Proceedings of the Twenty-Ninth International Joint Conference on Artificial Intelligence (IJCAI-20)
Journal Track

The Computational Complexity of Angry Birds (Extended Abstract)*

Matthew Stephenson[1], **Jochen Renz**[2] and **Xiaoyu Ge**[2]
[1]Department of Data Science and Knowledge Engineering, Maastricht University, the Netherlands
[2]Research School of Computer Science, Australian National University, Canberra, Australia
matthew.stephenson@maastrichtuniversity.nl, {jochen.renz xiaoyu.ge}@anu.edu.au

Abstract

In this paper we present several proofs for the computational complexity of the physics-based video game Angry Birds. We are able to demonstrate that solving levels for different versions of Angry Birds is either NP-hard, PSPACE-hard, PSPACE-complete or EXPTIME-hard, depending on the maximum number of birds available and whether the game engine is deterministic or stochastic. We believe that this is the first time that a single-player video game has been proven EXPTIME-hard.

1 Introduction

In this paper, we analyse the complexity of playing different variants of the video game Angry Birds, which is a physics-based puzzle game with a semi-realistic environment, see Figure 1. The objective of each level in this game is to hit a number of pre-defined targets (pigs) with a certain number of shots (birds) taken from a fixed location (slingshot), often utilising or avoiding blocks and other game elements to achieve this. This game has been heavily researched by the AI community over the past decade, due to the complex planning and physical reasoning required to solve its levels, similar to that of many real-world tasks [Renz *et al.*, 2019].

While there has been an extensive amount of research into the computational complexity of video games over the past decade [Demaine *et al.*, 2016a; Demaine *et al.*, 2016b; Forišek, 2010; Viglietta, 2014b; Gualà *et al.*, 2014; Bosboom *et al.*, 2018], Angry Birds presents a very different problem from those typically studied. In physics-based environments the attributes and parameters of various objects are often imprecise or unknown, making it very difficult to accurately predict the outcome of any action taken. Angry Birds also differs from many previously investigated games in terms of its control scheme, as the player always makes their shots from the same location within each level and can only vary the speed and angle at which each bird travels from it. This heavily reduces the amount of control that the player has over the bird's movement, with the game's physics engine being used to determine the outcome of shots after they are made.

*This paper is an extended abstract of an article in Artificial Intelligence [Stephenson et al., 2020].

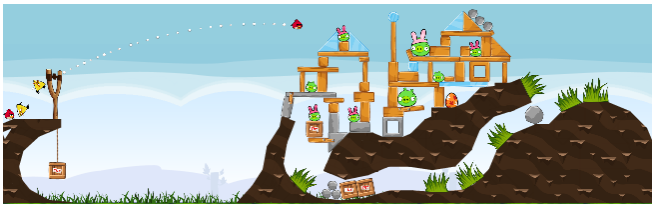


Figure 1: Screenshot of a level for the Angry Birds video game.

2 Angry Birds Game Definition

The decision problem we are considering can be formalised as: Given an Angry Birds level description, is there a strategy that always results in all pigs being killed? The complexity of this problem depends on two key factors:

Number of Birds. The first factor is whether the maximum number of birds that the player can have is polynomial or exponential relative to the size of the level description.

Probabilistic Model. The second factor is whether the physics engine used by the game is deterministic or stochastic. For Angry Birds, the source of this stochasticity comes from a random amount of noise that is included when collisions occur within the game's physics-engine, causing the object(s) involved to move slightly differently each time.

Table 1 shows how altering these factors within the Angry Birds game affects its complexity. For each of our subsequent complexity proofs, we will use the appropriate version of Angry Birds as defined by this table.

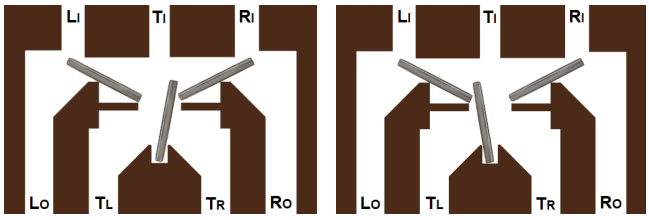
3 Gates

Before presenting our complexity proofs we first define three different "gates", that help dictate the outcomes of shots taken by the player. These gates are constructed using Angry Birds game entities and are designed to emulate the functionality of logic gates. By arranging multiple gates within a level, we can obtain equivalent versions of known complexity problems within the Angry Birds game environment.

Selector Gate. The Selector gate implementation for Angry Birds is shown in Figure 2. The Selector gate can exist in one of two states, "select-left" or "select-right", and essentially mimics the behaviour of a 2-output demultiplexer. A bird which enters a Selector gate at L_I or R_I, will exit at L_O or

Name	No. Birds	Model	Complexity
ABPD	Polynomial	Deterministic	NP-hard
ABED	Exponential	Deterministic	PSPACE-complete
ABPS	Polynomial	Stochastic	PSPACE-hard
ABES	Exponential	Stochastic	EXPTIME-hard

Table 1: Complexity results summary.



(a) (b)

Figure 2: Models of Selector gate (a) in the "select-left" position and (b) in the "select-right" position.

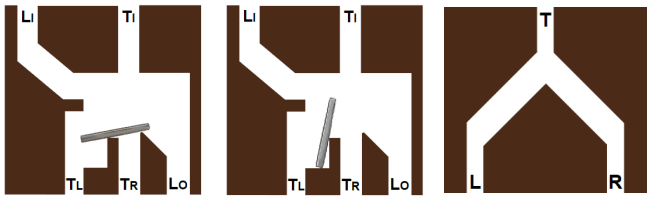
R_O, and set the gate to the select-left or select-right position respectively. A bird which enters a Selector gate at T_I will exit at either T_L or T_R based on the gate's current state.

Automatically Unsetting Transfer Gate. The automatically unsetting transfer (AUT) gate implementation for Angry Birds is shown in Figure 3 (a/b). The AUT gate can exist in one of two states, "select-left" or "select-right". A bird which enters an AUT gate at T_I will exit at T_L and set the gate to the select-right position, if and only if the gate is in the select-left position, otherwise the bird will exit at T_R and not change the gate's position. A bird which enters an AUT gate at L_I will exit at L_O and set the gate to the select-left position.

Random Gate. The Random gate implementation for Angry Birds is shown in Figure 3 (c). The Random gate can only be used in variants of Angry Birds with a stochastic game engine, and essentially mimics the behaviour of a random binary splitter. A bird which enters a Random gate at T has a non-zero probability for exiting at both L and R.

4 PSPACE-Completeness of ABED

For our proof of PSPACE-completeness, we will reduce from the PSPACE-complete problem TQBF, which consists of determining if a given quantified 3-CNF Boolean formula is "true". To demonstrate that Angry Birds is PSPACE-hard, we present a framework for converting any given quantified Boolean formula into an Angry Birds level, which can only be solved if the quantified Boolean formula is true. Our proof of PSPACE-hardness is based on a heavily modified version of the general framework described in [Aloupis *et al.*, 2014; Viglietta, 2014a; Viglietta, 2014b]. This framework uses a systematic procedure to verify if a quantified Boolean formula is true, provided that several "Gadgets" can be constructed within the game's environment. A gadget is a collection of several interconnected gates which serves a distinct purpose within the framework. The same gadget can often occur many times within a single framework (i.e. frameworks are made up of gadgets, and gadgets are made up of gates).



(a) (b) (c)

Figure 3: Models of AUT gate (a) in the "select-left" position and (b) in the "select-right" position. Model of Random gate (c).

The framework for this proof uses an Existential Quantifier (EQ) gadget, a Universal Quantifier (UQ) gadget and a Clause gadget, which can be used to mimic the properties of each variable (be it existentially or universally quantified) and clause in a quantified 3-CNF Boolean formula.

While we will still be using this same TQBF verification process for our proposed Angry Birds proof, the overall design of the framework for applying this procedure will be significantly different from those of previous game examples. This is mostly due to the fact that Angry Birds does not have a single controllable "Avatar", and thus has no easy way of achieving a sense of "player traversal". This means that cyclic loops within the framework are not possible (all birds must always fall downwards). Figure 4 shows our modified TQBF framework design for Angry Birds, using the quantified Boolean formula $\exists x \forall y \exists z \forall w((x \vee y \vee w) \wedge (y \vee \neg z \vee \neg w) \wedge (\neg x \vee \neg y \vee z))$ as an example. One particular point of difference is that the UQ gadget has been split into two subgadgets, referred to as the UQ-F and UQ-T gadgets. There is also a Finish gadget, which the player must be able to "pass through" in order to solve the level and will be permanently blocked if the player performs any "illegal" actions.

Rather than a complete formal proof we will instead attempt to give a general overview of each gadget's design and function within Angry Birds, and how they can be combined to create an equivalent representation of any quantified 3-CNF Boolean formula.

Definition 1. (enabled, disabled, current, next, next adjacent, next UQ-F, previous, first, last): *Each gadget has two possible states,* "enabled" *and* "disabled". *The* "current" *gadget (Q_i) is the (vertically) lowest enabled gadget in our general framework diagram (GFD, see Figure 4). The* "next" *gadget (Q_{i+1}) for the current gadget is indicated by the arrows in our GFD, which represent the scope of each quantifier. For each UQ-F gadget there are two possible next gadgets, the next gadget for the UQ-T gadget associated with its variable (horizontal output arrow) referred to as the* "next adjacent" *gadget, and the UQ-F gadget directly below it (vertical output arrow) referred to as the* "next UQ-F" *gadget. The* "previous" *gadget (Q_{i-1}) refers to the most recent current gadget. We also define the terms* "first" *gadget and* "last" *gadget with respect to the vertical position of specific gadget types in our GFD. The highest of a particular gadget type is the first gadget of that type, whilst the lowest is the last gadget.*

EQ Gadget. If an EQ gadget is enabled then the player can use it to set the value of its associated variable to either posi-

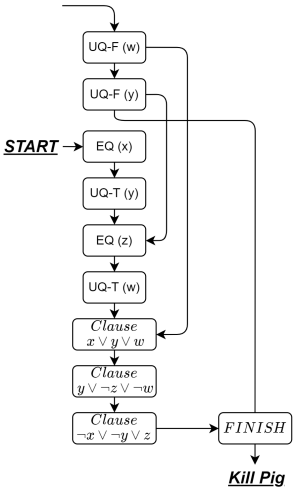


Figure 4: General framework diagram (GFD) for ABED.

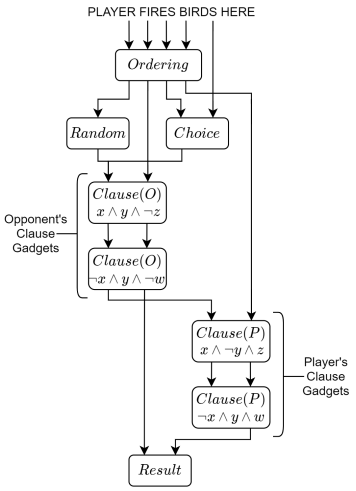


Figure 5: General framework diagram for ABES.

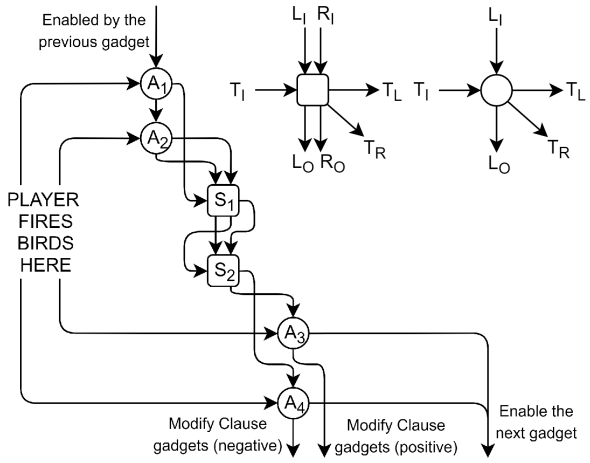


Figure 6: Structure of the Existential Quantifier gadget in ABED.

tive or negative. Doing this disables the EQ gadget and allows the player to enable the next gadget.

A schematic representation of how such a gadget can be constructed in Angry Birds using our defined gates is shown in Figure 6. Squares represent Selector gates, while circles represent AUT gates. The positions of arrows into or out of each gate correspond to the entrances and exits shown on the key. All AUT gates in this gadget have traverse paths that can be shot into by the player. An EQ gadget is enabled if A_1, A_2, S_1 and S_2 are open. Full schematic diagrams for all other gadgets are presented in [Stephenson *et al.*, 2020].

UQ-T Gadget. If a UQ-T gadget is enabled then it automatically sets the value of its associated variable to positive. The player can then enable the next gadget which also disables the UQ-T gadget.

UQ-F Gadget. If a UQ-F gadget is enabled then it alternates between allowing the player to do either of the following two actions: (A) the player can set the value of its associated variable to negative, which disables the UQ-F gadget and allows the player to enable the next adjacent gadget; or (B) the player can disable the UQ-F gadget and enable the next UQ-F gadget. The last UQ-F gadget does not have a next UQ-F gadget, and enabling this gadget will instead attempt to pass through the Finish gadget and solve the level.

Clause Gadget. A Clause gadget is "activated" if and only if its associated clause is satisfied (i.e. at least one of the literals in the associated clause is true). The level can be solved if and only if all Clause gadgets can be activated for each possible value combination of all universally quantified variables (abbreviated to UQVC). If the current gadget is a Clause gadget that is both enabled and activated, then the next gadget can be enabled.

Finish Gadget. The Finish gadget can be enabled if and only if all Clause gadgets are both enabled and activated.

Framework Design. The gadget associated with the quantifier with the largest scope (leftmost quantifier in Boolean Formula) is initially enabled (gadget pointed to by Start label in Figure 4), with the UQ-T version of the gadget being enabled if it is a universal quantifier, whilst all other gadgets are disabled. The player can enable the first UQ-F gadget at any time, but doing so when the Finish gadget is disabled will put the level into an unsolvable state. Enabling the first UQ-F gadget also disables all Clause and Finish gadgets. This action of enabling the first UQ-F gadget begins a new "framework cycle", with each framework cycle testing a specific UQVC. Once all possible UQVCs have been tested the player can pass through the Finish gadget and solve the level, assuming that the level is not in an unsolvable state.

Any given TQBF problem can be expressed as an ABED level in this form, and solving this level is equivalent to finding a solution to this problem. Thus, ABED is PSPACE-hard. We can extend this proof to PSPACE-completeness, by showing that ABED is in NPSPACE along with Savitch's theorem [Arora and Barak, 2009] that NPSPACE = PSPACE.

5 NP-Hardness of ABPD

By using a similar framework, we can also show that solving levels for ABPD is NP-hard. Our proof of NP-hardness reduces from the NP-complete problem 3-SAT, which involves deciding whether a given 3-CNF Boolean formula can be satisfied. Any 3-CNF Boolean formula can be represented using our TQBF framework by making all variables existentially quantified. This removes the need for any UQ-F or UQ-T gadgets, relying only on the EQ and Clause gadgets. The Finish gadget is also replaced by a single pig, that the player can kill if all clause gadgets are enabled and activated.

6 PSPACE-Hardness of ABPS

Another variant of the TQBF framework can also be used to show that solving levels for ABPS is PSPACE-hard. In this version all UQ-F gadgets are removed, and UQ-T gadgets are replaced by a new UQ-R gadget. These UQ-R gadgets are similar to the UQ-T gadget, except that when they are enabled they randomly set the value of their associated variable to ei-

ther positive or negative. The Finish gadget is also replaced by a single pig, similar to the previous ABPD proof.

The player can solve the level if all Clause gadgets are activated after the player has selected a value for each existentially quantified variable, and the value for each universally quantified variable has been (randomly) set to positive or negative. In order to guarantee that such a level can be solved, it must be possible to solve the level regardless of the outcome of the UQ-R gadgets. Essentially, we are no longer testing out every possible UQVC but are testing a single possible UQVC that is selected at random, which is equivalent for the purposes of determining if a level is always solvable.

7 EXPTIME-hardness of ABES

To show that solving levels for ABES is EXPTIME-hard we will reduce from a known EXPTIME-complete decision problem. For our proof we will use the problem of determining whether a player can force a victory for the two-player game G2, as shown in [Stockmeyer and Chandra, 1979].

While many classical two-player games such as Chess, Go and Checkers contain the mechanics necessary to mimic games such as G2, Angry Birds does not on first glance appear to be a suitable choice. Angry Birds is a single-player game and so does not inherently feature an opponent, in the traditional sense, against which to play. However, we can instead use the stochasticity of the physics engine as the opponent we will be facing. This stochasticity allows us to create situations where the player is uncertain about the exact outcome of shots that they make. By utilising this element of uncertainty in shot outcomes, we can create a "random" opponent, that will make random moves after each of the player's moves. Even though an opponent that just makes random moves may seem very easy to beat, the complexity of determining whether the player can force a victory for a given G2 instance is the same when facing both an opponent that plays optimally and one that plays randomly, as it is always possible that the random opponent will, by pure chance, actually play optimally (i.e. the player must assume Murphy's Law).

For our proof of EXPTIME-hardness we describe a method of combining several new types of gadget to create an ABES representation for any given setup of the game G2. A general framework diagram showing how these gadgets connect within the level space is shown in Figure 5, which uses the example Boolean formulas $(x \wedge \neg y \wedge z) \vee (\neg x \wedge y \wedge w)$ for the player and $(x \wedge y \wedge \neg z) \vee (\neg x \wedge y \wedge \neg w)$ for the opponent. The framework also contains an Ordering, Random, Choice and Result gadget, which are described below.

Ordering Gadget. The Ordering gadget ensures that the correct order of actions is followed by the player. All actions must be repeatedly performed in the following order:

1. Player makes their move (player can also pass).
2. Player checks whether their Boolean formula is satisfied.
3. Player makes a random move for the opponent (passing may occur as a random possibility).
4. Player checks whether the opponent's Boolean formula is satisfied.

Choice Gadget. The Choice gadget allows the player to make a single choice about which of their assigned variables will change in value during their move. The player should also have the option to pass. When a bird enters the Choice gadget via the Ordering gadget, the location at which it will exit is based on this choice made by the player. Depending on where the bird exits, the value of a single variable assigned to the player will either be changed or kept the same (pass).

Random Gadget. The Random gadget makes a random choice between multiple options, based on the stochasticity of the game engine. When a bird enters the Random gadget there are several possible locations where it can exit, each of which has a probability of occurring that is greater than zero. Depending on where the bird exits, the value of a single variable assigned to the opponent will either be changed or kept the same (pass).

Clause Gadget. Each Clause gadget represents a specific clause from either the player's or opponent's Boolean formula, and is "activated" if its associated clause is satisfied.

Result Gadget. The Result gadget is used to decide if the player has won (i.e. level solved). If the player's Boolean formula is satisfied after they have made a move, then the player can travel to the Result gadget from one of their activated Clause gadgets, allowing them to "pass through" the Result gadget and solve the level. If the opponent's Boolean formula is satisfied after a random move was made for them, then the player will be forced to travel to the Result gadget from one of the opponent's activated Clause gadgets, which will then block the Result gadget and make the level unsolvable.

Framework Design. The player fires a bird into the Ordering gadget to make the majority of actions, as well as into the Choice gadget to dictate which of their assigned variables will change in value for their next move. For our general framework diagram (Figure 5), an arrow into the left side of a Clause gadget indicates that the value of a variable is being changed, while an arrow into the right side indicates that the Clause gadget is being checked for activation (i.e. check if the associated clause is satisfied). The arrow into the left side of the Result gadget signifies that the level is lost (unsolvable), while the arrow into the right side signifies that the level is won (solved). Lastly, the arrow into the left side of the Choice gadget carries out the player's chosen move, while the arrow into the right side allows the player to specify the move they wish to make next.

Any given setup of G2 can be expressed as an ABES level in this form, and solving this level is equivalent to winning that game of G2 (against a random opponent).

8 Conclusion

In this paper, we have proven that the task of deciding whether a given Angry Birds level can be solved is either NP-hard, PSPACE-hard, PSPACE-complete or EXPTIME-hard, depending on the version of the game being used. Our use of unknown and changing environmental variables as the opponent which the player is facing, is a unique view of the problem and opens up the possibility of proving many other games EXPTIME-hard using this methodology.

References

[Aloupis *et al.*, 2014] Greg Aloupis, Erik D. Demaine, Alan Guo, and Giovanni Viglietta. Classic Nintendo games are (computationally) hard. In *Proceedings of the 7th International Conference on Fun with Algorithms*, pages 40–51, 2014.

[Arora and Barak, 2009] Sanjeev Arora and Boaz Barak. *Computational Complexity: A Modern Approach*. Cambridge University Press, New York, NY, USA, 1st edition, 2009.

[Bosboom *et al.*, 2018] Jeffrey Bosboom, Erik D. Demaine, Adam Hesterberg, Jayson Lynch, and Erik Waingarten. Mario Kart is hard. In *MIT Open Access Articles*, pages 1–12, 2018.

[Demaine *et al.*, 2016a] Erik D. Demaine, Joshua Lockhart, and Jayson Lynch. The computational complexity of Portal and other 3D video games. *CoRR*, arXiv:1611.10319:1–24, 2016.

[Demaine *et al.*, 2016b] Erik D. Demaine, Giovanni Viglietta, and Aaron Williams. Super Mario Bros. is harder/easier than we thought. In *Proceedings of the 8th International Conference on Fun with Algorithms*, pages 1–15, 2016.

[Forišek, 2010] Michal Forišek. Computational complexity of two-dimensional platform games. In *Proceedings of the 5th International Conference on Fun with Algorithms*, pages 214–227, 2010.

[Gualà *et al.*, 2014] Luciano Gualà, Stefano Leucci, and Emanuele Natale. Bejeweled, Candy Crush and other match-three games are (NP-)hard. In *Proceedings of the 2014 IEEE Conference on Computational Intelligence and Games*, pages 1–8, 2014.

[Renz *et al.*, 2019] J. Renz, X. Ge, M. J. B. Stephenson, and P. Zhang. AI meets Angry Birds. *Nature Machine Intelligence*, 1(7):328–328, 2019.

[Stephenson *et al.*, 2020] Matthew Stephenson, Jochen Renz, and Xiaoyu Ge. The computational complexity of Angry Birds. *Artificial Intelligence*, 280:103232, 2020.

[Stockmeyer and Chandra, 1979] Larry J. Stockmeyer and Ashok K. Chandra. Provably difficult combinatorial games. *SIAM Journal on Computing*, 8(2):151–174, 1979.

[Viglietta, 2014a] Giovanni Viglietta. Gaming is a hard job, but someone has to do it! *Theory of Computing Systems*, 54:595–621, 2014.

[Viglietta, 2014b] Giovanni Viglietta. Lemmings is PSPACE-complete. In *Proceedings of the 7th International conference on Fun with Algorithms*, pages 340–351, 2014.