A Generic Framework for Procedural Generation of Gameplay Sessions

Leonardo V. Carvalho, Átila V. M. Moreira, Vicente V. Filho, Marco Túlio C. F. Albuquerque, Geber L. Ramalho
Center of Informatics (CIn)
Federal University of Pernambuco
Recife-PE, Brazil
{lvc, avmm, vvf, mtcfa, glr}@cin.ufpe.br

***Abstract* — this paper studies the application of procedural content generation techniques in the development of endless games. A chunk-based approach is used along with a 4-step level generation process that takes in consideration the need for an infinite chunk-placement cycle. Chunks are defined as playable game segments with fixed width and height, on which gameplay elements such as coins are placed. The proposed process consists of the definition of a difficulty curve, chunk generation, chunk evaluation, and insertion of the chunks in real time during the gameplay session. For the purposes of this work, we focus on the chunk evaluation step, which employs neural networks that are capable of evaluating each chunk in terms of difficulty. The parameters used by these networks include both non-controllable features - i.e. amount of coins collected by the player - and controllable features - i.e. amount of obstacles present in the chunk. The system was employed on the game Boney the Runner during its development, and as of the time of writing the game is available for download on the Apple and Google app stores. To validate the work, neural networks based on player data were built, and the performance of such networks was compared to that of a human designer. The performance of the neural networks varied according to the parameters used, but the best one was capable of correctly classifying 90% of the chunks from the pool, while a human designer was capable of correctly classifying 52% of said chunks.**

Keywords— procedural generation; endless game; neural network; difficulty modelling

I. Introduction

Procedural content generation, or PCG, consists of the use of algorithms in the process of creating content, like game levels, maps or music. Increasingly more game development companies have been employing these techniques on their processes, often motivated by the elevated monetary costs of creating the content manually and by the variety of content that can be generated procedurally [1].

The use of PCG techniques in games dates back to the 1980s, with games like Rogue [2], a graphical role-playing game on which a virtually infinite amount of levels is generated on demand. Even though the generated levels do not present a high complexity when compared with what can be created manually, the game has shown that it is possible to greatly increase the replay factor by using non-supervised level generation [3].

Currently in the game industry, PCG is being employed both in the generation of content on demand - with the goal of increasing a game's replay value or adapting the game to the player – and during the game's production stage – in order to make the content generation process more efficient [4]. Overall there is no restriction regarding the kind of content that can be generated procedurally – textures, sounds, maps, cities, levels and even whole systems can be generated algorithmically with various levels of complexity [5].

Some of the games that illustrate well the potential of PCG include the games .kkrieger [6] and Zettai Hero Project [7]. The former is a 3D first person shooter on which all game assets are generated procedurally. That includes textures, tridimensional models, sound effects and levels; however, no gameplay aspect is affected by the procedural generation. The latter is a rogue-like game on which PCG is used in order to increase the game's replay factor through the generation of levels, enemies and equipment. Unlike .kkrieger, on Zettai Hero Project no PCG techniques are used in the creation of artifacts such as textures and sounds.

Given the potential and widespread use of PCG on games, this paper has the purpose of analyzing the application of PCG techniques on a game genre that has become prominent with the popularization of mobile platforms – endless running. This genre consists of an infinite race on which the player's goal is to obtain a high score while avoiding obstacles present along the way. One of the features that influence the score is the maximum distance that the player can reach, but other elements can also influence, such as collectible items. Since games of this genre are very dependent of a high replay value, it's important to present levels with a high variety, which is something that a can be done through the use of PCG.

The endless running genre is relatively new, with much of its recent popularity attributed to mobile games such as Jetpack Joyride, which can be seen on Fig 1. This game had over 170 million downloads across various online stores and received various rewards from the specialized press [18] as of writing. On this game, the player must survive for as long as possible while avoiding traps and collecting items along the way, and the final distance reached is used as the score.

Because the genre is still relatively new, there is still a scarcity of materials that focus on the generation of gameplay sessions for this kind of game, given that most solutions focus on generating levels with clear initial and final points [3] [9] [11], instead of an endless track.

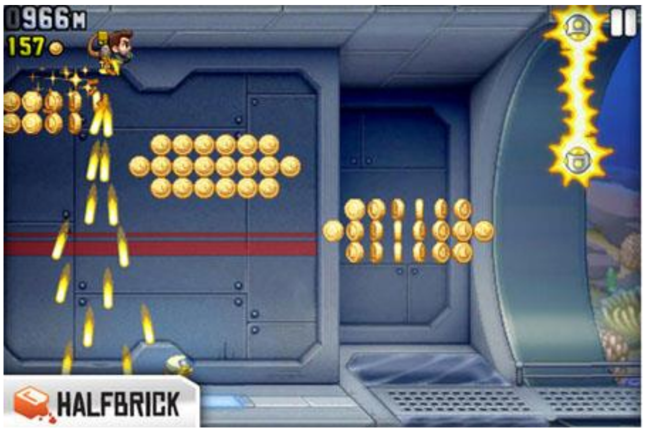

Fig. 1. A gameplay session of Jetpack Joyride.

As it will be shown on the next section, many of the existing level generation solutions present the possibility of generating a virtually infinite amount of levels for games [8] [10], but said levels commonly have a start and end point. On an endless running game, there isn't the concept of a level with a pre-determined end, since the gameplay sessions consist of a continuous challenge which is generated on demand and whose end depends on the player's skills. Also, to the best of the authors' knowledge, there still aren't any relevant surveys that focus on endless games.

Because of that, it's necessary to define not only how to generate playable chunks, which are essentially fixed-size game segments on which gameplay elements are placed, but also how those chunks are positioned along the track on two moments: during the initial player progression and during an endless cycle. This paper does not try to tackle all these questions simultaneously; it focuses on the evaluation and positioning of playable chunks, while the remaining aspects utilize heuristics defined by game designers.

II. Related Work

As it will be seen on this section, most of the works that were analyzed on our studies focus on the generation of finite levels or level networks. In [3], the concept of rhythm is used on the level generation process. Basic components like platforms or spikes are combined according to a rhythm pattern, and the result of this combination is an entity denominated "pattern". Said patterns are then encapsulated on "cells", which can be used to create both linear and non-linear levels. The levels are determined by "cell structures", which allow for the creation of levels with branching paths, hubs and many other commonly used structures.

The work presented by [8] is similar to the previous one, but it presents some substantial changes on the way that the level is generated: instead of creating chunks directly from rhythm patterns, an intermediary structure is created from said patterns. From this structure, a great variety of chunks can be created by combining basic gameplay components, so this allows for a greater variety of playable chunks for the same kind of rhythm pattern.

The approach used by [9] consists of a generic framework aimed at automating the level creation process by using a top-down approach. This work utilizes a genetic algorithm to generate levels that respect certain constraints defined by game designers, and each level is comprised of various "design elements", which correspond to the basic gameplay elements. The best levels are chosen by the fitness function in the genetic algorithm, which is capable of determining the amount of fun that each level provides based on the difficulty perceived by the player over time.

On the work presented by M. Kerssemakers, J. Tuxen, J. Togelius and G. N. Yannakakis [10], a meta-procedural level generator is presented with the purpose of minimizing the effort of creating a new level generator for each game. This generator works with two cycles, an internal one which uses agents to create levels, and an external one which provides a visual representation of each level generator. The level generators have to be analyzed by a human designer, who is responsible by choosing the best ones.

The work presented by C. Pedersen, J. Togelius and G. N. Yannakakis [11] focuses on modelling the player experience on the game Super Mario Bros. On this work, levels are created by using heuristics that randomly position most gameplay elements on the levels, the only exceptions are elements that affect player experience, which are positioned on certain fixed positions. Players were invited to play sets of game sessions during which gameplay and level data were collected. Once the sets of sessions are over, the players answered a short form to tell about the game experience, and all this information was used to build a neural network responsible for modelling the player experience.

On most of the works presented, the authors focus on the process of generating the content, but mostly don't mention how to balance the difficulty once the content is generated. So on this paper we focus on the difficulty evaluation based on both authorial content, which we refer to as controllable features, and gameplay sessions information, which we refer to as uncontrollable features. For this purpose, neural networks were employed to classify chunks, but the classification technique itself is not the scope of this work and other techniques could be used for this purpose, such as decision trees or nearest-neighbor classifiers.

III. Testbed Game

The testbed game that was used for our studies is the endless running title Boney the Runner [12], which as of writing is available for download on the Apple Store [12] and Google Play Store [13].

On Boney the Runner the player controls the character Boney, which is a skeleton that came back to life and is being chased by a pack of hungry dogs. Fig. 2 shows a gameplay session from the game during its development, and the main game elements are also presented on this image: the skeleton

Boney, the dogs, golden coins and a tomb, which is an obstacle.

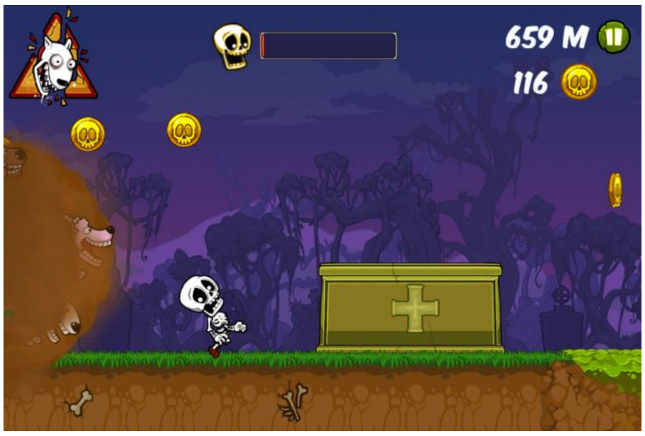

Fig. 2. A gameplay session of the game Boney the Runner.

During a gameplay session, Boney automatically runs to the right, and the player's input consists of touching the screen to make Boney jump – the longer the player touches the screen, the higher the jump, up to a maximum height. The dogs chase after Boney during the whole game, but since they are slower, Boney is capable of outrunning them. However, some obstacles can affect Boney negatively and make him walk slower or stop, and at that moment, the dogs get closer. Once the dogs catch Boney, the game ends.

As Boney runs, various obstacles and elements are presented along the way, these are highlighted in red on Fig. 3, which is a composition of various gameplay images, each containing one element and numbered for convenience. Below is a description of each element:

1) Tomb – Tombs can be avoided if Boney jumps over them, but if Boney fails to jump over, he will have to climb. While he is climbing, Boney stands still on the same spot for some time, which allows the dogs to get closer.

2) Zombie Mud – While over the Zombie Mud, Boney gets significantly slower and only recovers his maximum speed once he gets out of the dark-brown area on the ground.

3) Ghost – A hovering obstacle that stands still in the air. When Boney fails to avoid a Ghost and touches it, he will stand still for some seconds, but after that, he will start running again.

4) Moss – The moss is a green element placed on the ground, and once Boney starts to walk over it, his speed will greatly increase. This effect only happens while Boney is in contact with the moss, so if Boney jumps during it, he will slowly go back to his normal speed.

All these elements are combined in structures defined as Chunks, which are randomly placed in the track based on the difficulty that a player would face to traverse them.

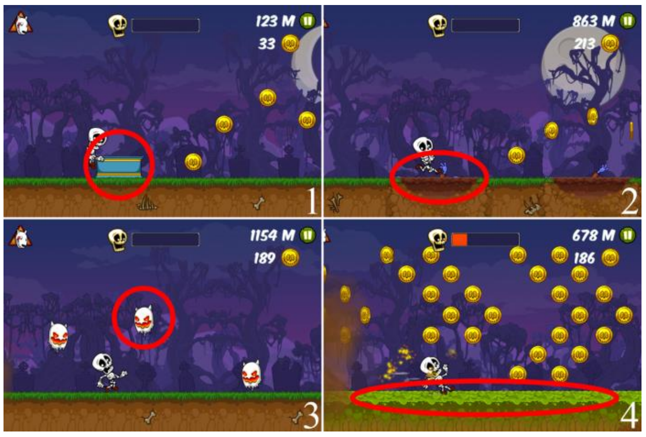

Fig. 3. The various kinds of gameplay elements present on the game. These elements are numbered to facilitate the identification, number one corresponds to the tomb, number two are the zombie hands, number three corresponds to the ghosts and number four is the moss which covers the ground.

IV. Session Generation Process

The session generation process that was adopted consists of four steps:

1) Definition of how the chunks will be placed along the track – this definition must take in consideration the fact that gameplay sessions may last for a potentially infinite amount of time.

2) Chunk generation – all chunks are created beforehand so that they can be placed in the track during the session.

3) Chunk classification – the chunks generated on the previous step are evaluated and classified according to their difficulty.

4) Chunk placement – as the gameplay session progresses, the segments must be placed on the track while following the constraints defined on the first step.

During the development of Boney the Runner, all these steps, except for the chunk placement, were executed manually. In order to determine how chunks were placed, two curves of difficulty-over-time were used, which can be seen on Fig. 4. The first difficulty curve indicates how the game difficulty flows along the first minutes of gameplay. The second corresponds to another curve that runs on an infinite loop, so when the player reaches its end, this curve starts again.

As previously stated, the difficulty curves are used by the game to determine the sequence of chunks that should be placed. Attempts at using curves based on continuous values were made during the development process, but that made the difficulty evaluation of chunks by the designers harder, so the team opted to use only discrete values for difficulty instead.

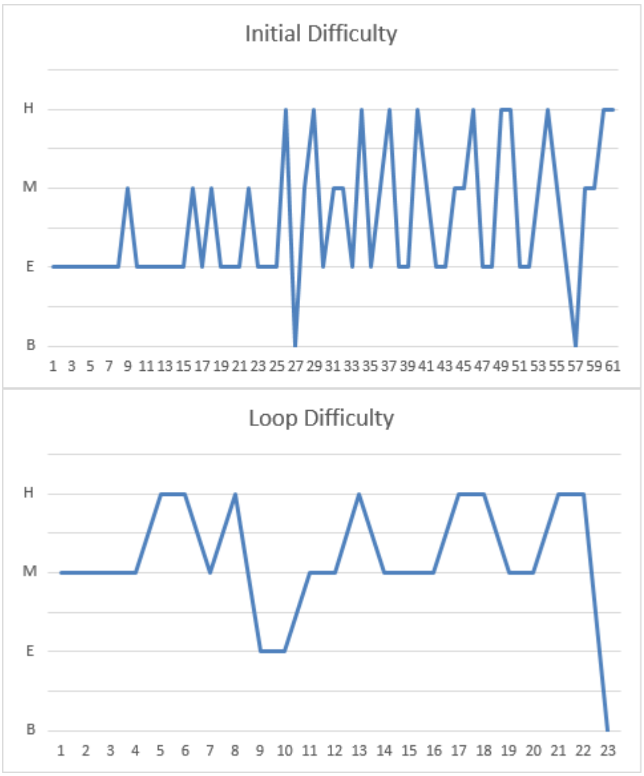

Fig. 4. Difficulty curves defined by the designers. On the horizontal axis the chunk number can be seen, on the vertical axis, the four chunk difficulties are available. The difficulties are "B" for bonus, "E" for easy, "M" for Medium and "H" for Hard.

The chunks are placed during the gameplay sessions exclusively according to the curves, meaning that each chunk is placed independently and the difficulty of previous chunks do not affect which ones will be placed later on the game.

Chunks were manually created and evaluated by game designers, which classified them according to the difficulty that the player would face in order to traverse them. Once the chunks were classified, they were divided into three pools according to their difficulty – namely Easy, Medium and Hard.

In order to guarantee more coherence during chunk classification, each chunk had follow two pre-determined constraints:

1) All chunks must have precisely 1536 pixels in width and 640 pixels in height.
2) Chunks must be assembled only by using the basic components: coin, tomb, zombie mud, ghost and moss.

For the first constraint, no information regarding the ideal chunk size was found on the literature, so the production team decided to determine the size based on the base device for which the game was projected, the iPod Touch 4 [17]. Thus, the chunk height is the same as the device's height when on landscape mode, which is 640 pixels, and the chunk's width is equivalent to 1.6 times the device width on landscape mode, which is 960 pixels. The production team opted to make the chunks horizontally larger to give the game designers more freedom when designing the chunks.

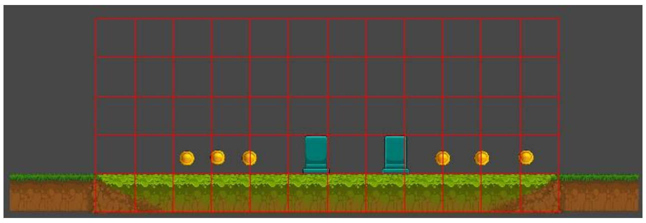

Fig. 5. A chunk seen inside the chunk editor.

The second constraint was chosen based on the approaches used by other works [8] [9], on which the use of only basic elements made the creation process more straightforward.

Fig. 5 shows a chunk as it is seen inside the game's chunk editor, note that the red grid limits the chunk's horizontal and vertical reach.

During a gameplay session, the chunks are placed on the track according to the difficulty determined by the current difficulty curve. Therefore, every time that a new chunk has to be placed on the level, the currently expected difficulty has to be obtained from the difficulty curve, and after that, a random chunk of the given difficulty is picked from the corresponding pool.

Although this process works well for the game, it presents one major flaw: the chunk classification. Since the classification is based on the game designers' intuition, it might present mistakes. In addition, normally the game designers have a vast amount of experience with the game during its development, and that might affect how he perceives the chunks' difficulties.

As previously stated, on this paper we focus on the third step, chunk classification. Our goal is to create a system capable of automatically evaluating and classifying chunks according to pre-defined criteria in order to increase the correctness of the chunk classification process.

V. DATA COLLECTION

To perform the chunk classification, we used an approach similar to that employed by [11] – our system collects data from various gameplay sessions, and this data is used to model the perceived difficulty of each chunk. The main difference that our approach presents is that we model the chunk difficulty instead of players' emotions, as is the case with [11].

That way, the methodology that we chose to use is the following:

1) Players are invited to play game sessions, and on each session, we obtain data related to the player's performance.
2) Based on the player's performance, the chunks are reclassified utilizing the key metric "Difference between needed time and regular time" (GF_TD), which corresponds to the difference between the amount of time that the player needed to traverse

the chunk and the amount of time that the player would need if there were no influences of external elements.

3) Two neural networks responsible for reclassifying the chunks are created by using data collected from the players' game sessions.

Therefore, it can be said that four classifications exist:

1) The original classification made by the designers.
2) The reclassification based on the GF_TD metric which was collected during gameplay sessions. This is considered the Baseline Classification, because it directly reflects actual gameplay data.
3) The other two classifications made by the neural networks.

In order to conduct the experiments, a special version of the game was created. On this version, 230 chunks are available, all of which were created and classified by game designers. Of said chunks, 99 were classified as Easy, 62 as Medium, 55 as Hard and 14 were special Bonus chunks. The bonus chunk do not offer any danger to the player and they serve the purpose of creating regions where the player can relax for some seconds and collect coins before the next chunks.

During the collection process, two types of data were collected for each chunk:

1) Controllable Features – these correspond to the chunk aspects that can be directly influenced by the designer, such as obstacle amount and the width of each obstacle.
2) Gameplay Features – these are related to how the player interacts with the game. It covers information such as how long the player takes to traverse a chunk.

The features that were collected for each kind of data were determined by discussions with game designers. On these discussions, the game designers were questioned about what they took in consideration to decide the chunk's difficulty and what should be monitored during gameplay sessions to validate said difficulty.

All game designers responsible for the game were present during these discussions and each one could give their opinions and feedbacks regarding their colleague's comments. At the end of the discussions, the authors of the paper and the designers would determine what features better encapsulate most of the relevant data present on a gameplay session. Each one of these features will be described in detail below.

A. Controllable features

The controllable features are calculated based on the positioning of basic elements such as tombs or coins on the chunk. Although the game designers don't directly specify these metrics, it's possible to affect them indirectly through the positioning of basic entities.

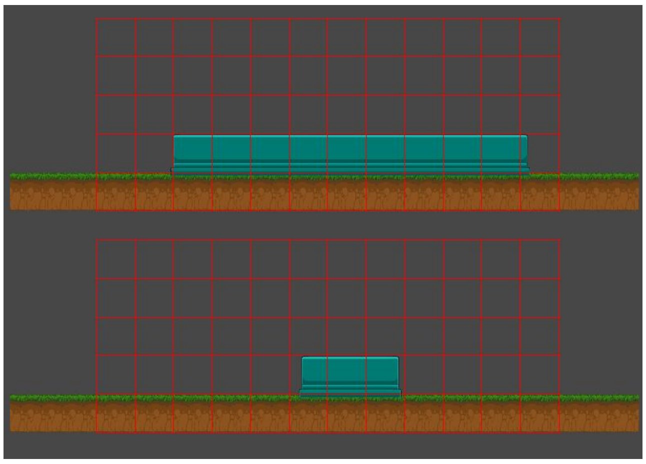

Fig. 6. Two chunk examples as seen from inside the chunk editor. This image shows the same element – the tomb – with different widths.

In addition, it's important to note that the tomb, mud and moss might present width variations, as it's possible to see on Fig. 6. Because of that, when calculating features related to these elements, we first divide them into smaller slices with the fixed size of 72 pixels, which is the minimal size for these entities. That way, a tomb with 720 pixels would be converted to 10 adjacent tombs, each with a width of 72 pixels. We denominated these slices as "segments", therefore on the previous examples we would have 10 tomb segments.

The measured controllable features were separated according to the type of element to which they belong in order to better organize the information.

For the tombs it's important to note that they can appear on three height varieties – small, average and tall – but their widths may be freely modified by the game designers. The tomb features are:

- Amount of tall tombs – by utilizing the previously described slicing method, the total amount of tall tomb segments can be obtained from the chunk.
- Amount of average tombs – concept similar to that of tall tombs, but applied to average tombs.
- Amount of small tombs – concept similar to that of tall tombs, but applied to small tombs.
- Average height – the average height of all types of tomb segments present in the chunk.
- Height Variance – the height variance of all kinds of tomb segments present in the chunk.
- Average position – the average position on the x-axis of all kinds of tomb segments.
- Position variance – the position variance on the x-axis of all kinds of tomb segments.
- Percentage of area with tombs – percentage of the terrain that is covered by tomb segments.

For the ghost element, the following features were considered relevant:

- Ghost Amount – the total amount of ghosts on the chunk.
- Average X position – the average position of ghosts on the x-axis.
- X position variance – the position variance on the x-axis.
- Average Y position – the average position of ghosts on the y-axis.
- Y position variance – the position variance on the y-axis.

The following features were chosen for the zombie mud:

- Segment amount – the total amount of mud segments on the chunk.
- Average segment position – the average position on the x-axis of mud segments.
- Segment position variance – the position variance on the x-axis of all of mud segments.
- Percentage of area with mud – percentage of the terrain that is covered by mud segments.

The features chosen for the moss are similar to those used for the zombie mud due to how these elements are arranged on the chunks:

- Segment amount.
- Average segment position.
- Segment position variance.
- Percentage of area with mud.

The features chosen for the coins are similar to those used for the ghosts due to how these elements are arranged on the chunks:

- Coin Amount.
- Average X position.
- X position variance.
- Average Y position.
- Y position variance.

B. Gameplay features

The gameplay features are the result of player actions during the sessions. For each session, the system collects data regarding events that have occurred on each chunk of the track. The gameplay features are explained below:

- Initial distance between Boney and the dogs – The distance between Boney and the dogs on the moment that Boney enters the chunk.
- Final distance between Boney and the dogs – the distance between Boney and the dogs on the moment that Boney exits the chunk.
- Time to traverse the chunk – the time that the player has taken to go from the start to the end of the chunk.
- Death occurred on the chunk – assumes the value 1 if Boney was captured by the dogs on the chunk.
- Difference between needed time and regular time (GF_TD) – as previously defined, this metric is related to how much time the player has spent on the chunk. Since all the chunks have a fixed width, it's possible to determine the time that the player would need to traverse the chunk if no other element was present on the chunk, this time is 2,4s. This feature will present positive values if the player was delayed by an element, negative values if he was propelled by the moss, and zero if nothing affected him while he was traversing the chunk.
- Total amount of collected coins – the amount of coins that the player collected on this chunk.
- Percentage of collected coins – the percentage of coins that the player collected on this chunk.

C. Collection

The data was collected from sessions played by individuals with varied levels of experience and skill with games of this genre. In total, we collected data from 184 sessions, from which 53 were considered invalid, thus leaving 131 sessions for analysis. After some experiments, we concluded that if the game did not receive any input from the player, Boney could safely reach the fifth chunk most of the time, but would rarely get past it. So these might be sessions where the player forgot the game open and left it running, which is something that would provide us with invalid data. Therefore, these are the sessions that we considered as invalid.

It is important to reiterate that all these features were collected on a per-chunk basis. So that means that every time that Boney got out of a chunk or was captured by the dogs on a chunk, the data was submitted and was associated with that given chunk.

Since various sessions were played, we obtained a considerable amount of repeated data for the same chunks, so we averaged the data for each chunk. That way only the averages were associated with each chunk. After that, all data was normalized for an interval between 0 and 1 so that it could be properly used in the neural networks.

VI. Modelling Chunk Difficulty

With the collected data, we tried to make an association between the features and the perceived chunk difficulty. Based on the findings from the collected data and through discussions with game designers, we concluded that it would be reasonable to relate perceived chunk difficulty with the time that the player has lost on each chunk, which corresponds to the GT_TD metric. For this kind of game, a chunk can be considered hard when the player has a higher chance of committing mistakes, which consequently leads to wasting more time than expected on the chunk.

Keeping this concept in mind, we removed bonus chunks from our list since they did not offer dangers to the player and are inserted in the game only to create a moment where the player can relax. With the removal of the bonus chunks, 216 chunks remained.

The GF_TD metric was then used to create a new chunk classification, which is the previously mentioned Baseline Classification. To create such a classification, the chunks were sorted in ascending order according to this feature, and new sets were created. These sets had the same sizes that the game designers originally determined: 99 easy chunks, 62 medium chunks and 55 hard chunks.

After this division, it was possible to establish the association between the perceived chunk difficulty and the GF_TD feature, as it can be seen below:

- Easy chunk – GF_TD value below 0.45
- Medium chunk – GF_TD value superior to 0.45 but inferior to 0.52
- Hard chunk – GF_TD value superior to 0.52

With this reclassification, it was possible to notice a divergence between the original classification by the designers and the findings from the collected data – 48% of the chunks that were classified by human designers were in in a different group after this reclassification.

However, only this reclassification is not enough, after all the system must be capable of classifying the chunks by using all the data that was collected previously, specially the controllable features that can be easily modified during the chunk construction. Because of that, two neural networks with different purposes were created: one, which receives only controllable features as input – Network A –, and another one which receives both controllable and non-controllable features as input – Network B. Both networks have chunk difficulty as the output.

The purpose of creating two networks lies in the different needs that exist during the game lifecycle. During the process of creating chunks, we do not have access to gameplay data, after all no player has yet had the opportunity of playing such chunks. Of course, it would be possible to include external players during this step, but this would increase the production overhead, which is undesirable. Besides, we are aiming at creating an automatic system that makes the process faster and more precise, so the need for more human interaction would remove some of the autonomy from this system.

Therefore, the Network A is destined to be used during the initial stages of development, during which we only have access to controllable features from the chunks.

The Network B, on the other hand, should be used to make periodic adjustments to the game after it has been made available. With the release of the game, it's possible to collect data related to the gameplay features and use this information in combination with the controllable features to reclassify existing chunks. That way, as more data becomes available, there is an overall tendency of the chunks converging towards the real difficulty perceived by the players.

Both networks were created using the software Weka [14], and are Multilayer Perceptrons with one hidden layer each. The amount of nodes on the hidden layer is determined by (1), which is the default value defined by Weka.

$$N = (A + C) / 2 \tag{1}$$

On (1), the variables are:

- N – the amount of nodes on the hidden layer
- A – the amount of attributes
- C – the amount of classes

To train and validate the networks, the data was randomly divided in two sets: a training set with 2/3 of the data, and a validation set with 1/3. The number of epochs used for both networks was 500, and the maximum amount of validation errors was 20. We created various networks by combining four values for the learning rate – 0.15, 0.3, 0.45, and 0.6 – with four values for the momentum – 0.2, 0.4, 0.6, and 0.8 – totalizing 16 combinations for each network. The networks with the lowest test Mean Square Error, or MSE, were chosen on each case.

For the Network A, the best performing configuration had a learning rate of 0.15 and a momentum of 0.4, while for the Network B, the best performing configuration had a learning rate of 0.15 and a momentum of 0.8. More details about the networks, such as the number of nodes on the hidden layer, can be seen on Table 1.

TABLE I. CONFIGURATIONS USED FOR THE NEURAL NETWORKS

	Network A	*Network B*
Epochs	500	500
Maximum validation errors	20	20
Learning rate	0.15	0.15
Momentum	0.4	0.8
Nodes on the hidden layer	15	18

VII. DISCUSSION

Both networks presented a good performance. Network A was capable of correctly identifying 70% of the instances with a Mean Square Error of 0.4, while the Network B was capable of correctly classifying 90% of the instances with a Mean Square Error of 0.25.

As it can be seen on Table 2, the removal of the gameplay features caused a great impact on Network A when comparing it to Network B, which relates to the advantage of collecting gameplay data in order to model the perceived chunk difficulty.

TABLE II. RESULTS OBTAINED ON EACH NETWORK

	Network A	*Network B*
Test Mean Square Error	0.4	0.25
True Positive Rate	0.7	0.904
False Positive Rate	0.18	0.041
Area under the ROC curve	0.81	0.951

Still, even though Network A did not present a performance as good as that of Network B, it has shown a good capacity of modelling the perceived difficulty of a chunk, especially when compared to the original classification presented by the game designers, for which 52% of the chunks were in the same groups as in the Baseline Classification.

The fact that the game designers were capable of predicting only 52% of the chunks might be attributed to some aspects of the game production process, such as the fact that three different designers were responsible for creating and evaluating each chunk during different work sessions. Since each designer might have different perceptions of the chunk difficulty, that increases the chance of divergent classification.

Besides that, it's possible that the designers' perception of difficulty might be affected when producing various different chunks over a period of time due to various human factors, such as experience with the game and fatigue. This is certainly a very relevant topic and that would warrant further investigation on future works.

VIII. CONCLUSION

On this work, we presented an approach for the generation of gameplay sessions on endless games, a genre that still is not widely explored on the literature. This approach includes a 4-step process that goes from the creation of the necessary content to the content placement across the gameplay sessions.

A robust chunk evaluation system was also presented. This system uses both controllable features that can be manipulated by a human designer or level generation system and gameplay features that can be obtained from gameplay sessions. The use of two neural networks is also a novel approach, which goes in line with the concept of game as a service, after all our system is designed not only to work during the game development process, but also support it along the game's lifecycle.

Although the system was tested only on an endless running game, there is no reason why it shouldn't be applicable on other kinds of endless titles. As long as the concept of chunks and difficulty curves are present, it should be easy to adapt the system to different kinds of games.

The work also presents great potential for improvements and expansion. For instance, the other three steps of the process could also be automated in order to create a full-fledged system capable of generating chunks, evaluating and placing the chunks accordingly. In addition, the current version of the system only considers local chunk difficulty, so it would be possible to consider the accumulated difficulty or to take in consideration the difficulty of previous chunks to decide the placement of upcoming chunks. Other possible improvement would be the use of dynamic difficulty adjustment systems to tailor the game's difficulty to each individual player.

ACKNOWLEDGMENT

The authors would like to thank BigHut Games [15], the developers of Boney the Runner, for allowing the use of the game on this study and for also for allowing discussions with the development team. And also, our sponsor CAPES [16].

REFERENCES

[1] C. Morris, "Analysis: Are Long Development Times Worth The Money?", gamasutra.com, September 10, 2010. [Online]. Available: http://www.gamasutra.com/view/news/30339/Analysis_Are_Long_Development_Times_Worth_The_Money.php [Acessed: October 13, 2012].

[2] G. R. Wichman, "A Brief History of "Rogue"", web.archive.org, 1997. [Online]. Available: http://web.archive.org/web/20080612193401/http://www.wichman.org/roguehistory.html [Acessed: October 13, 2012].

[3] K. Compton and M. Mateas, "Procedural Level Design for Platform Games," in Proc. 2nd Artificial Intelligence and Interactive Digital Entertainment Conference (AIIDE '06), Stanford, CA, 2006.

[4] N. Shaker, G.N. Yannakakis, and J. Togelius, "Towards Automatic Personalized Content Generation for Platform Games," Proc. Artificial Intelligence and Interactive Digital Entertainment,pp. 63-68, Oct. 2010.

[5] M. Hendrikx, S. Meijer, J.V.D. Velden, and A. Iosup, "Procedural Content Generation for Games: A Survey," ACM Transactions on Multimedia Computing, Communications and Applications, Vol. -, No. -, Article 1, January 2012.

[6] farbrausch [Online]. http://www.farb-rausch.de/ [Accessed: July 12, 2013].

[7] Zettai Hero Project [Online]. http://nisamerica.com/games/zhp/ [Accessed: July 12, 2013].

[8] G. Smith , M. Treanor , J. Whitehead, and M. Mateas, "Rhythm-based level generation for 2d platformers," Proc. Found. Digit. Games, pp.175 -182 2009.

[9] N. Sorenson, and P. Pasquier, "Towards a Generic Framework for Automated Video Game Level Creation," Proc. European Conf. Applications of Evolutionary Computation, pp. 130-139, 2010.

[10] M. Kerssemakers, J. Tuxen, J. Togelius, and G. N. Yannakakis, "A procedural procedural level generator generator," IT University of Copenhagen, 2300 Copenhagen, Denmark.

[11] C. Pedersen, J. Togelius, and G.N. Yannakakis, "Modeling Player Experience in Super Mario Bros," Proc. IEEE Symp. Computational Intelligence and Games. pp. 132-139, Sept. 2009. Boney the Runner.

[12] Apple App Store. [Online]. Available: https://itunes.apple.com/app/boney-the-runner/id573242168?mt=8 [Accessed: July 12, 2013].

[13] Google Play. [Online]. Available: https://play.google.com [Acessed: October 13, 2012].

[14] Weka. [Online]. Available: http://www.cs.waikato.ac.nz/ml/weka/ [Acessed: October 13, 2012].

[15] BigHut Games. [Online]. Available: http://www.bighutgames.com/ [Acessed: October 13, 2012].

[16] CAPES. [Online]. Available: http://www.capes.gov.br/ [Accessed: July 12, 2013].

[17] iPod touch (4th generation) - Technical Specifications. [Online]. Available: http://support.apple.com/kb/sp594 [Acessed: August 28, 2013].

[18] Jetpack Joyride Google Play Store Listing. [Online]. Available: https://play.google.com/store/apps/details?id=com.halfbrick.jetpackjoyride [Acessed: August 28, 2013].