Our first submission here, we hope you will all enjoy it as much as we did while making it!

This is a double submission. We were unsure whether to include the Training levels or not, so we will let the community decide it. This is the version WITHOUT the Training levels, the submission with them is available [\_\_TODO: link to the other submission\_\_|here].

----

[*https://www.youtube.com/watch?v=w4gxOXmBLd8*](https://www.youtube.com/watch?v=w4gxOXmBLd8)

%%TOC%%

*\_\_TODO: better introduction\_\_*

''*Kuru Kuru Kururin*'' is an unusual game that is a mix of racing/puzzle genres. The player controls a slowly spinning stick called the Helirin, and must maneuver it through a series of mazes. The player controls the direction and speed of movement but not the Helirin's constant rotation.

This TAS is a significant improvement to the [1307M|2009 movie by Ryuto] due to game-changing discoveries: wall clipping and out-of-bounds goal zones.

**!!!Categories**

\*Emulator used: Bizhawk 2.3.1, mGBA core

\*Aims for fastest time

\*Takes damage to save time

\*Heavy glitch abuse

\*Genre: Puzzle

\*Genre: Racing

**!!!Rules**

\*Reach the credits as fast as possible

\*All glitches allowed

This means that we have to complete the thirty “main” levels of the game while saving the birds trapped in the third level of each world.

**!!!The basics**

**!!Objective of a level**

Reach the goal zone (=yellow zone).

When you hit a wall, you loose a heart. If you loose all your three hearts, the Helirin breaks and you must restart the level from the beginning. In some levels there are heart zones on your way that heal you.

Sometimes there is a collectible in the level, which is either a cosmetic bonus or a bird. Only birds are necessary in order to reach the credits.

**!!Position**

Position is given by a triple (X,Y,R) with X,Y coordinates and R the angle of the Helirin.

\*X and Y are both a couple (pixel,subpixel) with the subpixel in the range [0..255].

\*R is a single 16-bit value in the range [0..65535]. 0 and 32768 are vertical.

Axes:

― — > X

|

|

v

Y

**!!Speed under ”normal” behavior**

Speed depends on how many buttons A/B are held. The more buttons you hold, the faster you go.

The table below gives examples of a horizontal/vertical input and a diagonal input. Speed values are in (pixel,subpixel) format.

||Direction|| #A/B held || X speed || Y speed || Remark ||

| Right | 0 | +1,128 | 0 | / |

| Right | 1 | +2, 64 | 0 | / |

| Right | 2 | +3, 0 | 0 | / |

| Down-right| 0 |+1, 15/16 | +1, 15/16| 2-cycle |

| Down-right| 1 |+1,151/151/151/152|+1,151/151/151/152| 4-cycle |

| Down-right| 2 | +2, 31 | +2, 31 | / |

Holding buttons makes you respectively 1.5x and 2x faster. The cyclic behavior on diagonal speed values guarantees exactness every two or four frames. (Beware that the X-cycle and the Y-cycle can be desynced).

Rotation is at a constant +/-182 rate (+ is clockwise).

**!!Collision**

When you hit something, you get pushed back. It has an impact on all three parameters:

\*you get an X/Y bump speed that decreases with time (it takes them about 30 to 40 frames to get back to zero).

\*your rotation is set to +/-1024 and gradually gets back to its initial value (it takes about 15 frames). Sometimes, collision is messy and rotation is set with the wrong sign (meaning that rotation is sped up instead of being pushed back).

**!!Springs**

Some levels include blue springs. They are the only way to switch between clockwise and counterclockwise motion. They can also be used to speed up rotation.

**!!Moving objects**

Some levels include cylinders and spiked balls. They hurt. Their movement is cyclic and fixed: if you hit them, they will not deflect.

**!!Horns**

The best feature of the game. While in a level, press L or R to have fun. Holding A/B while pressing L or R gives other horn sound effects for a grand total of six.

## !!!Glitches and tricks used

**!!Invulnerability frames**

Whenever you take damage, the game gives you 20 frames of invulnerabilty. This allows to hit a wall multiple times and only loose one heart, which is especially useful to set up rotation and to perform wall “surfing” (detailed below).

\_\_TODO: gif?\_\_

**!!Wall clipping**

When colling with a wall, the game does not behave in the same way depending on which part of the Helirin is colliding. In particular if it is the center, then the game will only rely on your last input and push you in the oppposite direction. But simply going towards the wall would not work, as the game would push you in the opposite direction. So, in order to get the center into the wall, we typically use the bump speed from an earlier collision. This allows then to not make any input while still moving towards the wall. A little flaw in the game physics with huge consequences!

\_\_TODO: slow-mo gif\_\_

**!!Wall surfing**

Once the center of the Helirin is in a wall, as the game only takes into account yout last input and push you the other way, pressing inputs parallel to the wall makes you travel along the wall at very high speed.

\_\_TODO: gif\_\_

**!!Out-of-bounds stuff**

Because of the use of the way collisions are computed (see section “Technical details”), some objects are ''physically'' replicated: even though they are not visible, you can interact with them.

\*Replicated: walls and starting/heart/goal zones

\*Not replicated: birds, moving objects, springs, and cosmetic bonuses

Replicated goal zones are especially interesting. Some of them are closer to the starting zone than the regular one. This has been a major dicovery since they have been used in about half of the levels. See section “Times per level” for a recap of where they have been used.

**!!(No) restart trick**

The [1307M|2009 movie by Ryuto] restarted the game after completing the third level of each world. These levels induced an additional cutscene from saving a bird, which was skipped by restarting the game (saving 68 frames each time). Doing this after any other level would loose 24 frames instead.

However, it looks like this was only made possible because of using emulator VBA-rr 19.3, which skipped the GameBoy introduction. As Bizhawk 2.3.1 does not skip it by default and soft resetting is not a thing in this game, we did not include this trick in our movie.

## !!!Tools

**!!Memory watch**

A .wch file is available [https://github.com/E-Sh4rk/KururinTAS|here].

|| Name || Address (IWRAM)|| Size || Description ||

| X / Y | 0x4544 / 0x4548 | 32 bits | Position of the center of the helirin. If we only look at the 16 most significant bits, we get the position in pixels. |

| XB / YB | 0x454C / 0x4550 | 32 bits | Bump speed. A bump speed is applied when the helirin hits a wall, then it decreases gradually. |

| XS / YS | 0x4554 / 0x4558 | 32 bits | Input speed. It only depends on the direction pressed at the previous frame. |

| Angle | 0x4572 | 16 bits | Angle of the helirin, 0 and 32768 being vertical. 2⸢⸢16⸣⸣ corresponds to 360°. For instance: (90 / 360) \* 2⸢⸢16⸣⸣ = 16384 is 90°. |

| Angle Rate | 0x4574 | 16 bits | Rotation speed. It is 182 or -182 by default, but it momentarily changes when the helirin hits something. |

| Default Rate | 0x4576 | 16 bits | 182 when the helirin rotates clockwise, and -182 when it rotates counter-clockwise. Can change when hitting a spring. |

| Invulnerability | 0x4585 | 8 bits | Number of invulnerability frames left. Grows to 20 when the Helirin looses a heart, then decreases by 1 every frame until 0. This value is decremented before being used, so having it to 1 is equivalent to having it to 0. |

| MapW and MapH | 0x313C / 0x313E | 16 bits | Size of the map in a number of tiles. A tile is 8x8 so multiplying these numbers by 8 gives the size of the map in pixels. |

| Collision Mask | 0x45D4 | 32 bits | Indicates which parts of the helirin are in collision with something. |

**!!TAStudio**

As this game does not have any known desync or lag issue, progress was nonlinear and we just had to glue the segments together at the end. This has made TAStudio a very efficient and comfortable interface to work with.

**!!Speed HUD**

Lua script written by ThunderAxe31. Displays X/Y position, speed and bump speed. Available [=userfiles/info/49530161300647551|here].

\_\_TODO: add image\_\_

**!!OOB viewer**

Lua script created by ThunderAxe31, then improved in conjoint work with E-Sh4rk. Available on [https://github.com/E-Sh4rk/KururinTAS/tree/master/OoB%20Viewer|Github].

\_\_TODO: add image\_\_

**!!KuruBot**

Bot designed by E-Sh4rk. Efficient and customizable tool to help finding new strategies in Kuru Kuru Kururin’s levels (more details in the “Technical details” section). Available on [<https://github.com/E-Sh4rk/KururinTAS/tree/master/KuruBot>|Github].

\_\_TODO: add image\_\_

## !!!Technical details

**!!Game physics and map storage**

**!!OOB stuff**

Because of the use of the modulo operator in collision computation, some objects are ''physically'' replicated: even though they are not visible, you can interact with them. In particular:

\*Replicated: walls and starting/heart/goal zones

\*Not replicated: birds, moving objects, springs, and cosmetic bonuses

However, the way walls and zones are replicated is very different as you can see in these examples:

\_\_TODO: add Grasslands 1 and Machine Land 1 image examples\_\_

**!!KuruBot**

Why would anyone bother to recode the game physics when there is an emulator available? Well, simulating through Bizhawk would not be very efficient, as the search speed is bound to emulator speed. But we do not need most of what the emulator is computing (sound, graphics), all we need to know is the update on speed and position. Thus re-implementing the physics allows to keep computations at their minimum and thus explore dozens of thousands of states per second. This is KuruBot’s reason for existence.

KuruBot was implemented in C# and the exact physics of the game described above was replicated. Our take was to use a custom A\* algorithm. A\* relies on a cost map to guide the search: if we have a good guess on what an optimized path should look like, then we can prioritize the search on the corresponding area.

Unfortunately, even with the current most efficient shortest path algorithms, solving the problem exactly would be impossible (there are more than 2^100 reachable states). STATE REDUCTION

Keep in mind that due to the complexity of the problem KuruBot ''cannot'' solve this problem exactly. There is no guarantee that the path found is optimal.

The behavior and the level of approximation can be easily configured through some predefined configuration files:

\*We can choose a precise search, or a less precise but faster one.

\*We can choose a focused search or a more explorative one.

\*We can focus on specific paths, such as damageless paths, paths without OOB, etc.

With a relevant configuration file, the bot is able to solve “easy” levels on its own. For instance, in this TAS, `Grasslands 1` has been entirely computed by KuruBot.

For bigger levels, we had to plan the global itinerary and then use KuruBot on smaller segments.

Moving objects were not implemented since they actually rarely constituted a threat for fastest completion.

**!!(obsolete) Pixel and subpixel adjustment**

Before the “KuruBot era”, getting wall clips required precise setups, as bump speeds were rarely above a few dozen subpixels. As speed is always greater than one pixel/frame whatever inputs you do, it can be difficult to reach a precise range of positions. But it turns out that with help from diagonal inputs, it is possible to adjust precisely the Helirin position within a few frames.

Below are some examples. They were heavily used in the [\_\_TODO: link\_\_|v0 movie]. You can recognize some of them if you take a closer look at the inputs done right before some wall clips. Who knows, they might become useful again someday!

\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*

Notation:

-- X(pixel,subpixel) Y(pixel,subpixel)

-- "R2": R = direction in {U,L,D,R,UL,UR,DL,DR}

2 = speed in {0,1,2} (=number of held buttons in {A,B})

\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*

Methods to change X pixel with minor impact on X subpixel:

DL2+UL2 [[-4, 62]]

R1 [[+2, 64]]

R2 [[+3, 0]]

=> [[+1, 2]] using 4 frames

Note: with reverse moves: [[-1, 2]] using 4 frames

Note: there is a similar method for Y

\*

DR2+UR2 [[+4, 62]]

L1 [[-2, 64]]

=> [[+1,254]] using 3 frames

Note: it can be reversed + similar method for Y

\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*

Methods to adjust X subpixel:

DL0+UL0 [[-2,31]]

R1 [[+2,64]]

=> [+0, 33] using 3 frames

\*

DR1+UR1 [[+3,46]]

L2 [[-3, 0]]

=> [+0, 46] using 3 frames

\*

DL2+UL2 [[-4, 62]]

2\*R1 [[+4,128]]

=> [+0, 66] using 4 frames

\*

DL2+UL2 |[-4, 62]]

DR1+UR1 [[+3, 46]]

R0 [[+1,128]]

=> [+0,112] using 5 frames

\*

DR0+UR0 [[+2, 31]]

L0 [[-1,128]]

=> [[+0,159]] using 3 frames

Note: they can be reversed + similar methods for Y

## !!!Times per level

! Why not IGT?

Although the game gives an in-game time per level, we decided to go for real-time. The main reason is that the game gives a 3s penalty every time you take damage while the main tricks that we used required to take damage. These 3s penalties are consistent with the game objective to go for damageless levels, which is rewarded by a star on the map. However this would make for a very different TAS. IGT is also weird because it only starts when you leave the starting zone.

! Table

\*”Strategy” indicates whether the regular goal zone or an out-of-bounds one was reached.

\*RTA starts at Helirin control and ends at IGT stop (= when Kururin looks happy).

\*”Matt’s RTA” refers to the [https://www.youtube.com/watch?v=DXB5DTYUei4|2015 “cheat bot” partial run by Matt Shepcar]. Neither all levels were done nor all birds were saved, so we only referenced the times that were comparable.

\*”Ryuto’s RTA” refers to the [1307M|2009 movie by Ryuto].

||Level name||Author||Strategy||RTA||Matt’s RTA||Ryuto’s RTA||

|Training 1 |E-Sh4rk|regular| 1.27| 1.42| / |

|Training 2 |E-Sh4rk|regular| 1.57| 1.62| / |

|Training 3 |E-Sh4rk|regular| 2.28| 2.28| / |

|Training 4 |E-Sh4rk|regular| 1.30| 1.80| / |

|Training 5 |E-Sh4rk|regular| 2.00| 2.00| / |

|Grasslands 1 |E-Sh4rk|regular| 1.90| 2.48| 5.87|

|Grasslands 2 |E-Sh4rk|OOB | 1.80| 6.05|10.72|

|Grasslands 3 |E-Sh4rk|regular| 6.68| / |11.40|

|Ocean 1 |mohoc |OOB | 1.40| 4.58| 5.35|

|Ocean 2 |mohoc |regular| 2.60| 2.72| 8.93|

|Ocean 3 |mohoc |regular| 6.33| / |15.75|

|Jungle 1 |mohoc |OOB | 2.67| 4.88|10.55|

|Jungle 2 |mohoc |OOB | 1.67|10.75|10.33|

|Jungle 3 |mohoc |regular| 3.20| / |14.22|

|Cake Land 1 |mohoc |OOB | 1.88| 4.00|12.70|

|Cake Land 2 |mohoc |regular| 2.80| 3.40|25.93|

|Cake Land 3 |mohoc |regular| 9.40| / |19.90|

|Cave 1 |mohoc |regular| 0.70| 1.03| 8.67|

|Cave 2 |mohoc |OOB | 5.13| 6.80|16.03|

|Cave 3 |E-Sh4rk|OOB | 3.95| 6.02|15.30|

|Cloud Land 1 |mohoc |OOB | 3.82| 5.17| 6.97|

|Cloud Land 2 |mohoc |OOB | 2.98| 4.90| 7.97|

|Cloud Land 3 |mohoc |regular| 7.08| / |10.07|

|Star Land 1 |E-Sh4rk|regular| 2,48| 2.72| 3.87|

|Star Land 2 |mohoc |OOB | 2.23| 2.95| 3.20|

|Star Land 3 |mohoc |OOB | 5.97| / |11.62|

|Ice Land 1 |E-Sh4rk|regular| 3.63| 3.73| 6.27|

|Ice Land 2 |mohoc |regular| 0.92| 1.93| 8.32|

|Ice Land 3 |mohoc |OOB | 4.78| / |27.95|

|Machine Land 1|mohoc |OOB | 3.10| 5.55| 8.30|

|Machine Land 2|mohoc |OOB | 2.58|17.53|18.35|

|Machine Land 3|mohoc |regular|13.77| / |29.03|

|Ghost Castle 1|mohoc |OOB | 3.12| / |20.97|

|Ghost Castle 2|mohoc |regular| 1.65| / |41.15|

|Ghost Castle 3|mohoc |regular| 8.80| / |28.13|

## !!!Closing

**!!mohoc’s personal comments**

I would like to thank several people:

\*E-Sh4rk

\*ThunderAxe31

\*Matt Shpecar

\*ViGadeomes