Fireboy & Watergirl is PSPACE-Complete

Kurt Gu Christopher Liu

Introduction

Fireboy & Watergirl (F&W) is a two-player co-op puzzle game created by Oslo Albert and Jan Villanueva. Each level is a puzzle room full of different contraptions, and the goal of the players is to get to their respective exits and escape (Figure 1). Fireboy can step into lava but not water, while Watergirl can step into water but not lava. They can also collect diamonds to gain points: red diamonds can only be picked up by Fireboy, and blue diamonds can only be picked up by Watergirl. They can help each other get to their exits by pushing buttons and pulling levers to move doors and get past obstacles. Levers and buttons control doors of the same color. Levers can be set to a constant state, while buttons are temporary and need to be held down to change the state of the door. Fireboy and Watergirl have to be both at their respective exit to pass the level. All 6 games in the series can be played online here.

Figure 1: Level 1 of F&W (Assets © Oslo Albert)

Problem Statement

We will show that F&W is PSPACE-hard by reducing TQBF to the decision problem for F&W. Specifically, we reduce TQBF to a restricted version of F&W using only buttons, levers, and sliding doors/platforms, which demonstrates the hardness of F&W in general. This involves 3 parts:

showing that F&W can model 3-SAT, showing that F&W can be constructed to force the player to count in binary in Binary Reflected Gray Code order, and finally combining the previous two parts to model TQBF. Even though this is a two-player game, its co-op nature also gives the game a single-player flavor, as most levels can be solved by one player playing as both characters. This means that a reduction to Geography would not be appropriate here, and two players will mostly be utilized to simplify, and add (some) parallelism to, the reduction.

Modeling 3-SAT

If we allow levers to toggle the open/close states of all doors with the same color, we can model 3-SAT in the following way, as shown in figure 2. This particular configuration models the following 3-SAT formula: $(x_1 \vee \overline{x_2} \vee \overline{x_3}) \wedge (x_3 \vee x_4 \vee \overline{x_1}) \wedge (\overline{x_1} \vee x_2 \vee x_4)$, where x_1= green, x_2= blue, x_3= yellow, x_4= red. Each open door represents a TRUE value literal, and each closed door represents a FALSE value literal, and each floor level represents a clause in the formula. Toggling a lever will cause an open door to become closed and vice versa.

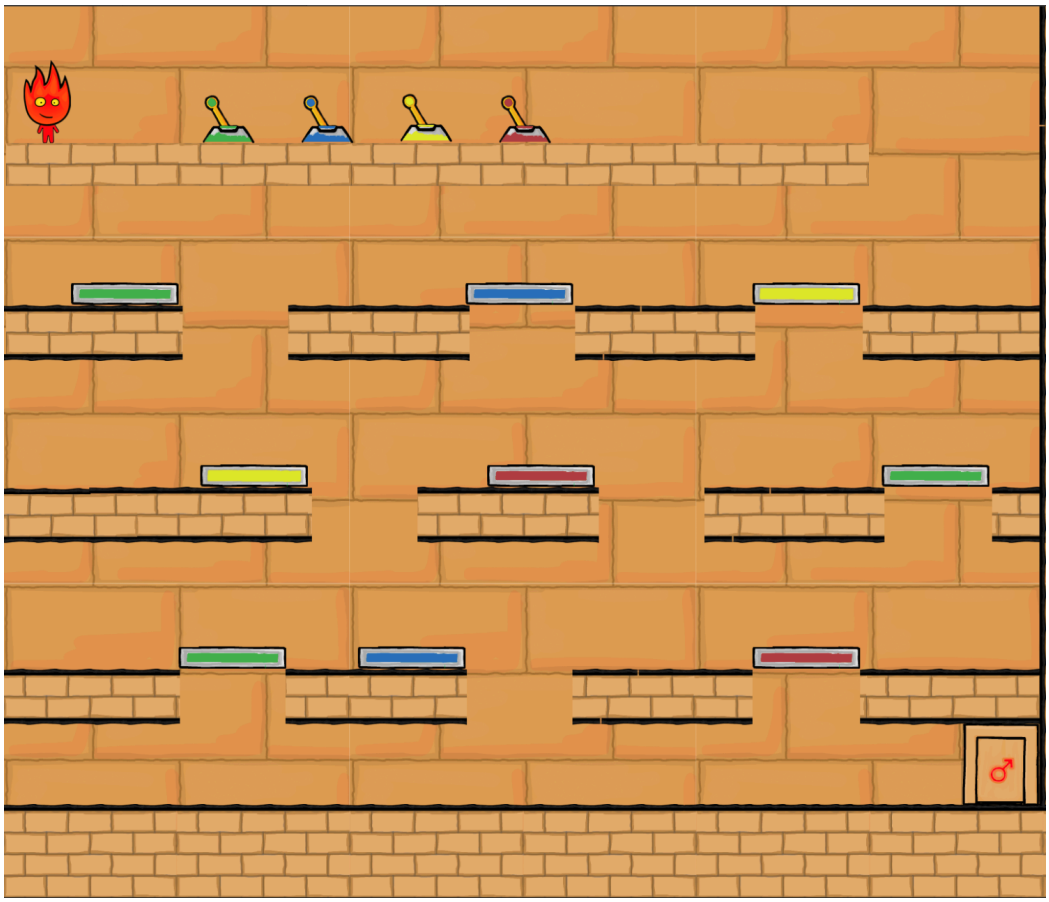

Figure 2: 3-SAT Gadget

To get to the exit at the bottom right of the map, Fireboy needs to set the levers such that there is at least one path from the top to the exit. If such a path exists, then at each level, there is at least one open door. This means that there must be at least one true literal in each clause, and our variables in the 3-SAT formula have a valid assignment. On the other hand, if there is no path from the top to the exit, then on at least one level, all doors are closed. Therefore at least one clause in our formula cannot be satisfied, in which case the 3-SAT formula does not have a valid assignment.

One-lever-one-door Gadget

If we want to restrict each lever to only be able to toggle one door to be more faithful to the original game, we can do so by extending each of the levers at the beginning of the level using the following gadget:

Initial **Toggled**

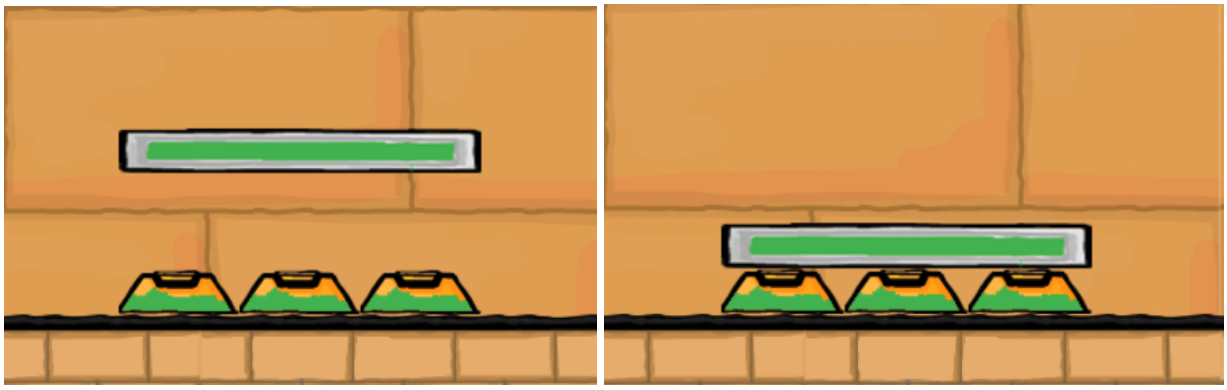

Figure 3: Example toggling gadget for X_1

Since levers can toggle doors over a long distance, this gadget can be put in an area of the map that is not accessible to the player to avoid cheating. Each button corresponds to one sliding door representing an instance of X_1, and the lever at the beginning in front of the player now toggles the large panel door that slides vertically, as pictured in Figure 3. When the lever is toggled by the player, the panel would drop down and compress all the buttons, each toggling the state of their respective doors. Buttons then would be reset if the player returned the lever to the original position.

Binary Reflected Gray Code

In order to reduce TQBF to F&W, it is necessary to force the player to test the clauses for every combination of values for the $\forall$ TQBF variables. In other words, since a TQBF formula alternates $\exists$ and $\forall$ variables, we need some mechanism to force players to attempt all permutations of the $\forall$ variables. We do this by constructing a gadget which forces the player to

count in Binary Reflected Gray Code, or BRGC. Each lever in the level represents a bit in the BRGC, as well as the value of a $\forall$ variable. We call this level **Gray**(n) for n levers, bits, and $\forall$ variables. **Gray**(3) is illustrated in Figure 4. Flipping each colored lever will move the horizontal door of the same color to open the chamber beneath which contains the next lever, and will also move the vertical door of that color to block the path beyond. Flipping the lever again will reset their positions. The first lever is the only one accessible from the start.

In order for the player to reach the exit, they must unblock their path by toggling the nth lever. Doing so requires toggling the lever n-1, which requires n-2, so on, until the 1st. We can also observe that since toggling any but the nth lever causes the path to the next lever and the exit to be blocked by a vertical door, after toggling the 1st lever and 2nd lever, we must toggle the 1st again, and more generally, toggling the kth lever for $k > 1$ requires us to afterwards repeat the process involved in toggling the k-1th lever. Each toggle of a lever models proceeding to the next BRGC string, as each BRGC string differs from the previous one by a single bit. Thus, when the player has successfully reached the exit, they will have iterated through all 2^n BRGC strings, and equivalently iterated through every permutation of $\forall$ variable values.

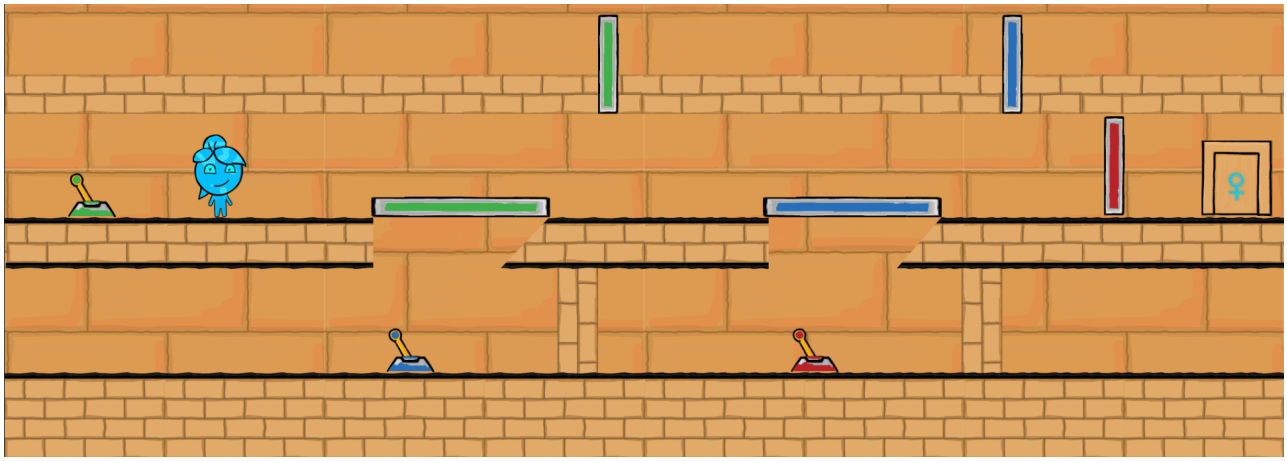

Figure 4: 3-Bit Binary Gray Code Counter

Putting It All Together

In order to complete our reduction of TQBF to F&W, we must combine our 3SAT and Gray Code gadgets, as well as implement the $\exists$ variables. To do this, we split up the level into two sections and place one of the controllable characters into each. One section includes our Gray Code gadget as well as additional mechanisms for the $\exists$ variables, and the other section includes the 3SAT clauses. The full level is shown in Figure 5.

For our counter section, we must make several additions to the Gray Code gadget. Depending on the $\forall$ lever being flipped, they also must have the opportunity to change the

assignments to some of the $\exists$ variables. Specifically, any $\exists$ variables coming later in the overall formula to the current $\forall$ variable being changed can also be changed. To accomplish this, we place a lever determining the $\exists$ variable positioned directly after a $\forall$ variable in the formula just before the vertical door controlled by the *next* $\forall$ variable. Observe that whenever we are able to flip a particular $\forall$ lever, the vertical door of the *next* $\forall$ variable in the formula is closed, blocking off the rest of the path. Thus, when flipping a $\forall$ lever we are only able to modify the $\exists$ variables coming after it in the clause. The first $\exists$ variable is set by a lever just before the final vertical door; it can be set at the very beginning but afterwards is blocked off until the level is essentially solved.

We also must make modifications to integrate the two sections together. Specifically, we must force the player character in the 3SAT section to traverse through all of the clauses between each time the character in the counter section changes any of the switches, to verify that each iteration is satisfiable. To accomplish this, we place two buttons b_1 and b_2 before and after the clauses, respectively. In the counter section, two vertical sliding doors around the starting location where the player character will drop down after their initial assignment of $\exists$ variables such that the doors block the player's path unless its associated button is being pressed. We place a door controlled by b_2 between the starting location and the 1st $\forall$ lever. We place a door controlled by b_1 between the starting location and the rest of the section.

Since b_1 is initially accessible without traversing the 3SAT clauses, the counter character can set all but the first $\exists$ variables at the start. However, before flipping the first $\forall$ switch, the 3SAT character must travel through all of the clauses to press b_2. Similarly, to flip any other switch, the 3SAT character must travel back through all of the clauses again to press b_1. Since the 3SAT character needs to sit on the button to let the counter character through, they cannot "cheat" by flipping a switch midway through traversal.

We can observe that whenever the counter section player moves between the first $\forall$ lever and any other $\forall$ lever, the 3SAT character must travel through all of the clause gadgets. Thus, in our configuration, because the counter character must flip the 1st $\forall$ switch every other time, then the 3SAT character must travel through all of the clauses before each flip of a $\forall$ switch. Thus, we can see that each iteration of variables must be checked for satisfiability.

By successfully reducing TQBF to F&W, we can thus conclude that the F&W decision problem is PSPACE-hard.

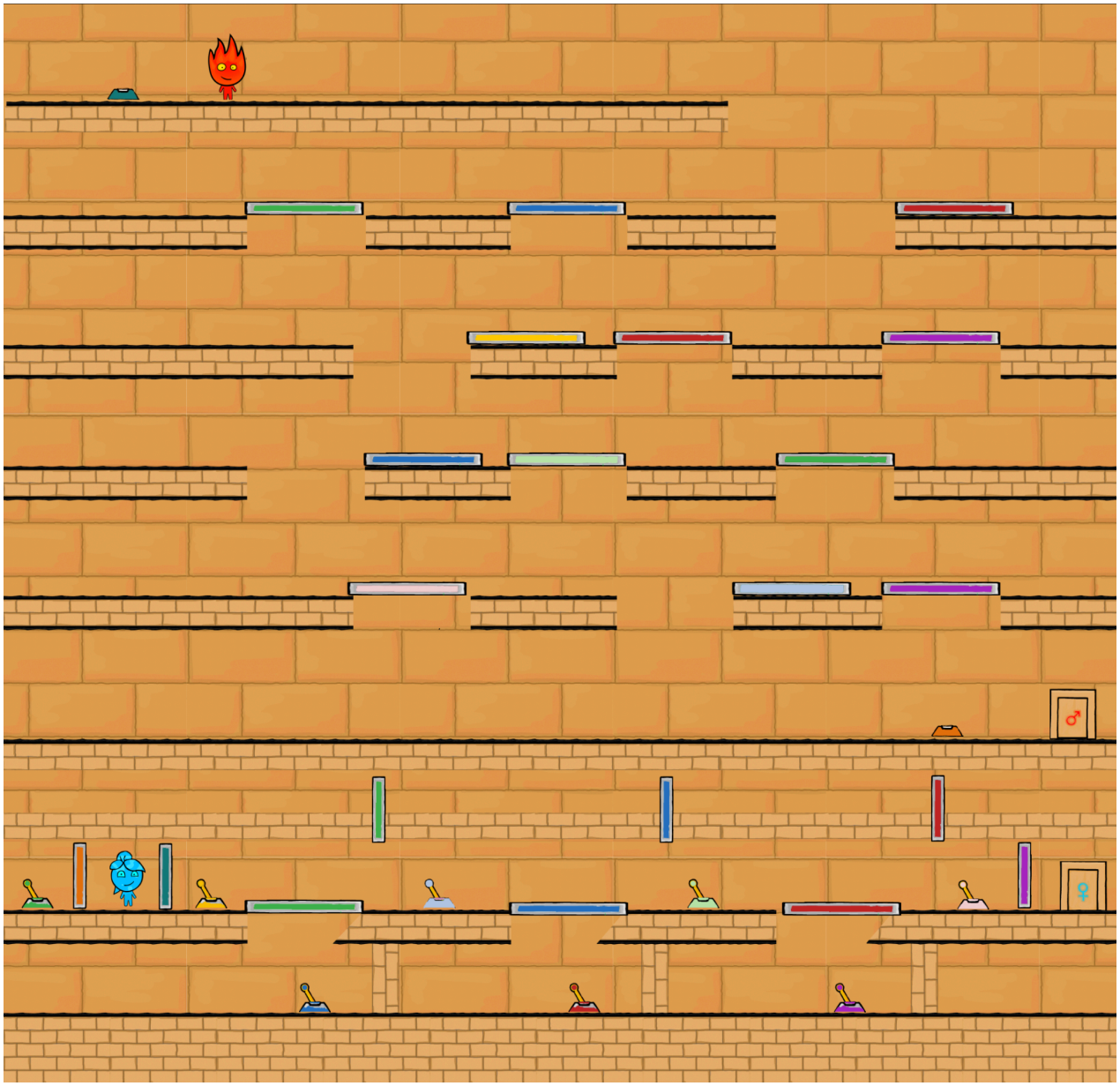

Figure 5: Modeling TQBF

PSPACE-Completeness

Since we have shown F&W to be PSPACE-Hard, we now need to establish its membership in PSPACE to demonstrate PSPACE-completeness.

Consider an r-by-c level with n different colors of doors. There are at most $2^n \cdot (r \cdot c)^2$ possible states for this level, where 2^n accounts for the states of each color of door/lever/button,

and $(r \cdot c)^2$ is an upper bound on the possible positions of both players. Therefore, each distinct state of the level can be encoded in $2n \cdot log_2(rc)$ bits. Hence F&W has membership in NPSPACE, since we can move Fireboy and Watergirl nondeterministically until they reach their respective exit, and these moves can be counted in $2n \cdot log_2(rc)$ bits if the level is solvable. By Savitch's theorem, since F&W is in NPSPACE, it also has membership in PSPACE. Since it is also PSPACE-hard, Fireboy & Watergirl is PSPACE-complete.