Playing games with observation, dependency and agency in a new environment for making construals

Meurig Beynon    Jonathan Foss
and the CONSTRUIT! project team
Making construals as a new digital skill for creating interactive open educational resources

construit.org
jseden.dcs.warwick.ac.uk/construit.c3

“The environment for making construals” – aka as “the MCE”
Playing *games* ...

Noughts-&-Crosses (N&C)! $\rightarrow$ OXO-like games

... *with observation, dependency and agency*

key concepts in “making construals”

... *in a new environment for making construals*

jseden.dcs.warwick.ac.uk/construit.c3
Imagine you are watching two people play N&C

Explain what you think is going on ... 

- there’s a grid, some squares
- players take turns to place Os and Xs
- if O/X makes a line of Os/Xs, they win
- if neither player makes a line, it’s a draw
Imagine two people playing what looks like N&C

Explain what you think is going on ...

- when an O disappears from the grid
- when X makes two successive moves
- when X wins in the position pictured above
- when the board rotates after each move
Some possible construals

Perhaps when we see the two ‘N&C players’ ...
- they are designing a board for N&C
- a piece falls of the board in play
- a player cheats
- the winning lines are different
- the playing protocol has been changed

```
 O X O
 __________
 | X |
 __________
 | X O |
```
What does it mean to make a construal?
We link what we think is going on with:
- what/who is *responsible for changes*
- what we/they must be *observing*
- what *connects the changes* we/they observe

These are *agency / observables / dependencies*
Making a digital construal

From which perspective is the maker making the construal e.g. Agents? Constraints?

Script of definitions of observables with associated network of dependencies

What interactions and interpretations is the maker familiar with? Convinced of? puzzled about?

What external subject does the maker have in mind when interacting with the construal?

A connection experienced by the maker
s4 is boardstate[4];
lin2 is [s4,s5,s6];
alllines is [lin1,lin2,lin3,lin4,]
    lin5,lin6,lin7,lin8];
owonI is (alllines[_i][1] == o) &&
    (alllines[_i][2] == o) &&
    (alllines[_i][3] == o);
owonls is owonI with _i is 1..alllines#;
owon is positionInList(true, owonls) != 0;
draw is (! xwon) && (! owon) && full;
status is (xwon?"X wins ":") //
    (owon?"O wins ":") //
    (draw?"Draw ":") // "";

**Script** of observable definitions

and **network** of dependencies
The OXO Laboratory

Winning lines

Status of an arbitrary OXO position

Square evaluation and chosen move

State of OXO board in play
The OXO Lab in the MCE

```java
import wmb/OXOlab/geometry;
import wmb/OXOlab/winninglines;
import wmb/OXOlab/status;
import wmb/OXOlab/boardcontent;
import wmb/OXOlab/selectmove;
import wmb/OXOlab/statusrules;

/*
  geometry - the grid lines and key locations in the grid
  winninglines - the conceptual triples of winning lines
  status - where the pieces are on the board
  boardcontent - the pieces as visually represented by labels
  selectmove - enabling pieces to be placed using the mouse
  statusrules - interpreting the current position
  with reference to the rules of noughts-and-crossed
 */

showObservables("X|O|X|O|X|O");
```
Geometry
• lines of the grid (NESE)
• endpoints of lines (NW)
• locations for tokens (sq4)
• size of the grid (size)

Winning lines
• eight winning triples (lin6)
Modelling the OXO grid

```plaintext
WN is Point(100+size*0.5, 100+size*0.5);
NW is Point(100+size*2.5, 100+size*0.5);
NE is Point(100+size*2.5, 100+size*2.5);
SE is Point(100+size*0.5, 100+size*2.5);

sq1 is Point(100+size*1.5, 100+size*0.5);
sq2 is Point(100+size*3.5, 100+size*0.5);
sq3 is Point(100+size*3.5, 100+size*2.5);
sq4 is Point(100+size*1.5, 100+size*2.5);

NSW is Line(["x"],NW,"[y]",SW,"[x]",&SW,"[y]");
NNE is Line(["x"],NE,"[y]",NESE,"[x]",&NESE,"[y]");
gridlines is [WN, WSE, NSW, NESE];
picture is gridlines;
size = 40;

all squares is [s1,s2,s3,s4,s5,s6,s7,s8,s9];
nof squares is all squares#;

lin1 is [s1,s2,s3];
lin2 is [s4,s5,s6];
lin3 is [s7,s8,s9];
lin4 is [s1,s4,s7];
lin5 is [s2,s5,s8];
lin6 is [s3,s6,s9];
lin7 is [s1,s5,s9];
lin8 is [s3,s5,s7];

all lines is [lin1,lin2,lin3,lin4,lin5,lin6,lin7,lin8];
```
Observing the board state

**Status**
- current position (boardstate)
- tokens on squares (x / o / u)

**Board content**
- visual token (lab5)
- located visual token (piece5)

**boardstate:** [o,x,o,u,x,u,x,o,u]

**x / o / u:** 1 / -1 / 0

**lab5:** “X”

**piece5:** “X” at sq5
The state of the OXO grid

boardstate = [o,x,o,u,x,u,x,o,u];
Modelling players’ agency

Select move

• whose turn is it?
  (chooseplayer / chooseplayer_value / player )
• where on board to play?
  (mouseXnear2 / mouseYnear1)
• which square?
  (mouseXselect / mouseYselect)

chooseplayer
chooseplayer_value
mouseXnear2
mouseYnear1

(player is the current player : O)

(mouseX, mouseY)
(mouseXselect, mouseYselect)
Placing Os and Xs on grid

chooseplayer

chooseplayer_value

mouseXnear2

mouseYnear1

(mouseX, mouseY)

(mouseXselect, mouseYselect)

player is the current player : O
## Maker-defined functions

```
## illustrating the use of functions

func nofpieces {  
    para p1, p2;     p1 is a list of squares 
    auto count, total; p2 is a ‘o’ or ‘x’
    total = 0;
    for (count=1; count <= p1#; count++)
        if (p1[count] == p2) 
            total++;
    return total;
}
```

Total number of Xs on the grid (nofx)

```
nofx is nofpieces(allsquares, x);
nofo is nofpieces(allsquares, o);
full is (nofx + nofo == nofsquares);
```
Using the **with-construct**

```ruby
# illustrating the with construct

xwonI is (alllines[_i][1] == x) && (alllines[_i][2] == x) && (alllines[_i][3] == x);
xonIs xwonI with _i is 1..alllines#;

ownI is (alllines[_i][1] == o) && (alllines[_i][2] == o) && (alllines[_i][3] == o);
ownIs ownI with _i is 1..alllines#;

xwon is positionInList(true, xwonIs) != ø;
owon is positionInList(true, ownIs) != ø;
```

"X has a winning line"

"Has X or O won the game? == ‘Is there a winning line for X or O?’"

"The line indexed by _i is a winning line for X"

"The line for which _i is 8"
Interpreting the rules

- p1 is a list of squares
- p2 is a ‘o’ or ‘x’

“X has a winning line”

“The _i-th line is a winning line for X”

Total number of Xs on the grid

“The line for which _i is 8”
O seeks a winning move

• Is the game over?
  end_of_game is owon || xwon || draw;

• Does line lin have just 2 Os on it?

• Does the line with index ix have just 2 Os on it?
  winlineix is lin_w with lin is alllines[ix];

• Does any line have just 2 Os on it?
  winlines is winlineix with ix is 1..8;

• Where is there an empty square on line lin?
  gapinlin is 1 if lin[1]==0 else (2 if lin[2]==0 else (3 if lin[3]==0 else 0));

• Where the line with index ix has a space ... where each line has a space
  playonlinix is gapinlin with lin is alllines[ix];
  playonlines is playonlinix with ix is 1..8;
O makes a winning move

Register the winning lines as sets of indices of cells on the grid, not via their contents
alllinesindices is alllines

with s1 is 1, s2 is 2, s3 is 3, s4 is 4, s5 is 5, s6 is 6, s7 is 7, s8 is 8, s9 is 9;

• The index of the line with index _index if it is a winning line for O
  winindex is _index if winlines[_index] else 0;

• The set of indices of line that are winning lines for O – possibly empty
  iswinindex is winindex with _index is 1..8;

• An index of a winning line for O, if there is one
  wline is max(iswinindex);

O is to play         there is a winning line         the game isn’t over
when ((player==o) &&     wline>0                  &&     !end_of_game       ) {
  boardstate[ alllinesindices [wline] [playonlines[wline]] ] = o;
}

index of winning line    index of gap on winning line
Reflections on the MCE

• Tension between ...
  bricolage vs. eliminate redundancy, conform to standards, clarify through abstraction
  cf. functions and procedures 'objective' with's and when's 'agent-oriented'

• Challenges
  ? Project Manager interface
  ? intuitive way to represent with's
Acknowledgments

Nick Pope, Elizabeth Hudnott, Joe Butler, Tim Monks: The JS-Eden environment
Simon Gardner: The original OXO laboratory (1999)
Mike Joy: The first OXO construal prototype (1994)

References

Thank you
Any questions?