

Incongruity-Based Adaptive Game Balancing

Giel van Lankveld, Pieter Spronck, H. Jaap van den Herik,
and Matthias Rauterberg

Tilburg centre for Creative Computing
Tilburg University, The Netherlands

{g.lankveld,p.spronck,h.j.vdnherik}@uvt.nl, g.w.m.rauterberg@tue.nl

Abstract. Commercial games possess various methods of game balancing. Each of them modifies the game’s entertainment value for players of different skill levels. This paper deals with one of them, viz. a way of automatically adapting a game’s balance which is based on the theory of incongruity. We tested our approach on a group of subjects, who played a game with three difficulty settings. The idea is to maintain a specific difference in incongruity automatically. We tested our idea extensively and may report that the results coincide with the theory of incongruity as far as positive incongruity is concerned. The main conclusion is that, owing to our automatically maintained balanced difficulty setting, we can avoid that a game becomes *boring* or *frustrating*.

1 Introduction

The main goal of many commercial computer games is to provide entertainment to the player. To support player-experienced entertainment, the gaming industry has invested substantial efforts in improving game attributes that contribute to this goal. In particular, it is remarked that attributes such as graphics, animation, and physics have seen a rapid increase in technical detail and accuracy over the past two decades. Although the attributes mentioned have an actual impact on the entertainment value of a game, they are still a relatively small factor in this respect. More important for the entertainment value are elements such as design, diversity, story, game balancing, and gameplay.

In this paper we deal with game balancing, which is defined as the adaptation of the game difficulty towards a player’s skill. More specifically, we focus on the relationship between a game’s complexity and a player’s abilities. In a well-balanced game, a player is challenged by the complexity of the game, but not to the extent that he¹ becomes frustrated. It is generally assumed that a balanced game has a higher entertainment value than one that is too easy or too hard [3]. If a game is considered to be balanced, then for the purpose of entertainment that game should aim at remaining balanced from start to end, even while the player learns to do better at handling specific game challenges. However all players are

¹ For the sake of brevity, we use ‘he’ and ‘his’ whenever ‘he or she’ and ‘his and her’ are meant.

different in their skills and learning abilities, therefore meeting this aim means adapting the game's complexity automatically to the observed player skill.

The incongruity theory [7] states that every context (such as a game) has a level of complexity. To deal adequately with a context humans have internal models of the varying levels of complexity. Incongruity is defined as is the difference between the complexity of a context and the complexity of the internal human model of the context. When the incongruity is too large, the human loses interest in the context, i.e., he loses interest in the game. This means that the entertainment value of the game decreases.

In this paper we apply the incongruity theory to game balancing. To do so, we measure the incongruity while a game is being played, and adapt the challenge of the game automatically to maintain the incongruity at a constant level. This level should be one that the human player experiences as entertaining, regardless of skills and capabilities. For our investigations we present a new game called *Glove*, developed in our laboratory. *Glove* contains a novel approach to keep incongruity at a desired level. According to the incongruity theory, a large incongruity is less entertaining than a small incongruity.

The outline of the paper is as follows. In Section 2 we provide background information on game balancing and incongruity theory. Section 3 describes *Glove*, including our approach to tactical balancing. The experimental setup is given in Section 4. Our results are presented in Section 5 and discussed in Section 6. Finally, in Section 7 we provide our conclusions and look at future work.

2 Background

In this section, we discuss the existing work currently done on game balancing (Subsection 2.1), and provide details on the incongruity theory (Subsection 2.2).

2.1 Game Balancing

Commercial games usually provide a manual way of setting the difficulty at the start of a new game. This method sometimes results in an inadequate difficulty setting, e.g., if the player makes an unsuitable choice or if his skills improve during play [12]. For example, the commercial game *Max Payne* features what game developers refer to as dynamic difficulty adjustment (DDA). The DDA monitors the amount of damage received, and adjusts the player's auto-aim assistance and the strength of the enemies [13]. DDA is easily recognised by the players, which implies that it breaks the flow of the game [2]. Therefore, DDA may cause players to accept extra damage to prohibit an increase of the difficulty.

Recently, computer science researchers have started to investigate methods that measure the entertainment value of a game [1,10,11], and adapt the game automatically in order to increase entertainment [4,8]. Yannakakis [9] describes two ways of optimising player enjoyment, namely implicit and explicit optimisation. In implicit optimisation, machine learning techniques such as reinforcement learning, genetic algorithms, probabilistic models, and dynamic scripting,

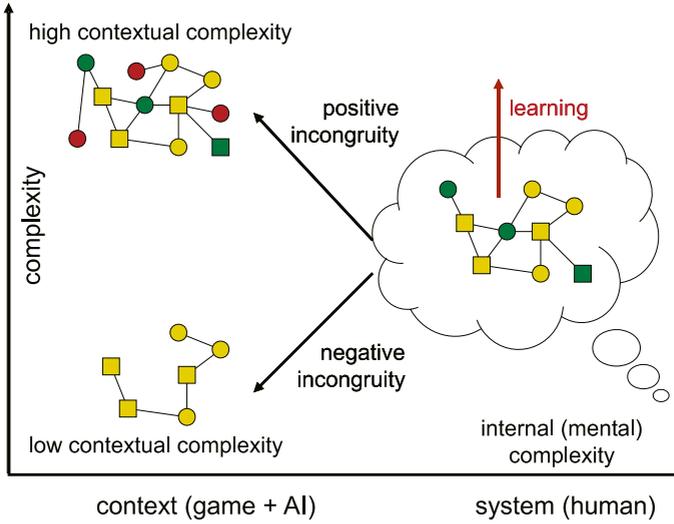


Fig. 1. Schematic representation of incongruity

are used for optimisation. He also describes user modelling techniques as explained in interactive narration. In explicit optimisation he describes adaptive learning mechanisms that optimise what he calls *user-verified ad-hoc entertainment*. Two well known techniques are: (1) neuro-evolution mechanisms and (2) player modelling through Bayesian learning.

2.2 Incongruity Theory

People continuously form mental models about the world in which they live. These models allow them to estimate how the world is going to react to different types of interaction and how the world will change over time. Incongruity theory attempts to explain the emotions that arise during interaction between these models and the world. For a proper understanding of this paper we introduce below the terminology used in the literature on incongruity.

Incongruity theory uses the term ‘*context complexity*’ to describe the complexity of the world or a part of the world. The term ‘*system complexity*’ is meant to describe the complexity of a mental model that a person has of the world. Complexity can be *high*, *intermediate* or *low* for both the context and the system. The term ‘incongruity’ is used to describe the difference in complexity between the system and the context. When system complexity is higher than context complexity we speak of negative incongruity. When context complexity is higher than system complexity we speak of positive incongruity. A schematic visualisation of these concepts can be found in figure 1.

According to the incongruity theory, the difference between the system complexity and the context complexity give rise to three types of emotion: boredom, frustration, and pleasure. *Boredom* is a feeling of reduced interest, it arises with

high negative incongruity. *Frustration* is a feeling of annoyance or anger, it arises with high positive incongruity. According to Rauterberg [7], in situations of large negative incongruity, people start to look for new a stimulation. *Pleasure* is a feeling of entertainment, it arises when context complexity is roughly equal to or slightly higher than system complexity.

Learning is stimulated in situations where incongruity is positive: it raises the system complexity. Thus, learning can bring players from a large positive incongruity via low or no incongruity to negative incongruity.

In a game context we can easily identify the concepts of incongruity. First the difficulty of the game is equivalent to the context complexity. Then, the player's skill at the game is equal to the system complexity. If the player's skill is too low in order to cope with the current game difficulty setting, we may speak of negative incongruity, which is expected to result in frustration. If the player's skill is too high in comparison to the game difficulty, we may speak of positive incongruity, which is expected to result in boredom. When the player's skill matches the game difficulty, there is low or no incongruity and a feeling of pleasure or, otherwise stated, entertainment is expected to occur.

Because system complexity is part of the human mind there is usually no direct way to measure system complexity in games. One possibility is to measure the complexity of the player's behaviour in a game and infer the system complexity from that. Rauterberg [7] states that low system complexity will lead to the player's behaviour being largely determined by heuristics, while expert players, with high system complexity, use other, more straightforward methods.

3 Glove

In our experiments we use a new game called *Glove*. It is derived from the classic game *Gauntlet*. In this section we describe the *Glove* game world (Subsection 3.1), and the balancing mechanism built into the game (Subsection 3.2).

3.1 Game World

Glove (depicted in Figure 2) is a two-player turn-based game between a (human) player and a computer system. The player controls a knight. The knight is placed in a world that consists of 2000 cells. The world is 10 cells high, and 200 cells wide. Each cell is either passable (grass), or impassable (water or mountain). The knight occupies one cell. The world also contains enemies, each of which also occupies one cell only.

The knight starts at the leftmost end of the world, and his goal is to reach the rightmost end of the world. The game ends in victory for the player when the knight reaches the goal. It ends in a defeat for the player when the knight dies before reaching the goal (i.e., the computer system wins). A knight dies when he has no health left. Health is measured in hitpoints, of which the knight has 100 at the start of the game. As soon as the number of hitpoints reaches zero, the knight dies. On each turn, the player can let the knight perform one of two actions: he can either *move*, or *attack*.



Fig. 2. Glove

When moving, the knight leaves the cell that he occupied, and moves over to any of the eight adjacent cells. Each move costs the knight 0.5 hitpoint. This means that if he moves steadily and unobstructed through the world, he has exactly enough health to win the game.

When attacking, the knight executes an attack to one of the eight adjacent cells. He can either attack with his sword that he always carries with him, or with a rock, which he may have picked up in the game world (by moving over it). The knight can carry at most one rock at a time. The difference between attacking with the sword and a rock, is that the rock actually attacks two cells, namely the cell which is attacked, and the one directly behind it (according to the movement of the knight). There are three types of enemies that can attack the cell containing the knight (dragon, ninja, and witch, see below). If an attacked cell contains an enemy, the enemy dies. Upon dying, the enemy leaves behind a health token, which the knight may pick up (by moving over it). This grants the knight 5 hitpoints (they are added to the knight's current amount up to a maximum of 100).

There are three different kinds of enemies in the world, a number of which are spawned at regular intervals. Each time that an enemy attacks and hits the knight, the knight becomes damaged and loses 5 hitpoints. The three types of enemies differ in their behaviour and their abilities. The three enemy types are the following.

1. **Dragon:** The dragon approaches the knight using a shortest-path method. The dragon is visible. When the dragon is next to the knight, he attacks the cell in which the knight resides. Arguably, the dragon's behaviour is the easiest behaviour of all three to deal with by the player.
2. **Ninja:** The ninja has the same basic behaviour as the dragon, but he has an additional ability: he can become invisible. So, he will use this ability when he is within a certain range of the knight, and will remain invisible for a certain number of turns. As soon as he attacks the knight, he will become visible again. The ninja's behaviour is reasonably predictable, even when invisible, for players who possess a good mental model of the game.
3. **Witch:** The witch approaches the knight in the same way as the other two enemy types, but stops when she is within a distance of three cells of the knight. At that point, she will start to throw one fireball per turn in the direction of the knight. Fireballs move at a speed of one cell per turn. When there are few enemies on the screen, fireballs can usually be avoided easily. However, the knight must approach the witch to be able to attack her, at which time avoidance may be difficult. Damage statistics produced in the experiment lead us to conclude that the witch is the hardest enemy to deal with.

3.2 Balancing Glove

Interaction with the game world is limited to move and attack actions, and there is little diversity in the challenges that the player faces. This is done on purpose. The aim of *Glove* is to provide the player with an entertaining experience, by *only* varying the number and types of enemies with which the knight is confronted.

The basic game has three difficulty levels, named *easy*, *balanced*, and *hard*. While it is possible to add more difficulty levels, for the present experiment these three were deemed sufficient. When the difficulty is easy, the game aims at having the knight win the game with about 50% of his health remaining. When the difficulty is hard, the game aims at having the knight lose the game when he has progressed through about 50% of the game world. When difficulty is balanced, the game aims at having the knight experience a narrow victory or a narrow defeat. The game accomplishes the result envisaged by controlling the number and types of spawned enemies. In this way the easy and hard levels try to keep the incongruity stable and high, while the balanced level tries to keep incongruity stable and at a minimum.

For each enemy type, the game retains the *average damage* in hitpoints that the enemy type involved has in relation to the knight. The number of hitpoints can be positive or negative (or zero). If the number is positive, it means that the knight on average loses health due to an encounter with this enemy type. If the number is negative, it means that the knight on average gains health due to an encounter with this enemy type. Gaining health is possible because the enemies leave behind health tokens upon dying, and it is certainly possible to kill an enemy without the enemy being able to damage the knight.

The net result of the spawning procedure is that between 2 and N enemies are spawned, N being a number that depends on the difficulty setting and the

current progress of the knight. The spawned enemy types are such that, according to past experience with the enemies, the knight is expected to lose or gain the amount of health needed to achieve the goal of the difficulty setting, regardless of the player's skills.

The player can only see a part of the game world. He can see a 10 by 10 area centered around the knight. Enemies are spawned just outside the knight's vision, every 10 cells that the knight has progressed towards the right end of the world. The number and types of spawned enemies are determined by the game based on (1) the difficulty level, (2) the knight's progress, (3) the knight's health, and (4) the average damage numbers. To spawn enemies, the following algorithm is used (in pseudo-code).

```

procedure spawnEnemies
begin
    needed_health := getNeededHealth( getCurrentProgress() ) +
        getModifier( getDifficulty() );
    health_to_lose := getCurrentHealth() - needed_health;
    expected_health_loss := 0;
    spawned := 0;

    while
        (spawned < 2) or
        ((expected_health_loss < health_to_lose) and
        (spawned < getMaxSpawn(
            getLastSpawned()+1, getDifficulty() )))
    do
    begin
        enemyType := spawnRandomEnemy( health_to_lose );
        health_to_lose := health_to_lose -
            getAverageDamage( enemyType );
        spawned := spawned + 1;
    end;
end;

```

This algorithm uses the current game state and the settings of the game to determine the amount and type of enemies to spawn. It consists of three parts. The first part considers (1) the progress that the knight has already made through the game world and (2) the health that he is expected to have. Part two deals with spawning enemies until the combined enemy difficulty is at the level required by the game settings. Finally, part three is used for smoothing the increase of difficulty. It ensures that the amount of spawned enemies increases gradually rather than instantly. What follows now is a detailed explanation of eleven subroutines used in the algorithm.

`getCurrentProgress()` returns a percentage that expresses how far the knight has progressed through the game world. `getNeededHealth()`, uses the knight's current progress as a parameter, and returns the number of hitpoints that the knight needs when traversing the remaining part of the game world,

if unobstructed by enemies. `getDifficulty()` returns the difficulty level (easy, balanced, or hard), and `getModifier()`, has the difficulty level as a parameter; it returns a number that is 50 for the easy difficulty level, 5 for balanced, and -50 for hard. `getCurrentHealth()` returns the current health of the knight. The result of the first two lines of the algorithm is that `health_to_lose` contains the number of hitpoints that the knight should lose for the game to reach the goal determined by the difficulty level. This number can be negative, which indicates that the knight should actually gain health.

`spawnRandomEnemy()` spawns an enemy. This function has `health_to_lose` as a parameter, by which it determines (1) whether it should spawn enemies that are likely to gain the knight some health, or (2) whether it should spawn enemies that cause the knight to lose health. To avoid the algorithm becoming into an endless loop, when the function should make the knight gain health, it will always allow dragons to be spawned; moreover, when it should make the knight lose health, it will always allow ninjas and witches to be spawned. We note that with more enemies, it is harder to avoid damage; even if the player has reached a skill level in which he manages to gain health from all enemy types, he will consider the game harder if he will be surrounded by more of them.

`getLastSpawned()` returns the number of enemies that were spawned at the last time. `getMaxSpawn()` returns the maximum number that can be spawned. It has two parameters, the first is a maximum that cannot be exceeded, and the second is the difficulty level, which is used to determine a maximum number: 5 for easy, 7 for balanced, and 9 for hard. Finally, `getAverageDamage()` returns the average damage done by the enemy type that is used as the parameter.

4 Experimental Setup

To test the effect of our game-balancing approach, and to investigate whether *boredom* and *frustration* are indeed associated with a decreased entertainment value and with increased incongruity, we requested a number of human test subjects to play *Glove*. The experimental setup was as follows. Each human subject played the game four times. The first time was a training run, in which the player should experience the game controls. In the training run, at each spawn point the same three enemies are spawned, namely one of each type. The player was allowed to interrupt the play whenever he wanted, to start the actual experiment.

In the actual experiment, the subject had to play the game three times, viz. once with an easy difficulty setting, once with a balanced one, and once with a hard one. The order in which the difficulty settings were presented to the subject was varied, each possible order being tested about an equal number of times. The subject was not aware of the difficulty setting of his current game. A digital questionnaire was presented to the subject after each game.

The questionnaire contained a total of 26 items. The items were all in Dutch because our subjects were all Dutch native speakers. The items fell into five categories, namely *boredom*, *frustration*, *pleasure*, *concentration*, and *curiosity*. *Boredom*, *frustration* and *pleasure* are the experiences expected to occur during

play according to the incongruity theory. *Concentration* and *curiosity* items were added to assess whether the subjects were fully focused on the game, rather than other thoughts or things in their surroundings. Each item was administered using a seven-point Likert scale [6]. The seven points range from “does not apply to me at all” to “completely applies to me”. The English translation of the Dutch questionnaire can be found in the appendix.

In our preliminary experiments 24 subjects participated. The subjects’ age ranged from 16 to 31 years. The subjects were selected from family, friends, and the student population. All were Dutch native speakers. None of them had prior knowledge of the game before playing. The subjects had a varying background, and varying experiences with computers and games. The exact subject background did not matter for this experiment, since the game balances itself automatically to the skills of the player.

5 Results

On the questionnaires, scores ranged from zero (0) to six (6) on a Likert scale. So, assuming a continuous scale then the average would be 3. For each subject and then for each category, the average of the answers to the questions belonging to the category was calculated. Subsequently, for each of the difficulty settings, the means of these averages over all test subjects were calculated. The means are presented in Figure 3.

For a statistical analysis of the results, we had to remove one subject from the pool because of an input error, leaving 23 subjects ($N = 23$). To compare the means of the variables, an ANOVA is sufficient. However, (1) we had multiple conditions (easy, balanced, hard) for the prediction of the five variables and (2) we applied all three test conditions to each subject, therefore a repeated-measures MANOVA test was needed. Straightforwardly, using an ANOVA test would have ignored possible interaction and repetition effects.

The repeated-measures MANOVA multivariate test produced significant effects ($P < 0.01$). Thereafter a post-hoc univariate analysis and contrast analysis were performed in order to examine (1) the differences between the five measured variables and (2) the differences of the difficulty on these variables.

We found that the effect of order was not significant ($P > 0.05$). A subsequent analysis was performed to see if there were significant effects of experience with computer games. This effect was also not significant ($P > 0.05$).

Next, we tested the effect of the difficulty setting on each of the five categories of the questionnaires. We found no significant results for the categories *boredom*, *concentration*, and *curiosity* ($P > 0.05$ for all of them). However, we *did* find significant effects for the categories *frustration* ($P < 0.01$) and *pleasure* ($P < 0.05$).

Contrasts showed that for the category *frustration*, the differences between easy and balanced, and between balanced and hard were both significant ($P < 0.01$). In particular, we found that *Glove* is significantly more frustrating for a balanced difficulty compared to an easy difficulty, and significantly more frustrating for a hard difficulty compared to a balanced difficulty. The estimated

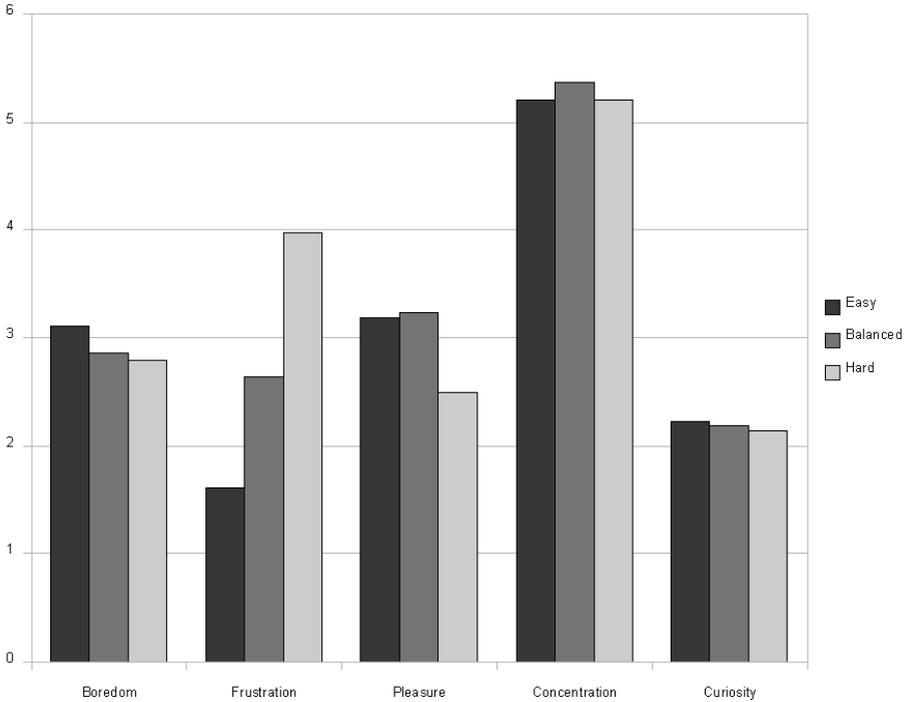


Fig. 3. Means for each category per difficulty setting

marginal means for the category *frustration* were 1.64 for easy difficulty, 2.67 for balanced difficulty, and 4.01 for hard difficulty.

For the category *pleasure* we found significant effects for the difference between balanced and hard difficulty ($P < 0.05$). In particular, we found that *Glove* provides significantly more pleasure for a balanced difficulty than for a hard difficulty. We did not find a significant effect for the difference between an easy and a balanced difficulty. The estimated marginal means for the category *pleasure* were 3.24 for easy difficulty, 3.25 for balanced difficulty, and 2.50 for hard difficulty.

Our tests show that our approach to game balancing, based on incongruity, can influence both the frustration level and the entertainment level of a game. The results reproduce the incongruity theory findings that a high positive incongruity is correlated to frustration, and that, at least for *Glove*, a balanced difficulty setting is more entertaining than a hard difficulty setting.

6 Discussion

The results of our experiments reproduce incongruity theory predictions in part rather well. The frustration effect follows the expectations of incongruity theory, while boredom (which should be significantly higher for easy difficulty) does not

follow the expectations. The entertainment effect is also according to expectations, at least for the balanced and hard difficulty settings.

It is likely that entertainment would also be as expected for easy difficulty, if easy difficulty was considered to be boring by the test subjects. Therefore it is interesting to examine why the easy difficulty setting was not found to be boring. We did not actually investigate this issue, but offer two possible explanations. First, incongruity theory was originally applied to (relatively old) web interfaces [7], and the increased visual and functional interactivity of our game, even in its simplicity, might cause a sufficiently high increase in complexity to be interesting in all modes of difficulty. Second, it is definitely possible that our easy difficulty setting is still sufficiently complex to create positive incongruity. In future work, we will examine this possibility by introducing a ‘very easy’ difficulty setting, in which the knight is confronted with just a handful of enemies, and does not lose any health moving.

We believe that our method of adaptive game balancing overcomes some of the problems of which commercial games suffer with their method of difficulty scaling, as our balanced difficulty setting manages to avoid that the game becomes boring or frustrating.

7 Conclusions and Future Work

In this paper we examined (1) the relationship between game balancing and incongruity, and (2) how adaptive game balancing can be used to increase the entertainment value of a game. For our game *Glove*, we found that *frustration* increases with difficulty, while the *entertainment* remains roughly the same for easy and balanced difficulty, but drops for hard difficulty. So, we may conclude that our results coincide with the incongruity theory as far as positive incongruity is concerned. Furthermore, we may conclude that our approach to adaptive game balancing is suitable to maintain a game’s entertainment value by keeping incongruity at a balanced value.

The pool of test subjects used for our experiments was relatively small, yet the results on which we base our conclusions are highly significant. Still, we could not discover significant results for all the categories which we examined. Significant results for the remaining categories might be obtained by a higher number of test subjects. Therefore, in future work, we will (1) continue our experiments with a bigger subject pool, (2) introduce a ‘very easy’ difficulty setting, to examine whether the boredom expectations of incongruity theory can also be confirmed, (3) implement our adaptive game balancing approach in an actual commercial game, and (4) test its effect on the entertainment value. Such an experiment is expected to demonstrate the applicability of our approach to commercial game developers, and may have an impact on how games are constructed in the near future.

Acknowledgements

This research was supported by the “Knowledge in Modelling” project of the Dutch National Police Force (KLPD).

References

1. Beume, N., Danielsiek, H., Eichhorn, C., Naujoks, B., Preuss, M., Stiller, K., Wessing, S.: Measuring Flow as Concept for Detecting Game Fun in the Pac-Man Game. In: Proc. 2008 Congress on Evolutionary Computation (CEC 2008) within Fifth IEEE World Congress on Computational Intelligence (WCCI 2008). IEEE, Los Alamitos (2008)
2. Csikszentmihalyi, M., Csikszentmihalyi, I.: Introduction to Part IV in Optimal Experience: Psychological Studies of Flow in Consciousness. Cambridge University Press, Cambridge (1988)
3. Charles, D., Black, M.: Dynamic Player Modelling: A Framework for Player-Centric Games. In: Mehdi, Q., Gough, N.E., Natkin, S. (eds.) Computer Games: Artificial Intelligence, Design and Education, pp. 29–35. University of Wolverhampton, Wolverhampton (2004)
4. Hunicke, R., Chapman, V.: AI for Dynamic Difficulty Adjustment in Games. In: Proceedings of the Challenges in Game AI Workshop, 19th Nineteenth National Conference on Artificial Intelligence. AAAI 2004 (2004)
5. Iida, H., Takeshita, N., Yoshimura, J.: A Metric for Entertainment of Boardgames: Its Implication for Evolution of Chess Variants. In: Nakatsu, R., Hoshino, J. (eds.) Entertainment Computing: Technologies and Applications, pp. 659–672. Kluwer Academic Publishers, Boston (2002)
6. Likert, R.: A Technique for the Measurement of Attitudes. Archives of Psychology, New York (1932)
7. Rauterberg, M.: About a framework for information and information processing of learning systems. In: Falkenberg, E., Hesse, W., Olive, A. (eds.) Information System Concepts—Towards a consolidation of views (IFIP Working Group 8.1), pp. 54–69. Chapman and Hall, London (1995)
8. 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)
9. Yannakakis, G.N.: How to Model and Augment Player Satisfaction: A Review. In: Proceedings of the 1st Workshop on Child, Computer and Interaction. ICMI 2008, Chania, Crete, October 2008. ACM Press, New York (2008)
10. Yannakakis, G.N., Hallam, J.: Towards Capturing and Enhancing Entertainment in Computer Games. In: Antoniou, G., Potamias, G., Spyropoulos, C., Plexousakis, D. (eds.) SETN 2006. LNCS (LNAI), vol. 3955, pp. 432–442. Springer, Heidelberg (2006)
11. Yannakakis, G.N., Hallam, J.: Modeling and Augmenting Game Entertainment Through Challenge and Curiosity. International Journal of Artificial Intelligence Tools 16(6), 981–999 (2007)
12. <http://www.casualgamedesign.com/?p=39>
13. http://www.gameontology.org/index.php/Dynamic_Difficulty_Adjustment#Max_Payne

A Questionnaire (English Translation)

For the sake of clarity, the questions were translated from the original Dutch version into English.

1. In my opinion the game was user friendly
2. I am interested in how the game works
3. I had fun while playing the game
4. I want to know more about the game
5. I can easily concentrate on what I need to do during the game
6. The time passed quickly
7. I got distracted
8. I felt involved in the task
9. The game frustrated me
10. The game made me curious
11. I felt challenged
12. The time passed slowly
13. The task fascinated me
14. I was thinking about other things during play
15. I found the game to be fun
16. I was bored
17. The game was tedious
18. I would like to ask questions about the game
19. I thought the game was hard
20. I was alert during the game
21. I was day dreaming during the game
22. I want to play the game again
23. I understood what I was supposed to do in the game
24. The game was easy
25. I feel I was not doing well during the game
26. I was annoyed during play