A Mixed-Integer Programming Model for TAS Optimization

ERGC Xander

May/June 2023

1 Introduction

This report derives a formulation and proves correctness of a Mixed-Integer, Quadratically-Constrained Program for Super Mario 64. The reader is assumed to have an understanding of SM64, convex programming, and computer programming.

Special thanks to Cryolite for helping develop and test the initial model.

2 Constraining Air Movement

The ideal function for modelling as a convex program is `update_air_without_turn(...)` (Appendix A) because Mario's yaw is fixed. The goal will be to constrain all cases of air movement in order to handle complex situations. This will be done by slowly relaxing strong assumptions about the speed during a segment of air travel from frame f_0 to frame f_n.

2.1 Joystick Input

Within the code, input is handled as pair of (`f32 intendedMag`, `s16 intendedDYaw`). In order to create non-trigonometric constraints, variables for the vertical and horizontal components of `intendedDYaw` on frame f can be defined by relaxing `intendedDYaw` to a real-valued angle:

$$\begin{aligned} a_f &:= \sin(\texttt{intendedDYaw}) \\ b_f &:= \cos(\texttt{intendedDYaw}) \\ a_f^2 + b_f^2 &= 1 \end{aligned} \tag{1.1}$$

However, to make this constraint convex, it needs to be relaxed to an inequality. Since a_f and b_f contribute directly to position and speed, objective functions that aim to maximize travel distance or speed will likely result in equality being held. Thus, such a relaxation is reasonable. Also, due to floating point rounding errors, there are some entries in the game's trig table such that `sins(x)*sins(x) + coss(x)*coss(x) > 1` (where `sins`, `coss` are the sine and cosine lookup functions for a signed short `x`). So, for the program to be feasible, a small adjustment factor, $\epsilon_0 > 0$, needs to be introduced:

$$a_f^2 + b_f^2 \leq 1 + \epsilon_0 \tag{1.2}$$

Additionally, since `intendedMag` is multiplied by both a_f and b_f when determining speed, it must be assumed constant in order to create convex constraints. Thus, this model works under the following primary assumption:

$$\texttt{intendedMag} = 1 \tag{A1}$$

As seen in the joystick update code (Appendix B), the maximum value of `controller->stickMag` is 64. So, `m->intendedMag` has a maximum value of 32 (when not squished) and consequently `intendedMag` has a maximum value of 1. Again, when optimizing for travel distance or speed, this assumption is reasonable.

2.2 Fast Forwards Movement

Depending on how fast Mario is moving, different amounts of drag are applied to adjust his speed. In the simplest case, Mario is moving sufficiently fast enough for consistent drag. Let $s_f \in \mathbb{R}$ be Mario's horizontal speed on frame f (i.e. `m->forwardVel` as of line 26 in Appendix A). Note that since s_f is stored as a 32-bit floating point number in the game, this is a relaxation of its value. Also, let $D_1 \in \{32, 48\}$ be the constant `dragTrheshold` based on Mario's action (line 8). Assume:

$$s_f > D_1 \tag{A2}$$

Then, since $s_f \geq D_1 > 0.35$, `aproach_f32(...)` (line 9) will reduce the speed by 0.35 (see Appendix C), and the fast drag threshold (lines 20-22 in Appendix A) will subtract 1 as well, leading to the following speed change:

$$s_f = s_{f-1} + 1.5b_f - 1.35 \tag{2.1}$$

After the speed is updated, Mario's horizontal velocity is updated, producing a change in position. First, note that since Mario's yaw (`m->faceAngle[1]`) is fixed, the following values are constant

$$C_0 := \texttt{sins(m->faceAngle[1])}$$
$$C_1 := \texttt{10.0f * sins(m->faceAngle[1] + 0x4000)}$$
$$C_2 := \texttt{coss(m->faceAngle[1])}$$
$$C_3 := \texttt{10.0f * coss(m->faceAngle[1] + 0x4000)}$$

Let $x_f, z_f \in \mathbb{R}$ be Mario's x and z coordinates on frame f (as with s_f, this is a relaxation). By the end of each frame, the velocities are added to the previous positions, resulting in the following constraints:

$$x_f = x_{f-1} + C_0 s_f + C_1 a_f \tag{3}$$
$$z_f = z_{f-1} + C_2 s_f + C_3 a_f \tag{4}$$

Constraints (1.2), (2.1), (3), (4) are sufficient for modelling air movement under (A2). This is applicable to scenarios such as freefall after sliding or a fast dive recover.

2.3 Fast Drag Threshold

Relax (A2) to consider crossing the drag threshold:

$$s_f \geq 0.35 \tag{A3}$$

For convenience, let $s'_f \in \mathbb{R}$ be Mario's horizontal speed before fast drag is applied (i.e. `m->forwardVel` as of line 18 in Appendix A). Under (A3):

$$s'_f = s_{f-1} + 1.5b_f - 0.35 \tag{5.1}$$

$$s_f = \begin{cases} s'_f - 1 & \text{if } s'_f > D_1 \\ s'_f & \text{otherwise} \end{cases}$$

Let $d_f \in \{0, 1\}$ be an indicator variable for whether the fast drag threshold is surpassed or not on frame f. That is, $d_f = 1 \iff s'_f > D_1$. To reflect the desired result of the indicator variable, update the speed constraint:

$$s_f = s'_f - d_f \tag{2.2}$$

To ensure this indicator variable behaves as intended, define an upper bound

$$M := s_{f_0} + 1.15n + 1 \geq s'_f \geq s_f$$

Which is based on the maximum acceleration of 1.15 units per frame and a given initial speed s_{f_0}. Also, let $\epsilon_1 > 0$ be the smallest 32-bit float such that $D_1 + \epsilon_1 > D_1$. Then add the following constraints:

$$(D_1 + \epsilon_1)d_f \leq s'_f \tag{6.1}$$

$$D_1 + Md_f \geq s'_f \tag{7.1}$$

Suppose $s'_f \leq D_1$. If $d_f = 1$, then by (6.1) $s'_f \geq D_1 + \epsilon_1 > D_1$. This is a contradiction, so $d_f = 0$ and the constraints become $0 \leq s'_f \leq D_1$ which are satisfied. Instead suppose $s'_f > D_1$. If $d_f = 0$, then by (7.1) $s'_f \leq D_1$. This is a contradiction, so $d_f = 1$ and the constraints become $D_1 + M \geq s'_f \geq D_1 + \epsilon_1$ which is satisfied (since the lower bound is equivalent to $s'_f > D_1$). Thus, the indicator behaves as intended.

Constraints (1.2), (2.2), (3), (4), (5.1), (6.1), (7.1) are sufficient for modelling air movement under (A3), which applies to initially accelerating situations such as a first long jump.

2.4 Backwards Movement

Next, consider relaxing (A3) to allow for negative speed, while still ignoring the lower bound drag and low speed drag:

$$-16 \leq s_f \leq -0.35 \qquad \text{or} \qquad 0.35 \leq s_f \tag{A4}$$

Under (A4), `aproach_f32(...)` either adds or subtracts 0.35 from the previous speed:

$$s'_f = \begin{cases} s_{f-1} + 1.5b_f - 0.35 & \text{if } 0.35 \leq s_{f-1} \\ s_{f-1} + 1.5b_f + 0.35 & \text{if } -16 \leq s_{f-1} \leq -0.35 \end{cases}$$

Let $r_f \in \{0, 1\}$ be an indicator variable for the direction of the speed on frame f. That is, $s_f > 0 \Rightarrow r_f = 1$ and $s_f < 0 \Rightarrow r_f = 0$. The desired effect can be achieved with:

$$s'_f = s_{f-1} + 1.5b_f - 0.7r_{f-1} + 0.35 \tag{5.2}$$

If the previous speed was positive ($r_{f-1} = 1$) then 0.35 is subtracted, and if the previous speed was negative ($r_{f-1} = 0$), then 0.35 is added. To ensure the indicator behaves as intended, modify the previous bounding constraints:

$$(D_1 + \epsilon_1)d_f - 16(1 - r_f) \leq s'_f \tag{6.2}$$

$$D_1 r_f + M d_f \geq s'_f \tag{7.2}$$

Also, for simplicity of analysis, only allow fast drag to be applied when moving forwards:

$$d_f \leq r_f \tag{8}$$

Suppose $s_f > 0$ and assume $r_f = 0$. Then, $d_f = 0$ by (8). It follows from (7.2) and (2.2) that $0 \geq s'_f = s_f$. This is a contradiction, so $r_f = 1$. Observe that (6.2) and (7.2) then reduce to (6.1) and (7.1) so the fast drag analysis holds as before.

Instead suppose $-16 \leq s_f < 0$ and assume $r_f = 1$. Then

$$\begin{aligned} s_f &= s'_f - d_f && \text{by (2.2)} \\ &\geq (D_1 + \epsilon_1)d_f - d_f && \text{by (6.2)} \\ &= (D_1 + \epsilon_1 - 1)d_f \\ &\geq 0 && \text{since } D_1 \in \{32, 48\},\ d_f \in \{0, 1\} \end{aligned}$$

This is a contradiction, so $r_f = 0$. Thus, $d_f = 0$ by (8), and subsequently (6.2) and (7.2) reduce to $-16 \leq s'_f \leq 0$ which is satisfied. Therefore, the direction indicator behaves according to the speed as required.

These constraint adjustments aren't applicable to any new non-trivial scenarios, but they lay the foundation for further complexity. Also, note that if $s_f = 0$, then $r_f \in \{0, 1\}$. This case will be handled later.

2.5 Lower Bound Drag

Now, instead of assuming $s_f \geq -16$, the drag for speed below this threshold (lines 23-25 in Appendix A) will be handled accordingly. Further relax (A4) into:

$$|s_f| \geq 0.35 \tag{A5}$$

The speed update equation becomes:

$$s_f = \begin{cases} s'_f + 2 & \text{if } s'_f < -16 \\ s'_f - d_f & \text{otherwise} \end{cases}$$

Let $l_f \in \{0, 1\}$ be an indicator variable for whether the lower bound drag is applied or not on frame f. That is, $l_f = 1 \iff s'_f < -16$. The new speed adjustment can be achieved in the

same way as the fast drag:

$$s_f = s'_f - d_f + 2l_f \tag{2.3}$$

$$l_f \leq 1 - r_f \tag{9}$$

If $l_f = 1$, then $r_f = 0$ by (9) and $d_f = 0$ by (8). Thus, $s_f = s'_f + 2$ by (2.3). If instead $l_f = 0$, then $s_f = s'_f - d_f$ by (2.3). So, the speed is modified as required. To ensure the indicator behaves as expected, define a trivial lower bound on speed:

$$L := \min\{s_{f_0},\ -16\}$$

Since the drag applied to Mario with sufficient negative speed is greater than his backwards acceleration, the lowest possible speed will either be Mario's speed on the first frame of movement, or the drag threshold itself. Also, let $\epsilon_2 > 0$ be the smallest 32-bit float such that $-16 - \epsilon_2 < -16$. Adjust the bounding constraints as follows:

$$(D_1 + \epsilon_1)d_f - 16(1 - r_f) + Ll_f \leq s'_f \tag{6.3}$$

$$D_1 r_f + Md_f + (-16 - \epsilon_2)l_f \geq s'_f \tag{7.3}$$

It will be proven simultaneously that $l_f = 1 \iff s'_f < -16$ and $s_f \geq L$ and r_f still behaves as intended. Note that the base case ($s_{f_0} \geq L$ and r_{f_0} is correct) is given trivially. Assume for induction that $s_{f-1} \geq L$ and r_{f-1} is correct.

Suppose $s'_f < -16$ and assume $l_f = 0$. Then,

$$\begin{aligned}
s'_f &\geq (D_1 + \epsilon_1)d_f - 16(1 - r_f) && \text{by (6.3)} \\
&\geq -16(1 - r_f) && \text{since } d_f \in \{0, 1\},\ D_1 + \epsilon_1 > 0 \\
&\geq -16 && \text{since } r_f \in \{0, 1\}
\end{aligned}$$

This is a contradiction, so $l_f = 1$. Then, $r_f = 0$ by (9) (which is correct) and $d_f = 0$ by (8). Thus, (7.3) reduces to $s'_f \leq -16 - \epsilon_2$ which is satisfied because it is equivalent to $s'_f < -16$, and (6.3) reduces to $s'_f \geq L - 16$. Observe that this is also satisfied:

$$\begin{aligned}
s'_f &= s_{f-1} + 1.5b_f - 0.7r_{f-1} + 0.35 && \text{by (5.2)} \\
&\geq L + 1.5b_f - 0.7r_{f-1} + 0.35 && \text{by inductive hypothesis} \\
&\geq L + 1.5bf - 0.35 && \text{since } r_{f-1} \in \{0, 1\} \\
&\geq L + 1.5(-\sqrt{1 + \epsilon_0}) - 0.35 && \text{by (1.2)} \\
&\geq L - 16 && \text{since } -16 < -0.35 - 1.5\sqrt{1 + \epsilon_0}
\end{aligned}$$

Additionally, note that

$$\begin{aligned}
s_f &= s'_f + 2 && \text{since } d_f = 0, l_f = 1 \\
&\geq (L - 1.5\sqrt{1 + \epsilon_0} - 0.35) + 2 && \text{from the work above} \\
&\geq L && \text{since } 1.65 \geq 1.5\sqrt{1 + \epsilon_0}
\end{aligned}$$

Instead suppose that $s'_f \geq -16$ and assume $l_f = 1$. Then, $r_f = d_f = 0$ by (8) and (9), and then $s'_f \leq -16 - \epsilon_2$ by (7.3). This is a contradiction, so $l_f = 0$. Then, (6.3), (7.3) reduce to (6.2), (7.2) and the analysis holds as before. Therefore, the lower bound drag indicator and direction indicator behave as intended, and speed is correctly bounded below by L.

Constraints (1.2), (2.3), (3), (4), (5.2), (6.3), (7.3), (8), (9) correctly model the lower bound drag threshold. So, these could be used to simulate fast backwards movement.

2.6 Slow Speed Drag

Finally, drop (A5) completely, and instead account for all types of drag. Consider the true effect of `approach_f32(...)` (Appendix C):

$$s'_f = \begin{cases} s_{f-1} + 1.5b_f - 0.7r_{f-1} + 0.35 & \text{if } |s_{f-1}| \geq 0.35 \\ 1.5b_f & \text{if } -0.35 < s_{f-1} < 0.35 \end{cases}$$

Note that when $s_{f-1} = 0.35$, either choice produces the same value, so this case can be ignored in later analysis. Let $u_f \in \{0, 1\}$ be an indicator variable for whether low speed drag is used or not. That is, $u_f = 1 \iff |s_{f-1}| < 0.35$. Since the desired effect has a variable amount of drag (cancelling off s_{f-1}), equality (5.3) must be split into inequalities that can be arbitrarily satisfied:

$$3L(1 - u_f) + 1.5b_f \leq s'_f \tag{10.1}$$
$$2M(1 - u_f) + 1.5b_f \geq s'_f \tag{11.1}$$
$$(2L - M)u_f + s_{f-1} + 1.5b_f - 0.7r_{f-1} + 0.35 \leq s'_f \tag{12.1}$$
$$(M - 2L)u_f + s_{f-1} + 1.5b_f - 0.7r_{f-1} + 0.35 \geq s'_f \tag{13.1}$$

First, note that $M \geq s_{f-1} + 1.5b_f - 0.7r_{f-1} + 0.35 \geq L - 16 \geq 2L$ by previous analysis and definition of L. Also, redefine M (if necessary) such that $-M \leq 1.5b_f \leq -L$ trivially.

If $u_f = 1$, then (10.1) & (11.1) reduce to $1.5b_f \leq s'_f \leq 1.5b_f$ (i.e. $s'_f = 1.5b_f$). Consider the lower bound produced by (12.1): $2L - M + s_{f-1} + 1.5b_f - 0.7r_{f-1} + 0.35 \leq s'_f$. Since $s_{f-1} + 1.5b_f - 0.7r_{f-1} + 0.35 \leq M$, this produces a greatest lower bound of $2L - M + M = 2L$ which is trivially satisfied. Similarly, since $s_{f-1} + 1.5b_f - 0.7r_{f-1} + 0.35 \geq 2L$, the least upper bound produced by (13.1) is $s'_f \leq M$ which is also trivially satisfied.

If $u_f = 0$, then (12.1) & (13.1) reduce to $s'_f = s_{f-1} + 1.5b_f - 0.7r_{f-1} + 0.35$. Since $-M \leq 1.5b_f \leq -L$, the greatest lower bound produced by (10.1) is $3L - L = 2L \leq s'_f$, and the least upper bound produced by (11.1) is $2M - M = M \geq s'_f$ which are both trivially satisfied.

Thus, speed is correctly changed with u_f. To ensure that the low speed drag indicator works as intended ($u_f = 1 \iff |s_{f-1}| < 0.35$), add the following constraints:

$$0.35r_{f-1} + L(1 - r_{f-1}) - 0.7u_f \leq s_{f-1} \tag{14}$$
$$Mr_{f-1} - 0.35(1 - r_{f-1}) + 0.7u_f \geq s_{f-1} \tag{15}$$
$$L(1 - u_f) - 0.35u_f \leq s_{f-1} \tag{16}$$
$$M(1 - u_f) + 0.35u_f \geq s_{f-1} \tag{17}$$

Recall that $s_f > 0 \Rightarrow r_f = 1$ and $s_f < 0 \Rightarrow r_f = 0$ and $s_f = 0 \Rightarrow r_f \in \{0, 1\}$. This creates four cases to check for the indicator's behaviour.

Case 1: $0 \leq s_{f-1} < 0.35$, $r_{f-1} = 1$
Assume $u_f = 0$. Then, $0.35 \leq s_{f-1}$ by (14). This is a contradiction, so $u_f = 1$. Then, (14) & (15) become $-0.35 \leq s_{f-1} \leq M + 0.7$, and (16) & (17) become $-0.35 \leq s_{f-1} \leq 0.35$ which are both satisfied in this case.

Case 2: $-0.35 < s_{f-1} \leq 0$, $r_{f-1} = 0$
Assume $u_f = 0$. Then, $-0.35 \geq s_{f-1}$ by (15). This is a contradiction, so $u_f = 1$. Then, (14) & (15) become $L - 0.7 \leq s_{f-1} \leq 0.35$ which is satisfied in this case. Also, as in Case 1, (16) & (17) are satisfied.

Case 3: $0.35 < s_{f-1}$, $r_{f-1} = 1$
Assume $u_f = 1$. Then, $0.35 \geq s_{f-1}$ by (17), which is a contradiction. So, $u_f = 0$. Then, (14) & (15) become $0.35 \leq s_{f-1} \leq M$ and (16) & (17) become $L \leq s_{f-1} \leq M$ which are both satisfied in this case.

Case 4: $s_{f-1} < -0.35$, $r_{f-1} = 0$
Assume $u_f = 1$. Then, $-0.35 \leq s_{f-1}$ by (16), which is a contradiction. So, $u_f = 0$. Then, (14) & (15) become $L \leq s_{f-1} \leq -0.35$ which is satisfied in this case. Also, as in Case 3, (16) & (17) are satisfied.

Now, with this low speed drag, Mario's speed might increase from $s_{f-1} = -0.35$ to $s_f = 1.5$ in one frame. So, the upper bound must be updated to $M := s_{f_0} + 1.85n + 1$.

2.7 Diving

When Mario is in a single jump, double jump, steep jump, triple jump, or freefall, he can dive to gain a boost of speed. Let $v_f \in \{0, 1\}$ represent whether Mario dives on frame f or not. Firstly, under normal circumstances, Mario can only dive once:

$$v_{f_0} + \cdots + v_{f_n} = \sum_{f=f_0}^{f_n} v_f \leq 1 \tag{18}$$

So, if Mario dives on any two frames, then constraint (18) is violated. Notice that this allows for choosing to not dive at all.

If Mario is in a single jump or double jump, he requires more than 28 speed to dive (see Appendix D for relevant code). If Mario is in any other action that allows a dive, he can dive at any speed. However, due to complications with drag[1], this model will assume these actions require more than 0.35 speed. In actions where Mario cannot dive, the speed requirement will be set infeasibly high. Let $D_2 \in \{0.35, 28, M\}$ be the threshold speed for diving, and let ϵ_3 be the smallest 32-bit float such that $D_2 + \epsilon_3 > D_2$,

$$(D_2 + \epsilon_3)v_f + L(1 - v_f) \leq s_{f-1} \tag{19.1}$$

On the frame Mario dives, $v_f = 1$, and (19.1) becomes $D_2 + \epsilon_3 \leq s_{f-1}$ which is equivalent to $s_{f-1} > D_2$. On any other frame, $v_f = 0$ and this becomes $L \leq s_{f-1}$ which is satisfied from before.

[1] An example scenario that is possible in game is $s_0 = -16$, $s_1 = -14.5$, $l_1 = 1$ (so $r_1 = 0$ by (9)), $b_2 = 0$, $v_2 = 1$, $s_2 = 0.15$. Since the dive on frame 2 increases Mario's speed above 0.35, the basic drag should subtract 0.35 when calculating s'_2. In the current model, this requires $r_1 = 1$. This issue can likely be fixed by introducing a dive specific drag indicator. Note that M would need to be updated too because Mario's speed could increase from -15.35 to 1.5 in a single frame.

Next, consider that the dive will give a bonus 15 speed and caps the speed at 48 before calling the air update function. So, when accounting for basic drag, on the frame that Mario dives:

$$s'_f = \begin{cases} 1.5b_f + 47.65 & \text{if } s_f + 15 > 48 \\ s_{f-1} + 1.5b_f + 14.65 & \text{otherwise} \end{cases}$$

Due to the bonus speed, update the upper bound $M := s_{f_0} + 1.85(n-1) + 16$. Now, let $w_f \in \{0, 1\}$ indicate whether the upper bound on dive speed is used or not. That is, $w_f = 1 \iff s_{f-1} + 15 > 48$, or equivalently $s_{f-1} > 33$. Let ϵ_4 be the smallest 32-bit float such that $33 + \epsilon_4 > 33$. Enforce the behaviour of w_f with the following constraints:

$$(D_2 + \epsilon_3)(v_f - w_f) + L(1 - v_f) + (33 + \epsilon_4)w_f \leq s_{f-1} \tag{19.2}$$

$$33v_f + M(1 - v_f) + Mw_f \geq s_{f-1} \tag{20}$$

$$w_f \leq v_f \tag{21}$$

Equation (21) guarantees the bound on dive speed can only apply when diving, that is, if $v_f = 0$, then $w_f = 0$. Observe that if Mario is not diving, i.e. $v_f = 0$, then (19.2) & (20) reduce to $L \leq s_{f-1} \leq M$ which is satisfied.

Consider when $v_f = 1$. If $w_f = 1$, then $s_{f-1} \geq (33 + \epsilon_4) > 33$ by (19.2) and (20) becomes $33 + M \geq s_{f-1}$ which is satisfied. If $w_f = 0$, then $33 \geq s_{f-1}$ by (20) and $s_{f-1} \geq D_2 + \epsilon_3 > D_2$ by (19.2). So, this indicator behaves as expected, and the minimum dive speed is still enforced.

Now, the desired effect of v_f and w_f can be achieved by modifying the speed constraints:

$$3L(1 - u_f - w_f) + 1.5b_f + 47.65w_f \leq s'_f \tag{10.2}$$

$$2M(1 - u_f - w_f) + 1.5b_f + 47.65w_f \geq s'_f \tag{11.2}$$

$$(2L - M)(u_f + w_f) + s_{f-1} + 1.5b_f - 0.7r_{f-1} + 15v_f + 0.35 \leq s'_f \tag{12.2}$$

$$(M - 2L)(u_f + w_f) + s_{f-1} + 1.5b_f - 0.7r_{f-1} + 15v_f + 0.35 \geq s'_f \tag{13.2}$$

Observe that if $v_f = 0$, then $w_f = 0$ by (21), and (10.2), (11.2), (12.2), (13.2) reduce to (10.1), (11.1), (12.1), (13.1) so the analysis holds as before.

Note the bounds on the expression below:

$$\begin{aligned} M &\geq s_{f-1} + 1.5b_f - 0.7r_{f-1} + 15v_f + 0.35 && \text{by definition of } M \\ &\geq s_{f-1} + 1.5b_f - 0.7r_{f-1} + 0.35 && \text{since } v_f \in \{0, 1\} \\ &\geq 2L && \text{shown previously} \end{aligned}$$

If $v_f = 1$, then $s_{f-1} > D_2 \geq 0.35$ by (19.2). So, $u_f = 0$ and $r_{f-1} = 1$ by previous analysis. If $w_f = 1$, then $s'_f = 1.5b_f + 47.65$ by (10.2) & (11.2). Also, in the same way as the previous analysis of these speed constraints, (12.2) & (13.2) produce a greatest lower bound and least upper bound of $2L \leq s'_f \leq M$ which is satisfied. If instead $w_f = 0$, then $s'_f = s_{f-1} + 1.5b_f + 14.65$ by (12.2) & (13.2) and (10.2) & (11.2) produce a trivial greatest lower bound and least upper bound as well. Therefore, the dive indicators provide the desired speed boost.

3 Fully Constrained Air Movement Program

To summarize the program for modelling air movement without turning, in its entirety, assuming the joystick inputs are at full magnitude:

Let s_f, x_f, $z_f \in \mathbb{R}$ be Mario's horizontal speed, x-coordinate, and z-coordinate on frame f respectively. Let $s'_f \in \mathbb{R}$ be the horizontal speed before considering fast speed and lower bound drag. Let a_f, $b_f \in \mathbb{R}$ be the perpendicular and parallel components of the intended difference in yaw on frame f.

Let d_f, r_f, l_f, u_f, v_f, $w_f \in \{0, 1\}$ be the respective indicator variables for applying fast drag, direction of speed, applying lower bound drag, applying slow speed drag, diving, and applying the dive speed bound.

Let f_0, $f_n \in \mathbb{N}$ be the given first and last frames of air movement. Let s_{f_0}, x_{f_0}, $z_{f_0} \in \mathbb{R}$ be the given initial speed and position. Let $L := \min\{s_{f_0}, -16\}$ be a trivial lower bound on speed. Let $M := s_{f_0} + 1.85(n-1) + 16$ be a trivial upper bound on speed. Let $D_1 \in \{32, 48\}$ be the constant `dragTrheshold` based on Mario's action (48 in a long jump, 32 otherwise). Let $D_2 \in \{0.35, 28, M\}$ be the threshold speed for diving based on Mario's action (28 in a single jump or double jump, 0.35 in a steep jump, triple jump or freefall, M otherwise).

Let ϵ_0, ϵ_1, ϵ_2, ϵ_3, $\epsilon_4 > 0$ be the smallest 32-bit floating point numbers such that $D_1 + \epsilon_1 > D_1$, $-16 - \epsilon_2 < -16$, $D_2 + \epsilon_3 > D_2$, $33 + \epsilon_4 > 33$ and `sins(x)*sins(x) + coss(x)*coss(x) <= 1 + `ϵ_0. Also, define

$$C_0 := \texttt{sins(m->faceAngle[1])}$$
$$C_1 := \texttt{10.0f * sins(m->faceAngle[1] + 0x4000)}$$
$$C_2 := \texttt{coss(m->faceAngle[1])}$$
$$C_3 := \texttt{10.0f * coss(m->faceAngle[1] + 0x4000)}$$

Let $f(s, x, z)$ be a given convex objective function. The program is as follows:

minimize $f(s, x, z)$ subject to

$$a_f^2 + b_f^2 \leq 1 + \epsilon_0 \tag{1.2}$$
$$s_f = s'_f - d_f + 2l_f \tag{2.3}$$
$$x_f = x_{f-1} + C_0 s_f + C_1 a_f \tag{3}$$
$$z_f = z_{f-1} + C_2 s_f + C_3 a_f \tag{4}$$
$$s'_f \geq (D_1 + \epsilon_1) d_f - 16(1 - r_f) + L l_f \tag{6.3}$$
$$s'_f \leq D_1 r_f + M d_f + (-16 - \epsilon_2) l_f \tag{7.3}$$
$$d_f \leq r_f \tag{8}$$
$$l_f \leq 1 - r_f \tag{9}$$
$$s'_f \geq 3L(1 - u_f - w_f) + 1.5 b_f + 47.65 w_f \tag{10.2}$$
$$s'_f \leq 2M(1 - u_f - w_f) + 1.5 b_f + 47.65 w_f \tag{11.2}$$
$$s'_f \geq (2L - M)(u_f + w_f) + s_{f-1} + 1.5 b_f - 0.7 r_{f-1} + 15 v_f + 0.35 \tag{12.2}$$
$$s'_f \leq (M - 2L)(u_f + w_f) + s_{f-1} + 1.5 b_f - 0.7 r_{f-1} + 15 v_f + 0.35 \tag{13.2}$$
$$s_{f-1} \geq 0.35 r_{f-1} + L(1 - r_{f-1}) - 0.7 u_f \tag{14}$$
$$s_{f-1} \leq M r_{f-1} - 0.35(1 - r_{f-1}) + 0.7 u_f \tag{15}$$
$$s_{f-1} \geq L(1 - u_f) - 0.35 u_f \tag{16}$$
$$s_{f-1} \leq M(1 - u_f) + 0.35 u_f \tag{17}$$
$$1 \geq v_{f_0} + \cdots + v_{f_n} \tag{18}$$
$$s_{f-1} \geq (D_2 + \epsilon_3)(v_f - w_f) + L(1 - v_f) + (33 + \epsilon_4) w_f \tag{19.2}$$
$$s_{f-1} \leq 33 v_f + M(1 - v_f) + M w_f \tag{20}$$
$$w_f \leq v_f \tag{21}$$

4 Implementation Details & Extending the Model

The equations in the previous section define one particular model, however, slight modifications may have more applications. Also, it is non-trivial to solve such a program. So, this section provides some brief comments on additional information needed to use this model.

4.1 Choice of Objective Function

Good objective functions are often problem-specific, but some are generally applicable. One example is to maximize speed on the final frame which can be done with

$$f(s, x, z) = -s_{f_n}$$

Another is to get as close as possible to a goal point $(x', z') \in \mathbb{R}^2$, which can be done by minimizing the squared distance with

$$f(s, x, z) = (x_{f_n} - x')^2 + (z_{f_n} - z')^2$$

Or, both can be combined by selecting a parameter, $\alpha \in \mathbb{R}$, $0 \leq \alpha \leq 1$, representing how important minimizing point distance is over maximizing speed ($\alpha = 1$ means only distance

matters, $\alpha = 0$ means only speed matters)

$$f(s, x, z) = \alpha((x_{f_n} - x')^2 + (z_{f_n} - z')^2) - (1 - \alpha)s_{f_n}$$

4.2 Choice of Integer Program Solver

Despite the fact that solving 0-1 Integer Programs is NP-Hard, good, efficient approximation algorithms exist. In an implementation using Mosek, the resulting solution is optimal to beyond 32-bit floating point accuracy [2].

Also, the choice of solver is important for finding any solution at all. This is because Integer Program feasibility itself is NP-Hard. So, weaker solvers may require initial feasible values for every variable. This can still be achievable when a TAS already exists and the goal is to optimize some extra units of distance or speed. This is not an issue for the most basic model presented in section 2.2 because that model doesn't include any integer variables. In that case, CVXPY's [3] default solver can quickly find an optimal solution without an initial feasible solution.

4.3 Choice of Epsilon

Abusing the fact that the fast drag only applies when $s'_f \geq D_1 + \epsilon_1$ is called the ".99 trick" because aiming for 31.99 or 47.99 speed avoids the additional drag for as long as possible (usually maximizing speed on subsequent frames). By nature of a precise model, it will likely output speeds so close to the threshold that they are not achievable in practice. Especially when considering that the model assumes , due to the loss of precision when turning the model's output into joystick inputs for the game, So, it may be beneficial to consider applying drag in the model when $s'_f \geq D_1 - \epsilon_1$ instead, and adjust the constraints accordingly:

$$D_1 d_f + (-16 + \epsilon_2)(1 - r_f) + L l_f \leq s'_f \tag{6.3'}$$

$$(D_1 - \epsilon_1) r_f + M d_f + (-16 - \epsilon_2) l_f \geq s'_f \tag{7.3'}$$

This same idea can be applied to the lower bound drag threshold as well. Also, instead of choosing the smallest constant, a bigger constant like $\epsilon_1 = 10^{-4}$ can be used.

4.4 Variable Starting Conditions

Any combination of $x_{f_0}, z_{f_0}, s_{f_0}$ can be left as variables in order to solve for ideal starting conditions. Though (importantly) sufficient additional constraints are needed to prevent unbounded programs (i.e. by arbitrarily starting really far away with very high speed).

[2] See details on Mosek's Mixed-Integer Program Optimizer
[3] See CVXPY

4.5 Trig Table Calculations

The game stores `m->faceAngle[1]` as a signed short (-32768 to 32767). However, most RAM viewing tools display it as an unsigned short (0 to 65535). To Calculate the C constants, reference the trig table directly[4] or the Wafel[5] module in python can be used:

```
import wafel
game = wafel.Game("C:\\...\\sm64_us.dll")
u16 = lambda x: (x + 0x10000) % 0x10000
sins = lambda x: game.read(f"gSineTable[{u16(x) >> 4}]")
coss = lambda x: game.read(f"gSineTable[{(u16(x) >> 4) + 0x400}]")
yaw = ... # m->faceAngle[1] as an unsigned short
C0 = sins(yaw)
C1 = 10 * sins(yaw + 0x4000)
C2 = coss(yaw)
C3 = 10 * coss(yaw + 0x4000)
```

4.6 Additional Constraints

If Mario's position needs to be within a specific region on frame f', then extra constraints can be added as necessary. For example, $z_{f'} \geq -100$. If a specific quarter frame $q \in \{1, 2, 3\}$ after frame f' is required to be in a specific region, say $x_{(f'+\frac{q}{4})} \leq x'$, then by using the fact that Mario's x velocity on frame f' is $(C_0 s_f + C_1 a_f)$, the following constraint will suffice

$$x_f + \left(\frac{q}{4}\right)(C_0 s_{f+1} + C_1 a_{f+1}) \leq x'$$

If Mario needs to kick on frame f', then the constraints $s_{f'} \leq 28$ and $\forall f$, $v_f = 0$ must be included.

If Mario lands for one frame, and then double jumps on frame f', this can be treated as one continuous segment of air movement, where Mario's speed is multiplied by a factor of 0.8 due to the double jump:

$$\begin{aligned} v_{f'-1} &= 0 \\ s'_{f'} &\geq (2L - M)(u_{f'} + w_{f'}) + 0.8 s_{f'-1} + 1.5 b_{f'} - 0.7 r_{f'-1} + 15 v_{f'} + 0.35 \\ s'_{f'} &\leq (M - 2L)(u_{f'} + w_{f'}) + 0.8 s_{f'-1} + 1.5 b_{f'} - 0.7 r_{f'-1} + 15 v_{f'} + 0.35 \end{aligned}$$

Note however that this action transition happens before `update_air_without_turn(...)` is called, so this will have the same issue as diving where it cannot properly handle a transition that crosses the slow speed drag threshold. So, the output will only be valid if $s_{f'-1} \geq 0.4375$.

With some creativity, even more situations can likely be accounted for!

[4] See sm64/include/trig_tables.inc.c and for `sins`/`coss` definitions see sm64/src/engine/math_utils.h
[5] See Wafel library

5 Appendix

A Update Air Speed Code

```c
void update_air_without_turn(struct MarioState *m) {
    f32 sidewaysSpeed = 0.0f;
    f32 dragThreshold;
    s16 intendedDYaw;
    f32 intendedMag;

    if (!check_horizontal_wind(m)) {
        dragThreshold = m->action == ACT_LONG_JUMP ? 48.0f : 32.0f;
        m->forwardVel = approach_f32(m->forwardVel, 0.0f, 0.35f, 0.35f);

        if (m->input & INPUT_NONZERO_ANALOG) {
            intendedDYaw = m->intendedYaw - m->faceAngle[1];
            intendedMag = m->intendedMag / 32.0f;

            m->forwardVel += intendedMag * coss(intendedDYaw) * 1.5f;
            sidewaysSpeed = intendedMag * sins(intendedDYaw) * 10.0f;
        }

        //! Uncapped air speed. Net positive when moving forward.
        if (m->forwardVel > dragThreshold) {
            m->forwardVel -= 1.0f;
        }
        if (m->forwardVel < -16.0f) {
            m->forwardVel += 2.0f;
        }

        m->slideVelX = m->forwardVel * sins(m->faceAngle[1]);
        m->slideVelZ = m->forwardVel * coss(m->faceAngle[1]);

        m->slideVelX += sidewaysSpeed * sins(m->faceAngle[1] + 0x4000);
        m->slideVelZ += sidewaysSpeed * coss(m->faceAngle[1] + 0x4000);

        m->vel[0] = m->slideVelX;
        m->vel[2] = m->slideVelZ;
    }
}
```

B Joystick Update Code

```c
void adjust_analog_stick(struct Controller *controller) {
    //...
    controller->stickMag =
        sqrtf(controller->stickX * controller->stickX + controller->stickY *
    controller->stickY);

    // Magnitude cannot exceed 64.0f: if it does, modify the values
    if (controller->stickMag > 64) {
        controller->stickX *= 64 / controller->stickMag;
        controller->stickY *= 64 / controller->stickMag;
        controller->stickMag = 64;
    }
}

void update_mario_joystick_inputs(struct MarioState *m) {
```

```
    struct Controller *controller = m->controller;
    f32 mag = ((controller->stickMag / 64.0f) * (controller->stickMag / 64.0f)) *
    64.0f;

    if (m->squishTimer == 0) {
        m->intendedMag = mag / 2.0f;
    } else {
        m->intendedMag = mag / 8.0f;
    }

    if (m->intendedMag > 0.0f) {
        m->intendedYaw = atan2s(-controller->stickY, controller->stickX) + m->area
    ->camera->yaw;
        m->input |= INPUT_NONZERO_ANALOG;
    } else {
        m->intendedYaw = m->faceAngle[1];
    }
}
```

C Low Speed Drag Code

```
f32 approach_f32(f32 current, f32 target, f32 inc, f32 dec) {
    if (current < target) {
        current += inc;
        if (current > target) {
            current = target;
        }
    } else {
        current -= dec;
        if (current < target) {
            current = target;
        }
    }
    return current;
}
```

D Relevant Diving Code

```
s32 check_kick_or_dive_in_air(struct MarioState *m) {
    if (m->input & INPUT_B_PRESSED) {
        return set_mario_action(m, m->forwardVel > 28.0f ? ACT_DIVE :
    ACT_JUMP_KICK, 0);
    }
    return FALSE;
}

static u32 set_mario_action_airborne(struct MarioState *m, u32 action, u32
    actionArg) {
    switch (action) {
        case ACT_DIVE:
            if ((forwardVel = m->forwardVel + 15.0f) > 48.0f) {
                forwardVel = 48.0f;
            }
            mario_set_forward_vel(m, forwardVel);
            break;
    }
}
```