BinG: A Framework for Dynamic Game Balancing using Provenance

Felipe Figueira, Lucas Nascimento, Jose da Silva Junior
Computing Institute
Instituto Federal do Rio de Janeiro (IFRJ)
Rio de Janeiro, Brazil
felchado@gmail.com, nascimentol@outlook.com, and
jose.junior@ifrj.edu.br

Troy Kohwalter, Leonardo Murta, Esteban Clua
Computing Institute
Universidade Federal Fluminense (UFF)
Niteroi, Brazil
tkohwalter,leomurta,esteban@.ic.uff.br

***Abstract*—Among different reasons that can lead a player to stop playing a game, frustration due to challenges that do not fit to their skills may be one of the most critical. Besides that, the players' skills improve along the time, and the previously selected difficulty level may become inappropriate due to the player's improvement. This can result on decreasing the motivation for the player retention, as they could get bored because of the easy challenges or frustrated due to the harsh difficulty. In this paper, we propose a new approach based on gathered provenance data for dynamically tuning the game's challenge according to the current player skills. To do so, we developed BinG, a framework responsible for collecting and processing data provenance, allowing for the development of different balancing models to be used externally by the game. BinG uses the concept of logical programming to deliver facts and rules observed during a game session, allowing querying over the database to understand what happened. Additionally, we conducted a study using a game developed in-house and a dynamic balancing model customized to that game through BinG. This study was performed with five volunteers, who played the game using the default balancing and our dynamic balancing. Through this experiment, we showed a performance discrepancy reduction of almost 50% for the most skilled player in relation to the less skillful player when using dynamic balancing.**

I. Introduction

Video games are gradually becoming more complex and expensive to be developed. The world wide video game market is a billionaire industry that reached US$ 101.1 billion in 2016 with a forecast of US$ 128.5 billion in 2019 according to the Global Gamer Market [1]. The game industry must continually improve the game quality in order to attract and maintain the interest of players to generate income and finance the development of new titles. Several elements can improve quality such as the story, artwork, genre, physics, artificial intelligence, and so on. Besides that, publishers normally try to reach out for the largest public possible, inserting different types of gameplay elements to provide more content, which translates as an increase of gameplay hours. However, according to Gregory [1], the success of a game is intimately related to how fun and interesting a game is. In this case, rapidly identifying the fun factor of a game is a top priority for this marketplace [2]. Additionally, according to Davis et al. [3], the first moments of the game are critical, as if the player finds it compelling and interesting, it is more likely that he or she will continue playing.

One of the key factors for players' retention is the game balance. A video game is a formal closed system [4] explicitly designed for resources manipulation between the virtual world and the player. This manipulation is known as the game economy [5]. The availability of these resources in a game is called balancing. Balancing is a critical factor for game success [6], being challenging and requiring a great effort to be reached. However, balance is normally difficult to perceive as there are not always clean-cut mathematical methods to balance a game. Moreover, difficulty comes from the necessity of balancing elements in the game that function completely different from each other, such as in a *fighting* game, where each character has different abilities. A poorly balanced game allows for the emergence of dominant strategies, limiting other lesser strategies.

During development, game balancing is normally performed in a develop-test-develop approach [3], where parameters of the game are changed and tested. Another common practice is the beta test approach [3], which is used when the game is near its release state. The beta testing approach consists in collecting feedback from a certain number of participants (beta testers) who play the game in order to find technical issues, bugs, or gameplay unbalance in the game. Normally, these beta-testers are volunteers who play an early pre-release build of the game, being a valuable source of information about technical issues in the game. This approach allows little control about the gameplay experience or the environment as they can play the game at home. Besides that, this type of analysis may be subjective and dependent on the complexity of the game, being difficult to reproduce the same decisions and actions that lead to a desired effect. Both approaches focus on statically tuning the game to improve its overall quality.

Normally, the player expects the game's difficulty to be closer to their skill while playing the game. The player skills tend to improve with time by getting used to the game's mechanics and the response of the commands. Thus, the game's difficulty should also increase as well. On the other hand, the player gets frustrated or bored when the game becomes too difficult or too easy, respectively [4]. A solution for such issues includes dynamically adjusting the game's difficulty based on the current players skills, which implies in monitoring players' actions and their performance throughout the game. Such

[1]https://newzoo.com/insights/articles/the-global-games-market-will-reach-108-9-billion-in-2017-with-mobile-taking-42/

approaches are referred in the literature as ”dynamic game balancing (DGB)” and ”dynamic difficult adjustment (DDA)” [6]. Several approaches for performing such dynamic balancing have been proposed, including prediction models [6]–[8], Evolutionary Fuzzy Cognitive Maps [9], neuro-evolution [10], challenge based prediction [11], and game scenario adaption [12]. However, most of these approaches are statically defined and difficult to adapt to other games' genres and mechanics. Besides that, they do not take into account the cause and effect relationships that can emerge during a gameplay.

In a previous paper [13], we presented how provenance can be used for understanding the relationships among elements in a game. Previously, Jacob et al. [14] proposed a non intrusive way for collecting data provenance from a video stream at the cost of inferring data from processing a video stream. On the other hand, Kohwalter et al. [15] introduced the PinG framework for capturing information during a game session and maps it to provenance terms, which have been used for analyze an education game [16]. However, neither approaches automatically evaluates the player's performance during a game session for dynamic balancing.

The goal of this paper is to present BinG (Balance in Games), a framework to collect provenance information from a game session to understand the underlying relationships found among the different elements presented in the game (such as items and enemies). This provenance information is then used to produce new parameters for balancing the game according to the current player's skills. We choose to use PinG [15] for capturing provenance data in order to avoid precision problems from the non-intrusive approach. BinG allows the usage of different balancing model, which can be activated when desired by the developer in order to produce new balancing factors to be adjusted in the game. Furthermore, BinG is implemented as an external module, being called through an interface inside the game. This implies in fewer modifications in the game in order to use it. BinG uses the tracked provenance data to produce a database of facts and rules according to a defined schema, allowing the usage of logical and imperative programming paradigm as well. We developed a game and a dynamic balancing model to validate BinG. We conducted a study with five participants that played the game using the default game balancing and our dynamic balancing. At the end, we analyzed the results in order for finding evidences about the effectiveness of dynamic balancing in relation to default balancing. We found that employing BinG for dynamic balancing reduced the average discrepancy in 41.33% and 47.18% for time lived and score achieved in the game, respectively, among participants in relation to default balancing.

This paper is organized as follows: Section II presents the related works, while Section III presents the underlying BinG framework and its components. Section IV presents a use case for using BinG. Section V presents the materials and methods used for the experiment and Section VI presents our evaluation. Section VII discuss some threats to the experiment. Finally, Section VIII concludes this paper and points to future works.

II. Related Work

Data collection from player's session and its analysis is crucial for performing dynamic difficulty adjustment. Different approaches were developed aiming at collecting and analyzing game flux. However, for the best we know, there is no approach similar to BinG in the literature for performing a fair comparison.

Chandler and Noriega [17] present a framework used to evaluate, based on the past collected data, knowledge about success and failure in modern games. In that work, they claim that a game must be challenging but not too hard or too easy, suggesting an automatic difficulty level based on the user's skills. However, the aim of this framework is to help game designers to find pitfalls and errors during development.

The majority of DDA systems are based on prediction and intervention as the fundamental strategy. Hamlet [6] takes advantage of the flow model developed by Csikszentmihalyi [18], that defines the player's states in skill and challenge dimensions. According to this, they suggest that game challenge should match the current player's skill, increasing the preference of certain states in relation to others. Hamlet tries to predicts the player's state and adjust the game dynamically to prevent inventory shortfalls.

Hawkins et al. [7] use player's risk profile for predicting their performance inside a probabilistic DDA model. According to them, cautious players wait longer for more evidence, increasing the likelihood of success in detriment of game time.

Missura and Gärtner [8] use prediction in a probabilistic framework. They try to predict the desired difficulty by formulating the tasks in a game as an online learning problem on partially ordered tasks. All the approaches based on predictions does not considers the cause and effect relationships.

Tijs et al. [19] proposed performing dynamic difficult balancing by using the player's emotional stage during gameplay. The main drawback of this approach is breaking the flow state in the game [4], as the player needs to answer questions from time to time to adjust the game difficulty. Vasconcelos de Medeiros and Vasconcelo de Medeiros [20], on the other hand, used offline collected player's data for procedural level design balancing. Unfortunately, due to their static approach, the increase on the player's skills was not taken into account.

When considering artificial intelligence based approach, Prez et al. [9] use fuzzy and probabilistic causal relationships through cognitive maps to dynamically increase or decrease the spawn rate of obstacles in the game as well as power ups. Olesen et al. [10] use Neuro Evolution of Augmenting Topologies in real time for adjusting game challenge. Using a developed game, they analyzed the factors that contribute in the level of difficulty and developed a model based on these factors. However, the main drawback of these approaches is that they consider the actual game state for performing such difficulty adjustments, without considering past history states during the session.

III. BinG Framework

A well balanced game is one of the crucial factors for its success. Achieving balance in a game involves understanding how each of the game elements impacts in the player's performance, and adjusting its mechanics accordingly. BinG is a framework that employs provenance to represent the cause and effect relationship among the pieces of information in the game, allowing the detection of possible balancing issues and reacting to them dynamically. According to Freire et al. [21] provenance data is the historical documentation of an object or the documentation of the process life cycle of digital objects. In the context of the game, Kohwalter et al. [15] proposed a mapping of provenance elements, where entities are inanimate game objects, such as weapons and obstacles, while agents are characters such as NPCs and players, and activities are actions or interactions performed by actors among them or with entities, such as attacking or using an healing item. Figure 1 presents this relationships among those elements.

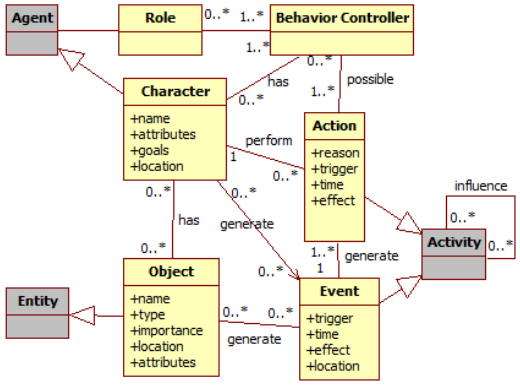


Figure 1. Provenance in Games [15] UML class diagram. Gray classes represent generic provenance classes.

Performing dynamic balancing requires understanding how the relationships among elements in a game change across time based on changes to the player's skills and modifying the behavior of these elements appropriately. For this, BinG uses provenance data collected from the game, producing knowledge over this information that can be used during the process of dynamic balancing. BinG is implemented as an external module that is called by the game as necessary, being it at the end or during a game session, without jeopardizing the gameplay. As a result, BinG provides new balancing parameters according to the balancing model used. Conceiving BinG as an external module reduces the coupling with the game. BinG's infrastructure can be see in Figure 2.

The component that interacts with the game is the **Importer**. When desired, this component is invoked by the game, sending a provenance data to be processed. The collected provenance data is performed by the game using the PinG framework, storing the data into a XML file. The Importer component is responsible for reading the provenance file and recreating the graph of vertices and relationships among these vertices, according to the model presented in Figure 1.

The **Data Analysis** component is called after processing the vertices and relationships. This is one of the most important components in BinG and is responsible for transforming gathered data to use it for the generation of new balancing parameters for the elements in the game. The concept of knowledge involves understanding how the behavior of game elements and their relationships changes across time and impacts in the performance of the player. In this case, we choose to represent this knowledge as facts and rules in a logical programming paradigm, specifically using Prolog. Prolog is a general-purpose declarative programming language strongly based on first order logic. We believe that using a logical programming paradigm is the most straightforward way for expressing the reasoning process. The Data Analysis component converts PinG provenance data into facts in Prolog automatically. However, rules are defined by the game design through an external schema [2].

The knowledge produced can be used by a **Balancing Model** to analyze facts and rules, reacting accordingly. At this point, the developer can use this information for developing the dynamic balancing module through adjustments in the balancing factors of all elements in the game before the game release. This module can be written using Python, Prolog, or both. In addition, all the Prolog results are available for Python, allowing for developing complex dynamic balancing models. The possibility of mixing procedural and logical paradigms, and extracting the best of each one, is one of the key factors in BinG.

After the new balancing parameters are processed, they are returned to the game. Due to the BinG's low coupling, the usage of dynamic balancing modules from other similar games becomes possible.

IV. Use Case

In this section, we present an instantiation of BinG for an in house game developed called *MorphWing*. Considering the mechanics of the *MorphWing* game, we describe a dynamic balancing model and how it changes the *MorphWing* game elements.

A. Game

We projected and developed a 2D shooting game prototype, called *MorphWing*, using Unity3D [22] game engine in order to evaluate our proposed approach for dynamic difficulty balancing. It is important to state that it is just a use case. Any other game type could be used by just replacing the balancing model. The main goal of the player in *MorphWing* is to stay alive for the longest time possible. They do so by avoiding getting hit by enemies and preventing a countdown timer to hit zero. Destroying enemies increases the player's score and time limit. Items are spawned during the game, which can positively or negatively affect the player. A screenshot of the game is presented in Figure 3. In *MorphWing*, the player loses health points when touching an enemy or when being hit by

[2]The format and parameters of this schema can be found at https://github.com/josericardojr/BinG.

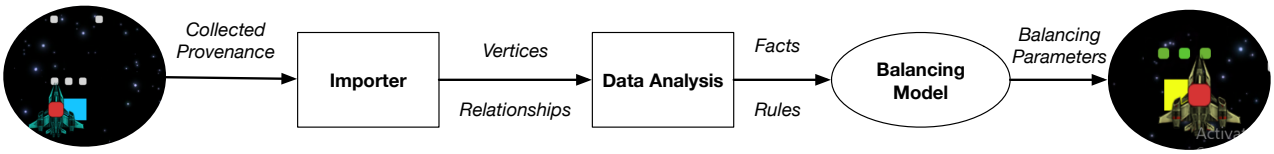


Figure 2. BinG architecture.

their projectiles. In both cases, the player gets invincible for a brief moment, without taking damage.

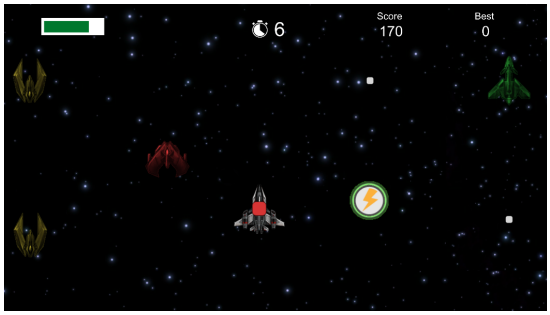


Figure 3. *MorphWing* screenshot, presenting a spawned item (Speed Up) and some types of enemies.

MorphWing has four different types of enemies and a maximum of four enemies can be on the player's screen at the same time. The characteristics and behavior of each type of enemy are as follow:

- **Straight:** moves on a straight direction until it reaches the other side of the screen, disappearing afterwards.
- **Chaser:** chases the player by moving towards him, causing damage until destroyed.
- **Boomerang:** after appearing from a random corner of the screen, moves straight for a moment, stops, shoots 4 bullets in the up, down, left and right directions, and moves back toward its spawn point, disappearing when it reaches its spawn point again. As just one enemy of this type is on screen, we called time to prepare the interval for this enemy to shoot. Higher interval means the player has more chance to hit it.
- **Round Shooter:** after appearing from a random corner of the screen, moves straight for a moment, stops, then shoots bullets clockwise in 8 directions on a circular pattern, starting by the top.

In addition to the enemies, *MorphWing* also spawn items that can cause negative or positive effects to the player. These items, after spawned, disappear after some time if they are not collected by the player. In total, up to three items can be presented on the screen simultaneously. The items available are **Damage Up** and **Damage Down**, that temporarily increases or decreases the players damage output, respectively, and **Speed Up** and **Speed Down** that temporarily increases or decreases the players movement speed, respectively.

B. Balancing Model

Considering the mechanics of the *MorphWing* game, a dynamic balancing model should consider changing enemies' parameters such as their speed and damage inflicted to the player. However, it must be made in a way that the user does not perceive it. Considering the behavior of each type of enemy in the game, Table I presents the characteristics that can be changed by our dynamic balancing model during the game. *MorphWing* uses the default balancing values (Def. column) when the model is not used. When using the dynamic balance model, the range for each characteristic is described by Min and Max columns.

TABLE I. ENEMIES CHARACTERISTICS THAT ARE CHANGED ALONG THE GAME, WITH THEIR DEFAULT (DEF.), MINIMUM (MIN.), AND MAXIMUM (MAX.) VALUES.

Enemy	Characteristic	Def.	Min.	Max.
Straigh	*Velocity*	1.70	1.02	4.59
Chaser	*Health Point*	5.00	3.60	8.50
	Velocity	0.70	0.51	1.19
Boomerang	*Time to Prepare*	0.80	0.28	2.04
	Bullet Speed	5.00	1.75	12.75
Round	*Bullet Cooldown*	0.70	0.60	1.33
	Bullet Speed	5.00	2.25	9.50

In our model, the characteristics presented in Table I changes based on a unique balance factor for each enemy, through a linear function described as Equation 1,

$$F(bf) = m + (n - m) * bf \quad (1)$$

where m and n are the minimum and maximum values presented in Table I, respectively, and bf is the balancing factor (ranging from 0.0 to 1.0). The dynamic balancing model is responsible for choosing the best bf according to the current player's skill.

We employ Algorithm 1 to find a balancing factor for each enemy type. In our model, each type of enemy possesses a corresponding numerical identifier (*enemyId*). In order to retrieve the performance of each enemy type against the player, we first retrieve the number of times that each type of enemy hit the player (function $enemyHits()$). Additionally, we retrieve the number of times each enemy tried to attack the player (function $enemyAttacks()$). With both values, we can calculate the *hit rate* of that enemy and understand how effective this type of enemy can cause damage to the player. BinG uses this *hit rate* value to increase or decrease the difficulty of this enemy for the next session. We use the function $balanceFactor()$ to count for the intrinsic mechanics each enemy presents in order to weight this *hit rate* value. We

can increase the difficulty of certain types of enemies faster than others, because they are inherently easier. The same is applied to the opposite situation: in the case of a specific enemy type not causing damage to the player, BinG uses the ($missHit()$) function that slightly increases its balancing factor to make it more challenging. Finally, it is important to state that a calculated balance for each type of enemy is applied for all its balancing characteristics presented in Table I.

Algorithm 1: Finding an enemy balancing factor.

```
Data: enemyId
Result: enemyBalance
if enemyAttacks(enemyID) > 0 then
    eHits = enemyHits(enemyID);
    eAttacks = enemyAttacks(enemyId);
    hitRate = 1 - (eHits / eAttacks);
    enemyBalance = hitRate * balanceFactor(enemyID);
else
    enemyBalance = missHit(enemyID) *
     balanceFactor(enemyID);
end
```

As presented in Section III, data being interchangeable between a logical (Prolog) and procedural (Python) programming language is one of the keys of BinG, allowing it to extract the best of each paradigm depending on the desired task. In Algorithm 1 both $enemyAttacks()$ and $enemyHits()$ functions use logical programming (Prolog) for retrieving these values. As explained in Section III, relationships (activities) performed by each element in the game are automatically represented as facts. Besides, rules evaluated automatically for each vertex by BinG can be created during the dynamic balancing model definition through a schema. In our case, some of the facts and rules used are described in Listing 1. In this case, some rules describe the context of each vertex in the data provenance in Listing 1. Additionally, we defined two rules to be used during dynamic balancing. In fact, the function $enemyAttacks(enemyID)$ (implemented in Python) in Algorithm 1 uses the rule $enemyAttack(ENEMYID)$ (implemented in Prolog), by counting the total of vertex returned by this rule.

```
% Facts from relationships
hit(vertex1, vertex2).
hit(vertex3, vertex1).
hit(vertex4, vertex1).
⋮

% Automatic Rules
vertexType(vertex3, straight).
vertexType(vertex2, chaser).
vertexType(vertex1, round).
vertexType(vertex4, straight).

% Manual Rules
beingHit(X, Y) :- hit(Y, X).
enemyAttack(ENEMYID) :- setof(Y,
    ↪ (vertexType(X, player),
    ↪ beingHit(X,Y),vertexType(Y,ENEMYID)),
    ↪ OUT).
⋮
```

Listing 1: Example of logical programming used by our dynamic balancing model.

Besides the harm that enemies can cause to the player, the types, amount, and distance of the items in relation to the player can contribute to the player's performance. If the item is spawned using a random position where the player is unable to reach until the item disappears, then it becomes useless. In addition, randomly selecting an item to be spawned can also be useful for the player depending on the actual state of the game. In order to avoid such problems, we also implemented the process for spawning items using the dynamic balancing model. The Algorithm 2 describes how we can perform such dynamic balancing for the items.

Algorithm 2: Item spawn through dynamic balancing.

```
Data: playerPosition
Result: item
foreach item ∈ ItemTypes do
    foreach enemy ∈ EnemyTypes do
        rate = hitRate(enemy);
        item.distance += rate * itemWeight(item.type,
         enem.typey);
    end
    item.probability = defaultChance(item.type) *
     item.distance ;
end
item = weightProbability(items);
```

The dynamic balancing for the items uses the items' distance to the player as well as the type of the item to be spawned. Each type of item spawns in a certain distance relative to the player's current position, considering a minimum (1 unit radius) and maximum (5.1 units radius) distance. The distance is defined by the relation that the spawned item had with helping the player overcome specific enemies (function $itemWeight()$), and the current difficulty values of the enemies (function $hitRate()$). If an enemy's difficulty is currently high and the spawned item helped dealing with this enemy, then the item will start appearing farther from the player to maintain the challenge against this type of enemy. For example, the function $itemWeight()$ for the item that increases the player's speed is higher for Chaser enemy, as it allows the player to move fast and evade this kind of

enemy. When using default balancing, the items are randomly spawned in any position on the game field.

In addition to the distance, each item type has its spawn frequency determined dynamically during the game. Items that helped the player against a certain type of enemy have a higher probability to be spawned as soon as the level of difficulty of the enemy increases. That means, when an item is going to be spawned, its type is calculated based on the actual level of difficulty of all kinds of enemies. So, the probability of spawning an item is represented by its default chance to be spawned weighted by the level of difficulty for all enemies. At the end, we calculate the item to be spawned (function $weightProbaility()$), leading to higher frequency for the ones with higher probability.

Finally, our dynamic balancing model also changes the effectiveness of the player's weapon shot according to Equation 2.

$$Dmg_p = \frac{h_p}{(bh_p * bf * \frac{1}{time}})} \quad (2)$$

, where h_p and bh_p represent the number of times the player hit and were hit by an enemy.

The damage that the player's weapon inflicts is changed along the sessions based on the amount of times their shots hit an enemy during the last session by the duration of the last game, in seconds. In conclusion, the effectiveness of player's weapon is changed according to their rate of hits over time based in the last session performance. The higher the rate, the lesser the damage, and vice-versa. The damage varies from 0.7 to 2 points of damage, being 1 point of damage the default value.

V. Materials and Methods

We are interested in knowing how the level of difficulty impacts in the player's performance during the game, considering the default and dynamic balancing. For this, we used the game and the model described in Section IV to evaluate how BinG can be used for performing dynamic balancing. Our evaluation is focused on answering the following research questions:

RQ1: How is the overall impact in the player's performance when using dynamic balancing?

RQ2: Does the balancing factor changes across sessions in terms of enemy's difficulty?

RQ3: Does adjusting the player's damage based on his/her current skills contributes to his/her performance?

RQ4: Does the usage of dynamic balancing for items contributes to the player's performance?

In order to answer these questions, we conducted an experiment with participants having different game abilities and skills to see their performance while using the game's default balancing and BinG's dynamic balancing. To ease the analysis, we chose to call BinG between sessions for performing dynamic balancing. The experiment was conducted with 5 participants (named P1-P5) who have never played *MorphWing* before. Table II shows the general demographic of the participants.

TABLE II. Participants' characteristics.

Participant	Classification	Play / Week	Gender
P1	Don't Know	<1 hour	M
P2	Median	<1 hour	M
P3	Don't Know	3 - 4 hours	M
P4	Hard Core	>7 hours	M
P5	Casual	<1 hour	F

Participants were instructed on the overall experiment setup to get their (informed) consent. The following step of the experiment involves giving instructions to each of the participants, explaining about the enemies in the game and their behavior, as well as the items and their effects. We also answered any questions that came out during this step. Subsequently, the player had the chance to play the game once in order to understand its mechanics. After this first session, we asked the participant to play the game six times using the default balancing, which were considered for the evaluation. After giving a break of two hours for letting the participant to rest, we asked the participant to play the game for seven times with the dynamic balancing activated. As the dynamic balancing occurs at the end of the session, the first session is used as a training session to initialize the model, being the last six sessions considered for the evaluation. We used six sessions in order to get more data samples without tiring so much the participant.

It is important to state that, in order to avoid bias, the participant was unaware of the balancing used during the sessions (default or dynamic balancing). Finally, we conducted an exit interview with each participant. Enemies and items spawn point, as well as the number of enemies by type, have been fixed across all the sessions during this experiment to avoid bias due to random enemy and spawn point item location. Their types and effects were also fixed.

VI. Results

Here we present our evaluations results structured around our research questions.

A. How is the overall impact in the player's performance when using dynamic balancing?

We choose to consider how long the participant lived as well as his/her score in order to verify the impact of dynamic balancing over the participant's performance. As six sessions are being considered during the experiment (for default and dynamic balancing), we calculated the average time and score for each participant as well as their standard deviation. The results are presented in Table III.

According to Table III, the **default balancing** presents a high variation for both time (column $\overline{Time}$) and score (column $\overline{Score}$) among the participants. For instance, P3 was the participant that survived the longest in the game (157.01 seconds), leading to the highest score (35,755.00 points).

On the other hand, P1 was the participant that survived the less (28.80 seconds), achieving the smallest score (4,299.17 points). The difference between times is 128.21 (157.01 - 28.80) seconds, while for the score is 31,465.83 (35,765.00 - 4,299.17) points. In fact, just two participants (P2 and P3) had an average time above the mean, while just P3 had the score above the mean. By looking at Table III, it is possible to state that P3 is the participant with the highest skill among the participants. In fact, when asked about the difficulty of the game for P3, he scored it as too easy.

On the other hand, when considering the usage of BinG for **dynamic balancing**, another behavior emerges. First of all, it is possible to see in Table III that the average time is almost half of the one resulted from the default balancing and score. Additionally, the difference between the highest and lowest values are 36.76 (62.97 - 26.21) seconds and 9,059.17 (13,214.17 - 4,155.00) points for time and score, respectively. P3 continues as the most skilled participant in the game but now it does not diverge too much from the others, as the difference from him to P4 (the second most skilled participant) was 19.74 seconds and 4,245.84 points for time and score, respectively. Based on this result, it is possible to observe that the game increased the level of difficulty for P3. In fact, when asked about the difficulty of the game using the dynamic balancing, P3 scored it as medium in terms of difficulty. Based on the premise stated in the beginning of this paper that players gets better across the time, our model fulfills the criteria of increasing the game difficult instead of make it more easier. The model can also facilitate the game, according to the developer desires, but it goes beyond the scope of this paper.

TABLE III. PARTICIPANTS' TIME LIVED AND SCORE PERFORMANCE FOR DEFAULT AND DYNAMIC BALANCING.

Participant	Default Balancing		Dynamic Balancing	
	Time	*Score*	*Time*	*Score*
P1	28.80	4,299.17	26.21	4,155.00
P2	69.46	14,118.33	35.21	6,871.67
P3	157.01	35,765.00	62.97	13,214.17
P4	48.52	13,030.83	43.23	8,968.33
P5	30.13	5,030.83	28.28	4,949.17
Average	**66.78**	**14,448.83**	**39.18**	**7,631.67**
Std. Dev.	**59.48**	**14,452.25**	**23.58**	**5,610.79**

In Figure 4, it is possible to see the answer of the participants about their perception of the difficulty for default and dynamic balancing (where 1 is too easy and 5 to hard). It is possible to see that neither of the participants found the game to be too easy after playing with the dynamic difficulty balancing. On the other hand, their perception of difficulty increased for P2, P3, and P4, being the same for P1 and P5. This is an indication of a possible tweak needed in our model in order to keep it at a medium difficulty. Finally, one interesting aspect to notice is that, according to Table II, P3 and P1 did not know his classification. Although, P3 was the most skilled participant, achieving the best time and score, possibly being classified as a hardcore player. In contrast, P1 was the worst for both time and score.

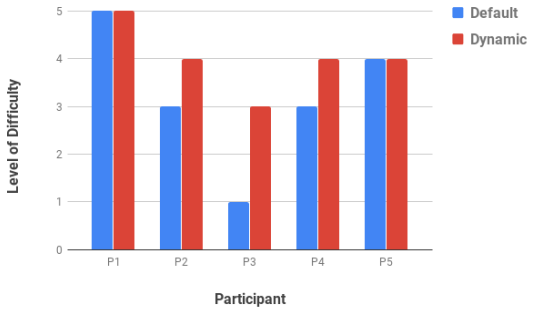


Figure 4. Perception of default and dynamic balancing difficulty by the participants.

B. RQ2: Does the balancing factor changes across sessions in terms of enemy's difficulty?

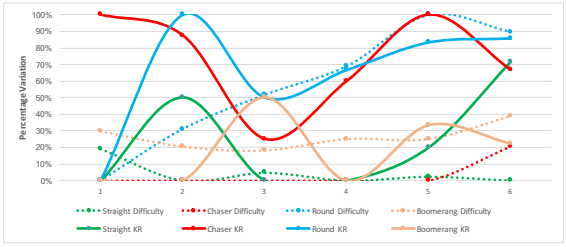
Besides investigating the time and score variation from default to dynamic balancing, it is also important to understand how this balancing factor varied across the sessions. Table IV presents the *Kill Rate (KR)* by each type of enemies for the default and the dynamic balancing, considering all sessions of a participant. The *KR* is defined as enemies killed per enemies spawned. It is possible to observe that the *Chaser* enemy is the one that had the highest kill rate among the participants. P3, for instance, reached almost 100% of kill rate. The dynamic model implemented for BinG could detect this unbalance and increased the difficulty for this type of enemy accordingly, changing the enemies' characteristics. In fact, it is also important to remember that, besides changing the enemy's characteristics parameters, the damage caused by the player's weapon is also slightly changed, as described in Section IV-B.

On the other hand, the opposite (i.e., enemies becoming easier) can also be observed on Table IV. When looking at P1, the *kill rate* for both *Straight* and *Round* enemies have increased from 0.26 to 0.32 and from 0.44 to 0.71, respectively. It is important to remember that the total of enemies spawned by each type for both default and dynamic balancing was the same. In fact, when looking for the total kill rate for all enemies for both default and dynamic balancing, they are very close to each other. It indicates that, in general, the difficulty of the whole game using default and dynamic balancing stays the same.

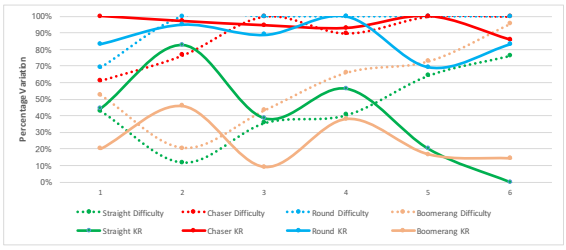
Figure 5 presents this variation along the six sessions for the lowest (P1) and highest (P3) score participant in order to see how the balance factors change along time and influences the player. When looking at Figure 5(a), it is possible to observe that at the first session, P1 was only able to kill *Chaser* enemies (a 100% kill rate), having difficulty to deal with the other kind of enemies. It indicates that this participant had no skill to deal with the behavior presented by the *Straight*, *Round* and *Boomerang* enemies as their kill rate were zero. Based on this observation, our proposed dynamically balancing model started to decrease the *Boomerang* and *Straight* enemies level of difficulty. However, in order to not increase so much the game difficulty, the *Chaser* difficulty stayed almost as the same. Based on this new parameters for the difficulty,

TABLE IV. PARTICIPANTS' PERFORMANCE ON KILLING ENEMIES (KILL RATE) FOR DEFAULT AND DYNAMIC BALANCING.

Participant	Default Balancing				Dynamic Balancing			
	Straight	Chaser	Boomerang	Round	Straight	Chaser	Boomerang	Round
	Kill Rate				*Kill Rate*			
P1	0.26	0.78	0.23	0.44	0.32	0.72	0.19	0.71
P2	0.63	0.96	0.37	0.90	0.65	0.89	0.41	0.77
P3	0.73	0.98	0.44	0.98	0.59	0.95	0.30	0.90
P4	0.79	0.96	0.32	0.87	0.91	0.87	0.38	0.81
P5	0.64	0.76	0.13	0.59	0.57	0.71	0.25	0.83
Total	**3.05**	**4.44**	**1.49**	**3.78**	**3.04**	**4.14**	**1.53**	**4.02**



(a) Dynamic difficulty balancing for enemies in P1's sessions.



(b) Dynamic difficulty balancing for enemies in P3's sessions.

Figure 5. Dynamic difficulty balancing for enemies. KR states for *kill rate*.

in session two P1 was able to deal more easily with the *Straight* enemies. In fact, P1 even increased the kill rate for the *Round* enemies. Based on this, we observed that keeping the difficulty for *Boomerang* and *Straight* enemies highly impacted the performance of this participant in the whole game. This observation happened again when BinG slightly increased the difficulty for the *Straight* enemy, as even BinG constantly increased the difficulty for the *Round* enemy, the amount of enemies killed in session three decreased. Besides that, it is possible to observe a direct relationship between the performance of P1 when changing the difficulty of *Boomerang* enemy. A slightly increase on its difficulty clearly impacted on the performance of the participant for this type of enemy. On the other hand, both *Round* and *Chaser* enemies were not highly impacted by changing its difficulty even when increasing or decreasing the difficulty for *Straight* enemy. It is important to notice that this kind of analysis cannot be done by just looking at the numbers of enemies killed. The provenance information of cause-and-effect relationships that is used by BinG is a key feature for achieving such result.

For P3, in Figure 5(b), the chart is completely different. As this participant was the most skilled one, the difficulty for all types of enemies had a high variation from session to session, but a tendency to increase (see the decreasing slope in most of the curves). Using the data collected in the first session, it was possible to see that *Chaser* and *Round* enemies were the easiest for P3, achieving a kill rate of 100% and 83%, respectively. In this case, in order to be challenging to the player, these enemies increased their difficulty abruptly from the first to the second session. On the other hand, P3 had difficulty when dealing with *Boomerang* and *Straight* enemies, leading to a decrease in their difficulty. Based on this new balancing parameter for *Boomerang* and *Straight*, P3 had a hard time for killing them, as observed in session 3. On the other hand, his kill rate for both *Chaser* and *Round* enemies increased. In fact, it is possible to conclude that difficulty changes for *Boomerang* and *Straight* enemies highly impacted in P3's kill rate, meaning they had a tidy relationships over the P3's performance when they were together. In this case, reaching a challenging game for this participant involves changing the *Boomerang* and *Straight* enemies' difficulty.

This analysis clearly shows that achieving balancing for different players involves changing different parameters for the same game. The reduced standard deviation found in Table III for dynamic balancing in relation to default balancing is based on lowering and increasing the enemies' level of difficulty. It is important to state that we only performed enemy's dynamic balancing using just six sessions. We believe that increasing the number of sessions tends to reach an equilibrium state for these balancing factors, reducing even more the standard deviation. This hypothesis is subject of future work.

C. *RQ3: Does adjusting the player's damage based on his/her current skills contributes to his/her performance?*

As presented before, our balancing model also changes the player's weapon damage based on how well the participant is at hitting the enemies. In this case, slightly changing the effectiveness of the player's weapon damage output would contribute to increase the performance of the participant without letting him know that the game is helping him/her. In Figure 6, it is possible to see the variation of the effectiveness of the damage caused by the player's weapon along the six sessions for each participant. It is important to notice that there is a high variation in this damage between sessions for P1 and P5 in relation to the others. For instance, the player's weapon damage changed from 1 in session 1 to 2 in session 4. This indicates that this participant was having troubles with defeating the enemies. In fact, according to Figure 7, that shows all enemies kill rate for each participant across

all sessions, it is possible to see a decreasing kill rate from session 1 to session 2. On the other hand, when looking at P1, he started the game with the lowest kill rate, leading to a high increase at his weapon damage.

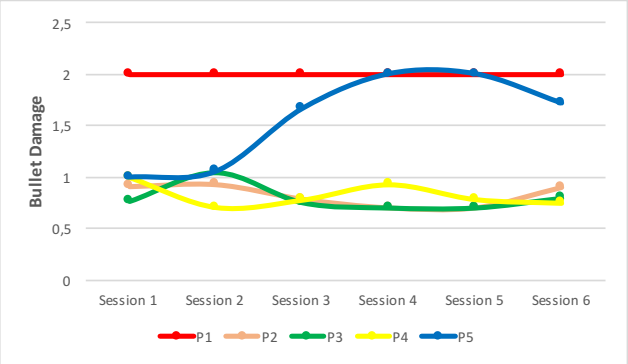


Figure 6. Participant's bullet damage variation across sessions.

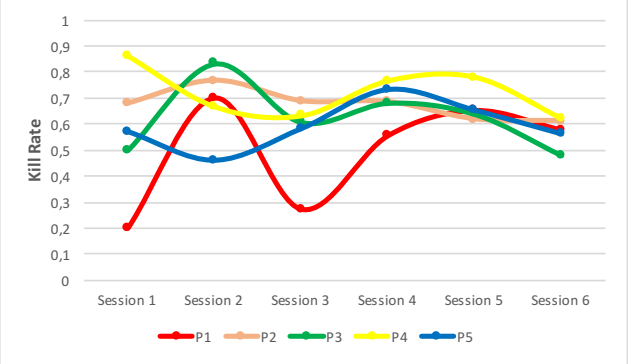


Figure 7. Participant's kill rate for all enemies across the sessions.

One important fact to observe is that, although there is a high variation on the weapon damage among the participants in Figure 6, their kill rate almost converged. When looking at Figure 7, a high variation on the kill rate for each participant is observed in session 1. At the end, in session 6, this difference is much smaller, indicating they are converging, even when each one had different skills. Giving the same opportunities for players with different skills is one of the key factors during balancing a game.

D. RQ4: Does the usage of dynamic balancing for items contributes to the player's performance?

Finally, the items available during the game can contribute greatly for the performance of the participant. Depending on the challenges the participant is facing, it is important to spawn the most indicated item that could help him/her to overcome the obstacles. In addition, spawning an item that cannot be reached by the player makes it useless. Table V shows the total of items spawned and items collected, by minute, for each participant considering all sessions using default and dynamic balancing. From Table III, P1 is the participant which survived less in comparison with the other participants when using both default and dynamic balancing, making him an appropriate candidate to be helped. However, the frequency of items that cause negative effects to the player (*Damage Down* and *Speed Down*) is almost the same as the positive ones (*Damage Up* and *Speed Up*) for the default balancing. On the other hand, when using the dynamic balancing, the number of negative items spawned decreased, while the number of positive items increased in order to lower the game difficulty. In addition, neither of these negative items have been collected by the participant in the dynamic balancing as they were spawned farther from the player.

On the other hand, when looking at P3, the participant who lived longer according to Table III, a high number of positive items had been spawned, contributing even more for this participant to stay alive longer. In this case, in order to challenge this participant, the number of positive items should be lowered. It is what happened when looking at spawned items during the the dynamic balancing sessions. The number of positive items spawned decreased by almost half of the initial value. Additionally, the number of *Speed Up* items increased, indicating this participant was having problem for dodging enemies. During the analysis process, BinG concluded that he was a skilled player at killing enemies so the effectiveness of his damage had been lowered by this item.

VII. Threats to Validity

Normally, players get better in a game as they play it. Thus, a threat to validity is related to players playing all the sessions in the same day from both the Default and Dynamic balancing in sequence. In order to mitigate some bias toward "learning" the game, we changed the order of the experiments (Default and Dynamic) with some participants. Another possible threat is related to when the dynamic balancing occurred, which was only before a game session started and not in real time. This did not allow BinG to adapt during a game session or try to correct its initial assumptions on the fly for that game session and only after it concluded.

Finally, the low number of participants in the experiment can also be viewed as another threat. However, we tried to compensate this by performing a detailed analysis of the data.

VIII. Conclusion and Future Work

This paper presented BinG, a framework for performing dynamic balancing in games using logical programming. Our framework facilitates the process of detecting balancing issues and acts appropriately in real time through a pluggable balancing model. Moreover, we showed how this balancing model could be developed and used in a game developed from scratch. Besides that, we analyzed how dynamic balancing influences the player's performance based on a study with five participants, who played the game using the default game difficulty and the dynamic balancing. This analysis has been guided by understanding how the participant's performance varies when changing the entities behavior characteristics in the game. At the end, we showed that using dynamic balancing tends to give more opportunities for participants with lower

TABLE V. FREQUENCY OF ITEMS SPAWNED | COLLECTED BY MINUTE FOR EACH PARTICIPANT USING DEFAULT AND DYNAMIC BALANCING.

Participant	Default Balancing								Dynamic Balancing							
	Damage Up		Damage Down		Speed Up		Speed Down		Damage Up		Damage Down		Speed Up		Speed Down	
P1	2.43	1.04	1.74	0.35	2.08	1.39	2.08	0.00	4.96	2.29	0.38	0.00	3.05	1.53	0.38	0.00
P2	2.30	1.87	2.30	0.43	2.59	2.16	2.45	0.00	3.69	3.41	0.00	0.00	4.83	4.26	0.57	0.00
P3	4.61	4.61	1.85	0.45	2.10	1.91	3.25	0.70	2.29	2.07	0.95	0.00	3.81	3.34	0.64	0.00
P4	3.09	2.89	2.27	0.62	1.44	1.03	2.68	0.82	3.01	2.78	0.46	0.00	5.32	4.63	0.69	0.00
P5	1.33	1.33	2.66	0.00	3.65	2.99	1.33	0.33	2.83	2.48	0.71	0.00	3.89	3.54	1.41	0.00

skills, reducing the average survival time and achieved score. In addition, we found less standard deviation over these parameters when compared to the default balancing.

BinG allows the usage of different balancing models as desired by developer. However, we envision a way that players themselves choose the desired balancing model as an extended option for difficulty selection. In this case, instead of using statically defined parameters for the available levels of difficult in the game, the player can indicate how the difficulty should increase dynamically through BinG.

We are currently working on expanding the schema file, allowing more complex rules to be extracted automatically. In addition, we are also researching a way to couple machine learning in BinG to detect patterns during a game session. The detected patterns could be available to be used by the dynamic balancing model. Furthermore, we are also implementing a tool to allow a real time simulation of the balancing model. We believe this tool can easy the process of validating the balancing model parameters in a faster way, without the necessity to run the game every time for testing the model.

ACKNOWLEDGMENT

The authors would like to thank CAPES, NVidia, CNPq, and FAPERJ for the financial support.

REFERENCES

[1] J. Gregory, *Game Engine Architecture, Second Edition*. CRC Press, 2014.

[2] L. Jacob, E. Clua, and D. de Oliveira, "Oh gosh!! why is this game so hard? identifying cycle patterns in 2d platform games using provenance data," *Entertainment Computing*, vol. 19, pp. 65 – 81, 2017.

[3] J. P. Davis, K. Steury, and R. Pagulayan, *A survey method for assessing perceptions of a game: The consumer playtest in game design*. Game Studies 5, 2005.

[4] J. Schell, *The Art of Game Design: A Book of Lenses*. Morgan Kaufmann Publishers, 2008.

[5] T. G. Hallam, "David g. luenberger: Introduction to dynamic systems, theory, models, and applications. new york: John wiley & sons, 1979, 446 pp," *Behavioral Science*, vol. 26, no. 4, pp. 397–398, 1981.

[6] R. Hunicke, "The case for dynamic difficulty adjustment in games," in *Proceedings of the 2005 ACM SIGCHI International Conference on Advances in Computer Entertainment Technology*, ser. ACE '05. New York, NY, USA: ACM, 2005, pp. 429–433.

[7] G. Hawkins, K. Nesbitt, and S. Brown, "Dynamic difficulty balancing for cautious players and risk takers," *Int. J. Comput. Games Technol.*, vol. 2012, pp. 3:3–3:3, Jan. 2012.

[8] O. Missura and T. Gärtner, "Predicting dynamic difficulty," in *Advances in Neural Information Processing Systems 24*, J. Shawe-Taylor, R. S. Zemel, P. L. Bartlett, F. Pereira, and K. Q. Weinberger, Eds. Curran Associates, Inc., 2011, pp. 2007–2015.

[9] L. J. F. Prez, L. A. R. Calla, L. Valente, A. A. Montenegro, and E. W. G. Clua, "Dynamic game difficulty balancing in real time using evolutionary fuzzy cognitive maps," in *2015 14th Brazilian Symposium on Computer Games and Digital Entertainment*, Nov 2015, pp. 24–32.

[10] J. K. Olesen, G. N. Yannakakis, and J. Hallam, "Real-time challenge balance in an rts game using rtneat," in *2008 IEEE Symposium On Computational Intelligence and Games*, Dec 2008, pp. 87–94.

[11] F. Mourato, F. Birra, and M. P. dos Santos, "Difficulty in action based challenges: Success prediction, players' strategies and profiling," in *Proceedings of the 11th Conference on Advances in Computer Entertainment Technology*, ser. ACE '14. New York, NY, USA: ACM, 2014, pp. 9:1–9:10.

[12] A. Rietveld, S. Bakkes, and D. Roijers, "Circuit-adaptive challenge balancing in racing games," in *2014 IEEE Games Media Entertainment*, Oct 2014, pp. 1–8.

[13] T. C. Kohwalter, F. M. de Azeredo Figueira, E. A. de Lima Serdeiro, J. R. da Silva Junior, L. G. P. Murta, and E. W. G. Clua, "Understanding game sessions through provenance," *Entertainment Computing*, vol. 27, pp. 110 – 127, 2018.

[14] L. B. Jacob, T. C. Kohwalter, A. F. V. Machado, E. W. G. Clua, and D. d. Oliveira, "A non-intrusive approach for 2d platform game design analysis based on provenance data extracted from game streaming," in *Proceedings of the 2014 Brazilian Symposium on Computer Games and Digital Entertainment*, ser. SBGAMES '14. Washington, DC, USA: IEEE Computer Society, 2014, pp. 41–50.

[15] T. Kohwalter, E. Clua, and L. Murta, "Provenance in games," in *Proceedings of the 2012 Brazilian Symposium on Computer Games and Digital Entertainment*, 2012, pp. 162–171.

[16] T. C. Kohwalter, E. G. Clua, and L. G. Murta, "Game flux analysis with provenance," in *10th International Conference on Advances in Computer Entertainment - Volume 8253*, ser. ACE 2013. New York, NY, USA: Springer-Verlag New York, Inc., 2013, pp. 320–331.

[17] C. Chandler and L. Noriega, "Games analysis how to stop history repeating itself," in *WSEAS International Conference on Multimedia, Internet & Video Technologies*, 2006, pp. 47–52.

[18] M. Csikszentmihalyi and I. S. Csikszentmihalyi, *Optimal Experience: Psychological Studies of Flow in Consciousness*. Cambridge University Press, 1992.

[19] T. J. Tijs, D. Brokken, and W. A. Ijsselsteijn, "Dynamic game balancing by recognizing affect," in *2Nd International Conference on Fun and Games*. Berlin, Heidelberg: Springer-Verlag, 2008, pp. 88–93.

[20] R. J. V. d. Medeiros and T. F. V. d. Medeiros, "Procedural level balancing in runner games," in *2014 Brazilian Symposium on Computer Games and Digital Entertainment*, Nov 2014, pp. 109–114.

[21] J. Freire, D. Koop, E. Santos, and C. T. Silva, "Provenance for computational tasks: A survey," *Computing in Science and Engg.*, vol. 10, no. 3, pp. 11–21, May 2008.

[22] "Unity: Game development tool," http://unity3d.com/, accessed: 2017-01-20.