Unity3D-based Neuro-Evolutive Architecture to Simulate Player

Adriano Mendes Gil, Paulo Renato de Barros Mendonça, Bruno George de Melo Monteiro
INdT

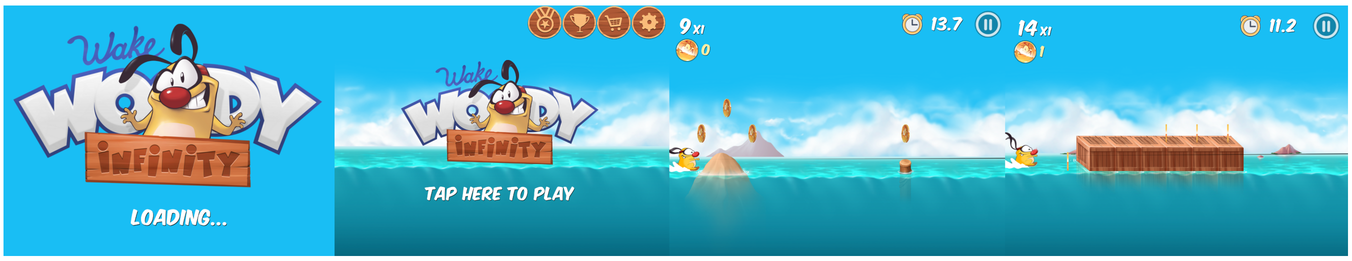


Figure 1: *Some screenshots from Wake Woody Infinity*

Abstract

Procedural Content Generation has interesting advantages, like fast production of game assets, and also working as a straightforward way to increase variability of game elements and gameplay. One of the challenges of a procedural approach is how to evaluate the generated content. Player action simulation can be used to test out levels and gameplay issues, feasibility of human-made levels and can help defining an ideal curve for progressing the difficulty level throughout the game. We present a Neuro-Evolutive agent to simulate players in games, specifically in the infinite runner genre. In some endless runners, levels are formed by combining level chunks, which are basic level structures consisting of sets of game elements arranged by a level designer or an algorithm. A tool for analysing level chunks was implemented on Unity3D and employed on the mobile cross-platform game WakeWoody Infinity. This tool allows fast evaluation of generated level chunks in terms of feasibility and difficulty classification. In this paper, the learning capacity of the proposed architecture is evaluated to achieve simple objectives, maximising coins collected and distance travelled.

Keywords: Neural Network, Genetic Algorithms, Simulated Game Playing

Author's Contact:

adriano.gil@indt.org.br
paulo.r.mendonca@indt.org.br
bruno.monteiro@indt.org.br

1 Introduction

Procedural Content Generation (PCG) on games is very useful for decreasing costs of content creation. Current methods can generate several types of content, such as sound, textures, terrains, buildings, cities, game elements behaviors, and game scenery. However PCG is difficult, since the generator has to follow all constraints imposed by a human game designer and return instances that make sense to the player experience.

How to understand the value of generated content in terms of how it impacts the overall gaming experience? PCG content must be evaluated in order to better fit in the desired player experience. A model of player behaviour could help the task of predicting the induced experience from generated game content.

General Game Playing techniques could be useful to create a smart player model for evaluating procedurally generated content. As a first step towards this goal, this works proposes a specific game playing approach targeting endless runners.

Wake Woody Infinity is a 2D side scrolling endless runner mobile game that's been released at the Windows Phone Store, iTunes and Google Play. In an endless runner, the player needs to keep running as fast as possible while avoiding hitting obstacles. Endless runners' levels are usually created on the fly by an algorithm tasked with spawning small segments of the level. Each one of these segments are called level chunks. Figure 1 shows some screenshots of the game, where some chunks are placed. Wake Woody Infinity gameplay requires fast reflexes from the player, who needs to give the commands necessary to avoid obstacles while getting as much coins and power-ups as possible. These commands are usually taps, double taps or long presses on the smartphone's screen.

Unity3D [uni] is a well known tool in the games market, specially among independent developers. It offers a features-rich game engine, with a components-based architecture that supports Javascript and C# scripting. Unity3D also works for 2D games through an API that involves 2D versions components for rendering, physics simulations and others. One of the great advantages of this engine is its flexibility, allowing developers to extend the engine by means of plugins or classes using the UnityEditor API.

Wake Woody Infinity was developed in C# using the Unity3D engine. Considering that an endless runner game requires a heavy use of colliders, rigid bodies, i.e., physics components, to properly detect collisions, a model of the player should also react using reasonable physics. Therefore an implementation of a player model inside Unity3D is advantageous because it will use the same physics component that the real game also utilize.

This paper approaches the problem of evaluating procedurally generated content in endless runner games using a model for player behaviour. To this end, we propose a Unity3d-based AI agent architecture using NeuroEvolution to aid human game designers to test out level chunks in terms of feasibility and player learning speed.

2 Related Work

A survey of Procedural Content Generation methods is presented in [Hendrikx et al. 2013], where some uses of PCG are listed. That work claims that current commercial games demand a crescent need for PCG methods. Since the production of AAA-quality games may require a big development team, the required budget to allocate the necessary resources could be an issue. Then PCG could lighten the burden on artists and designers.

[Hendrikx et al. 2013] recommends that a (semi-)automatic PCG system should also evaluate the generated content. Its necessary to understand how the generated content influences player behaviour and how can such content fit the game in a better way. Such questions help in comprehending how to approach the PCG problem and how can it be used to create the best possible game experience.

In [Nelson and Mateas 2007] is described a set of methods to generate games in the style of Nintendos WarioWare mini games. The generated games are evaluated according to a heuristic measure based on a number of constraints. The work [Togelius and Schmidhuber 2008] proposes generation of complete games using a measure of fun. Fun is achieved by the right amount of challenge, which could be measured by the time a learning mechanism would take

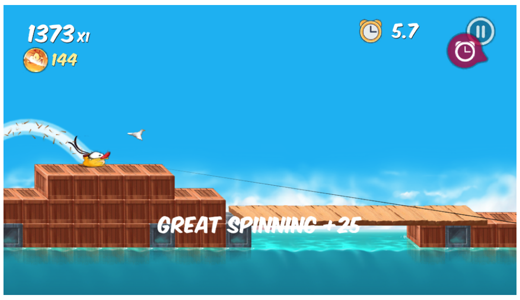


Figure 2: *Woody landing on a wooden box*

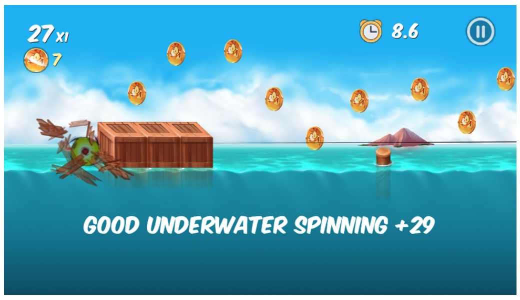


Figure 3: *Wooden boxes can be broken*

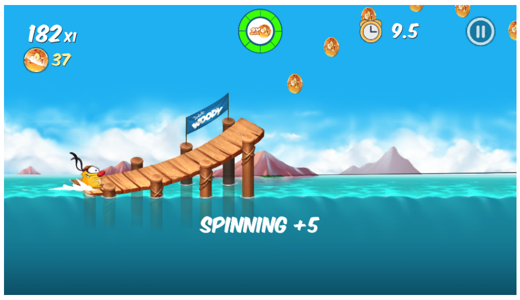


Figure 4: *Ramp can increase Woody velocity*

to learn how to beat that game. In addition, a neural controller trained by an evolutionary computation is employed in measuring such learning.

As shown in these works, algorithms that learn how to play games could be very useful in production. In the literature there are some works that use it to automatically discover exploits/bugs in games [Denzinger et al. 2005], modelling behaviour of human players [Fountas et al. 2011], or allowing player to train NPCs [Stanley et al. 2005]. Our purpose is an agent architecture to learning how to play endless runner games in order to evaluate automatically generated content.

One of possible roles of an algorithm that learned how to play a game is to improve the player experience. That can be reached by inducing the immersion state [Machado et al. 2012] or flow state [Chen 2007]. The Schmidhuber's theory of artificial curiosity [Schmidhuber 2006] relates the flow state to the predictability of environments. Predictable behaviours break down the immersion and, consequently, reduces the fun factor.

In [Carvalho et al.] a set of organised basic elements - aka level chunk - of an endless runner is evaluated according to its estimated difficulty. Neural Networks are employed to classify the difficulty level of level chunks according to controllable features and player performance data. The paper relates level chunk difficulty with the time the player takes to cross that chunk.

According to [Machado et al. 2012] an agent could be represented by a weighted sum of a set of variables representing specific features of the game as shown in figure 1. In that work, such variables and weights are defined by the observation of behaviours to infer an agent model. Such representation could also be used as a fitness function to train an agent how to better proceed in order to achieve the game objectives.

According [Machado et al. 2012] an agent could be represented as a weighted sum of a set of variables representing specific features of the game as shown in equation 1. In that work, such variables and weights are defined by observation of behaviours to infer agent model. Such representation could also be used as a fitness function to train an agent how to proceed better in order to achieve game goals.

$$P_m = \sum_{i=1}^{N} w_i c_i \qquad (1)$$

3 Wake Woody Infinity

Wake Woody Infinity is the sequel to the Windows Phone game Wake Woody [wak], introducing a new gameplay mechanic to the series. The game is a side scroller, endless runner with a time trial mechanic, with the player having to cross checkpoints in time in order to keep running, or in this case, wakeboarding. In this game, the player controls Woody, a cartoonish dog that loves wakeboarding. One of the player goals is to perform awesome tricks in order to increase his score (figure 2). The total distance travelled is also added to the player score. As Woody, the player must perform tricks using touch commands, like tap, double tap and long presses.

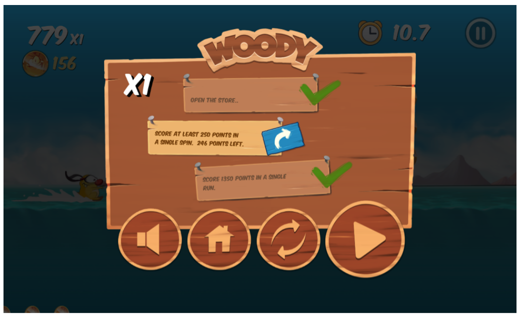
With these simple inputs, the player can jump and double jump with varying heights (short jump, long jump), perform spin moves as part of double jumps (this moves award points) and dive underwater to avoid obstacles. The player must also jump over obstacles, collect coins (figure 3), boosts (figures 4, 5) and time checkpoints (figure 5) to earn more gameplay time and keep raising his score. The final score is one of the core elements of the game and can be shared via social networks and will be tracked in the game ranking, engaging and challenging the players and their friends to beat their records. A leagues system is also featured in the game, ranking users according to their performances, pairing them with players with scores and levels similar to their own.

With the collected coins the player accumulates with each run, it is possible to upgrade power ups and buy helpful consumable items in the in-game store. If the user lacks the coins necessary to buy something, the game offers in-app purchases in the form of coin packs. There is a mission system in the game. The user always have sets of three missions to complete. These missions objectives varies widely, often teaching the game in early stages, but also challenging the user to play well in later stages. Each stage corresponds to a level. The user upgrades his level completing missions and going to the next stage. The player level also multiplies the final score of the game, so higher scores can be achieved more easily by those dedicated players that are willing to complete missions and progress further in the game. Anytime the user finds himself stuck in a mission, its possible to skip it by spending 2000 coins and prevent that mission from halting the players progress in the game.

As Woody advances, some obstacles and collectibles are presented along the way. Below, each element is described:

1. Sand banks and islands - Basic obstacles, Woody can jump over or dive under them. If the player collides with them, Woody becomes temporarily dizzy and its velocity decreases.
2. Platforms - Elements that help Woody perform some tricks and reach higher areas.
3. Ramps - Ramps that give a speed boost for some seconds and also allow Woody to jump higher.
4. Speed Rings - Floating rings that give a speed boost, helping Woody get to the checkpoint in time.
5. Checkpoints - When Woody crosses checkpoints his remaining time become 15 seconds. Meaning the player has 15 more seconds to get to the next checkpoint.

Figure 5: *Jump into a Speed Ring can increase Woody velocity*



OPEN THE STORE..

SCORE AT LEAST 250 POINTS IN A SINGLE SPIN. 246 POINTS LEFT.

SCORE 1350 POINTS IN A SINGLE RUN.

Figure 6: *Missions that player have to complete*

A set of such elements is called a level chunk. The whole level in each gameplay session is defined by the selection and placement of level chunks. Such placement is constrained by the current difficulty. A difficulty curve raises the difficulty level according to the player success in the active run.

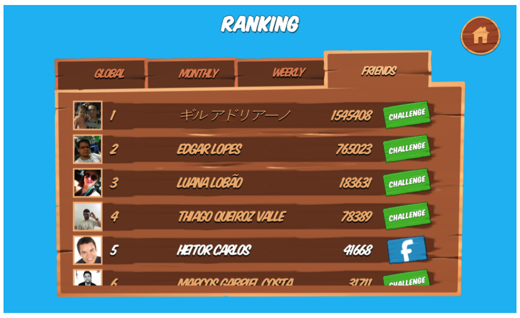
4 Neuro-Evolutive Architecture

The AI agent architecture depicted on figure 8 can be broken into three major components: perception, learning mechanism and output. The perception module combines information from the environment modelling and produces a feature vector. This feature vectors feeds the learning mechanism with a floating value output. Then, the output is converted in a player action inside the environment (jump, double jump, etc.). That architecture follows the abstract concept of a learning agent as described in [Russell and Norvig 2002].

The perception function's implementation was based on a series of raycasts traced from the player character perspective as shown on figure 9. Each frame, rays are cast and each one gives the following set of information:

- a boolean value that indicates if a game element is hit
- which type of game element, e.g, islands, platforms...
- the total distance from the player to that game element

In order to accomplish the challenge of making the player agent



1	ギルアドリアーノ	1545408	CHALLENGE
2	EDGAR LOPES	765023	CHALLENGE
3	LUANA LOBÃO	183631	CHALLENGE
4	THIAGO QUEIROZ VALLE	78389	CHALLENGE
5	HEITOR CARLOS	41668	f
6	MARCOS GABRIEL COSTA	31711	CHALLENGE

Figure 7: *Ranking among player friends*

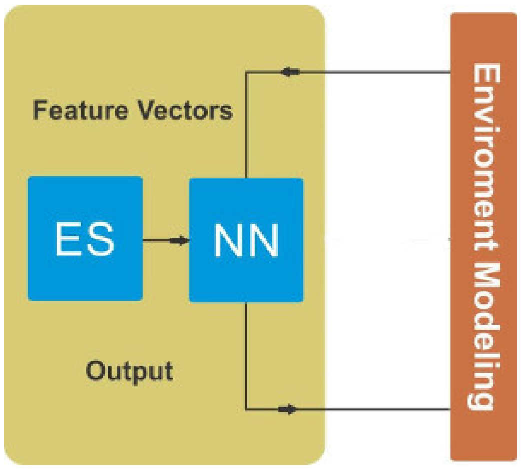


Figure 8: *Architecture of Agents High-level functions*

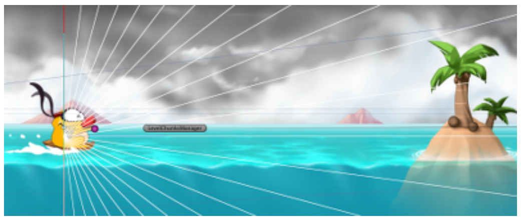


Figure 9: *Scene view showing Raycasts that are part of the agent perception*

learn how to play, some learning methods are used: 1) Neural Network to model player behaviour; 2) Genetic Algorithms - also called evolution strategy (ES) component - to alter Neural Network parameters in order to achieve the best possible fitness value. It was used a Neural Network implementation from AForge.NET Framework [afo]. A MonoBehaviour script called AIAgentBehaviour was defined with a Neural controller as class member.

Each frame the AI Agent takes a feature vector from the *Perception* object and feed it to the Neural Network instance. The output represents a successfully touch if the output is greater than a predefined value. Initially, a genetic algorithm creates a population of 30 instances. For each simulation, the neural network receives weights from the current GA instance. When the simulation is over, the current genotype is evaluated according to the fitness function. The fitness function was defined according to the equation 2:

$$P_m = w_1 c_1 + w_2 c_2 + w_3 c_3 \qquad (2)$$

Where c_1 means the total distance travelled , c_2 the total coins collected and c_3 the total points scored. The weights were defined experimentally, respectively as 0.00007, 0.001 and 0.00002. A Unity3D Editor extension was coded to launch simulations using the proposed architecture. The AI Agent implementation is totally decoupled from the Player implementation by means of the input simulation. That means that different endless runner games or even platformers can be compatible with this implementation.

5 Evaluation

To define the best parameter configuration of the neural architecture and the number of GA generations, several simulations took place in a Unity3D test scene. One of the goals with this experiment was to verify how many generations it is necessary so the AI agent can learn to jump over simple obstacles.

Figure 10 shows a graph of the fitness value along generations comparing with different neural architectures. The best results are achieved with neural architecture 145-8-1, the simplest architecture in tests.

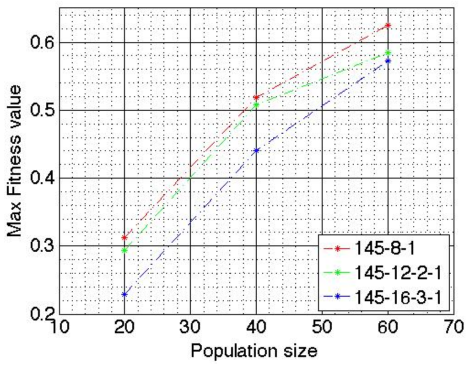


Figure 10: *Fitness value versus population size according to neural architecture*

6 Conclusion

As PCG methods become more commonplace on game production, we should think of how to evaluate it. One possible approach is using, an AI agent to test generated content. Our work describe an architecture based on NeuroEvolution and implemented on the Unity3D game engine. The architecture was used to simulate player action inside the game Wake Woody Infinity, a side scrolling endless runner. This architecture is a straightforward approach to define desirable AI behaviours using only a weighted sum as the fitness value. A more complex behaviour could be learned by using an iterative approach and several simulations with different versions of the fitness function.

The best neural architecture between current simulations was $145-8-1$, i.e., 145 inputs and 8 neurons on the first layer and 1 neuron on the output layer. One possible reason of such result is that heavy architectures require more generations to converge, while simple architectures converge faster.

The process of evolving an AI agent to play a game allows measuring the difficulty of learning how the player should interact with specific game elements. Measurements of learning depend on the precision of the modeled perception. When such metrics are derived from the game environment itself, i.e., from the Unity3d scene, the perception and action of the AI agent are the most precise model. It turns out that Unity3d is a natural benchmarking environment to spawn algorithms that learn how to play a game.

In the endless runner context, a learning mechanism could be used for classification of level chunks. Estimated player performance data can be derived from executing the AI Agent on the generated content.

GA allows a more creative AI game controller than one based only on simulated input. In future works, we plan to compare the learning performance between a Neural agent trained via an evolutive estrategy and another one trained using a supervised approach as Neural Network Backpropagation.

News ways to model perception could be achieved in order to obtain a more precise player vision model, for example, parallel rays could be traced to get a broader view of the current game scene. Another approach would be using a screenshot of the scene as the input to the learning mechanism, but it would require a heavier processing.

Acknowledgements

We would like to thank INdT for all the support during this research.

References

Aforge.net framework. `http://www.aforgenet.com/`. Accessed: 2014-07-23.

Carvalho, L. V., Moreira, Á. V., Vicente Filho, V., Albuquerque, M. T. C., and Ramalho, G. L. A generic framework for procedural generation of gameplay sessions.

Chen, J. 2007. Flow in games (and everything else). *Communications of the ACM 50*, 4, 31–34.

Denzinger, J., Loose, K., Gates, D., and Buchanan, J. W. 2005. Dealing with parameterized actions in behavior testing of commercial computer games. In *CIG*.

Fountas, Z., Gamez, D., and Fidjeland, A. 2011. A neuronal global workspace for human-like control of a computer game character. In *Computational Intelligence and Games (CIG), 2011 IEEE Conference on*, 350–357.

Hendrikx, M., Meijer, S., Van Der Velden, J., and Iosup, A. 2013. Procedural content generation for games: A survey. *ACM Trans. Multimedia Comput. Commun. Appl. 9*, 1 (Feb.), 1:1–1:22.

Machado, M. C., Pappa, G. L., and Chaimowicz, L. 2012. Characterizing and modeling agents in digital games. In *Proceedings of the XI Brazilian Symposium on Computer Games and Digital Entertainment (SBGames), SBGames*, 26–33.

Nelson, M. J., and Mateas, M. 2007. Towards automated game design. In *AI* IA 2007: Artificial Intelligence and Human-Oriented Computing*. Springer, 626–637.

Russell, S. J., and Norvig, P. 2002. Artificial intelligence: a modern approach (international edition).

Schmidhuber, J. 2006. Developmental robotics, optimal artificial curiosity, creativity, music, and the fine arts. *Connection Science 18*, 2, 173–187.

Stanley, K. O., Bryant, B. D., and Miikkulainen, R. 2005. Real-time neuroevolution in the nero video game. *Evolutionary Computation, IEEE Transactions on 9*, 6, 653–668.

Togelius, J., and Schmidhuber, J. 2008. An experiment in automatic game design. In *Computational Intelligence and Games, 2008. CIG'08. IEEE Symposium On*, IEEE, 111–118.

Unity3d website. `http://unity3d.com/`. Accessed: 2014-07-10.

Wake woody. `http://www.windowsphone.com/pt-br/store/app/wake-woody/864a1e57-32db-4c82-b246-2b0ffb2bd55e`. Accessed: 2014-07-23.