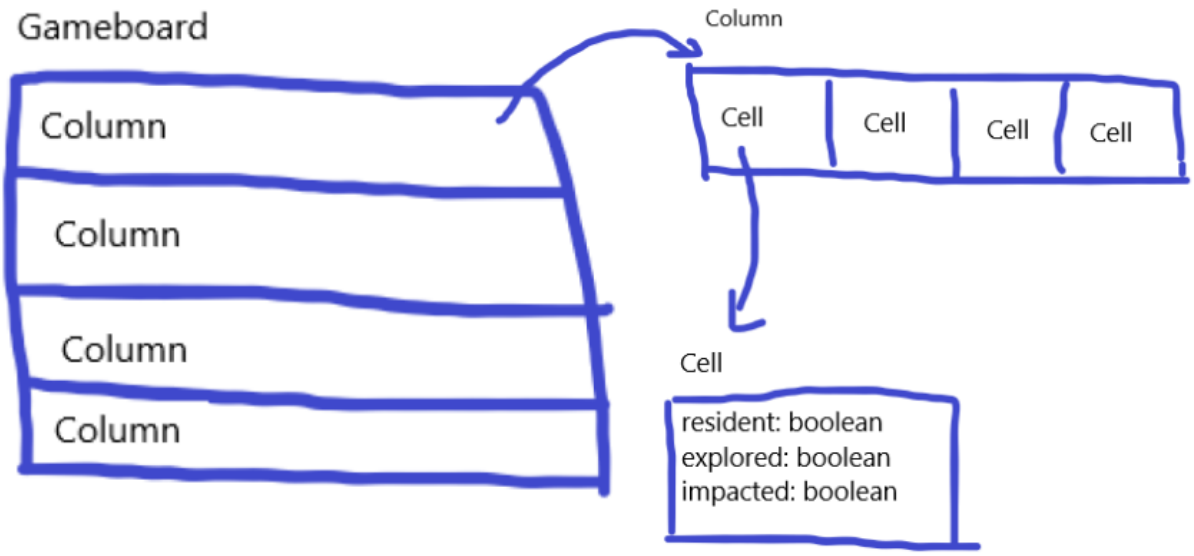
Kuncheng Feng
CSC 466

Progress presentation #1 - Hello World

What has been implemented?

Currently, 3 models have been created in CLOS, cell, column, and board.

- The board object contains a list of column objects.
- The column object contains a list of cell objects.
- The cell object contains a few values that will be useful in the future.



The methods to create and display them have also been implemented as well.

Cell

Demo

```
> (demo--displayCells)
Demonstrating: displayCells
5 empty cells:
|   |   |   |   |   |
5 explored cells:
| x | x | x | x | x |
5 impacted cells:
| o | o | o | o | o |
NIL
```

Code

```
(defclass cell()
	(
		(resident :accessor cell-resident :initarg :resident :initform nil)
		(explored :accessor cell-explored :initarg :explored :initform nil)
		(impacted :accessor cell-impacted :initarg :impacted :initform nil)
	)
)

(defmethod newCell()
	(make-instance 'cell
		:resident nil
		:explored nil
		:impacted nil
	)
)

(defmethod newCells(number)
	(cond
		((<= number 0)
			(list)
		)
		(t
			(cons (newCell) (newCells (- number 1)))
		)
	)
)

; Note: it does not display any ship yet
(defmethod displayCell((c cell))
	(cond
		((equal (cell-impacted c) t)
			(format t "| o ")
		)
		((equal (cell-explored c ) t)
			(format t "| x ")
		)
		(t
			(format t "|   ")
		)
	)
)
```

```
(defmethod displayCells((cs list))
	(cond
		((not (equal cs nil))
			(displayCell (first cs))
			(displayCells (rest cs))
		)
		(t
			(format t "|~%")
		)
	)
)


; ---------------------- Demo ---------------------
(defmethod demo--displayCells(&aux cells)
	(setf cells (newCells 5))
	(format t "Demonstrating: displayCells ~%")
	(format t "5 empty cells: ~%")
	(displayCells cells)

	(dotimes (n 5)
		(setf (cell-explored (nth n cells)) t)
	)
	(format t "5 explored cells: ~%")
	(displayCells cells)

	(dotimes (n 5)
		(setf (cell-impacted (nth n cells)) t)
	)
	(format t "5 impacted cells: ~%")
	(displayCells cells)
)
```

Column

Demo

```
[]> (demo--displayColumns)
Demonstrating: displayColumns
An column of width 2:
+---+---+
|   |   |
+---+---+
An column of width 5:
+---+---+---+---+---+
|   |   |   |   |   |
+---+---+---+---+---+
10 columns of width 10:
+---+---+---+---+---+---+---+---+---+---+
|   |   |   |   |   |   |   |   |   |   |
+---+---+---+---+---+---+---+---+---+---+
|   |   |   |   |   |   |   |   |   |   |
+---+---+---+---+---+---+---+---+---+---+
|   |   |   |   |   |   |   |   |   |   |
+---+---+---+---+---+---+---+---+---+---+
|   |   |   |   |   |   |   |   |   |   |
+---+---+---+---+---+---+---+---+---+---+
|   |   |   |   |   |   |   |   |   |   |
+---+---+---+---+---+---+---+---+---+---+
|   |   |   |   |   |   |   |   |   |   |
+---+---+---+---+---+---+---+---+---+---+
|   |   |   |   |   |   |   |   |   |   |
+---+---+---+---+---+---+---+---+---+---+
|   |   |   |   |   |   |   |   |   |   |
+---+---+---+---+---+---+---+---+---+---+
|   |   |   |   |   |   |   |   |   |   |
+---+---+---+---+---+---+---+---+---+---+
|   |   |   |   |   |   |   |   |   |   |
+---+---+---+---+---+---+---+---+---+---+
NIL
```

Code

```
(defclass column()
        (
                (cells :accessor column-cells :initarg :cells :initform nil)
        )
)
```

```
(defmethod newColumn(width)
	(make-instance 'column
		:cells (newCells width)
	)
)

(defmethod newColumns(width height)
	(cond
		((<= height 0)
			(list)
		)
		(t
			(cons (newColumn width) (newColumns width (- height 1)))
		)
	)
)

; This is set as global variable for convenience
(setf cellLength 0)

(defmethod displayLine()
	(dotimes (n cellLength)
		(format t "+---")
	)
	(format t "+~%")
)

(defmethod displayColumn((c column) &aux cells)
	(setf cells (column-cells c))
	(setf cellLength (length cells))
	(displayLine)
	(displayCells cells)
)

(defmethod displayColumns((cs list))
	(cond
		((not (equal cs nil))
			(displayColumn (first cs))
			(displayColumns (rest cs))
		)
		(t
			(displayLine)
		)
	)
)


; ---------------------- Demo ------------------
(defmethod demo--displayColumns()
	(format t "Demonstrating: displayColumns ~%")
	(format t "An column of width 2: ~%")
	(displayColumns (newColumns 2 1))
	(format t "An column of width 5: ~%")
	(displayColumns (newColumns 5 1))
	(format t "10 columns of width 10: ~%")
	(displayColumns (newColumns 10 10))
)
```

Board

Demo

There is no demo at the moment 😕, I didn't feel the need to yet as it look exactly like the column demo.

Code

```
(defclass board()
	(
		(columns :accessor board-columns :initarg :columns)
	)
)

(defmethod newBoard(width height)
	(make-instance 'board
		:columns (newColumns width height)
	)
)

(defmethod displayBoard((b board))
	(displayColumns (board-columns b))
)
```