The Computational Complexity of Angry Birds and Similar Physics-Simulation Games

Matthew Stephenson and **Jochen Renz** and **Xiaoyu Ge**

Research School of Computer Science
Australian National University
Canberra, Australia
matthew.stephenson@anu.edu.au, jochen.renz@anu.edu.au, xiaoyu.ge@anu.edu.au

Abstract

This paper presents several proofs for the computational complexity of the popular physics-based puzzle game Angry Birds. By using a combination of different gadgets within this game's environment, we can demonstrate that the problem of solving Angry Birds levels is NP-hard. Proof of NP-hardness is by reduction from a known NP-complete problem, in this case 3-SAT. In addition, we are able to show that the original version of Angry Birds is within NP and therefore also NP-complete. These proofs can be extended to other physics-based games with similar mechanics.

Introduction

The computational complexity of playing different video games has been the subject of much investigation over the past decade, with many papers demonstrating specific video games to be either NP-hard or NP-complete. However, this has mostly been carried out on traditional style platformers (Aloupis et al. 2014; Forišek 2010) or primitive puzzle games (Kendall, Parkes, and Spoerer 2008; Viglietta 2014). In this paper, we analyse the complexity of playing the original version of the video game Angry Birds, which is a sophisticated physics-based puzzle game.

The objective of each level in this game is to hit a number of predefined targets (pigs) with a limited number of shots (birds), often utilising or avoiding blocks and other game elements to achieve this. This game differs greatly from those previously investigated due to the fact that the player always makes their shots from the same location (slingshot position) and can only vary the speed and angle at which a bird travels from it. This heavily reduces the amount of control that the player has over each bird's movement, with the game's physics engine being used to determine the outcome of shots after they are made. The absence of a single highly controllable Avatar means that the frameworks applied to most previous game types, such as platformers, are no longer applicable and new ones must be created.

In order to prove the computational complexity of solving levels for Angry Birds we will reduce from a known NP-complete problem. For our proofs we will use reduction from the problem 3-SAT, which has previously been

used to show the complexity of many different video games. These include Lemmings (Cormode 2004), Portal (Demaine, Lockhart, and Lynch 2016), Candy Crush (Walsh 2014), Bejeweled (Gualà, Leucci, and Natale 2014) and multiple classic Nintendo games (Aloupis et al. 2014). Alternative compelxity proofs for a variety of other video games include both older titles, such as Tetris (Demaine, Hohenberger, and Liben-Nowell 2003), Minesweeper (Kaye 2000) and Pac-Man (Viglietta 2014), as well as more modern games, such as Crash Bandicoot (Forišek 2010) and multiple first-person shooters (Demaine, Lockhart, and Lynch 2016).

Complexity proofs have also been presented for many different block pushing puzzle games, including Sokoban (Cullberson 1998), Bloxorz (van der Zanden and Bodlaender 2015) and many varieties of PushPush (Demaine, Demaine, and O'Rourke 2000; Demaine, Hearn, and Hoffmann 2002; Demaine, Hoffmann, and Holzer 2004). These proofs have been used to advance our understanding of motion planning models, due to their real-world similarities (Demaine et al. 2001). It is therefore important that the computational complexity of physics-based games is investigated further, as playing video games such as Angry Birds has much in common with other real-world AI and robotics problems (Renz et al. 2016).

The remainder of this paper is organised as follows: The next section formally defines the Angry Birds game; We then present a proof that playing Angry Birds is NP-hard by reduction from 3-SAT; This proof is then extended to NP-complete, by demonstrating that the original version of Angry Birds is also in NP; We then describe how these proofs can be generalised to other physics-based games; Lastly, we conclude this work and propose future possibilities.

Angry Birds Game Definition

Angry Birds is a popular physics-based puzzle game in which the objective is to kill all the pigs within a 2D level space using a set number of birds. An example Angry Birds level is shown in Figure 1. Each level has a predefined size and any game element that moves outside of its boundaries is destroyed. The area below the level space is comprised of solid ground that cannot be moved or changed in any way, although other elements can be placed on or bounced off of it. Players make their shots sequentially and in a predefined order, with all birds being fired from the location of the
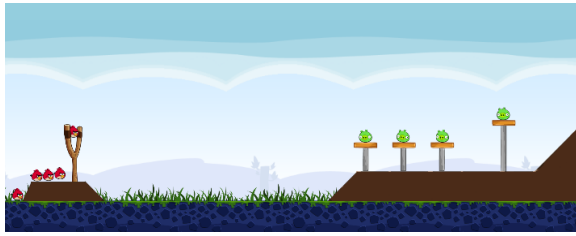
Figure 1: Screenshot of a level for the Angry Birds game.

slingshot. The player can alter the speed (up to a set maximum) and angle with which these birds are fired from the slingshot but cannot alter the bird's flight trajectory after doing so, except in the case of some special bird types with secondary effects that can be activated by the player. The level space can also contain many other game elements, such as blocks, static terrain, explosives, etc. All game elements have a positive fixed mass, friction, dimensions and shape (based on their type), and no element may overlap any other. The level itself also has a fixed gravitational force that always acts downwards. Calculations done with regard to object movement and resolving collisions are simulated using a simplified physics engine based on Newtonian mechanics.

The description of an Angry Birds level can be formalised as $Level = (1^{L_x}, 1^{L_y}, slingshot, birds, pigs, other)$.

- L_x is the width of the level in pixels.

- L_y is the height of the level in pixels.

- $slingshot$ is the pixel coordinates (x, y) from which the player makes their shots.

- $birds$ is a list containing the type and order of the birds available.

- $pigs$ is a list containing the type, angle and pixel coordinates (x, y) of all the pigs.

- $other$ is a list containing the type, angle and pixel coordinates (x, y) of all other game elements.

The top left corner of a level is given the coordinate $(0, 0)$ and all other coordinates use this as a reference point. The width and height of a level must be specified as integer values, and all pixel coordinates (x, y) must be defined as integers within the level space. For technical reasons L_x and L_y are specified in Unary notation, so that the size of the level description is polynomial to these values themselves rather than their logarithms. There is also a finite sized list which contains all the types of birds, pigs and other game elements, as well as their properties (e.g. mass, friction, size, etc.). This list is fixed in size and so is not relevant to the complexity of the game.

A strategy for solving a given level description consists of a sequence of ordered pixel coordinates (x, y) which determines the speed and angle with which each of the available birds is fired (release points). While the speed with which a bird can be fired is bounded, and therefore can only be determined to a set level of precision, the angle of the shot can be any rational value determined by the release point given. Therefore, the precision with which shots can be specified,

as well as the number of bits required to define a shot and the number of distinct shots possible, is polynomial relative to the size of the level's description. A tap time is also included for activating each bird's secondary effect if it has one. The general decision problem we are considering in this paper is whether, for a given Angry Birds level description, there exists a strategy that results in all pigs being killed.

For the proofs described in this paper only the following game elements are required:

- **Red Birds:** These are the most basic bird type within the game and possess no special abilities. Once the player has determined the speed and angle with which to fire this bird it follows a trajectory determined by this and the gravity of the level, which the player cannot subsequently affect. This bird has no secondary effect so a tap time is not needed.

- **Small Pigs:** These are the most basic pig type within the game and are killed once they are hit with either a bird or block.

- **Breakable Blocks:** These are blocks that are removed from the level if they are hit either by a bird or another block. They are represented in this paper by blocks made of glass.

- **Unbreakable Blocks;** These are blocks that do not break if they are hit but instead react in a semi-realistic physical way, moving and rotating if forces are applied to them. They are represented in this paper by blocks made of stone.

  Note. In Angry Birds, each block has a health value that dictates how much damage it can take before breaking. When a block is hit by another game element it takes damage (reduces health) proportional to the speed and mass of the impacting object. When the health of a block falls below zero it is removed from the level space. To create breakable blocks we can simply set the health of the blocks to zero, and for unbreakable blocks we set the health value high enough such that the player cannot break these blocks with the birds they have (i.e. a health greater than the combined energies of all game elements in the level).

- **Static Terrain:** This is simply a set area of the level that cannot move or be destroyed. It is represented in this paper by plain, untextured, brown areas. The ground at the bottom of the level space behaves in the same way as this.

For our proofs, we assume that the size of a level is not bounded by the game engine and that the player's next shot only occurs once all game elements are stationary. This latter restriction is only a simplification to make the construction process easier to understand. We also assume that the physics calculations performed by the game engine are not affected as the size of the level increases (no glitches or other simulation errors) and that there is no arbitrary fixed precision with regard to the angles that shots can have. As the exact physics engine parameters used for Angry Birds are not currently available for analysis, all assumptions made about the game and its underlying properties are determined through careful observation of the original levels.
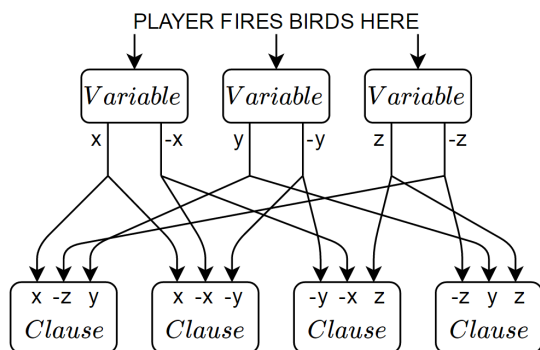
Figure 2: General framework for NP-hardness.

Figure 3: Example model of the Variable gadget used.

Angry Birds is NP-hard

Theorem 1. *The problem of solving levels for Angry Birds is NP-hard.*

For our proof of NP-hardness we will use a variation of a general framework for platformers, similar to that used for many past games (Aloupis et al. 2014; Demaine, Demaine, and O'Rourke 2000; Demaine, Lockhart, and Lynch 2016), see Figure 2. This framework can be used to prove that a game is NP-hard by constructing the necessary gadgets. This framework reduces from the NP-complete problem 3-SAT, which consists of deciding whether a 3-CNF Boolean formula can be made "true" for any combination of variable values. For example, Figure 2 uses the Boolean formula $(x \vee \neg z \vee y) \wedge (x \vee \neg x \vee \neg y) \wedge (\neg y \vee \neg x \vee z) \wedge (\neg z \vee y \vee z)$. For each variable in the Boolean formula there is an associated Variable gadget and for each clause in the Boolean formula there is an associated Clause gadget.

The player can fire a bird into any of the Variable gadgets within the level but cannot directly fire into any other gadget. Each Variable gadget allows the player to set the truth value of the associated Boolean variable, but this choice may only be made once. Either choice then "activates" the Clause gadgets containing the chosen literal. Crossover gadgets are used to deal with overlapping lines between Variable and Clause gadgets (not needed for every game). Once all Clause gadgets have been "activated" the level is solved. If all Clause gadgets can be activated, then there exists a solution to the associated Boolean formula. Thus, any game can be shown to be NP-hard if the required gadgets can be successfully implemented within the game's environment and the reduction from Boolean formula to level description can be achieved in polynomial time.

Variable Gadget

An example of a Variable gadget implementation for Angry Birds is shown in Figure 3. This gadget allows the player to choose the truth value of an associated Boolean variable.

Lemma 1.1. *A Variable gadget can be used to indicate one of two Binary choices, positive or negative, and can only be used once.*

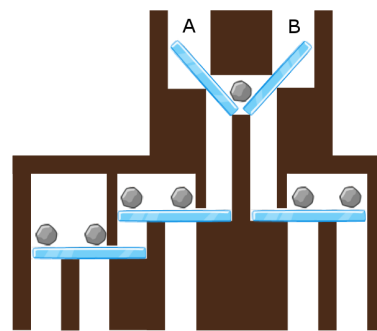*Proof.* The player can fire a bird into either the left entrance (A) to indicate a positive value, or the right entrance (B) to indicate a negative value, for the associated Boolean variable. Depending on the player's choice this causes one of the angled glass blocks to break, resulting in the highest stone ball falling into either the left hole if a positive literal was selected, or the right hole if a negative literal was selected. The bird itself cannot fall down any hole as the gaps between the entrances and the holes are too small for it to pass through. As there is only one ball at the top of the gadget the player can only make this choice once. □

Lemma 1.2. *A Variable gadget can be used to activate as many Clause gadgets as necessary.*

Proof. Once the player has made their shot the ball will fall down the selected hole and break the glass block below it. This then causes the balls supported by the glass block to fall either down the tunnels below them (which lead to the corresponding Clause gadgets for the selected literal), or onto another glass block which supports more balls. Each glass block is wide enough to support a maximum of two stone balls, so if more balls are needed the second ball will break another glass block which supports another two balls. This process continues until as many balls fall down tunnels as there are Clause gadgets that contain the literal chosen. Each of these balls then travel down tunnels that lead them to specific Clause gadgets, which are then activated. □

Lemma 1.3. *The width and height of a Variable gadget, as well as the number of game elements it contains, is polynomial with respect to the number of Clause gadgets that contain its associated Boolean variable.*

Proof. Let V_W and V_H be constants representing the width and height respectively of the smallest non-redundant Variable gadget, with only one clause containing each of its literal choices (i.e. contains only four glass blocks and three stone balls). For each additional clause that contains the Boolean variable associated with this Variable gadget, at most one glass block and two stone blocks are needed on each side. Therefore, the width and height of any Variable gadget is bounded by the polynomial expressions $V_W + 2C(G_W - B_W)$ and $V_H + 2C(G_H + B_H)$ respectively, where C is the number of clauses in the associated Boolean formula, G_W and G_H are the width and height of the glass rectangular block, and B_W and B_H are the width and height
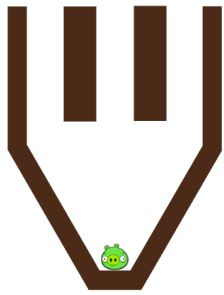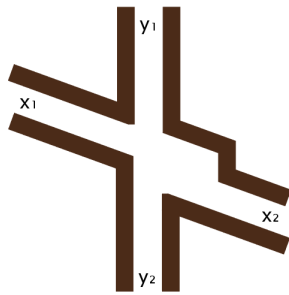
Figure 4: Model of the Clause gadget used.

Figure 5: Model of the Crossover gadget used.

of the stone ball. Likewise, the number of glass and stone blocks in any Variable gadget is bounded by the polynomial expressions $2C + 2$ and $4C + 1$ respectively. □

Clause Gadget

The Clause gadget implementation for Angry Birds is shown in Figure 4. The balls from each Variable gadget fall down tunnels (based on the player's choice) which lead to the corresponding Clause gadgets that contain the chosen literal, as defined by the associated Boolean formula.

Lemma 1.4. *A Clause gadget can be used to represent a chosen clause from any 3-CNF Boolean formula.*

Proof. The three tunnels leading into the top of the Clause gadget each come from a particular literal choice within a Variable gadget, as determined by the 3-CNF Boolean formula. Any ball that ends up in the Clause gadget will hit the pig and "activate" the Clause gadget. A level of Angry Birds is solved once all pigs have been killed, i.e. once all Clause gadgets are "activated" or all clauses within the Boolean formula are "true". □

Lemma 1.5. *The width and height of a Clause gadget, as well as the number of game elements it contains, is constant, regardless of the Boolean formula being used.*

Crossover Gadget

The Crossover gadget implementation for Angry Birds is shown in Figure 5. This gadget is used whenever two tunnels between Variable and Clause gadgets cross. The leftmost intersecting tunnel enters at x_1 and exits at x_2, whilst the rightmost tunnel enters at y_1 and exits at y_2.

Lemma 1.6. *A Crossover gadget can be used to transport balls from x_1 to x_2 without leakage to y_1 or y_2, or from y_1 to y_2 without leakage to x_1 or x_2.*

Proof. The Crossover gadget consists of two tunnels, a vertical tunnel and a tunnel at a fixed rational angle (θ). Any ball that enters the gadget at y_1 will fall straight downwards and exit out of y_2 without any risk of entering the angled tunnel. Any ball that enters the gadget at x_1 will roll down the slope, assuming that the angle θ is greater than or equal to the necessary angle to overcome the rolling friction between

the ball and the ground, until it overlaps with the vertical tunnel. Once this happens the ball will start to fall downwards but its momentum will continue to carry it horizontally until it no longer overlaps the vertical tunnel, assuming that x_2 is placed low enough to ensure this. The necessary downwards drop (D) for the angled tunnel can be easily calculated based on the mass and friction of the ball, as well as the gravitational force of the level and the angle θ. □

Lemma 1.7. *The width and height of a Crossover gadget, as well as the number of game elements it contains, is constant, regardless of the Boolean formula being used.*

Level Construction

As Angry Birds is a game that relies heavily on physics simulations to resolve player actions, the positions of the gadgets within a level are extremely important. Elements within the game are bound by the physics of their environment and the only immediate control the player has is with regard to the shots they make. For this reason, it is necessary to confirm that the gadgets described can be successfully arranged throughout the level space.

Lemma 1.8. *Any given 3-SAT problem can be reduced to an Angry Birds level description in polynomial time.*

Proof. We have already shown that each of the necessary gadgets can be created using a polynomial amount of space and elements, and can therefore also be described in polynomial time. Consequently, the only remaining requirement is that all the gadgets can be successfully arranged throughout the level in polynomial time, relative to the size of the 3-CNF Boolean formula. As the number of gadgets required is clearly polynomial, it suffices to describe a polynomial time method for determining the location of each gadget, as well as the level's width, height, slingshot position and number of birds.

Although the speed at which a bird can be fired from the slingshot is bounded (less than or equal to a maximum velocity v_M), we can still ensure that all gadgets are reachable from the slingshot by placing them lower in the level. As there is no air resistance, the trajectory of a fired bird follows a simple parabolic curve for projectile motion, $y = x \tan(\phi) - \frac{g}{2v_0^2 \cos^2(\phi)} x^2$, where v_0 is the initial velocity of the fired bird, ϕ is the initial angle with which the bird was fired, and g is the gravitational force of the level. This means that in order to ensure that all Variable gadgets are reachable they must be placed at a distance below the slingshot equal to or greater than $-V_T + \frac{g}{v_M^2} V_T^2$, where V_T is the combined width of all Variable gadgets. We can also use the same formula to calculate the maximum height that a bird fired from the slingshot can reach, $\frac{v_M^2}{2g}$. Using this we can set the position of the slingshot to $(0, \frac{-v_M^2}{2g})$ and place all Variable gadgets the required distance below this in a horizontal alignment against the left side of the level.

With the positions of the Variable gadgets defined, we can now place the Clause gadgets relative to them. All Clause gadgets are horizontally aligned next to each other

and placed directly to the right of the Variable gadgets. The Clause gadgets are then moved downwards a distance equal to or greater than $T(S + D(T - 1) + W(\tan(\theta)))$, where T is the total number of Variable gadget tunnels (equivalent to $3C$), W is the combined width or all Variable gadgets and Clause gadgets, D and θ are the same as in Lemma 1.6, and S is the size of the Variable gadget tunnels (must be wide enough for ball to fit down). Each Variable gadget tunnel is associated with a specific Clause gadget tunnel that contains the literal associated with it. These are allocated based on horizontal positioning, so the leftmost Variable gadget tunnel for a specific literal is associated the leftmost Clause gadget that contains this literal and vice versa. Each Variable gadget tunnel is then also assigned a number based on its x-axis position, with the leftmost tunnel getting the value one, and the rightmost tunnel getting the value T. The space between the Variable and Clause gadgets is divided up into T evenly sized rows, each of which should have a height of at least $S + D(T - 1) + W(\tan(\theta))$. This row size allows for the worst-case scenario where a tunnel intersects every other tunnel on the way to its allocated Clause gadget. Each row is assigned a number based on its y-axis position, with the bottom row getting the value one, and the top row getting the value T.

For each Variable gadget tunnel perform the following. Firstly, drop the tunnel vertically down until it reaches the row corresponding to its assigned number. Secondly, direct the tunnel at an angle of θ towards its associated Clause gadget tunnel until it is directly above it. Finally, drop the tunnel vertically down until it reaches its associated Clause gadget tunnel. Any intersections that occur between two tunnels will always be between a tunnel directed straight down, and one at angle θ. This situation is dealt with using the Crossover gadget previously described. There is no risk of balls colliding within a Crossover gadget, as the tunnels associated with a specific literal never intersect. This construction process can be easily accomplished in polynomial time, relative to the number of Clause gadgets. An example diagram showing how these tunnels lead from the Variable gadgets to the Clause gadgets is shown in Figure 6, using the same example Boolean formula as in Figure 2. This is not a complete to scale construction, as the angled portion of each tunnel should be contained within its own allocated row, but has been compressed here to save space.

In addition, we need to guarantee that there are enough release points available to allow for a bird to be fired into either entrance for each Variable gadget. To ensure this we will move everything constructed so far V_T pixels to the right. This means that the required width and height of the level space needed for placing all necessary gadgets can now be calculated. The required width of a level is equal to $(2V_T + C_T)$, where C_T is the combined width of all Clause gadgets. The required height of a level is equal to,
$$-V_T + \frac{g}{v_M^2}V_T^2 + \frac{v_M^2}{2g} + T(S + D(T - 1) + W(\tan(\theta))).$$
Lastly, the number of birds needed is equal to the number of Variable gadgets. □

As we have constructed the necessary gadgets and can position them within the game's environment in polynomial

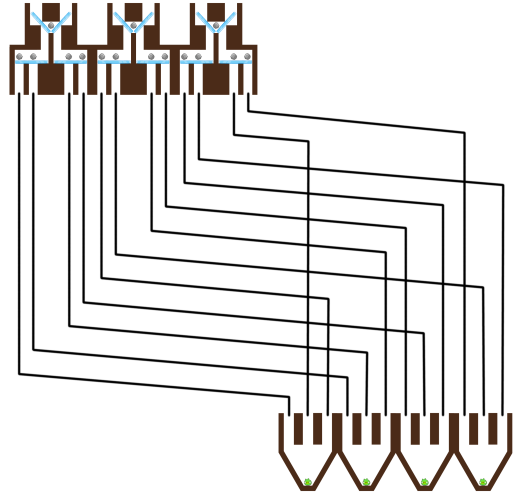Figure 6: Framework construction example (not to scale).

time, the problem of solving Angry Birds levels is NP-hard.

Angry Birds is NP-complete

Theorem 2. *The problem of solving levels for Angry Birds is NP-complete.*

Having shown that Angry Birds is NP-hard, the only remaining requirement for completeness is that it also be in NP. The problem of solving an Angry Birds level can be defined as within NP if it is possible to solve any level in polynomial time using a non-deterministic Turing machine. This requirement is equivalent to showing that any strategy for a given level can be verified on a deterministic Turing machine in polynomial time, relative to the size of the level's description, and that there are a finite number of states and strategies for any given level.

Lemma 2.1. *There are a finite number of states and strategies for any given Angry Birds level.*

Proof. The state of a level is defined based on the current attribute values of all the elements within it. All these values are defined as rational numbers that each take up a finite amount of memory. Therefore, it must also be possible to define the current state of any given level in a finite amount of memory. Thus, the total number of states for any given level is finite. As the number of shots and release points for any given level is polynomial, relative to the size of the level's description, the number of possible strategies for a level is also finite. □

Lemma 2.2. *Any strategy for a given Angry Birds level can be verified in polynomial time.*

Proof. The number of elements within a level is clearly polynomial, relative to the size of its description. The total amount of energy that a non-static game element has at any given time can be defined as the sum of four energy values:

- Kinetic Energy, which is determined by its velocity and mass $(\frac{mv^2}{2})$.

- Gravitational Potential Energy, which is determined by its location and mass (mgh).
- Effect Energy, which is any extra energy that can be released by the element due to its effect. This type of energy is only possessed by specific game elements (e.g. TNT or black birds) and is always constant depending on the element's type.
- Shot Potential Energy, which is the maximum amount of energy that the slingshot can add to the element. This type of energy is only possessed by birds that are yet to be fired from the slingshot, and is determined by the bird's mass and the maximum velocity at which it can be fired ($\frac{mv_M^2}{2}$).

The total amount of energy within a level directly after initialisation is equal to the combined energies of all the elements within it (E_L). No energy is ever added to the level after this point, only removed. Energy is removed from a level either when one game element collides with another, or moves out of bounds (all the element's remaining energy lost). There is a minimum velocity for a moving element (set by the game engine), which means that there is also a minimum non-zero amount of kinetic energy that an element can possess, which must be equal to the minimum amount of energy (E_m) lost during a collision (assume that there is always a loss of some energy during a collision). Because of this, the maximum number of collisions that can occur for a given level is $\frac{E_L}{E_m}$. The longest amount of time that can pass without at least one collision occurring, or an element moving out of bounds, is $2\sqrt{\frac{2L_y}{g}}$ (following parabolic path from lowest point in level to highest point and back down under the influence of gravity). Thus, the maximum theoretical amount of time (T) that any strategy can take to carry out (ignoring time between one shot ending and the next being performed) is described by equation (1):

$$T = \frac{2E_L}{E_m}\sqrt{\frac{2L_y}{g}} \qquad (1)$$

(E_m and g are fixed constants defined by the game engine and must be greater than zero)

This means that any given strategy for an Angry Birds level can be verified in polynomial time. $\square$

As we have shown that any given level within this game environment has a finite number of states/strategies, and that a strategy for solving it can always be verified in polynomial time, we can conclude that the problem of solving Angry Birds levels is in NP, and thus also NP-complete. This particular proof of completeness does not hold for all versions of Angry Birds, as some newer incarnations of the game feature "bounce pads", continuously moving platforms, or other elements that do not possess a finite amount of energy.

Generalisation

The NP-hardness proof described in this paper can be easily replicated in many other games similar to Angry Birds, as long as the necessary gadgets can be constructed. In general, this means that the computational complexity of any physics-based game can likely be established using our framework, as long as the following requirements hold:

- A level within the game is solved by, or can only be solved after, hitting a set number of targets.
- The game contains both static and non-static elements.
- The game contains elements that can either be destroyed or moved as a result of the player's actions.
- The physics engine utilised by the game allows for rudimentary systems of gravity and momentum (almost all simple physics engines should contain this) which affect certain non-static elements.
- The only influence a player has over elements within the gadget framework is a single binary decision made for each Variable gadget, with regard to the movement of a non-controllable game element (i.e. interaction with the gadget framework is only through Variable gadget choices).
- The game must be able to accommodate any number of Variable/Clause gadgets, and the player should be able to make at least as many decisions as Variable gadgets within each level (i.e. the size of a level and the number of decisions the player can make must be able to increase indefinitely).

Whilst we cannot be certain that this generalisation is applicable to all games that contain these features, using them as loose restrictions allows us to show that many other physics-based games are NP-hard. This includes both games that are similar in play style to Angry Birds, such as Siege Hero or Fragger, as well as games that play considerably differently, such as Where's My Water, Cut the Rope 2 or The Incredible Machine. Proofs for these games cannot be provided here due to lack of sufficient space, but will hopefully be presented in greater detail at some future date.

Conclusion

In this paper we have proven that the task of solving levels for the original version of Angry Birds is NP-complete. This means that this problem is at least as hard as any other problem in NP and can be reduced to or from any other NP-complete problem. Additional complications such as imprecise or noisy input data, results of actions being affected by unknown or random values, and a huge state and action space, make the task of solving Angry Birds levels even more challenging. We have also shown how these proofs can be generalised to other physics-based games with similar mechanics.

This work greatly increases the variety of games that have been investigated within the field of computational complexity, dealing both with the introduction of physics constraints and limitations, as well as the lack of a single highly controllable Avatar. However, there is still a huge collection of physics-based and other non-traditional puzzle games that are available for future analysis, which do not follow the typical structure of those previously studied. We are therefore hopeful that this work will inspire future research into a more diverse range of game types and problems.

References

Aloupis, G.; Demaine, E. D.; Guo, A.; and Viglietta, G. 2014. Classic Nintendo games are (computationally) hard. In *Proceedings of the 7th International Conference on Fun with Algorithms*, 40–51.

Cormode, G. 2004. The hardness of the Lemmings game, or oh no, more NP-completeness proofs. In *Proceedings of the 3rd International Conference on Fun with Algorithms*, 65–76.

Cullberson, J. C. 1998. Sokoban is PSPACE-complete. In *Proceedings of the International Conference on Fun with Algorithms*, 65–76.

Demaine, E. D.; Demaine, M. L.; Hoffmann, M.; and O'Rourke, J. 2001. Pushing blocks is hard. In *Proceedings of the 13th Canadian Conference on Computational Geometry*, 21–36.

Demaine, E. D.; Demaine, M. L.; and O'Rourke, J. 2000. PushPush and Push-1 are NP-hard in 2D. In *Proceedings of the 12th Canadian Conference on Computational Geometry*, 211–219.

Demaine, E. D.; Hearn, R. A.; and Hoffmann, M. 2002. Push-2-F is PSPACE-complete. In *Proceedings of the 14th Canadian Conference on Computational Geometry*, 31–35.

Demaine, E. D.; Hoffmann, M.; and Holzer, M. 2004. PushPush-k is PSPACE-complete. In *Proceedings of the 3rd International Conference on FUN with Algorithms*, 159–170.

Demaine, E. D.; Hohenberger, S.; and Liben-Nowell, D. 2003. Tetris is hard, even to approximate. In *Computing and Combinatorics, 9th Annual International Conference*, 351–363.

Demaine, E. D.; Lockhart, J.; and Lynch, J. 2016. The computational complexity of Portal and other 3D video games. *CoRR* arXiv:1611.10319.

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

Gualà, L.; Leucci, S.; and Natale, E. 2014. Bejeweled, Candy Crush and other match-three games are (NP-)hard. In *Proceedings of the 2014 IEEE Conference on Computational Intelligence and Games*, 1–8.

Kaye, R. 2000. Minesweeper is NP-complete. *The Mathematical Intelligence* 22:9–15.

Kendall, G.; Parkes, A.; and Spoerer, K. 2008. A survey of NP-complete puzzles. *ICGA Journal* 31:13–34.

Renz, J.; Ge, X.; Verma, R.; and Zhang, P. 2016. Angry Birds as a challenge for artificial intelligence. In *Proceedings of the 30th AAAI Conference*, 4338–4339.

van der Zanden, T. C., and Bodlaender, H. L. 2015. PSPACE-completeness of Bloxorz and of games with 2-buttons. In *Algorithms and Complexity: 9th International Conference*, 403–415.

Viglietta, G. 2014. Gaming is a hard job, but someone has to do it! *Theory of Computing Systems* 54:595621.

Walsh, T. 2014. Candy Crush is NP-hard. *CoRR* arXiv:1403.1911.