How To Compute Without Variables, Logic Operators or Measurements

I call this a stateless computer. The logic is pure connectionism, only using connections and nothing else. I look at it as a geometry of information, or geometric logic. I think it could be described as “mechanical”. My system performs the logic only using one command without numeric variables, without logic operators, and without measurements. This is neither digital nor analog logic.

This is not theory, I have built a working model using this logic that demonstrates if-then, do-while, a randomizer, a relational database and other logic. In the working model I only use one command for hooking in the input, a few commands for output, but all the logic in between is one command that does nothing but link commands together.

The logic demonstrated in this model uses the command “alias”, which is used in a FPS video game called Counter-Strike, which is a modification for a video game made by Valve called Half-Life, which is based on id Software's QuakeWorld engine. This command is used to link various commands together to provide a way for customizing the interface of the game. This logic requires input and output provided in the game - which, no doubt, uses Boolean logic to perform, but the logic itself is contained to using the one command “alias”.

The slapaho11.zip file included with this document is a script for Counter-Strike version 1.6 that performs logic using these techniques.

Copies of slapaho11.zip can be found at these web sites:

https://app.box.com/s/5gcqav1ddjic80k55eyq

In slapaho.txt you can read about how to use the system, and in slapaho.gif you can see the keyboard layout for the functions.

I have the complete evolution of the script from its start and through one complete rewrite.

This really comes down to it all just being links that are breakable and buildable, of multiple connections cascading throughout the system. This design could be parallel if it was not on top of a conventional Turing Machine.

If you do not want to read about how it works, at least skip to page seven and look at the grenade throwing part of the script, which performs all the permutations of a math problem that is solved using Pascal’s Triangle in a new way.

When I write “system command” I mean a command built into the system. When I write “user command” I mean a command that the user made. When I write “command” I mean both of the above together.

I've had to build commands for the system with only four types of system commands to start with: 1) “wait”, which gives a few milliseconds of interrupt and is not important other than to overcome a bug in the system I used, 2) Input from the user through the command “bind” which will bind a key like “keypad_0” or “enter” to a command (exclusively, just as a tool to cause a command to happen, you cannot read variables as input into the scripting system from the computer). 3) Various outputs like “say” words to chat and “echo” words to the console and toggles like “+reload”, “-reload”, “+use”, “-use”, “+jump”, “-jump”, etc., designed for when you press a key down, + happens, and then release the key, - happens, usually the + starts the action and the – stops the action. 4) The last system command is by far the most powerful, “alias”. This allows you to created a user command that is a linking together of other commands, both system and user made.

You can link multiple things together like:

alias superuse "+use ; wait ; -use"

Which will create the user command “superuse” that is not a toggle, but the pressing down stroke that will achieve the same thing as an on/off toggle where you have to press and release the key to get the whole effect.

Also, aliases can be used together like:

alias superusejump "superuse ; +jump ; wait ; -jump"

Or as a toggle:

alias +superusejump2 "superuse ; +jump"
alias -superusejump2 "-jump"

Here is volume control, note that "volume *.**" is an output system command, where the *.** is a number from zero to one (full volume). The system command “echo” puts text in the console (output). Two keys are bound to user commands (input), “volup” and “voldn”, and user command (“v0”-”v10”) “v5” is called at the end to initialize the system:

```
// Game Volume
alias v0 "volume 0.00;echo Volume Muted;alias volup v1; alias voldn v0"
alias v1 "volume 0.10;echo Volume 10%;alias volup v2; alias voldn v0"
alias v2 "volume 0.20;echo Volume 20%;alias volup v3; alias voldn v1"
alias v3 "volume 0.30;echo Volume 30%;alias volup v4; alias voldn v2"
alias v4 "volume 0.40;echo Volume 40%;alias volup v5; alias voldn v3"
alias v5 "volume 0.50;echo Volume 50%;alias volup v6; alias voldn v4"
alias v6 "volume 0.60;echo Volume 60%;alias volup v7; alias voldn v5"
alias v7 "volume 0.70;echo Volume 70%;alias volup v8; alias voldn v6"
alias v8 "volume 0.80;echo Volume 80%;alias volup v9; alias voldn v7"
alias v9 "volume 0.90;echo Volume 90%;alias volup v10; alias voldn v8"
alias v10 "volume 1.00;echo Volume 100%;alias volup v10; alias voldn v9"

bind + “volup”
```

bind – “voldn”
v5

I made the following code part of the model for jumping in Counter-Strike games. User command “+sjump1” lets you jump onto ladders and stay on them, since it does the jump in one downward key stroke, instead of system command “-jump” upon release, which if you did would cause you to system command “-jump”/bounce off the ladder when you release the button.

It allows you to switch from (starting with) jump and duck to just jump to just duck, with the toggle. “bind” is the system command to link a key with a command, whether it is a command created by a programmer with system commands “alias”, or like “+duck”. The two keys bound are for the actually execution of the action and the rotation of what the action does. User command “togglerotation1” is called at the end to initialize the system.

So, “alias” “bind” “wait” “+jump” “-jump” “+duck” “-duck” are system commands, “null” “+sjump1” “-sjump1” “+sjump2” “-sjump2” “toggle” “togglerotation1” “togglerotation2” “togglerotation3” “superjump” are all user made commands with the “alias” system command, and “space” and “enter” are the input keys to be bound to commands with the system command “bind”:

```
alias null ""
alias +sjump1 "+jump ; wait ; -jump ; +duck"
alias -sjump1 "-duck"
alias +sjump2 "+jump ; wait ; -jump"
alias -sjump2 "null"
alias togglerotation1 "alias +superjump +sjump1 ; alias toggle togglerotation2"
alias togglerotation2 "alias +superjump +sjump2 ; alias toggle togglerotation3"
alias togglerotation3 "alias +superjump +duck ; alias toggle togglerotation1"

bind space +superjump
bind enter toggle
togglerotation1
```

Conditional pseudo-randomizer

The randomizer is the easiest to understand of the complex code I have created.

The insult.cfg file in slapaho11.zip shows how I did pseudo-randomization with this system based upon player input. It is pseudo-randomization because if you follow the player input you can predict the resulting taunt that is said.

If set to "AutoTaunt Extremely Obnoxious" the script has a 30% chance of saying a general taunt, a 10% chance to say a secondary taunt, a 10% chance to say a weapon specific taunt, and a 50% chance to say nothing when the "gorand" command is used to push the randomizer, and the taunt is said when the gun is fired. In this mode, I estimate that overall it has less than a 25% chance to say anything when I fire, because the "gorand" command is not run often.

If set to "AutoTaunt Low" there is a 10% chance to say a general taunt, a 2.5% chance to say a secondary taunt, a 2.5% chance to say weapon specific taunt, and an 85% chance to say nothing when the "gorand" command is used and I fire my gun.

Firing a gun is when a taunt gets said - in slapaho6.cfg, line 19, the "burst" command, and line 20, the "singlefire" command, have the command "saytnt" which is linked to the current taunt to be said. This command "saytnt" in line 4 of insult.cfg is the command to actually say a taunt. Notice that the command "tnt" is followed by the command "ntnt" which stands for null-taunt, which clears the "tnt" command by placing the command "null" into it, and is defined in line 5 of insult.cfg as "alias tnt null". This is done to clear the "tnt" command after something is said once, so that a particularly loaded taunt will not be repeatedly said when firing a gun.

The command "gorand" in lines 12 and 13 of insult.cfg is what pushes the randomizer, used in slapaho2.cfg when moving forward (+sforward), moving back (+sback), finishing planting the C4 bomb (-suse), finishing getting a hostage/opening a door/pushing a switch/defusing the bomb (-suse2), and finishing reloading a weapon (-sreload). The "gorand" command is also used in slapaho5.cfg when throwing a grenade in lines 42, 44, 46 and 48. It is also used in slapaho4.cfg in line 24 when zooming weapons.

Note that this pseudo-randomizer is conditional, in particular because when using the "+sback" command it uses the command "ntnt" after using the command "gorand" to clear the "tnt" command, because I do not want it to say a taunt when retreating.

Here is the main part of the code that actually does the randomization:

dv1 and dv0 are old commands that do not work anymore to echo information from the console onto the screen. You can still pull down the console and see the information there.

gtnt, wptnt, stnt are in the relational database as taunts that are said, that part is after this.

Bind saytnt to a key for output, and bind gorand to a key to push the randomizer.

```
// AutoTaunt System
alias saytnt "tnt;ntnt"
alias ntnt "alias tnt null"
alias rstaiskip "rnullb1;alias ciskip1 ntnt;alias ciskip2 ntnt;alias ciskip3 ntnt;alias ciskip4 ntnt"
alias atskipr1 "alias atskipr atskipr2;rstaiskip;dv1;echo AutoTaunt LOW;dv0"
alias atskipr2 "alias atskipr atskipr3;alias ciskip1 gtnt;alias ciskip2 wptnt;dv1;echo AutoTaunt MEDIUM;dv0"
alias atskipr3 "alias atskipr atskipr4;alias ciskip3 gtnt;alias ciskip4 stnt;dv1;echo AutoTaunt HIGH;dv0"
alias atskipr4 "alias atskipr atskipr1;alias rnullb rnull;dv1;echo AutoTaunt EXTREMELY OBNOXIOS;dv0"
```

```
// Overall Incurrence
alias rnulla0 "alias gorand rnulla1;rnullb"
alias rnulla1 "alias gorand rnulla0;ntnt"
alias rnullb0 "alias rnullb rnullb1;rnull"
alias rnullb1 "alias rnullb rnullb0;ntnt"

// Categorical Incurrence
alias rnull0 "alias rnull rnull1;gtnt"
alias rnull1 "alias rnull rnull2;ciskip1"
alias rnull2 "alias rnull rnull3;wptnt"
alias rnull3 "alias rnull rnull4;gtnt"
alias rnull4 "alias rnull rnull5;ciskip3"
alias rnull5 "alias rnull rnull6;gtnt"
alias rnull6 "alias rnull rnull7;ciskip2"
alias rnull7 "alias rnull rnull8;stnt"
alias rnull8 "alias rnull rnull9;gtnt"
alias rnull9 "alias rnull rnull0;ciskip4"
```

The relational database part is very simple, as seen in insult2.cfg in the slapaho11.zip file. When a weapon specific taunt is chosen, it is possible to choose a taunt that is used in a weapon type taunt. For example, under "// Awp Taunts" it is possible for the script to choose one of the commands "tnt43" or "tnt44" as AWP (a sniper rifle) only taunts, but it can also point back to "tnt39" in "// Sniper Taunts", also in a relational way. Notice how many main categories with taunts relating the type of gun, and many sub categories with taunts relating to the specific weapon.

Here are secondary taunts, it is a simple rotation like the volume script:

```
// Secondary Taunts
alias stnt0 "alias tnt tnt10;alias stnt stnt1";alias tnt10 "say I am unstoppable!"
alias stnt1 "alias tnt tnt11;alias stnt stnt2";alias tnt11 "say Am what?"
alias stnt2 "alias tnt tnt12;alias stnt stnt3";alias tnt12 "say Just Great!"
alias stnt3 "alias tnt tnt13;alias stnt stnt4";alias tnt13 "say Ohhh, I am goood"
alias stnt4 "alias tnt tnt14;alias stnt stnt5";alias tnt14 "say Hey, wheres the cream filling?"
alias stnt5 "alias tnt tnt15;alias stnt stnt6";alias tnt15 "say Newbie!"
alias stnt6 "alias tnt tnt16;alias stnt stnt7";alias tnt16 "say Victory is ours!"
alias stnt7 "alias tnt tnt17;alias stnt stnt8";alias tnt17 "say Laaaaaaammme"
alias stnt8 "alias tnt tnt18;alias stnt stnt9";alias tnt18 "say Is this too hard for you?"
alias stnt9 "alias tnt tnt19;alias stnt stnt10";alias tnt19 "say LIT YOU UP!"
alias stnt10 "alias tnt tnt20;alias stnt stnt11";alias tnt20 "say Lalalala this is boring..."
alias stnt11 "alias tnt tnt21;alias stnt stnt12";alias tnt21 "say Everyone should be as good as me!"
alias stnt12 "alias tnt tnt22;alias stnt stnt13";alias tnt22 "say Your ass is somewhere over there..."
```

```
alias stnt13 "alias tnt tnt23;alias stnt stnt14";alias tnt23 "say That head will look nice in my trophy room"
alias stnt14 "alias tnt tnt24;alias stnt stnt15";alias tnt24 "say BANG! Your dead"
alias stnt15 "alias tnt tnt25;alias stnt stnt16";alias tnt25 "say TERMINATED"
alias stnt16 "alias tnt tnt26;alias stnt stnt17";alias tnt26 "say Bad to the bone!"
alias stnt17 "alias tnt tnt27;alias stnt stnt18";alias tnt27 "say SCOOOOOOOOOOOORE!!!!"
alias stnt18 "alias tnt tnt28;alias stnt stnt19";alias tnt28 "say Got one!"
alias stnt19 "alias tnt tnt29;alias stnt stnt20";alias tnt29 "say Sorry, did that hurt?"
alias stnt20 "alias tnt spam;alias stnt stnt0"
```

In the slapaho11.zip file there is a configuration file called slapaho3.cfg, this file has an example of the "if this and that then do ..." logic. With this script I have the ability to program a single buy-all button, which can buy virtually all combinations of the various equipment available in the game. In the game, by default, I am given a knife and a weak pistol with the ability to buy another pistol and/or a non-pistol weapon, such as a MP5 or an AK-47. One of the options in my script is to buy the best pistol in the game (the .50 caliber Desert Eagle) in combination with a greater non-pistol weapon. I can only have one pistol and one non-pistol, so if the option of buying a Desert Eagle as a secondary weapon is used in combination with buying another pistol, my script knows not to buy the Desert Eagle.

The command "buyall" in line 4 of slapaho3.cfg is the command for the buy all bound button. In this line of code, near the end, is the command "decsbuy" to buy the Desert Eagle as a secondary weapon, in combination with a greater non-pistol primary weapon. If I use the command "debuy1" in the 25th line of slapaho3.cfg, it puts the command "decsb" into the command "decsbuy". If I use the command "de1c" in the 15th line of slapaho3.cfg, it puts the command "debsec" in the 20th line into "decsb", completing a chain of links to buy the Desert Eagle.

Basically it goes: "buyall" -> "decsbuy" -> "decsb" -> "debsec" -> "deagle;secammo" where "deagle" is the Counter-Strike command to buy the Desert Eagle and "secammo" is the Counter-Strike command to buy secondary ammo. If the command "de0c" in line 14 of slapaho3.cfg is used, through a bound button that turns buying the Desert Eagle as a secondary weapon on and off, it breaks the chain, turning off buying of the Desert Eagle by putting the command "null" into the "decsb" command. Conversely, if you use the command "de1c" in line 15 it puts the command "debsec" into the "decsb" command. If I choose a pistol to buy through a bound button, for example the P288 in line 31 of slapaho3.cfg, it will use the command "debuy0" in line 24, putting "null" into "decsbuy" which will also break the chain, turning off buying of the Desert Eagle. Conversely, if I choose a non-pistol weapon, through a bound button, like the MP5 in line 41, the script will use the command "debuy1" in line 25, putting "decsb" into "decsbuy". If everything is linked properly, the Desert Eagle as a secondary weapon will be bought. If the chain is broken anywhere, the script will not buy the Desert Eagle as a secondary weapon.

The grenade tossing do-while loop is the most complicated to understand.

I did something kind of interesting afterward figuring out the word problem and then a man named Ron Allen answering it. The application took nothing into account as something you have to count; like with the ancient Chinese Pascal's Triangle question: the old how many combinations of hats you can have on pegs, where they didn't count empty pegs as something to count as part of the combination, I had to. Nothing has value. You learn that when most of your learning has been in 0 and 1.

Here is a word problem I came up with that sums up what slapaho5.cfg performs:

You have a combination padlock with four dials on it. Each dial has the numbers 0 through 4 on them. The lock can have as many 0s as dials, and is set to 0000 by default. The lock does not allow you to use any number between 1 and 4 two or more times in the combination. The following combinations are valid: 0123 1234 0103 0010 4031. The following combinations are invalid: 0113 4014 0202 4444. How many possible combinations are there?

The solution to this word problem is here, notice that it is a new use for Pascal's Triangle, because it values nothing as something:

http://answers.yahoo.com/question/index;_ylt=Aicrr4ngCQthePBgy063rmrsy6IX?qid=20060710103458AAVr9ih

More formulas can be found here:

http://mathhelpforum.com/discrete-math/17147-combination-lock.html

This code performs all of the permutations the math says is possible. The code seems convoluted, but to expand it with more numbers and dials would be trivial once the code is understood. The problem is that what this code does could be done in much less space and less convolution with a conventional Turing machine.

In this example, there are 209 possible combinations for tossing four possibly bought grenades, and counts empty slots as something. With this script, you can achieve any one the 209 possible combinations in four key strokes or less. If this part of the script was to be scaled up to 18 wheels and (base) 18 numerals, there would be 2,968,971,264,021,448,999 or almost $3x10^{18}$ possible combinations, each reachable within 18 or less key strokes.

Bind +gtoss to the button you want to toss from, and bind four keys to grnslta, grnsltb, grnsltc, grnsltd for selecting which grenades you want to buy and use in what order. Slot 1 will always throw the first grenade, slot 2 the second, etc. When using X C V B to select grenades, HE is first, FB second, FB third, Smoke last.

For example, if I wanted a Smoke first, FB second, HE third, and FB fourth I would press V C B X in that order.

Notice that it finishes by picking all of the types of grenades to throw, most important first.

This part of the script is whole except for rstgof2 which is a command to setup a three button mouse with the third button throwing grenades, it is located in slapaho2.cfg, grengtbst located in slapaho4.cfg, and gorand which pushes the randomizer.

You also have to initialize the system with gsrst, gbrst, skiprst, grnrt0, and grnrtb1

```
// Grenade Throwing System
alias null ""
alias gsrst "alias +gs1 null;alias -gs1 gbutrot;alias +gs2 null;alias -gs2 gbutrot;alias +gs3 null;alias -gs3 gbutrot;alias +gs4 null;alias -gs4 gbutrot"

alias gbrst "alias gbslt1 null;alias gbslt2 null;alias gbslt3 null;alias gbslt4 null;alias gbutrot gbutrot1;alias +gtoss +gtall;alias -gtoss -gtall;rstgof2"

alias skiprst "alias skip1 skip1rst;alias skip2 skip2rst;alias skip3 skip3rst;alias skip4 skip4rst"

alias skip1rst "alias +gtoss +gs2;alias -gtoss -gs2;alias gbutrot gbutrot3;skip2"
alias skip2rst "alias +gtoss +gs3;alias -gtoss -gs3;alias gbutrot gbutrot4;skip3"
alias skip3rst "alias +gtoss +gs4;alias -gtoss -gs4;alias gbutrot skip4rst;skip4"
alias skip4rst "alias +gtoss +gtall;alias -gtoss -gtall;alias gbutrot null;rstgof2"

alias grst1 "alias grnslta grnrta1;alias grnsltb grnrtb1;alias grnsltc grnrtc1;alias grnsltd grnrtd1;gsrst;gbrst;skiprst"
alias grst2 "alias grnslta grnrta2;alias grnsltb grnrtb2;alias grnsltc grnrtc2;alias grnsltd grnrtd2;alias gbhook rstnew2"
alias grst3 "alias grnslta grnrta3;alias grnsltb grnrtb3;alias grnsltc grnrtc3;alias grnsltd grnrtd3"
alias grst4 "alias grnslta grnrta4;alias grnsltb grnrtb4;alias grnsltc grnrtc4;alias grnsltd grnrtd4"
alias grst5 "alias grnslta grnrt0;alias grnsltb grnrt0;alias grnsltc grnrt0;alias grnsltd grnrt0"

alias grnrt0 "grst1;alias gbhook null;dv1;echo FULL RESET of Gren Slots;dv0"
alias grnrta1 "grst2;alias +gs1 +gthe;alias -gs1 -gthe;alias gbslt1 hegren;alias skip1 null;dv1;echo Gren Slot 1 HE;dv0"
alias grnrta2 "grst3;alias +gs1 +gtfb;alias -gs1 -gtfb;alias gbslt1 flash;alias skip1 null;dv1;echo Gren Slot 1 FB1;dv0"
alias grnrta3 "grst4;alias +gs1 +gtfb;alias -gs1 -gtfb;alias gbslt1 flash;alias skip1 null;dv1;echo Gren Slot 1 FB2;dv0"
alias grnrta4 "grst5;alias +gs1 +gtsmk;alias -gs1 -gtsmk;alias gbslt1 sgren;alias skip1 null;dv1;echo Gren Slot 1 SMOKE;dv0"
alias grnrtb1 "grst2;alias +gs2 +gthe;alias -gs2 -gthe;alias gbslt2 hegren;alias skip2 null;dv1;echo Gren Slot 2 HE;dv0"
alias grnrtb2 "grst3;alias +gs2 +gtfb;alias -gs2 -gtfb;alias gbslt2 flash;alias skip2 null;dv1;echo Gren Slot 2 FB1;dv0"
alias grnrtb3 "grst4;alias +gs2 +gtfb;alias -gs2 -gtfb;alias gbslt2 flash;alias skip2 null;dv1;echo Gren Slot 2 FB2;dv0"
alias grnrtb4 "grst5;alias +gs2 +gtsmk;alias -gs2 -gtsmk;alias gbslt2 sgren;alias skip2 null;dv1;echo Gren Slot 2 SMOKE;dv0"
alias grnrtc1 "grst2;alias +gs3 +gthe;alias -gs3 -gthe;alias gbslt3 hegren;alias skip3 null;dv1;echo Gren Slot 3 HE;dv0"
alias grnrtc2 "grst3;alias +gs3 +gtfb;alias -gs3 -gtfb;alias gbslt3 flash;alias skip3 null;dv1;echo Gren Slot 3 FB1;dv0"
alias grnrtc3 "grst4;alias +gs3 +gtfb;alias -gs3 -gtfb;alias gbslt3 flash;alias skip3 null;dv1;echo Gren Slot 3 FB2;dv0"
alias grnrtc4 "grst5;alias +gs3 +gtsmk;alias -gs3 -gtsmk;alias gbslt3 sgren;alias skip3 null;dv1;echo Gren Slot 3 SMOKE;dv0"
alias grnrtd1 "grst2;alias +gs4 +gthe;alias -gs4 -gthe;alias gbslt4 hegren;dv1;alias skip4 null;echo Gren Slot 4 HE;dv0"
alias grnrtd2 "grst3;alias +gs4 +gtfb;alias -gs4 -gtfb;alias gbslt4 flash;dv1;alias skip4 null;echo Gren Slot 4 FB1;dv0"
alias grnrtd3 "grst4;alias +gs4 +gtfb;alias -gs4 -gtfb;alias gbslt4 flash;dv1;alias skip4 null;echo Gren Slot 4 FB2;dv0"
alias grnrtd4 "grst5;alias +gs4 +gtsmk;alias -gs4 -gtsmk;alias gbslt4 sgren;alias skip4 null;dv1;echo Gren Slot 4 SMOKE;dv0"

alias gbutrot1 "alias +gtoss +gs1;alias -gtoss -gs1;alias gbutrot gbutrot2;skip1"
alias gbutrot2 "alias +gtoss +gs2;alias -gtoss -gs2;alias gbutrot gbutrot3;skip2"
alias gbutrot3 "alias +gtoss +gs3;alias -gtoss -gs3;alias gbutrot gbutrot4;skip3"
alias gbutrot4 "alias +gtoss +gs4;alias -gtoss -gs4;alias gbutrot skip4rst;skip4"

alias +gthe "grengtbst;weapon_hegrenade;+attack;hewarn;gorand"
alias -gthe "-attack;gbutrot"
alias +gtfb "grengtbst;weapon_flashbang;+attack;fbwarn;gorand"
alias -gtfb "-attack;gbutrot"
alias +gtsmk "grengtbst;weapon_smokegrenade;+attack;gorand"
alias -gtsmk "-attack;gbutrot"
alias +gtall "grengtbst;weapon_smokegrenade;weapon_flashbang;weapon_hegrenade;+attack;gorand"
alias -gtall –attack
```

What I really need is recognition that this is not conventional logic; that this really is something different and outside of the box.

I want to patent this and put it in the public domain under a license similar to what Linux uses, with no strings attached – I do not want anyone to have rights over it, even me.

.