Deceptive Level Generation for Angry Birds

Chathura Gamage, Vimukthini Pinto, Jochen Renz
School of Computing
The Australian National University
Canberra, Australia
{chathura.gamage,u7069622,jochen.renz}@anu.edu.au

Matthew Stephenson
Department of Data Science and Knowledge Engineering
Maastricht University
Maastricht, the Netherlands
matthew.stephenson@maastrichtuniversity.nl

***Abstract*—The Angry Birds AI competition has been held over many years to encourage the development of AI agents that can play Angry Birds game levels better than human players. Many different agents with various approaches have been employed over the competition's lifetime to solve this task. Even though the performance of these agents has increased significantly over the past few years, they still show major drawbacks in playing deceptive levels. This is because most of the current agents try to identify the best next shot rather than planning an effective sequence of shots. In order to encourage advancements in such agents, we present an automated methodology to generate deceptive game levels for Angry Birds. Even though there are many existing content generators for Angry Birds, they do not focus on generating deceptive levels. In this paper, we propose a procedure to generate deceptive levels for six deception categories that can fool the state-of-the-art Angry Birds playing AI agents. Our results show that generated deceptive levels exhibit similar characteristics of human-created deceptive levels. Additionally, we define metrics to measure the stability, solvability, and degree of deception of the generated levels.**

***Index Terms*—deceptive games, level generation, game playing agents, Angry Birds**

I. Introduction

Procedural Level Generation (PLG) is a key area of investigation that focuses on the algorithmic generation of game levels in video game research [1], [2]. Game-playing agent development has benefited from PLG, since PLG can be used to generate a large number of training data in a short period of time [3]. Most of the current learning-based approaches embedded in AI agents require a huge amount of training data to be able to perform well [4].

Video games are frequently used by AI researchers as testbeds for their research [5]. Angry Birds is one such example, for both PLG [6] and agent development [7]. Angry Birds is a physics-based puzzle game which provides interesting challenges for AI agents when solving the game levels. PLG is also non-trivial in Angry Birds due to the physics constraints in the game. Any PLG algorithm should adhere to the physical constraints of the game, and positions of the game objects should be determined with greater precision to ensure the expected outcomes can be obtained when playing. Similar to an agent in a physical environment, Angry Birds playing agents have a continuous action space. Hence, generating levels that can only be solved by intended actions is very difficult, especially when the levels are complex and require appropriate reasoning capabilities to solve.

A deception for an AI agent can be seen as a characteristic of a task which "tricks" the agent into making poor actions by exploiting its biases or limitations [8]. A deceptive game level has a reward structure that can lead the agent away from the optimal strategy [9]. Previous work on Angry Birds presented six categories of deceptions for the existing Angry Birds playing agents [10]. Using handcrafted levels for those deception categories, they showed the vulnerabilities of the state-of-the-art Angry Birds playing agents. The drawbacks of existing agents in handling deceptions show that there is room for further advancements of agent capabilities. This urges the necessity of creating a sufficient number of challenging deceptive levels to satisfy the data requirement of current learning approaches.

In this paper, we present a methodology to generate deceptive levels in Angry Birds. We consider the six deception categories presented in [10] and define level templates that enable the automatic generation of varied levels for each of these deception categories. In addition to the level itself, a salient feature of this approach is that the solution for the level is also generated and can be utilized by the agents in the learning process. To evaluate our methodology, we define metrics to measure the stability, solvability, and deceptiveness of the generated levels. Moreover, to measure the characteristics of the generated levels compared to human-created levels, we evaluate AI agents' behaviour on both generated levels and handcrafted levels that require similar capabilities to solve.

II. Background and Related Work

A. Angry Birds

Angry Birds is a 2D physics simulation game where players shoot birds from a slingshot to kill pigs. The game levels consist of dynamic objects such as birds, blocks, pigs, and TNT explosives that can move when forces are applied, and static objects such as platforms that are not affected by forces. Dynamic objects have health points that get reduced on collisions and they get destroyed and disappear when health points reach zero. There are three types of materials that blocks are made of: wood, stone, and ice. There are physical entities (i.e., structures) that are made up of various arrangements of these blocks. There are also five types of birds, some of which have special powers and strengths against certain materials:

- Red bird: No special power.

- Blue bird: Splits into three birds when tapped, strong against ice.
- Yellow bird: Accelerates forward when tapped, strong against wood.
- Black bird: Explodes when tapped, strong against stone.
- White bird: Drops an explosive egg when tapped.

The player cannot change the order of the birds given in a level. A player's action is a combination of 2 elements, 1) the release point of the bird from the slingshot 2) the tap time to activate the power of the bird, if available. A level is solved once all pigs are killed, using the provided birds. As the original Angry Birds game is not open-sourced, we used a research clone of Angry Birds developed in Unity [11].

B. PLG in Angry Birds

Researchers have previously used several diverse approaches for procedurally generating game levels for Angry Birds [11]–[15]. The annual Angry Birds level generation competition [6] also promotes research into developing level generators for Angry Birds. This prior work mainly focuses on preserving the stability of the physical structures in the levels, adjusting the difficulty and enjoyability of the levels, and ensuring the levels are solvable. None of them focuses on generating deceptive game levels that require challenging reasoning and planning capabilities to solve. In this work, we focus on generating game levels that AI agents are not capable of solving through simple intuitive approaches.

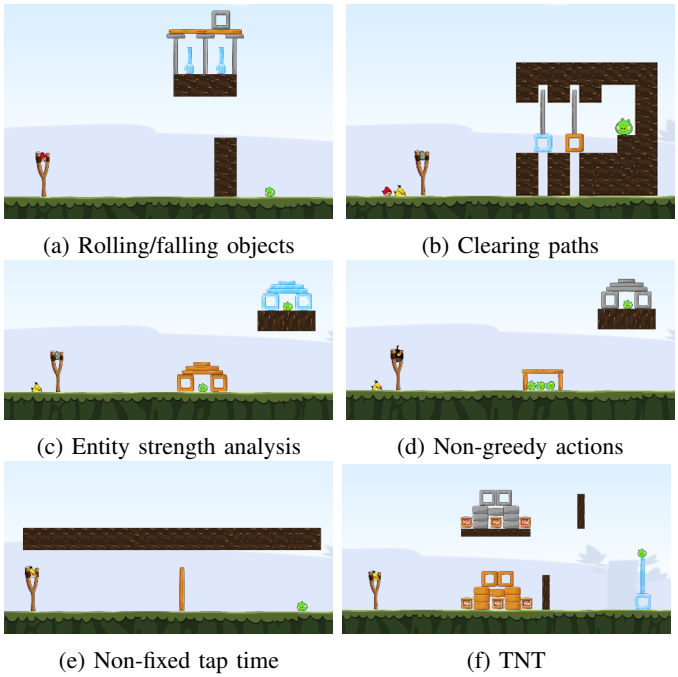
C. Deceptive Games

The concept of deceptive games was first presented in the study by Anderson et al. [8]. They create a suite of deceptive arcade-style games for the General Video Game AI (GVGAI) framework [16] and explore the effect of those games on game-playing agents. Building on that idea, an approach to generate deceptive levels for games in the GVGAI platform is discussed in [17] and a methodology to evaluate agents on deceptive levels is presented in [18].

In the context of complex physics simulation games, as already mentioned in Section I, the work on deceptive Angry Birds levels by Stephenson and Renz [10] suggests six different categories of deceptions that can trick or exploit the current state-of-the-art Angry Birds playing agents. While these deceptions do not affect all Angry Birds agents equally, no agent was able to successfully handle all deception categories. We consider these same six deception categories in our generation process. The six deception categories are described below and examples for those categories are shown in Fig. 1.

1) Rolling/falling objects: This deception uses the fact that objects of one entity can fall or roll on to another entity to create an impact. An agent needs to understand that an object can fall or roll and that object can be used to hit targets.

2) Clearing paths: This deception occurs when there are obstacles that need to be removed or destroyed before a target can be reached. To deal with this deception, an agent needs to understand that in order to reach a target, it should first clear the path to the target.



(a) Rolling/falling objects (b) Clearing paths

(c) Entity strength analysis (d) Non-greedy actions

(e) Non-fixed tap time (f) TNT

Fig. 1: Six example handcrafted levels for each deception category presented in [10]. The solutions for the levels are, (a) target the structure which collapses and falls on top of pig, (b) use first two birds to clear path for third, (c) use blue and yellow birds respectively on ice and wood structures, (d) first make the non-greedy shot that kills less pigs with black bird, (e) tap the yellow bird (accelerate) before hitting the block, and (f) target pig directly and ignore TNTs.

3) Entity strength analysis: This deception requires an agent to analyse the strengths of the entities in a level. The strength of an entity depends on the factors such as its material, shape, and size. An agent should be capable of determining the physical weaknesses/strengths of individual entities and interact accordingly to solve the level.

4) Non-greedy actions: In this deception, an action that appears to be less effective in the short term compared to another possible action, will gain higher advantages in the long term. The agent should look ahead and plan the actions using its knowledge of the environment rather than performing a greedy action that gives a higher short term reward.

5) Non-fixed tap time: The special powers of the birds are activated by tapping while the bird is in flight. In this deception, an agent needs to activate the special powers of the birds at non-fixed times, opposed to tapping at a predetermined fixed time.

6) TNT: The TNT explosives in the game explode when hit, causing damage or pushing nearby objects. In this deception, an agent needs to use TNTs to kill pigs or TNTs are used to distract the agent from the objective of killing pigs.

III. Terminology

In this paper, we use the term *environment* to refer to the space where an agent can sense and act, which is not a part

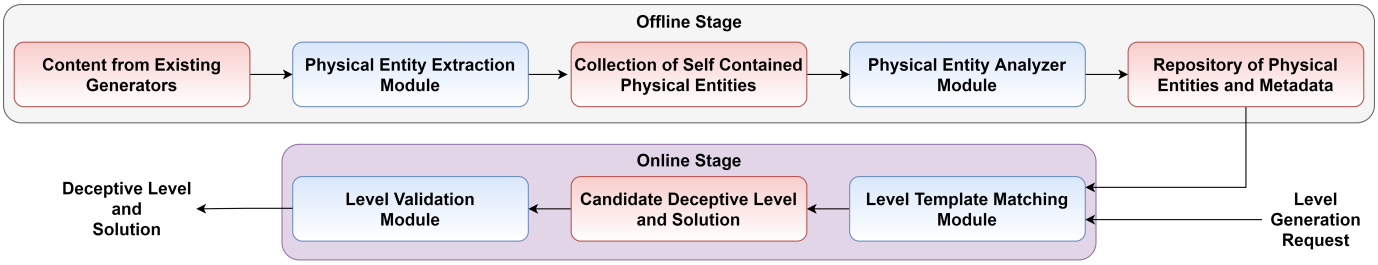


Fig. 2: Deceptive level generation methodology.

of the agent. A stand-alone, self-contained set of physically stable objects in the environment is termed as a *physical entity*. A *deception* is considered as a characteristic of a task that exploits the cognitive biases of an agent and causes it to make sub-optimal decisions [8]. A *strategy* is a sequence of an agent's actions (i.e., bird release points and tap times) that involves interacting with the environment. A *solution strategy* is a specific strategy that solves a given level when executed. Finally, a *tactic* is a plan that an agent tries to solve a level (e.g. shooting birds targeting at pigs). For a tactic, there can be multiple strategies which may or may not solve the level.

IV. Proposed Procedure

In this section, we present the proposed deceptive level generation procedure for Angry Birds. Fig. 2 shows the main components of the generation procedure. The four components shown in blue are the modules in the procedure and the four components in red are the inputs/outputs of those modules. The generation procedure does not involve creating physical entities; instead, it extracts physical entities created from existing content generators. This provides access to a wide range of content from various generation methods. Similarly to using existing content generators, customized handcrafted content can be used as well. The generation procedure consists of two stages: an offline stage and an online stage. The offline stage is shown inside the grey box (first row) and the online stage is shown inside the purple box (second row). The offline stage needs to be executed only once for an extracted set of physical entities. When a level generation request is received, the online stage is executed to generate a level using the offline stage's stored data. In subsequent subsections, we discuss the four components in our generation procedure: *Physical Entity Extraction Module*, *Physical Entity Analyzer Module*, *Level Template Matching Module*, and *Level Validation Module*.

A. Physical Entity Extraction Module

This module takes already generated game instances from Angry Birds content generators and extracts physical entities from those instances. The game instances can be complete game levels or parts of game levels with physical entities. We use a collection of existing Angry Birds level generators [6] to generate a set of game instances. Entities are copied from these instances using a qualitative reasoning process that iteratively extracts entities. Algorithm 1 shows the physical entity extraction process that can be applied to a game instance with multiple physical entities. When formulating the algorithm, we define the term supporter using the support graph similar to how authors have defined it in [19]. If the bottom horizontal edge of an object O_i is in contact with the top horizontal edge of another object O_j (i.e., O_i is resting on top of O_j) then the support graph contains an edge pointing from object O_i to O_j. In a given support graph, if there exists a path from object O_i to object O_j, then object O_j is considered as a supporter of the object O_i (O_j supports O_i). Objects placed on the ground do not have any supporter and hence has an empty support graph. Fig. 3 shows an example of how a physical entity is extracted iteratively using this algorithm.

Algorithm 1 Extract physical entities

Input: Game instance with physical entities
Output: Set of extracted physical entities

```
entitiesExtracted = {}
objectsRemaining = all objects in the instance
while objectsRemaining is not empty do
    topObject = topmost object of objectsRemaining
    entityExtracting = topObject and supporters of topObject
    while True do
        boundingBox = bounding box of entityExtracting
        entityExtracting = objects inside boundingBox
        if boundingBox size not increased then
            Break
    add entityExtracting to entitiesExtracted
    remove objects of entityExtracting from objectsRemaining
return entitiesExtracted
```

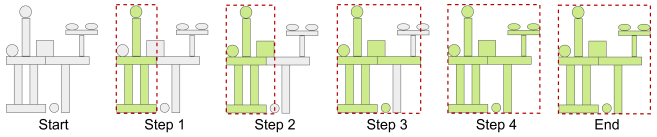


Fig. 3: An illustration of the extraction process for a single entity. The leftmost figure (start) shows the entity that needs to be extracted. Each step corresponds to an iteration of the algorithm and the rightmost figure (end) shows the extracted entity. Selected objects in each step are shown in green, the bounding box of selected objects are shown in red dotted lines.

B. Physical Entity Analyzer Module

After extracting physical entities, the *Physical Entity Analyzer Module* is used to analyze those entities individually by interacting with them and observing the outcome. In Angry Birds, interactions with the entities can only be done by shooting birds from the slingshot and tapping to activate

the bird's special power. To predict the result of physical interactions we can use either qualitative methods [20], [21], which are typically faster but less accurate, or simulation-based methods [22], which are typically more accurate but slower. Using qualitative methods in Angry Birds for physical predictions has proven to be less accurate and robust for large complex entities [14]. Therefore, we use a simulation-based method to perform interactions and record the outcomes.

We developed a portfolio agent which can play Angry Birds with four different tactics adopted from previous Angry Birds playing AI agents [23]. They are, 1) shooting birds (without special powers) targeting pigs in the entity, 2) shooting birds (without special powers) targeting at TNTs in the entity, 3) shooting birds (without special powers) targeting at reachable blocks in the entity, and 4) shooting birds with special powers (activated at different times) targeting at pigs. The above tactics also have different variants (i.e., strategies) based on the order of the targets (if multiple targets exist), the shooting angle, and the activation time of the bird's special power.

Each entity is tested with a predetermined fixed set of strategies. When interacting with the entities, the objects in the entity can be subjected to move, collide, damage, or destroy, due to the force applied on the entity. For each interaction (i.e., a bird shot), data is recorded in four steps:

- Entity data before the interaction (e.g. the number of objects in the entity, the entity's bounding box size).
- Interaction data (e.g. the magnitude and the location of the force applied on the entity due to the interaction).
- Dynamic data when the objects in the entity are moving as a result of the interaction (e.g. objects that have gone out of the entity's original bounding box, location and the velocity of the objects gone out).
- Entity data after the interaction when all the objects become stationary (e.g. the number of objects in the entity, the entity's bounding box size).

The data gathered through this analysis is attached to the entity as metadata and a repository of entities with metadata is maintained. This is the end of the offline stage of the generation process, the next steps are done online.

C. Level Template Matching Module

The online stage of the physical deceptive level generation process starts with a generation request coming to the *Level Template Matching Module*, specifying the desired deception category. This module generates a candidate level for that deception category using pre-defined level templates. For a deception, a level template contains constraints that entities should satisfy and rules for level generation. When generating a level, the level template considers possible interactions that an agent can perform and the outcomes of those interactions obtained from the entities' metadata. The entities that satisfy the constraints in the template are used to generate the level, following the generation rules.

When generating levels, *Level Template Matching Module* also creates the solution for the level using the interaction data (bird shots) in the metadata of the entities used to generate the level. The solution has the interaction data that solves the level (i.e., the solution strategy of the level). When designing the level templates, we attempt to make the deceptive level solvable only by using the generated strategy. We assume if an agent uses the solution strategy to solve the level, the agent understood the deception. Restricting the solvability only to the generated solution prevents the agents from solving the levels using other tactics without realizing the deception.

Algorithm 2 Rolling/falling objects

Input: Sender entity, Receiver entity
Output: A level with rolling/falling objects deception

```
if sender has suitable objects that can be rolled/fallen then
    if receiver has an OSS then
        while generation unsuccessful do
            get sender's rolling/falling object's trajectory
            get OSS of receiver
            generate level by matching sender and receiver
            verify reachability of targets
            if generation successful then
                break
            if maximum generation attempts reached then
                return none
        allocate birds to level
        generate solution strategy
        return generated level
return none
```

To facilitate the template design discussion, on top of the terminology discussed in Section III, we define the following terms. The *goal* of an Angry Birds player is, killing all the pigs in the level using the given number of birds. We refer to the term *solving an entity* as killing all the pigs within an entity. An entity is deemed solvable if all the pigs within the entity can be killed using the given birds. An *outperforming solution strategy (OSS)* refers to the solution strategy that uses the minimum number of birds to solve an entity compared to other solution strategies tested. If there are multiple solution strategies with the fewest number of birds needed to solve an entity, then there is no OSS for that entity. The level templates designed for the six deception categories are explained below.

1) Rolling/Falling Objects: A level with this deception is created using two types of entities, called senders and receivers. A sender gives an object out either by rolling or falling when an agent interacts with the entity. A receiver uses the impact of the sender's rolling/falling object towards achieving the goal (i.e., killing the pigs). The locations of the two entities are determined such that the rolling/falling object's impact can be used to replace the impact of an action in the receiver's OSS. The placement of the two entities should also ensure that the receiver's OSS no longer allows the agent to solve the level. This is done by placing the sender such that it blocks the bird trajectories of the actions in the receiver's OSS. The reachability of the target objects for the birds shot from the slingshot is verified to ensure the solvability of the level. Algorithm 2 shows the pseudocode of this level template.

2) Clearing Paths: A level with this deception is created using two types of entities, called obstacle and obstructed. The obstacle blocks the OSS of the obstructed entity. The agent should first substantially collapse the obstacle to clear a path

Algorithm 3 Clearing paths

Input: Obstacle Entity, Obstructed Entity
Output: A level with clearing paths deception

```
if obstacle entity can be cleared substantially then
    if obstructed entity has an OSS then
        while generation unsuccessful do
            generate level by matching obstacle and obstructed entities
            verify reachability of targets
            if generation successful then
                break
            if maximum generation attempts reached then
                return none
        allocate birds to level
        generate solution strategy
        return generated level
return none
```

Algorithm 4 Entity strength analysis

Input: Entity 1, Entity 2
Output: A level with entity strength analysis deception

```
for birdX in all bird types do
    for birdY in all bird types except BirdX do
        if entity1 solvable by birdX and entity2 solvable by birdY then
            if entity1 birdX usage < entity2 birdX usage and entity2 birdY
            usage < entity1 birdY usage then
                generate level with entity1, entity2, birdX, birdY
                break
        else if entity1 solvable by birdY and entity2 solvable by birdX then
            if entity1 birdY usage < entity2 birdY usage and entity2 birdX
            usage < entity1 birdX usage then
                generate level with entity1, entity2, birdX, birdY
                break
if generation successful then
    verify reachability of targets
    generate solution strategy
    return generated level
return none
```

to reach the obstructed entity. To ensure the agent interacts with the obstacle with the objective of collapsing to clear the path, entities that can only be collapsed with a specific strategy are selected as obstacles. Algorithm 3 shows the pseudocode of this level template.

3) Entity Strength Analysis: The strength of an entity is considered with respect to its capability of protecting pigs within it. In this level template, different bird types are used, as this affects the maximum damage that can be done to an entity. An agent needs to analyze the strength of the entities in the level and determine the correct bird type to use on each entity. Algorithm 4 shows the pseudocode of this level template. The number of birds that needs to solve an entity is termed as the *bird usage* in the pseudocode. The template creates levels by selecting two entities: one entity that is strong against one bird type (i.e., high bird usage) but weak against another (i.e., low bird usage) and another entity that shows opposite strengths and weaknesses to the same bird types.

4) Non-greedy Actions: In Angry Birds, greedy agents tend to kill the most pigs in a single action. In this level template, an entity that has more pigs and easy to solve (termed as a greedy entity) are combined with an entity that has a few pigs and hard to solve (termed as a non-greedy entity). The easiness/hardness of an entity is based on the type and the number of birds needed to solve it. To ensure the level can only

Algorithm 5 Non-greedy actions

Input: Greedy Entity, Non-greedy Entity
Output: A level with non-greedy actions deception

```
if 2 entities' 2 OSSs use different birds then
    if greedy entity is easier to solve than non-greedy entity then
        while generation unsuccessful do
            generate level with greedy entity and non-greedy entity
            allocate birds in the order for non-greedy and greedy entities
            verify reachability of targets
            if generation successful then
                break
            if maximum generation attempts reached then
                return none
        generate solution strategy
        return generated level
return none
```

Algorithm 6 Non-fixed tap time

Input: Entity 1, Entity 2
Output: A level with non-fixed tap time deception

```
for birdX in all bird types with special powers do
    if entity1 only solvable by birdX's specific tap time (Tx) then
        for birdY in all bird types with special powers do
            if entity2 only solvable by birdY's specific tap time (Ty) then
                if Tx ≠ Ty then
                    generate level with entity1, entity2, birdX, birdY
                    break
if generation successful then
    verify reachability of targets
    generate solution strategy
    return generated level
return none
```

be solved by doing the non-greedy action first, the two selected entities should have OSSs with different bird types. The order of birds in the level is selected such that the birds needed to solve the non-greedy entity are given first. As the agent cannot change the order of the birds, if it chooses to solve the greedy entity first, then it cannot solve the non-greedy entity. Algorithm 5 shows the pseudocode of this template.

5) Non-fixed Tap Times: The effect of a bird's special power on an entity depends on the time that an agent taps the bird during its flight. A bird can make more damage to the entity if it is tapped at the correct time. The correct tap time needs to be determined considering the distance to the target and the bird's special power. Entities that can only be solved by using a specific tap time of a bird are selected as feasible entities to generate levels with this deception. Two entities with two different tap times are used to generate a level, to ensure that agents with fixed tap times fail to solve the level. Algorithm 6 shows the pseudocode of this level template.

6) TNT: TNT deception is generated using two level templates. Using the first template, if there is an OSS for an entity and that strategy involves targeting TNTs, then a level is generated directly with this entity and the birds needed for its OSS. This form of the deception needs an agent to understand that TNTs explode when hit to cause greater damage, which can be used to achieve the goal. In the second template, TNTs are used to distract an agent. This template selects an entity with TNTs and without pigs (termed as a distracting entity) along with another entity with pigs (termed as a distracted

Algorithm 7 TNT

Input: Distracting Entity, Distracted Entity
Output: A level with TNT deception

```
if distracting entity has TNTs and no pigs then
    if distracted entity has an OSS then
        generate level with distracting entity and distracted entity
        allocate birds needed for the distracted entity's OSS
        verify reachability of targets
        generate solution strategy
        return generated level
return none
```

entity). The birds allocated to the level are the birds needed for the OSS of the distracted entity. Shooting a bird to explode the TNTs will not help solve the level, and will not leave enough birds to solve the distracted entity. The pseudocode for the second template is shown in Algorithm 7.

Once a deceptive level and its solution have been generated using one of these templates, they are passed to the *Level Validation Module*.

D. Level Validation Module

The last module in the generation process validates the physical stability and solvability of the generated levels. In Angry Birds, all the objects should remain stationary at the start of the level. The *Level Validation Module* first verifies the stability of the level using the Box2D physics engine. The velocities of objects in the level are observed after two seconds of simulating the level to determine the stability of the level. After confirming stability, the *Level Validation Module* then verifies the solvability of the level, using its generated solution strategy. The final outcome of this process is a stable deceptive level, along with its confirmed solution.

V. Results and Evaluations

In this section, we present the results and evaluation of the proposed deceptive level generation procedure. The majority of our implementation was coded using Python 3.7. The exceptions were the *Physical Entity Analyzer Module* and the *Level Validation Module* that required simulating the Box2D physics engine in Unity, coded in C#. The simulations were done by speeding up the physics engine by 50 times. The software was run on Windows 10 desktop PC with an i9-9900KS CPU and 64GB RAM. The average time (for 100 runs) consumed by each module is: *Physical Entity Extraction Module* took 21 milliseconds to extract an entity with 16 blocks, *Physical Entity Analyzer Module* took 85.71 seconds to analyze an entity using 10 strategies, *Level Template Matching Module* took 2.64 seconds to generate a level, and *Level Validation Module* took 9.01 seconds to validate a level. Therefore, the online stage of the generation process can generate a deceptive level in 11.65 seconds on average. Twelve levels generated for the six deception categories are shown in Fig. 4.

In the following sections we evaluate our generated levels using four metrics that measure the stability, solvability, deceptiveness, and similarity to human-created levels. Even though both stability and solvability are normally verified by the *Level Validation Module*, this module was disabled for these experiments in order to assess how often a generated level passed each of these validity checks.

TABLE I: Stability rate (R_i), solvability rate (S_i), the average deceptive score (D_i), and the average difference of the solve rates for human-created and generated levels (C_i) for the six deception categories.

Deception Category	R_i	S_i	D_i	$C_i(\%)$
Rolling/falling Objects	1.00	0.82	0.91	3.41
Clearing Paths	0.98	0.95	0.90	-1.75
Entity Strength Analysis	0.98	0.90	0.89	-6.67
Non-greedy Actions	1.00	0.81	0.95	7.56
Non-fixed Tap Time	0.97	0.87	0.97	0.00
TNT	0.96	0.92	0.95	-9.26

A. Stability

The stability rate (R_i) of a deception category i is calculated as the percentage of levels which were stationary within the Box2D physics engine at the start of the game levels. A set of 100 levels for each deception category was used (600 levels in total). The results are presented in the second column of Table I. This shows that our generation procedure is capable of generating levels with a stability rate of almost 100%.

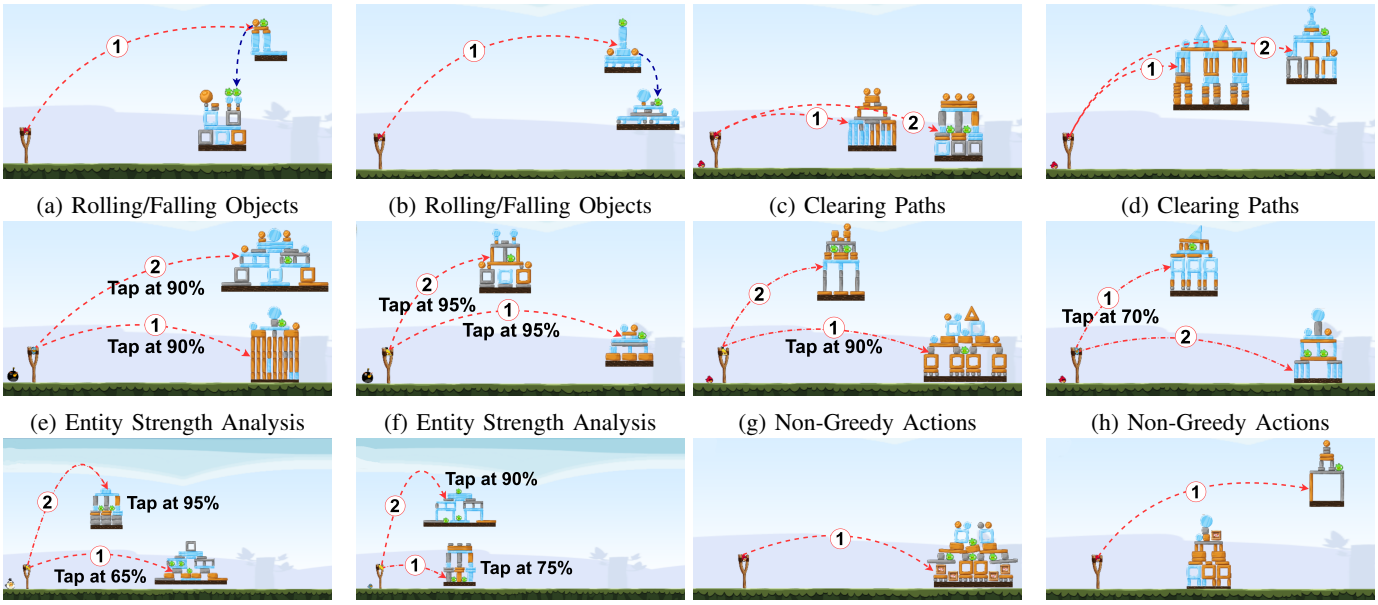
B. Solvability

The physical outcomes of the interactions considered in the generation process might differ from the outcomes when actually playing the level, typically due to differences in the location of entities when generating levels versus their original location during entity analysis, as discussed in Section IV-B. Therefore, a generated level might not always be solvable using the intended solution strategy (i.e., the generated solution). Hence, we evaluate the solvability of the generated levels to measure the competence of the generator in creating levels that can be solved with the intended solution strategy.

The solvability rate (S_i) of a deception category i is calculated as the percentage of levels which could be solved using its provided solution strategy. A set of 50 levels for each deception category was used (300 levels in total). The results are presented in the third column of Table I. This shows that our generation procedure is capable of generating levels with a solvability rate higher than 81% for the six deceptions.

C. Deceptiveness

To measure the degree of deception of the levels in a deception category we define a metric, average deceptive score. A level is considered less deceptive if it can be solved by multiple strategies without understanding the deception. The solution strategy created when generating a level is the strategy that an agent would follow if the deception is understood. To calculate the deceptive score of a level, the level is tested with different known strategies and the number of strategies that solved the level, excluding the solution strategy, is used. Equation 1 shows the average deceptive score (D_i) calculated for a deception category i by averaging the deceptive scores of all the levels tested for the deception. N is the total number



(a) Rolling/Falling Objects (b) Rolling/Falling Objects (c) Clearing Paths (d) Clearing Paths

(e) Entity Strength Analysis (f) Entity Strength Analysis (g) Non-Greedy Actions (h) Non-Greedy Actions

(i) Non-fixed Tap Time (j) Non-fixed Tap Time (k) TNT (Solution) (l) TNT (Distraction)

Fig. 4: Examples of generated levels for the six deception categories. The solution strategy for each level is illustrated using the red dotted arrows. The arrow path shows the trajectory of the bird and the arrowhead is pointed to the target. Numbers show the order of bird shots. "Tap at $x\%$" means the bird is needed to tap at the length of $x\%$ in its trajectory path, which is only available for birds with special powers. The blue lines in (a) and (b) are the trajectories of falling objects.

of levels tested, T_n is the total number of strategies used to test the n^{th} level, and P_{nt} is 1 if the n^{th} level is solved by t^{th} strategy or 0 otherwise. $D_i \in [0, 1]$ and a higher score means a higher deceptiveness.

$$D_i = \frac{1}{N}\sum_{n=1}^{N}\frac{1}{T_n}\sum_{t=1}^{T_n}(1 - P_{nt}) \quad (1)$$

For this experiment, we developed and ran a portfolio agent with 10 variants of tactics mentioned in Section IV-B. A set of 50 levels for each deception category was used (300 levels in total). The fourth column of Table I shows the average deceptive score calculated for the six deceptions. The results show that our method can generate deceptive levels with an average deceptive score over 0.89 for the six deceptions.

D. Comparison with Human-created Levels

From human capabilities, we are very adept at creating deceptive levels. Therefore, we evaluate generated levels against human-created levels by examining whether the generated levels exhibit similar characteristics to the human-created levels for AI agents. We compare the solve rates of agents for human-created and generated levels that belong to the same deception category. The average level solve rate difference between the human-created and generated levels (C_i) for a deception category i is shown in Equation 2. A is the number of agents tested, M is the total number of human-created levels, and N is the total number of generated levels. P_{am} is 1 if the a^{th} agent solved m^{th} human-created level or 0 otherwise, similarly P_{an} is 1 if the a^{th} agent solved n^{th} generated level or 0 otherwise. A positive value for C_i indicates that the generated levels are more difficult to solve than the human-created levels on average for AI agents and vice versa.

$$C_i = \frac{1}{A}\sum_{a=1}^{A}\left(\frac{1}{M}\sum_{m=1}^{M}P_{am} - \frac{1}{N}\sum_{n=1}^{N}P_{an}\right) \quad (2)$$

For this evaluation, we used 30 handcrafted Angry Birds levels from the previous work [10] representing the six deception categories (five levels on average per category). We generated 300 levels from our generator (50 levels per category). Three state-of-the-art Angry Birds agents Datalab, Eagle's Wing, and Bambirds from previous AIBirds competitions [23] were used for the experiment. The solve rates of the three agents for the six deceptions for human-created and generated levels are shown in Fig. 5. This figure depicts that the solve rates of the generated levels are correlated to the solve rates of the human-created levels. This portrays that agents show similar behaviours when playing both human-created and generated levels. The C_i values (percentage) calculated for the six deceptions from the results of the three agents are in the fifth column of Table I. The levels generated for two deception categories were more difficult to solve than the human-created levels while three deception categories were easier. The human-created and generated levels of non-fixed tap time deception were equally difficult for the agents.

VI. Conclusions and Future Work

In this paper, we have presented a methodology to generate deceptive game levels for Angry Birds. The proposed methodology can generate levels for six deception categories that the state-of-the-art Angry Birds playing agents are vulnerable

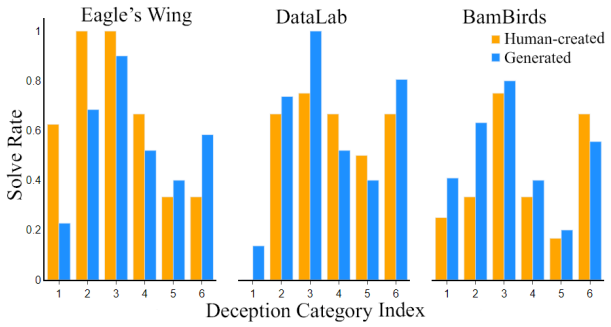


Fig. 5: Level solve rates of three agents for human-created and generated levels. The deception indexes on the x-axis are in the order: rolling/falling objects (1), clearing paths (2), entity strength analysis (3), non-greedy actions (4), non-fixed tap time (5), and TNT (6).

to. Even though the idea of handcrafting deceptive levels for Angry Birds has been previously investigated, this is the first attempt at generating deceptive levels for a complex physics-based game like Angry Birds. In addition, our approach generates the solution for the levels, which is a feature that is not available in any of the existing Angry Birds level generators, and which could be beneficial for learning agents in their training process. The levels generated from the proposed procedure were evaluated using four metrics: stability, solvability, deceptiveness, and a comparison with human-created levels. The results of these metrics demonstrate that the generation process can competently create deceptive levels for the six deceptions considered.

This work aims to enable the development of advanced Angry Birds playing agents that can perform well under deceptions by providing sufficient training/testing data. To deal with these deceptions, the AI techniques of the current agents can be improved to expand their reasoning, planning, and generalizations skills. Future level generation research can involve generating more complex levels by combining multiple deception categories. Additional deception categories may also be proposed, if more flaws within the state-of-the-art agents can be identified. The deception categories we considered, rolling/falling objects, clearing paths, entity strength analysis, and non-greedy actions can also be seen in a real physical environment. Therefore, this research can serve as a base for generating deceptive tasks in a real physical environment.

Acknowledgments

This research was sponsored by the Defense Advanced Research Projects Agency (DARPA) and the Army Research Office (ARO) and was accomplished under Cooperative Agreement Number W911NF-20-2-0002. The views and conclusions contained in this document are those of the authors and should not be interpreted as representing the official policies, either expressed or implied, of the DARPA or ARO, or the U.S. Government. The U.S. Government is authorized to reproduce and distribute reprints for Government purposes notwithstanding any copyright notation herein.

References

[1] J. Freiknecht and W. Effelsberg, "A survey on the procedural generation of virtual worlds," *Multimodal Technologies and Interaction*, vol. 1, no. 4, 2017.

[2] M. Hendrikx, S. Meijer, J. Van Der Velden, and A. Iosup, "Procedural content generation for games: A survey," *ACM Trans. Multimedia Comput. Commun. Appl.*, vol. 9, Feb. 2013.

[3] J. Togelius, "AI researchers, video games are your friends!," *CoRR*, vol. abs/1612.01608, 2016.

[4] X. Zhu, C. Vondrick, C. C. Fowlkes, and D. Ramanan, "Do we need more training data?," *International Journal of Computer Vision*, vol. 119, no. 1, pp. 76–92, 2016.

[5] N. Justesen, M. S. Debus, and S. Risi, "When are we done with games?," in *2019 IEEE Conference on Games (CoG)*, pp. 1–8, IEEE, 2019.

[6] M. Stephenson, J. Renz, X. Ge, L. Ferreira, J. Togelius, and P. Zhang, "The 2017 AIBIRDS level generation competition," *IEEE Transactions on Games*, vol. 11, no. 3, pp. 275–284, 2019.

[7] J. Renz, "AIBIRDS: The angry birds artificial intelligence competition," in *Proceedings of the AAAI Conference on Artificial Intelligence*, vol. 29, 2015.

[8] D. Anderson, M. Stephenson, J. Togelius, C. Salge, J. Levine, and J. Renz, "Deceptive games," in *International Conference on the Applications of Evolutionary Computation*, pp. 376–391, Springer, 2018.

[9] P. Bontrager, A. Khalifa, D. Anderson, M. Stephenson, C. Salge, and J. Togelius, ""superstition" in the network: Deep reinforcement learning plays deceptive games," in *Proceedings of the AAAI Conference on Artificial Intelligence and Interactive Digital Entertainment*, vol. 15, pp. 10–16, 2019.

[10] M. Stephenson and J. Renz, "Deceptive angry birds: Towards smarter game-playing agents," in *Proceedings of the 13th International Conference on the Foundations of Digital Games*, FDG '18, (New York, NY, USA), Association for Computing Machinery, 2018.

[11] L. N. Ferreira, "A search-based approach for generating angry birds levels," *IEEE Conference on Computatonal Intelligence and Games, CIG*, 08 2014.

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

[13] L. Calle, J. J. Merelo, A. Mora-García, and J.-M. García-Valdez, "Free form evolution for angry birds level generation," in *International Conference on the Applications of Evolutionary Computation (Part of EvoStar)*, pp. 125–140, Springer, 2019.

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

[15] F. Abdullah, P. Paliyawan, R. Thawonmas, T. Harada, and F. A. Bachtiar, "An angry birds level generator with rube goldberg machine mechanisms," in *2019 IEEE Conference on Games (CoG)*, pp. 1–8, IEEE, 2019.

[16] D. Perez-Liebana, S. Samothrakis, J. Togelius, T. Schaul, and S. Lucas, "General video game AI: Competition, challenges and opportunities," in *Proceedings of the AAAI Conference on Artificial Intelligence*, 2016.

[17] A. Zafar, H. Mujtaba, M. O. Beg, and S. Ali, "Deceptive level generator.," in *AIIDE Workshops*, 2018.

[18] N. U. Sabah and A. Zafar, "Evaluating the performance of agents for deceptive levels," in *Manchester journal of artificial intelligence and applied sciences*, vol. 2, pp. 1–7, 2021.

[19] M. Stephenson, J. Renz, X. Ge, and P. Zhang, "Generating stable building block structures from sketches," *IEEE Transactions on Games*, vol. 13, pp. 1–10, 2021.

[20] P. A. Walega, M. Zawidzki, and T. Lechowski, "Qualitative physics in angry birds," *IEEE Transactions on Computational Intelligence and AI in Games*, vol. 8, no. 2, pp. 152–165, 2016.

[21] P. Zhang and J. Renz, "Qualitative spatial representation and reasoning in angry birds: The extended rectangle algebra," in *Proceedings of the Fourteenth International Conference on Principles of Knowledge Representation and Reasoning*, KR'14, p. 378–387, AAAI Press, 2014.

[22] P. W. Battaglia, J. B. Hamrick, and J. B. Tenenbaum, "Simulation as an engine of physical scene understanding," *Proceedings of the National Academy of Sciences*, vol. 110, no. 45, pp. 18327–18332, 2013.

[23] M. Stephenson, J. Renz, X. Ge, and P. Zhang, "The 2017 AIBIRDS competition," *arXiv preprint arXiv:1803.05156*, 2018.