δ-logit : Dynamic Difficulty Adjustment Using Few Data Points

William Rao Fernandes, Guillaume Levieux

δ-logit : Dynamic Difficulty Adjustment Using Few Data Points

William Rao Fernandes and Guillaume Levieux

CNAM CEDRIC, Paris, France
{william.rao_fernandes,guillaume.levieux}@cnam.fr

Abstract. Difficulty is a fundamental factor of enjoyment and motivation in video games. Thus, many video games use Dynamic Difficulty Adjustment systems to provide players with an optimal level of challenge. However, many of these systems are either game specific, limited to a specific range of difficulties, or require much more data than one can track during a short play session. In this paper, we introduce the δ-logit algorithm. It can be used on many game types, allows a developer to set the game's difficulty to any level, with, in our experiment, a player failure error prediction rate lower than 20% in less than two minutes of playtime. In order to roughly estimate the difficulty as quickly as possible, δ-logit drives a single metavariable to adjust the game's difficulty. It starts with a simple $+/-\delta$ algorithm to gather a few data points and then uses logistic regression to estimate the players failure probability when the smallest required amount of data has been collected. The goal of this paper is to describe δ-logit and estimate its accuracy and convergence speed with a study on 37 participants playing a tank shooter game.

Keywords: Difficulty · Dynamic Difficulty Adjustment · Game Balancing · Player Modeling · Motivation · Video Games

1 Introduction

Difficulty in video games is a fundamental factor of enjoyment and motivation [16, 15, 22, 20, 14]. Flow Theory suggests that one can reach a state of optimal enjoyment when a task level of challenge is set with regard to their own perceived skills [18]. To adjust the balance between challenge and skills, video games can either use static, predefined difficulty levels or rely on Dynamic Difficulty Adjustment (DDA) systems. However, few of these systems can target any level of difficulty, are generic enough and use a small amount of data. In this paper, we introduce and evaluate such a system, that we call *δ-logit*.

First, we want our system to be as **generic** as possible: we want to be able to use it with as many different games as possible and rely on a measure of difficulty that allows comparison between games. Following our previous work, we model a game as a follow up of *challenges* that can either be won or lost and whose difficulty can be manipulated using a set of variables [3]. Indeed, the notion of success and failure is at the core of video games: in many of them,

the players have clear goals and their performance is constantly evaluated. Each of the players' success or failure have an impact on the game's progression and are conveyed to them using audio, visual or haptic feedbacks. We thus propose to start from these events to define a set of challenges, and then track players' failures and successes to estimate their failure probability for these challenges. Such a challenge can be, for instance, jump on a platform, shoot at another player, or win a battle against enemy tanks. We consider that failure probability to these challenges is close to how challenging and difficult a video game is.

Second, we want to be able to choose **any level of difficulty**. As we will see, some simple algorithms only balance difficulty towards a 0.5 failure probability. We, however, want to be able to instantly select any level of difficulty, as many games do not target 0.5 balanced difficulty [1]. Indeed, some imbalance between skills and challenge can lead to desirable emotional states e.g. arousal, control or relaxation [18].

Third, we want to propose a model that uses **as few data points as possible**, gathered from one player only. We record one data point every time the player tries a challenge, and want to predict difficulty using less than 20 points. The two previous goals can be achieved using various techniques, but many of them require a lot of data, either tracked from many players or generated. In this paper, we want our system to handle a cold start and reach a sufficient accuracy within the shortest playtime. This way, our model can be used in offline games, where the only data available can be the data of a single local player starting the game with no tracked data. Of course, we will thus have to define what *sufficient accuracy* means for our game. In this study, we only use the predictive part of our algorithm when the prediction error rate is higher than 40%, as we thus consider that our predictions are too close to randomness to be used. Our experiment shows that we can reach error rates lower than 20% in less than two minutes of playtime, and our actual player failure rates will be close to our targeted failure rates (fig. 2). Our experiment shows what our δ-logit approach is able to achieve in the context of a shooter game, but it is to note that the required accuracy for a DDA system is still an open question, as perception of difficulty is a very complex matter [7, 6].

2 Simple DDA Algorithms

One of the simplest DDA algorithms is to slightly raise the difficulty when the players won and slightly lower it when they failed. We call it the $+/-\delta$ algorithm. Constant et al. used this algorithm to study the link between DDA and confidence[6]. In the mainstream video game *Crash Bandicoot*, when the player dies a lot, the game gives them power-ups or checkpoints so their progression is eased, making the game more balanced [10].

Another approach is the rubber band AI, especially prevalent in racing games like Mario Kart [24]. The goal of the rubber band AI is to adjust the parameters of the computer-controlled opponents with regard to their distance to the player

as if a rubber band was pulling them towards the player. Opponents ahead of the player will be slowed down, while those behind him will be sped up.

Such simple systems can be manually tuned to provide a balanced play experience using a very small amount of data. However, none of them can't directly adapt the difficulty towards a specific failure probability. Moreover, rubberband algorithm is only suited to games with opponents.

As we will see in section 4, δ-logit uses $+/-\delta$ as a fallback strategy when not enough data is available. But as soon as possible, we need to switch to a more advanced strategy to be able to target any level of difficulty.

3 Advanced DDA Algorithms

Other difficulty adaptation methods were developed using learning algorithms. Andrade et al. extended Q learning to dynamically adapt a policy when playing against a human [2]. However, this approach is only possible for games involving some sort of AI that can first play against itself to develop a policy. Spronck et al. propose a similar technique called dynamic scripting, which can be considered close to Q-learning except that actions are replaced by manually authored action scripts [21]. Thus, this DDA policy suffers from similar drawbacks as the previous one. The same goes for DDA systems that rely on Monte Carlo Tree Search (MCTS) to build an adapted opponent [11, 8, 13], or when real-time neuroevolution of opponents' AI is used [19]. These approaches can only be used when the game features some kind of opponent whose decision can be modeled using either a tree structure or a neural network, and where the game features some kind of synthetic player, allowing the AI to build its strategy while quickly exploring the game space by fighting this player.

Hocine et al. adapt a therapeutic game using a generic DDA system, that can be applied to any video game as long as a success probability can be estimated [12]. They base their evaluation of difficulty on player failure probability, but follow a strategy similar to the $+/-\delta$ algorithm, as they lower the difficulty when the player loses and raise it when they succeed and thus can only target a 0.5 balanced state. Zook et al. use tensor reduction to adapt the difficulty of a custom RPG game [25]. Their approach allows to predict spell effectiveness for a specific player at a specific time but does not provide a more generic measure of difficulty like failure probability. Allart and Constant used a mixed-effect logistic regression to evaluate both commercial and experimental games difficulty [6, 7, 1]. Logistic regression seems indeed well suited to predict a failure probability from few samples and binary outcomes. In these works, mixed-effect logistic regression was used because these studies had access to the data of many players with repeated trials of the same challenge. In our case, we want to use few data points from the current player only and thus we can only use a fixed effect logistic regression. Also, these studies did not use logistic regression to dynamically balance the game, but to evaluate the difficulty for post-experiment analysis purposes. Thus, they only compute the regression at the end, when all the experiment data is available. We thus still need to experiment whether using

logistic regression from the start of the play session, in real time, is a viable option or not.

4 δ-logit : DDA using few Data Points

4.1 A single metavariable: θ

Our goal, as explained in the previous sections, is to develop a model that can instantly target a specific failure probability while using as few data points as possible, i.e. while having only observed a few attemps of the player to win the challenge. To do so, we propose to only rely on a single meta-parameter to balance the gameplay, that we name θ. Using this single parameter limits our ability to fine-tune the game's difficulty, but also drastically limits the search space of our δ-logit algorithm.

Following [3], we define a challenge as a goal players are trying to reach, and for which they may win or fail e.g. shoot a target, jump on a platform, finish a mission. We then find a subset of variables that we can modify to change this challenge's difficulty, from one player's try to another. Then, we define two challenge configurations, that is, two sets of values for these variables. Those two configuration have to be defined manually by a designer. The first configuration is the easiest challenge that the algorithm is allowed to create, while the second is the hardest. These extreme configurations prevent us from proposing challenges that we consider undesirable to any player, due to their extreme values. Then, θ is used to linearly interpolate each parameter between the very easy and very hard challenge configurations. δ-logit thus only drives θ to adjust the game's difficulty for each specific player. θ varies between 0 and 1, if the parameter is not continuous, we interpolate it and then round it to the nearest integer.

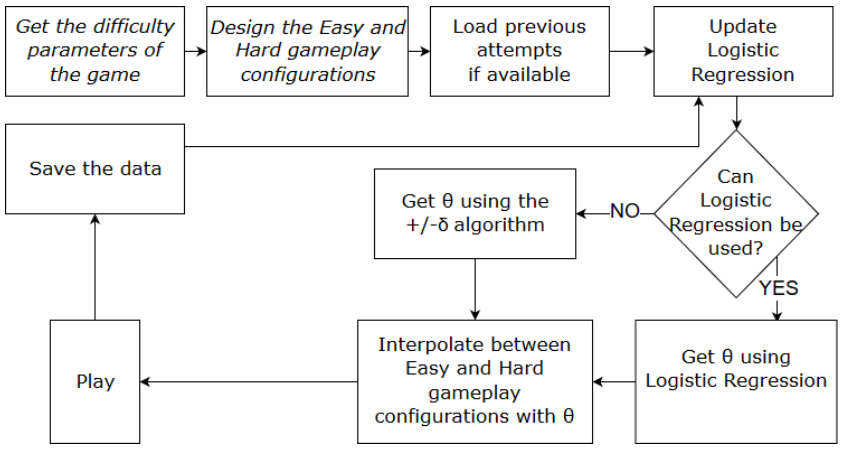

Fig. 1. Flowchart of the δ-logit algorithm
Steps in italic are design steps that needs to be done manually

4.2 Exploring with +/-δ Algorithm

When no data is available, as the player starts playing for the first time, δ-logit uses a very simple algorithm to explore the game space while balancing the gameplay. We chose to use the +/-δ algorithm as it is not specific to a particular game genre and adapts the difficulty using only the player's last result. If the player wins, +/-δ raises the difficulty by δ_{win}, if they fail it lowers it by δ_{fail}. If $\delta_{win} = \delta_{fail}$, the difficulty eventually oscillates around a gameplay configuration where the player has a 0.5 failure probability.

During this exploration phase, we are just able to drive the difficulty toward a balanced state and can't target a specific failure probability. But if we do not simulate the game in advance using a synthetic player and do not possess any data about the player, this exploration phase is mandatory. We propose to start from the very easy challenge configuration and let the +/-δ raise the difficulty.

More specifically, we propose to use a δ_{win} and δ_{fail} value close to 0.05. Indeed, it is often considered that logistic regression can only be performed with a minimum of 10 to 20 data points per variable [5]. Thus, if we start from the very simple case where $\theta = 0$, we can reach $\theta = 0.5$, just in between the hardest and easiest configuration, if the player wins 10 times and the +/-δ adds $10 * 0.05 = 0.5$ to θ. This would allow us to have data points spread between $\theta = 0$ and $\theta = 0.5$ as soon as we start to estimate the logistic regression, as it is the case in our experiment. However, this is just a rule of thumb we followed and one may choose to use different values of δ_{win} and δ_{fail} to provide players with a slower or steeper learning curve.

It is to note that if δ_{win} and δ_{fail} are fixed values, the +/-δ will always sample a limited set of data points. Indeed θ may start at 0, then be 0.05 if the player wins and then again 0 if they fail. But we will never sample values between 0 and 0.05. To ensure a better exploration of θ values, we apply a random uniform noise to δ_{win} and δ_{fail}. In our experiment, δ_{win} and δ_{fail} were drawn from a uniform distribution $\mathcal{U}(0.05, 0.1)$, ensuring that δ was never under 0.05 but could still vary up to twice this value, allowing us to sample θ values at a wider, variable range.

4.3 Adding Logistic Regression

δ-logit switches to logistic modeling of difficulty as soon as the +/-δ provided enough data points. Logistic regression allows us to estimate a probability of failure from binary results, in a continuous way, that can start to provide an estimation with as few as 10 data points[5]. Once the regression is performed, the model is able to estimate the value of θ that corresponds to the desired probability of failure.

To update the logistic regression, we iteratively fit a logistic function to the available data points using the Newton-Raphson method. We adapted the C# code provided by McCaffrey to use it in the Unity Game Engine to perform our experiment [17].

To estimate if we can switch from the +/-δ algorithm to the Logistic Regression, we perform several tests, summarized in Figure 1. First, we do not compute the regression if we gathered less than 10 data points, following [5]. Then, as these first data points are mostly sampled on the lowest difficulty levels, we only start to perform the regression if we gathered at least 4 successes and 4 failures. If these basic conditions are met, we perform the logistic regression and check its accuracy using 10-fold cross-validation. Cross-validation is computed by using our logistic regression as a binary predictor[1] of success and failure and by comparing predictions to actual results. In our experiment, we only use the logistic regression if it's estimated accuracy is higher than 0.6. We empirically chose to use 4 successes/failures and a 10-fold cross-validation score higher than 0.6. These values performed well in our experiment but it might be worth running other experiments to investigate different values.

When it has switched to the logistic regression, δ-logit is thus able to estimate the value of θ that corresponds to a specific failure probability. This value can be driven by any process: for instance, a designer might want to target difficulty values following a curve that oscillate around an 0.5 value, allowing the player to experience a globally balanced gameplay, while having periods of arousal when the difficulty is higher and a feeling of control when it is lower, as suggested by [18].

As for the +/-δ, we want our algorithm to keep exploring different values of θ. To do so, we propose to add noise to the failure probability requested by the game. Empirically, we add a value drawn from a uniform distribution $\mathcal{U}(-0.05, 0.05)$ to the requested failure probability when using the logistic regression to estimate θ. That way, if a designer asks for a difficulty of 0.2, the model will estimate the value of θ for a difficulty randomly picked between 0.15 and 0.25. This allows us to have values of θ that always vary and we consider that the player's perception of difficulty is not accurate enough to perceive such a subtle difference [7]. However, difficulty perception is a complex matter and in further studies, we should investigate the impact of this parameter on both perception of difficulty and difficulty estimation accuracy.

5 Adapting a Shooting Game

We implemented δ-logit in a tank shooting game (Figure 2). We started from the Unity Tutorial *Tank Shooter Game*[23], modified the controls so that the player still manipulates the tank using the keyboard arrows but can shoot in any direction using the mouse. We also added AI to the enemy tanks. The flow of the game is very simple: the player and one or two enemy tanks are spawned. They can shoot at each other and move. A tank shell explodes when it hits the ground or a tank and applies damage to the tanks close to the explosion. If the player kills the enemies they win and if they die they fail. Every time, we record the value of θ and the game's result, i.e. whether enemy tanks were destroyed

[1] If *p(fail)* > 0.5, predict failure and predict success otherwise

or not. At the beginning, we do not have enough samples so δ-logit starts from the easy condition $\theta = 0$ and follows the $+/-\delta$ algorithm. Then, as soon as the logistic regression is ready (see Section 4.3), δ-logit uses it to estimate the value of θ corresponding to the chosen difficulty and spawn the player and the enemy tanks again.

The enemy tanks have different characteristics that can be modified to change the game's difficulty, as shown in Table 1. It is to note that the number of enemies will double as θ crosses the 0.5 value. We compute the value of θ with our DDA system and use it to interpolate these parameters between easy and hard settings.

Table 1. Easy and Hard Settings

Game Parameters	Easy Setting	Hard Setting
Nb of Enemies	1	2
Moving Speed	1.2	9.6
Turning Speed	18	900
Time Between Shot	3s	0.5s
Accuracy	0	15

Turning speed in degrees.s^{-1}. Moving Speed is in unit.s^{-1}, a tank's length is 2 units and the game space is 76 units per 47 units. Accuracy: we add a random 2D vector of size 0 to 15 units to the targeted position.

5.1 Methodology

Participants played 60 turns, thus spawned 60 times, corresponding on average to less than ten minutes of gameplay. Play sessions were short because we wanted players to stay concentrated, and because the experiment's main goal is to evaluate the accuracy of our model when few data points are available. We use δ-logit described in section 4, starting with no data and thus with the $+/-\delta$ algorithm.

When δ-logit switches to the logistic regression, as described in section 4.3, we target specific levels of difficulty. We chose to evaluate our model for failure probabilities of 0.2, 0.5 and 0.7. The 0.2 difficulty is far from the 0.5 balanced setting that can be reached with the simple $+/-\delta$ algorithm, while still being a bit challenging. It is also close to the average difficulty of some AAA games [1]. We target the 0.2 difficulty until turn 44. Then we test if, having sampled many data points while playing at a low difficulty level, we are able to create accurate difficulty peaks. So from turn 45, we start a cycle of three turns with different difficulties, beginning at 0.2, rising to 0.5, ending at 0.7. We repeat this cycle five times, up to turn 60. Each turn takes on average less than 8 seconds to complete if the player understands the goal of the game. We follow such a difficulty curve because it follows many game's difficulty pacing: slowly raise the difficulty when the player discovers the game rules, then propose a certain level of challenge, until you reach a difficulty peak, like the *bosses* of many games.

6 Results

37 participants played our shooter game (26 male, 11 female), with a mean age of 30 (σ = 8.6). All participants played 60 turns, for an average of 7 minutes of playtime (σ = 82 seconds). Figure 3 describes the evolution of the difficulty parameter θ for all the participants.

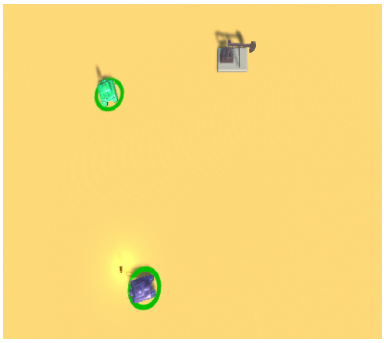

Fig. 2. The tank shooting game
The player, at the bottom of the screen, is being shot at by an enemy tank.

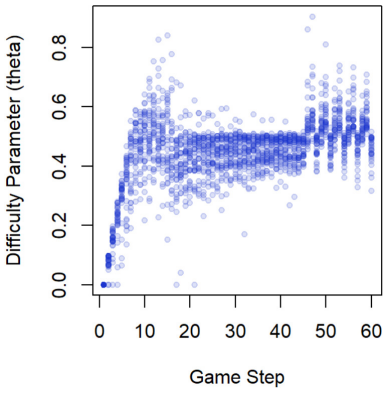

Fig. 3. Evolution of difficulty parameter θ
One can distinguish the +/-δ phase for 15 turns on average, followed by the 0.2 difficulty phase up to turn 44, and then the 5 difficulty peaks.

We first checked the level of the participants, by using the mean of our difficulty parameter θ across the whole game session. As difficulty is dynamically adapted to have all players experience the same failure probabilities, best players will have higher values of θ. Players levels are ranging from 0.3 to 0.6 ($\mu = 0.46$, $\sigma = 0.05$).

δ-logit performs logistic regression to estimate the failure probability from the values of θ and each turn outcome. Figure 5 illustrates this estimation : for most of the participants, the game starts to be challenging when failure probability starts raising, at $\theta = 0.4$ and is very hard when $\theta >= 0.6$ as failure probability is above 0.75.

We then looked at the model convergence[2] speed. We first calculated, for each participant, the number of +/-δ turns before the model switched to the logistic regression for the first time. The model took on average 15 turns to converge ($\sigma = 1.82$ turns), corresponding to 105 seconds of gameplay ($\sigma = 24.52$ seconds).

We also calculated the model variability for each turn. We used the model at turn t to predict the failure probability for 21 values of θ, from 0 to 1 by

[2] We consider that our model has *converged* when it is able to use logistic regression to adapt the difficulty

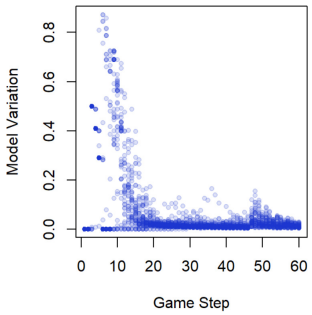

Fig. 4. Model's variation for each participant at each step
Logistic regression's variation between each turn. Variability drops when 20 data points are gathered. Variability peaks at turn 45, when we create the 5 difficulty pikes.

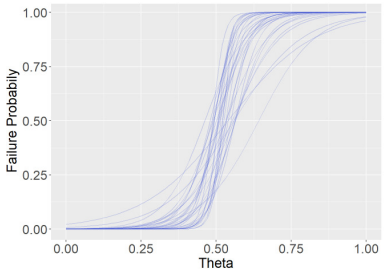

Fig. 5. Logistic regression for each participant at the 60th turn
Players always win on the easy setting, always fail on the hard one. Difficulty changes very quickly around $\theta = 0.5$, being still very low at $\theta = 0.4$ and already very high at $\theta = 0.6$.

steps of 0.05. We then computed the root-mean-square error (RMSE) between predictions at step t and those at steps $t-1$, $t-2$ and $t-3$ as given by the equation (1). We computed the distance with the last steps to have larger values for models varying in the same direction than for those oscillating around a value.

$$RMSE = \sqrt{\sum_{i=1}^{3} \frac{\sum_{j=0}^{20} (p_t(\theta = j/20) - p_{t-i}(\theta = j/20))^2}{3 * 21}} \tag{1}$$

The model variability can be examined over time, as shown in Figure 4. On average, the logistic regression was used after 15 turns, and we can see that from turn 20, the prediction tends to be much more stable. One can also notice a peak of variation after turn 45, corresponding to the difficulty peaks we included in the game. Those peaks forced the model to explore higher values of θ, and thus to readjust accordingly.

Another way to look at the model accuracy and convergence speed is to look at the cross-validation result over time. When logistic regression is used, the obtained accuracy has a mean of 0.82 ($\sigma = 0.06$). Interestingly, it is to note that for 4 out of 31 players, we switched back to +/-δ algorithm even long after the 15 first steps. These players stayed in +/-δ for an average of 2.75 steps ($\sigma = 2.22$), meaning that the model can occasionally lose accuracy.

We targeted three levels of difficulty (*p(fail)* = 0.2, 0.5 and 0.7) and the model was able to achieve the failure probabilities presented in Table 2. Actual failure probabilities are estimated for all participants by taking the mean of their actual success (1) and failures (0) when the model was either targeting *p(fail)* = 0.2, 0.5 or 0.7. It is to note that the model never exactly targeted these values, as

a uniform noise $\mathcal{U}(-0.05, 0.05)$ was applied to them, see Sec. 4.3. Target failure probabilities are centered on 0.2, 0.5 and 0.7, but have a standard deviation of 0.03.

Table 2. Actual failure frequencies for each target failure probabilities

Target difficulty	Objective difficulty
0.2	0.14 [0.12, 0.16]
0.5	0.55 [0.47, 0.62]
0.7	0.74 [0.67, 0.80]

For each target difficulty, we provide the observed failure frequencies. Values between brackets are the 95% CI values given by an Exact Binomial Test.

7 Discussion

Our algorithm was able to successfully adapt the difficulty of the game to match the difficulty curve wanted by a designer, see Table 2. One can note, however, that for *p(fail)* = 0.2 we are slightly lower (0.14), whereas for *p(fail)* = 0.5 and 0.7 we are slightly above (0.53 and 0.76). This might be explained by the nature of gameplay's progression.

As explained before, we change the difficulty variables all together following θ: tanks become more accurate, faster and come in larger numbers at the same time. On the one hand, this allows us to have a continuous and monotonic difficulty: if we had chosen to first change speed and then accuracy, both these variables may not have the same impact on objective difficulty and create a change in progression when switching from one variable to the other. On the other hand, this approach might have the drawback of compressing objective difficulty in a short range of θ. Indeed, the objective difficulty might grow exponentially with θ as all the parameters raise at the same time. This can clearly be seen in our difficulty curve mapping θ to objective difficulty (Figure 5): the game is very easy when $\theta <= 0.4$ and very hard when $\theta >= 0.6$.

Moreover, we chose to have a gameplay progression variable that has only two values: the number of enemy tanks. We think that this might explain why objective difficulty is lower than the targeted difficulty in the easy setting and higher in the hard setting: when $\theta > 0.5$, meaning there are two tanks to beat, the difficulty rises much faster than when $\theta < 0.5$. A tank at $\theta = 0.49$ is almost as strong as a tank at $\theta = 0.5$, but the number of tanks creates a difficulty peak at $\theta = 0.5$ and changes the slope of the impact of θ on objective difficulty when θ crosses 0.5.

Our model takes on average 15 turns (an average 105 seconds of gameplay) to converge, which is quick enough for our game, allowing players to discover the gameplay during few minutes starting from the easy condition. It is to note that each turn is relatively quick, taking less than 10 seconds. As we estimate a

probability, we need to be able to gather player failures/successes. To have the model converge as quickly as possible, it is thus important to be able to split the gameplay into multiple short challenges, as explained in [3].

During the first steps of δ-logit, we rely on the +/-δ algorithm. Of course, it is impossible for us to predict the difficulty of the game without having explored the player's abilities a little bit. We tuned the +/-δ so that it starts with a low difficulty level and slowly raises the difficulty (or lower it when the player fails). This is consistent with self-efficacy theories stating that failure, when a subject discovers a new task, can dampen their motivation [4]. However, +/-δ might be configured to start from any difficulty level, for instance from a difficulty level chosen by the player. As we designed a very easy and very hard difficulty setting, we could interpolate between them to provide the player with a starting easy, medium and hard difficulty setting. However, when starting from an easy setting, we quickly explore the easy difficulty levels, gaining quickly more information about the player's abilities with low θ values than if we started from a medium level and adapted toward p(fail) = 0.5.

As long as a game can be expressed as challenges that the player repeatedly tries to achieve and that these challenges are driven by a set of variables that have a monotonic impact on this failure probability, our model could be used to drive these challenges' difficulty. Of course, such challenges can be harder to identify in more complex games. In an open world AAA game like *The Legend of Zelda: Breath of the Wild*, when the player finds a new object, does it only change the difficulty of the current challenge (e.g. for a slightly more powerful weapon), or does it change the gameplay so much that a new challenge is to be defined (e.g. when using a bow instead of a sword) [9]? And if the player rides a horse while using their bow, is it a new challenge or an extension of the bow one? However, as our model uses very few samples, we think that δ-logit could be used to drive these challenges difficulty, when properly identified. We may indeed postulate that few games propose original challenges that the player can't try more than 15 times.

8 Conclusion

In this paper, we propose a DDA algorithm that starts with no data and then uses as few data points as possible gathered from a single player. Many models have been proposed to dynamically adjust the difficulty of a game, but none of them really addresses the problem of using very few data from one player while targeting any failure probability.

Our model starts with a +/-δ algorithm that drives difficulty towards a 0.5 failure probability and then uses logistic regression as soon as possible to follow a specific difficulty curve, described in terms of failure probabilities, with correct accuracy. In our example, a tank shooting game, the model takes on average 105 seconds to switch from +/-δ to logistic regression and targets difficulties of 0.2, 0.5 and 0.7, with actual difficulties of 0.14, 0.55 and 0.74. The model's failure prediction accuracy was 0.82 (σ = 0.06). We evaluate our model in a

realistic and challenging setting: shooting is a widely used game mechanic, we adapt enemy's AI number and behavior, and follow a difficulty curve providing a learning phase, a low difficulty plateau and difficulty peaks at levels that were almost never sampled before.

Of course, we discuss that our approach has drawbacks. Having only one metavariable must have an impact on accuracy. Also, our model is continuous and discontinuities in gameplay variables might not be correctly modeled.

We think that our model might be very useful for the design of many games. Even more, we think that by considering any game as a collection of various challenges [3], one may use multiple instances of our model to adapt more complex gameplays.

9 Acknowledgement

This research is part of the *Programme d'investissement d'avenir E-FRAN* project *DysApp*, conducted with *Caisse des Dépôts* and supported by the French Government.

References

1. Allart, T., Levieux, G., Pierfitte, M., Guilloux, A., Natkin, S.: Difficulty influence on motivation over time in video games using survival analysis. In: Proceedings of the 12th International Conference on the Foundations of Digital Games. p. 2 (2017)
2. Andrade, G., Ramalho, G., Santana, H., Corruble, V.: Extending reinforcement learning to provide dynamic game balancing. In: Proceedings of the Workshop on Reasoning, Representation, and Learning in Computer Games, 19th International Joint Conference on Artificial Intelligence (IJCAI). pp. 7–12 (2005)
3. Aponte, M.V., Levieux, G., Natkin, S.: Measuring the level of difficulty in single player video games. Entertainment Computing **2**(4), 205–213 (2011)
4. Bandura, A.: Self-efficacy: toward a unifying theory of behavioral change. Psychological review **84**(2), 191 (1977)
5. Concato, J., Peduzzi, P., Holford, T.R., Feinstein, A.R.: Importance of events per independent variable in proportional hazards analysis i. background, goals, and general strategy. Journal of clinical epidemiology **48**(12), 1495–1501 (1995)
6. Constant, T., Levieux, G.: Dynamic difficulty adjustment impact on players' confidence. In: Proceedings of the 2019 CHI Conference on Human Factors in Computing Systems. pp. 463:1–463:12. CHI '19 (2019)
7. Constant, T., Levieux, G., Buendia, A., Natkin, S.: From objective to subjective difficulty evaluation in video games. In: IFIP Conference on Human-Computer Interaction. pp. 107–127. Springer (2017)
8. Demediuk, S., Tamassia, M., Raffe, W.L., Zambetta, F., Li, X., Mueller, F.: Monte carlo tree search based algorithms for dynamic difficulty adjustment. In: 2017 IEEE Conference on Computational Intelligence and Games (CIG). pp. 53–59 (Aug 2017)
9. Fujibayashi, H., Aonuma, E., Toda, A., Takizawa, S.: The legend of zelda: Breath of the wild. Game [Nintendo Switch].(3 March 2017) (2017)

10. Gavin, A.: Making crash bandicoot part 6 (2011), https://all-things-andy-gavin.com/2011/02/07/making-crash-bandicoot-part-6, accessed: 2018-09-06
11. Hao, Y., He, S., Wang, J., Liu, X., Huang, W., et al.: Dynamic difficulty adjustment of game ai by mcts for the game pac-man. In: Natural Computation (ICNC), 2010 Sixth International Conference on. vol. 8, pp. 3918–3922. IEEE (2010)
12. Hocine, N., Gouaïch, A.: Therapeutic games' difficulty adaptation: An approach based on player's ability and motivation. In: Computer Games (CGAMES), 2011 16th International Conference on. pp. 257–261. IEEE (2011)
13. Ishihara, M., Ito, S., Ishii, R., Harada, T., Thawonmas, R.: Monte-carlo tree search for implementation of dynamic difficulty adjustment fighting game ais having believable behaviors. In: 2018 IEEE Conference on Computational Intelligence and Games (CIG). pp. 1–8. IEEE (2018)
14. Klimmt, C., Blake, C., Hefner, D., Vorderer, P., Roth, C.: Player performance, satisfaction, and video game enjoyment. In: ICEC. pp. 1–12 (2009)
15. Lazzaro, N.: Why we play games: Four keys to more emotion without story (2004)
16. Malone, T.W.: Heuristics for designing enjoyable user interfaces: Lessons from computer games. In: Proceedings of the 1982 conference on Human factors in computing systems. pp. 63–68. ACM (1982)
17. McCaffrey, J.: Test run - coding logistic regression with newton-raphson (2012), https://msdn.microsoft.com/en-us/magazine/jj618304.aspx, accessed: 2018-09-19
18. Nakamura, J., Csikszentmihalyi, M.: The concept of flow. In: Flow and the foundations of positive psychology, pp. 239–263. Springer (2014)
19. Olesen, J.K., Yannakakis, G.N., Hallam, J.: Real-time challenge balance in an rts game using rtneat. In: Computational Intelligence and Games, 2008. CIG'08. IEEE Symposium On. pp. 87–94. IEEE (2008)
20. Ryan, R.M., Rigby, C.S., Przybylski, A.: The motivational pull of video games: A self-determination theory approach. Motivation and emotion **30**(4), 344–360 (2006)
21. Spronck, P., Sprinkhuizen-Kuyper, I., Postma, E.: Difficulty scaling of game ai. In: Proceedings of the 5th International Conference on Intelligent Games and Simulation (GAME-ON 2004). pp. 33–37 (2004)
22. Sweetser, P., Wyeth, P.: Gameflow: a model for evaluating player enjoyment in games. Computers in Entertainment (CIE) **3**(3), 3–3 (2005)
23. Unity: Tanks tutorial (2015), https://unity3d.com/fr/learn/tutorials/s/tanks-tutorial, accessed: 2018-09-19
24. Yasuyuki, O., Katsuhisa, S.: Racing game program and video game device (2003), https://patents.google.com/patent/US7278913, accessed: 2018-09-18
25. Zook, A., Riedl, M.O.: A temporal data-driven player model for dynamic difficulty adjustment. In: AIIDE (2012)