A. N. Onymous[1⋆]

Anonymous Institute

Abstract.

1 Introduction

⋆ No Institute

1.1 Motivation

It is common for AI researchers to turn to games in general as a testing environment for AI techniques. Some of the earliest problems that AI attempted to solve were checkers and chess in the 50s, even before AI was defined and recognized as a field in the Dartmouth workshop in 1956 [**?**].

What made board games attractive in the first place? They have a rather simple set of rules but winning a game could be a challenging task even for a human brain. It is not surprising that soon video games too drew researchers' attention—along with board games. They offer a wide range of dynamic and competitive elements that resemble real-world problems to some extent.

Of course, this interest works in both ways. Many video games use AI techniques to deliver better experiences, mainly involving Non-Playable Character(NPC) behaviour and Procedural Content Generation (PCG).

Having this relationship between academia and industry in mind, the IEEE Conference on Computational Intelligence and Games started as a symposium in 2005, and as a conference in 2009. It brings professionals from both fields together to discuss the latest advances in AI and Computational Intelligence and how they apply to games.[**?**]

Among other events, the conference hosts competitions. In 2018, most proposed competitions are centred around playing AI agents for specific games or genres. One of the exceptions is the *3rd Angry Birds Level Generation Competition*. Participants must build computer programs that are able to generate levels for the *Angry Birds* game.

Angry Birds is a mobile game by Finnish company Rovio Entertainment Corporation[**?**], first launched in 2009. The game was a huge success in the *AppStore*—being the most downloaded mobile game in history. Having been ported to several other platforms since, it has an animated movie and way too many *spin offs*—around seventeen. In the game, green pigs have stolen the birds' eggs, and they proceed to rescue them. The pigs have built a variety of defensive structures made out of blocks, so the player has to fire the birds from a slingshot—apparently they never learnt to fly—so the structure is destabilized.

However, what is interesting to us is not its wide success, but how heavily the game relies in gravity to create interesting puzzles. The main challenge is to build stable structures that are robust enough to take more than a single shot before crumbling to the ground.

1.2 Objectives

The ultimate objective of this project is to build a program capable of creating levels for *Angry Birds*. This by itself is too vague to convey the desired approach for the project, so we can break it down to the following, more concise, objectives:

- Explore the expressiveness and variability of SBPCG using evolutionary programs.
- Adapt the game to extract data from execution, with the aim of evaluating the levels.

– Produce stable structures under gravity.
– Place other elements on the structures to complete the levels.

1.3 The necessity of Procedural Content Generation

Computer games are a relatively new form of media whose popularity has been increasing non-stop since they appeared in the 70s. Many challenges have arisen—and will keep doing so—throughout the evolution of the industry, from the early arcades to the most complex modern open-world video games. Developers have come up with all sorts of creative ways to overcome hardware limitations, delivering better graphics and audio. They have pushed the boundaries of the medium by finding new forms of interaction and engaging players with compelling storytelling and original game mechanics. Many of them are related to the fast pace at which the game industry is growing, reaching every passing year to broader audiences that demand a wider range of experiences. Crafting them requires great effort and a high consumption of resources. How do we create a vast amount of content that suits players expectations with lower investment? The answer can be replayability, adaptative content or reduction of designers' workload. All of which can be tackled using PCG.[**?**]

Replayability, also referred to as replay value, relies on how interesting playing a game more than once is. It is easy to understand why it would be a desirable feature for both players and developers: from the player's point of view, they can extend the game experience further—past the credits roll. For designers, replay value means their product offers more with less manually crafted content.

Games can also engage players by adapting gameplay elements to each individual player. In a literal sense, this would be impracticable. Instead, users are usually presented with options to adjust to their preferred style. What is more interesting, the game itself can change based on in-game player behaviour. It can regulate its difficulty level to fit the player's learning curve or create content that matches the player's taste.

Both applications above use PCG as a replacement for human designers. However, PCG can be used as a tool to assist developers. It can suggest what might be a base for later development, enhancing human creativity rather than displacing it.

1.4 What is Procedural Content Generation

The definition of Procedural Content Generation (PCG) has been broadly discussed but there is no agreement. We have plenty of examples of what *is* and what *is not* PCG, but every definition struggles to cover all cases, either being too inclusive or too exclusive. The one we choose here balances well between the two, defining PCG as *the algorithmical creation of game content with limited or indirect user input* [**?**].

Although PCG often uses AI techniques, this definition does not include all uses of AI in games. We do not consider NPC behaviour or AI playing agents as content, thus they are not PCG either. Aesthetic elements, game rules, levels, items, stories and characters among others are considered content in this definition.

Note that neither computers nor video games are mentioned in the definition. In fact, PCG has its roots in analogical games. This may conflict with the *limited or indirect user input*, but it is reasonable to assume that following a detailed set of instructions—even if it is done by a human—is not *input*. The underlying concepts used much earlier by non-digital games still prevail in modern video games. Using an algorithm to assemble pre-designed pieces is a common technique in tabletop roleplaying guides—where the algorithm usually consists in several dice rolls—such as *Dungeon & Dragons*. It is not surprising that one of the early adaptations of PCG to digital platforms aimed to generate monsters and dungeons for physical games.[**?**]

Taxonomy There are many PCG methods and it is necessary to look at some traits that characterize and differentiate them from each other: [**?**]

- *Online/offline*: In online generation, the PCG occurs during the game session, while the user is playing the game. If it is done before the game session or during development, we have offline generation.
- *Necessary/optional*: Procedurally generated content may be part of the essential structure of the game or can be additional to the game experience and can be discarded. In the first one the content is *necessary* and needs to be correct while the latter is *optional*.
- *Degree and dimensions of control*: As any other algorithm, a PCG method can have a number of parameters which affect the output. If it uses random numbers, the seed is one of them.
- *Generic/adaptative*: Adaptative generation takes into account player behaviour while generic does not. Although there are exceptions, most commercial games choose generic over adaptative.
- *Stochastic/deterministic*: Deterministic PCG will produce the same output given the same input, in contrast to stochastic generation which is not as easy to replicate.
- *Constructive/generate-and-test*: Generate-and-test produces potentially correct solutions that are tested and adjusted in each iteration before giving the actual output. Constructive methods build partial solutions and add on them.
- *Automatic/mixed authorship*: PCG can be used as an assisting tool for designers, whether the output is used as a base or as an interactive process. Then we talk about mixed authorship, as opposed to the automatic generation where the designer does not take part.

Search-based Procedural Content Generation A special kind of *generate-and-test* approach to PCG is Search-based Procedural Content Generation (SBPCG), which is usually—but not always—tackled with Evolutionary Algorithms. The problems faced by SBPCG are not very far from those encountered in Evolutionary problems.

The test function does not determine if a solution is valid, but it grades how good the solution is. This is often called *fitness function*. In SBPCG, how

to evaluate the quality of a solution has no straightforward answer. It requires formalizing what is fun, exciting or engaging content as a *fitness function*, which are usually based in tricky assumptions.

There are three main classes of *fitness functions* in SBPCG[**?**]:

- *Direct Fitness Function* where certain features are extracted from the generated content and directly mapped to a fitness value. Those features must be easily measurable.
- *Simulation-Based Fitness Function* where an agent plays throw some part of the game that involves the generated content. The fitness is calculated using features from the agent's gameplay.
- *Interactive Fitness Function* where the fitness value is obtained from the player, whether it be explicitly by asking them or implicitly by measuring certain responses to the game.

2 Background and State of the Art

Let's have a look at some participants in previous editions of the Angry Birds Level Generation Competition.

2.1 Constructive approach

In the two solutions proposed by Matthew Stephenson and Jochen Renz [**?**] [**?**] the structures displayed on the level are constructed from the top down, in several phases.

First, they build a structure recursively, each row composed of a single type of block (with a fixed rotation). The likelihood of selecting a certain block is given by a probability table. Then, the blocks are placed using a tree structure, where the first selected block is the peak and blocks underneath it are split into subsets that support the previous row with one, two or three blocks. This ensures local stability, but not global stability, which is tested once the whole structure is completed. The second solution also tries to add variety by replacing some blocks with others of the same height.

After that, other objects (pigs and TNT) are placed. Potential positions for those objects are recorded, first trying to place them in the centre of each block, then right above the edges of it, checking if there is enough space. Available positions are ranked based on structural protection, dispersion, etc. and then filled with the desired number of objects.

The second solution also selects material—which can be stone, ice or wood—based on trajectory analysis, clustering, row grouping, structure grouping or randomly. Weak points are set to stone material and the rest using one of the previous strategies. Trajectory analysis-based strategy sets to the same material all blocks in the trajectory of a shot aimed at a particular pig. Clustering strategy takes a random block, sets its material and propagates to the surrounding blocks that have not been assigned yet. Row grouping and structure blocking apply the same material to a whole row or structure respectively.

Materials also determine which kind of birds will be used in the level, since some birds are more efficient against certain materials than others. The number of birds available for the player to solve the level is estimated using AI agents designed for a different competition. If the AI agents are unable to solve the level, it is discarded. The lowest number of birds used by the agents is the chosen for the level.

The probability table was tuned using search-based optimization methods.

2.2 Search-based approach

Lucas Ferreira and Claudio Toledo[**?**] present a solution based on SBPCG using a genetic algorithm and a game clone developed in Unity Engine to evaluate the levels.

In the genetic algorithm individuals correspond to levels, each represented by an array of columns. Each column is a sequence of blocks, pigs and predefined compound blocks, using an identification integer. This representation also includes the distances between different columns.

The population is initialized randomly following a probability table which defines the likelihood of a certain element being placed in a certain position inside a column.

For the evaluation, levels are executed in the game which stores data about the simulation. The fitness function is described as:

$$f_{ind} = \frac{1}{3}(\frac{1}{n}\sum_{n-1}^{i=0} v_i + \frac{\sqrt{(|b| - B)^2}}{Max_b - B} + \frac{1}{1 - |p|})$$

where v_i is the average magnitude of the velocity vector for block i, $|b|$ is the total of blocks in the level and $|p|$ the count of pigs. The rest are parameters: B the desired amount of blocks and Max_b the maximum number of elements.

The recombination process uses an Uniform Crossover where the new individual is generated picking for each position a column from one of the parents which occupies the same position. When the parents are not the same length, the remaining columns are selected with a 50% of probability. Mutation simply changes with a certain probability each element of the individual randomly (either being a column element or the distance between columns).

Open Source Simulator The fitness evaluation requires a simulation to test the behaviour of the levels under the game's physics. Angry Birds is not open source software and so the code is not available; therefore, a game clone was developed to fill this gap, using the Unity Engine.

Levels are described in XML files which the game takes as an input. They are parsed to run the simulation which then generates new XML files as an output. Those files contain information about the execution of a certain level such as average velocity of each element, the amount of collisions and the final rotation.

3 Problem Description: TORCS

4 Conclusions and Future Work