"Creating Tool Assisted Speedruns in Retro Videogaming"

Grant Godel

Union University Symposium – 29/4/14

Faculty Advisor: Dr J.S. Kirk
Super Mario Brothers III – 1.1
SMB III Stats

WR Run:
• 14.158s
• 593 Re-Records
• Edition: Prog O

My Run:
• 14.464s
• 118 Re-Records
• Edition: Prog O
Shoutouts

• Thanks to the speedrunning community
• Thanks to faculty
• Special thanks
  – RoosterTeeth Ray
  – PJ
  – SpikeVegeta
  – Raelcun
• Thanks to my roommates
What Is Speedrunning?

• Simply put, Speedrunning is the act of completing some facet a videogame with the bounds of defined parameters
• Many times, defeating the ending boss is the only parameter
• Single segment runs vs. full runs
Different Quantifiable Characteristics of Speedrunning

• 100% Perfection (All Items, Weapons, etc.)
• Defeat Final Boss
• View Final Cutscene
• No Jumping
• No Damage Taken
• No Shooting
History of Speedrunning

- Doom and Quake
- Way to add “achievements” to older generation games
- 1st TAS Video circulated around BBS and other groups
  - Came from released Doom Source Code
- June 1999: Esko Koskimaa, Peo Sjoblom, Joonatan Donner created the first TAS sharing website
- Morimoto’s Super Mario Bros. 3 TAS Speedrun in 2003 was widely regarded as the first top tier TAS Video
- Mile High Club was first true mainstream speedrun (COD: MW1)
- AGDQ
Phrases Used in TAS Speedrunning

- Input File
- Emulator
- ROM
- Frame (Advance)
- Glitch
- Category
- Lag
- Re-Recording
- Savestate
Differences between Speedrunning and TAS Speedrunning

• Computer-Assisted
• TASs can handle RNGs
• Sidescrollers lose much of their appeal
• Differentiating timing conventions (power-on, first input, last input)
• Seek true perfection
Emulators

• Examples
  – BizHawk (Atari, NES, SNES, Sega)
  – FCEUX (NES)
  – VisualBoyAdvance (GB, GBC, GBA)
• Use Lua Scripting
• RAM Watching
• Optimal Graphing Paths
• Major Problem with newer consoles:
  – You need processing power of N*the power of the machine you are emulating
Legality

• Theory – If you own the actual physical console and the physical game, you are in the clear (Second Copy Rule)
• Emulators themselves are not illegal
• Most games fall under the “abandonware” title
Picking a Game

• Pick a game you like, not a game that you love
• It is easier to pick a game that other people enjoy (for community)
• For ease, pick a game that is open-world and overambitious (Oblivion/Zelda/Fable)
• Games near the end of a console’s lifecycle, or backwards compatible ports to older generation consoles
Finding Glitches

- Start with traditional glitches/abuses
  - Pausing
  - Cases of Invulnerabilities
  - No score countdowns
  - Jumping Speeds
  - RNG Manipulation
  - Text Language/Speed & Cutscenes
  - Difficulty Levels

- Look at flags
- Look at ramifications of double inputs
- Use scripting to find collision parameters
- Camera movement
Game Data

• Specific Scripts
• Mapping Flags
• Trigger Areas
• Enemy Controls
• Data Tables
  – Enemy Health at distinct levels
  – Item Drops with certain flags
Pause Trick Example
Falling Example
Collision Example
Examples of Lua Scripting (Gui ID)

- `guid event.onframestart(LuaFunction luaf, [string name])`
- `guid event.onframeend(LuaFunction luaf, [string name])`
- `guid event.onmemorywrite(LuaFunction lauf, [int address], [string name])`
Examples of Lua Scripting (Memory)

- string memory.getcurrentmemorydomain (void)
- memory.registerwrite (int address, {int size}, function)
- void gui.drawLine(int x, int y, int x2, int y2)
- luatable mainmemory.readbyterange(int address, length)
# RAM Watching

<table>
<thead>
<tr>
<th>Address</th>
<th>Value</th>
<th>Notes</th>
</tr>
</thead>
<tbody>
<tr>
<td>00BD</td>
<td>56</td>
<td>X Speed</td>
</tr>
<tr>
<td>0075</td>
<td>3</td>
<td>X Level 2nd</td>
</tr>
<tr>
<td>0090</td>
<td>111</td>
<td>X Level</td>
</tr>
<tr>
<td>074D</td>
<td>112</td>
<td>X Subpix</td>
</tr>
<tr>
<td>00FD</td>
<td>239</td>
<td>X Camera</td>
</tr>
<tr>
<td>00AB</td>
<td>128</td>
<td>X Screen</td>
</tr>
<tr>
<td>0000</td>
<td>16</td>
<td></td>
</tr>
<tr>
<td>00CF</td>
<td>0</td>
<td>Y Speed</td>
</tr>
<tr>
<td>0087</td>
<td>1</td>
<td>Y Level 2nd</td>
</tr>
<tr>
<td>00A2</td>
<td>112</td>
<td>Y Level</td>
</tr>
<tr>
<td>075F</td>
<td>112</td>
<td>Y Subpix</td>
</tr>
<tr>
<td>0001</td>
<td>102</td>
<td></td>
</tr>
<tr>
<td>03DD</td>
<td>127</td>
<td>P Meter</td>
</tr>
<tr>
<td>056E</td>
<td>0</td>
<td>P Kill Counter</td>
</tr>
<tr>
<td>0515</td>
<td>15</td>
<td>Next P</td>
</tr>
<tr>
<td>0002</td>
<td>1</td>
<td></td>
</tr>
<tr>
<td>0079</td>
<td>2</td>
<td>X Map Position</td>
</tr>
<tr>
<td>0014</td>
<td>0</td>
<td>Back to Map</td>
</tr>
<tr>
<td>ABCD</td>
<td>0</td>
<td></td>
</tr>
<tr>
<td>7CFA</td>
<td>0</td>
<td>Boss HP</td>
</tr>
<tr>
<td>FFFF</td>
<td>0</td>
<td></td>
</tr>
<tr>
<td>05F1</td>
<td>28</td>
<td>Clock Timer</td>
</tr>
<tr>
<td>055D</td>
<td>5</td>
<td>8 Frame Timer</td>
</tr>
<tr>
<td>0015</td>
<td>95</td>
<td>Card Timer</td>
</tr>
<tr>
<td>0781</td>
<td>54409</td>
<td>RNG</td>
</tr>
</tbody>
</table>
Human vs Computer

• To a certain extent, some humans could do many examples throughout many games with enough practice and time
Super Mario 64 O-Star Run
Super Mario 64 Stats

WR Run:
• 5:02.25
• (Snark, Kyman, sonicpacker, Mickey, ToT)
• 96969 Re-Records

My Run:
• 5:23.13
• 4283 Re-Records
### Ocarina of Time Stats

**WR Run:**
- 20:09.98
- (Ganon Time of 16:57.69)
- JPN v1.0
- (Bloobiebla & MrGrunz)
- 151896 Re-Records

**My Run:**
- 21:29.19
- (Ganon Time of 18:17.22)
- JPN v1.0
- 13944 Re-Records
Ocarina of Time
Research – Universal Console Verified Program

• Certain console verifications already exist
• RNGs are still being sorted out
• Universal written in C# - Major Problems

• NES/SNES takes a frame every 8 bytes
• Emulators pad differently
• Controllers with different voltages
• Possible? - Yes