# Learning to Speed Up Evolutionary Content Generation in Physics-based Puzzle Games

Leonardo T. Pereira and Claudio Toledo
Institute of Mathematics and Computer Science
University of São Paulo
São Carlos, Brazil
{leonardop,claudio}@icmc.usp.br

Lucas N. Ferreira
Augmented Design Lab.
University of California, Santa Cruz
Santa Cruz, CA, USA
lferreira@ucsc.edu

Levi H. S. Lelis
Departamento de Informática
Universidade Federal de Viçosa
Viçosa, Brazil
levi.lelis@ufv.br

***Abstract*—Procedural content generation (PCG) systems are designed to automatically generate content for video games. PCG for physics-based puzzles requires one to simulate the game to ensure feasibility and stability of the objects composing the puzzle. The major drawback of this simulation-based approach is the overall running time of the PCG process, as the simulations can be computationally expensive. This paper introduces a method that uses machine learning to reduce the number of simulations performed by an evolutionary approach while generating levels of Angry Birds, a physics-based puzzle game. Our method uses classifiers to verify the stability and feasibility of the levels considered during search. The fitness function is computed only for levels that are classified as stable and feasible. An approximation of the fitness that does not require simulations is used for levels that are deemed as unstable or unfeasible by the classifiers. Our experiments show that naively approximating the fitness values can lead to poor solutions. We then introduce an approach in which the fitness values are approximated with the average fitness value of the levels' parents added to a penalty value. This approximation scheme allows the search procedure to find good-quality solutions much more quickly than a competing approach—we reduce from 43 to 25 minutes the running time required to generate one level of Angry Birds.**

## Introduction

Procedural content generation (PCG) for games is used since the early days of the video games industry. For example, the 80's games of Rogue and Elite use algorithms to generate dungeons and planets during gameplay. The initial motivation to automatic content generation was the memory limitation imposed by the machines of that time. By using PCG the games would have to store only a few lines of code to be able to generate a very large number of game components such as dungeons and planets. In addition to reducing the memory requirements, this dynamic approach to content generation can enhance the players' enjoyment by offering them a possibly different experience each time they play the game. The ability to enhance the player experience is what motivated modern games such as Minecraft and Spelunky to also use PCG.

Another motivation for using PCG methods is their support to the generation of user-tailored content, which can be achieved by collecting data during game sessions. This data is then used to adapt the content generated for future sessions [1]. PCG has also been used as tools to assist game developers to produce content more quickly. For example, the development of the game Borderlands was assisted by a tool for generating models of weapons. This tool allowed the developers to create approximately 17 millions unique weapons for the game [2].

PCG methods have to consider expressivity and feasibility aspects of the content generated [3]. Often PCG methods use simulations with artificial agents to ensure feasibility of the content generated. For example, in the work of Liapis et al. [4] dungeons are generated by a genetic algorithm and evaluated by an artificial agent that verifies if the dungeons are passable.

A recent problem in PCG is the generation of levels for physics-based puzzle games, where simulations are critical for evaluating feasibility [5]. Several works studied the level generation problem in the context of the game of Angry Birds (AB) [6]–[12]. In AB the player's goal is to destroy pigs by throwing birds with a slingshot placed on the left-hand side of the level. Pigs are distributed on the level altogether with blocks that can be piled up to form structures. A screenshot of a typical level of a clone of AB is shown in Figure 1.

Level generation for games such as AB poses several challenges to PCG algorithms such as choosing a set of objects to compose the level (blocks, pigs, and birds) and placing the objects in a way that it will be challenging and interesting to play. While choosing and placing the set of objects, one has to ensure that the level is feasible and stable. In the context of AB, we say that a level is feasible if there is a sequence of shots that will result in a level clear of pigs, and it is stable if no objects fall unless as a result of a shot. This paper builds upon the evolutionary algorithm presented by Ferreira [8], which handles these two problems through simulations.

The major problem of a simulation-based approach to PCG is the system's resulting running time, as simulations can be computationally expensive. For example, simulations are responsible for approximately 97% of the running time of the evolutionary approach described by Ferreira [8].

In this paper we introduce a method that uses machine learning to reduce the number of simulations performed by Ferreira's evolutionary approach while generating AB levels. Our method uses classifiers to verify stability and feasibility of the levels considered during search. The fitness function is computed only for levels that are classified as stable and feasible. An approximation of the fitness that does not require

simulations is used for levels that are classified as unstable or unfeasible. Our experiments on a clone of the game of AB [7] show that naively approximating the fitness values can lead to poor solutions. We then introduce an approach in which the fitness values are approximated with the average fitness value of the levels' parents added to a penalty value. This approximation scheme allows the search procedure to find good-quality solutions much more quickly than Ferreira's method. Namely, we reduce from 43 to 25 minutes the running time required to generate one level of AB.

## Related Work

This paper relates to PCG methods that use simulations to evaluate the quality or feasibility of the content generated. This paper also relates to works that use machine learning to enhance the search procedure of evolutionary algorithms.

Simulations have been used to evaluate the quality of content in several games. For example, [13] presents an evolutionary approach for generating racing tracks that can provide a large degree of diversity while also being adequately challenging. In that work, the fitness of a track is calculated based on the track's curvature and speed profile, which is calculated by artificial agents that test the tracks generated.

Another genre in which simulation-based PCG was employed is First Person Shooters (FPS). Cardamone et al. [14] proposed an evolutionary approach to generate maps of FPS games. The maps' quality was measured by the average playing time. This metric assumes that the map's quality is directly linked to the playing time. By maximizing the average playing time, the PCG system tends to generate larger maps, which is intuitively good as very small maps do not leave enough space for the placement of weapons and spawning locations. A level's average playing time is approximated with a 10-minute match simulation that uses artificial agents.

Board games can be automatically designed by the method proposed by Browne and Maire [15], which evaluates the games with a Minimax search. A number of metrics are extracted from investigating the performance of the Minimax search on the game, including how long it takes to finish the game, how often the game ends in a draw, and how many game rules are used. These values are combined in a weighted sum of the metrics where the weights are based on empirical investigations of the properties of successful board games.

Generation of levels for physics-based games has been recently explored in the context of Angry Birds, which is a game with simple mechanics but that supports the study of complex problems such as the stability of stacked blocks, the presence of noise in simulation-based fitness functions, and the development of intelligent agents for solving physics-based puzzles. In this context, Stephenson and Renz [11] presented a level generator which can create stable structures using a variety of objects. The proposed algorithm creates levels consisting of various self-contained structures placed throughout a 2D area. The authors argue that once they have access to all the physical attributes of the objects composing the level, it is possible to verify whether the level is stable or not by solving a system of linear equations [16]. However, since physics engines are usually imprecise due to simplifications, levels that are theoretically stable might collapse when simulated in an engine. Thus, Stephenson and Renz also use a simulation to evaluate the stability of the generated levels.

Another level generator of Angry Birds levels that uses simulations is presented by Ferreira and Toledo [7]. This generator is based on a genetic algorithm where the fitness function penalizes unstable levels and tries to generate levels with at least one pig and an amount of birds desired by the user. This algorithm was extended by Ferreira [8], where an encoding scheme allowed the evolutionary approach to better control the number of birds in the levels generated. Also, a novel fitness function was proposed where levels which are deemed as unfeasible by an artificial agent are penalized. We describe Ferreira's algorithm in detail below.

The idea of using machine learning to improve the performance of evolutionary algorithms was also explored by others. For example, Handoko et al. [17], [18] use a support vector machine classifier to enhance a constrained memetic algorithm by helping it locate the objective function's global optimum.

Machine learning has been used in several contexts in computer games. For example, Weber and Mateas [19] applied machine learning algorithms to learn and predict enemies' strategies; Weber et al. [20] modeled player experiences during gameplay to support content creation. Lanzi et al. [21] used data mining algorithms to discover interesting design patterns, flaws, and opportunities to improve a mobile game with few hours of play-testing. A machine learning algorithm proposed by Fink et al. [22] correctly predicts non-player character behaviors in the game of Pong with minimum game knowledge.

## Original Evolutionary Approach

In the present section we review the evolutionary approach of Ferreira and Toledo to automatically generate Angry Birds levels [7], [8]. First, we present their level encoding and then we discuss the fitness function they employ.

### *Level Encoding*

We review Ferreira and Toledo's level encoding with an example. Consider in Figure 1 an AB level, its encoding (in evolutionary computation terms a level is referred as an individual and in this paper we use the words level and individual interchangeably) accounts for the number of birds and sequences of stacked-up blocks. Each block is defined as a pair of integers $(x, y)$, where $x$ represents the type of block and $y \in \{0, 1\}$ indicates whether a block is duplicated (1) or not (0) in the stack. A block is duplicated when a stack contains two blocks of a given type side by side, as shown by Stack 2 in Figure 1. We refer the reader to the work of Ferreira and Toledo for details regarding the block types. The encoding also allows for empty spaces in between stacks; empty spaces are marked with an "X" in Figure 1.

| | | | | | | | | |
|---|---|---|---|---|---|---|---|---|
| | | (31, 0) | | | | (31, 0) | | |
| | (31, 0) | (13, 0) | | | | (13, 0) | | |
| | (13, 0) | (28, 0) | | | | (28, 0) | | |
| | (12, 0) | (29, 1) | | | | (29, 0) | | |
| | (13, 1) | (10, 1) | | (31, 0) | | (15, 0) | | (31, 0) |
| 4 | (11, 0) | (24, 1) | X | (18, 0) | X | (25, 1) | X | (18, 0) |
| Birds | Stack 1 | Stack 2 | Stack 3 | Stack 4 | Stack 5 | Stack 6 | Stack 7 | Stack 8 |

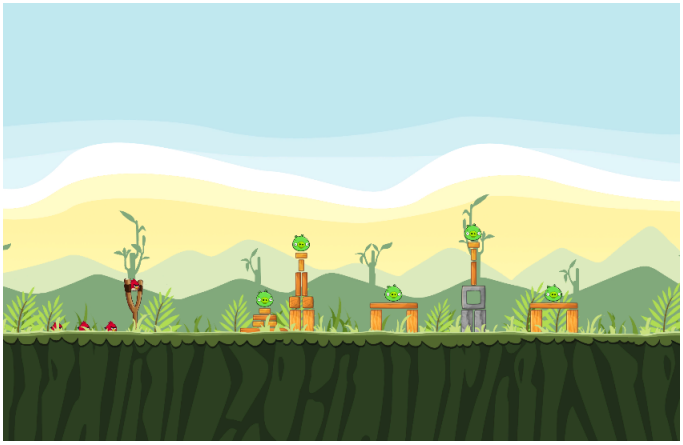

Fig. 1. An example of an Angry Birds level and its representation [8].

*Fitness Function*

Another key component in Ferreira and Toledo's approach is the fitness function employed, which the evolutionary optimization procedure tries to minimize. For a given set of objects $O$ defining an individual, $O$'s fitness value is defined as,

$$F(O) = |B_u - B_d(O)| + |L_u - L_d(O)| + P_l(O) + V(O)\,. \quad (1)$$

Here, $B_u$ and $L_u$ are input parameters defined by the user representing the desired number of birds and the number of blocks the generated level must have, respectively. All the other variables in Equation 1 have their values defined by a simulation with an artificial agent that plays $O$ (refer to [8] for details on the strategy employed by the agent to play the game). $B_d(O)$ returns the number of birds thrown by an artificial agent during the simulation and $L_d(O)$ returns the amount of blocks in the level before the simulation. Thus, the first term of $F$ penalizes $O$ if the artificial agent does not use all $B_u$ birds specified by the user. The second term penalizes $O$ if it does not have the desired number of blocks $L_u$.

If the artificial agent finishes the level by throwing fewer bids than $B_d$, then the individual is penalized by the first term of $F$. If the artificial agent does not finish the level even after throwing all birds (i.e., there are pigs left in the level), then the individual is penalized by the third term of $F$, $P_l(O)$, which is the number of pigs remaining in the level by the end of the simulation. Finally, the last term of $F$ measures the level stability: $V(O)$ is the sum of the average speed of all objects in $O$ before the first bird is shot. If the level is stable, then $V(O) = 0$. However, in practice, due to imprecisions of the physics simulator, we consider a level stable if $\sum_{o \in O} v_o < 10^{-5}$, where $v_o$ is the average speed of object $o$.

The EA approach of Ferreira and Toledo also includes a mutation and a crossover procedure for inserting diversity to the process. We refer the reader to their original work for details of these operations [7]. For the current work it is important to highlight that in order to compute the $F$-value of a given individual, one has to employ an artificial agent to play the game to verify the level's feasibility and a physics simulator to verify the level's stability. Since the EA approach has to compute $F$ for all individuals evaluated during the optimization procedure, the resulting level generation process can be very slow. On average, it takes Ferreira and Toledo's approach 43 minutes to generate an AB level, and approximately 97% of this time is spent on simulations.

## Learning to Simulate

We now describe our approach to decrease the running time of Ferreira and Toledo's evolutionary approach to AB level generation. Instead of simulating gravity to verify stability and executing an artificial agent to estimate feasibility, we train two classifiers to bypass, when possible, these two computationally expensive steps. The first classifier approximates the level's stability, and the second the level's feasibility.

We use *connected structures* instead of complete AB levels to train the stability classifier. We define a connected structure as follows. Let $G = (V, E)$ be a graph describing an AB level $O$, where every vertex $v \in V$ represents an object in $O$, and there is an edge $(v_1, v_2) \in E$ between vertices $v_1$ and $v_2$ if the objects represented by $v_1$ and $v_2$ are in physical contact. We say that a subset $V' \subseteq V$ is a connected structure if for any pair of vertices $v_1, v_2 \in V'$, there is a path between $v_1$ and $v_2$. Figure 1 shows four connected structures. From left to right, the first connected structure holds two pigs, and the other three connected structures hold one pig each. We train our classifier on connected structures instead of complete AB levels because we believe the task of learning stability of smaller structures to be easier than the task of learning the stability of more complex structures.

In order to train the stability and the feasibility classifiers we generated two training sets. The stability dataset was generated by using the original EA described above to generate 5,567 connected structures. The labels of the dataset (stable or unstable) were computed through simulations; 71.2% of the connected structures in the dataset were unstable. The feasibility dataset was also generated by the original EA and contained 4,553 levels. The labels (feasible or unfeasible) were computed through simulations with Ferreira and Toledo's artificial agent; 76.7% of the levels were unfeasible.

We note that we disregard the time spent in the process of generating these datasets of levels and connected structures. This is because this step is performed only once and its computational cost can be amortized by using the trained classifiers to generate a large number of AB levels.

Also aiming at facilitating learning, we modify the EA encoding proposed by Ferreira and Toledo. In the original encoding different levels could have different representation sizes. Since the differences in level size could be a problem

TABLE I
ENCONDING OF THE LEVEL SHOWN IN FIGURE 1 USING OUR MATRIX REPRESENTATION

| | | | | | | | |
|---|---|---|---|---|---|---|---|
| -1 | -1 | -1 | -1 | -1 | -1 | ··· | -1 |
| ··· | ··· | ··· | ··· | ··· | ··· | ··· | ··· |
| -1 | -1 | 31 | -1 | -1 | -1 | ··· | -1 |
| 31 | -1 | 13 | -1 | -1 | -1 | ··· | -1 |
| 13 | -1 | 28 | -1 | -1 | -1 | ··· | -1 |
| 12 | -1 | 29 | 29 | -1 | -1 | ··· | -1 |
| 13 | 13 | 10 | 10 | 31 | -1 | ··· | -1 |
| 11 | -1 | 24 | 24 | 18 | -1 | ··· | -1 |

to supervised learning algorithms, we use a matrix $\mathbf{M}$ of fixed size as an encoding scheme to AB levels. An entry $(i, j)$ of $\mathbf{M}$ corresponds to a position of a block in the level and the value stored in $(i, j)$ corresponds to the type of the block occupying that position. If object $o_1$ occupies position $(i, j)$ and object $o_2$ occupies position $(i + 1, j)$ in $\mathbf{M}$, then $o_2$ is on top of $o_1$ in the level. Similarly, if object $o_1$ occupies position $(i, j)$ and object $o_2$ occupies position $(i, j + 1)$ of $\mathbf{M}$, then $o_2$ is on the right-hand side of $o_1$ in the level. If a given position is empty, then that entry has value of -1. Our matrix has size $M \times N$, where $M$ is the maximum number of blocks one can place on top of each other, and $N$ is the maximum number of side-by-side objects a level can have. For the feasibility classifier we add an integer to the encoding to specify the number of birds the level has. As an example, Table I shows the matrix representation of the level shown in Figure 1.

## EMPIRICAL EVALUATION OF CLASSIFIERS

In this section we empirically test the accuracy of four supervised learning algorithms for the task of classifying stable and feasible AB levels. We test the following algorithms: Decision Trees (J48), Bayesian Network (BN), Radial Basis Function Neural Network (RBF), and Random Forests (RF). The input parameters of each algorithm is chosen in a 10-fold cross validation process. We present in Table II the cross validation results for the best set of input parameters for each algorithm. The results are presented in terms of accuracy (percentage of levels or structures correctly classified) and mean absolute error (MAE). The best results are highlighted in bold; the numbers are truncated to two decimal places.

The tree-based approaches performed better than BN and RBF in our experiments, with the RF yielding better results overall for both the feasibility and the stability classification tasks. The accuracy of RF is above 90% in both tasks, which indicates the potential of employing these classifiers as part of the PCG process for generating AB levels.

We recall that RF's accuracy of 97% in the feasibility task is with respect to an approximation to the actual feasibility of the levels tested. This is because the labels (feasible or unfeasible) of our training data was obtained by approximating the levels' feasibility with a simple artificial agent. If our artificial agent classifies a given level $O$ as feasible, then $O$ is certainly feasible as the agent was able to complete $O$. However, if the agent classifies $O$ as unfeasible, $O$ could still be feasible but the agent was not able to complete the level.

TABLE II
RESULTS FROM THE CREATION OF THE FEASIBILITY CLASSIFIER MODELS WITH A BASE CONTAINING 4,553 INDIVIDUALS AND THE STABILITY CLASSIFIER MODELS WITH A BASE CONTAINING 5,567 INDIVIDUALS

| Feasibility Results | | | | |
|---|---|---|---|---|
| | J48 | BN | RBF | RF |
| Accuracy (%) | 95.94 | 78.32 | 81.92 | **97.27** |
| MAE | **0.05** | 0.22 | 0.29 | 0.08 |
| **Stability Results** | | | | |
| | J48 | BN | RBF | RF |
| Accuracy (%) | 86.62 | 67.27 | 66.70 | **91.93** |
| MAE | 0.16 | 0.35 | 0.43 | **0.13** |

**Algorithm 1** Classifier-Based Evolutionary Approach

**Require:** Maximum number of generations $E$, maximum number of generations without improvement $I$, stability $C_s$ and feasibility $C_f$ classifiers, desired number of birds $B_d$, desired number of blocks $L_d$, and penalty value $\lambda$.
**Ensure:** One Angry Birds level

```
Generate a population Γ of 100 random individuals with at most 5 birds and at most 100 blocks.
e ← 0
while number of generations without improving the F-value has not exceeded I, e has not exceeded E, and did not find an individual O with F(O) = 0 do
    Initialize next population Γ' ← ∅
    while |Γ'| < |Γ| do
        Randomly select individuals O1 and O2 from Γ.
        if F(O1) < F(O2) then
            P1 ← O1
        else
            P1 ← O2
        Repeat previous step to select P2.
        Reproduce P1 with P2 generating children O'1 and O'2 (crossover rate of 95% and mutation rate of 10%).
        Compute F(O'1) and F(O'2) using Algorithm 2.
        Γ' ← Γ' ∪ {O'1, O'2}
    Γ ← Γ'
    e ← e + 1
return argmin_{O∈Γ} F(O)
```

Since RF was able to perform best in our experiments, RF is the method we use in the EA PCG system described next.

## CLASSIFIER-BASED EVOLUTIONARY APPROACH

Our evolutionary approach to AB level generation follows Ferreira and Toledo's approach, with the difference that we employ two RF classifiers to reduce the number of simulations required to compute the fitness values during search. Our approach is described in Algorithms 1 and 2.

Algorithm 1 receives as input the maximum number of generations $E$ the EA is allowed to create, and the maximum number $I$ of generations the EA is allowed to create without improving the $F$-value of the fittest individual in the population; the algorithm returns an AB level. Our EA approach initially generates a population $\Gamma$ of 100 random

individuals with at most 5 birds and 100 blocks. These values were chosen empirically in preliminary experiments. Variable $e$ counts the number of generations (line 2); the counter for generations without improvement is not shown explicitly in the pseudocode. The stopping condition of our EA approach is shown in line 3 of Algorithm 1. We stop the procedure (i) after completing $E$ generations; (ii) after completing $I$ generations without improving the best $F$-value within the current population; (iii) or after finding an optimal solution (i.e., $F$-value of zero).

In each generation (denoted by an iteration of the outer loop of Algorithm 1) we create a new population $\Gamma'$ with the same number of individuals of the current population $\Gamma$. We generate two individuals $O'_1$ and $O'_2$ with a tournament selection approach. That is, we randomly select two individuals $O_1$ and $O_2$ from $\Gamma$ and select the one with lowest $F$-value to be parent $P_1$; we repeat the same process to select a parent $P_2$. Next, $P_1$ reproduces with $P_2$ generating individuals $O'_1$ and $O'_2$, which are added to $\Gamma'$ (see lines 6–14 in Algorithm 1).

We compute the $F$-values for all individuals generated (line 13) according to Algorithm 2. Note that for the initial random population we compute the $F$-values according to Equation 1 directly as the individuals in that population do not have parents, which is required as input in Algorithm 2.

In addition to individual $O$ and its parents $O_{p_1}$ and $O_{p_2}$, Algorithm 2 requires the stability $C_s$ and feasibility $C_f$ classifiers as input. $C_s(O)$ and $C_f(O)$ return true if $O$ is stable and feasible, respectively. Algorithm 2 also receives as input the desired number of birds $B_d$ and blocks $L_d$, which are chosen by the user, and a penalty value $\lambda$.

If both $C_s$ and $C_f$ return true (line 1), we compute $F(O)$ with Equation 1 directly. That is, we run the stability simulator to obtain $V(O)$ and the feasibility simulator through the artificial agent to obtain $L_u(O)$ and $P_l(O)$ (lines 2–4). Our method does not compute the actual value of $F(O)$ if according to the classifiers $O$ is either unstable or unfeasible. Instead, Algorithm 2 returns the average of $O$'s parent added to the penalty $\lambda$. The value of $\lambda$ reduces the chances of $O$ winning a tournament and passing its genes to the next generation of the evolutionary approach. Note, however, that it might be important to be able to distinguish amongst different unstable individuals and amongst different unfeasible individuals. This is because, for example, there could be two unstable individuals where one of them is much closer to being stable than the other (i.e., a small modification to the near-stable individual would make it stable). By using the average $F$-value of $O$'s parents added to $\lambda$ as $O$'s $F$-value we expect to distinguish near-stable individuals from far-from-stable ones and near-feasible individuals from far-from-feasible ones.

## EMPIRICAL RESULTS

We are primarily interested in comparing the running time of our proposed approach with that of the original method (Original). In order to show that we are able to reduce the running time while generating levels with similar quality, we also show the average fitness value of the level generated by each approach.

**Algorithm 2** Classifier-Based Fitness Function

**Require:** Individual $O$ and its parents $O_{p1}$ and $O_{p2}$, stability $C_s$ and feasibility $C_f$ classifiers, desired number of birds $B_d$, desired number of blocks $L_d$, and penalty value $\lambda$.
**Ensure:** Fitness value of $O$.
1: **if** $C_s(O) \wedge C_f(O)$ **then**
2: Obtain $L_u(O)$ and $P_l(O)$ with artificial agent
3: Obtain $V(O)$ with physics simulator
4: return $|L_u(O) - L_d| + |B_u(O) - B_d| + P_l(O) + V(O)$
5: **else**
6: return $\frac{F(O_{p1})+F(O_{p2})}{2} + \lambda$

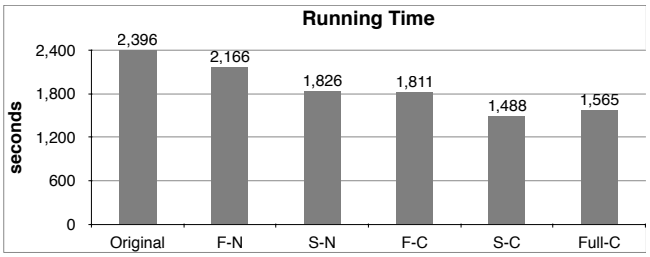


Fig. 2. Average running time of each approach.

In addition to Original and the approach described in Algorithms 1 and 2, which we name Full-C, we also test the following Full-C variants: a method that uses only the stability classifier in line 1 of Algorithm 2 (S-C) and a method that uses only the feasibility classifier in line 1 of Algorithm 2 (F-C). We also test naive versions of S-C and F-C in which we use $\lambda$ instead of the average fitness value of $O$'s parents added to $\lambda$ as the fitness value of $O$ when the expression of line 1 of Algorithm 2 returns false. We call such naive variants S-N and F-N, for S-C and F-C, respectively.

All results presented are averages over 10 independent runs of each method. Also, all runs were executed in the same machine. Each approach was tested with the same set of parameters: $|\Gamma| = 100$, $E = 25$, $I = 10$, $\lambda = 100$, $B_d = 5$, and $L_d = 100$, crossover and mutation rates of 95% and 10%.

The results for running time are presented in Figure 2. All proposed approaches but F-N present a substantial reduction in running time compared to Original. A Pearson correlation test suggests that the running time data is likely to be independent (correlation of $0.051$); Levene's test for homoscedasticity suggests that the hypothesis that the variables have equal variance cannot be discarded ($p > 0.05$); finally, Anderson-Darling suggests that normality cannot be discarded ($p > 0.05$). Thus, we apply the parametric ANOVA test which yields $p = 0.008$, suggesting a significant difference between the average running time of different systems. Then, we use Fisher's test for pairwise comparisons to discover that both S-C and Full-C are significantly faster than Original ($p < 0.05$).

The results for the average fitness value of the individual

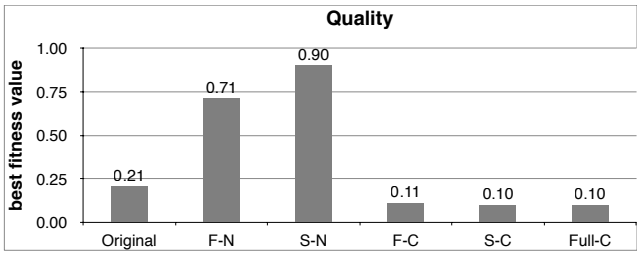


Fig. 3. Average fitness value of the individual returned by each approach.

returned by each approach are shown in Figure 3. The average fitness values do not satisfy the three criteria to apply a parametric test. Thus, the non-parametric Friedman test is applied instead. The Friedman test yields $p = 0.165$, showing that no statistical difference was found.

## DISCUSSION

The running time results shown in the previous section demonstrated that our proposed approach is able to substantially and significantly reduce the running time of the original evolutionary approach to level generation in the game of AB. Namely, S-C and Full-C reduced from 2,396 seconds to 1,488 and 1,565 seconds, respectively, the running of the PCG procedure. The naive approach which employs only the stability classifier (S-N) was also able to substantially reduce the running time—from 2,396 seconds to 1,826 seconds. However, the average fitness values of the individuals returned by the naive approaches were much worse than the original method. That is, while the original method returns individuals with fitness value of 0.21 on average, the naive approaches return individuals with fitness values of 0.71 and 0.90 on average. Thus, although the naive approaches have better running time than Original, they produce levels of lower quality.

By contrast, F-C, S-C, and Full-C have substantially better running time than Original and they generate levels of similar quality. This result shows that the strategy of using the average fitness value of the parents of an individual added to the penalty $\lambda$ instead of only $\lambda$ allows the evolutionary approach to generate solutions of much better quality. We conjecture that, by returning only $\lambda$ as the fitness value of the individuals for which the classifier returns false, F-N and S-N are unable to distinguish the near-stable and near-feasible levels from those that are far from becoming stable or feasible. We believe that being able to distinguish these levels is specially important in the early stages of the evolutionary procedure. This is because the initial random population $\Gamma$ is composed almost solely by unstable and unfeasible levels. This way, in the naive approaches, all individuals in the early populations have the same $F$-value, which might misguide the search procedure to poor solutions.

Another approach that could be used to reduce the running time of our evolutionary approach is the employment of an exact and fast method such as Blum et al.'s [16] to verify the stability of a given level. The drawback of such an approach is that the physical engine in which the levels are going to be played only approximates the real-world physics and are imperfect. This way, a level that is theoretically stable according to Blum et al.'s method might be unstable once simulated in the physics engine. Our method learns even the imperfections of the physics engine.

We believe there are other ways of further improving the running time of our evolutionary approach. One interesting direction of future work is to test other forms of evolutionary operations that do not require the computation of the $F$-value for all individuals in $\Gamma$ as is currently done by our approach.

## CONCLUSIONS

The presented work employs machine learning techniques to reduce the running time of a procedural content generation system used to generate levels of the physics-based puzzle game of Angry Birds. The original approach for generating content for Angry Birds heavily relies on simulations to evaluate both the stability and the feasibility of the levels during the search procedure. The drawback of solely using simulations is the resulting computational cost of the fitness function used to guide the search. In our approach we only use simulations if two classifiers predict that the level is both stable and feasible. This way we are able to substantially and significantly reduce the running time of the original evolutionary approach to generate Angry Birds levels.

We also showed that naively applying a penalty value to the individuals which are classified as either unstable or unfeasible can lead the search procedure to poor solutions. To overcome this problem, instead of using a penalty value as the individual's fitness, we use the average fitness value of the parents of the individual added to a penalty value. This way the search procedure is able to distinguish the near-stable and near-feasible levels from those with worse quality. As a result of our strategy, our system is able to find good solutions much more quickly than the original approach.

## ACKNOWLEDGMENTS

Financial support for this research was in part provided by CAPES/CNPq Science Without Borders program, FAPEMIG, and Gapso. Lucas N. Ferreira acknowledges financial support from CNPq Scholarship (200367/2015-3).

## REFERENCES

[1] G. N. Yannakakis and J. Togelius, "Experience-driven procedural content generation," *IEEE Transactions on Affective Computing*, vol. 2, no. 3, pp. 147–161, 2011.

[2] G. Smith, E. Gan, A. Othenin-Girard, and J. Whitehead, "Pcg-based game design: enabling new play experiences through procedural content generation," in *Proceedings of the 2nd International Workshop on Procedural Content Generation in Games*. ACM, 2011, p. 7.

[3] A. Liapis, G. N. Yannakakis, and J. Togelius, "Constrained novelty search: A study on game content generation," *Evolutionary computation*, vol. 23, no. 1, pp. 101–129, 2015.

[4] A. Liapis, C. Holmgård, G. N. Yannakakis, and J. Togelius, "Procedural personas as critics for dungeon generation," in *European Conference on the Applications of Evolutionary Computation*. Springer, 2015, pp. 331–343.

[5] N. Shaker, M. Shaker, and J. Togelius, "Evolving playable content for cut the rope through a simulation-based approach." in *AIIDE*, G. Sukthankar and I. Horswill, Eds. AAAI, 2013. [Online]. Available: http://dblp.uni-trier.de/db/conf/aiide/aiide2013.html#ShakerST13

[6] L. Ferreira and C. Toledo, "Generating levels for physics-based puzzle games with estimation of distribution algorithms," in *Proceedings of the 11th International Conference on Advances in Computer Entertainment*, ser. ACE'14, 2014. [Online]. Available: http://www.lucasnferreira.com/papers/2014/ace-edaab.pdf

[7] L. Ferreira and C. Toledo, "A search-based approach for generating angry birds levels," in *Proceedings of the IEEE International Conference on Computational Intelligence in Games*, 2014.

[8] L. N. Ferreira, "An evolutionary approach for procedural generation of levels in physics-based puzzle games," Master's thesis, University of Sao Paulo, Sao Carlos, Sao Paulo, Brasil, 2015.

[9] Y. Jiang, M. Kaidan, C. Y. Chu, T. Harada, and R. Thawonmas, "Procedural generation of angry birds levels using building constructive grammar with chinese-style and/or japanese-style models," *arXiv preprint arXiv:1604.07906*, 2016.

[10] R. Lara-Cabrera, A. Gutierrez-Alcoba, and A. J. Fernández-Leiva, "A spatially-structured pcg method for content diversity in a physics-based simulation game," in *European Conference on the Applications of Evolutionary Computation*. Springer, 2016, pp. 653–668.

[11] M. Stephenson and J. Renz, "Procedural generation of complex stable structures for angry birds levels," in *IEEE Computational Intelligence and Games Conference*, 2016.

[12] M. Stephenson and J. Renz, "Procedural generation of levels for angry birds style physics games," in *The AAAI Conference on Artificial Intelligence and Interactive Digital Entertainment*, 2016.

[13] D. Loiacono, L. Cardamone, and P. L. Lanzi, "Automatic track generation for high-end racing games using evolutionary computation," *IEEE Transactions on Computational Intelligence and AI in Games*, vol. 3, no. 3, pp. 245–259, 2011.

[14] L. Cardamone, G. N. Yannakakis, J. Togelius, and P. L. Lanzi, "Evolving interesting maps for a first person shooter," in *European Conference on the Applications of Evolutionary Computation*. Springer, 2011, pp. 63–72.

[15] C. Browne and F. Maire, "Evolutionary game design," *IEEE Transactions on Computational Intelligence and AI in Games*, vol. 2, no. 1, pp. 1–16, 2010.

[16] M. Blum, A. Griffith, and B. Neumann, "A stability test for configurations of blocks," 1970.

[17] S. D. Handoko, K. C. Keong, and O. Y. Soon, "Using classification for constrained memetic algorithm: A new paradigm," in *Systems, Man and Cybernetics, 2008. SMC 2008. IEEE International Conference on*, Oct 2008, pp. 547–552.

[18] S. D. Handoko, C. K. Kwoh, and Y. S. Ong, "Feasibility structure modeling: An effective chaperone for constrained memetic algorithms," *IEEE Transactions on Evolutionary Computation*, vol. 14, no. 5, pp. 740–758, Oct 2010.

[19] B. G. Weber and M. Mateas, "A data mining approach to strategy prediction," in *2009 IEEE Symposium on Computational Intelligence and Games*, Sept 2009, pp. 140–147.

[20] B. G. Weber, M. Mateas, and A. Jhala, "Using data mining to model player experience," in *FDG Workshop on Evaluating Player Experience in Games*, ACM. Bordeaux, France: ACM, 06/2011 2011.

[21] P. L. Lanzi, D. Loiacono, E. Parini, F. Sannicoló, D. Jones, and C. Scamporlino, "Tuning mobile game design using data mining," in *2013 IEEE International Games Innovation Conference (IGIC)*, Sept 2013, pp. 122–129.

[22] A. Fink, J. Denzinger, and J. Aycock, "Extracting npc behavior from computer games using computer vision and machine learning techniques," in *2007 IEEE Symposium on Computational Intelligence and Games*, April 2007, pp. 24–31.