Tic Tac Toe

Qianji Zheng and J. Nelson Rushton, Texas Tech University

This is an LED program that defines a simple tic tac toe game to run in the easel game engine.

The game begins with an empty gridTic Tac Toe and it being x's turn to move. When an empty cell is clicked, the player whose turn it occupies that cell and it becomes the other player's turn, until the game is over. When the game is over a message is displayed giving the result of the game. The player can press the 'reset' button at any time to restart the game.

1 Data model

A *player* is either x or o. In this program, the variable p will range over players.

A *cell* is an integer in $\{1..9\}$. In this program, the variable c will range over cells.

Cells represent squares on the tic tac toe board as pictured below:

```
1|2|3
-----
4|5|6
-----
7|8|9
```

A *move* is a pair (p, c) where p is a player and c is a cell. The move (p, c) represents a move by player p in cell c. In this program, the variable $m4$ will range over moves.

A *state* is a set of moves, thought of as the set of moves made far in the game. In this program, the variable S will range over states.

Tell the translator to produce a SequenceL file that runs with Easel: **ledGame** The following are recognized as constant symmbols in the Easel environment

1. *currentState* (same type as user-defined *initialState*)

2. *mouseClicked* (boolean, true if the left mouse button is clicked in the current frame)

3. *mouseX*, *mouseY*: integer

2 Game rules

This section defines the rules of tic-tac-toe in LED.

Since the state of the game is the set of moves that have been made, the beginning state *initialState* is the empty set.

$$initialState \ := \ \varnothing \tag{1}$$

Player p *occupies* cell c if the move (p, c) is a member of *currentState*. Cell c is *occupied* if it is occupied by x or by o.

$$occupies\,(p, c) \ \equiv \ ((p, c)) \ \in (currentState) \tag{2}$$

$$occupied\,(c) \ \equiv \ (occupies\,(`x, c)) \vee (occupies\,(`o, c)) \tag{3}$$

A *row* is a set of cells that form three in a row either horizontally, vertically, or diagonally.

$$rows \ := ((hRows) \cup (vRows)) \cup (diagonals) \tag{4}$$
where $(((hRows) = (\{\{1,2,3\}, \{4,5,6\}, \{7,8,9\}\})) \wedge ((vRows) = (\{\{1,4,7\}, \{2,5,8\}, \{3,6,9\}\}))) \wedge ((diagonal$

threeInRow(p) if player p occupies all of the cells in some row.

$$threeInRow\,(p) \ \equiv \exists(R) \ \in (rows) \ . \ \forall(c) \ \in (R) \ . \ occupies\,(p, c) \tag{5}$$

boardFull if all cells are occupied.

$$boardFull \ \equiv (|currentState|) \ = (9) \tag{6}$$

gameOver if either the board is full, or one of the players three in a row.

$$gameOver \;\equiv\; ((boardFull) \vee (threeInRow\,(`x))) \vee (threeInRow\,(`o)) \tag{7}$$

playerToMove is x if an even number of moves have been made, and o otherwise.

$$playerToMove \;:=\; \begin{cases} `x & \text{if } even\,(|currentState|) \\ `o & \text{otherwise} \end{cases} \tag{8}$$

$$even\,(n) \;\equiv\; ((n) \mod (2)) \;= (0) \tag{9}$$

legalToMoveIn(c) means that it is legal for the player whose turn it is to move in cell c in the current state of the game — that is, if the game is not over and the cell is not occupied.

$$legalToMoveIn\,(c) \;\equiv\; (\neg occupied\,(c)) \wedge (\neg gameOver) \tag{10}$$

3 Video output

The default color used in this game is $BLACK$.

$$BLACK \;:=\; color\,(0, 0, 0) \tag{11}$$

$$WHITE \;:=\; color\,(255, 255, 255) \tag{12}$$

$$BLUE \;:=\; color\,(0, 0, 255) \tag{13}$$

$$GREEN \;:=\; color\,(0, 255, 0) \tag{14}$$

$$RED \;:=\; color\,(255,0,0) \tag{15}$$

The purpose of this section is to define the *display* function, which specifies the images to display on the screen in each game state. The *gridDisplay* consists is a set of four line segments that make up the tic tac toe playing area.

$$gridDisplay \;:=\; \{L1, L2, L3, L4\} \tag{16}$$
where $((((L1) = (segment\,(point\,(200,700)\,, point\,(200,400)\,, BLACK))) \wedge ((L2) = (segment\,(point\,(300,700)\,, p$

The default font size for displayed text in this program is 36.

$$fontSize \;:=\; 36 \tag{17}$$

$centerX(c)$ and $centerY(c)$ are the x and y coordinates of the center of cell c, respectively.

$$centerX\,(c) \;:=\; (150) + ((100) \times (((c) - (1)) \mod (3))) \tag{18}$$

$$centerY\,(c) \;:=\; (650) - ((100) \times (\lfloor((c) - (1)) \;/\; (3)\rfloor)) \tag{19}$$

$cellDisplay(c)$ is a display of a text character 'x' or an 'o' in cell c, or the empty display, respectively in case cell c is occupied by x, occupied by o, or not occupied in the current game state.

$$xImage\,(c) \;:=\; text\,(\texttt{"x"}, point\,(centerX\,(c)\,, centerY\,(c))\,, fontSize, BLUE) \tag{20}$$

$$oImage\,(c) \;:=\; text\,(\texttt{"o"}, point\,(centerX\,(c)\,, centerY\,(c))\,, fontSize, GREEN) \tag{21}$$

$$cellDisplay\,(c) \;:=\; \begin{cases} \{xImage\,(c)\} & \text{if } ((\text{`}x, c)) \in (currentState) \\ \{oImage\,(c)\} & \text{if } ((\text{`}o, c)) \in (currentState) \\ \varnothing & \text{otherwise} \end{cases} \tag{22}$$

cellDisplays is the set of all images of x's and o's on the board in the current state.

$$gameBoard \ := \ \{1 \ .. \ 9\} \tag{23}$$

$$cellDisplays \ := \bigcup_{(c) \ \in \ (gameBoard)} cellDisplay\,(c) \tag{24}$$

If the game is not over, *currentPlayerDisplay* is a text in the upper left hand region of the game window indicating the player to move, either “play x's turn” or “play o's turn”,

$$currentPlayerDisplay \ := \begin{cases} \{text\,(\texttt{"x's turn"}, point\,(100, 750)\,, fontSize, BLACK)\} & \text{if } (playerToMove) = (\text{`} \\ \{text\,(\texttt{"o's turn"}, point\,(100, 750)\,, fontSize, BLACK)\} & \text{otherwise} \end{cases} \tag{25}$$

The *restart button* consists of a rectangle around a “restart” text, displayed in the upper right region of the screen. Formally, it is a set of four line segments and a text.

$$restartLeft \ := \ 350 \tag{26}$$

$$restartRight \ := \ 550 \tag{27}$$

$$restartBottom \ := \ 725 \tag{28}$$

$$restartTop \ := \ 775 \tag{29}$$

$$restartBottomLeftPoint \ := \ point\,(restartLeft, restartBottom) \tag{30}$$

$$restartBottomRightPoint \ := \ point\,(restartRight, restartBottom) \tag{31}$$

$$restartTopLeftPoint \;:=\; point\,(restartLeft, restartTop) \tag{32}$$

$$restartTopRightPoint \;:=\; point\,(restartRight, restartTop) \tag{33}$$

$$mid\,(a, b) \;:=\; ((a) + (b)) \;/\; (2) \tag{34}$$

$$restartMidX \;:=\; mid\,(restartLeft, restartRight) \tag{35}$$

$$restartMidY \;:=\; mid\,(restartBottom, restartTop) \tag{36}$$

$$restartButton \;:=\; \{A1, A2, A3, A4, txt\} \tag{37}$$
where $(((((A1) = (segment\,(restartBottomLeftPoint, restartBottomRightPoint, BLACK))) \wedge ((A2) = (segm$

If the game is over, *gameResultDisplay* is a display in the upper left region of the screen telling the outcome of the game, either “x won”, ”o won”, or “cat got it”

$$gameResultDisplay \;:=\; \begin{cases} \{text\,(\texttt{"x won"}, point\,(200, 750)\,, fontSize, BLUE)\} & \text{if } threeInRow\,(`x) \\ \{text\,(\texttt{"o won"}, point\,(200, 750)\,, fontSize, GREEN)\} & \text{if } threeInRow\,(`o) \\ \{text\,(\texttt{"cat got it"}, point\,(200, 750)\,, fontSize, RED)\} & \text{otherwise} \end{cases} \tag{38}$$

The “display” is the screen display for the current game state. The grid, cell displays, and restart button are always displayed. Additionally, the display includes the game results if the game is over, and the player to move if the game is not over.

$$images \;:=\; \begin{cases} gameOverDisplay & \text{if } gameOver \\ inPlayDisplay & \text{otherwise} \end{cases} \tag{39}$$
where $(((alwaysDisplay) = (((gridDisplay) \cup (cellDisplays)) \cup (restartButton))) \wedge ((inPlayDisplay) = ((al$

4 Mouse input

This section defines *update*, which specifies the program's response to mouse input. The variable *pt* will vary over points.

$xMin(c)$, $xMax(c)$, $yMin(c)$, and $yMax(c)$ denote the graphical boundaries of cell c, in the obvious manner.

$$xMin\,(c) \;:=\; (100) + ((100) \times (((c) - (1)) \mod (3))) \tag{40}$$

$$xMax\,(c) \;:=\; (200) + ((100) \times (((c) - (1)) \mod (3))) \tag{41}$$

$$yMin\,(c) \;:=\; (600) - ((100) \times (\lfloor((c) - (1)) \;/\; (3)\rfloor)) \tag{42}$$

$$yMax\,(c) \;:=\; (700) - ((100) \times (\lfloor((c) - (1)) \;/\; (3)\rfloor)) \tag{43}$$

$cellClicked(c)$ means that cell c has been clicked.

$$cellClicked\,(c) \;\equiv ((((mouseClicked) \wedge ((mouseX) > (xMin\,(c)))) \wedge ((mouseX) < (xMax\,(c)))) \wedge ((mouseY) > \tag{44}$$

$restartClicked$ means that the most recent mouse click is inside the region of the play again button.

$$restartClicked \;:= ((((mouseClicked) \wedge ((mouseX) > (restartLeft))) \wedge ((mouseX) < (restartRight))) \wedge ((mo \tag{45}$$

$moveMade(c)$ means that cell c has been clicked and the player to move may legally move there.

$$moveMadeIn\,(c) \;\equiv\; (cellClicked\,(c)) \wedge (legalToMoveIn\,(c)) \tag{46}$$

$$movesMade \;:=\; \{(playerToMove, c) \mid ((c) \in (gameBoard)) \wedge (moveMadeIn\,(c))\} \tag{47}$$

$$newState \;:=\; \begin{cases} initialState & \text{if } restartClicked \\ (currentState) \cup (movesMade) & \text{otherwise} \end{cases} \tag{48}$$

5 Test cases

This section contains test cases for the SequenceL interpreter.

```
cmd:>pp(initialState_)
"{}"
cmd:>pp(rows)
"{{1, 2, 3}, {4, 5, 6}, {7, 8, 9}, {1, 4, 7}, {2, 5, 8}, {3, 6, 9}, {1, 5, 9}, {3, 5, 7}}"
cmd:>pp(BLACK)
"an Easel color"
cmd:>pp(WHITE)
"an Easel color"
cmd:>pp(BLUE)
"an Easel color"
cmd:>pp(GREEN)
"an Easel color"
cmd:>pp(RED)
"an Easel color"
cmd:>pp(gridDisplay)
"{an Easel image, an Easel image, an Easel image, an Easel image}"
cmd:>pp(fontSize)
"36"
cmd:>pp(gameBoard)
"{1, 2, 3, 4, 5, 6, 7, 8, 9}"
cmd:>pp(restartLeft)
"350"
cmd:>pp(restartRight)
"550"
cmd:>pp(restartBottom)
"725"
cmd:>pp(restartTop)
"775"
cmd:>pp(restartBottomLeftPoint)
"an Easel point"
cmd:>pp(restartBottomRightPoint)
"an Easel point"
cmd:>pp(restartTopLeftPoint)
"an Easel point"
cmd:>pp(restartTopRightPoint)
"an Easel point"
cmd:>pp(restartMidX)
"450"
```

```
cmd:>pp(restartMidY)
"750"
cmd:>pp(restartButton)
"{an Easel image, an Easel image, an Easel image, an Easel image, an Easel image}"
```

LED engine took: 20 secs.