Desert of the Real

Leon Rische

[2020-04-05 Sun 19:41]

Contents

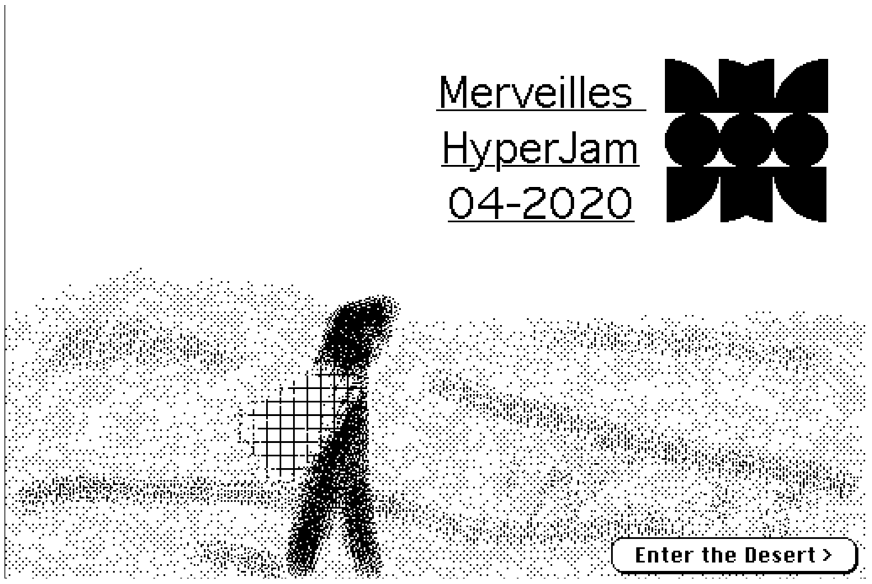


Figure 1: Title screen

Entry for the Merveilles HyperCard jam.

- Download: Desert of the Real Disk

- On itch.io: `https://l3kn.itch.io/desert-of-the-real`
- Other Submissions: `https://itch.io/jam/merveilles-hyperjam/entries`

1 Concept

The name of the game comes from Jean Baudrillard's book "Simulacra and Simulation".

> Today abstraction is no longer that of the map, the double, the mirror, or the concept.
>
> Simulation is no longer that of a territory, a referential being, or a substance. It is the generation by models of a real without origin or reality: a hyperreal.
>
> The territory no longer precedes the map, nor does it survive it. It is nevertheless the map that precedes the territory - precession of simulacra - that engenders the territory, and if one must return to the fable, today it is the territory whose shreds slowly rot across the extent of the map.
>
> It is the real, and not the map, whose vestiges persist here and there in the deserts that are no longer those of the Empire, but ours. The desert of the real itself.

One theme of the book is how reality has been reduced to (and can be reproduced from) integrated circuits, the simplified languages of computer science, genetic codes.

This reminded me of how Quadtree Grammars can produce intricate structures from a set of simple rules.

> No more imaginary coextensivity: it is genetic miniaturization that is the dimension of simulation. The real is produced from miniaturized cells, matrices, and memory banks, models of control - and it can be reproduced an indefinite number of times from these.

To contrast the clean shapes of the generated structures, everything else is drawn using the spray can tool, which fits nicely into the desert theme.

The player enters the desert of the real, encounters multiple structures reduced to (or reproduced from) simple rule systems, and upon reaching the final pyramid, unlocks the editor that allows them to view and modify the rules behind each of the structures.

2 Images

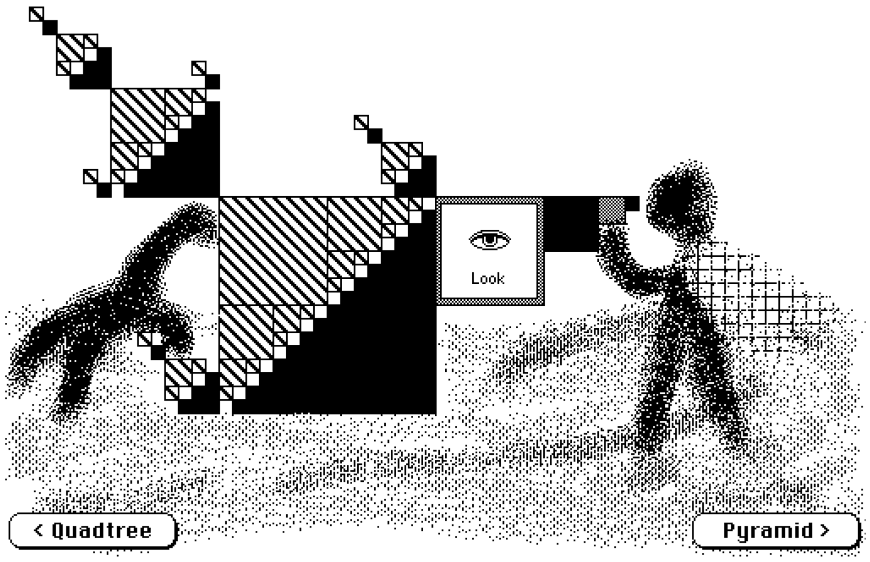


Figure 2: Telescope in the desert

3 Annotated Source Code

Each stack in HyperCard can have multiple backgrounds and each background can have multiple cards.

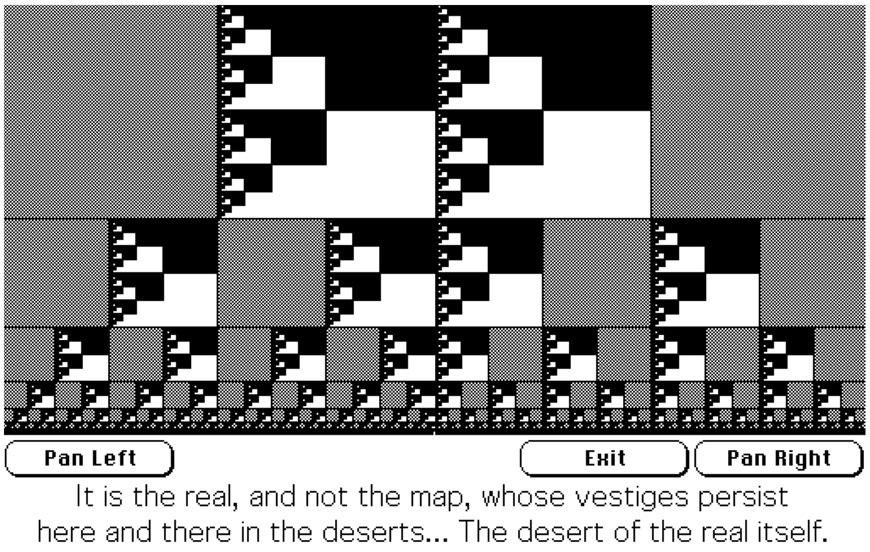
Backgrounds and cards can contain text fields, buttons and pictures (black / white / transparent).

In my stack, almost every card has its own background, used to draw the quadtree structures, with everything else drawn in the foreground.

I wanted to have a way of editing a card's background without switching to a special editor card, regenerating the quadtrees, then copying the results back to the original card.

To allow me to still change the editor interface without having to update all card backgrounds, I decided to build the whole editor interface without using any fields or buttons, just by drawing rectangles and using the text tool (which draws its text directly in the card / background picture).

The editor logic is placed in the stack script (shared between all cards and backgrounds) with a `on mouseUp` handler and a global variable to keep



It is the real, and not the map, whose vestiges persist
here and there in the deserts... The desert of the real itself.

Figure 3: Looking through the telescope

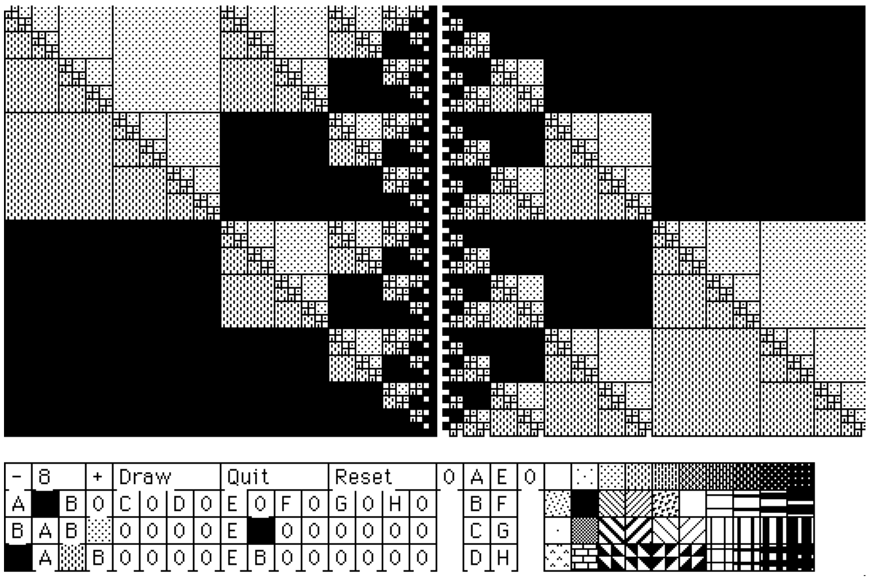

- 8 + Draw Quit Reset 0 A E 0
A B 0 C 0 D 0 E 0 F 0 G 0 H 0 B F
B A B 0 0 0 0 E 0 0 0 0 0 0 C G
A B 0 0 0 0 E B 0 0 0 0 0 0 D H

Figure 4: Quadtree editor

track of whether the editor is open or not (`gOpen`).

This way, editing the ruleset for a card's background is as simple as setting `gOpen` to `true`, then switching to the card's background.

language=C,label= ,caption= ,captionpos=b,numbers=none on editBg global gOpen put true into gOpen doMenu "background" setMinSize 8 end editBg

on quitBg global gOpen doMenu "background" put false into gOpen end quitBg

On opening a card, `gOpen` is set to false to deactivate the editors click handler.

language=C,label= ,caption= ,captionpos=b,numbers=none on open-Card global gOpen put false into gOpen end openCard

To determine if an element of a rule is a pattern or a reference to another rule, we need to check if it is a number. For performance reasons, only the first character is checked.

language=C,label= ,caption= ,captionpos=b,numbers=none function isDigit s return offset(s, "0123456789") is not 0 end isDigit

function isNumber s return isDigit(char 1 of s) end isNumber

One interesting feature of HyperCard is that there are no special functions for drawing shapes or text. Instead, you have to choose the correct tool, then call `draw from pos1 to pos2` to simulate clicking on the screen and dragging the mouse, e.g. to draw a rectangle.

Because switching tools is slow, there are two functions for drawing rectangles, one that switches tools first and one that assumes the `rect` tool is active (`drawRectF`).

Both handlers take an origin point `ox,oy`, a size `sx,sy` and a fill-pattern `p` as arguments.

language=C,label= ,caption= ,captionpos=b,numbers=none on drawRectF ox,oy,sx,sy,p if p is not 0 then set filled to true set pattern to p drag from ox,oy to (ox+sx),(oy+sy) end if end drawRectF

on drawRect ox,oy,sx,sy,p choose rect tool drawRectF ox,oy,sx,sy,p choose browse tool end drawRect

Text is drawn the same way, this time using the `text` tool and `type` to simulate entering the text using the keyboard.

language=C,label= ,caption= ,captionpos=b,numbers=none on draw-Text x,y,t choose text tool click at x,y type t choose browse tool end draw-Text

Fields in the rule editor are either filled with one of the 40 patterns or with a letter "0", "A".."H" for each of the eight rules.

`drawField` is called with a position `x,y` and a pattern / rule name `p`.

The patterns are named `1` to `40`, so we can check whether `p` is a number, then draw either a filled square or a text field.

language=C,label= ,caption= ,captionpos=b,numbers=none on drawField x,y,p if isNumber(p) and p > 0 then put (x * 16) into x put (y * 16) + 256 into y drawRect x,y,16,16,p else drawTextField x,y,1,p end if end drawField

on drawTextField x,y,w,p put (x * 16) into x put (y * 16) + 256 into y drawRect x,y,16*w,16,1 drawText (x+4),(y+13),p end drawTextField

Next, we need to draw each element of the quadtree editor.

language=C,label= ,caption= ,captionpos=b,numbers=none on drawMenu global gOpen put false into gOpen drawMainMenu selectPattern 0 drawRuleMenu drawPatternMenu1 drawPatternMenu2 put true into gOpen end drawMenu

on selectPattern p global gPattern put p into gPattern drawField 16,1,p end selectPattern

on drawPatternMenu1 drawField 17,1,"A" drawField 17,2,"B" drawField 17,3,"C" drawField 17,4,"D" drawField 18,1,"E" drawField 18,2,"F" drawField 18,3,"G" drawField 18,4,"H" drawField 19,1,"0" end drawPatternMenu1

on drawPatternMenu2 repeat with x = 0 to 9 repeat with y = 0 to 3 put 1+x+(y*10) into p put x*16+256+64 into xx put y*16+256+16 into yy drawRect xx,yy,16,16,p end repeat end repeat end drawPatternMenu2

on drawMainMenu drawTextField 0,1,1,"-" drawTextField 3,1,1,"+" setMinSize 8 drawTextField 4,1,4,"Draw" drawTextField 8,1,4,"Quit" drawTextField 12,1,4,"Reset" end drawMainMenu

Rules are stored in a hidden background field, with one rule on each line. Each line has the form `A -> A 12 A A or 1` (name of the rule `->` the four elements it expands to `or` fallback pattern).

language=C,label= ,caption= ,captionpos=b,numbers=none on drawRuleMenu repeat with x = 0 to 7 put word 8 of line x + 1 of background fld rules into p drawField x*2+1,2,p drawTextField x*2,2,1,numtochar(x + 65) end repeat repeat with x = 0 to 15 repeat with y = 0 to 1 put 1 + (x div 2) into r put word (x mod 2) + 3 + 2*y of line r of background fld rules into p drawField x,y+3,p end repeat end repeat end drawRuleMenu

on setRule l,w global gPattern put gPattern into word w of line l of background field "rules" end setRule

Each time the mouse is clicked, we check if the editor is open, then pass the click location to `handleClick`.

language=C,label= ,caption= ,captionpos=b,numbers=none on mouseUp global gOpen if gOpen then put the clickH into x put the clickV into y handleClick x,y end if end mouseUp

`handleClick` is just a big `if~/~else` that checks which of the elements has been clicked and calls the handler for it.

A neat trick here is the use of `numToChar` to translate click positions into the characters "A" to "H".

language=C,label= ,caption= ,captionpos=b,numbers=none on handleClick x,y global gMinSize, gPattern if y >= 256 then put x div 16 into x subtract 256 from y put y div 16 into y if x < 16 then if y = 1 then if x = 0 then setMinSize gMinSize div 2 else if x = 3 then setMinSize gMinSize * 2 else if x > 3 and x < 8 then drawRules else if x > 8 and x < 12 then quitBg else if x > 12 and x < 16 then resetRules end if

put 1+(x div 2) into r if y = 2 then if x mod 2 = 1 then drawField x,y,gPattern setRule r,8 end if else if y = 3 or y = 4 then drawField x,y,gPattern setRule r,2+(1+(x mod 2)+(y-3)*2) end if end if

if x = 17 and y >= 1 then selectPattern numToChar(64+y) end if if x = 18 and y >= 1 then selectPattern numToChar(68+y) end if if x = 19 and y = 1 then selectPattern 0 end if

– Pattern Selector if x >= 20 and y >= 1 then subtract 20 from x subtract 1 from y selectPattern 1 + x + (y * 10) end if end if end handleClick

`getRule` does the reverse, using `chartonum` to translate a character "A".."H" into a line number from 1 to 8.

language=C,label= ,caption= ,captionpos=b,numbers=none function getRule n return line chartonum(n) - 64 of background field "rules" end getRule

`expandRule` expands a rule element (pattern or rule reference) into a quadtree of size `s` at position `ox,oy`.

If the size is below `gMinSize`, the rules fallback pattern is used instead.

language=C,label= ,caption= ,captionpos=b,numbers=none on expandRule rule,ox,oy,s global gMinSize if s > gMinSize then put s div 2 into h put word 3 of rule into w if isNumber(w) then drawRectF ox,oy,h,h,w else expandRule getRule(w),ox,oy,h end if

put word 4 of rule into w if isNumber(w) then drawRectF ox+h,oy,h,h,w else expandRule getRule(w),ox+h,oy,h end if

put word 5 of rule into w if isNumber(w) then drawRectF ox,oy+h,h,h,w else expandRule getRule(w),ox,oy+h,h end if

put word 6 of rule into w if isNumber(w) then drawRectF ox+h,oy+h,h,h,w else expandRule getRule(w),ox+h,oy+h,h end if else drawRectF ox,oy,s,s,word 8 of rule end if end expandRule

Before (re)drawing the quadtrees, we need to clear the background.

language=C,label= ,caption= ,captionpos=b,numbers=none on clearRect choose select tool drag from 0,0 to 512,256 doMenu "Clear Picture" end clearRect

language=C,label= ,caption= ,captionpos=b,numbers=none on setMinSize m global gMinSize put min(128,max(1,m)) into m drawTextField 1,1,2,m put m into gMinSize end setMinSize

To draw the quadtrees, we expand the rule "A" at position `0,0` and the rule "E" at position `256,0`.

language=C,label= ,caption= ,captionpos=b,numbers=none on drawRules clearRect choose rect tool expandRule getRule("A"),0,0,256 expandRule getRule("E"),256,0,256 choose browse tool end drawRules

To reset the rules, reset the contents of the rule field, then redraw the rule menu.

language=C,label= ,caption= ,captionpos=b,numbers=none on resetRules repeat with i = 1 to 8 put (numtochar(64+i) " -> 0 0 0 0 or 0") into line i of background fld rules end repeat drawRuleMenu end resetRules

`initBackground` is a helper function for setting up a new background. It creates a hidden field for the rules, then draws the menu.

language=C,label= ,caption= ,captionpos=b,numbers=none on initBackground doMenu new field set the name of background field 1 to "rules" hide background field rules repeat with i = 1 to 8 put (numtochar(64+i) " -> 0 0 0 0 or 0") into line i of background fld rules end repeat drawMenu end initBackground