Portal’s physics engine rebuilt in 25KB—on a graphing calculator

A student's years of coding work allows us all to calculate with portals.

by Casey Johnston - Dec 5, 2012

http://arstechnica.com/gadgets/2012/12/portals-physics-engine-rebuilt-in-25kb-on-a-graphing-calculator/

A 20-year-old college student has rebuilt *Portal*, Valve's 2007 space-bending game, from the ground up, on—wait for it—a graphing calculator. In a display that puts the old calculator versions of *Mario* and*Tetris* to shame, Alex Marcolina posted to a gaming forum and reddit on Sunday about his re-engineered version of *Portal.* It took three years to build and cannot, due to resource constraints on TI-83/84 calculators, execute more than 16 kilobytes of code.

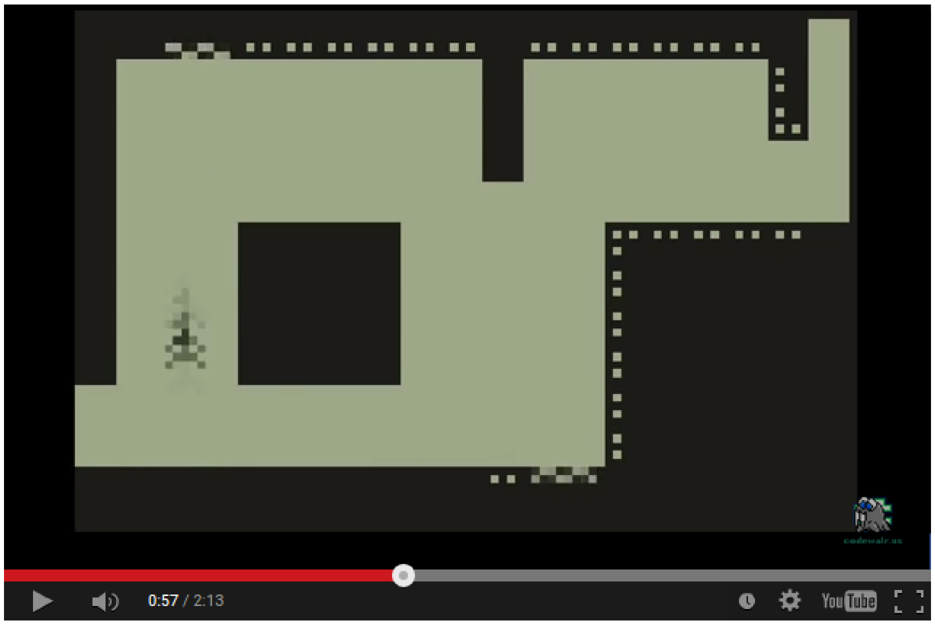
When Marcolina set out to rebuild *Portal* on TI’s graphing calculator platform, he was 17. Now, he’s a 20-year-old game design major at UC-Santa Cruz who programs games mainly for computers, but likes to dabble in graphing calculator games on occasion because it's “a fun challenge to make a game for a platform that is not supposed to even support games."

The native language for the TI-83 and 84 calculators is called TiBasic. But when it comes to making games, creators favor a language called Axe, developed by a member of the calculator and PC gaming forum Omnimaga. Marcolina points out the syntax for Axe is “very loose, but it allows for good optimization in the translation from code to assembly.”

Calculating with Portals

To represent portal travel, Marcolina told Ars he had to create two separate sets of variables: `x` and `y` for regular space, and `i` and `j` for “Portal Space” (when the player is moving through a portal). `I` represents how far into the portal the player is, and `j` the side-to-side movement relative to the portal.

The entire source code for the project is freely available, but Marcolina highlights the physics portion for us here:



```
:2
:While →θ
:  If R
:    H→{L1+30}r
:    GO()
:    1-I→I
:    RO()
:    Pt-On({L1+22}r,{L1+24}r,Pic2+r1)
:    !If I-1
:      E-(O*256/5)→{L1+28}r
:      {L1+22}r→{L1+16}r*256/5+1→D
:      {L1+24}r→{L1+18}r*256/5+1→E
:      M()
:    End
:  End
:  Exch(oN,oS,10)
:  θ-1
:End
```
The subroutine GO converts normal space coordinates to "Portal Space" for the entrance portal, while RO does the opposite for the exiting portal. When the player is only partially through the portal, the code draws a "shadow object" on the exiting side, but the player's position is only moved once they're entirely through.

Subroutine M represents the fun part: the satisfying launches through the air players can achieve by dropping through portals from high up on a ledge. M dictates how the game redirects the player's velocity when going through the exit portal based on how fast they were going when entered the other side, but translated to the direction of the new portal.

Marcolina told Ars the code governing the portal gun and placement of portals was more complex than the bit controlling how the player moves. As in the original *Portal*, portals can't be placed anywhere, but instead must be "bumped" from the location they're shot to fit with the geometry of the level. "There needs to be a good amount of code in place to make sure the portals don't get bumped into invalid locations, or even outside the level," Marcolina said. Marcolina also had to attempt to replicate the characteristic *Portal*level design, but under severe memory constraints (graphing calculators are barely modern devices any more, having only just recently gotten color screens). Apps in a graphing calculator are limited to executing 16 kilobytes of code, a severe constraint given the amount of animation involved, Marcolina said.

Overall, he favored levels that are more mentally challenging, like the earlier levels of the original *Portal*, rather than the later levels that require good timing and hand-eye coordination. "Levels that require extreme dexterity to solve wouldn't work very well due to the less than optimal control," Marcolina said.

While programming for graphing calculators admittedly doesn't give Marcolina's work a wide reach, he points out that the constraints of the platform teach him to be a better coder. "Oftentimes on computers, when you are first learning to code, going the lazy way has little effect on the end result because computers are so fast anyway that you can't tell the difference," he said. "But on a calculator you have to work very hard to make sure you make every last instruction of your code be as useful as possible."