Generating Levels for Physics-based Puzzle Games with Estimation of Distribution Algorithms

Lucas Ferreira
Institute of Mathematics and Computer Science
University of São Paulo
São Carlos, Brazil
lucasnfe@icmc.usp.br

Claudio Toledo
Institute of Mathematics and Computer Science
University of São Paulo
São Carlos, Brazil
claudio@icmc.usp.br

ABSTRACT

This paper presents an estimation of distribution algorithm (EDA) to generate levels for physics-based puzzle games with the Angry Birds mechanics. The proposed EDA keeps three probability tables during its evolutionary process to sample new individuals that encode informations about the amount and placement of game objects inside the level. Sampled individuals are evaluated by a simulation-based fitness function, which considers the stability and the amount of the game objects inserted in a level. The best individual sampled from the probability tables is used to update them. Experiments indicated that the proposed EDA was capable of creating stable structures related to the Angry Bird gameplay.

Author Keywords

Procedural content generation; games; estimation of distribution algorithm; physics-based puzzle; Angry Birds.

ACM Classification Keywords

I.2.1 Artificial Intelligence: Applications and Expert Systems—*Games*; K.8 Personal Computing: [Games]

INTRODUCTION

Level generation is one of the most common types of procedural content generation (PCG) inside digital games [6]. The authors in [13] report several games using PCG either to allow a unique gameplay experience or to generate a huge amount of levels without actually store them in memory. For example, the dungeons in the classic game *Rogue* are generated dynamically every time a new game starts [8].

Besides being used in the game industry, procedural level generation (PLG) has been studied by several authors in the research community. Recently, there are a plenty of papers reporting the use of evolutionary algorithms for PLG in different game genres. For example, [1] generated tracks for racing games, [9] generated levels for platform games and [14] generated maps for the real time strategy game *Starcraft*.

Although PLG is one of the most common types of PCG, there are only a few works applying it to physics-based puzzle games [10, 11, 12]. This genre has recently become very popular, especially on mobile devices, and some of the main titles are Angry Birds, Cut the Rope and Tower of Goo. It is an interesting application for PCG, because it has several physics constraints to be considered when evaluating the quality of the content generated. Therefore, the playability evaluation is another issue once it needs to be done with a physics simulator [10].

This paper presents an estimation of distribution algorithm (EDA) for generating Angry Birds levels. EDAs are stochastic optimization techniques that explore the space of potential solutions building and sampling explicit probabilistic models of promising candidate solutions [5]. Its output is an estimation of the probability distribution from good solutions, which is typically represented by a probability array [7].

In EDAs for PCG, the probability array may represent the likelihood of a feature to occur inside the content generated. Thus, this approach may allow the user customizing the content generator, manually adjusting the values in the probability array to sample content with different characteristics. To the best of the authors' knowledge, there is no application of EDAs inside the PCG field. Therefore, this is the major contributions of this paper.

Angry Birds is a popular commercial physics-based puzzle game developed by Rovio Entertainment and released in 2009 for iOS devices [3]. This game has been a huge success, reaching the mark of 1 billion downloads in 2012 [4]. Thus, it was used to support the experiments in this paper. However, its source code is not available, so we implemented a clone [1] using Unity engine [15] and the original art assets. The experiments performed used metrics such as frequency, linearity and density to analyze the expressivity of the proposed algorithm. Results showed the proposed EDA can generate interesting stable structures in the levels.

THE ESTIMATION OF DISTRIBUTION ALGORITHM

Proposed EDA samples l individuals from a probability distribution represented by three tables, where l is a parameter of the algorithm. These l individuals are evaluated using the developed Angry Birds clone and the best one is used to update all the tables, which will sample l new individuals in the next iteration. First probability table has the likelihood of an object to occur in each sector of the level. Second table has the likelihood of the amount of objects stacked in a column.

[1] https://github.com/lucasnfe/AngryBirdsCover

ACE '14, November 11 - 14 2014, Funchal, Portugal.

http://dx.doi.org/10.1145/2663806.2663847

Third table stores the probability of horizontal distance between two columns to belong a determined range of values. The evolutionary process, described in the Algorithm 1, is executed until a number of generations have been reached.

Algorithm 1: The proposed EDA

1 $currentGeneration \leftarrow 0$;
2 InitializeProbabilityTables();
3 **while** $currentGeneration < maxGeneration$ **do**
4 SampleIndividuals();
5 EvaluateIndividuals();
6 $bestInd \leftarrow SelectBestIndividual()$;
7 UpdateProbabilityTables(bestInd);
8 $currentGeneration \leftarrow currentGeneration + 1$;
9 **end while**

The initialization stage (line 2) set all probability tables to represent a uniform distribution. While the maximum number of generations is not exceeded (lines 3), a number l of individuals are sampled using current probability tables (line 4). During the evaluation stage (lines 5), individuals are played using the Angry Bird clone. Physics information collected during the game is used by the fitness function to evaluate the l individuals. The best individual is selected (line 6) and used to update probability tables (lines 8).

Level Representation

An individual in the proposed algorithm represents an Angry Birds level encoded as an array of columns. Its size c is an user-defined parameter and it is the total amount of columns in the level. Each column is composed by zero or more game objects stacked, which are represented by integer values.

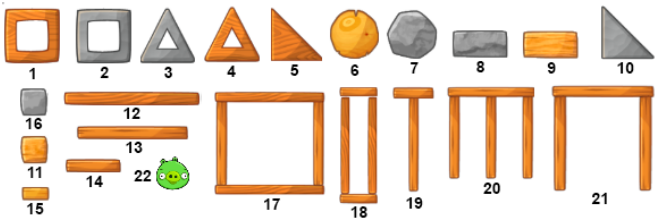

Figure 1. The 22 elements used to build levels.

There are three possible objects: an elementary block, a pig or a composed block. A composed block is a predefined structure made by elementary blocks. A total of 22 game objects are used to build levels as showed in Figure 1. Objects 17-21 are composed blocks, object 22 is the pig and the others are elementary blocks.

Horizontal distance between each column is also encoded in the individual and it is represented by an array with size $c-1$. An element $d_{min} \leq d_i \leq d_{max}$ of this array stands for the horizontal distance (in pixels) from the center of the column i to the center of the column $i+1$. Figure 2 illustrates a level and its representation.

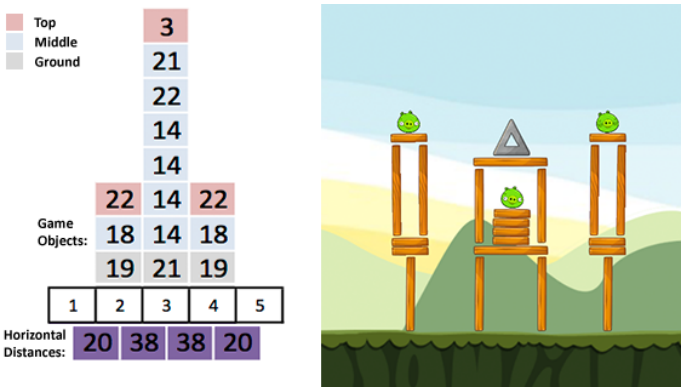

Figure 2. Individual representing a level an its expanded format.

Probability Tables

Proposed EDA uses three tables to represent the levels' probability distribution. First table represents the likelihood of a game object being placed in each sector of the level: ground, middle and top. It is used to control which objects will be placed in each sector of the level and it is useful for preventing unstable objects being placed in the ground or middle sectors.

Game Obj.	Ground	Middle	Top
1	0.03	0.3	0
2	0	0	0.02
3	0.17	0.1	0
4	0	0	0
5	0	0	0.18
6	0	0	0
7	0	0	0.1
8	0.05	0.12	0
9	0.05	0.18	0
10	0	0	0.05
11	0	0	0.15
12	0.1	0.2	0
13	0.1	0	0
14	0.1	0.1	0
15	0	0	0.1
16	0	0	0.1
17	0.1	0	0
18	0.1	0	0
19	0.1	0	0
20	0.08	0	0
21	0.02	0	0
22	0	0	0.3

Table 1. Example of a game objects' probability table.

First column represents the index i of the game objects. Second, third and fourth columns represent the probability of object i being placed in the first position (ground sector), intermediary position (middle sector) and in the last position (top sector) of a column, respectively. Note that the sum of each column must be one. Table 1 illustrates an example of game objects' probability table.

Second table represents the likelihood of amount of objects stacked in columns of the level. Its dimension is $c * (h_{max} +$

1), where c is the number of columns in the level and h_{max} is the maximum amount of objects in a column. (both values are parameters of the algorithm). An element $e_{i,j}$ in this table represents the probability of the amount j of objects stacked in the column i. Table 2 shows an example considering $c = 5$ and $h_{max} = 2$.

Objs. stacked	Col. 1	Col. 2	Col. 3	Col. 4	Col. 5
2	0.4	0.2	0	0	0.7
1	0.4	0.6	0.1	0	0.2
0	0.2	0.2	0.9	1.0	0.1

Table 2. Example of a probability table for objects stacked.

Third table represents the probability distribution of horizontal distances between columns. Distances between two columns are values inside the interval (d_{min}, d_{max}), which are method parameters. This table stores the likelihood of ranges that belong to (d_{min}, d_{max}). Range sizes are defined as $(d_{max} - d_{min})/r$, where r is a parameter that controls the size of ranges. Table 3 illustrates an example considering $c = 5$, $d_{min} = 0$, $d_{max} = 80$ and $r = 4$.

Dist.	1-2	2-3	3-4	4-5
[60,80]	0.25	0.0	0.1	0.2
[40,60)	0.25	0.0	0.0	0.4
[20,40)	0.25	0.5	0.1	0.2
[0,20)	0.25	0.5	0.8	1.2

Table 3. Example of a probability table for horizontal distances.

Sampling and Evaluating Levels

Levels are sampled column by column based on the three probability tables described in the last section. First, the amount of objects h_i for column i is randomly sampled considering probabilities in column i of the second table. If $i < c - 1$, the distance $d_{i,i+1}$ from column i to column $i + 1$ is randomly sampled considering probabilities in column i of the third table.

After defining the amount of objects stacked (h_i) for column i and the distance from column i to column $i+1$, all the game objects of column i of the level are randomly sampled using the probabilities in column j of the first table. Value j defines the sector of the current object: $j = 0$ is the ground sector, $1 \leq j \leq h_i - 1$ is the middle sector and $j = h_i$ is the top sector. Algorithm 2 describes this process.

During the evaluation stage of the proposed algorithm, the l sampled individuals are evaluated by a fitness function (1) that considers the average velocity of objects during the game and the total amount of blocks and pigs.

$$f_{ind} = \frac{1}{n}\sum_{i=0}^{n-1} v_i + \frac{\sqrt{(|b| - B)^2}}{Max_b - B} + \frac{1}{1 + |p|} \qquad (1)$$

In fitness function (1), $0 \leq v_i \leq 1$ is the average velocity of object i during the game, n is the total amount of objects (blocks and pigs), $|b|$ is the amount of blocks and $|p|$ is the amount of pigs. Parameter B defines the desired amount of blocks and it is set by the user. Max_b is the maximum amount of objects in a level and it is determined by the number of columns c and maximum amount of objects staked h_{max}.

Algorithm 2: Sampling new levels.

```
1  i ← 0;
2  while i < c do
3      h_i ← SampleColumnHeight(i);
4      if i < c − 1 then
5          level.distances[i] ← SampleColumnWidth(i);
6      end if
7      j ← 0;
8      while j < h_i do
9          level.objects[i][j] ← SampleGameObject(j, h_i);
10         j ← j + 1;
11     end while
12     i ← i + 1;
13 end while
```

First term of equation (1) evaluates the stability of the objects in the level through their average velocity during the game. It is influenced by all the probability tables, the first one is responsible for the arrangement of objects in columns, so it can reduce their velocity by placing them in a configuration that does not fall. Second table defines the total amount n of objects in the level, which is used to calculate their average velocity. Third table defines the distance between columns, so as the further away they are from each other, the amount of collision between their objects during the game is reduced, which consequently reduces their average velocity.

Second term is the normalized distance between the desired amount of objects B and the total amount of objects $|b|$ in the level. Third term is a function to force the creation of at least one pig. These two terms are influenced by the first and second tables, because they define the total amount of blocks and pigs. Proposed algorithm searches for levels that minimize this fitness function, i.e. levels with fitness approximately zero. It happens when the level has exactly B blocks, more than one pig and all the objects do not move during the game.

Updating Probability Tables

After evaluating all the sampled levels, the proposed algorithm selects the best of them to update the three probability tables. First table is updated based on the objects frequency in each sector of the best level. For example, considering the level in Figure 2, the ground sector of Table 1 is updated as showed in Figure 3. Lines 19 and 21 were updated with $\frac{2}{3} * \frac{1}{n}$ and $\frac{1}{3} * \frac{1}{n}$ because only objects 19 and 21 were sampled in this sector and their frequencies are $\frac{2}{3}$ and $\frac{1}{3}$, respectively. The value n is a parameter of the algorithm to control the update factor in the tables.

Once the sum of all the probabilities in a column of this table must be one, every time a sector is entirely updated, all the non updated objects in this sector have their probability

19	21	19

$$table1[19][0] = table1[19][0] + \frac{2}{3} * \frac{1}{n}$$
$$table1[21][0] = table1[21][0] + \frac{1}{3} * \frac{1}{n}$$

Figure 3. Updating ground sector of Table 1.

equally reduced. For example, after updating lines 19 and 21 in Figure 3, the other probabilities are reduced by $\frac{1}{n} * \frac{1}{20}$.

Second table is updated column by column considering the amount of stacked objects in columns of the best individual. This process is showed in Algorithm 3. Line 3 gets the amount of stacked objects in current column i of the best level and line 4 adds $\frac{1}{n}$ to this probability in the column i of the second table. The sum of columns of this table also must result in one, so lines 5-12 reduce the probability of unchanged lines to fix this sum.

Algorithm 3: Updating probability table for objects stacked.

```
1  i ← 0;
2  while i < c do
3      h ← bestLevel.objects[i].size();
4      table2[h][i] ← table2[h][i] + 1/n;
5      prodToReduce ← 1/n * 1/(c−1)
6      j ← 0
7      while j < c do
8          if i ≠ j then
9              table2[j][i] ← table2[j][i] − prodToReduce;
10         end if
11         j ← j + 1;
12     end while
13     i ← i + 1;
14 end while
```

Last table is updated using a similar approach as the one described in Algorithm 3. However, in this case it increases the probability of widths' range which appeared in the best level. Instead of getting the amount of objects stacked in a column i (line 3), it gets the horizontal distance from the column i to the column $i + 1$. Then, this distance is divided into a range which has the probability updated in the table. If any probability becomes bigger than one when updating tables, it is adjusted to one. In the same way, if any probability becomes smaller than zero, it is adjusted to zero.

EXPERIMENTS AND RESULTS

Experiments were conducted aiming to indicate the strengths and weaknesses of proposed EDA by analysing its expressivity, i.e. the space of all levels that it can create [10]. It is done using three metrics based on those proposed in [6] and [10]: *frequency*, *linearity* and *density*. Frequency is the number of a specific game object in the generated level. Linearity is the average amount of objects stacked in columns within the level and density is the sparseness of the columns through the level [2].

Experiments were performed taking into account three different values for parameter n: 10, 100 and 1000. It acts like an update factor for the probability tables and it influences the convergence of values in probability tables. For each value of n, the algorithm was executed 200 times. After the end of each execution, the three probability tables were used to sample only one level, which was evaluated in terms of frequency, linearity and density.

Levels were generated with $B = 15$ blocks, $c = 5$ columns and maximum amount of objects stacked $h_{max} = 10$. Minimum and maximum horizontal distances are $d_{mim} = 20$ and $d_{max} = 100$. Distance range r was set to 10, so the horizontal distance table has $(100 - 20)/8 = 10$ ranges. The amount of sampled levels in each generation of the proposed EDA was 200 and the maximum number of generations is 1000. The execution time for each level was set to 0.08 seconds. These parameters were empirically determined based on some previous computational tests.

Frequency

First experiment evaluates the relative frequency of game objects in the ground sector for the three different values of n considered. The results of this experiment are illustrated in Figure 4. Considering $n = 10$, the game object 1 has a probability larger than 50% of appearing in the ground sector. This object has a square shape, which provides sustainability for the columns. The other two objects with higher probabilities are object 13 and 9. They have rectangular shapes, so they are also safe objects to be used in the ground. Other objects have probabilities lesser than 0.1. Figure 4 also shows composed blocks and pigs have probability zero.

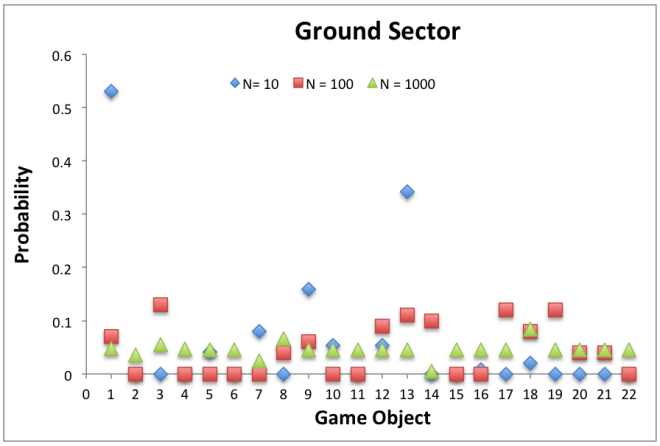

Figure 4. The game objects average frequency in the ground sector for the three different values of n.

Considering $n = 100$, the proposed algorithm gave a probability around 0.1 for objects 3, 9, 12, 13, 14, 17, 18 and 19. It also avoided using in the ground sector other objects that can affect the columns' stability, such as objects 5, 6, 7 and 10. For $n = 1000$, most objects have probability around 0.05.

However, the object 14 has almost probability zero and object 18 has almost probability one.

Figure 5 shows game objects relative frequency in the middle sector for the three different values of n analyzed. In this case, the most used game objects are objects 2, 8, 9 and 12 for $n = 10$. The first one has a square shape and the others have a rectangular shape, so they can be safely placed in the middle sector. Once the pig has a circular shape, its probability in the middle sector is zero too. With $n = 10$, neither composed objects or pigs are used in the middle sector. Most game objects used in the middle sector, for $n = 100$, are the objects 1, 2, 8, 9, 14 and 15. All objects have the probability around 0.5 considering $n = 1000$.

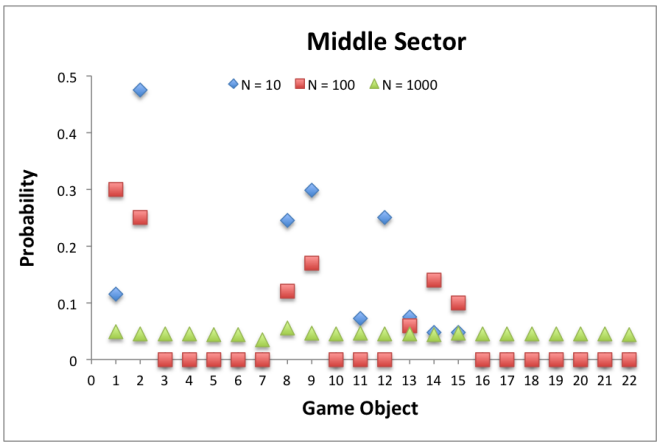

Figure 5. The game objects average frequency in the middle sector for the three different values of n.

Last experiment evaluates the relative frequency of game objects in the top sector and its results are reported in Figure 6. For $n = 10$, it shows almost 65% of the sampled object are from type 1. However, there is a probability around 0.1 for objects 6, 7, 8, 13, 16 and 18. For this value of n, the algorithm does not use pigs in the top sector when it is generating levels. On the other hand, for $n = 100$ the pig object has almost 50% of probability of being placed in the top sector. Nevertheless, objects 7, 10, 11, 15 and 16 have a probability around 0.1. Figure 6 shows the probability of all the blocks is around 0.05 again for $n = 1000$, but only objects 8 and 12 have almost probability zero and the object 9 has probability around 0.1.

Density and Linearity

Frequency only evaluates which game objects were used more through levels, while linearity and density give an idea about the orientation of the game objects. As explained before, linearity l measures the average amount of objects stacked in columns of a level and density $0 \leq d \leq 1$ evaluates columns sparseness based on the distance between them. A low density indicates the columns are very separated and a high density indicates they are very close to each other. Table 4 compares these metrics for the three values of n analyzed. The average (μ) and the standard deviation (σ) values for the best 200 levels generated are depicted.

n	Linearity (μ|σ)	Density
10	3.1937|1.3404	0.5301
100	3.0355|1.5020	0.5126
1000	1.9581|4.6855	0.9789

Table 4. The Linearity and Density when n is 10, 100 and 1000.

Linearity average and standard deviation for $n = 10$ and $n = 100$ are similar. Thus, both settings generate levels with approximately 3 objects stacked in columns. Standard deviations are both around 1.5, so there are no large variations in the amount of objects stacked in columns. Since the desired amount of blocks is $B = 15$ blocks and the number of columns is $c = 5$, all columns in the levels will have at least two blocks. Linearity average is 1.9581 for $n = 1000$, but its standard deviation has a considerable variation with more chance for empty columns.

In the first and second experiments, densities were around 0.5 which means columns disposed through the entire level without large distances between them. For $n = 1000$, the density is almost one meaning that columns are very close to each other. This proximity can lead to several column structures to fall down when some of them are unstable.

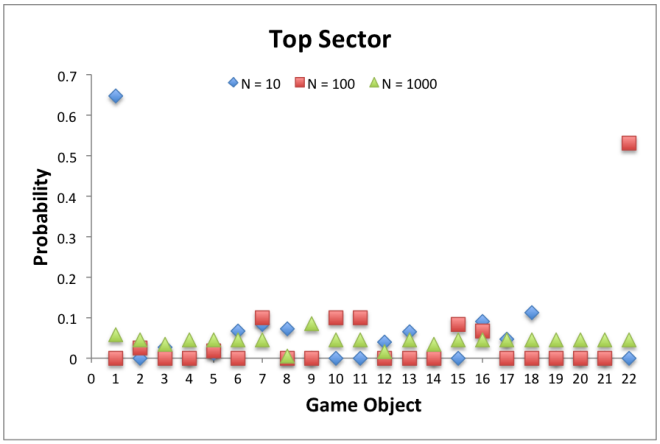

Figure 6. The game objects average frequency in the top sector for the three different values of n.

Generated Levels

Results achieved for $n = 10$ give high probabilities for a few sets of square and rectangular objects. Considering update factor $n = 10$, the proposed method did not explore well all the objects, because it used only objects 1, 2, 9, 12 and 13 and it did not use any composed block. Also, the main problem of using $n = 10$ is that pigs are not sampled which turns the levels unplayable.

Proposed EDA did not explore well the objects probability when $n = 1000$. In the experiments, most objects had a probability around 0.05 independently of the sector. This large value for n becomes small the impact of the update factor through the generations of the algorithm. Thus, the probability estimation process is too slow for this case.

Regarding frequency, linearity and density, the best evaluated value for n is 100. The proposed algorithm used the whole

variety of objects in the three sectors of the level. It is important to highlight that the update factor $n = 100$ used more composed blocks and gave a high probability for pigs in the top of levels. Figure 7 shows an example of level generate with this setting. It meets the restriction of 15 blocks and it has two pigs, one at the top of the third column and another one in the last column.

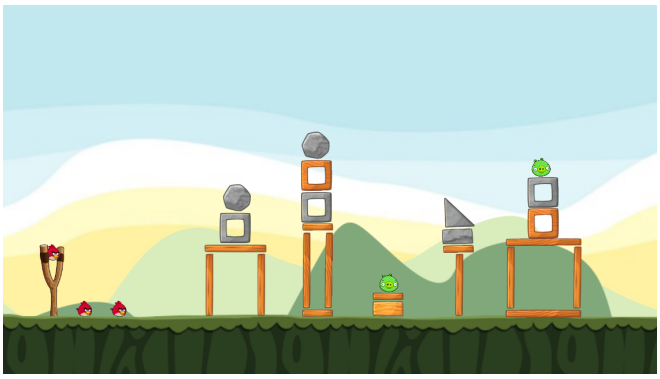

Figure 7. Example of a level generated with $n = 100$

CONCLUSION

This paper presented an evolutionary approach for generating levels in physics-based puzzle games with the Angry Birds mechanics. The proposed algorithm is based on an estimation of distribution algorithm (EDA) and it uses three probability tables to sample levels during its generations. The best level of the sampled ones is used to update the tables, which represent the probability distribution of good solutions. The fitness function considers the average velocity of the game objects during a simulation of the level. The total amount of blocks and pigs is also considered to evaluate a level.

Proposed algorithm was evaluated in terms of its expressivity. We defined three metrics to evaluate the generated levels: frequency, linearity and density. We analyzed this metrics considering the different values for the parameter n of the algorithm: 10, 100 and 1000. This parameter represents the update factor during the algorithm generations.

Expressivity analysis indicated the block types used in a level and the orientation of them. Results showed 100 as the best configuration for n, because it balanced well the block types used through the levels and it was the only setting that had a big probability for pigs on the top sector of the level. Considering $n = 100$, the algorithm creates levels structures with similar amount of objects stacked, indicating a small variation in the arrangement of structures.

REFERENCES

1. Cardamone, L., Loiacono, D., and Lanzi, P. L. Interactive evolution for the procedural generation of tracks in a high-end racing game. In *Proceedings of the 13th annual conference on Genetic and evolutionary computation*, GECCO '11, ACM (New York, NY, USA, 2011), 395–402.
2. Cook, M., and Colton, S. Multi-faceted evolution of simple arcade games. In *Computational Intelligence and Games (CIG), 2011 IEEE Conference on* (2011), 289–296.
3. Entertainment, R. Angry birds game. `http://www.rovio.com/en/our-work/games/view/1/angry-birds`, 2009. [Online; accessed 15-April-2014].
4. Entertainment, R. 1 billion angry birds downloads! `http://www.rovio.com/en/news/blog/162/1-billion-angry-birds-downloads`, 2012. [Online; accessed 19-May-2014].
5. Hauschild, M., and Pelikan, M. An introduction and survey of estimation of distribution algorithms. *Swarm and Evolutionary Computation 1*, 3 (2011), 111 – 128.
6. Horn, B., Dahlskog, S., Shaker, N., Smith, G., and Togelius, J. A comparative evaluation of procedural level generators in the mario ai framework. In *Proceedings of Foundations of Digital Games* (2014).
7. Larrañaga, P., and Lozano, J. A., Eds. *Estimation of Distribution Algorithms: A New Tool for Evolutionary Computation*. Kluwer, Boston, MA, 2002.
8. lenn R. Wichman. A brief history of "rogue". `http://www.wichman.org/roguehistory.html`, 1997. [Online; accessed 15-April-2014].
9. Mourato, F., dos Santos, M. P., and Birra, F. Automatic level generation for platform videogames using genetic algorithms. In *Proceedings of the 8th International Conference on Advances in Computer Entertainment Technology*, ACE '11, ACM (New York, NY, USA, 2011), 8:1–8:8.
10. Shaker, M., Sarhan, M. H., Naameh, O. A., Shaker, N., and Togelius, J. Automatic generation and analysis of physics-based puzzle games. In *CIG*, IEEE (2013), 1–8.
11. Shaker, N., Shaker, M., and Togelius, J. Evolving playable content for cut the rope through a simulation-based approach. In *AIIDE*, AAAI (2013).
12. Shaker, N., Shaker, M., and Togelius, J. Ropossum: An authoring tool for designing, optimizing and solving cut the rope levels. In *AIIDE*, G. Sukthankar and I. Horswill, Eds., AAAI (2013).
13. Shaker, N., Togelius, J., and Nelson, M. J., Eds. *Procedural Content Generation in Games: a Textbook and an Overview of Current Research*. pcgbook.com, 2013.
14. Togelius, J., Preuss, M., Beume, N., Wessing, S., Hagelbäck, J., Yannakakis, G. N., and Grappiolo, C. Controllable procedural map generation via multiobjective evolution. *Genetic Programming and Evolvable Machines*, 2, 245–277.
15. Unity. Unity game engine. `https://unity3d.com`, 2014. [Online; accessed 15-April-2014].