Polymorph: Dynamic Difficulty Adjustment Through Level Generation

Martin Jennings-Teats, Gillian Smith, Noah Wardrip-Fruin
Expressive Intelligence Studio
University of California, Santa Cruz
Santa Cruz, CA, USA
{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 present Polymorph, which employs techniques from level generation and machine learning to understand game component difficulty and player skill, dynamically constructing a 2D platformer game with continually-appropriate challenge. We believe this will create a play experience that is unique because the changes are both personalized and structural, while also providing an example of a promising new research and development approach.

Categories and Subject Descriptors
K.8.0 [**Personal Computing**]: General – *Games.* I.2.6 [**Artificial Intelligence**]: Learning.

General Terms
Design, Human Factors.

Keywords
Games, level design, dynamic difficulty adjustment, procedural content generation.

1. 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 [8]. 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).

This paper describes the vision and implementation of Polymorph. The goal of Polymorph is to automatically generate 2D platformer levels during play as a means of dynamic difficulty adjustment. Specifically, rather than being authored by hand, game levels will be procedurally generated as the player moves through the level, one chunk at a time as needed. The generation of these chunks will be customized to match the player's performance, so that each player will be 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 the game will correct for this in the next section, hopefully avoiding difficulty-related player frustration and boredom and providing an example of a promising new approach to DDA.

We tackle the DDA problem by creating a statistical model of difficulty in 2D platformer levels along with a model of the player's current skill level. These models are gleaned with machine learning techniques from play traces collected with a game-like tool. The models are used 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 [14], which is described in more detail in section 2.2.

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 is a work in progress: a data collection tool, the level generator, the game engine, and a pilot study have been completed.

2. RELATED WORK

2.1 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 [3] [6]. 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

PCGames 2010, June 18, Monterey, CA, USA

every player's skill level, and frustrating mismatches between a player's skill and a game's difficulty are common [5] [12].

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 [5]. 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. It uses a statistical model of player performance with "advisor" 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 [7] [13]. Hunicke 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 [5]. What these DDA strategies 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 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 [2] [18]. Pedersen et al. created a version of Infinite Super Mario Bros from which they derived a statistical model of player challenge and frustration, among other emotional states [11]. 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 [10]. 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.

2.2 Procedural Level Generation

Procedural level generation has been used in games for decades, with popular RPGs such as Rogue and Diablo, as well as some 2D platformers such as Spelunky [1] [17] [19]. These games typically work by fitting together hand-authored level chunks into random combinations. Pedersen et al.'s 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 [10]. 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 [15]. These approaches differ from Polymorph by generating geometry in larger granularity, as well as in Polymorph's unique learning features (see section 4).

Smith et al. 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. This approach starts with a rhythm

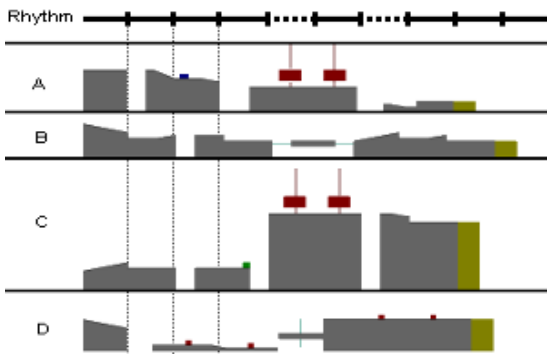

Figure 1. Several possible mappings of geometry onto rhythm.

of desired player actions, such as run, jump or wait. 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 [14]. 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 [16]. 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 [9]. This focus on the world's elements differs substantially from Polymorph's focus on difficulty.

3. DATA COLLECTION MECHANISM

3.1 Tool

In order to generate parts of a level to match a player's skill level, we need 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 in section 4. 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 in order to avoid subjective difficulty ratings.

The level segments are generated by an adaptation of the action rhythm-based generator from [14], described briefly in section 2.2. We do not put any restrictions on the rhythms used to generate level segments, since we want to consider all playable segments. We believe 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 (see section 4). This belief 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 attempt to control which independent variables, in this case level component interactions, might be resulting in the difficulty label the player assigned to the segment. 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.

One potential drawback to this method of data collection is that a player might not have a good understanding of the game mechanics their first time through a short segment and might therefore rate it as more difficult than they would after they gained more experience. However, we believe that this is representative of playing a full game, where the player learns the game and increases in skill as they progress, so that a player participating in our data collection over many level segments will help us to model difficulty for an average player. Also, this short level segment seems ideal as the amount of granularity for custom, player performance-based, generated level chunks as the player progresses through a level of the final game. Each time the player successfully passes through a segment of this length (or dies), another segment of the same length will be generated and placed in front of them.

The tool is Flash-based so that it can be easily distributed and used through most web-browsers by many simultaneous players. We have completed a pilot study with more than one hundred undergraduate game design students from UC Santa Cruz, allowing us to refine our instruments and the machine learning features discussed in section 4. The preliminary data and resulting refinements have been encouraging for the goal of the data collection: to model player performance and difficulty in 2D platformer levels. We are currently preparing to distribute the tool much more widely to collect data from thousands of participants over multiple level segment playthroughs.

3.2 Generate and Test

One of the early challenges in the development of Polymorph was the tendency of the level generation algorithm to create many more easy level segments than difficult level segments, though the generator is capable of expressing levels across a wide range of difficulty. This was due to the many possible rhythm-geometry mappings that do not present a challenge for the player—a flat surface with several short gaps, for instance. This was problematic for the data collection mechanism, since the players assigned labels for low difficulty levels far more often than for high difficulty levels. The distribution of the data was therefore skewed toward the low-difficulty end of the spectrum.

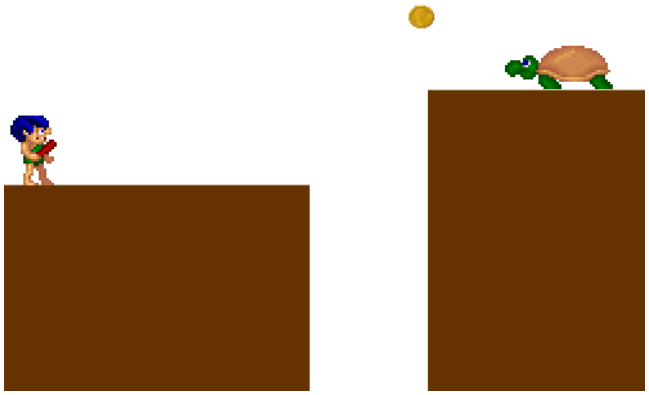

Figure 2. Part of a level segment in Polymorph's data collection tool with the player character on the left. The segment includes a jump up over a gap, a coin and a moving enemy.

The distribution of level segments has been corrected by creating several heuristic critic modules. When a level segment is first generated, each of these critics will estimate its difficulty on a particular metric. For example, one critic examines the level segment's action-density, while another simply counts the number of potentially deadly level components present, since some components do not create the possibility of player death on their own. Once all of the critics have examined the level segment, it is classified into several estimated difficulty categories, which the data collection tool samples for segments presented to the player rather than choosing a random unplayed level segment, thus modifying the distribution of segments so that it will be more spread across the range of level difficulty. The critics only decide which segments to show to players and are not considered for the final ratings of challenge.

4. LEARNING FEATURES

The first statistical model that we want Polymorph to learn from the collected data is a ranking of level segments according to their difficulty. As mentioned previously, we believe 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 needs a statistical model of the player's current skill level. 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 will apply a machine-learned ranking algorithm such as Ranking SVM to rank all of the level segments generated during play of the final game [4]. Meanwhile, Polymorph will be collecting the player behavior features, which will be evaluated on a model trained with the data from the collection tool. Then, before the player progresses into the next segment of the level a new segment from the ranked list will be chosen according to the model of the player's current skill level. This way, as the player learns to play the game better and improves their personal skill, the level will increase in difficulty to compensate and maintain an appropriate challenge. Alternatively, if it becomes clear that the player is struggling, the next segment of the level will be chosen to reduce challenge.

5. CONCLUSION & FUTURE WORK

We have described the vision, completed work, and further plans for dynamic difficulty adjustment in the game Polymorph. The player-specific adjustment is achieved by procedurally generating the level during play. At this time we have created the game engine, the level generator, and the data collection tool, as well as run a pilot study. To complete the game we will collect data on a much larger scale with the online tool, and we will process the data as discussed in section 4 to create models of level difficulty and player performance. Final game polish will be applied, with commissioned artwork and tweaking of the mechanics based on evaluative playtests.

The largest challenge for the development of Polymorph has been the task of designing features to collect from the generation of level segments. The features need to be broad enough to represent the difficulty of a level for an average player while remaining specific enough to be generalizable to other, very different level designs. We are confident in the features we have chosen, but we do not claim to have created a perfect model of difficulty in 2D platformer levels. Considering the pair-wise adjacencies of level components will help to address the problem of difficulty arising from the interaction of level components rather than from the presence of individual components. However, in the analysis of challenging hand-authored platformer levels it seems that the interaction of more than two components is common. Creating features to represent these more complicated interaction settings would be an improvement and is a direction we would like to pursue with future iterations of Polymorph.

Procedurally generating level segments online, in real-time as a method of dynamic difficulty adjustment allows for intervention that is both a structural change and personalized to the player's skill and experience. We believe this will give Polymorph a unique play experience and demonstrate the strength of combining techniques from level generation and machine learning for dynamic difficulty adjustment.

6. REFERENCES

[1] Blizzard Entertainment 1997. Diablo.

[2] Booth, M. 2009. The AI Systems of Left 4 Dead. Keynote, Fifth Artificial Intelligence and Interactive Digital Entertainment Conference (AIIDE '09). Stanford, CA. October 14 – 16, 2009.

[3] 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.

[4] Herbrich, R., Graepel, T., and Obermayer, K. 2000. Large Margin Rank Boundaries for Ordinal Regression. Advances in Large Margin Classifiers, 115-132, Liu Press.

[5] 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 (Valencia, Spain, June 15 - 17, 2005). ACE '05, vol. 265. ACM, New York, NY, 429-433.

[6] 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.

[7] 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

[8] Nintendo 1985. Super Mario Bros.

[9] 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. London, Ontario. October 10 – 12, 2006.

[10] 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 (Politecnico di Milano, Milano, Italy, September 07-10, 2009).

[11] Persson, M. Infinite Mario Bros.

[12] 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

[13] Ritual Entertainment 1998. SiN Episodes.

[14] 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 (Orlando, FL, USA, April 26-30, 2009).

[15] 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).

[16] Togelius, J., Yannakakis, G., Stanley, K., and Browne, C. 2010. Search-based Procedural Content Generation. To be presented at Evostar (Istanbul Technical University, Istanbul, Turkey, April 07-09, 2010).

[17] Toy, M. and Wichman, G. 1980. Rogue.

[18] Valve Software. 2008. Left 4 Dead.

[19] Yu, D. 2009. Spelunky.