structures and challenges.

Areas of future work may include user studies to

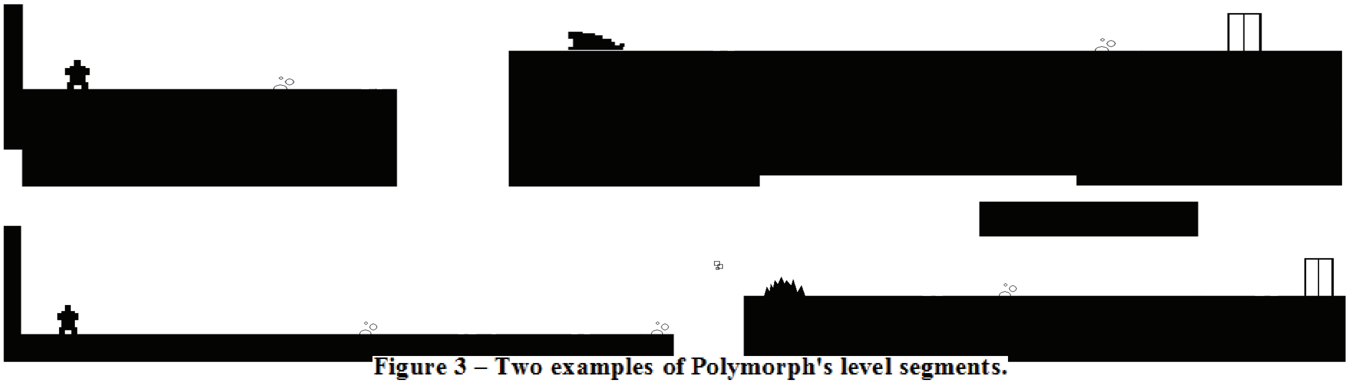

Figure 3 – Two examples of Polymorph's level segments.

validate the effectiveness of the game at achieving its stated goal of personalized Dynamic Difficulty Adjustment. One difficulty with this is the possibility that DDA is 'invisible' to the player when it is doing its job correctly. We would also like to develop a stronger model of player performance to determine if a player is struggling before they die, though it is not desirable to prevent the player from ever losing (Juul, 2009).

The same techniques applied in this paper could be used for other game designs and genres, providing that the appropriate tools were available: data collection for how difficulty is impacted by combinations of level components —including structural differences—and a level generator to create short segments that can be strung-together in real time into playable levels. Some existing commercial middleware might assist in collecting player behavior data in different game genres.

Game Availability

Polymorph can be played online at http://users.soe.ucsc.edu/~mjennin1/polymorph/polymorph.html. The data collection device is still online as of this writing at http://users.soe.ucsc.edu/~mjennin1/segments/PlayTestLevelSegment.html.

References

Blizzard Entertainment 1997. Diablo.

Booth, M. 2009. The AI Systems of Left 4 Dead. Keynote, Fifth Artificial Intelligence and Interactive Digital Entertainment Conference (2009).

Chen, J. 2007. Flow in games (and everything else). Commun. ACM 50, 4 (Apr. 2007), 31-34.

Csikszentmihalyi, M. Flow: The Psychology of Optimal Experience. Harper Perennial, London, 1990.

Falstein, N. 2005. "Understanding Fun—The Theory of Natural Funativity". Introduction to Game Development, ed. Steve Rabin. Charles River Media, Boston, 71-98.

Fullerton, T., Swain, C., and Hoffman, S. 2004. Improving player choices. Gamasutra (March 2004). http://www.gamasutra.com/features/20040310/fullerton_01.shtml. Online Feb. 1, 2005.

Hunicke, R. 2005. The case for dynamic difficulty adjustment in games. In Proceedings of the 2005 ACM SIGCHI international Conference on Advances in Computer Entertainment Technology (2005). ACE '05, vol. 265. ACM, New York, NY, 429-433.

Jin, R. and Si, L. 2004. A Study of Methods for Normalizing User Ratings in Collaborative Filtering. The 27th Annual International ACM SIGIR Conference (SIGIR 2004), pp. 568-569.

Juul, J. 2009. Fear of Failing? The Many Meanings of Difficulty in Video Games. The Video Game Theory Reader 2, B. Perron and M. Wolf, Ed. Routledge, London.

Kazemi, D. 2008. Metrics and Dynamic Difficulty in Ritual's SiN Episodes. OrbusGameWorks.com. http://orbusgameworks.com/blog/article/70/metrics-and-dynamic-difficulty-in-rituals-sin-episodes-part-1

Nicollet, Victor. 2004. Difficulty in Dexterity-Based Platform Games. GameDev.net (March 2004). http://www.gamedev.net/reference/articles/article2055.asp

Nintendo 1985. Super Mario Bros.

Nitsche, M., Ashmore, C., Hankinson, W., Fitzpatrick, R., Kelly, J., and Margenau, K. 2006. Designing Procedural Game Spaces: A Case Study. In Proceedings of FuturePlay (2006).

Olesen, J., Yannakakis, G., and Hallam, J. 2008. Real-time challenge balance in an RTS game using rtNEAT. Proceedings of the 2008 IEEE Symposium on Computational Intelligence and Games (2008).

Pedersen, C., Togelius, J., and Yannakakis, G. 2009. Modeling Player Experience in Super Mario Bros. Proceedings of the 2009 IEEE Symposium on Computational Intelligence and Games (2009).

Persson, M. Infinite Mario Bros.

Phillips, B. 2009. Staying Power: Rethinking Feedback to Keep Players in the Game. Gamasutra.com. http://www.gamasutra.com/view/feature/4171/staying_power_rethinking_feedback_.php

Ritual Entertainment 1998. SiN Episodes.

Smith, G., Treanor, M., Whitehead, J., Mateas, M. 2009. Rhythm-Based Level Generation for 2D Platformers. Proceedings of the 2009 Int'l Conference on the Foundations of Digital Games (2009).

Togelius, J., De Nardi, R., and Lucas, S. 2007. Towards automatic personalised content creation for racing games. Proceedings of the 2007 IEEE Symposium on Computational Intelligence and Games (2007).

Togelius, J., Yannakakis, G., Stanley, K., and Browne, C. 2010. Search-based Procedural Content Generation. To be presented at Evostar (2010).

Toy, M. and Wichman, G. 1980. Rogue.

Valve Software 2008. Left 4 Dead.

Yu, D. 2009. Spelunky.