AN EXPERIMENTAL TESTBED TO ENABLE AUTO-DYNAMIC DIFFICULTY IN MODERN VIDEO GAMES

Christine Bailey and Michael Katchabaw
Department of Computer Science
The University of Western Ontario
London, Ontario, Canada
E-mail: cdbailey@csd.uwo.ca, katchab@csd.uwo.ca

KEYWORDS

Auto-dynamic difficulty, difficulty adjustment in games

ABSTRACT

Providing gameplay that is satisfying to a broad player audience is an appealing goal to game developers. Considering the wide range of player skill, emotional motivators, and tolerance for frustration, it is simply impossible for developers to deliver a game with an appropriate level of challenge and difficulty to satisfy all players using conventional techniques. Auto-dynamic difficulty, however, is a technique for adjusting gameplay to better suit player needs and expectations that holds promise to overcome this problem.

This paper presents an experimental testbed to enable auto-dynamic difficulty adjustment in games. Not only does this testbed environment provide facilities for conducting user studies to investigate the factors involved in auto-dynamic difficulty, but this testbed also provides support for developers to build new algorithms and technologies that use auto-dynamic difficulty adjustment to improve gameplay. Initial experiences in using this auto-dynamic difficulty testbed have been quite promising, and have demonstrated its suitability for the task at hand.

INTRODUCTION

The goal of producing a game is to provide many things to players, including entertainment, challenge, and an experience of altered state. Ultimately, however, a game must be fun. One major source of polarization among players on this issue is the level of difficulty in a game. There is a great degree of variation in players with respect to skill levels, reflex speeds, hand-eye coordination, tolerance for frustration, and motivations.

In (Csikszentmihalyi 1996), the concept of "flow" is used to refer to an individual's "optimal experience". In a state of flow, the individual experiences intrinsic enjoyment from undertaking a task that feels almost effortless and natural, while also causing the individual to feel focussed and challenged. One facet of flow is that there is a balance between the challenge presented by the task and the increasing skill of the individual, as discussed in (Falstein 2004). This closely resembles the concept of a zone of proximal development as discussed in (Woolfolk et al. 2003), in which a balance between skill and challenge is needed in educational settings in order for learning to take place. This zone of proximal development was found to be different for each student, and that tasks considered easy by some may be too difficult for others. Assigning difficulty levels in games must address this problem so as to hit the "optimal experience" for as many players as possible, each of which may have very different zones of proximal development. These issues are important because, as noted in (Miller 2004) and (Rouse 2004), a game must balance challenging and frustrating the player to provide the best overall level of satisfaction and enjoyment.

Several approaches have been used in the past to attempt to provide appropriate difficulty levels in a variety of different ways, such as having a single static difficulty level (chosen either by the designer or through play-testing of the target audience), having several different static difficulty levels to choose from at the start of the game, or providing cheat-codes. However, each method has its drawbacks and limitations, and ultimately cannot provide an appropriate difficulty level to all players, particularly as their skills improve as they play and learn. In the end, this can drastically limit the success of a game.

Auto-dynamic difficulty refers to the ability of a game to automatically adapt the difficulty level of gameplay to match the skills and tolerances of a player. If done properly, this can provide a satisfying experience to a wider variety of players. The concept of auto-dynamic difficulty is not new; it has been used in early arcade games such as *Xevious* to more recent titles such as *Max Payne* (Miller 2004). Typically, however, this technique is used in an ad hoc and unrepeatable fashion, applied to a particular game or gameplay element within a game. Often, there is little regard or understanding for the various factors that influence player experience and how these factors interact with one another; as long as the current game is improved, that is all that matters.

In this paper, we discuss the development of an experimental testbed to facilitate the development of auto-dynamic difficulty enabling technologies for games. This testbed serves two key purposes. The first is to support experimentation to better understand what shapes player experience and how gameplay and difficulty can be altered to produce the best experience possible. The second is to

serve as a vehicle for testing new algorithms and methodologies for supporting auto-dynamic difficulty developed as part of this work. The goal is that this work will provide both a better understanding of how to create more enjoyable and satisfying gameplay experiences for a wider range of players, and that it will deliver enabling technologies to make use of this new understanding in a wide variety of games and gameplay scenarios.

The remainder of this paper is organized as follows. We begin in the next section with a background discussion of auto-dynamic difficulty adjustment, describing what adjustments are possible, and the deciding factors in determining when and how such adjustments should be made. We then present the architecture and implementation of our auto-dynamic difficulty experimental testbed environment. We then provide a brief discussion of our experiences to date in using this experimental testbed. Finally, we conclude this paper with a summary and discussion of potential future work in this area.

AUTO-DYNAMIC DIFFICULTY ADJUSTMENT

Before discussing our experimental testbed environment, it is first important to further explore the key issues behind auto-dynamic difficulty adjustment. Of critical importance is to recognize what adjustments can and should be made, as well as when and how to make these adjustments. Any adjustments must be made with care in such a way that they enhance the satisfaction and enjoyability of the game, without disrupting the game in a negative fashion. (For example, changes that are too abrupt could disrupt the immersion of the player, causing a negative effect on the overall experience.)

What to Adjust

Designed properly, a good portion of a game's gameplay elements can have difficulty that is adjustable dynamically (Bailey 2005). This includes the following:

Player character attributes. The attributes of the player's character can be tuned according to the desired level of difficulty in a game. As examples, to make a game easier, the player could be made stronger, move faster, jump higher and farther, have more health, have better armour, attack with more damage, attack more frequently, and so on. To make a game harder, these attributed can be adjusted in the opposite directions.

Non-player character attributes. Likewise, the attributes of non-player characters controlled by the game's artificial intelligence can change. Not only does this include the attributes affecting the actions they take as above, but this also includes the decision making processes used. To make a game less difficult, non-player characters can make poorer decisions, provided that these decisions do not make the characters appear artificially stupid. As examples, path-finding can be adjusted to make the player harder to find, aiming can be adjusted so that attacks are less successful, and so on. Similarly, steps can be taken to make better decisions that make the game more difficult.

Game world and level attributes. Various elements of how the game world and its levels are designed can affect game difficulty, including both the structure of the levels, and their contents (Bates 2004). With advancements in game engine technologies, it is now possible to do this dynamically from within the game. Adjusting level structure depends heavily on the gameplay occurring within the gameplay. For example, in a platformer-genre game involving a lot of jumping puzzles, level geometry can be adjusted dynamically to make gaps smaller or larger to make the game easier or more difficult. In a shooter game, as another example, the amount of cover can be adjusted appropriately to make the game easier or more difficult as well. Level contents can also be tuned dynamically to adjust difficulty. By adding or removing items such as ammunition, health upgrades, and so on, a game can be made easier or more difficult. Varying the quantity and spawning locations of enemy non-player characters can also affect difficulty.

Puzzle and obstacle attributes. As discussed in (Bates 2004), there are several ways of adjusting the level of difficulty provided by puzzles and obstacles within a game. Fortunately, many of these techniques can be applied dynamically. While it might not always be possible to dynamically adjust the attributes of the current puzzle or obstacle faced by the player (for consistency and other reasons), it might be possible to instead adjust the difficulty in puzzles faced in the future. For example, if a player is finding one type of puzzle to be difficult to solve, in the future, the solution to that same type of puzzle can be placed closer to the puzzle itself, making it inherently easier to solve (Bates 2004).

As discussed in (Miller 2004), most earlier attempts at auto-dynamic difficulty focussed on a restricted subset of gameplay, typically in the adjustment of player or non-player character attributes. With this rationale applied throughout the game, as discussed above, it is possible to create a better overall player experience.

When and How to Adjust

To determine when to adjust game difficulty and how to do so, data must be collected on players and their progression through the game. To provide the best level of challenge, we must have a measure of the current skill level of the player, as well as their success and failure rates at the various elements of gameplay encountered to date in the game. Since different players will tolerate and accept different levels of challenge at different times, we must also have a sense of the player's general type, motivations, frustration tolerance, and emotional state.

Measuring a player's level of skill in a game, as well as their success and failure rates, is inherently tied to the particular game or game genre. Typically, however, there are multiple metrics that are applicable and can be measured from within the game itself. For example, in a platformer game with a sequence of jumping puzzles, the number of attempts before success and time to completion could be useful metrics. In a shooter game, the percentage

of enemies eliminated per level, the amount of damage taken per level, and time to completion could be useful metrics. One must give careful thought to the metrics selected, however, as they could indicate unanticipated styles of play or other player activity, and not the skill of the player. For example, tracking the number of game saves and loads might be problematic. One might think that a high frequency of saves and loads is indicative of an unskilled player, but this pattern of activity could also be encountered by a player playing the game during short coffee breaks (Bailey 2005). Counting the number of player character deaths might also be misleading, as an unskilled player could get frustrated after a single death and quit the game with a relatively low death count only to return later. So, while there might be multiple methods of tracking player progression through a game, care and thought must be put into the process.

Determining a player's type and internal factors is more difficult to do within a game, but not impossible. For example, (Sykes and Brown 2003) found that the pressure of button and key presses correlated strongly to frustration and difficulty levels within a game. The work in (IP and Adams 2002) examined ways of quantitatively measuring levels of "core" and "casual" in a given player. As discussed in (Bailey 2005), elements of player types identified in (Bartle 1996) and (Lazzaro 2004) could be identified by tracking player movement and progress through a game. For example, the explorer type identified in (Bartle 1996) could be detected by observing players lingering in areas of the game world for extended periods of time without paying attention to game goals, while the achiever type could be detected by observing a linear and timely progression through game goals. As pointed out in (Bailey 2005), however, there are ultimately some internal factors that are not easily measurable from within a game world, and we must rely upon external studies and experimentation to calibrate the game and assist in correlating observed player behaviour and emotion state.

Using measurements of player skill, as well as success and failure rates, it is not hard to determine when a player is encountering difficulty with a certain element of gameplay. While these measurements are important, we must be careful to also take into consideration player type and internal factors; otherwise, we again fall into a "one-size-fits-all" mentality that does not produce appealing results to a broad audience. It is also important to consider the impact of characteristics of the gameplay on the motivation of the player, including whether the necessity of the gameplay element, the rewards for success, the consequences of failure, and so on (Bailey 2005).

In the end, it is possible to develop rudimentary rules to guide when difficulty adjustments should be made and how, based on this information. For example, if a player is encountering a challenging task, but is exhibiting characteristics of the achiever type, then difficulty should not be adjusted as this player type is more likely to enjoy the challenge than not (Bartle 1996). However, to assist in the formulation and validation of these rules and decision models, experimentation is necessary. A thorough investigation in this area is clearly warranted. This reality, in part, motivated developing the experimental testbed discussed in this paper.

AUTO-DYNAMIC DIFFICULTY EXPERIMENTAL TESTBED

To facilitate the study of auto-dynamic difficulty, our current work focuses on the construction of an experimental testbed that will enable experimentation with players and development of new technologies to better tune game difficulty automatically to meet their needs. This testbed is depicted in Figure 1, and discussed in more detail in the sections below.

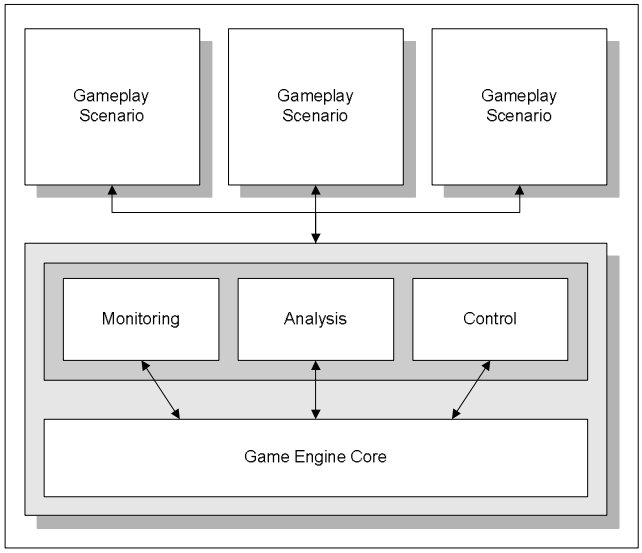

Figure 1: Auto-Dynamic Difficulty Experimental Testbed Architecture

Game Engine Core

The game engine core is used to provide all of the fundamental technologies required to drive a game or gameplay scenario. This includes graphics, audio, animation, artificial intelligence, networking, physics, and so on. One could then layer new gameplay logic and content on top of this engine to have a complete game, without the burden of developing all of the underlying technologies. This saves considerable development time in building the testbed, and also allows the use of professional-grade tools to produce a gameplay experience of very high quality.

At the core of our testbed is Epic's Unreal engine (Epic Games 2004). The Unreal engine is a modern, state-of-the-art game engine that can be used to support a wide variety of game genres and gameplay elements. It also supports rendering in both first and third person views, which makes it easier to support more varied gameplay. The Unreal engine itself is written in C and C++, but provides a flexible object-oriented scripting language, UnrealScript, to make it easy to extend the engine and deliver new functionality. Since this engine is based on leading edge technologies and still in use commercially today, there are no concerns of confounding that could have arisen from using older,

obsolete technologies. (For example, in such a case, one would have to determine if a player had an unsatisfactory experience because of game difficulty or because the game's graphics were not up to the standards set by modern games.) In the end, the Unreal engine was a natural choice of foundation on which to build our testbed.

Monitoring, Analysis, and Control

Monitoring, analysis, and control services are used in the testbed to support both auto-dynamic difficulty experimentation and software developed to implement new auto-dynamic difficulty algorithms and methodologies. These services are used by gameplay scenarios, and directly make use of the game engine core.

To conduct experimentation within a particular gameplay scenario, the experimental environment must monitor and collect the appropriate player and progression data, as discussed in the previous section. The analysis service is used to provide support in the aggregation and correlation of data collected through monitoring. The control service is used to manipulate the experiment in the gameplay scenario, including starting, suspending, resuming, and halting a particular experiment. It is important to note that some aspects of monitoring, analysis, and control may need to be completed offline outside of the testbed software. (For example, augmenting recorded data with audio and video recordings, as well as surveys and interviews currently must be done offline. In the future, it is hoped to add these elements to the testbed software as well for a more integrated solution. Analyses of these elements would still likely require manual intervention, however.)

To support new auto-dynamic difficulty algorithms and methodologies, the monitoring service still collects player and progression data as before. The analysis service in this case is now focussed more on analysing this data to formulate decisions on when and how to adjust game difficulty using rules and decision models formulated based on experience and experimentation conducted using the testbed. The control service in this case still manipulates the gameplay scenario, but is focussed this time on the relevant attributes of the player character, non-player characters, the game world, or game puzzles and obstacles to adjust the game's difficulty according to the decisions developed by the analysis service.

In our testbed, the monitoring, analysis, and control services are written in UnrealScript. All three services are integrated into a single new Unreal game type derived from the base Unreal game type class. This new game type provides instrumentation suitable for embedding in gameplay scenarios to enable monitoring, analysis, and control activities. This facilitates the development of new gameplay scenarios and entire games using these auto-dynamic difficulty services, as these new games would simply need to derive their own game type from this new type, instead of the base class.

At present, rudimentary monitoring, analysis, and control services are provided; more sophisticated facilities are currently under development. Currently, the monitoring service can collect time to completion, success and failure rates, and other metrics, the analysis service can support simple correlations and decision rules, and the control service can control experiment operation, and tune certain player and non-player character attributes, as well as selected game world attributes.

Gameplay Scenarios

Gameplay scenarios are used to contain playable elements of games and game content. These can range in scale from mini-games depicting as few as one game activity for the player, all the way up to complete entire games.

In the current version of the testbed, we have implemented a variety of mini-game gameplay scenarios using UnrealScript and UnrealEd (Busby et al. 2005). These include two jumping mini-games (one with fatal consequences, the other with no failure consequences), a timed maze navigation mini-game, a turret mini-game requiring the player to navigate a short hallway lined with automated, indestructible gun turrets, and a fighting mini-game requiring the player to make their way through a room full of heavily armed enemy non-player characters. A screenshot from one of these scenarios is given below in Figure 2. Recognizing the limitations of experimenting with mini-games, as discussed in the next section, we are also building the Neomancer project (Katchabaw 2005) based on our new game type, to provide a complete action/adventure/role-playing game experience for experimentation and development activities.

Figure 2: Screenshot from Turret Hallway Mini-Game

EXPERIENCES AND DISCUSSION

Initial user studies and testing using the auto-dynamic difficulty experimental testbed were conducted with a small number of family members and co-workers of researchers at Western. Results of this early experimentation have been rather positive, indicating that the testbed is suitable for the task at hand. While the initial version of the testbed can only monitor a small number of player and progression metrics, analyse through simple correlations and a restricted rule set, and control through only simple operations, we have been able to gather interesting results from experimentation and implement several auto-dynamic

difficulty algorithms. It is clear, however, that more thorough experimentation using a large study group is necessary, both to better understand the interplay of the factors involved in auto-dynamic difficulty, and to develop better algorithms and technologies for games (Bailey 2005).

During initial experimentation using the mini-game game scenarios, it also became apparent that mini-games on their own might not be sufficient for investigating auto-dynamic difficulty fully. Mini-games, by their very nature, do not have a broader story, context, or reward system, which was found to produce a different emotional state in the player than playing a full game. Repetition of mini-games was also found to grow tedious, resulting in a negative impression of the mini-game independent of its challenge or difficulty. Consequently, it is necessary to have a complete gaming experience to fully explore auto-dynamic difficulty. Fortunately, through our development efforts in the Neomancer project (Katchabaw 2005), we have access to a commercial scale action/adventure/role-playing game that will fill this need nicely.

Game performance is a crucial factor to game players and game developers alike. Consequently, it is critical to ensure that there be minimal overhead imposed by auto-dynamic difficulty on the game as it plays. During initial experimentation, frame rate tests were conducted using the Unreal engine's own frame rate monitors, with and without the use of auto-dynamic difficulty and the instrumentation required for monitoring and control. This testing found that there was no measurable difference between frame rates delivered with and without auto-dynamic difficulty in place, and so performance was deemed acceptable.

The approach to auto-dynamic difficulty currently taken in this work is reactive. In other words, once measurements indicate that a game is too easy or too difficult for the player, gameplay can be adjusted to produce a more favourable experience. Unfortunately, a reactive approach means that a player must encounter such problems before any corrective actions are taken, and that the player could lose patience with the game before auto-dynamic difficulty has a chance to become active. It was found during initial experimentation that some mini-game scenarios could be made so easy or so difficult that the player is turned off almost instantaneously, sharply reducing the benefits of reactive auto-dynamic difficulty in these extreme situations. Proactive auto-dynamic difficulty, on the other hand, attempts to adjust game difficulty before a player encounters the above problems through an analysis of non-critical gameplay tasks. Doing so, however, would likely require calibration through more user studies to develop an appropriate predictive model, and introduces other problems if predictions are inaccurate. It would seem, however, that investigating proactive adjustments, perhaps in conjunction with reactive techniques, would be a worthwhile endeavour.

CONCLUDING REMARKS

Delivering satisfying gameplay experiences to a variety of players is a challenging task. To do so, gameplay difficulty must be tuned to suit player needs, as in auto-dynamic difficulty. Our current work is aimed at addressing this, by providing an experimental environment for studying this problem and assisting in the formulation of acceptable solutions. Initial experience through using this auto-dynamic difficulty experimental testbed has been quite positive, showing much promise for the future.

In the future, there are many interesting avenues for continuing research to take. We plan to refine the monitoring, analysis, and control capabilities of the testbed, to enable more thorough user studies. Using this enhanced testbed, we intend to expand experimentation to include a larger, more diverse player population. Based on the results of this experimentation, we will develop additional rules and decisions models for use in the testbed's analysis service to better support a wider variety of auto-dynamic difficulty algorithms. At the same time, we will continue work on the Neomancer project to provide a full length, feature rich gameplay scenario for studies with the testbed. Finally, we plan to continue investigating other open research issues in auto-dynamic difficulty adjustment, including reactive versus proactive techniques.

REFERENCES

C. Bailey. "Auto-Dynamic Difficulty in Video Games". *Undergraduate Thesis. Department of Computer Science, The University of Western Ontario*. April 2005.

R. Bartle. "Hearts, Clubs, Diamonds, Spades: Players who Suit MUDs". *Journal of MUD Research, 1(1)*. 1996.

B. Bates. *Game Design*. Second Edition. Thomson Course Technology. 2004.

J. Busby, Z. Parrish, and J. Van Eenwyk. *Mastering Unreal Technology: The Art of Level Design*. Sams Publishing. 2005.

M. Csikszentmihalyi. *Creativity: Flow and the Psychology of Discovery and Invention*. New York, NY: HarperCollins Publishers. 1996.

Epic Games. *Unreal Engine 2, Patch-level 3339*. Nov. 2004.

N. Falstein. "The Flow Channel". *Appeared in Game Developer Magazine*. May 2004.

B. Ip and E. Adams. "From Casual to Core: A Statistical Mechanism for Studying Gamer Dedication". *Article published in Gamasutra and is available online at http://www.gamasutra.com/features/20020605/ip_pfv.htm.* June 2002.

M. Katchabaw, D. Elliott, and S. Danton. "Neomancer: An Exercise in Interdisciplinary Academic Game Development". *In the Proceedings of the DiGRA 2005 Conference: Changing Views – Worlds in Play*. Vancouver, Canada, June 2005.

N. Lazzaro. "Why We Play Games: Four Keys to More Emotion without Story". *Presented at the 2004 Game Developers Conference*. San Francisco, California, March 2004.

S. Miller. Auto-Dynamic Difficulty. *Published in Scott Miller's Game Matters Blog (http://dukenukem.typepad.com/game_matters/2004/01/autoadjusting_g.html).* January, 2004.

R. Rouse III. *Game Design: Theory and Practice*. Second Edition. Wordware Publishing, Inc. 2004.

J. Sykes and S. Brown, S. "Affective Gaming: Measuring Emotion through the Gamepad". *Proceedings of the CHI 2003 Conference on Human Factors in Computing Systems*. Fort Lauderdale, Florida, April 2003.

A. Woolfolk, P. H. Winne, and N. E. Perry. *Educational Psychology*. Toronto, Ontario: Pearson Education. 2003.