AI for Dynamic Difficulty Adjustment in Games

Robin Hunicke, Vernell Chapman

Northwestern University
Computer Science Department
1890 Maple - Evanston, IL 60201
hunicke@cs.northwestern.edu, vernell@northwestern.edu

Abstract

Video Games are boring when they are too easy and frustrating when they are too hard. While most single-player games allow players to adjust basic difficulty (easy, medium, hard, insane), their overall level of challenge is often static in the face of individual player input. This lack of flexibility can lead to mismatches between player ability and overall game difficulty.

In this paper, we explore the computational and design requirements for a dynamic difficulty adjustment system. We present a probabilistic method (drawn predominantly from Inventory Theory) for representing and reasoning about uncertainty in games. We describe the implementation of these techniques, and discuss how the resulting system can be applied to create flexible interactive experiences that adjust on the fly.

Introduction

Video games are designed to generate engaging experiences: suspenseful horrors, whimsical amusements, fantastic adventures. But unlike films, books, or televised media – which often have similar experiential goals – video games are *interactive*. Players create meaning by interacting with the game's internal systems.

One such system is inventory – the stock of items that a player collects and carries throughout the game world.[1] The relative abundance or scarcity of inventory items has a direct impact on the player's experience.[2] As such, games are explicitly designed to manipulate the exchange of resources between world and player. [Simpson, 2001]

This network of producer-consumer relationships can be viewed as an economy – or more broadly, as a dynamic system [Castronova, 2000, Luenberger, 79].

[1] Inventory items for "first-person shooters" include health, weapons, ammunition and power-ups like shielding or temporary invincibility.

[2] A surplus of ammunition affords experimentation and "shoot first" tactics, while limited access to recovery items (like health packs) will promote a more cautious approach to threatening situations.

Game developers iteratively refine these systems based on play testing feedback – tweaking behaviors and settings until the game is balanced. While balancing, they often analyze systems intuitively by tracking specific identifiable patterns or types of dynamic activity. It is a difficult and time consuming process [Rollings and Adams, 2003].

While game balancing and tuning can't be automated, directed mathematical analysis can reveal deeper structures and relationships within a game system. With the right tools, researchers and developers can calculate relationships in less time, with greater accuracy.

In this paper, we describe a first step towards such tools. Hamlet is a Dynamic Difficulty Adjustment (DDA) system built using Valve's *Half Life* game engine. Using techniques drawn from Inventory Theory and Operations Research, Hamlet analyzes and adjust the supply and demand of game inventory in order to control overall game difficulty.

Prior Work

Commercial DDA

While it has potential benefits, DDA does not come cheaply. Ultimately, DDA systems take control away from the designer and put it in the hands of code, which has obvious drawbacks.

Few commercial developers have implemented adjustment systems for their games, and even fewer have shipped them. For a discussion of commercial DDA system designs and failures, see [Arey, Wells, 2001].

To the extent that commercial systems have been formally proposed or published, there are two basic approaches. The first is predominantly manual; designers annotate game tasks and obstacles with information about their difficulty prior to evaluation [Pfeiffer, 2003]. The second champions a combination of data mining and off-line analysis [Kennerly, 2003].

We propose a probabilistic technique that dynamically evaluates the difficulty of given obstacles based on user performance, as the game is running.

Genres

Difficulty adjustment can be applied in almost any game genre, but is often discussed within the context of Massively Multiplayer Online Games. Due to the size of these games, and their largely stats-based nature, even *slight* imbalances can have an extremely negative impact on the overall player experience – but tweaking the live game is also disruptive.

As a result, MMOG developers want to know which elements should be tuned, and how that can be achieved successfully in a live game. Specifically – when is it necessary to tweak an in-game system? Which adjustments will result in direct, predictable and desirable changes to the system dynamics? [Carpenter, 2003]

FPS game economies are relatively simple compared to online game economies. But the combat economics of FPS games are often repeated in sub-economies of MMOG and other game spaces. We investigate the FPS environment as a first step towards understanding the dynamics of larger, more complex game economies.

Hamlet System

Architecture

The Hamlet system is primarily a set of libraries embedded in the *Half Life* game engine. This includes functions for

- Monitoring game statistics according to pre-defined metrics.
- Defining adjustment actions and policies.
- Executing those actions and policies.
- Displaying data and system control settings.
- Generating play session traces.

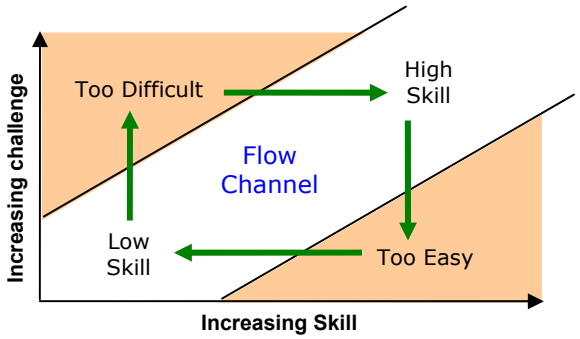
As the player moves throughout the game world, Hamlet uses statistical metrics to monitor incoming game data. Over time, Hamlet estimates the player's future state from this data. When an undesirable *but avoidable* state is predicted, the system intervenes and adjusts game settings as necessary.

Basically, we are trying to predict when the player is "flailing" – repeatedly edging towards a state where her current means can no longer accomplish necessary and immediate ends. When we detect flailing behavior, we want to intervene – helping the player progress through the game.

Flow

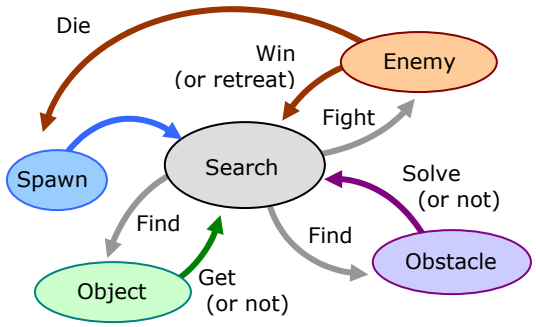
Game adjustment is not as simple as adding health when the player is in trouble; it is a design problem that involves estimating when and how to intervene. Keeping the player challenged and interested is especially difficult in interactive contexts. [LeBlanc et al, 2001-2004].

One common approach to player investment is the flow model developed by M. Csikszentmihalyi. Here our goal is to keep the player in the *flow channel* – away from states where the game is far too challenging, or way too easy.



Flow state as it transitions over the course of a dynamic experience. Challenge and pacing must ramp to match skill, in order to support continued engagement.

Looking at the FPS environment, we can abstract gameplay with a relatively simple state transition diagram. Players engage in loops of searching, retrieving, solving and fighting. With each new level, new enemies and obstacles are introduced. Overall, difficulty increases with time, as does skill acquisition.



A simplified state transition diagram for the First Person Shooter game genre.

Hamlet is designed to keep the player in the flow channel by encouraging certain states, and discouraging others. Effectively, our goal is to keep the player in engaging interaction loops, for the most appropriate period of time, given their level of overall skill and game-specific experience.

By conducting a cheap, *abstract simulation* of the player's progression through state space, we hope to predict when the player is repeatedly entering an undesirable loop, and help them get out of it.

Goals

There are many takes on whether this type of adjustment is "desirable" or "necessary" for a good game experience. Without delving into a philosophical discussion of play aesthetics or game design, we can parameterize our goals with respect to adjustment as follows.

We want Hamlet's modifications to encourage both local and long-term success and engagement. We want the system to *support* the player – without eliminating negative feedback and making everything very easy and predictable We want Hamlet to intervene *just enough* – so that system behavior is relatively stable and predictable over time [Rollings, Morris, 2000]. As such, we aim to:

- Assess when adjustment is necessary
- Determine which changes should be made
- Execute changes as seamlessly as possible

We will now discuss assessment and adjustment in light of these goals, state our observations regarding overall system behavior, and close with preliminary conclusions.

Assessment

Our goal is to determine when a player is *flailing*. In many FPS games, this is characterized by repeated inventory shortfalls – places where the player's available resources fail to meet the immediate demands.

By observing trends in the player's inventory expenditure, we can watch for potential shortfalls and thus, pinpoint potential adjustment opportunities. Our first task is to establish metrics for assessing statistical data within the game world. Let us start with the direct observation of player health – and more specifically, the damage a player takes over time.

Damage

We begin by assuming a sequence of random measurements of damage **x**(t), each of which has some probability distribution p_d.

The total damage at a given point in time is then the sum of these random variables. By the Central Limit Theorem, this sum converges to a Gaussian distribution.[3]

[3] In practice, these probabilities will vary depending on the application domain. Here, we assume the probability of being hit is constant, and so the process is stationary. We also assume that the probability of being hit at one moment is independent of whether you were hit before, so the process is uncorrelated, or "white".

Because we are summing the distributions

$$\sum_{i=1}^{t} \mathbf{x}(i)$$

The mean μ of the resulting sum is the sum of the means

$$\mu = \mathrm{E}\left(\sum_{i=1}^{t} x(i) \right) = t\mathrm{E}(x(t)) = t\mu_d$$

And the variance is the sum of the variances

$$\sigma^2 = V\left(\sum_{i=1}^{t} x(i) \right) = t{\sigma_d}^2$$

$$\sigma = \sqrt{t}\sigma_d$$

The distribution of damage then becomes

$$p(x) = \frac{1}{\sigma\sqrt{2\pi}} e^{-(x-\mu)^2/2\sigma^2}$$

We can now state the cumulative distribution function of the Gaussian distribution – also known as the *error integral*. Here, $F(d)$ represents the probability of receiving d or less damage on a given tick:

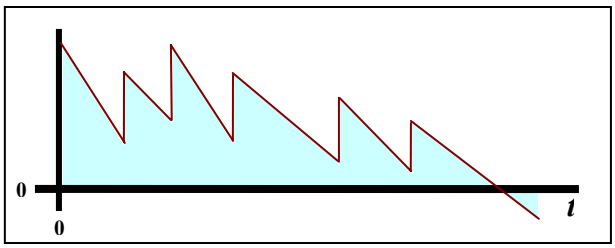
$$F(d) = \int_{-\infty}^{d} p(x)dx = 1 \Big/ \left[\sigma\sqrt{2\pi} \right] \int_{-\infty}^{d} e^{(x-\mu)^2/(2\sigma^2)} dx$$

Obviously, this is not a closed form equation. But lucky for us, the error integral is implemented as erf(x) in the standard C and C++ math libraries.

Now that we know what the distribution of damage looks like, we can start to approximate inventory levels for player health.

Inventory

In any given system, the level of inventory will depend on the rates of flow in and out. This is often discussed in terms of supply (input flow) and demand (output flow) [Mankiw, 1997].



Inventory level as it fluctuates over the period of time *t.* A shortfall occurs when demand surpasses supply.

In the FPS domain, player inventory continually shifts. Items are supplied as the player explores the environment (found in the open, discovered inside crates, or field-stripped from the bodies of foes), and depleted by various interactions within it (ammunition and health spent in combat, damage taken from falls, exposure to poison and so on).

Using inventory theory equations, we can model the player's overall inventory as well as the flow if specific inventory items in and out of the system.

Shortfalls

To predict an inventory shortfall, let z denote the inventory level at the given time. The expected shortfall is the cumulative probability of damage exceeding the initial level or P_s, calculated as follows:

$$P_s = P(x(t) > z) = 1 - P(x(t) < z) = 1 - F(z)$$
$$= 1 - \frac{1}{\sigma\sqrt{2\pi}} \int_z^{\infty} e^{-(t-\mu)^2/2\sigma^2}\, dt$$

With this equation, and our handy *erf* functions, shortfall can be computed as a function of initial health, mean and standard deviation of loss of health over time.[4]

We can use this calculation of eminent health shortfalls as a crude indicator of when the player is in need.[5] Then, we can move beyond assessment to adjustment – analyzing the systems that impact the inventory item in question, and adjusting the game accordingly.

[4] This approach is best at predicting events at times in the not-too-distant future. With small sample sizes, the CLT is less accurate and standard deviation more likely to fluctuate.

[5] In application, these computations can be modified according to the specific game under consideration. Different games may require more complicated distributions, exhibiting a significantly different mean or standard deviation.

Adjustment

We are currently experimenting with a number of adjustment protocols in the Hamlet system. Adjustment **actions**, when combined with **cost estimations**, will form adjustment **policies**. What follows is a description of action, cost evaluation and policy designs, and an example illustration of the final goals for player-system interaction.

Actions

When completed, Hamlet will support two types of adjustment actions.

> **Reactive** actions will adjust elements that are "in play" or "on stage" (i.e. entities that have noticed the player and are attacking). This includes directly manipulating the accuracy or damage of attacks, strength of weapons, level of health, and so on.

> **Proactive** actions will adjust "off stage" elements (i.e. entities that are spawned but inactive or waiting to spawn). This includes changes to the type, spawning order, health, accuracy, damage and packing properties of entities that have yet to appear on screen.

While proactive changes give us more power over the game's behavior, they happen at a greater distance from the point of action – which can introduce uncertainty as to their effectiveness. As a result, they are harder to evaluate. In the worst case, they may actually require subsequent reactive adjustments, causing a spiraling loop of change, resulting in erratic and unpredictable behavior.

Reactive adjustments map more directly to changes in the gameplay experience. They are simple to execute, and happen closer to the point of intersection/interaction between the player and game system. Changing the strength or accuracy of an entity that is under attack has a straightforward impact on its life expectancy – and upon on the life expectancy of the player.

In the end, it is a tradeoff. Reactive changes run the risk of disrupting the player's sense of disbelief. They can make it difficult to interpret the immediate behavior of in-game obstacles, causing the game to appear schizophrenic.[6] Proactive changes, because they happen "in the wings", are less likely to interrupt the player's suspension of disbelief.

[6] A problem that plagues many interactive AI research projects [Sengers, 1998].

Cost

Determining the cost of a given action, then, is clearly critical to the process of gradual but effective adjustment. If the system intervenes in the same way every time, the game might start to feel trivial, repetitive or boring. If it continually makes interventions without considering the player's perception of those interactions, it may interrupt their overall suspension of disbelief.

Individual adjustment actions will be offset by observations about

- The player's current location in the level.
- How far along the player is in the game.
- How often they've died (at this location, in the level, and in the game)
- How often they've repeated the current encounter
- How many times we've intervened in the past.

By calculating a cost for each intervention, and modifying it as our understanding of the player's performance changes, we hope to dynamically adjust the game in more responsive and responsible fashion.

Policies

Hamlet can combine these actions and cost calculations to create "modes of control" – overall adjustment policies that control the supply and demand of goods according to overall player experience goals. Two examples follow.

> **Comfort Zone:** This policy will keep players within a mean range of health points over time. Entities will be tuned to shoot less often and less accurately, and health is readily available. The goal is to keep the player at about 50% health, occasionally dipping near 25% or cresting to 75%.

Trial and error are important in this policy – enemies will be tuned only if they repeatedly overwhelm the player. Much like a benevolent babysitter, it intervenes frequently, but leaves room for mistakes. Overall, the policy will be characterized by steady demand and predictable supply.

> **Discomfort Zone:** This policy is designed for more experienced players. The entities in a DZ game are increasingly accurate, ammo and health relatively scarce. The goal here is to keep the player "on the edge of her seat" – constantly on the alert for enemies, and fighting back from 15 or 20% health most of the time.

This policy is much more like an aggressive drill sergeant or boxing coach. It sets high standards, but delivers enough positive feedback to keep the player energized and engaged. This policy will be characterized by scarce but goal-oriented supply and sporadic but high demand.

Example

A player reaches the second engagement of *Case Closed*. There are four enemies, and she has 45% health. Hamlet observes her inventory stats, to see if she is flailing – in this case, repeatedly dying before completing the engagement, often with a majority of enemies standing.

The second enemy encounter of Case Closed.

In immediate response to observed flailing, Hamlet may

- Donate a health pack somewhere in the scene.
- Upgrade the strength of her ammo.
- Reduce the accuracy or strength of enemy attacks.

Depending on the success of these individual actions, Hamlet can intervene again. If, over time, the player requires *significant* adjustments to the initial levels set by the game designer, Hamlet can also

- Reduce initial strength and health of pending enemies.
- Pack more health and ammunition on the bodies of vanquished foes.
- Replace pending enemies with a type the user has had more success with in the past.

The key here is that Hamlet will intervene iteratively, allowing for trial and error. The game will gradually change to accommodate the current player.

Preliminary Conclusions

Simulation

It's clear that work of this nature involves tradeoffs with respect to the type, number and frequency of adjustments performed. There is also a performance tradeoff here between accuracy and generality/speed.

If our goal was to do a "correct" simulation of each individual user's performance, we would need to perform several trials per user. However, as we are trying for a "best fit" given current information, this is not necessary.

In fact, we assume that a perfect simulation of the player's progress through the environment would actually be *less useful* due to the time constraints of real-time gameplay and the limitations of commercial gaming hardware.

By conducting an *abstract simulation* of the user's trajectory, we essentially generate an *envelope of possible locations* within the state space, and construct our system to direct that envelope away from undesirable areas of gameplay.

Evaluation

While more advanced techniques can map out many of the finer relationships in game systems, it is unclear to us at this time whether this is necessary, or even desirable. Future work may comment on the viability and value of alternative metrics, policies and so on.

In the meantime, we plan to evaluate a broad sample of users interacting with the Hamlet system. We will attach a simple heart-rate monitor to a variety of players, have them play the game on both settings, and then map out the results.

If the adjusted game is able to keep test subjects equally aroused over time, while dynamically adjusting to keep them alive longer, then we will consider the work a success. [7]

References

Adams, E., 2002. *Balancing Games with Positive Feedback.* Gamasutra.com.

Arey, D., Wells, E., 2001. *Balancing Act: The Art and Science of Dynamic Difficulty Adjustment*. 2001 Game Developers Conference, San Jose

Bethke, E., 2003. *Game Development and Production.* Plano, Texas: Wordware.

Castronova, E., 2001. *Virtual Worlds: A First-Hand Account of Market and Society on the Cyberien Fronier.* Working Paper, Center for Economic Studies and IFO Institute for Economic Research: Munich, Germany.

Carpenter, A., 2003. *Applying Risk Analysis to Play-Balance RPGs.* Gamasutra.com.

Csikszentmhalyi, M.1990. *Flow: The Psychology of Optimal Experience.* NY, NY: Harper Collins
Berkely, California: Osborne/McGraw-Hill.

Crawford, C. 1984. *The Art of Computer Game Design.* Berkely, California: Osborne/McGraw-Hill.

Jensen, P. A., Bard, J. F., 2002. *Operations Research Models and Methods.* Hoboken, NJ, Wiley.

Kennerly, D., 2003. *Better Game Design through Data Mining.* Gamasutra.com.

Khoo, A., Hunicke, R., et al, 2002. *FlexBot, Groo, Patton and Hamlet: Research using Computer Games as a Platform*. Technical content paper for Intelligent Systems Demonstration, *Proceedings of the Eighteenth National Conference on Artificial Intelligence*.

LeBlanc, M., Hunicke, R., et al 2001-2004 – *Game Tuning and Design Workshop*. Game Developers Conference: 2001, San Jose.

Luenberger, D. G., 1979. *Introduction to Dynamic Systems: Theory, Models, and Applications*. New York: John Wiley and Sons, Inc.

Mankiw, N. G., 1997. *Principles of Microeconomics*. Fort Worth, Texas: Dryden.

Pfeiffer, B., 2003. *AI to Control Pacing in Games.* Proceedings, IC2 GameDev Workshop, Univeristy of Texas, Austin.

Rabin, S., 2002. editor, *AI Programming Wisdom,* Hingham, Massachusetts: Charles River Media.

Rollings, A., Adams, E., 2003. *On Game Design*. Indianapolis: New Riders.

Rollings, A., Morris, D., 2000. *Game Architecture and Design*. Scottsdale, Arizona: Corliolis.

Rouse, R., 2001. *Game Design: Theory and Practic,* Plano, Texas: Wordware.

Stengel, R. F., 1994. *Optimial Control and Estimation.* New York: Dover.

Sengers, P., 1998. *Anti-Boxology: Agent Design in Cultural Context.* PhD thesis, Carnigie Melon University, Pittsburg.

Simpson, Z., 1999. *The In-game Economics of Ultima Online.* Game Developers Conference: 2000, San Jose.

[7] And then - we'll try it on our parents!