Dynamic Difficulty Adjustment in Digital Games Using Genetic Algorithms

Matheus Weber
PPGEEC
Universidade Presbiteriana Mackenzie
São Paulo, Brazil
matheus.jk.weber@gmail.com

Pollyana Notargiacomo
PPGEEC / FCI
Universidade Presbiteriana Mackenzie
São Paulo, Brazil
pollyana.notargiacomo@mackenzie.br

***Abstract*—The difficulty of a game is intrinsically connected with the experience of immersion in it and with its success. One of the main reasons for a player to drop a game is that the game is either too easy or too hard for him/her. In practice, players become either bored or frustrated if playing a game that is not balanced for them. An approach to prevent this kind of behavior is to dynamically adjust the difficulty of a game so that the game adapts to the player's experience by evaluating the difficulty of a game and changing its environment to become easier or harder for the player. In this paper, we propose a real-time solution using a Genetic Algorithm which helps to provide the exact amount of challenge that a player needs to not be bored or frustrated thus balancing the difficulty of a game. We review several other papers that approached this problem, which characteristics an algorithm has to have to approach the problem, and how to balance this in a generic way. The main idea of this paper is to create an approach that can be modified and coupled to any kind of game by using a Genetic Algorithm.**

***Index Terms*—dynamic difficult adjustment, genetic algorithm, artificial intelligence, digital games, dynamic difficulty balance**

I. Introduction

The difficulty of a game and its ability to provide fun are intrinsically connected; the only objective of a game is to engage the player to rise to challenges and achieve objectives. A game is a dynamic system in which many elements work together to create a complex system, and in which such elements can be formal or emotional. Moreover, limits and well-defined rules aim at entertaining players [1].

The nature of fun experiences in games has been studied for many years in many different areas. There are a number of works related to the topic in psychology focusing on the engagement in play games. One of these works is Csikszentmihalyi's Theory of Flow, which describes a state of mind in which an individual is so completely engaged in a task that he or she loses his or her sense of self. In order to achieve this state, the task being performed by the player must not be either so hard that it is frustrating or so easy that it is boring [2].

For a game to be fun, it cannot be either too hard or too easy, it has to be balanced using the player's experience as a parameter and the player's skills must be determinant of his or her success [3]; the psychological factor of a player directly influences the fun he or she has when playing the game, which generates the necessity to keep the player interested in it [4]. One approach to achieve this is to link the difficulty of the game to the player's skills through dynamic game balancing [5].

In this paper, we approach this problem by developing a model and creating a generic Genetic Algorithm (GA) followed by the definition of metrics to evaluate whether a game needs to be adjusted or not. Later, a game developed to be used as subject for this work is described, and, finally, the genetic algorithm is customized and integrated into the game.

The game developed for this study consists of players taking turns to compete for resources and buildings on a board with each turn having two stages. Before each turn, the GA will generate a set of configurations to be used in that stage of the game, and at the end of a turn, it will evaluate that set of configurations. The best configurations will have bigger suitability and higher chance to continue to be used throughout the game. All these steps will be performed using GA concepts that will be customized for the game developed for this work.

To sum up, the core contributions of this paper are related to the Dynamic Difficulty Adjustment (DDA) approach to any game by using a Genetic Algorithm, a challenge rate that can be adjusted to any game and input into the GA, and a real time DDA that successfully improves the challenge rate during the game. To approach these elements, the present paper is organized in the following way: in "Related Works", we will review related work in DDA approaches; secondly, we will introduce the Genetic Algorithm and a proposal for use; then, we will present the metrics defined to evaluate the need of the game for adjustment; after that, we will present the game used as subject for this work and how the algorithm is connected to the game. Finally, we discuss the results and conclusions of this study and analyze further work that can stem from this study.

II. Related Work

The increase of investment in the game industry has led to a growing concern to make games more realistic and challenging, and to prioritize the quest for an immersive experience. One of the ways to reach this objective is by making the game more challenging for the user, which motivates him or her to keep playing it. Because of that, many studies are conducted to identify and balance the difficulty of a game such as studies to identify the player's characteristics and motivations to play games [6].

Many studies were and are conducted in Artificial Intelligence to increase engagement and immersion in digital games: the main objectives are to identify how well-balanced a game is, what to change if it is not well-balanced, and how much to change it. There are studies involving metrics, challenge rates and even physiological responses [7].

For example, the game "*Left 4 Dead*" uses an Artificial Intelligence algorithm called "*Director*" in order to control the difficulty balancing of the game. Depending on the player's level of skill and the game's level of difficulty, the Artificial Intelligence algorithm balances the game by changing the number of enemies, the items the player will find on the map and other elements [9].

In order to achieve Dynamic Difficulty Adjustment, a method must meet three requirements: i) the game must automatically identify the skills of the player and adapt to them; ii) the game must maintain its difficulty in pair with the player's abilities; iii) this process must not be noticed by the player [10]

To calculate the difficulty of a game, and thus balance it, it is important to study the motivations that each player has. This information allows to prepare the game to challenge each particular player. Several studies have been carried out in this area among which the work of Bartle [8] stands out: it states that players can be divided into four groups based on their behavior and their motivations. They are:

- **Achievers** - players who are always searching for achievements, which can be represented by money, prizes, items, among others.
- **Explorers** - players who are always trying to learn more about the universe of the game.
- **Socializers** - players who like to play together with other players.
- **Killers** - players who like to cause distress to other players.

Various definitions about how to balance the difficulty of a game have been described and reported in many papers and books. Adams [3] stated in his book that for a game to be fun it needed to be balanced; it cannot be, from the player's standpoint, either too easy or too hard; he also defines that a well-balanced game has these characteristics: I) it has significant choices; II) a player's luck is not more important than his or her ability; and III) a player must perceive the game as being fair.

A game's difficulty can be adapted to a player's profile through the adjustment of the game's characteristics such as weapon power, healing capacity, storytelling, and even its scenario. Therefore, it is necessary to perform a study to understand the impact that each characteristic has on the game in order to make it possible to balance it [11].

Many studies were conducted to solve the *Dynamic Difficulty Adjustment* problem, each with a different strategy that differs in many aspects but have two points in common:: **Prediction** and **Intervention**, The algorithm must **predict** when the game is too easy/hard for the player and **intervene** to adjust the difficulty [13].

Some works used the environment of the game to predict the difficulty of the game and intervene to adjust it. Hunicke [14], Christyowidiasmoro [27], Shin-Hung Chang, Nai-Yan Yang and Silva [29], and also Van Lankveld et al. [30] used this strategy of detecting the difficulty by monitoring characteristics of the game's environment such as: player's *Health Points*, *Attack Power*, *Resources*, *Level*, among others. After predicting if the game's difficulty needed adjustment, the algorithm intervened by changing aspects of the game environment such as number of player's enemies appearing on the map, item's chance to drop by defeating an enemy, experience gained by completing the quest, among other characteristics. The result was algorithms that were able to improve the player's immersion and experience by adjusting the difficulty without their perceiving that something had been changed.

In Baldwin's paper [15], a framework is created to classify many techniques used in DDA for multiplayer games, because the approach to multiplayer style must be different from the approach to single player style. The framework designed is able to classify seven characteristics that describe where and how a DDA algorithm must act. They are: *Determination*, *Automation*, *Recipient*, *Skill Dependency*, *User Action*, *Duration* and *Visibility*. It is also remarked that the difficulty balancing in a game has three dependencies: *Activation Trigger*, *Rules Affected* and the *Effect Scope*.

Another approach to adjust difficulty is by using Artificial Intelligence to calculate the weight that the characteristics of the game have and adjust them automatically while the algorithm is learning the connection between them. In "*Dynamic Game Difficulty Balancing in Real Time Using Evolutionary Fuzzy Cognitive Maps*" [16], the authors used evolutionary maps to predict and intervene in the game's difficulty . In "*Real-time challenge balance in an RTS game using rtNEAT*" [17] and "*Real-Time Game Adaptation for Optimizing Player Satisfaction*" [18], neural networks were used to achieve the adjustment, while the "*Monte carlo tree search based algorithms for dynamic difficulty adjustment*" describes a *Monte Carlo* algorithm that also achieves balance [19]. "*Challenge-Sensitive Action Selection: an Application to Game Balancing*" [20] uses reinforcement learning as a technique to balance games. All of these techniques are able to dynamic adjust the difficult of games by analyzing and manipulating the characteristics of the game such as *obstacles*, *items*, *speed* and others. The result of these approaches is the finding of the weight of the influence that a characteristic has and how much to change it to adjust the game difficulty in real time.

A more complex approach to calculate a player's skill is studied in *"TrueSkillTM: A Bayesian Skill Rating System"*, in which a new **Bayesian** model is described to calculate how to balance multiplayer games [21].

Another way to define metrics to the difficulty of a game is by using physiological responses. In "*Dynamic Game Balancing by Recognizing Affect*" and in "EEG-triggered dynamic difficulty adjustment for multiplayer games", the authors were able to adjust the difficulty of a game when analyzing the physiological responses linked to the player's emotions [22], [23]. Physiological responses that can be analyzed are: level of skin conductance, number of responses from skin conductance, heart rate, breath rate, eyebrow movements, cheek movements and the force used to click the mouse [24], [25].

Finally, another way to predict the game difficulty is by using the player's own opinion. In Medeiros's paper, difficulty was dynamically adjusted by using explicit feedback from the players to feed a machine learning algorithm and determine what characteristics of their game must be changed in order to improve the balancing. In their *Runner* game, it was possible to modify these characteristics: *firing drones*, *spiky drones*, *number of drones*, *darkness*, *wind* and *number of tracks*. By changing such characteristics, they were able to achieve the level of difficulty necessary to not bore or frustrate their players [26]

All of these studies show the importance of measuring the difficulty of a game and of adjusting it. In order to follow the *Flow Theory* [2], many of these studies use the characteristics of the game to determine the level of difficulty and to increase or decrease the same. Table I indicates the strategy used by each study described. In table I, the numbers in the column "Work" refer to:

1) Dynamic Difficulty Adjustment for Maximized Engagement in Digital Games
2) Measuring level of difficulty in game using challenging rate (CR) on 2D Real time Strategy Line Defense game
3) DCA: Dynamic Challenging Level Adapter for Real-time Strategy Games
4) Difficulty Scaling through Incongruity
5) A framework of Dynamic Difficulty Adjustment in competitive multiplayer video games
6) Real-time challenge balance in an RTS game using rtNEAT
7) Dynamic Game Difficulty Balancing in Real Time Using Evolutionary Fuzzy Cognitive Maps
8) Real-Time Game Adaptation for Optimizing Player Satisfaction
9) Monte carlo tree search based algorithms for dynamic difficulty adjustment
10) Challenge-Sensitive Action Selection: an Application to Game Balancing
11) TrueSkillTM: A Bayesian Skill Rating System
12) Dynamic Game Balancing by Recognizing Affect
13) EEG-triggered dynamic difficulty adjustment for multiplayer games
14) Procedural Level Balancing in Runner Games

TABLE I
CLASSIFICATION OF STUDIES

Work	Strategy
1	Game Environment
2	Game Environment
3	Game Environment
4	Game Environment
5	Framework
6	Artificial Intelligence
7	Artificial Intelligence
8	Artificial Intelligence
9	Artificial Intelligence
10	Artificial Intelligence
11	Bayesian algorithm
12	Physiological Response
13	Physiological Response
14	Explicit Feedback

III. Genetic Algorithm

Genetic Algorithms (GAs) are search and optimization algorithms based on the Theory of Evolution. These algorithms are based on natural selection and survival of the fittest as inexorable for the evolution of a population. Such algorithms codify one possible solution in data structures based on chromosome composition and apply evolutionary techniques such as *mutation* and *crossover* in order to reach an optimal solution to a problem [6].

A GA works by generating populations of individuals that adapt by using genetic operators and, after it, they select the best individuals and remove the rest. Through this, a GA preserves the best genes to evolve to the next generation of the population until finding an optimal solution to the problem which is being treated [31]. A GA is not like traditional search methods, it differs in the sense it: i) works with parameter coding; ii) uses populations; iii) uses evaluation function; iv) and uses probabilistic concepts instead of deterministic ones [32].

The main elements of a Genetic Algorithm are [32]:

- *Gene* - represents one characteristic that has to change so that optimal solution can be found.
- *Individual* or *Chromosome* - a data structure that codifies a possible solution to the problem. It can be represented as an array of any kind of elements (Genes) in which each element represents a characteristic of the problem. The coding must be as simple as possible; it cannot have any representation that is not a possible solution to the problem. If the problem has conditions and limitations, they must be represented in the coding.
- *Population* - a set of chromosomes that will suffer *mutation* and *crossover* and will undergo selection in order to find the optimal solution to the problem.
- *Evaluation Function* or *Fitness Function* - a function used to determine the quality of a chromosome, how close or not it is to the solution wished. It is used to discard individuals or choose them to survive in the selection step.
- *Selection* - a step of the algorithm that behaves as natural selection, in which the best chromosomes will reproduce and pass their genetic load on to the next phase or will be discarded. It must privilege the individuals with higher fitness and it must retain some the individuals with lower fitness, because even the least adapted individuals can have important genetic characteristics.
- *Crossover* - a genetic operator that will recombine chro-

mosomes in order to create new individuals while maintaining and mixing their genetic characteristics.
- *Mutation* - a genetic operator that will change a random element of a chromosome to a random value by "flipping a coin". By doing this, it will explore new characteristics with a random chance.

The genetic operators have a chance to happen which cannot be either too high or too low, because if they happen very frequently, the algorithm can become a random algorithm; but also, these operators set the speed at which the algorithm will achieve optimal solution [32].

The objective of a Genetic Algorithm is to find an optimal solution to a problem, and to do it by putting together several steps that make up the algorithm, as described in 1 and detailed as [34]:

- Step 1: Generate Population - It will generate an initial random population of individuals, or it will generate a population from the chromosomes that survive selection.
- Step 2: Evaluate Population - It will evaluate every chromosome of the population, adding a *fitness* value to each of them.
- Step 3: Stop Condition - It will assess if any of the evaluated chromosomes achieve the desired stop condition. For example, if any of them has a fitness value lower than some threshold number. If so, the algorithm can stop and that chromosome is the optimal solution to the problem.
- Step 4: Selection - It will select the "best" chromosomes of the population.
- Step 5: Crossover - It will apply crossover if necessary.
- Step 6: Mutation - It will apply mutation if necessary.

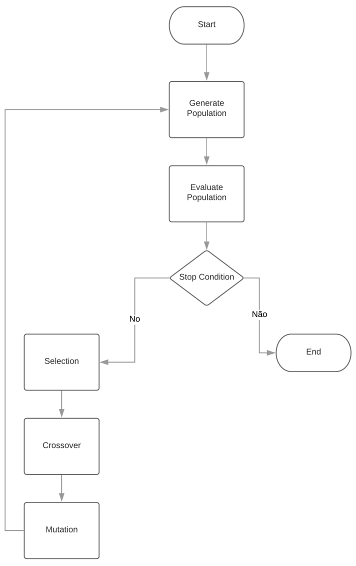

Fig. 1. Genetic Algorithm scheme. Source: Elaborated by authors.

IV. Metrics

In order to evaluate if a game needs to have its difficulty adjusted, it is necessary to define a way to calculate its difficulty and a way to adjust it by tuning the characteristics or the environment of the game to the player, and by increasing or decreasing the difficulty through an approach unnoticed by the player. Some works used the elements of the game itself to attribute a weight to each element and used the player's experience as output. By using neural networks of cognitive maps, and by knowing the weight of each element, the algorithm can manipulate elements to adjust the difficulty in a dynamic way [16], [17].

Other works opted to establish what the weight of each element would be and how it could be adjusted using an equation to determine the difficulty experienced by the user. By calculating this difficulty, it was able to change the game's environment to adjust difficulty in real time [14], [27], [29].

On the other hand, some papers tried to use player feedback to calculate if the game is too hard or too easy. This feedback can be collected using the players' physiological responses or even using forms. With this information, their algorithm was able to adjust the game using its own methods [24], [26].

All of these works tried to achieve the *flow* described in the theory created by Csikszentmihalyi [2], by detecting the difficulty and applying changes to the player's experience that are supposed to increase or decrease difficulty in real time.

This work opted to create a metric in a hybrid way: the difficulty of the game is calculated using an equation that obtains the difference between the score of the first player and the score of the last player of the game, as described in (1).

$$max(P) - min(P) = F \tag{1}$$

Where:

$$P = \{P_i | score\} \tag{2}$$

The weight of each characteristic of any game can be calculated by the Genetic Algorithm, which using any fitness function is capable to input weights across generations of a population. If a codified characteristic is important, it will be transmitted to the next generation; if not, it will not be transmitted. This is one of the reasons why the use of a Genetic Algorithm was chosen for Dynamic Difficulty Adjustment [32].

Besides that, for each game round, one set of characteristics will be evaluated using the fitness function by changing the game environment to reflect each one of those sets. At the end of the round, the *fitness value* will be evaluated and attached to the set [32].

The hypothesis is that by manipulating the chromosomes that codify characteristics of the game, these characteristics that maintain the player in the *flow* state will be more and more chosen throughout the game rounds until the game ends, and through this, the objective of this work will be achieved. Moreover, as the chromosomes are capable of codifying any kind of characteristic, they can be used for any kind of game, and as the *fitness function* can be anything related to the problem, it can also be used for any kind of game. In this work we decided to use the equation described in (1) for the subject game described in the next section.

V. Empire Game

The *Empire* Game was developed to be used as subject in this study. A Game Design Document (GDD) [33] was also developed for the game describing all the characteristics of the game, including the story, characters, powers and others. *Empire* is a board game that can be played on iPad and it was developed using the programming language *Swift*. It has single player and multiplayer options that allow many individuals to interact using the same iPad or allow a single player to play against Artificial Intelligence (AI) agents. The objective of the game is to score more than everyone. The highest scoring player wins the game.

The game takes place in space, and each player or AI agent will choose one of ten characters with a passive or active power and own history. Each player will have a citadel-spaceship that will be used to conquer a recently discovered planet that will serve the character's purpose, which is to finish the game with the highest score and thus conquer the planet.

Players will compete against each other for resource fields using their spaceships. Types of resources can be *Fuel*, *Iron*, *Uranium*, *Plasmidium* and *Nevidium*. Each field has limited places to be mined by players and each field has a limited chance of success of being mined. In order to occupy a field, the player sends one or more spaceships to the resource fields, and each spaceship has a chance to return with one or more resources.

The resources mined will be used to buy buildings which will compose the citadel-spaceship and so increase the player's score; there are also extra points for building all the five types available or for building only one type of building. In order to construct a building, the player will send a spaceship to occupy the building site in the same way as they do to mine resources, but the chance to return with a building is 100%.

Finally, a player can build spaceships by consuming resources to increase their presence in buildings and resource fields.

Fig. 2 shows the distribution of the field as quantity of ships, resources and buildings that the player owns.

The game ends when there are no more buildings to be built by the players or if a player manages to construct seven buildings.

In Fig. 3, a finite-state machine is described showing how the player will interact in the game. Before the game starts, all players must choose their characters; after this, the game will start in a loop that can be divided into two stages:

- Preparation: When the player will put his or her spaceship on the game board.
- Acting: When the player will retrieve his or her ship, build new ships, or use his or her power.

The loop will go on until the game ends and one player is declared the winner.

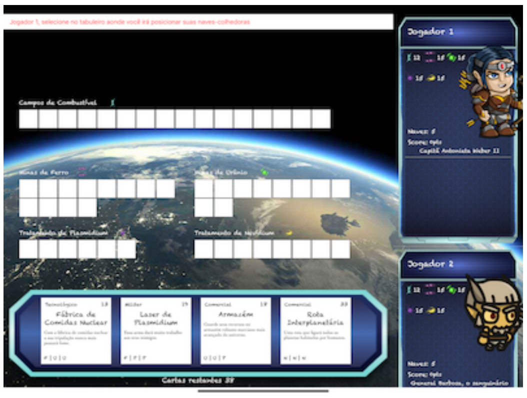

Fig. 2. Board screen. Source: Elaborated by authors.

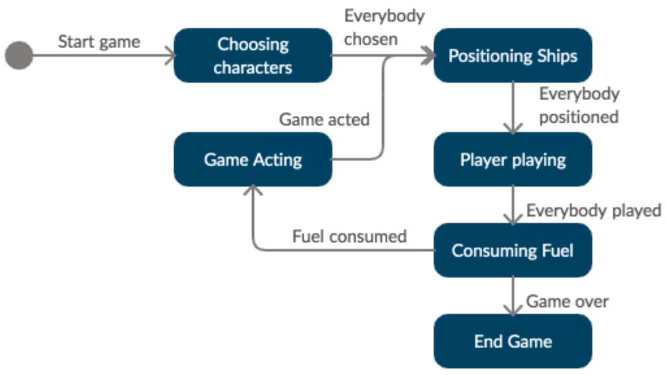

Fig. 3. Empire FSM. Source: Elaborated by authors.

VI. Integration

In order for this approach to work, it is necessary to integrate the developed Genetic Algorithm with the subject game used for this study.

To achieve this, the first change that the GA suffered was that each player will have one population for themselves, because each player has to have their own difficulty adjusted. Secondly, in order to be constantly adjusting the game, the genetic algorithm no longer needs a *Stop Condition*, so it was replaced by a verification of whether each chromosome of the population has been evaluated. The result is a change of the steps of the algorithm showed in Fig. 4.

After this, the genetic algorithm was developed using *Swift*, , because it is an object-oriented programming language that can be used to develop for *Ipad*, a platform chosen because it is the most used in the market nowadays [35], so the algorithm was developed based on this paradigm which is applicable to generalizing and specializing the Genetic Algorithm needed in order to develop DDA for any kind of games. The result was a set of classes described in Class Diagram 5 in which:

- `Practice` - is responsible for controlling all the tests described in this paper. Maintaining instances of the

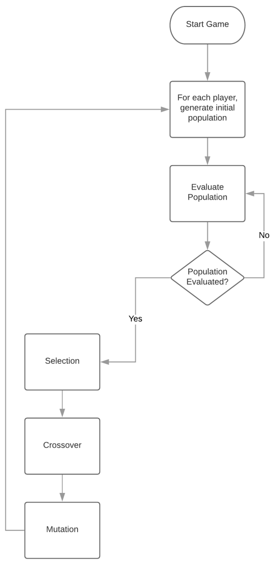

Fig. 4. Adapted genetic algorithm. Source: Elaborated by authors.

Empire Game and Genetic Algorithm, a new game can be simulated with AI and a new game can be started with actual players.

- `GeneticAlgorithm` - is responsible for the main flow described in Fig. 1.
- `Population` - is responsible for maintaining all individuals of a population, for evaluating the population, for applying mutation and crossover, for selecting the best elements of the population, and for generating an initial population.
- `Chromosome` - is responsible for maintaining all individuals that codify a possible answer to the problem. It can generate random chromosomes, an initial chromosome, and it can apply mutation to any gene.
- `Gene` - is responsible for codifying one characteristic of the game.

Furthermore, three classes were extended in order to specialize the Genetic Algorithm for the *Empire Game* by overwriting and creating new methods for the mother classes.

The `EmpireChromosome` is responsible for overwritten the methods `generateRandomChromosome` and `generateInitialChromosome` and for making them able to create `EmpireChromosome` instead of `Chromosome`, also the `mutate` method is overwritten in order to be able to apply mutation in a way that the **Empire Game** needs.

The `EmpirePopulation` is responsible for overwritten the methods `generateInitialPopulation` and `evaluatePopulation`; the former will now generate a population of `EmpireChromosome` and not simple `Chromosome` and the latter will be responsible for returning whether all population has been evaluated or not.

The `EmpireGeneticAlgorithm` is responsible for adding the `EmpireGame` instance to the genetic algorithm and also for overwritten the `getPopulation` method adding the capability to return an `EmpirePopulation` instead of a `Population`.

These classes can be extended and overwritten to suit any kind of game. For example, if this algorithm is used to balance a *Pacman* game, it is possible to create a `PacmanChromosome` by using the number of ghosts as one gene, the number of fruits as another gene and so on. As for the *Fitness Function*, it can be defined as how many points the player scores at each game.

Finally, the `Gene` must be defined to codify the characteristics of the **Empire Game**, so it was decided that `Chromosome` will have five genes that represent the probability that a resource can be mined in the game with minimum and maximum probability as Table II describes.

TABLE II
GENE CHARACTERISTICS

Resource	Min value	Max value
Fuel	45%	100%
Iron	35%	80%
Uranium	25%	70%
Plasmidium	15%	60%
Nevidium	13%	50%

As the scheme represented in Fig. 1 indicates, the game will generate an initial population for each player, and this population has a number n of chromosomes that represent a set of configurations. These configurations are the probability of mining each resource. If the probability is too low, it means that it is becoming harder for the player to mine that resource, which will make the game difficulty increase, and if the probability is too high, it means that it is becoming easier for the player to mine that resource, which requires that the game difficulty decrease.

At the beginning of each game round, a new chromosome is chosen from the population. This chromosome will be used as the configuration for that round. At the end of the round, the *fitness function* described in section IV will be calculated to evaluate this chromosome. When all chromosomes of a population have been calculated, the Genetic Algorithm takes control and applies `selection`, `crossover` and `mutation`; if the *fitness* is good the chromosome will survive and will be used again; if not, it will change, guaranteeing that the Dynamic Difficulty Adjustment is being pursued.

Finally, the algorithm will never stop, because the dynamic difficulty adjustment will happen all the time players are active in the game. If the players' skills increase, the game makes it harder for them and vice versa. Besides, the *Training* class has the capability to store the Genetic Algorithm's instances to reuse the chromosomes already calculated, making the algorithm achieve the flow state faster with each game.

VII. Results

The results of this paper were obtained through numerous tests with many different configurations for the Genetic Al-

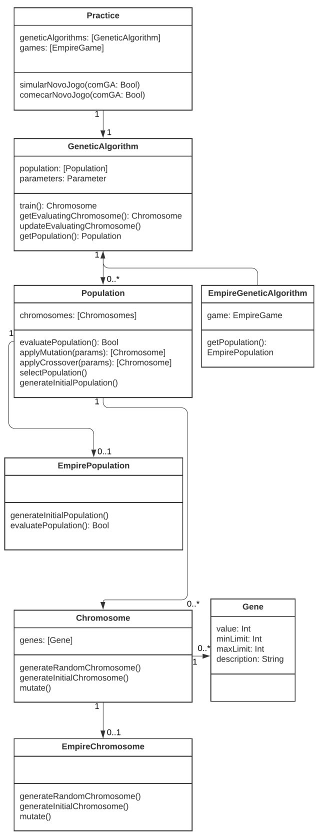
Practice
geneticAlgorithms: [GeneticAlgorithm]
games: [EmpireGame]
simularNovoJogo(comGA: Bool)
comecarNovoJogo(comGA: Bool)
1
1
GeneticAlgorithm
population: [Population]
parameters: Parameter
train(): Chromosome
getEvaluatingChromosome(): Chromosome
updateEvaluatingChromosome()
getPopulation(): Population
1
0..*
Population
chromosomes: [Chromosomes]
evaluatePopulation(): Bool
applyMutation(params): [Chromosome]
applyCrossover(params): [Chromosome]
selectPopulation()
generateInitialPopulation()
EmpireGeneticAlgorithm
game: EmpireGame
getPopulation(): EmpirePopulation
1
1
0..1
EmpirePopulation
generateInitialPopulation()
evaluatePopulation(): Bool
0..*
Chromosome
genes: [Gene]
generateRandomChromosome()
generateInitialChromosome()
mutate()
0..*
1
Gene
value: Int
minLimit: Int
maxLimit: Int
description: String
1
0..1
EmpireChromosome
generateRandomChromosome()
generateInitialChromosome()
mutate()

Fig. 5. Class diagram of Empire Game. Source: Elaborated by authors.

gorithm in order to reach the best configuration, the one that achieved the highest degree of difficulty balance.

In order to find the best configuration for the *Dynamic Difficulty Adjustment in Digital Games Using Genetic Algorithm*, intensive tests were carried out with a range of configurations in a set of tests. The configurations used can be observed in III

TABLE III
CONFIGURATIONS

Name	Values
Size of Population	[10, 50, 100, 150, 200, 500]
Mutation Rate	[5, 10, 15, 20, 50, 90]
Crossover Rate(%)	[10, 30, 50]

Tests were also carried out to assess the game's level of balance without using the balancing technique in order to compare it with the results of the algorithm.

The metrics used to measure the results of this work were:

- i) Continuous *Fitness*: analyzing whether it was possible to reduce the difference between the players' scores during the rounds;
- ii) Final *Fitness*: analyzing whether it was possible to reduce the difference between the score of the players across games;
- iii) Comparison between the game without the algorithm and with the algorithm.

For each set of parameters (108 sets), 500 tests were carried out, with the algorithm being restarted for each one of them, totaling 54,000 tests using a combination of the parameters described in the Table III for the Genetic Algorithm; and 500 tests for the game without the algorithm.

From these tests it was possible to measure the competitiveness of the game by considering the difference in scores at the end of the rounds. This can be seen in Table III, which compares the results using and not using the balancing technique.

TABLE IV
SCORE AVERAGE

	With algorithm	Without algorithm
Per round	64.01	71.03
Per end of game	119.3	126.31

It was also possible to measure competitiveness through the different configurations tested. The best results for the configurations are described in Table V.

TABLE V
SCORE AVERAGE

Config	Pop. Size	Mutation Rate	Crossover Rate	Average
1	200	20	50	115.9
2	200	50	10	116.7
3	500	20	10	117
4	500	50	10	117
5	10	50	50	117

In addition, the *Empire Game*, when played only with AI agents, has an average of **19** rounds using balancing, and an average of **23** rounds without balancing.

Moreover, it was possible to measure how the algorithm behaves throughout the rounds. Table VI shows data from five random matches:

VIII. Conclusions and Further Work

In this paper we introduced the *Dynamic Difficulty Adjustment* problem and why it is an important problem. We presented the *Flow Theory* that describes an approach to improve a player's immersion in the game and many authors that based their works on this theory.

Furthermore, we reviewed some important contributions to this area from the academy, which has many different ways to approach the *Dynamic Difficulty Adjustment*; the most common of them being the use of the game's environment itself by analyzing its characteristics and modifying them to achieve the needed adjustment, an approach that is very specialized for each game.

Our method proposes that the *Dynamic Difficulty Adjustment* problem can be solved in a very general way, and because of this, it can be specialized for any game. This solution was reached through the development of a very generic Genetic Algorithm that can be specialized for any kind of game. In

TABLE VI
MATCH PROGRESSION BY SCORE

Round	Game 1	Game 2	Game 3	Game 4	Game 5
1	26	14	18	20	16
2	29	18	26	9	26
3	61	16	46	12	34
4	31	28	72	21	47
5	29	64	56	18	34
6	53	60	59	34	53
7	38	28	47	18	56
8	63	56	57	42	53
9	85	30	85	25	56
10	72	76	78	32	53
11	70	74	66	26	74
12	75	39	49	72	74
13	52	58	52	105	109
14	85	62	71	74	79
15	127	86	81	91	11
16	141	77	73	99	97
17	129	65	65	88	107
18	125	80	70	91	END
19	130	88	END	126	END
20	121	69	END	119	END

order to test this, we developed a game specially designed for this paper, and extracted from it a fitness function that can measure the game difficulty for any player of the game. Using this function, the algorithm can modify the game and adjust difficulty.

The results indicate that the Genetic Algorithm developed for this study achieved its objective by maintaining an average score difference between the first and the last place of **119.3** points during the rounds, and of **64.01** at the end of the games, which in turn showed a slight improvement of **5.8%** and **10.93%** compared to the game without the balancing technique. In addition, analyzing round by round of all games, it was noticed that whenever the score difference (fitness) increases, the algorithm tries to correct the game settings to reduce it. The best configuration used for the GA was configuration 1 according to Table V. As shown in section VI, the algorithm can be coupled and specialized for other games. Finally, the **Empire Game** developed for this work proved possible to be played from beginning to end using AI agents as shown in the simulations performed.

The contributions of this work are: a review of the *Dynamic Difficulty Adjustment* area, a new open-source game and - the main contribution - is a method using Genetic Algorithm to balance any kind of games, which combines several approaches described in *Related Work*, and which innovates by using a Genetic Algorithm technique.

As for further research, it is necessary to explore other approaches where different sorts of matching with the GA are performed such as by using neural networks to calculate genes. Another experiment for the future is to couple this algorithm with other types of games using different genes and evaluation functions. Lastly, it appears to be promising to increase the number of affected characteristics in the *Empire Game* to detect if it is possible to reduce even more the score average.

REFERENCES

[1] Fullerton, T. Game Design Workshop. A Playcentric Approach to Creating Innovative Games, 2008.
[2] Csikszentmihalyi, M. Flow: The Psychology of Optimal Experience. Harper Perennial, 1991.
[3] Adams, E. Beginning Game Level Design. Cengage Learning PT, 2005.
[4] Yannakakis, G. How to model and augment player satisfaction: a review. First workshop on child, computer and interaction, 2008.
[5] de Araujo, B. Feijó, B. Evaluating dynamic difficulty adaptivity in shoot'em up game. 2013.
[6] Lacerda, E. and Carvalho, A., 1999. Sistemas Inteligentes. Porto Alegre: Ed. Universidade.
[7] [M. Machado, E. Fantini and L. Chaimowicz, "Player modeling: Towards a common taxonomy", 2011 16th International Conference on Computer Games (CGAMES), p. 34, 2011.
[8] R. Bartle, "Hearts, clubs, diamonds, spades: Players who suit MUDs", Journal of MUD research, vol. 1, p. 19, 1996.
[9] S. Rogers, Level up!, 1st ed. Chichester, UK: John Wiley Sons, 2010.
[10] Andrade, Gustavo Ramalho, Geber Santana, Hugo Corruble, Vincent. (2005). Extending Reinforcement Learning to Provide Dynamic Game Balancing. 7-12.
[11] M. Mateas, "Interactive Drama, Art and Artificial Intelligence", Ph.D, Carnegie Mellon University, 2002.
[12] Adams, E., 2014. Fundamentals Of Game Design. 3rd ed. Thousand Oaks: New Riders Publishing.
[13] S. Xue, M. Wu, J. Kolen, N. Aghdaie and K. Zaman, "Dynamic Difficulty Adjustment for Maximized Engagement in Digital Games", Proceedings of the 26th International Conference on World Wide Web Companion - WWW '17 Companion, 2017.
[14] Hunicke, R., 2005. The case for dynamic difficulty adjustment in games. Proceedings of the 2005 ACM SIGCHI International Conference on Advances in computer entertainment technology - ACE '05.
[15] Baldwin, A., Johnson, D., Wyeth, P. and Sweetser, P., 2013. A framework of Dynamic Difficulty Adjustment in competitive multiplayer video games. 2013 IEEE International Games Innovation Conference (IGIC).
[16] Perez, L., Calla, L., Valente, L., Montenegro, A. and Clua, E., 2015. Dynamic Game Difficulty Balancing in Real Time Using Evolutionary Fuzzy Cognitive Maps. 2015 14th Brazilian Symposium on Computer Games and Digital Entertainment (SBGames).
[17] Olesen, J., Yannakakis, G. and Hallam, J., 2008. Real-time challenge balance in an RTS game using rtNEAT. 2008 IEEE Symposium On Computational Intelligence and Games.
[18] Yannakakis and J. Hallam, "Real-Time Game Adaptation for Optimizing Player Satisfaction", IEEE Transactions on Computational Intelligence and AI in Games, vol. 1, no. 2, pp. 121-133, 2009..
[19] S. Demediuk, M. Tamassia, W. Raffe, F. Zambetta, X. Li and F. Mueller, "Monte Carlo tree search based algorithms for dynamic difficulty adjustment", 2017 IEEE Conference on Computational Intelligence and Games (CIG), 2017.
[20] G. Andrade, G. Ramalho, H. Santana and V. Corruble, "Challenge-Sensitive Action Selection: an Application to Game Balancing", IEEE/WIC/ACM International Conference on Intelligent Agent Technology.
[21] J. Platt, Advances in neural information processing systems 20. La Jolla, California: Neural Information Processing Systems Foundation, 2009, pp. 569-576.
[22] N. Ravaja, "Contributions of Psychophysiology to Media Research: Review and Recommendations", Media Psychology, vol. 6, no. 2, pp. 193-235, 2004.
[23] . Allanson and S. Fairclough, "A research agenda for physiological computing", Interacting with Computers, vol. 16, no. 5, pp. 857-878, 2004.
[24] Tijs, T., Brokken, D. and IJsselsteijn, W., 2008. Dynamic Game Balancing by Recognizing Affect. Fun and Games, pp.88-93.
[25] A. Stein, Y. Yotam, R. Puzis, G. Shani and M. Taieb-Maimon, "EEG-triggered dynamic difficulty adjustment for multiplayer games", Entertainment Computing, vol. 25, pp. 14-25, 2018.
[26] Medeiros, R. and Medeiros, T., 2014. Procedural Level Balancing in Runner Games. 2014 Brazilian Symposium on Computer Games and Digital Entertainment.
[27] Christyowidiasmoro, Putra, R. and Susiki, S., 2015. Measuring level of difficulty in game using challenging rate (CR) on 2D Real time Strategy Line Defense game. 2015 International Electronics Symposium (IES).

[28] S. Chang and N. Yang, ”DCA: Dynamic Challenging Level Adapter for Real-time Strategy Games”, 2012 IEEE 15th International Conference on Computational Science and Engineering, 2012. Available: 10.1109/iccse.2012.15 [Accessed 20 September 2020].

[29] Silva, M., Silva, V. and Chaimowicz, L., 2017. Dynamic difficulty adjustment on MOBA games. Entertainment Computing, 18, pp.103-123.

[30] G. van Lankveld, P. Spronck and M. Rauterberg, ”Difficulty Scaling through Incongruity”, AIIDE, 2008.

[31] Tanomaru, J., 1995. Motivação, Fundamentos e Aplicações de Algoritmos Genéticos. II Congresso Brasileiro de Redes Neurais, III Escola de Redes Neurais.

[32] Linden, R., 2012. Algoritmos Genético. 3rd ed. Ciência Moderna.

[33] Rogers, S., 2014. Level Up!. 2nd ed. Chichester, UK: John Wiley Sons.

[34] Whitley, D., 1994. A genetic algorithm tutorial. Statistics and Computing, 4(2).

[35] C. Gaglioni, ”Qual é o lugar dos tablets 10 anos após o lançamento do iPad”, Nexo Jornal, 2020. [Online]. Available: https://www.nexojornal.com.br/expresso/2020/01/28/Qual-é-o-lugar-dos-tablets-10-anos-após-o-lançamento-do-iPad. [Accessed: 05- May-2020].