A spatially-structured PCG method for content diversity in a Physics-based simulation game

Raúl Lara-Cabrera[1], Alejandro Gutierrez-Alcoba[2], and Antonio J. Fernández-Leiva[1]

[1] Departmento de Lenguajes y Ciencias de la Computación, Universidad de Málaga
`{raul,afdez}@lcc.uma.es`

[2] Departamento de Arquitectura de Computadores, Universidad de Málaga
`agutierrez@ac.uma.es`

Abstract. This paper presents a spatially-structured evolutionary algorithm (EA) to procedurally generate game maps of different levels of difficulty to be solved, in *Gravityvolve!*, a physics-based simulation videogame that we have implemented and which is inspired by the n-body problem, a classical problem in the field of physics and mathematics. The proposal consists of a steady-state EA whose population is partitioned into three groups according to the difficulty of the generated content (hard, medium or easy) which can be easily adapted to handle the automatic creation of content of diverse nature in other games. In addition, we present three fitness functions, based on multiple criteria (i.e:, intersections, gravitational acceleration and simulations), that were used experimentally to conduct the search process for creating a database of maps with different difficulty in *Gravityvolve!*.

Keywords: Content creation, Evolutionary algorithms, physics-based game, human evaluation

1 Introduction and motivation

The economic costs of producing a video game are very high: the development is a slow process that requires a large team of heterogeneous professionals who, in addition, are required to be highly qualified and specialized. Therefore, any improvement that is able to optimize both the time and resources required to create a video game is always welcome.

According to a recent analysis published in [1], the field of computational intelligence in video games is a vibrant, active field, which attracts new researchers each year and generates new publications. There has been a steady growth in the number of authors, which was accentuated mid-decade 2000–2010. Moreover, the number of publications per year from the community has been increasing since 2005, thus supporting the continued growth of the community.

One of the most promising areas in this field is Procedural Content Generation (PCG) which consists of generating game content through algorithms instead of creating it by hand, and refers to each component that makes up a video game except from the behaviour of the non-playable characters (NPCs) [2, 3]. Some examples of content susceptible to generation are maps, terrains, weapons, items, music or even the game's rules [4].

There are many advantages of producing video game content algorithmically using PCG techniques. In the first place, it allows us to substantially reduce the memory consumption of the game, although nowadays it should be seen as a secondary improvement, it was the main reason for the research and development of such techniques. Another important reason is the high cost of generating some game content manually, as we have already mentioned. Additionally, the game content can be automatically adapted to given criteria, such as the player ability, in such a way that the game offers the player a continuous challenge. If the algorithm is able to generate the content at the same time as the player is playing the game then we are able to create infinite games which offer a different game experience each time the player starts a new game.

It is well known that games can be catalogued according to a set of different genres whose frontiers are usually fuzzy and intersect with the space of other game genre, and it is not difficult to find games simultaneously catalogued, as belonging to distinct genres. In the last few years, the so-called *physics-based simulation games* (PbSGs) have emerged as one of the most exciting classes of games in the video games universe as developers are required to simulate real life physics with the aim of providing more realism and, as a consequence, to create more believable games. This opens new lines of research up as stated in [5] "Physics-based gaming can give your game development repertoire a huge boost, enabling *sandbox-style* game mechanics and *emergent gameplay*." In general, PbSGs (e.g., Angry Birds or Crayon Physics, to name a couple) are easy to play and provide simple game mechanics, but they introduce a number of challenging problems like, unfortunately (or fortunately), the movements of the rigid bodies have to be perfectly simulated which is not an easy task as these are subjected to real Physics Laws and to the interaction with the environment (represented as the game world). As a consequence of all the possible interactions, between all the game objects and the game scenario itself, the number of new possible states (i.e., movements) is huge (in fact, usually infinite) and even unknown a priori. Therefore, the only way to proceed from one state of the game to the following one is to simulate the moves realistically, and this is mandatory as one needs to measure the quality of the state transition (i.e., movements).

In spite of the research interest of PbSGs, generating content for physics-based simulation games is an area that has been explored timidly and, as far as we know, only [6] uses Grammatical Evolution to automatically generate levels for a clone of *Cut the Rope*, a commercial physics-based puzzle game.

Not to mention that PCG algorithms must ensure that the generated content meets some criteria in a way similar to if it had been generated by hand, but this goal is not always easy to satisfy and, in addition, it is difficult to find good mechanisms to evaluate whether the generated artefacts meets the criteria in reality (i.e., according to the player's game experience).

Moreover, it is not enough to simply automatically generate a great number of elements as one might be more interested in creating components that are both diverse and of high quality [7].

In this context, this paper presents the following contributions: first, it introduces *Gravityvolve!*, a physics-based simulation video game that we have implemented, inspired by the n-body problem, a classical problem in the field of physics and mathematics, which can be used by the CI/AI community for research purposes [8]. Moreover, this paper proposes a method, that can be generalized to other (not-necessarily physics-based) games, to procedurally design maps of diverse solving complexity (i.e., of distinct levels of difficult); the proposal consists of an evolutionary algorithm (EA) which is spatially-structured in a number of sub-populations that are co-evolved separately according to different properties required by the individuals of each sub-population (which should supposedly guarantee diversity). In addition, it presents a preliminary experimental study performed to check the suitability of the method.

This paper is structured as follows: Section 2 provides a discussion of related work and Section 3 describes the game *Gravityvolve!*, its rules, objective and the physical laws that guide the gameplay. The map generation algorithm and the fitness functions that measure the difficulty level of the maps are defined in Section 4. Finally, Section 5 discusses the conclusions and future work.

2 Background

In addition to aforementioned related work, this section provides a general overview on Procedural Content Generation (PCG) and Physics-based Simulation Games (PbSGs), and it mentions a number of papers that are directly related with these issues. That being said, the list of papers is far from exhaustive as a review of these fields is not a goal of this paper.

We can make several distinctions regarding the procedures to follow when it comes to the automatic content generation for video games. Following the taxonomy proposed in [9], the content generation should be made online during the gameplay (which provides us with the aforementioned advantages) or offline during the development phase of the game. In the same way, PCG techniques might generate all the content using random seeds (purely stochastic), vectors of parameters (deterministic) or a combination of both.

According to the necessity of the procedurally generated content for the player's progression within the game, we should distinguish between essential and optional content; the former must meet more restrictive criteria than the latter.

Depending on the objectives we want to accomplish, the generation might be done in a constructive manner, ensuring that the content is always valid; or following a generate-and-test scheme, so the content is verified after its creation and if it does not pass the test then the algorithm discards and recreates it.

A widely used and well-known class of PCG algorithms is the so-called Search Based Procedural Content Generation (SBPCG) [10], which is based on looking for the desired content in the complete landscape of solutions. These algorithms follow a generate-and-test scheme and assign real values to each solution in order to measure its quality, instead of accepting or rejecting them. Although evolutionary algorithms are a common choice when developing SBPCG techniques, they are not unique and we are able to use other kind of algorithms in this context (like, for example, planning methods [11]).

Presently, procedural content generation is a vibrant field of research with a large number of papers related to these techniques (the reader can find an analysis about the diversity of the content that may be generated in a procedural manner in [3]). A recurring objective is to generate levels/maps for a platform game. For instance, Noor Shaker et al. [12], proposed a system capable of adapting several parameters, which define the behavior of a level generator for a Super Mario Bros. clone, to the playing style of a certain player. Similarly, Pedersen et al. [13], researched the relationship between the parameters of a PCG algorithm and the game experience and feelings (frustration, fun, ...) that the generated levels provoked to the player.

Another type of content that is susceptible to evolution is a game's map/scenario. For example, Julian Togelius et al., designed a SBPCG multi-objective evolutionary algorithm whose objective was create maps for real-time strategy [14, 15] and racing [16] games. In a similar way, Ferreira and Toledo [17] presented a SBPCG approach for generating levels for the physics-based videogame Angry Birds. Lara et al. [18] presented a search-based procedural content generation method in the context of the real-time strategy game[3] *Planet Wars* (i.e., the Google AI Challenge 2010) whose objective was to generate maps that resulted in an interesting gameplay, focusing on properties of balance and dynamism. Furthermore, the authors expanded their PCG method by considering both new geometrical properties and topological measures that were not affected by rotation, scaling and translation with the aim of avoiding the generation of symmetrical maps that are conceptually equivalent with respect to the gameplay [19]. The topological measures were obtained from the sphere-of-influence graph induced by each map [20]. In turn, Frade et al., proposed a fitness function to guide the generation of accessible terrains with application to the video game "Chapas" [21, 22]. Hom and Marks [23] went further and they procedurally generated rules for a two-player board game with a certain requirement: maximize the balance between both players.

Moreover, there are several examples of PCG for optional content, such as the weapons that a player is able to use. Hastings et al. [24, 25], proposed a SBPCG algorithm for the game "Galactic Arms Race". In this case, the fitness of the generated weapons was computed based on the amount of time the players used them, hence measuring the player satisfaction without requiring the explicit attention of the players. Collins [26] introduced to procedural music in video games, exploring several approaches to procedural composition that had been used in the past. Font et al. [27] presented initial research towards a system that is able to create the rules for different card games. The authors of [28] developed

[3] `http://planetwars.aichallenge.org/`.

a prototype of a tool that automatically produces design pattern specifications for missions and quests for the role-playing game *Neverwinter Nights*.

The reader wishing to know more about the current state of PCG applied to games is referred to [4] for more information. There exist, however, specific references about the application of PCG methods to particular areas of game AI such as, for instance, procedural methods to generate dungeon game levels [29] maze-like levels [30], or music generation [31], just to name a few.

As for Physics-based Simulation Games (PbSGs), it is easy to find evidence of their success in the commercial world as, for instance, *Angry Birds*, *Tower of Goo*, *Crayon Physics*, or *jelly Car*, to name but a few. As mentioned, PbSGs provide realism in the simulation of the game and, therefore, increase the immersion of the player which surely positively influences her satisfaction and favors their involvement with the game. In fact, physics can be found even in the early phases in the history of video games; so, Super Mario Bros already exhibited, in 1985, elementary concepts of physics in the form of jumps, forward/backward movements, and object throwing, executed by the main character. However, in modern video games, physics is generally referred to as rigid body physics simulation subject to real physics laws (e.g., Newton's Three Laws of Motion or Newton's Law of Universal Gravitation, just to name a couple associated to the classical Physics). PbSGs games are very interesting not only from the player's perspective but also from a research point of view; so, there have been interesting papers recently published on the Physical Travelling Salesman Problem (PTSP), a real-time game that consists of a ship that must visit a number of waypoints scattered around a 2-D maze full of obstacles [32]. This problem can be viewed as a PbSG as all actions applied to the ship are forces that influence its position, orientation and velocity at each step of the game. Precisely, [33] and [34] employ algorithms based on Monte Carlo tree search [35] to handle the problem.

Furthermore, "A slower speed of light" [36] is a game developed by the MIT Game Lab to help students understand and visualize the effects of special relativity by artificially lowering the speed of light to walking pace. The game, which is based on a first-person relativity visualization engine that has been released as *OpenRelativity*, is a prototype in which players navigate through a 3D space while picking up orbs that reduce the speed of light.

3 The game: Gravityvolve!

There is a well-known problem inside the field of physics and mathematics, the so-called *n-body problem* the origin of which lies in Newton's Principia and classically consists of[4] "predicting the individual motions of a group of celestial objects interacting with each other gravitationally". This problem requires the existence of n rigid bodies and basically consists of determining the positions and velocities of these n particles in each instant of the time in accordance with Newton's Laws of Motion and of Universal Gravitation, starting with an initial

[4] Wikipedia. Accessed on 17th of January, 2016.

position and velocity for each particle and letting the gravitational forces act on the set of particles. For $n = 2$ the problem represents, in certain form, the most fundamental kind of interaction between two bodies, and the problem has no analytical solution (for $n \geq 3$); moreover, generally it can be only simulated using numerical integration methods [37]. The n-body problem considers n particles with specific masses $m_1, \ldots, m_n$ moving in a three dimensional space under the influence of mutual gravitational attraction.

Gravityvolve! is a game implemented by the authors of this paper [8], that has been inspired by the n-body problem, which includes some additional features that transforms an interactive simulation into a playable environment. In the following paragraphs we discuss some articular features of our game:

In the first place, the simulation is constrained to a 2D environment, because using a 3D environment would result on a senseless increase of the complexity and we want to preserve the simplicity of traditional PbSG such as *Angry Birds*. Players have a simpler visualization of the game using a bi-dimensional environment: they are able to watch the full playing area all the time, with an aerial view of the plane where the particles are located. This way, the particles' trajectories are more natural and understandable in 2D than in 3D.

Anyhow, dealing with the problem in this way does not represent a decrease in the generality of 3D games, especially those games whose game terrain is a surface that can be defined as the graph of a certain function $f : \mathbb{R} \times \mathbb{R} \to \mathbb{R}$. Thus, although the map has a three-dimensional appearance, the position of its components may be defined with two coordinates and its topology matches the plane $\mathbb{R} \times \mathbb{R}$.

Secondly, there is only a particle (i.e., a ball) affected by the gravitational force of the remaining particles during the simulation. This particle is the only one the player is able to interact with, by changing its velocity vector and guiding it over the screen, while the remaining particles remain static on their initial position. Once again, this restriction is done for the sake of simplicity, hence reducing the amount of information the player has to process and avoiding highly chaotic and unpredictable behaviors.

Figure 1 shows two screenshots of the game running the version with 5 planets. There are $n = 5$ particles (i.e., planets) with an associated mass distributed over the screen; each particle is represented by a planet with a radius that is as long as its mass. These particles (i.e. planets) remain static throughout the game. There exist two other bodies that are positioned on the surface of two distinct planets: the *ball*, represented by a small red circle, and the *hole*, represented by a red circumference. In each step of the game, the user has to interact with the game to set the magnitude and direction of the velocity vector associated with the ball. A green line segment (as displayed in Figure 1a) represents precisely these values to be fixed by the user (the orientation of this line segment indicates the direction in which the ball will be thrown and its length the magnitude of the force with which the ball is thrown). To help the player, a purple line shows the prediction of the ball's trajectory according to the user interaction; Figure 1b shows the movement of the ball after being affected by the user in

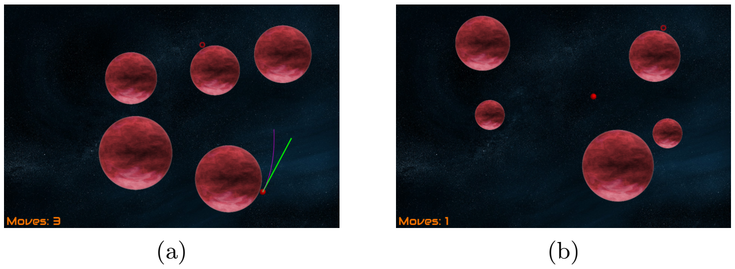


Fig. 1. Example of two different phases/maps of the game in a 5-planet version. (a) The length and orientation of the green line segment indicate the magnitude and direction of the velocity associated with the ball whereas the purple line is a hint (to the player) that provides a prediction about the trajectory of the ball according to the 'force' applied to the ball (b) The red ball has been thrown and is moving toward the objective according to both its velocity and the forces of gravitational system.

another 5-planet map. The objective of the game is to drive the ball from its point of origin to where the hole is placed, in the minimum number of moves. In a way, *Gravityvolve!* is equivalent to a Golf Game in the sense that a ball has to reach the objective of a hole in a minimum number of hits, whereas the tuning of the values associated with the magnitude and direction of the ball in *Gravityvolve!* corresponds to hitting the ball in golf.

Given the aforementioned objective and game features, the task of defining a map for this game consists of defining the mass and position for each planet in the map, as well as the initial positions of both the ball (i.e., the movable particle) and the hole. An additional effort consists of obtaining maps of varying difficulty which clearly depends on the arrangement of the planets and the original placement of both the ball and the hole.

3.1 Game physics

The gameplay in *Gravityvolve!* is based on Newton's Law of Universal Gravitation, as well as Newton's Three Laws of motion. In particular, the first law states that any two bodies in the universe attract each other with a force that is directly proportional to the product of their masses m_1, m_2 and inversely proportional to the square of the distance r between them. Note that as our objective was to create a simulation with our own magnitudes, the gravitational constant became irrelevant so we got by without it ensuring that the simulation keeps meeting Kepler's laws. That way, the acceleration that one body causes to another is the interesting magnitude, which is, in addition, an easily computable variable: the second of Newton's law states that the relationship between the acceleration a suffered by a body with a mass m_1 then the received force is $F = m_1 a$

Considering this force F the one produced by another body with a mass m_2, then the first body is affected by an acceleration $a = G\frac{m_2}{r^2}$

The aforementioned equation calculates the modulus of the acceleration vector, however, we need the coordinates of this vector in order to apply it to the movable particle. Using basic trigonometric equations: denoting by p_1 and p_2 the position vectors of both bodies, the acceleration that the second body provokes on the first one is $\boldsymbol{a} = \frac{m_2}{r^3}(\boldsymbol{p_2} - \boldsymbol{p_1})$

A simple implementation of the simulation that uses the aforementioned vector of acceleration consists of discretizing the time in such a way that, on each iteration, it updates the velocity and consequently the position of the particle with the acceleration vector depending on the elapsed discretized time. Although the Runge-Kutta integration method has more precision, we used the Newton integration method which is precise enough for this game and its rules. Magnitudes such as time discretization, the mean of the masses and the game screen dimension are strictly related with each other them while establishing the precision and velocity of the simulation.

4 The procedural map generator

The map generator is based on a steady-state evolutionary algorithm with a structured population divided into three sets (i.e., subpopulations) with the same number of elements. The first set contains those best adapted individuals according to the fitness function, which measures the difficulty level of the map. According to this, each sub-population groups the individuals with similar difficulty levels, so hard maps (i.e., those with their fitness value ranged between the theoretical maximum fitness and a 66% of this value) should rely on the first group as medium and easy maps should rely on second and third sub-populations, respectively.

Regarding the generation of the initial population, the algorithm generates random individuals and assigns them to their corresponding subpopulation until all of them are complete, hence ensuring a high population diversity at the beginning of the evolutionary process.

Upon each generation, the algorithm selects two random individuals from the population, which are then mutated and recombined using the operators described in Section 4.1. We decided to use this random selection mechanism to increase the diversity of the offspring. Then, the algorithm computes the fitness of the new individuals and inserts them into the population applying the following replacement policy: as the population is structured into three groups, the subpopulation where the new individual may be inserted depends on its fitness value and the theoretical maximum fitness value. Then, depending on the selected subpopulation, that is, Hard, Medium or Easy, the fitness of the new individual is compared to the fitness of the best, central or worst individual, respectively (see Figure 2).

Each individual of the algorithm represents a map, and every map is defined by the planets included in it and the position of both the ball and the hole. Each planet is made up of three genes: its x and y coordinates and its mass m (which, in addition, corresponds to its radius). Regarding the ball and the hole, they are

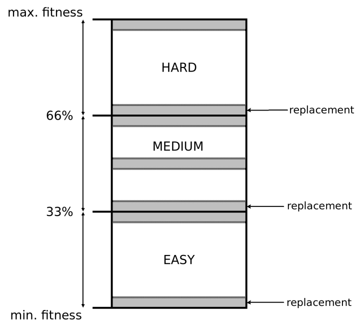


Fig. 2. Structure of the algorithm's population according to the fitness of their elements and both theoretical maximum and minimum fitness value. New individuals replace the worst individual of the corresponding subpopulation if their fitness is better. Gray coloured individuals are those selected for the experimental phase.

made of two genes each: the planet over which they are placed and the angle in radians that specify the position over the planet surface. Figure 3 shows an example of a map and its corresponding encoding using our evolutionary PCG method.

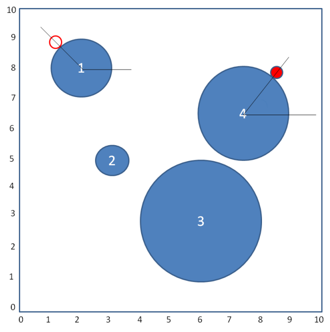
4.1 Operators

Regarding the mutation operator, on a map with n planets, the number of genes is $3n + 4$, one for each planet's X and Y coordinate and its mass, and two for both the ball and the hole, which corresponds to the planets on which ball and hole are placed as well as the angle in radians (see Figure 3). Note that planets' genes and those that encode the properties of the ball and the hole follows a different mutation process. With respect to the planets' genes, their mutation probability is $p = \frac{1}{3n}$, so the number of mutated genes from each map follows a binomial distribution $X \sim B(3n, \frac{1}{3n})$ so the mean quantity of mutated genes turns out to be 1.

When a gene is selected for mutation, its coordinates are modified adding a random value Δ_c that follows a normal distribution with $\mu = 0$ and $s^2 = 50$ to each coordinate. A similar modification is made to the planet's mass in such a way so that the planet may increase or decrease its radius (mass), the mutation step follows the same distribution as the mutation step of the coordinates.

On the other hand, for the mutation of both the ball and the hole, a new random planet and angle are assigned to them, with a mutation probability of 0.15.

After the mutation step, the algorithm checks the validity of the map and, if the map is no longer valid (i.e., a planet has moved out of bounds or there are overlapping planets), the algorithm reverts the mutation and repeats the process until the map remains valid after applying the mutation changes.



1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16
2	8	1	3	5	0.5	6	3	2	7.5	6.5	1.5	4	π/4	1	3π/4

Fig. 3. An example of a 4-planet *Gravityvolve!* map in a scenario of size 10×10, and its corresponding encoding. Planets are numbered for clarity; the first 12 genes represent the information of the planets — i.e., 3 genes per planet to encode its position in the 2D grid and its radius (i.e., mass). Genes 13th and 14th (resp., genes 15th and 16th) provide information for the ball (resp. the hole) in the form of the planet number over which it is placed and angle in radians that indicates its position in the planet's surface taking the circumference relative to the surface of the planet as reference.

The crossover operator, whose definition follows, is inspired by the one point recombination, but geometrically. In the first step the operator computes a random line that splits the map area and then, using two random points, defines the equation of the line. Secondly, the operator builds two sets of planets, one for those that stand above the line and another for those under the line. A map is considered to be a member of one set or the other if its center is above or below the line. This cut should be accepted as valid if there is at least one planet on either side of the line and the number of planets on each side is the same. The crossover operator works over two parents and, after a valid cut is computed for each parent, their slices are swept (left/up slice from parent one combined with right/down slice from parent two and vice versa) so new individuals get one slice from each parent.

It might seems that considering a cut as valid only if there is the same number of planets on both slices is a very restrictive condition, because as the number of planets raises, the probability of making a valid cut decreases: given n planets, this probability is roughly $\frac{1}{n}$ (provided that we are omitting the unlikely case where there are no planets on a slice). Hence the mean number of attempts to achieve a valid cut is n. However, this problem can be avoided introducing a minor change into the algorithm: a cut will be valid if there is a difference of k ($k << n$) between the number of planets on both slices and, in this case, positioning the remaining planets at random positions or moving them into the parent with the lowest number of planets.

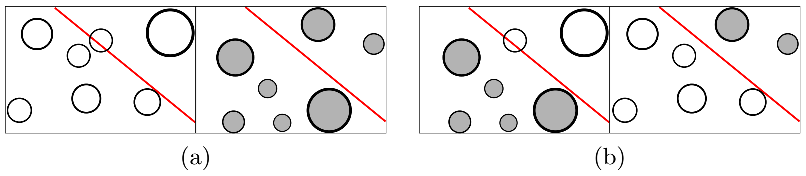


Fig. 4. Example of a crossover operation. The upper maps are the parents (a) and the lower are the children (b). Parent maps are divided into two slices following the same cut line and then swapped in order to obtain the children. Planets have been colored to distinguish which parent the planets on the child side have been taken from.

Figure 4 shows a valid example of a crossover. The red line is the line used to divide the map into two slices. For the sake of clarity, planets from each map are colored so it is easy to identify them before and after the crossover.

4.2 Measuring a map's difficulty level

We propose three different fitness functions representing level properties that try to measure the difficulty level of a map. Contrary to the aforementioned methods, these functions are strictly related to the game and may be difficult to adapt and use for other games.

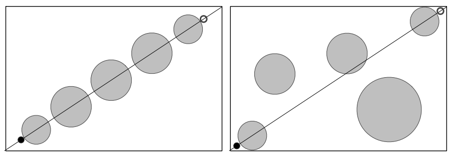


Fig. 5. Maps with maximum fitness values computed by the intersections (left) and simulations (right) methods. The black circle and circumference represents the ball and the hole, respectively. On the left side, the planets are aligned with a screen diagonal and the particle and hole are placed on opposite corners of the screen. On the right side, the ball is located on the corner after a simulated shoot and the hole is on the opposite corner, so the distance between them is the highest.

The first fitness function is based on the idea that the distance between the particle and the hole as well as the number of planets between them is a good estimator of how hard it is to place the particle over the hole. This method computes the equation of the line between the particle's center and the planet where the hole is located in order to find which planets intersect with this line and how far the hole is from the particle. Given the equation of the line $y = mx + n$ and the circumference of a certain planet $(x - a)^2 + (y - b)^2 = R^2$, the intersect points between that planet and the line are:

$$\begin{cases} x_1 = \pm\dfrac{\sqrt{R^2m^2 + R^2 - a^2m^2 + 2abm - 2amn - b^2 + 2bn - n^2}}{m^2+1} + \dfrac{a+bm-mn}{m^2+1} \\ \\ y_1 = mx_1 + n \end{cases}$$

The fitness value computed by this function is the sum of all these line segments defined by the aforementioned intersect points. We observe that, if there were no restrictions on the generated levels such as no two planets should overlap and there must exist a minimum distance between the planets so the ball is able to move between them without getting stuck, maps with a high fitness would be a level with all the planets aligned with a screen diagonal without any gap between them and the particle and hole placed on opposite corners of the screen, which defines, in turn, the map that evaluates to the maximum possible fitness value (see Figure 5). However, the restrictions applied to the generated maps increase the variability of the procedurally generated maps.

There is a another way of measuring the difficulty level of the game's objective: minimize the ball's force of attraction. An easy way to achieve this could be minimizing the mass of the planet that hosts the hole and increasing the respective masses of the other planets. For instance, the fitness function might be the defined as $\frac{m}{M}$, given the mass of the target planet m and the sum of the mass of the remaining planets M. Due to the simplicity of the solution, this could be implemented without using evolutionary algorithms but the resulting maps would be quite similar to each other and the position of the planets would not affect the generation process which in turn would lead to a loss in diversity in the solutions.

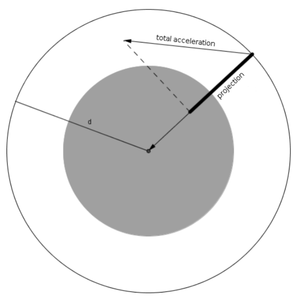


Fig. 6. Projection of the induced acceleration for an auxiliary particle by other planets over the vector between this particle and the center of the planet

For these reasons, we consider a more complex fitness function, based on the force of attraction, that takes into account the mass and position of each planet.

The function takes the mean acceleration $\bar{a}$ produced by all the planets over the planet with the hole (the target planet) as the fitness value for a certain map. The function puts $n = 50$ auxiliary particles uniformly distributed around the target planet within a certain distance d and then, computes the acceleration α induced for each auxiliary by the other p planets, which is projected over the vector between the auxiliary particle and the planet center (see Figure 6):

$$\bar{a} = \frac{\sum_{i=1}^{n} \text{proj}(\alpha_i)}{n}$$

In this case, the maximum fitness value, which defined the subpopulations, was the highest fitness value computed for the random individuals that were in the initial population.

Thirdly, we define a simulation-based fitness function that makes several shootings with different angles and forces, more precisely, the function simulates 10 throws towards the target planet with random velocities. So, the fitness of the map is the minimum distance between the ball after the shoot and the hole, hence as the fitness decreases, the difficulty level of the map also decreases. The reader should observe that the maximum theoretical value for this fitness function is reached when the ball, after the shoot, is located on a corner of the screen and the hole is on the opposite corner (see Figure 5).

5 Conclusions

This paper has presented a physics-based video game inspired by the so-called *n-body problem*, a well-known problem in the field of physics and mathematics. The maps for the games have been procedurally generated using a steady-state evolutionary algorithm whose population was structured in three groups or difficulty levels; our spatially-structured proposal is directed to obtain content of diverse nature (in the case of this paper of varying difficulty) and can be easily generalized to other games. In addition, three different fitness functions have being proposed to drive the search process inside our evolutionary algorithm. These functions compute how difficult a map is depending on multiple criteria: the number of planets in straight line between the particle and the target, the gravitational acceleration induced on the target planet by the rest of the map's elements and the distance between the target and the particle after shooting it using different angles and forces (i.e. simulations).

In [8], we provide a web link where the reader can play a number of maps (with 5 planets), of diverse difficulty, that were generated by our proposed PCG method (and by considering all the fitness functions defined here). We have performed a preliminary study on the system's capabilities to generate maps of diverse difficulty: the web platform has collected the participation of 58 unique users and an average number of votes per map of 55, because not all participants played and evaluated each of the 21 maps (see Table 1).

Note that the generated maps contains only 5 planets and, thus the maps are not difficult to play. However, some players (a 10%) considered some of our maps

Table 1. Each row (except the first row) represents the maps with a certain difficulty level from our testbed; column 1 indicates the fitness function employed to generate these maps whereas column 2 provides the group to which the map belongs according to the PCG method whose search was lead by the specified fitness function; columns 3 to 5 display the votes given by the users to each specific map according to their game experience, and column 6 shows the number of votes received for each map.

Fitness Type	**level**	**Hard**	**Medium**	**Easy**	**Total**
Intersections	Hard	10	45	54	109
	Medium	5	19	136	160
	Easy	12	56	44	112
Attraction	Hard	15	34	60	109
	Medium	26	40	106	172
	Easy	7	31	67	105
Simulations	Hard	17	44	49	110
	Medium	11	49	104	164
	Easy	2	21	89	112

as hard. This is a promising result although, in general, players perceived that the maps were of medium or easy difficulty. This is a preliminary experiment and we plan to generate maps with varying number of planets (e.g., between 5 and 15) and conduct a deeper analysis of the results.

We are aware that some of the concepts described in this paper might seem very specific of the game, however, the schema of our spatially-structured method can be adjusted to other games, and demonstrating this is also part of a future work.

Acknowledgments

This work has been partially funded by Spanish MICINN under project UMA-EPHEMECH[5] (TIN2014-56494-C4-1-P), by Junta de Andalucía under project P10-TIC-6083 (DNEMESIS[6]) and Universidad de Málaga. Campus de Excelencia Internacional Andalucía Tech.

References

1. Lara-Cabrera, R., Cotta, C., Fernández-Leiva, A.: An analysis of the structure and evolution of the scientific collaboration network of computer intelligence in games. Physica A: Statistical Mechanics and its Applications **395**(0) (2014) 523 – 536
2. Togelius, J., Champandard, A.J., Lanzi, P.L., Mateas, M., Paiva, A., Preuss, M., Stanley, K.O.: Procedural content generation: Goals, challenges and actionable steps. In Lucas, S.M., Mateas, M., Preuss, M., Spronck, P., Togelius, J., eds.:

[5] `http://blog.epheme.ch`

[6] `http://dnemesis.lcc.uma.es/wordpress/`.

Artificial and Computational Intelligence in Games. Volume 6 of Dagstuhl Follow-Ups. Schloss Dagstuhl - Leibniz-Zentrum fuer Informatik (2013) 61–75
3. Hendrikx, M., Meijer, S., Van Der Velden, J., Iosup, A.: Procedural content generation for games: A survey. ACM Transactions on Multimedia Computing, Communications, and Applications (TOMCCAP) **9**(1) (2013) 1
4. Shaker, N., Togelius, J., Nelson, M.J.: Procedural Content Generation in Games: A Textbook and an Overview of Current Research. Springer (2014)
5. Rivello, S.A.: Developing physics-based games with adobe flash professional EDGE, article 7. April 2010. Accessed on 15h of January, 2015.
6. Shaker, M., Sarhan, M.H., Naameh, O.A., Shaker, N., Togelius, J.: Automatic generation and analysis of physics-based puzzle games. In: 2013 IEEE Conference on Computational Inteligence in Games (CIG), Niagara Falls, ON, Canada, August 11-13, 2013, IEEE (2013) 1–8
7. Preuss, M., Liapis, A., Togelius, J.: Searching for good and diverse game levels. In: Computational Intelligence and Games (CIG), 2014 IEEE Conference on. (Aug 2014) 1–8
8. Lara-Cabrera, R., Fernández-Leiva, A.J.: Gravityvolve!
9. Togelius, J., Yannakakis, G.N., Stanley, K.O., Browne, C.: Search-based procedural content generation: A taxonomy and survey. IEEE Transactions on Computational Intelligence and AI in Games **3**(3) (2011) 172–186
10. Togelius, J., Yannakakis, G.N., Stanley, K.O., Browne, C.: Search-based procedural content generation: A taxonomy and survey. IEEE Trans. Comput. Intellig. and AI in Games **3**(3) (2011) 172–186
11. Yannakakis, G., Togelius, J.: A panorama of artificial and computational intelligence in games. Computational Intelligence and AI in Games, IEEE Transactions on **PP**(99) (2014) 1–1
12. Shaker, N., Yannakakis, G.N., Togelius, J.: Towards automatic personalized content generation for platform games. In: AIIDE. (2010)
13. Pedersen, C., Togelius, J., Yannakakis, G.N.: Modeling player experience in super mario bros. In: Computational Intelligence and Games, 2009. CIG 2009. IEEE Symposium on, IEEE (2009) 132–139
14. Togelius, J., Preuss, M., Yannakakis, G.N.: Towards multiobjective procedural map generation. In: Proceedings of the 2010 Workshop on Procedural Content Generation in Games, ACM (2010) 3
15. Togelius, J., Preuss, M., Beume, N., Wessing, S., Hagelback, J., Yannakakis, G.N.: Multiobjective exploration of the Starcraft map space. In: Computational Intelligence and Games (CIG), 2010 IEEE Symposium on, IEEE (2010) 265–272
16. Togelius, J., De Nardi, R., Lucas, S.M.: Towards automatic personalised content creation for racing games. In: Computational Intelligence and Games, 2007. CIG 2007. IEEE Symposium on, IEEE (2007) 252–259
17. Ferreira, L., Toledo, C.: A search-based approach for generating angry birds levels. In: Computational Intelligence and Games (CIG), 2014 IEEE Conference on. (2014) 1–8
18. Lara-Cabrera, R., Cotta, C., Fernández-Leiva, A.: On balance and dynamism in procedural content generation with self-adaptive evolutionary algorithms. Natural Computing **13**(2) (2014) 157–168
19. Lara-Cabrera, R., Cotta, C., Fernández-Leiva, A.: Geometrical vs topological measures for the evolution of aesthetic maps in a RTS game. Entertainment Computing **5**(4) (2014) 251 – 258
20. Toussaint, G.T.: A graph-theoretic primal sketch. Computational Morphology (1988) 229–260

21. Frade, M., de Vega, F.F., Cotta, C.: Evolution of artificial terrains for video games based on obstacles edge length. In: Evolutionary Computation (CEC), 2010 IEEE Congress on, IEEE (2010) 1–8
22. Frade, M., de Vega, F., Cotta, C.: Automatic evolution of programs for procedural generation of terrains for video games. Soft Computing **16**(11) (2012) 1893–1914
23. Hom, V., Marks, J.: Automatic design of balanced board games. In: Proceedings of the AAAI Conference on Artificial Intelligence and Interactive Digital Entertainment (AIIDE). (2007) 25–30
24. Hastings, E.J., Guha, R.K., Stanley, K.O.: Automatic content generation in the Galactic Arms Race video game. Computational Intelligence and AI in Games, IEEE Transactions on **1**(4) (2009) 245–263
25. Hastings, E.J., Guha, R.K., Stanley, K.O.: Evolving content in the Galactic Arms Race video game. In: Computational Intelligence and Games, 2009. CIG 2009. IEEE Symposium on, IEEE (2009) 241–248
26. Collins, K.: An introduction to procedural music in video games. Contemporary Music Review **28**(1) (2009) 5–15
27. Font, J., Mahlmann, T., Manrique, D., Togelius, J.: A card game description language. In Esparcia-Alcázar, A., ed.: Applications of Evolutionary Computation. Volume 7835 of Lecture Notes in Computer Science. Springer Berlin Heidelberg (2013) 254–263
28. Onuczko, C., Szafron, D., Schaeffer, J., Cutumisu, M., Siegel, J., Waugh, K., Schumacher, A.: Automatic story generation for computer role-playing games. In: AIIDE. (2006) 147–148
29. van der Linden, R., Lopes, R., Bidarra, R.: Procedural generation of dungeons. IEEE Trans. Comput. Intellig. and AI in Games **6**(1) (2014) 78–89
30. Ashlock, D., Lee, C., McGuinness, C.: Search-based procedural generation of maze-like levels. Computational Intelligence and AI in Games, IEEE Transactions on **3**(3) (Sept 2011) 260–273
31. Plans, D., Morelli, D.: Experience-driven procedural music generation for games. Computational Intelligence and AI in Games, IEEE Transactions on **4**(3) (Sept 2012) 192–198
32. : The physical travelling salesman problem Accessed on 10tht of December, 2015.
33. Perez, D., Powley, E.J., Whitehouse, D., Rohlfshagen, P., Samothrakis, S., Cowling, P.I., Lucas, S.M.: Solving the physical traveling salesman problem: Tree search and macro actions. IEEE Trans. Comput. Intellig. and AI in Games **6**(1) (2014) 31–45
34. Powley, E., Whitehouse, D., Cowling, P.: Monte carlo tree search with macro-actions and heuristic route planning for the physical travelling salesman problem. In: Computational Intelligence and Games (CIG), 2012 IEEE Conference on. (Sept 2012) 234–241
35. Browne, C., Powley, E., Whitehouse, D., Lucas, S., Cowling, P., Rohlfshagen, P., Tavener, S., Perez, D., Samothrakis, S., Colton, S.: A survey of monte carlo tree search methods. Computational Intelligence and AI in Games, IEEE Transactions on **4**(1) (March 2012) 1–43
36. Kortemeyer, G., Tan, P., Schirra, S.: A slower speed of light: Developing intuition about special relativity with games. In: International Conference on the Foundations of Digital Games, Chania, Crete, Greece, May 14-17, 2013. (2013) 400–402
37. Aarseth, S.J., Aarseth, S.J.: Gravitational N-Body Simulations: Tools and Algorithms. Cambridge University Press (2003)