The 2017 AIBIRDS Level Generation Competition

Matthew Stephenson, Jochen Renz, Xiaoyu Ge, Lucas Ferreira, Julian Togelius, and Peng Zhang

***Abstract*—This paper presents an overview of the second AIBIRDS level generation competition, held jointly at the 2017 IEEE Conference on Computational Intelligence and Games, and the 26th International Joint Conference on Artificial Intelligence. This competition tasked entrants with developing a level generator for the physics-based puzzle game Angry Birds. Submitted generators were required to deal with many physical reasoning constraints caused by the realistic nature of the game's environment, in addition to ensuring that the created levels were fun, challenging and solvable. This year's competition was a significant improvement over the previous year, with a greater number of participants and more advanced generators. Within this paper we describe the framework, rules, submitted generators and results for this competition. We also provide some background information on related research and other video game AI competitions, as well as discussing what can be learned from this year's competition. There are several game and real-world applications for this type of research, and we provide some examples of the types of levels we would like future competition entries to generate.**

***Index Terms*—Angry Birds, procedural content generation, level generation, physics-based games, AI competitions**

I. Introduction

Over the past several years, many different AI competitions focused around video games have become extremely popular. Many of these competitions have yielded promising results and improvements for the wider AI community, and have been hosted at several major international conferences including CIG, AIIDE, IJCAI, ECAI, GECCO and FDG to name just a few. Whilst competitions and challenges centred around AI playing classic board games, such as chess with Deep Blue and more recently Go with DeepMind's AlphaGo [1], have been incredibly popular and successful, video games typically provide a much more complex and challenging domain in which to interact. Past video game AI competitions have mostly focused on developing intelligent agents that can play the game(s) successfully, but other competition objectives are possible. One of the most popular focuses for video game AI competitions apart from agents, is that of procedural content generation (PCG).

PCG is the automatic creation of game content without manual interaction by a human designer [2] and is a major area of investigation within the video game industry from both a research and business perspective [3]. The most common reason for utilising PCG is that it can dramatically increase the range of available content within a game whilst still being cheap and effective. Creating a large amount of high quality content is extremely time consuming if performed manually by a designer. PCG can be a good solution for many large or low-budget games by dramatically reducing a game's development time, as well as expanding the available content and lowering memory consumption [4]. PCG can also be used to create an almost endless amount of content, and helps ensure that no two experiences are likely to be the same. In particular, the ability to automatically generate a huge range of varied and complete levels allows the player to keep playing nearly indefinitely, without the game becoming too repetitive.

The most common types of "game content" that are generated are usually levels or sub-sections of levels, referred to as procedural level generation (PLG). PLG has been previously implemented in many different game types, including real-time strategy [5], [6], platform [7], racing [8], arcade [9], role-playing [10], stealth [11] and rogue-like [12]. The General Video Game AI Competition has also worked on attempting to procedurally generate levels for multiple general games [13], [14], although the results so far are somewhat mixed. Several papers have also explored the use of PLG for physics-based puzzle games such as Cut the Rope [15], [16] and, more notably for this paper, Angry Birds [17], [18], [19], [20], [21], [22], [23]. The physics constraints employed in these types of games, along with the exceptionally large state and action spaces, create many problems for PLG [24].

PLG is particularly difficult for physics-based puzzle games, as the generator must not only deal with the physical constraints of the environment, but also still ensure that levels are fun, solvable and challenging for the player. One such popular game whose levels fit into this category is Angry Birds. This game has been of interest to the wider AI game research community for many years, with the annual competition focused around developing agents to play it (AIBIRDS agent competition) drawing dozens of participating teams [25]. The type of physical reasoning required to solve levels from this game is very similar to that needed for an agent to operate successfully in the real-world [26]. Physics-based games such as Angry Birds provide an effective and realistic simulation of the real-world for AI systems to try out their algorithms. Generators attempting to create levels for this game must be acutely aware of the game's physics and know how to create content that is viable within it. Angry Birds levels typically contain multiple blocks and other objects that are stacked or arranged together to create structures. Generating and positioning these structures such that they are not only stable but present an interesting and solvable puzzle for the user is by no means an easy task. For these reasons, we believe that the challenge of creating physically stable, enjoyable and feasible levels for a game such as Angry Birds is well worth exploring and researching.

M. Stephenson, J. Renz, X. Ge and P. Zhang are with the Research School of Computer Science, Australian National University, Canberra, A.C.T. 0200, Australia, e-mail: (matthew.stephenson@anu.edu.au).

L. Ferreira is with the Department of Computational Media, University of California in Santa Cruz.

J. Togelius is with the NYU Game Innovation Lab, Tandon School of Engineering, New York University.

This article has been accepted for publication in a future issue of this journal, but has not been fully edited. Content may change prior to final publication. Citation information: DOI 10.1109/TG.2018.2854896, IEEE Transactions on Games

In this paper we present the description, entrants, results and conclusions for the second AIBIRDS level generation competition. Participating competitors developed PLG algorithms for automatically creating Angry Birds levels. This year's competition added in many new game elements and compatibility features which allowed generators to create far more sophisticated and complex levels than had previously been possible. This included the addition of multiple bird and pig types, the ability to set the size of the level, and the inclusion of several new game objects such as TNT boxes. Generated levels must also satisfy certain user-defined criteria, the specifics of which are discussed later. Participants were also able to combine their level generator with the AI agents from previous AIBIRDS agent competitions, providing them with a way to analyse the difficulty and feasibility of their generated levels. The submitted generators were evaluated by several judging panels. These panels gave each generator a rating based on the enjoyment, creativity and difficulty of the levels it created.

The remainder of this paper is organized as follows: Section II provides the background to this competition, including past AI and PCG video game competitions, a description of the Angry Birds game, and details on the related AIBIRDS agent competition; Section III describes the competition itself, providing details on the clone that is used instead of the actual Angry Birds game, as well as the rules and judging procedure; Section IV contains descriptions of the five generators submitted to this year's competition; Section V provides the results of the competition. Section VI discusses the results of the competition, providing some possible uses and improvements for the generators as well as desired goals for future competitions; Section VII presents our final conclusions.

II. Background

A. Previous AI and PCG video game competitions

Examples of popular AI competitions (both past and present) include the Mario AI Championship, which originally revolved around developing agents for solving Super Mario Bros levels [27], [28] but also had a secondary track focussing on level generation [29], the StarCraft AI Competition [30], the Visual Doom AI Competition (ViZDoom) [31], the Geometry Friends Game AI Competition [32], the Fighting Game AI Competition [33], as well as the aforementioned AIBIRDS agent competition [26], [34]. The General Video Game AI (GVGAI) Competition has also run several tracks around developing agents for playing general video games. These include the single-player planning track [35], the two-player planning track [36], [37] and the learning track [38]. There have also been several additional GVGAI competition tracks focusing on general content generation, including the level generation track [13], [14], [38] and the rule generation track [39]. Physics-based games have also been recently added to the GVGAI game collection [40], although the physics system used is significantly limited in its current capabilities. Compared to other games from previous competitions, Angry Birds presents a complex physics-engine that level generators must effectively reason about in order to be successful.

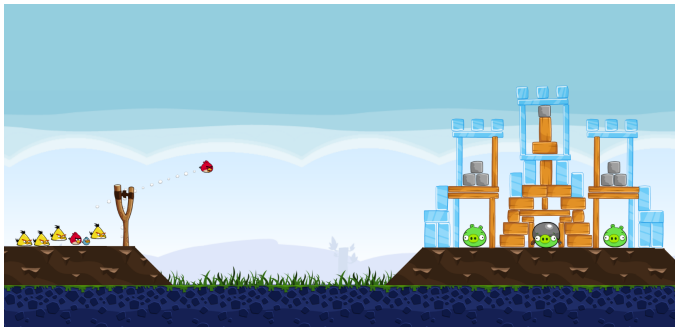

Fig. 1: Screenshot of a level from the Angry Birds game.

B. Angry Birds game

Angry Birds is a popular physics-based puzzle game where in each level the player uses a slingshot to shoot birds at structures composed of blocks, with pigs placed within or around them [41]. The player's objective is to kill all the pigs within a level using the birds provided. A typical Angry Birds level, as shown in Figure 1, contains a slingshot, birds, pigs and a collection of blocks arranged in one or more structures. All objects within the level have properties such as location, size, mass, friction, density, etc., and obey simplified Newtonian physics principles defined within the game's engine. Each block in the game can have multiple different shapes as well as being made of one of three materials (wood, ice or stone). Each bird is assigned one of five different types (red, blue, yellow, black or white). Each of these bird types are strong/weak against certain block materials, as well some types possessing secondary abilities which the player can activate during the bird's flight. The player can choose the angle and speed with which to fire a bird from the slingshot, as well as a tap time for when to activate the bird's special ability if it has one, but cannot alter the ordering of the birds or affect the level in any other way. Pigs are killed once they take enough damage from either the birds directly or by being hit with another object. The ground is flat but additional terrain squares, which are impenetrable and unaffected by gravity, can be added anywhere. TNT can also be placed within a level and will explode when hit by another object. The difficulty of this game comes from predicting the physical consequences of actions taken, and accurately planning a sequence of shots that results in success. Points are awarded to the player once the level is solved based on the number of birds remaining and the total amount of damage caused.

C. AIBIRDS agent competition

Although the AIBIRDS level generation competition is only in its second year, the AIBIRDS agent competition has been running annually since 2012. Entrants in this competition are tasked with developing an agent that can play and solve unknown Angry Birds levels. This competition was created as a means to promote the research and creation of intelligent agents that can reason and predict the outcome of actions in a physical simulation environment [34]. This type of physical reasoning problem is very different to traditional games as the attributes and parameters of various objects are often

This article has been accepted for publication in a future issue of this journal, but has not been fully edited. Content may change prior to final publication. Citation information: DOI 10.1109/TG.2018.2854896, IEEE Transactions on Games

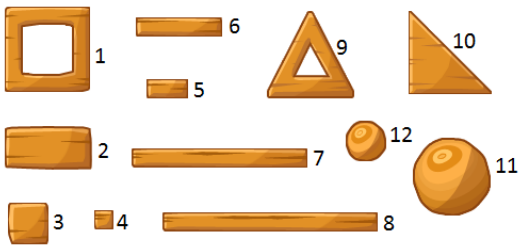

Fig. 2: The twelve different block shapes available.

Fig. 3: An example level of the Science Birds game.

```
<?xml version="1.0" encoding="utf-16"?>
<Level width="2">
  <Camera x="0" y="0" minWidth="20" maxWidth="26">
    <Birds>
      <Bird type="BirdRed"/>
      <Bird type="BirdBlue"/>
      <Bird type="BirdYellow"/>
      <Bird type="BirdBlack"/>
      <Bird type="BirdWhite"/>
    </Birds>
    <Slingshot x="-9" y="-2.5">
      <GameObjects>
        <Block    type="RectTiny"    material="wood"  x="2.54" y="-3.23" />
        <Block    type="RectTiny"    material="wood"  x="1.70" y="-3.23" />
        <Block    type="SquareHole"  material="wood"  x="2.15" y="-2.68" />
        <Block    type="SquareHole"  material="stone" x="2.2"  y="0.72"  />
        <Block    type="RectTiny"    material="stone" x="1.76" y="0.17"  />
        <Block    type="RectTiny"    material="stone" x="2.6"  y="0.17"  />
        <Block    type="RectTiny"    material="ice"   x="2.6"  y="2.91"  />
        <Block    type="RectTiny"    material="ice"   x="1.76" y="2.91"  />
        <Block    type="SquareHole"  material="ice"   x="2.2"  y="3.46"  />
        <Pig      type="BasicSmall"  material=""      x="2.15" y="-2.12" />
        <Pig      type="BasicSmall"  material=""      x="2.2"  y="1.38"  />
        <Pig      type="BasicSmall"  material=""      x="2.2"  y="4.12"  />
        <Platform type="Platform"    material=""      x="1.04" y="-0.31" />
        <Platform type="Platform"    material=""      x="1.64" y="-0.31" />
        <Platform type="Platform"    material=""      x="2.27" y="-0.31" />
        <Platform type="Platform"    material=""      x="3.47" y="-0.31" />
        <Platform type="Platform"    material=""      x="2.86" y="-0.31"  />
        <Platform type="Platform"    material=""      x="2.79" y="2.4"   />
        <Platform type="Platform"    material=""      x="3.4"  y="2.4"   />
        <Platform type="Platform"    material=""      x="2.2"  y="2.4"   />
        <Platform type="Platform"    material=""      x="1.58" y="2.4"   />
        <Platform type="Platform"    material=""      x="0.97" y="2.4"   />
        <TNT type="" x="1.11" y="-4" rotation="0" />
      </GameObjects>
    </Level>
```

Fig. 4: XML representation of the example Science Birds level from Figure 3.

III. AIBIRDS Level Generation Competition

A. Science Birds

Angry Birds is a commercial game developed by Rovio Entertainment who do not provide an open-source version of their code. Instead we use a Unity-based clone of the Angry Birds game developed by Lucas Ferreira called Science Birds [23], which is open-source and available to download from GitHub [43]. This clone provides many of the necessary elements to generate levels very similar to those of Angry Birds in a realistic physics environment. There are currently twelve different block shapes available, see Figure 2. Each block is assigned one of three materials (wood, ice or stone) and can also be rotated to any arbitrary angle. There are five different bird types (red, blue, yellow, black and white) as well as three different sizes of pig (small, medium and large). There are also TNT boxes that explode when hit, and terrain squares than can be used to make floating platforms or other static areas of the level.

The size and material of blocks impacts their physical properties and how much damage they can withstand before they are destroyed. The size of a pig also determines the amount of damage needed to kill it. The special abilities of each of bird type are described below, along with the materials that they are strongest/weakest against:

- Red bird: No special ability, neither strong nor weak against any specific material.
- Blue bird: Splits into three birds when tapped, strong against ice blocks, weak against stone blocks.
- Yellow bird: Shoots forward in a straight line with increased speed when tapped, strong against wood blocks, weak against ice blocks.
- Black bird: Explodes either when tapped or after hitting an object, strong against stone blocks.
- White bird: Drops an egg directly downwards when tapped, this egg explodes after hitting another object.

All of these described object types can be seen in the example Science Birds level shown in Figure 3. It has three blocks of each material, three pigs, a TNT box and five birds (one of each type). Moreover, it has two rows of static square platforms floating in the air.

Levels are represented internally using a XML format. This format is composed of the size of the level, the number, type and order of birds, the position of the slingshot, and a list of game objects, as shown in Figure 4. Each game object has four attributes:

- Type: String representing the type of the object.
- Material: String defining the material of a block. Valid values are only “wood”, “stone” and “ice”. Certain objects such as pigs, platforms and TNT do not need a material.
- X, Y: Float numbers representing the position of the game object. The origin (0,0) of the coordinates system is the centre of the level.
- Rotation: Float number that defines the rotation of the game object (optional).

B. Rules

To ensure that generators entered into the competition do not simply produce hand-designed levels, submitted generators must create levels in accordance with an input data file. This file contains the necessary requirements about the levels that will be generated. This is provided as four separate lines containing the following information in the given order:

- Number of levels to generate (positive integer)
- Forbidden block and material combinations (list of invalid materials/block shapes, e.g. "Stone Triangle")
- Range for number of pigs (two positive integers, minimum and maximum)
- Time limit to generate levels in minutes (positive integer)

During the competition, information about what levels to create is passed to each generator using this input file. These restrictions were not too severe, as the goal is not to generate levels for specific structure requirements, but to simply ensure that all levels are created autonomously without too much designer influence. The time limit value was always set to one hour for every ten levels, which based on past competition experience should not be an issue for most generators.

C. Baseline generator

All competition entrants were provided with a baseline level generator written in python, which provides a simple and effective method for generating levels within Science Birds. Participants were able to improve and enhance the baseline algorithm to create more advanced level generator software. It was also possible for participants to create their own level generators from scratch using any programming language, providing fresh ideas and insight into generating fun and exciting levels. For a more in-depth explanation of the baseline structure and level generation processes, as well as examples of its generated levels, please refer to the detailed competition instructions available from the AIBIRDS website [44]. Software for allowing Angry Birds agents developed for the AIBIRDS agent competition to play generated levels was also provided, along with a simple naive agent for solving levels. This naive agent always targets a randomly selected pig, but more sophisticated open-source agents are available from the AIBIRDS forum and can be integrated with the Science Birds program very easily.

D. Judging and scoring

During the competition, each generator created 10 levels from each of 10 different input files, giving 10 groups of 10 levels. A single level from each of these groups was then selected at random. The selected levels from each generator were then combined to form one single group, giving 10 levels for each generator, with the ordering of levels in each group randomised. This was done to ensure that no generator is unfairly punished by a particular input file. Generated levels were evaluated based on three different criteria. The first is how fun and enjoyable the level is to play and determines the overall competition ranking (main prize). The concept of "fun" was left deliberately vague to prevent biasing judges as much as possible. The second criterion is how creative the level design is (secondary prize). The third is how well balanced the difficulty of the level is (secondary prize). Several panels of judges evaluated each generator based on its levels, giving it a rating between zero (total failure, levels not generated or restrictions violated) and ten (perfectly designed levels) for each of the three level evaluation criterion. The judges also penalised any level generator that generated levels which were deemed too similar to each other (i.e. little variation between the levels generated). The final score for each level generator is the total rating across all judging panels.

IV. Competition Generators

A. MSG (v2.0)

The MSG (v2.0) generator was created by Matthew Stephenson from the Australian National University in Australia. It builds upon a previous level generator, originally described in [20], [18], which was the runner-up in the 2016 AIBIRDS level generation competition.

It generates levels consisting of a collection of independent structures, constructed using the twelve block shapes available. A probability table is used to determine the likelihood of a particular block shape being selected. Each block shape is given a probability of selection, with all probabilities summing to one. Structures generated using this algorithm are made up of rows, with each row initially consisting of a single block shape. Blocks within each row can also be randomly swapped with other block shapes that have the same height. These structures can have multiple peaks and feature a variety of placement methods for each row of blocks. Local stability requirements are enforced and more rows can be added until the structure reaches the desired size. Global structural stability is verified using quantitative analysis calculations, described in [45]. These structures are then distributed throughout the level, either on the ground (ground structures) or atop floating platforms (platform structures). The number of ground and platform structures, as well as their respective width and height limits, is determined randomly within a pre-defined range. Ground structures can also be placed on hills of varying heights, which are created using static terrain blocks.

Once these structures have been placed the level is populated with pigs, distributed on and within the created structures. Possible positions for pigs are identified and ranked based on a combination of structural protection and location dispersion. Pigs are then placed using this ranking until a desired number of pigs is reached. Possible TNT positions are also identified in the same manner, and are ranked based on a combination of projected damage and location dispersion. The material of each block within a structure is chosen randomly using one of several approaches. These include trajectory analysis (based on shot trajectories from the slingshot to a pig or TNT), clustering, row grouping, structure grouping and random selection. The generator then attempts to identify and protect critical weak points throughout the level. A weak point is defined as a block within a structure that can be hit directly by a player's shot (reachable) and that if removed would affect a large number of other blocks and/or pigs. Blocks that

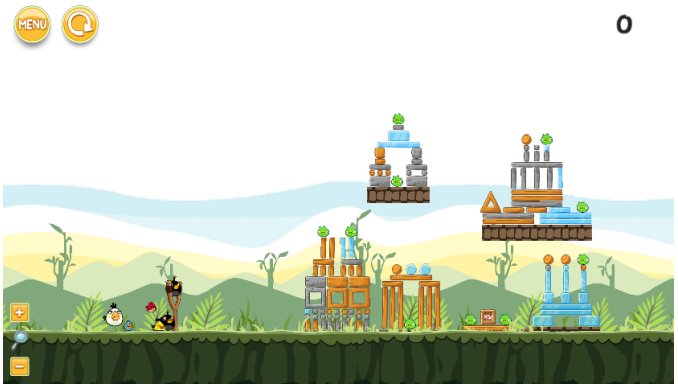

Fig. 5: Screenshot of a level from the MSG (v2.0) generator.

Fig. 6: Screenshot of a level from the Funny Quotes ft. Dominoes generator.

are identified as potential weak points can be protected by either placing additional protection structures next to it, adding additional support blocks within its structure row, or setting its material to stone.

The prevalence of certain block materials, as well as the degree to which pigs are reachable or protected, dictates the desired ratio and ordering of bird types. The number of birds is decided using a collection of intelligent agents from the previous AIBIRDS agent competitions, with the number of birds required by the best performing agent to solve the level selected. This ensures that every generated level is solvable, as an agent has already solved it beforehand. Further details on this generator can be found in [46]. An example level from this generator is shown in Figure 5.

B. Funny Quotes ft. Dominoes

The Funny Quotes ft. Dominoes generator was created by Yuxuan Jiang, Ryota Ishii, Tomohiro Harada and Ruck Thawonmas from Ritsumeikan University in Japan. It generates levels that consist of a quote, a formula, or a word combined with dominoes (a series of tall, thin blocks placed next to each other). It is based in part on a previous generator Funny Quotes [47] (the defending Champion from the 2016 AIBIRDS level generation competition), but to generate a combination of words and dominoes, Monte Carlo Tree Search (MCTS) [48] is used. In MCTS, a level is evaluated by the following criteria:

- Readability of generated characters forming a word.
- Variety of blocks.
- Usage of dominoes.
- Proximity of the proportion of the used area to the golden ratio (1.62).

The structure of each level type is as follows:

1) Quote levels: These levels consist of a popular quote, with each letter and punctuation made up using smaller blocks. There are 100 possible quotes, such as “We will be back”, “Need your Vote”, and “Relax bro!”. Pigs are placed on top of these quotes. with the number of pigs in each level selected randomly between the minimum and maximum. If there is insufficient space to place the minimum number of pigs required, then additional pigs are placed on a platform above the slingshot.

2) Formula-like levels: These levels consist of a simple mathematical formula using the mathematical symbols $+$, $-$, $\times$, $\div$ and $=$, as well as numbers. Similar to the quote levels, each of these symbols and numbers are made out of smaller block shapes. Pigs are used in these formulae to represent certain numbers (e.g. 8 pigs rather than the number 8). The number of pigs in each level is selected randomly between the minimum and maximum.

3) Word-plus-domino levels: The level’s area is divided into four sub-areas, each of which is then filled with either a word or dominoes. 75 different words are available, each having up to six characters, such as “Love”, “Happy”, and “Luck!”. Pigs are then placed on top of these words, as well as on top of domino blocks. If the number of initially assigned pigs is more than the maximum allowed, one pig is removed from the sub-area with the highest number of pigs. This is repeated until the number of pigs equals the maximum allowed. If the number of initially assigned pigs is less than the maximum allowed, additional pigs are placed on a platform above the slingshot.

The number of birds is always set to one more than the number of pigs. Further details on this generator can be found in [47]. An example level from this generator is shown in Figure 6.

C. MCTS ft. Blocks

The MCTS ft. Blocks generator was created by Yuxuan Jiang, Tomohiro Harada and Ruck Thawonmas from Ritsumeikan University in Japan. Inspired by the work of Graves [48], Monte Carlo Tree Search (MCTS) is used to place super-blocks (a stack of multiple blocks) and pig/TNT islands to create levels. The main difference between this generator and Graves’ is that this generator creates a super-block each time according to a set of rules which ensures stability, but Graves’ generator uses a collection of pre-determined stable structures. In addition, super-blocks and pig/TNT islands are randomly placed in the generated levels, while such objects are placed in a zig-zag fashion with Graves’ generator.

In order to create a super-block, a block shape and a material are selected randomly from the list of usable blocks. Selected blocks are then stacked up subject to some rules, sometimes

This article has been accepted for publication in a future issue of this journal, but has not been fully edited. Content may change prior to final publication. Citation information: DOI 10.1109/TG.2018.2854896, IEEE Transactions on Games

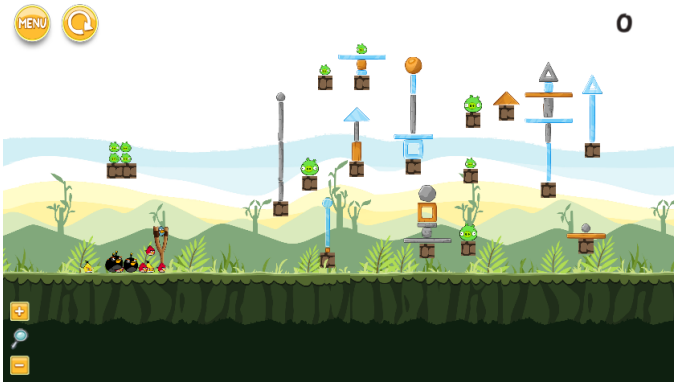

Fig. 7: Screenshot of a level from the MCTS ft. Blocks generator.

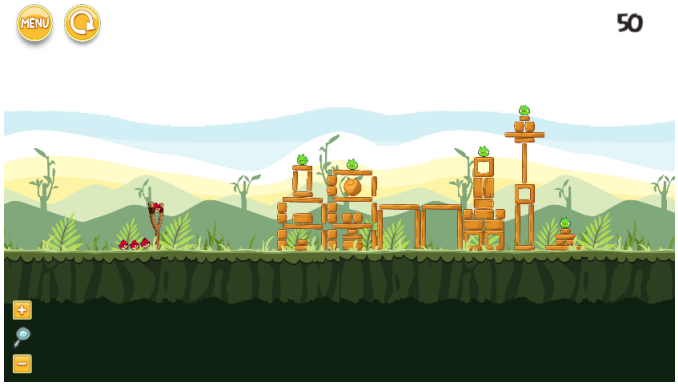

Fig. 8: Screenshot of a level from the Tanager generator.

with a pig on the top, to a predefined height. Next, a platform is added under a super-block. A platform is also added under a pig (or TNT) to form a pig (or TNT) island. MCTS is used to find the best combination of super-blocks, pig islands and TNT islands in the usable level area in terms of maximizing the following:

- Variety of blocks.
- Usage of pig or TNT islands.
- Proximity of the proportion of the used area to the golden ratio (1.62).

The number of pigs is always set to the maximum, and the number of birds is always set to one more than this. The level's area is divided into four sub-areas. If the number of initially assigned pigs is more than the maximum allowed, one pig is removed from the sub-area with the highest number of pigs. This is repeated until the number of pigs equals the maximum allowed. If the number of initially assigned pigs is less than the maximum allowed, additional pigs are placed on a platform above the slingshot. An example level from this generator is shown in Figure 7.

D. Tanager

The Tanager generator was created by Lucas Ferreira from the University of California in Santa Cruz, United States. It generates levels based on a genetic algorithm that is capable of producing stable and solvable levels. This genetic algorithm starts with an initial population composed of levels with randomly sampled stacks of blocks, pigs and birds. A fitness function evaluates stability, playability and structural characteristics of the levels via game simulations, where unplayable levels are penalized. A tournament method selects levels for reproduction based on their fitness values. New levels are created by crossover and mutation operators that try to keep the level stable. All new levels compose the population of the next generation, except the worst one that is replaced by the best level from the current generation (elitism). The generator stops after a given number of generations or if the fitness of the best level does not improve after multiple generations.

A level is encoded as a genotype composed of a number of birds and a list of stacks of blocks. The first element encodes the number of birds and all the others encode stacks. A block can be either elementary or composed, where elementary blocks are unitary pieces connected to form composed ones. Elementary or composed blocks can also be duplicated, and in this case they are added in the stack beside another one exactly like them. Each block is represented by a pair (i, b), where i is an integer representing the index of the block and b is a Boolean representing if that block is duplicated or not. The number of stacks can be different for each level and the stack sizes can change within a level.

A fitness function is used to measure if a level is fully stable and if the number of birds is enough to kill all the pigs. These two metrics are calculated using a game simulation with an intelligent agent. Stability is measured by the total velocity of the blocks during the first few seconds of the simulation. A level is only considered feasible if the number of pigs p_f at the end of the simulation is equal to zero. The fitness function is mathematically described by Equation 1.

$$fitness(x) = |\lfloor b_n * B \rfloor - B_u| + |\lfloor l_n * L \rfloor - L_b| + p_f + s \quad (1)$$

In this function, B is a constant that defines the maximum number of birds allowed in a level, B_u is the number of birds used during the simulation. The constant L defines the maximum number of blocks allowed in a level and L_b is the number of blocks in the beginning of the simulation. l_n and b_n represent the percentage of blocks in the level and the percentage of birds that is needed to kill all pigs respectively. s measures the stability of all blocks in the level.

The first term of the equation calculates the distance between the number of birds that should be used to kill all the pigs and the number that was actually used. The second term calculates the distance between the number of blocks desired in the level and the number of blocks that the level started with. If these terms are both zero, the level has all the desired characteristics. Further details on this generator can be found in [49]. An example level from this generator is shown in Figure 8.

E. Scrap Maps

The Scrap Maps generator was created by Ryota Ishii, Tomohiro Harada and Ruck Thawonmas from Ritsumeikan

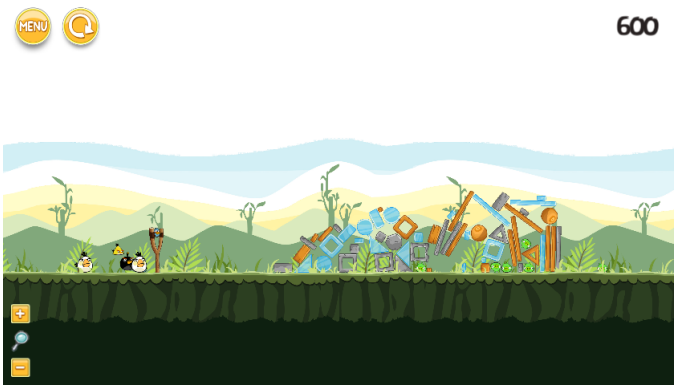

Fig. 9: Screenshot of a level from the Scrap Maps generator.

University in Japan. The Unity physics engine is used to simulate blocks falling from the sky, which are then saved to give the final level appearance.

A large collection of blocks and pigs are randomly selected from the list of usable objects each time a level is generated. These selected blocks and pigs are then temporarily placed at the top of the level (or the sky), each with a random position. The Science Birds game engine is then used to simulate these objects falling from the sky to ground. For this to work successfully, the game's settings are changed such that the blocks cannot be broken and the pigs cannot be killed. When all objects have fallen to the ground, the position and rotation of all blocks and pigs is saved. The number of pigs in a level is chosen randomly between the minimum and maximum values allowed. The number of birds is fixed to seven. The type of each bird is selected randomly. An example level from this generator is shown in Figure 9.

V. Results

As this competition was held jointly at CIG17 and IJCAI17, judging panels were used at both conferences. We had 7 panels of independent judges in Melbourne (IJCAI17) and 4 panels of independent judges in New York (CIG17). The level selection process for each generator was carried out separately beforehand. Judging panels were given the exact same levels from each generator, and were presented them in the exact same order. Each judging panel evaluated all 50 levels and reported the results back to the organisers in terms of scores between 0 and 10 for each of the five different generators, for each of the evaluation categories (Fun, Creativity and Difficulty). The identity of the generators and which levels belonged to which generator was kept a secret until after the judging scores were all aggregated and the ranking was finalised. It was vital to make sure this process was absolutely fair, as two of the five generators were from members of the organisation committee. Total and median scores for each generator are presented on the left side of Table I. A box plot for each generator based on the final scores from all judging panels is shown in Figure 10.

The Fun ratings for each generator determined the overall competition winner, with Creativity and Difficulty being additional secondary categories. The generator with the highest Fun rating was MSG (v2.0), with Funny Quotes ft. Dominoes second, and MCTS ft. Blocks third. Likewise for the Creativity category, MSG (v2.0) was the highest rated, followed by Funny Quotes ft. Dominoes and MCTS ft. Blocks in second and third place respectively. However, the Difficulty category was won by Funny Quotes ft. Dominoes, an improved version of last year's overall winner, with MCTS ft. Blocks second and MSG (v2.0) third.

Ten of the eleven judging panels rated MSG (v2.0) highest in terms of Fun, whilst one judging panel who liked the Scrap Maps generator the most rated MSG (v2.0) second best. Therefore, the MSG (v2.0) generator was the clear winner, with the Funny Quotes ft. Dominoes generator in second place, the MCTS ft. Blocks generator in third place, the Tanager generator in fourth place, and the Scrap Maps generator in fifth place.

A. Feature comparison

By comparing the properties of the generated levels against the judge's scores, we can attempt to identify whether there are certain level characteristics that may result in greater player enjoyment. There are four common measures that have been used previously to evaluate the expressivity of a level generator [50], [51]: frequency, linearity, density and leniency. Leniency is a measure of how difficult a level is to complete, and is often the hardest property to quantify and analyse. As the difficulty of the levels was already considered by judges during their evaluation, we chose not to investigate it further (agent performance could also have been used but was deemed less reliable than humans). Linearity represents the overall "profile" of a level, and is measured with the R^2 value from a linear regression calculation using the centre point of all game features present in a level's XML description. Density represents the compactness of a level and is calculated based on how much of a level's available space is taken up by objects. Frequency represents the number of certain level features that are present within a level. While there are many different and complex features for an Angry Birds level [52] that can be compared using frequency analysis, we decided to only focus on the basic features due to the limited size of our test set. The level features we considered were:

- Number of pigs.
- Number of blocks (for each material).
- Number of birds (for each type).
- Number of TNT boxes.

Due to the fact that some of the generated levels, particularly those created by the Scrap Maps generator, moved or responded to the physics of the game after initialisation, all values were recorded from levels after any initial movement had ceased. The average frequency of each feature across all 10 judged levels for each generator (normalised for each feature type separately) are presented in Figure 11. The average linearity and density values for each generator across these same levels are presented on the right side of Table I.

Spearman's rank correlation coefficients were calculated comparing the judging panel's rankings for each generator against the frequency of our selected level features, as well

This article has been accepted for publication in a future issue of this journal, but has not been fully edited. Content may change prior to final publication. Citation information: DOI 10.1109/TG.2018.2854896, IEEE Transactions on Games

TABLE I:
GENERATOR TOTAL (MEDIAN) SCORES AND EXPRESSIVITY ANALYSIS RESULTS

Generator	Fun	Creativity	Difficulty	Final	Linearity	Density
MSG (v2.0)	80.5 (8)	72.9 (7)	59.6 (5)	213.0 (20)	0.0516	28.86%
Funny Quotes ft. Dominoes	60.2 (5)	60.4 (6)	78.4 (8)	199.0 (19)	0.0603	37.57%
MCTS ft. Blocks	52.2 (5)	44.9 (4)	69.1 (6)	166.2 (16)	0.0216	15.66%
Tanager	44.5 (3)	37.7 (3)	46.7 (4)	128.9 (11)	0.0567	18.81%
Scrap Maps	28.0 (2)	28.0 (2)	21.0 (2)	77.0 (7)	0.0421	29.90%

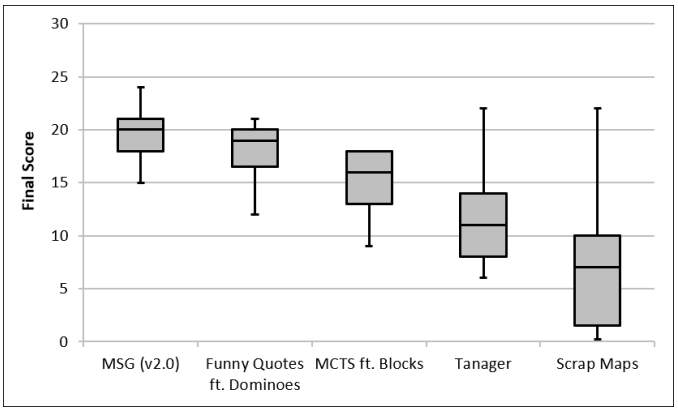

Fig. 10: Box plot for each generator based on the final scores from all judging panels.

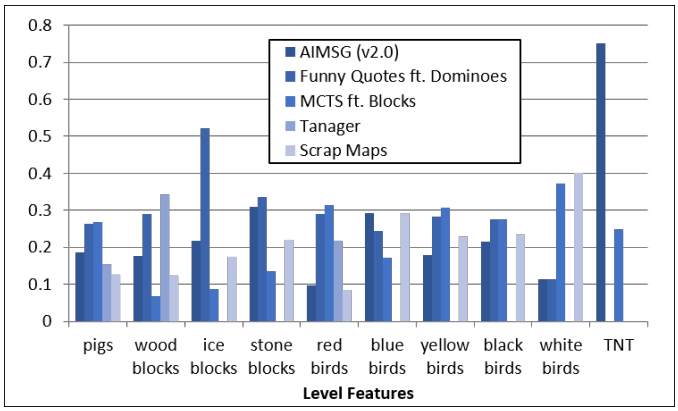

Fig. 11: Normalised average number of features (frequency measure) for each generator's judged levels.

as for both the density and linearity of the judged levels. Unfortunately, due to the limited number of levels that were judged for each generator, and the fact that each generator's levels vary so dramatically in design, it is unclear whether any identified correlations are merely due to random variation. For example, there was a high correlation ($\rho = 0.77$) between the number of TNT boxes in a level and the Fun score given by judges. However, this is likely due to the fact that only the MSG (v2.0) and MCTS ft. Blocks generators created levels that contained TNT, and the former far more than the latter. Another strong correlation ($\rho = 0.93$) was between the number of pigs and the Difficulty score for a level. This again could simply be due to the fact that the top two generators in this category, Funny Quotes ft. Dominoes and MCTS ft. Blocks, usually always give levels with the maximum number of pigs possible. You would also assume that the number of birds given to the player would impact the Difficulty score for a level, but this was not the case. It would seem that either Angry Birds is far too complex a game to have the enjoyment and challenge of its levels identified using only simple properties, or that not enough data is currently available for a reliable analysis. Bearing this in mind, general overall trends did seem to indicate that judges preferred levels with more pigs, blocks and TNT (i.e. more objects to interact with).

VI. DISCUSSION

A. Competition overview and limitations

Overall the competition went fairly smoothly, with all generators running successfully and levels displaying correctly. However, there are several changes or improvements that we feel could make it even better, particularly with how generators are compared. While there are defined evaluation categories, the values attributed to each level is currently decided solely by the judging panels. Deciding whether a generator creates levels that are deemed "too similar" is also left to the discretion of the judges. Ideally, it would be better to have a more reliable and unbiased means of scoring generators and the levels they create. However, this is a very difficult task, as there is currently no effective way of evaluating a level for complex concepts such as enjoyment. We can also see that the generator rankings for the Fun and Creativity categories are identical, suggesting that perhaps judges were slightly confused about the exact meaning of this latter category. Introducing additional criteria such as aesthetic appeal or level variety might help refine this evaluation process and allow different generators to focus on different level aspects. It may also be beneficial to ask judges why they preferred certain levels over others using either a questionnaire or ranking system. This would require more of the judge's time and additional human resources in interpreting the responses, but may help us develop better generators in the future.

B. Generator comparison

From the competition's results it seems very clear that the MSG (v2.0) generator was favoured by the majority of judging panels. While this generator competed in the previous year's competition it was only rated second out of three participating generators, losing to the previous version of the Funny Quotes generator. Its newfound success this year was likely due to a number of key improvements in terms of how levels were designed, as well as additional features such as

TNT placement, intelligent material/bird type selection, and using agents to guarantee solvability. The Funny Quotes ft. Dominoes generator was ranked second overall, but did win the Difficulty category. This was likely due to its sophisticated difficulty calculations, and the fact that the agents used by the MSG (v2.0) generator to playtest levels beforehand often resulted in giving the player more birds than was necessary. Scrap Maps was easily the least preferred generator which was likely caused by its unconventional level design, as well as the fact that many blocks and pigs would move or be destroyed after the level had initialised. The Tanager generator was also rated poorly which was probably due to the fact that it was not fully completed before the competition was run, and so could only produce structures made of wooden blocks and would always give the player only red birds.

C. Combining AIBIRDS competitions

There are several ways in which the two AIBIRDS competitions (agent and level generation) could be combined. As previously mentioned, the agents provided by the AIBIRDS agent competition can be used to evaluate and test the levels created by the generators. These agents could also be used to test a generated level against different playstyles, or to determine whether it is currently too hard or too easy based on the number of agents that can solve it and how long it takes them. Whilst the benefits that an agent can provide to a level generator should be clear, a level generator can also be used to improve the performance of AI agents. One major advantage of having level generators available is that it is now possible to create large numbers of training and test levels for developing improved AI agents for the AIBIRDS agent competition. In particular, agents based on deep reinforcement learning, a technique that has taken much of AI by storm and is very successful for many other games, would benefit greatly from a large number of available levels. Generating levels also provides a means of evaluating an agent beyond the original hand-designed levels that the game currently provides. In fact, several levels created by this year's submitted generators were converted and used in the most recent AIBIRDS agent competition. We also hope to be able to link both the AIBIRDS agent and level generation competitions in the future, perhaps with agents trying to beat generated levels and generators trying to create levels that are difficult for agents.

D. Generator improvements

While the depth and range of levels generated by the entries to this year's competition are very impressive, we believe that there are several ways they could be improved in the future. A typical improvement for PLG-based work is to create generators that can adapt to the skills and preferences of individual players [53], [54]. Generators would ideally be able to identify which "types" of levels a player is enjoying the most or which have the right amount of difficulty, and would then generate personalised levels that suit this player more. This would be challenging to perform in a competition style scenario, but would not be impossible. Another useful improvement would be to generate levels that require planning or creative reasoning to solve. A possible means of measuring this was postulated in [46], where it was suggested that levels which more advanced agents can solve but less skilled agents cannot, might be more likely to engage players as the solution would probably not be immediately apparent. Generators could also be more flexible in terms of the features their levels contain, allowing them to fulfil more specific designer requirements. Ideally an end goal for this work would be to develop generators that can create levels which are indistinguishable from "real" hand-designed levels. This is an extremely challenging task, especially for a game as complex and varied as Angry Birds, but would be an incredibly valuable scientific and industry resource if it could be achieved. Physical level generators, such as those submitted to this competition, might also have some additional real-world uses for automated design and construction.

VII. Conclusion

In this paper we have presented an overview of the second AIBIRDS level generation competition. The complexity and variety of levels created by this year's generators was significantly higher than in last year's competition. A greater number of judges were also used for evaluating levels from two separate locations, and the feedback received from the wider AI and video game research communities was extremely positive. We would also like to thank all members of our organising committee, competition entrants, judging panel volunteers, as well as all conference attendees at both CIG17 and IJCAI17 for their contribution and making this event possible. We hope to continue this competition next year and encourage all interested teams to participate in this exciting challenge.

References

[1] D. Silver, A. Huang, C. J. Maddison, A. Guez, L. Sifre, G. van den Driessche, J. Schrittwieser, I. Antonoglou, V. Panneershelvam, M. Lanctot, S. Dieleman, D. Grewe, J. Nham, N. Kalchbrenner, I. Sutskever, T. Lillicrap, M. Leach, K. Kavukcuoglu, T. Graepel, and D. Hassabis, "Mastering the game of Go with deep neural networks and tree search," *Nature*, vol. 529, no. 7587, pp. 484–489, 2016.

[2] J. Togelius, G. N. Yannakakis, K. O. Stanley, and C. Browne, "Search-based procedural content generation: A taxonomy and survey," *IEEE Transactions on Computational Intelligence and AI in Games*, vol. 3, no. 3, pp. 172–186, 2011.

[3] M. Hendrikx, S. Meijer, J. V. D. Velden, and A. Iosup, "Procedural content generation for games: A survey," *ACM Trans. Multimedia Comput. Commun. Appl.*, vol. 9, no. 1, pp. 1–22, 2013.

[4] S. Dahlskog and J. Togelius, "Patterns and procedural content generation: Revisiting Mario in world 1 level 1," in *Proceedings of the First Workshop on Design Patterns in Games*, 2012, pp. 1:1–1:8.

[5] J. Togelius, M. Preuss, N. Beume, S. Wessing, J. Hagelback, G. N. Yannakakis, and C. Grappiolo, "Controllable procedural map generation via multiobjective evolution," *Genetic Programming and Evolvable Machines*, vol. 14, no. 2, pp. 245–277, 2013.

[6] R. Lara-Cabrera, M. Nogueira-Collazo, C. Cotta, and A. J. Fernndez-Leiva, "Procedural content generation for real-time strategy games," *International Journal of Interactive Multimedia and Artificial Intelligence*, pp. 40–48, 2015.

[7] L. Ferreira, L. Pereira, and C. Toledo, "A multi-population genetic algorithm for procedural generation of levels for platform games," in *Proceedings of the Companion Publication of the 2014 Annual Conference on Genetic and Evolutionary Computation*, 2014, pp. 45–46.

[8] L. Cardamone, D. Loiacono, and P. L. Lanzi, "Interactive evolution for the procedural generation of tracks in a high-end racing game," in *Proceedings of the 13th Annual Conference on Genetic and Evolutionary Computation*, 2011, pp. 395–402.

[9] M. Cook and S. Colton, "Multi-faceted evolution of simple arcade games," in *Computational Intelligence and Games (CIG), 2011 IEEE Conference on*, 2011, Conference Proceedings, pp. 289–296.

[10] V. Valtchanov and J. A. Brown, "Evolving dungeon crawler levels with relative placement," in *The Fifth International C* Conference on Computer Science and Software Engineering*, 2012, pp. 27–35.

[11] Q. Xu, J. Tremblay, and C. Verbrugge, "Generative methods for guard and camera placement in stealth games," in *AAAI Conference on Artificial Intelligence and Interactive Digital Entertainment*, 2014, pp. 87–93.

[12] D. Stammer, H. Mannheim, T. Gnther, and M. Preuss, "Player-adaptive Spelunky level generation," in *2015 IEEE Conference on Computational Intelligence and Games (CIG)*, 2015, pp. 130–137.

[13] A. Khalifa, D. Perez-Liebana, S. M. Lucas, and J. Togelius, "General video game level generation," in *Proceedings of the Genetic and Evolutionary Computation Conference 2016*, ser. GECCO '16, 2016, pp. 253–259.

[14] X. Neufeld, S. Mostaghim, and D. Perez-Liebana, "Procedural level generation with answer set programming for general video game playing," in *2015 7th Computer Science and Electronic Engineering Conference (CEEC)*, 2015, pp. 207–212.

[15] N. Shaker, M. Shaker, and J. Togelius, "Evolving playable content for Cut the Rope through a simulation-based approach," in *Proceedings of the Ninth AAAI Conference on Artificial Intelligence and Interactive Digital Entertainment*, 2013, pp. 72–78.

[16] M. Shaker, N. Shaker, J. Togelius, and M. Abou-Zleikha, "A progressive approach to content generation," in *18th European Conference on the Applications of Evolutionary Computation, EvoApplications*, 2015, pp. 381–393.

[17] L. T. Pereira and C. F. M. Toledo, "Speeding up search-based algorithms for level generation in physics-based puzzle games," *International Journal on Artificial Intelligence Tools*, vol. 26, no. 05, 2017.

[18] M. Stephenson and J. Renz, "Procedural generation of levels for Angry Birds style physics games," in *Twelfth AAAI Conference on Artificial Intelligence and Interactive Digital Entertainment (AIIDE-16)*, 2016, pp. 225–231.

[19] L. T. Pereira, C. Toledo, L. N. Ferreira, and L. H. S. Lelis, "Learning to speed up evolutionary content generation in physics-based puzzle games," in *2016 IEEE 28th International Conference on Tools with Artificial Intelligence (ICTAI)*, 2016, pp. 901–907.

[20] M. Stephenson and J. Renz, "Procedural generation of complex stable structures for Angry Birds levels," in *2016 IEEE Conference on Computational Intelligence and Games (CIG)*, 2016, pp. 1–8.

[21] L. Ferreira and C. Toledo, "Generating levels for physics-based puzzle games with estimation of distribution algorithms," in *Proceedings of the 11th Conference on Advances in Computer Entertainment Technology*, 2014, pp. 25:1–25:6.

[22] M. Kaidan, T. Harada, C. Y. Chu, and R. Thawonmas, "Procedural generation of Angry Birds levels with adjustable difficulty," in *IEEE Congress on Evolutionary Computation (CEC)*, 2016, pp. 1311–1316.

[23] L. Ferreira and C. Toledo, "A search-based approach for generating Angry Birds levels," in *Computational Intelligence and Games (CIG), 2014 IEEE Conference on*, 2014, pp. 1–8.

[24] M. Shaker, M. H. Sarhan, O. A. Naameh, N. Shaker, and J. Togelius, "Automatic generation and analysis of physics-based puzzle games," in *Computational Intelligence in Games (CIG), 2013 IEEE Conference on*, 2013, pp. 1–8.

[25] AIBIRDS, "AIBIRDS agent benchmarks," https://aibirds.org/benchmarks.html, 2017, accessed: 2017-11-14.

[26] J. Renz, "AIBIRDS: The Angry Birds artificial intelligence competition," in *AAAI Conference on Artificial Intelligence*, 2015, pp. 4326–4327.

[27] J. Togelius, N. Shaker, S. Karakovskiy, and G. Yannakakis, "The Mario AI championship 2009-2012," *AI Magazine*, vol. 34, pp. 89–92, 2013.

[28] S. Karakovskiy and J. Togelius, "The Mario AI benchmark and competitions," *IEEE Transactions on Computational Intelligence and AI in Games*, vol. 4, no. 1, pp. 55–67, 2012.

[29] N. Shaker, J. Togelius, G. N. Yannakakis, B. Weber, T. Shimizu, T. Hashiyama, N. Sorenson, P. Pasquier, P. Mawhorter, G. Takahashi, G. Smith, and R. Baumgarten, "The 2010 Mario AI championship: Level generation track," *IEEE Transactions on Computational Intelligence and AI in Games*, vol. 3, no. 4, pp. 332–347, 2011.

[30] S. Ontan, G. Synnaeve, A. Uriarte, F. Richoux, D. Churchill, and M. Preuss, "A survey of real-time strategy game AI research and competition in StarCraft," *IEEE Transactions on Computational Intelligence and AI in Games*, vol. 5, no. 4, pp. 293–311, 2013.

[31] M. Kempka, M. Wydmuch, G. Runc, J. Toczek, and W. Jakowski, "ViZDoom: A Doom-based AI research platform for visual reinforcement learning," in *2016 IEEE Conference on Computational Intelligence and Games (CIG)*, 2016, pp. 1–8.

[32] R. Prada, P. Lopes, J. Catarino, J. Quitrio, and F. S. Melo, "The geometry friends game AI competition," in *2015 IEEE Conference on Computational Intelligence and Games (CIG)*, 2015, pp. 431–438.

[33] F. Lu, K. Yamamoto, L. H. Nomura, S. Mizuno, Y. Lee, and R. Thawonmas, "Fighting game artificial intelligence competition platform," in *2013 IEEE 2nd Global Conference on Consumer Electronics (GCCE)*, 2013, pp. 320–323.

[34] J. Renz, X. Ge, S. Gould, and P. Zhang, "The Angry Birds AI competition," *AI Magazine*, vol. 36, no. 2, pp. 85–87, 2015.

[35] D. Perez-Liebana, S. Samothrakis, J. Togelius, T. Schaul, S. M. Lucas, A. Coutoux, J. Lee, C. U. Lim, and T. Thompson, "The 2014 general video game playing competition," *IEEE Transactions on Computational Intelligence and AI in Games*, vol. 8, no. 3, pp. 229–243, 2016.

[36] R. D. Gaina, D. Prez-Libana, and S. M. Lucas, "General video game for 2 players: Framework and competition," in *2016 8th Computer Science and Electronic Engineering (CEEC)*, 2016, pp. 186–191.

[37] R. D. Gaina, A. Couetoux, D. Soemers, M. H. M. Winands, T. Vodopivec, F. Kirchgebner, J. Liu, S. M. Lucas, and D. Perez, "The 2016 two-player GVGAI competition," *IEEE Transactions on Computational Intelligence and AI in Games*, 2017.

[38] D. Perez-Liebana, S. Samothrakis, J. Togelius, S. Lucas, and T. Schaul, "General video game AI: Competition, challenges, and opportunities," in *30th AAAI Conference on Artificial Intelligence, AAAI 2016*. AAAI press, 2016, pp. 4335–4337.

[39] T. S. Nielsen, G. A. B. Barros, J. Togelius, and M. J. Nelson, "Towards generating arcade game rules with VGDL," in *2015 IEEE Conference on Computational Intelligence and Games (CIG)*, 2015, pp. 185–192.

[40] D. Perez-Liebana, M. Stephenson, R. D. Gaina, J. Renz, and S. M. Lucas, "Introducing real world physics and macro-actions to general video game ai," in *2017 IEEE Conference on Computational Intelligence and Games (CIG)*, 2017, pp. 248–255.

[41] "Angry Birds game," https://www.angrybirds.com/games/angry-birds/, accessed: 2017-11-14.

[42] J. Renz, X. Ge, R. Verma, and P. Zhang, "Angry Birds as a challenge for artificial intelligence," in *AAAI Conference on Artificial Intelligence*, 2016, pp. 4338–4339.

[43] L. N. Ferreira, "Science birds," https://github.com/lucasnfe/Science-Birds, 2017, accessed: 2017-12-12.

[44] AIBIRDS, "AIBIRDS homepage," https://aibirds.org, 2017, accessed: 2017-11-14.

[45] A. G. M. Blum and B. Neumann, "A stability test for configurations of blocks," Massachusetts Institute of Technology, Tech. Rep., 1970.

[46] M. Stephenson and J. Renz, "Generating varied, stable and solvable levels for Angry Birds style physics games," in *2017 IEEE Conference on Computational Intelligence and Games (CIG)*, 2017, pp. 288–295.

[47] Y. Jiang, T. Harada, and R. Thawonmas, "Procedural generation of Angry Birds fun levels using pattern-struct and preset-model," in *2017 IEEE Conference on Computational Intelligence and Games (CIG)*, 2017, pp. 154–161.

[48] M. Graves, "Procedural content generation of Angry Birds levels using monte carlo tree search," Master of Science in Engineering Thesis, The University of Texas at Austin, 2016.

[49] L. N. Ferreira and C. Toledo, "Tanager: A generator of feasible and engaging levels for Angry Birds," *IEEE Transactions on Computational Intelligence and AI in Games*, 2017.

[50] G. Smith and J. Whitehead, "Analyzing the expressive range of a level generator," in *Proceedings of the 2010 Workshop on Procedural Content Generation in Games*, 2010, pp. 4:1–4:7.

[51] B. Horn, S. Dahlskog, N. Shaker, G. Smith, and J. Togelius, "A comparative evaluation of procedural level generators in the Mario AI framework," in *Foundations of Digital Games 2014*, 2014, pp. 1–8.

[52] M. Stephenson and J. Renz, "Creating a hyper-agent for solving angry birds levels," in *AAAI Conference on Artificial Intelligence and Interactive Digital Entertainment*, 2017.

[53] G. N. Yannakakis and J. Togelius, "Experience-driven procedural content generation," *IEEE Transactions on Affective Computing*, vol. 2, no. 3, pp. 147–161, 2011.

[54] C. Pedersen, J. Togelius, and G. N. Yannakakis, "Modeling player experience for content creation," *IEEE Transactions on Computational Intelligence and AI in Games*, vol. 2, no. 1, pp. 54–67, 2010.