Procedural Level Balancing in Runner Games

Rubem José Vasconcelos de Medeiros
Nars Vera Studio

Tácio Filipe Vasconcelos de Medeiros
Nars Vera Studio

Abstract

Balancing a game is a long process and relies mainly on subjective feedback from human testers and selective interpretation from game developers. As a first step for completely automate game balancing, we propose a methodology to algorithmically choose features and calibrate their parameters for the procedural level generation of a simple runner game based on testers feedback. This methodology is used in a 30 seconds game demo with survey and each playthrough is recorded and fed to a reinforcement learning algorithm. We show that the average fun grade steadily grows, proving the effectiveness of the proposed method. The collected data can be further analysed for insights on new features and other major changes.

Keywords: Runner, Game Balancing, Game Flow, Machine Learning

Author's Contact:

kukamed@gmail.com
taciofvmed@gmail.com

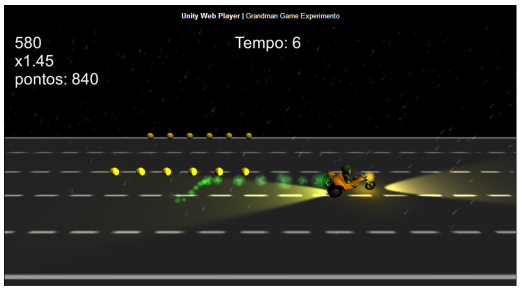


Figure 1: *Level with 5 lanes road*

1 Introduction

What makes games fun? The nature of fun experiences in games has been the topic of both speculation and more systematic inquiry for some time. There are a number of theories from psychology and game studies focusing on the experiences of playing a game. One pioneer work is Csikszentmihalyis concept of flow [Csikszentmihalyi 1991]. When in the state of flow, a human is performing a task such that he is fully concentrated on the task and loses his sense of self but has a sense of full control. In order to achieve the flow experience, the challenge of the task must be balanced accordingly. Sweetser and Wyeth[Sweetser and Wyeth 2005] adapted Csikszentmihalyis concept (originally generalist) to games. They proposed GameFlow, a measure of entertainment based on eight dimensions: concentration, challenge, player skills, control, clear goals, feedback, immersion and social interaction. For this paper, we created a basic endless runner game and added features to enhance or hinder control, challenge or concentration.

Endless runner game is a genre of platform game in which the player character is continuously moving forward through an endless procedurally generated world. The simplicity of gameplay makes it appealing for mobile platforms. Example of this genre are Canabalt (2009), Temple Run (2011) and JetPack Joyride (2011). Game developers tend to balance those kind of game in favor of casual gameplay. The wide variety of games in this genre have created a large population of runner gamers (potential testers) and its simplicity makes it easy to create new features, even to indie developers. But balancing the new features involves playtest.

Game balancing is the adjustment of game rules for a particular goal, generally either to make the possible choices equally fair, mostly used in competitive games; to adapt difficulty to player level and skill, which is known as Dynamic Difficulty Adjustment (DDA) when the tuning happens automatically during gameplay [Goetschalckx et al. 2010]; or simply to add, remove and improve features to calibrate the challenges and rewards for maximum enjoyment of most players in design time. Our goal is the latter, although the presented algorithm could be adapted for DDA. Automation of game balancing is widely used on various game genres [Marks and Hom 2007] [Yannakakis and Hallam 2009] [Houlette 2003].

Playtesting is meant to provide information about many aspects of a game, especially how immersive and entertaining it can be, in order to improve its desirability before official release. Various methods for obtaining consumer feedback have been applied to games with varying success. Some methods, such as traditional usability testing, can help to identify issues that prevent users from experiencing the fun of a game. Longer surveys and larger sample population improve the amount of information taken from each playtesting, but the typically small sample sizes, particularly for indie developers, prohibit generalization to a wider population.

In this paper, we propose a way to automatically rank all combinations of features according to how fun that level is, and by doing so, choose the combination of features with the best flow for our runner game. After each 30 second playthrough, the player is asked to rate how fun and how difficult was that level on a scale from 1 to 5.

The longer the test goes, the better the ranking of the feature combinations gets and after the playtesting is over, the developers just have to choose the top ones and discard the other level configurations.

2 Experiment Game Features Definition

There is a 3 or 5 lanes road and a character riding a tricycle is driving by. The player can input UP or DOWN commands to move between the lanes and LEFT or RIGHT to accelerate and decelerate thus moving closer the the left or right edge of the screen.

Several difficulties may happen to make a simple road trip dangerous or rewarding. They are features conceived based on GameFlow dimensions:

1. **Dificulty Features**

   Shooting drone: Comes from the right of the screen, one per wave

   Spikey drone: Comes from the left of the screen

   Double of Spikey drone: Two spikey drone per wave

   Double of Shooting drone: Two shooting drones per wave

   Half the distance between Spikey Drones: Time between shooting drones waves is halved

   Half the distance between Shooting drones: Time between spikey drones waves is halved

2. **Imersion Features**

   Darkness: The road is only lit by the vehicles front and rear lights

   Wind: The character is continuously and slowly pulled to a random direction

3. **Control Feature**

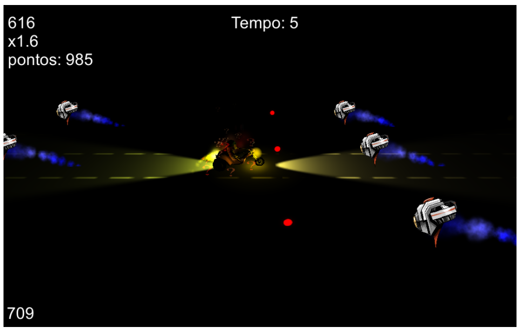


Figure 2: *Level with many active features: Darkness, Inverted controls, Shooting drone and 5 lanes*

Inverted controls: Up is Down and Left is Right, confusing the player

4. **Concentration Feature**

   More lanes: There are 5 instead of 3 lanes

Coins also appear on the road to be collected as reward and luring the player to a more dangerous path. One game ends after 30 seconds or when the character is knocked out by the drones.The distance traveled and number of coins picked up are recorded and the score is calculated based on both.

$Score = Distance \times (1 + Coins \times 0.025)$

When the game ends, a small survey form is shown, asking for the players rate on Difficulty and Fun and a free text comment area. Players are not interviewed but are asked to fill a form, minimizing the interviewing effects [Mandryk et al. 2006]. Also, the highscore is printed.

3 Fast Reinforcement Learning with Hamming Distance

Reinforcement learning[Sutton and Barto 1998] is an area of machine learning inspired by behaviorist psychology. It allows machines and software agents to automatically determine the ideal behaviour within a specific context in order to maximize a cumulative reward. A reinforcement signal is required as feedback for the agent to learn its behaviour.

In our setup, an agent acts on choosing a combination of features (state) for a playthrough. A reward is the feedback from the player at the end of each playthrough (sample). We propose and test two learning algorithms: a basic algorithm where only probabilities of each state are updated and a fast learning algorithm where information of the reward is propagated based on hamming distance of features.

3.1 Basic algorithm (experiment A)

Each level generated is a combination of the 10 binary features described in section 2. Thus, a total of 1024 different possible levels exist. Every time someone plays the game, one of those levels is randomly chosen and in the end, the outcome is recorded to a remote server. The new recorded playthrough will update the odds of that same level happening again. If the player rated that level as not fun, it will be randomly selected less often and if it was rated as very fun, the level will happen more often.

Formally, we have an initial uniform probability distribution of states. $C_i =$ state i

$P_{T=0}(C_i) = \frac{1}{2^{10}} \; \forall i$

A sample $P_{T=t}$ is defined by a pair of components: state and a rescaled fun rate between -1 and 1 (signal) (C_i, S).

An empirical learning rate α is set to 0.003.

And the learning algorithm is as follows:

$P_{T=t}(C_i) = \frac{P_{T=t-1}(C_i)+\alpha \times S}{\sum_{i=0}^{2^{10}} P_{T=t}(C_i)}$

But doing this way, the learning pace is very slow because only one state is significantly updated for each sample and a game with only 10 parameters for the level generation would require many thousand samples and signals to make the distribution converge.

3.2 Fast learning algorithm (experiment B)

Since one state is a combination of features, its easy to measure the similarity or distance between states. The more features they have in common, greater is their similarity.

This is the Hamming Distance[Vijay et al. 2012]. A Hamming Distance of 1 means that only one feature from all 10 have a different value.

$HammDist(C_i, C_j)$ =Count of $(C_i[k] \neq C_i[k])$ for every k in set of features.

With the help of this similarity function, the information gathered by one sample may be used to update the state actually played and also other states which share most of the features.

But the greater the distance between levels, less informative is the score. To reflect this intuition, the learning rate α is corrected by $2^{-HammDist}$

The new algorithm is as follows:

For j from 1 to 1024: $dist = HammDist(C_i, C_j)$
$P_{T=t}(C_j) = \frac{P_{T=t-1}(C_j)+(\alpha/2^{dist})\times S}{\sum_{i=0}^{2^{10}} P_{T=t}(C_j)}$

4 Experimental Setup

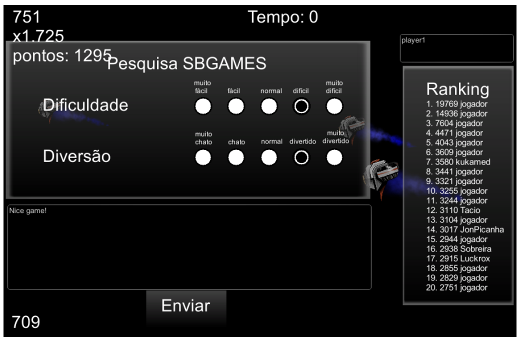
4.1 Server/Game

The game is a client / server application. The Client is the actual game and was developed in Unity3D 4 and all components programmed in C#

The Server is a web server that listens to HTTP request and answers in JSON format with the requested data. It was developed with Python 2.6 + Flask and MongoDB for data persistence, and hosted on Openshift, Red Hats PaaS (Platform as a Service) cloud environment.

The workflow of one gameplay follows these steps:

1. Player enters game URL on Web Browser and game loads.
2. Client requests level configuration id and experiment tag from server.
3. Server randomly draws tags A or B for this playthrough (50% chance for either).
4. Server randomly select one level configuration with odds of each configuration according to latest probability distribution update.
5. Server fetches top 20 highscore from all past gameplays.
6. Server responds to Client with those 3 informations.
7. Client receives configuration id and generates level accordingly.
8. Player play the game for 30 seconds or character dies.
9. Survey is rendered on Client, filled by player and submitted to Server.
10. Server records survey results and other gameplay metrics on database.
11. Server updates probability distribution using the simple or fast algorithm depending on which experiment tag was initially drawn and records new distribution on database.



751 Tempo: 0

x1.725

pontos: 1295

player1

Pesquisa SBGAMES

Dificuldade: muito fácil / fácil / normal / [✓ difícil] / muito difícil

Diversão: muito chato / chato / normal / [✓ divertido] / muito divertido

Nice game!

Enviar

709

Ranking
1. 19769 jogador
2. 14936 jogador
3. 7604 jogador
4. 4471 jogador
5. 4043 jogador
6. 3609 jogador
7. 3580 kukamed
8. 3441 jogador
9. 3321 jogador
10. 3255 jogador
11. 3244 jogador
12. 3110 Tacio
13. 3104 jogador
14. 3017 JonPicanha
15. 2944 jogador
16. 2938 Sobreira
17. 2915 Luckrox
18. 2855 jogador
19. 2829 jogador
20. 2751 jogador

Figure 3: *Survey form*

4.2 Gameplay Metrics

Besides the fun rate which is used for the learning algorithm, a number of other metrics are sent to be recorded on the database for future analysis.

- Score
- Fun rate (filled by player)
- Difficulty rate (filled by player)
- Free comments (filled by player)
- Commands list: a list of the commands issued by the player and the moment it was pressed.
- Life change list: a list of the moments when the character lost life points and the remaining life

4.3 Call for Testing

Once all the system was ready, the only thing missing was human testers so that the needed data could be gathered. We used social network websites to promote the experiment mainly on local gamers group and on indie game developers groups. As a result we had 417 playthroughs in 30 hours, 218 for experiment A and 199 for experiment B.

5 Results

5.1 Evaluating Learning Algorithms

We compare the performance of two algothim described as experiment A and B. Figure 4 shows the moving average (20 samples lag) of the fun rate of experiments A and B for every playtrough (x axis). Figure 4 also shows plot of a linear Function Fit to both data. Note that Experiment A has ans angular coeficient of the fitted function negative and Experiment B has a positive one, indicating that the fast learning algorithm indeed enhances learning of fun games. As number of playthroughs progress, only fast learning score rises. Altogh not shown on the graph, tt is expected that even the basic algorithm with enough samples, would show an increase in average score.

5.2 Evaluating GameFlow Features

Another result can be seen on figures 5 and 6. From the final probability distribution of features, we isolated each feature from the others, estimating the marginal distribution of features. The result is that features with low probability of occuring are less fun features. Note that feature 1 in general, but when combined with feature 2 is a little fun.

Those results are very informative of what feature works and what doesn't. This way, this figures can be directly used by game designers to improve their games. Another informative graph can be seen on figure 7. The same fast learning algorithm was applied to the probability distribution, but this time updating the answers about dificulty. The figure shows that in fact the dificulty features

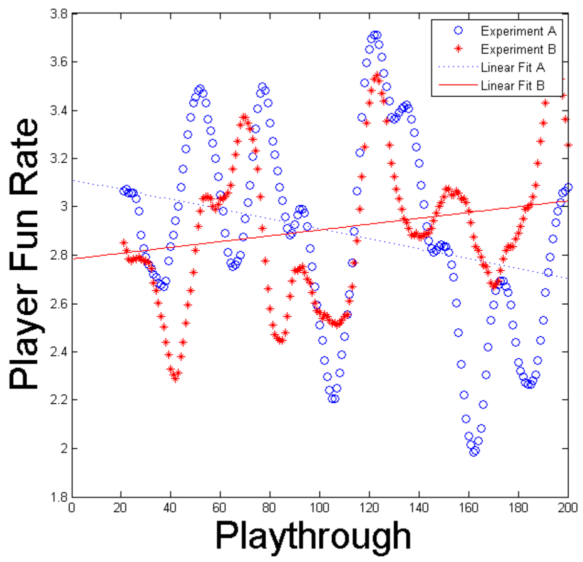


Figure 4: *Evolution of Player Fun Rate on Experiments*

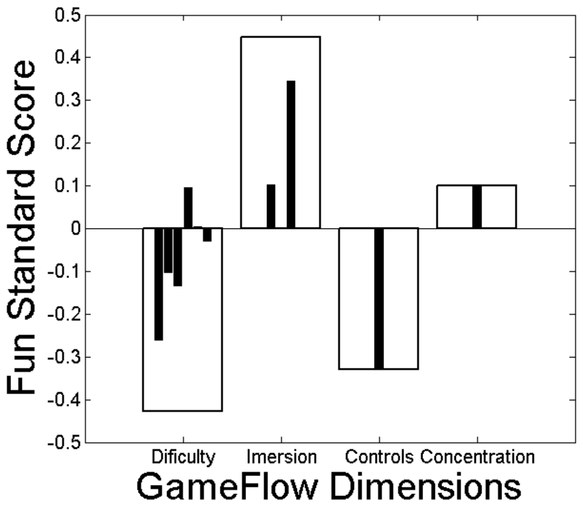


Figure 5: *GameFlow Dimensions x Stantard Score grouped by GameFlow Dimensions*

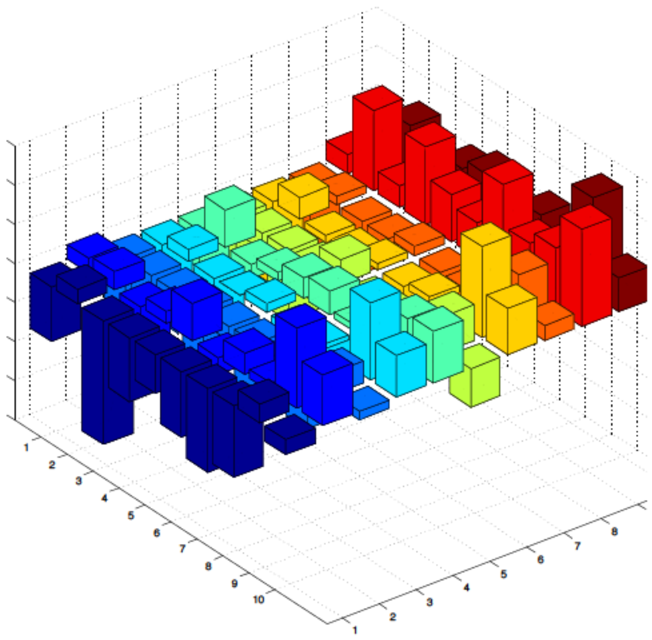


Figure 6: *Combined Features Fun Measure, 1-10 correspond to each feature described on section 2*

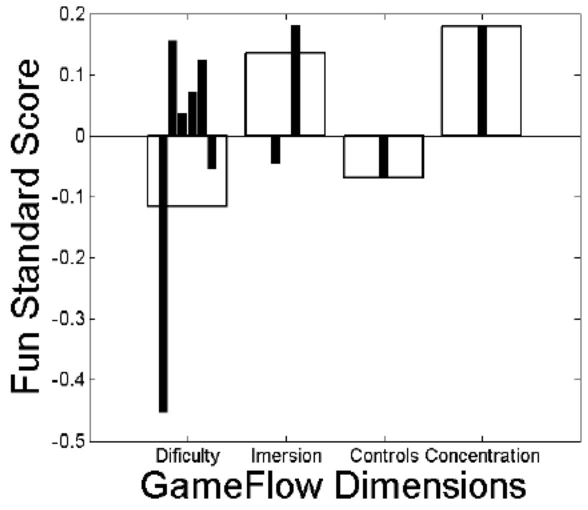


Figure 7: *GameFlow Dimensions x Difficulty Stantard Score grouped by GameFlow Dimensions*

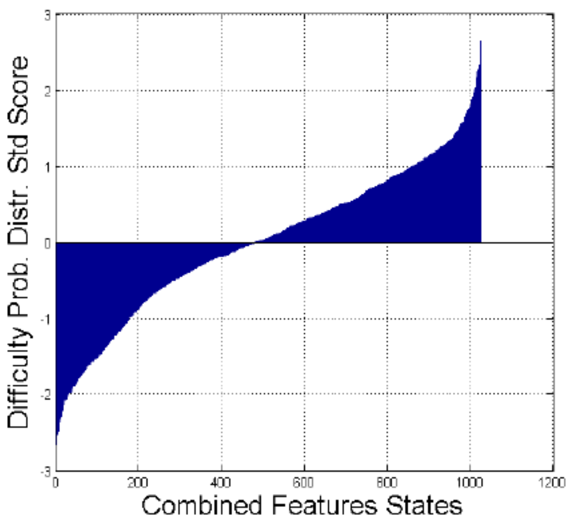


Figure 8: *Probability Distribution of all states after learning from difficulty survey answer*

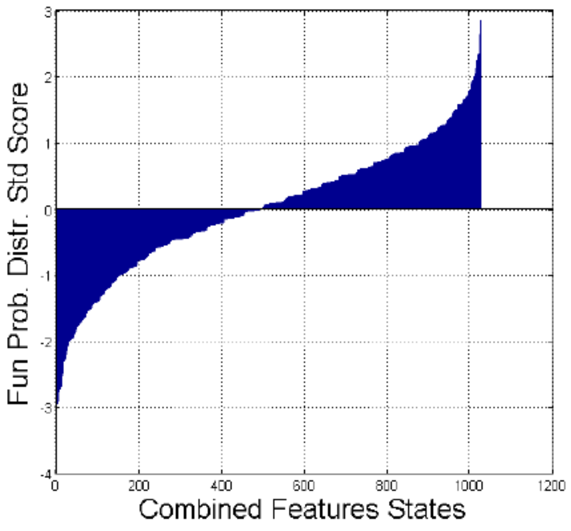


Figure 9: *Probability Distribution of all states after learning from difficulty survey answer*

are difficult, except, unexpectedly, the first one. This is another very informative graph game designer can take to create the level design.

Observe that from the same data of standard score of fun and dificulty, a procedurally generated level can be created. Features that don't add fun can be discarted and procedural level can take an increasing dificulty parameter. We can see on figures 8 and 9 the average score of the probability distributions. Observe that about 200 states are above 1 standard deviation on both distributions. For a procedural level, game design define some threshold of acceptable fun and dificulty for each of his levels.

6 Conclusion

Having a nice game mechanic, visually attractive artwork and impressive soundtrack and effects is not nearly enough to make a game enjoyable. For it to happen, the challenges and rewards must be carefully planned and adjusted to maximize game flow. Feature selection and combination is an essencial part of a game design process. In this article, a proof of concept of a feature selection model for procedural level balancing on design time of tested for an endless running game. The results showed that a reinforcement learning algorithm boosted by hamming distance of features resulted on a faster learning curve and actually improved overall fun of playtesters on that particularly game. Our model learns combination of features that enhances fun and is capable of procedural generating fun levels even if no one played and rated that particular configuration of level parameters.

Although previous studies [Yannakakis and Hallam 2006] [Yannakakis and Hallam 2007] have shown that feature selection combined with preference learning contributes to the generation of more effective fun models for the player, our model selects specific boolean game design features that game designer wants to be tested. For a better presentation of experimental data, we have also introduced an analysis framework to help game developers better understand data collected in play testing and make better decisions on improvements as features to focus on and features to remove.

It is important to note the drawbacks of the model, the framework and the proof of concept. The effects on the learning pace by varying learning factor α, maximum Hamming Distance cut-off should be futher investigated. Also, the algorithms are designed to cope boolean features. Extensions are possible to cope with integer or real numbers. This is an open possibility for futher work. Another drawback fo the research was the limited dataset of the experiment. Futher analysis is needed with a greater number of playtesters and with various genres of game. The improvement of fun by the proposed model showed on the results may work diferently on diferrent genres of game. Naturally, a commercial released game have many more features and much more complex mechanics, nonetheless the same tools here proposed might be used, once perfected. Much can still be done. We have only scratched the surface on this subject.

Futher work will be the main objective of the reserch: to completely automate level generation calibration. For a completly automated system, the the survey step must be replaced with gameplay metrics. Game analytics of some combination of gameplay metrics would be translated to the normalized signal used in the learning algorithm, similarly to what was accomplished by [Pedersen et al. 2009]. This way, a number of other analysis can still be done with the gathered data and possibly added to the learning algorithm or affect game design in other ways.

References

CSIKSZENTMIHALYI, M. 1991. *Flow: The Psychology of Optimal Experience*. Harper Perennial, New York, NY, March.

GOETSCHALCKX, R., MISSURA, O., HOEY, J., AND GAERTNER, T., 2010. Games with dynamic difficulty adjustment using pomdps.

HOULETTE, R. 2003. *Player Modeling for Adaptive Games*. Charles River Media, Dec.

MANDRYK, R., INKPEN, K., AND CALVERT, T. 2006. Using psychophysiological techniques to measure user experience with entertainment technologies. 141–158.

MARKS, J., AND HOM, V. 2007. Automatic design of balanced board games. In *Proceedings of the Third Artificial Intelligence and Interactive Digital Entertainment Conference, June 6-8, 2007, Stanford, California, USA*, The AAAI Press, J. Schaeffer and M. Mateas, Eds., 25–30.

PEDERSEN, C., TOGELIUS, J., AND YANNAKAKIS, G. N. 2009. Modeling player experience in super mario bros. In *Proceedings of the 5th International Conference on Computational Intelligence and Games*, IEEE Press, Piscataway, NJ, USA, CIG'09, 132–139.

SUTTON, R., AND BARTO, A. 1998. *Reinforcement learning: An introduction*, vol. 116. Cambridge Univ Press.

SWEETSER, P., AND WYETH, P. 2005. Gameflow: A model for evaluating player enjoyment in games. *Comput. Entertain. 3*, 3 (July), 3–3.

VIJAY, R., MAHAJAN, P., AND KANDWAL, R. 2012. Hamming distance based clustering algorithm. *IJIRR 2*, 1, 11–20.

YANNAKAKIS, G. N., AND HALLAM, J. 2006. Towards capturing and enhancing entertainment in computer games. In *Proceedings of the Hellenic Conference on Artificial Intelligence*, 432–442.

YANNAKAKIS, G. N., AND HALLAM, J. 2007. Feature Selection for Capturing the Experience of Fun. In *AIIDE'07 Workshop on Optimizing Player Satisfaction*, AAAI Press.

YANNAKAKIS, G. N., AND HALLAM, J. 2009. Real-time game adaptation for optimizing player satisfaction. *IEEE Trans. Comput. Intellig. and AI in Games 1*, 2, 121–133.