Grocery Tracker

New Mexico
Supercomputing Challenge
Final Report
April 05, 2010

Team 53
Los Alamos Homeschool

Team Members:
Ariel Koh
Aline Parliman

Teacher:
Dr. Aik-Siong Koh

Project Mentor:
Dr. Aik-Siong Koh

Table of Contents

Executive Summary:

When going to the grocery store, it is often a challenge to find groceries without walking around the store more than once. We made a program that will locate groceries and calculate one of the shortest routes to retrieve them. Our program uses Squeak Smalltalk, an object-oriented programing language. This problem contains elements from the Traveling Salesman and Shortest Path Problem.

To make the program, we did Squeak tutorials until we became comfortable with Squeak. After that we created our store and programmed our pathfinder.

It is well known that the exact answer to the Traveling Salesman problem requires O(n!) calculations, where n is the number of items to pickup. To save computer power and time we used a heuristic (i.e. rule of thumb) of O(n) to give a good path to take.

Using the layout of our local store and the locations of items and intersections of aisles, our heuristic retrieves the item with the shortest direct distance from the current intersection first. To reach that item, the heuristic picks the edge emanating from the current intersection whose other end is closest to the target item.

After calculating a path, our program displays a bird's eye view of a store and a path going from the entrance of the store to all the items requested. While our path is not always the perfect path, our program does makes shopping more efficient. It shows a shopper unfamiliar with a store where items are, and even experienced shoppers can find that they save steps. Our program can be used by delivery people or workers in stores and libraries.

Introduction:

When going to the grocery store, it is often a challenge to find groceries without wasting time visiting the same aisle more than once. Our program will locate a list of groceries and calculate one of the shortest routes to retrieve them.

- *Nearly two-thirds of consumers surveyed said difficulty locating items was a top reason they quit shopping or shopped less frequently at a particular store.*

TreoSystems' iPAL product locator webpage: http://www.treosys.com/ipalsuite.htm

Description:

Our store contains aisles, intersections and products which are represented by edges, nodes and points.

Our program models a store near us. We drew a store layout and roughly modeled the placement of 134 items. The number of items in our store is limited by the time it takes to enter the data. Theoretically, we could place a very large number of items in our store.

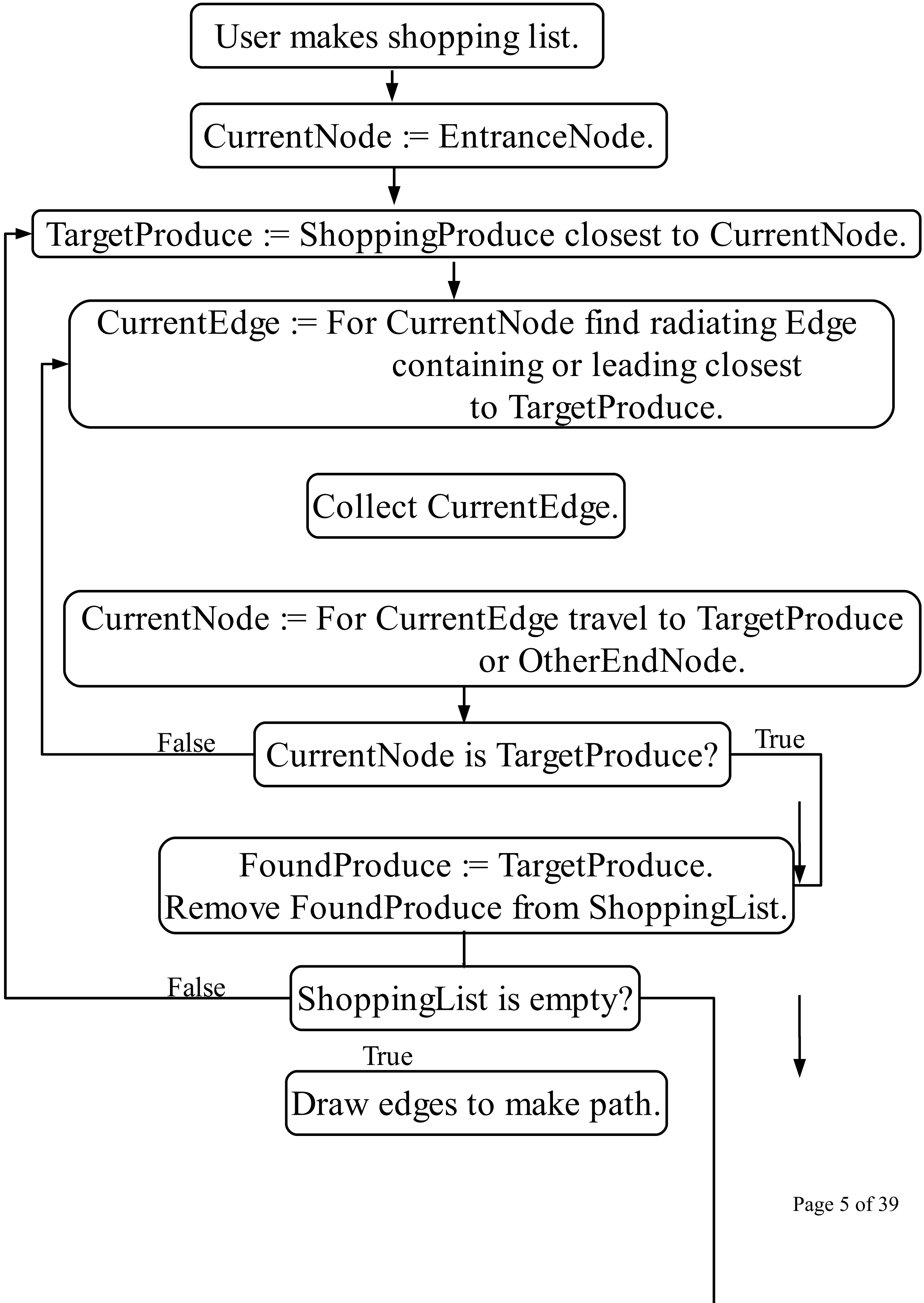
The pathfinder we made starts with setting the entrance as its current node. It then calculates the direct distance to all requested items from the current node. It chooses the closest item to head to.

The program tests to see if the target item is on any of the edges radiating from the current node. If the item is there, the pathfinder picks up the item and then starts over choosing the closest item to retrieve next with the item location as current node. If the item is not there, it looks at the opposite end of each available edge and checks which end has the shortest direct distance to the item. It then takes the edge containing that end and sets that end to be the current node. The pathfinder repeats this sequence until it finds the target item.(See Figure 1).

Our program shows a bird's eye view of a store. The user lists the items he/she wants, and a 'shortest' path will appear going from the entrance of the store to all the items listed. Our program makes shopping more efficient.

Figure 1

Main Algorithm



Scope:

The scope of our project was to create a computer program that will locate groceries on a store map and calculate one of the shortest routes to retrieve them. Our program always gives a good solution but not necessarily the perfect one. (see Mathematical Model)

Procedure:

We accomplished our project by doing the following:

- Using Squeak Smalltalk tutorials until we became familiar with programming
- Creating a store layout consisting of nodes, edges, and products
- Making a dialog where a user can search by item names and compose a shopping list
- Constructing a window and drawing the store map in it
- Showing items in the shopping list on a store map
- Programing a pathfinder to give a 'shortest' route to retrieve the items

Mathematical Model:

Our problem is similar to the Traveling Salesman Problem (TSP). The TSP algorithm would take a very long time to calculate the perfect path for a large number of items (n), because TSP looks at (n-1)!/2 number of paths. Our program only looks at a few individual edges not whole paths between items (see Description). Our program is much quicker at finding a good path but does not always give the perfect solution because it uses a heuristic to solve the problem. Our heuristic is taking the direct distance to find the item to retrieve first and then using the direct distance from the opposite end of the available edges from a node to see which path to take (see Description). Our problem differs from the Traveling Salesman Problem in that the salesman can travel directly from one location to the next. In our program the user must follow set pathways (store aisles). Therefore, our problem also includes the Shortest Path Problem (SPP). Fortunately SPP (O(n)) is easier to solve compared to TSP (O(n!)). Even so, exact SPP using Dijkstra's Algorithm seems unnecessary because it uses the entire store's information to find the shortest distance between every item to the next.

Using our heuristic, the calculation needed to go from one item to the next is proportional to the number of nodes traversed and the number of edges radiating out of each node. The number of nodes traversed can vary from 1 to 24, corresponding to items on the same aisle and items on opposite corners of the store. On average we can assume 10. The number of radiating edges per node can vary from 1 to 4, corresponding to the entrance and a crossing. On average we can assume 3. Therefore, the number of calculations is proportional to 10*3*n. Our program is O(n). Although Dijkstra's Algorithm is also O(n), the constant for it is proportional to the total number of store nodes squared (49^2 or 2401) which is a lot larger than 10*3 or 30.

Results:

We started our project in June 2009 and have spent roughly 100 hours on it so far. The most obvious truth we have discovered is that user-friendly software has taken many hours to make and needs constant upgrades.

Our first 30 hours were spent on the tutorials learning Squeak Smalltalk. Initially, we chose Squeak based on our mentor's recommendation. Over time we have come to see its advantages for being one of the most developed object-oriented programming languages. In object-oriented programming(C#, C^{++}) everything is an object (noun) and you tell an object to do something (verb), whereas in procedural programming(C) everything is a function (verb) that acts on data (noun). In procedural programming you could use the jump function on a chair data and the program would accept the inappropriate data and probably crash. In object-oriented programming, each object has certain actions it can do, so you cannot tell a chair to jump.

In late December, we began writing our program. Putting aisles in our store diagram proved challenging and time consuming. Our first effort only had two aisles, after finishing our pathfinder, we made it four aisles. Finally, this March we updated it to contain the 17 aisles our local store actually has.

We have also learned that communicating technical information to a lay audience is a challenge. Technical terminoligy needs to be defined to the lay person (i.e. nodes → intersections of aisles, edges → aisles).

Good Route:

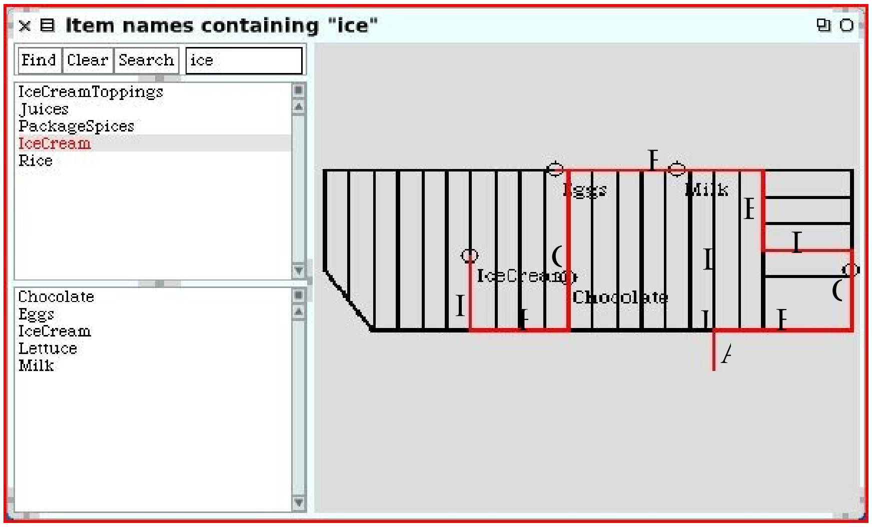


Figure 2

Under most circumstances our program shows a good route. For example, if you want lettuce, milk, eggs, chocolate, and ice cream, the pathfinder will start at the entrance and look to see which item is closest. It will see that the lettuce is closest so it will set off to get the lettuce first. It will travel up edge A until it comes to the end, then it will look and see if lettuce is on edge J, L or B. The pathfinder will see that lettuce is not there, but the endpoint of edge B is closer to lettuce than the other endpoint, so it will take edge B. The pathfinder will repeat this and come to edge C. At edge C, it will see that lettuce is there so it will go to the item and then look which unretrieved item is closest to lettuce and will select it to retrieve next. The pathfinder will see that milk is closer by the direct distance and thus get milk next. The pathfinder will continue to do this until it has retrieved all the items.

Problem Route:

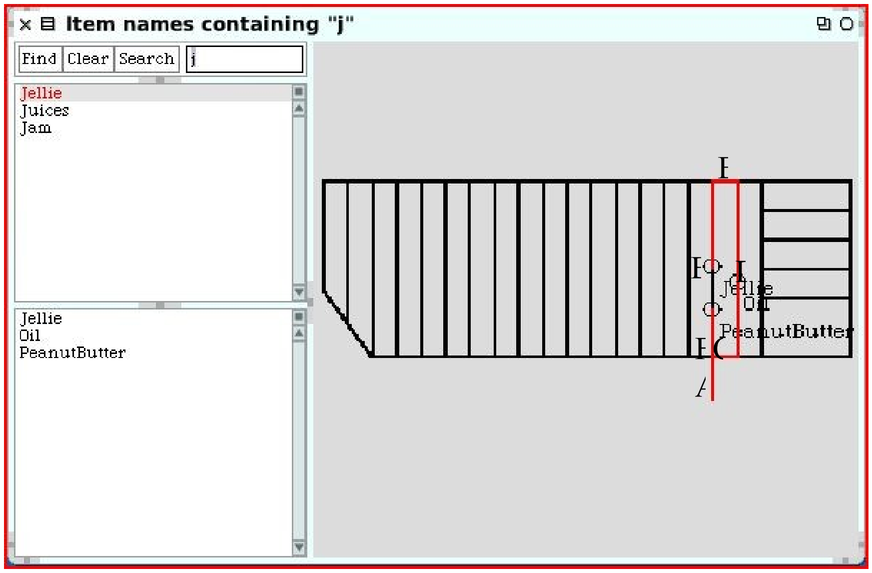


Figure 3

Our program does however have one very noticeable error. If you ask the program for peanut butter, jelly and oil, it will give you a longer route than necessary, but there is a reason for this. Our program does not understand aisles. To calculate the path, the pathfinder will start at the entrance and look to see which item is closest. The pathfinder will see that peanut butter is closest so it will set off to get the peanut butter first. It will travel up edge A until it comes to the end, then it will look and see if peanut butter is on edge B, C or F. It will see that peanut butter is on edge B so it will go to peanut butter and then determine which unretrieved item is closer to the peanut butter and will select it to retrieve next. The pathfinder will see that oil is closer by the direct distance and thus get oil first. It will look on edge B and see that oil is not there, but the endpoint at the bottom of edge B is closer to oil than the other endpoint, so it will go down. It will repeat this process till it reaches oil and then jelly. By going to get oil before jelly it travels over many edges twice, thus taking a longer route.

Conclusion:

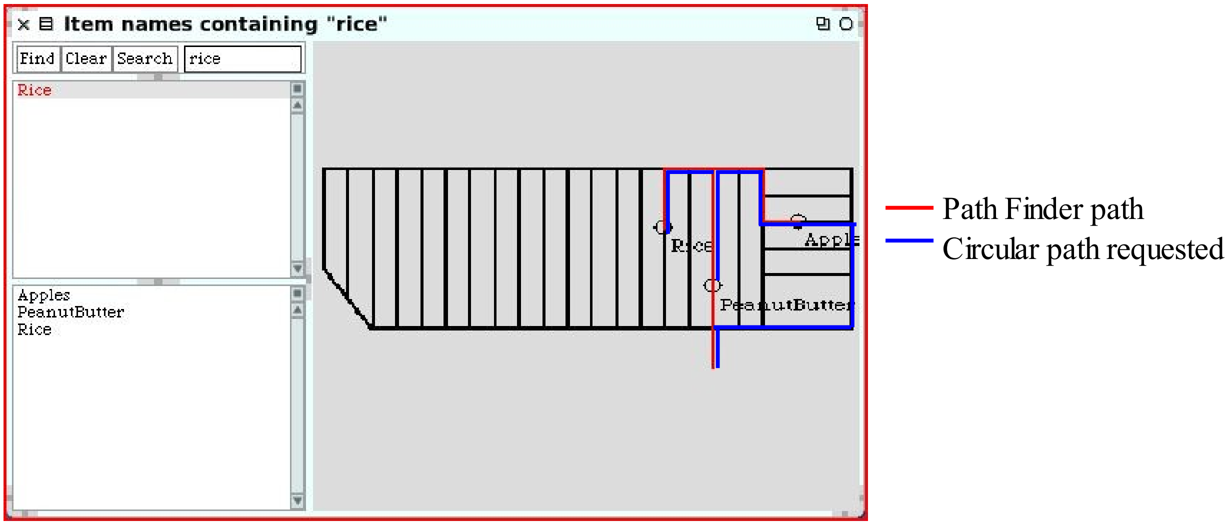


Figure 4

Our program works more often than not. Many people take a circular route through the store believing this to be the most efficient. To these people, our program may not seem advantageous, but we tested this theory and found our pathfinder saves shoppers steps. Above, the red route is the one our program calculated and the blue route is the one the shopper wanted to take. We measured both paths and found that our path saves you about 50 steps. We believe we have achieved our goal of creating a computer program to help people find their groceries quickly and efficiently at the store. Our program could be modeled for use in stores, libraries, and by delivery people.

Recommendations:

There are certainly concepts we could implement to make this project better. We could program Grocery Tracker to have priorities like getting the perishable products after the imperishable products have been retrieved. For example, if the grocery list was to consist of ice cream, chicken, pet food, and soap, the prioritized program would distinguish between the perishable products (i.e. ice cream and chicken) and the imperishable products (i.e. pet food and soap). It would choose to calculate the closest of the imperishable products for retrieval. After all the imperishable products have been retrieved, the program could then retrieve all the perishable products.

This program could also be prioritized to pick up the lightest products first and then the heavier ones as it went on, so that the amount of work used pushing the grocery cart would be minimized.

We could also optimize the pathfinder by taking two paths to a target product, instead of one, and measuring the distances for each path and picking the shorter one to take.

Also, we could have the program take into account the nodes of the targeted product's edge. Instead of the program measuring the direct distance between a node and the targeted product, the program would measure the direct distance from the current node to the closer of the target edge's nodes to calculate the path. We are unsure if this process would give a better route.

Acknowledgments:

We want to thank Dr. Aik-Siong Koh for teaching and showing us how to program in Smalltalk as we worked on Grocery Tracker. We are grateful for his encouragement.

We also want to thank Mrs. Tracy Koh, Mr. Bart Parliman, and Mrs. Grace Parliman for helping us write this report and providing feedback.

Truly, we have learned a lot from this experience, and we are grateful to everyone who has supported and encouraged us.\

Appendix:

Sample of Code:

```
initialize
        super initialize.
        self
           storeForm: (Form extent: 300 @ 300 depth: Display depth).
        self storeForm fillColor: Color lightGray lighter lighter lighter.
        shoppingList := SortedCollection new.
        aStore := ShopAppStore new.
        aStore fillWithNodes.
        aStore fillWithEdges.
        aStore fillWithProduce.
        aStore fillWithEntrance

inMorphicWindowWithInitialSearchString: initialString
        "Answer a morphic window with the given initial search string, nil if
        none "
        | window selectorListView firstDivider secondDivider horizDivider typeInPane
        searchButton plugTextMor findButton clearButton |
        window := (SystemWindow labelled: 'Shopping App')
                        model: self.
        aSystemWindow := window.
```

```
firstDivider := 0.07.
secondDivider := 0.5.
horizDivider := 0.50.
typeInPane := AlignmentMorph newRow vResizing: #spaceFill;
                    height: 14.
typeInPane hResizing: #spaceFill.
typeInPane listDirection: #leftToRight.
plugTextMor := PluggableTextMorph
                    on: self
                    text: #searchString
                    accept: #searchString:notifying:
                    readSelection: nil
                    menu: nil.
plugTextMor setProperty: #alwaysAccept toValue: true.
plugTextMor askBeforeDiscardingEdits: false.
plugTextMor acceptOnCR: true.
plugTextMor setTextColor: Color black.
plugTextMor setNameTo: 'Search'.
plugTextMor vResizing: #spaceFill;
   hResizing: #spaceFill.
plugTextMor hideScrollBarsIndefinitely.
plugTextMor setTextMorphToSelectAllOnMouseEnter.
searchButton := SimpleButtonMorph new target: self;
                    color: Color white;
                    label: 'Search';
                    actionSelector: #doSearchFrom:;
                    arguments: {plugTextMor}.
typeInPane addMorphFront: searchButton.
typeInPane addTransparentSpacerOfSize: 4 @ 0.
typeInPane addMorphBack: plugTextMor.
initialString isEmptyOrNil
  ifFalse: [plugTextMor setText: initialString].
clearButton := SimpleButtonMorph new target: self;
                    color: Color white;
                    label: 'Clear';
                    actionSelector: #doClearFrom:;
                    arguments: {nil}.
typeInPane addMorphFront: clearButton.
typeInPane addTransparentSpacerOfSize: 0 @ 0.
typeInPane addMorphBack: plugTextMor.
initialString isEmptyOrNil
  ifFalse: [plugTextMor setText: initialString].
findButton := SimpleButtonMorph new target: self;
                    color: Color white;
                    label: 'Find';
                    actionSelector: #doFindFrom:;
                    arguments: {plugTextMor}.
typeInPane addMorphFront: findButton.
```

```
typeInPane addTransparentSpacerOfSize: 0 @ 0.
typeInPane addMorphBack: plugTextMor.
initialString isEmptyOrNil
  ifFalse: [plugTextMor setText: initialString].
window
  addMorph: typeInPane
  frame: (0 @ 0 corner: horizDivider @ firstDivider).
selectorListView := PluggableListMorph
                  on: self
                  list: #selectorList
                  selected: #selectorListIndex
                  changeSelected: #selectorListIndex:
                  menu: #selectorListMenu:
                  keystroke: #selectorListKey:from:.
selectorListView menuTitleSelector: #selectorListMenuTitle.
window
  addMorph: selectorListView
  frame: (0 @ firstDivider corner: horizDivider @ secondDivider).
window
  addMorph: self buildMorphicMessageList
  frame: (0 @ secondDivider corner: horizDivider @ 1).
window
  addMorph: self buildSketchMorph
  frame: (horizDivider @ 0 corner: 1 @ 1).
initialString isEmptyOrNil
  ifFalse: [self searchString: initialString notifying: nil].
^ window
```

initialize

```
| xLength yLength |
super initialize.
xLength := 77.0 * 3.0.
yLength := 27.0 * 3.0.
dxVeg := 13.0 * 3.0.
dx := xLength - dxVeg / 18.0.
dy := yLength.
dy0 := yLength - (10.0 * 3.0).
dy1 := yLength - (5.0 * 3.0).
dyVeg := yLength / 6.0.
DrawingFactor := 290 / (xLength max: yLength).
DrawingOffset := 5 @ 80
```

fillStore

```
self createBackEdges.
self createFrontEdges.
self createAisles.
self createVegEdges.
self createEntranceEdge.
```

```
        self createNodesFromEdges.
        self createProduce.
```

addEdgeName: nameSymbol sx: startx sy: starty ex: endx ey: endy

```
        | startNode endNode aEdge |
        startNode := ShopAppNode
                name: (nameSymbol , 's') asSymbol
                x: startx
                y: starty.
        endNode := ShopAppNode
                name: (nameSymbol , 'e') asSymbol
                x: endx
                y: endy.
        aEdge := ShopAppEdge
                name: nameSymbol
                start: startNode
                end: endNode.
        cEdge at: nameSymbol put: aEdge.
        ^ aEdge
```

fillStore

```
        self createBackEdges.
        self createFrontEdges.
        self createAisles.
        self createVegEdges.
        self createEntranceEdge.
        self createNodesFromEdges.
        self createProduce.
```

createVegEdges

```
        self createLeftVegEdges.
        self createRightVegEdges.
        self createVegAisles
```

createAisles

```
        | startx endx starty endy nameSymbol |
        0
          to: 1
          do: [:i |
                startx := dx * i.
                endx := dx * i.
                starty := 0.0.
                endy := i = 0
                                ifTrue: [dy0]
                                ifFalse: [dy1].
                nameSymbol := ('aisle' , i printString) asSymbol.
                self
                        addEdgeName: nameSymbol
```

```
                    sx: startx
                    sy: starty
                    ex: endx
                    ey: endy].
    2
      to: 17
      do: [:i |
                startx := dx * i.
                endx := dx * i.
                starty := 0.0.
                endy := dy.
                nameSymbol := ('aisle' , i printString) asSymbol.
                self
                    addEdgeName: nameSymbol
                    sx: startx
                    sy: starty
                    ex: endx
                    ey: endy]
```

createFrontEdges

```
    | startx endx starty endy nameSymbol |
    1
      to: 2
      do: [:i |
                startx := dx * (i - 1).
                endx := dx * i.
                starty := i = 1
                                ifTrue: [dy0]
                                ifFalse: [dy1].
                endy := i = 1
                                ifTrue: [dy1]
                                ifFalse: [dy].
                nameSymbol := ('front' , i printString) asSymbol.
                self
                    addEdgeName: nameSymbol
                    sx: startx
                    sy: starty
                    ex: endx
                    ey: endy].
    3
      to: 18
      do: [:i |
                startx := dx * (i - 1).
                endx := dx * i.
                starty := dy.
                endy := dy.
                nameSymbol := ('front' , i printString) asSymbol.
                self
```

```
addEdgeName: nameSymbol
sx: startx
sy: starty
ex: endx
ey: endy]
```

createBackEdges

```
| startx endx starty endy nameSymbol |
1
  to: 18
  do: [:i |
      startx := dx * (i - 1).
      endx := dx * i.
      starty := 0.0.
      endy := 0.0.
      nameSymbol := ('back' , i printString) asSymbol.
      self
          addEdgeName: nameSymbol
          sx: startx
          sy: starty
          ex: endx
          ey: endy]
```

createEntranceEdge

```
| startx endx starty endy nameSymbol entranceEdge |
startx := dx * 16.
endx := dx * 16.
starty := dy.
endy := dy + 20.
nameSymbol := 'entrance' asSymbol.
entranceEdge := self
          addEdgeName: nameSymbol
          sx: startx
          sy: starty
          ex: endx
          ey: endy.
cEntrance add: entranceEdge end name
```

createLeftVegEdges

```
| startx endx starty endy nameSymbol |
1
  to: 5
  do: [:j |
      startx := dx * 18.
      endx := dx * 18.
      starty := dyVeg * (j - 1).
      endy := j = 5
                  ifTrue: [dy]
```

```
                              ifFalse: [dyVeg * j].
            nameSymbol := ('leftVeg' , j printString) asSymbol.
            self
                addEdgeName: nameSymbol
                sx: startx
                sy: starty
                ex: endx
                ey: endy]
```

createRightVegEdges

```
        | startx endx starty endy nameSymbol |
        1
          to: 5
          do: [:j |
            startx := dx * 18 + dxVeg.
            endx := dx * 18 + dxVeg.
            starty := dyVeg * (j - 1).
            endy := j = 5
                              ifTrue: [dy]
                              ifFalse: [dyVeg * j].
            nameSymbol := ('rightVeg' , j printString) asSymbol.
            self
                addEdgeName: nameSymbol
                sx: startx
                sy: starty
                ex: endx
                ey: endy]
```

createVegAisles

```
        | startx endx starty endy nameSymbol |
        0
          to: 5
          do: [:j |
            startx := dx * 18.
            endx := dx * 18 + dxVeg.
            starty := j = 5
                              ifTrue: [dyVeg * 6]
                              ifFalse: [dyVeg * j].
            endy := starty.
            nameSymbol := ('aisleVeg' , j printString) asSymbol.
            self
                addEdgeName: nameSymbol
                sx: startx
                sy: starty
                ex: endx
                ey: endy]
```

createNodesFromEdges

```
| aStartNode aEndNode nodecEdge |
cEdge
  do: [:aEdge |
        aStartNode := aEdge start.
        aEndNode := aEdge end.
        (cNode
                noneSatisfy: [:aNode | aNode equal: aStartNode tol: 0.1])
            ifTrue: [cNode add: aStartNode].
        (cNode
                noneSatisfy: [:aNode | aNode equal: aEndNode tol: 0.1])
            ifTrue: [cNode add: aEndNode]].
cNode
  do: [:aNode |
        nodecEdge := aNode cEdge.
        cEdge
            do: [:aEdge |
                (aNode equal: aEdge start tol: 0.1)
                    ifTrue: [nodecEdge add: aEdge.
                        aEdge start: aNode].
                (aNode equal: aEdge end tol: 0.1)
                    ifTrue: [nodecEdge add: aEdge.
                        aEdge end: aNode]]]
```

putProduceOnEdges

```
| foundEdge |
cProduce
  do: [:aProduce |
        foundEdge := cEdge
                    detect: [:aEdge | aProduce point
                            onLineFrom: aEdge start point
                            to: aEdge end point
                            within: 0.1].
        aProduce aEdge: foundEdge]
```

createProduce

```
self createProduceAisle0.
self createProduceAisle1.
self createProduceAisle2.
self createProduceAisle3.
self createProduceAisle4.
self createProduceAisle5.
self createProduceAisle6.
self createProduceAisle7.
self createProduceAisle8.
self createProduceAisle9.
self createProduceAisle10.
self createProduceAisle11.
self createProduceAisle12.
```

```
self createProduceAisle13.
self createProduceAisle14.
self createProduceAisle15.
self createProduceAisle16.
self createProduceAisle17.
self createBackEdges4.
self createBackEdges9.
self createBackEdges14.
self createVegAisles1.
self createVegAisles2.
self createVegAisles3.
self createVegAisles4.
self createVegAisles5.
self createLeftVegEdges2.
self createLeftVegEdges3.
self createLeftVegEdges5.
self createRightVegEdges1.
self createRightVegEdges2.
self createRightVegEdges3.
self createRightVegEdges4.
self putProduceOnEdges
```

createProduceAisle0

```
| block aProduce |
block := [:produceName :x :y |
        aProduce := ShopAppProduce
                        name: produceName
                        x: x
                        y: y.
        cProduce add: aProduce].
block
  value: #Chicken
  value: 0.0
  value: 10.0.
block
  value: #Beef
  value: 0.0
  value: 20.0.
block
  value: #Bread
  value: 0.0
  value: 30.0.
block
  value: #Fish
  value: 0.0
  value: 40.0.
```

createProduceAisle1

```
| block aProduce |
block := [:produceName :x :y |
        aProduce := ShopAppProduce
                        name: produceName
                        x: x
                        y: y.
        cProduce add: aProduce].
block
  value: #LightBulbs
  value: dx
  value: 11.0.
block
  value: #Cookery
  value: dx
  value: 22.0.
block
  value: #Utensils
  value: dx
  value: 33.0.
block
  value: #School
  value: dx
  value: 44.0.
block
  value: #LaundryBaskets
  value: dx
  value: 55.0
```

createProduceAisle2

```
| block aProduce |
block := [:produceName :x :y |
        aProduce := ShopAppProduce
                        name: produceName
                        x: x
                        y: y.
        cProduce add: aProduce].
block
  value: #CatFood
  value: dx * 2
  value: 11.0.
block
  value: #Automotive
  value: dx * 2
  value: 22.0.
block
  value: #PetSupplies
  value: dx * 2
  value: 33.0.
```

```
		block
		   value: #PetFood
		   value: dx * 2
		   value: 44.0
```

createProduceAisle3

```
		| block aProduce |
		block := [:produceName :x :y |
			aProduce := ShopAppProduce
						name: produceName
						x: x
						y: y.
			cProduce add: aProduce].
		block
		   value: #Brooms
		   value: dx * 3
		   value: 11.0.
		block
		   value: #Mops
		   value: dx * 3
		   value: 22.0.
		block
		   value: #BarSoap
		   value: dx * 3
		   value: 33.0.
		block
		   value: #DishSoap
		   value: dx * 3
		   value: 44.0.
		block
		   value: #LaundrySupplies
		   value: dx * 3
		   value: 55.0.
		block
		   value: #HouseholdCleaners
		   value: dx * 3
		   value: 66.0.
		block
		   value: #Bleach
		   value: dx * 3
		   value: 75.0
```

createProduceAisle4

```
		| block aProduce |
		block := [:produceName :x :y |
			aProduce := ShopAppProduce
						name: produceName
						x: x
```

```
                                        y: y.
                    cProduce add: aProduce].
        block
          value: #Deodorant
          value: dx * 4
          value: 11.0.
        block
          value: #Cosmetics
          value: dx * 4
          value: 22.0.
        block
          value: #HairAccessories
          value: dx * 4
          value: 33.0.
        block
          value: #Shampoo
          value: dx * 4
          value: 44.0.
        block
          value: #HairCare
          value: dx * 4
          value: 55.0
```

createProduceAisle5

```
        | block aProduce |
        block := [:produceName :x :y |
                    aProduce := ShopAppProduce
                                        name: produceName
                                        x: x
                                        y: y.
                    cProduce add: aProduce].
        block
          value: #PainRelievers
          value: dx * 5
          value: 11.0.
        block
          value: #FootCare
          value: dx * 5
          value: 22.0.
        block
          value: #ColdCare
          value: dx * 5
          value: 33.0.
        block
          value: #FluCare
          value: dx * 5
          value: 44.0.
        block
```

```
        value: #Toothpastes
        value: dx * 5
        value: 55.0.
    block
        value: #OralCare
        value: dx * 5
        value: 66.0.
    block
        value: #Vitamins
        value: dx * 4
        value: 75.0
```

createProduceAisle6

```
    | block aProduce |
    block := [:produceName :x :y |
            aProduce := ShopAppProduce
                                name: produceName
                                x: x
                                y: y.
            cProduce add: aProduce].
    block
        value: #FrozenFoods
        value: dx * 6
        value: 22.0.
    block
        value: #IceCream
        value: dx * 6
        value: 44.0.
    block
        value: #IceCreamToppings
        value: dx * 6
        value: 66.0
```

createProduceAisle7

```
    | block aProduce |
    block := [:produceName :x :y |
            aProduce := ShopAppProduce
                                name: produceName
                                x: x
                                y: y.
            cProduce add: aProduce].
    block
        value: #FrozenFoods1
        value: dx * 7
        value: 22.0.
    block
        value: #FrozenFoods2
        value: dx * 7
```

```
        value: 44.0.
    block
        value: #FrozenFoods3
        value: dx * 7
        value: 66.0
```

createProduceAisle8

```
    | block aProduce |
    block := [:produceName :x :y |
            aProduce := ShopAppProduce
                            name: produceName
                            x: x
                            y: y.
            cProduce add: aProduce].
    block
        value: #NewAgeDrinks
        value: dx * 8
        value: 11.0.
    block
        value: #EnergyDrinks
        value: dx * 8
        value: 22.0.
    block
        value: #SoftDrinks
        value: dx * 8
        value: 33.0.
    block
        value: #Sodas
        value: dx * 8
        value: 44.0.
    block
        value: #ColdDrinks
        value: dx * 8
        value: 55.0
```

createProduceAisle9

```
    | block aProduce |
    block := [:produceName :x :y |
            aProduce := ShopAppProduce
                            name: produceName
                            x: x
                            y: y.
            cProduce add: aProduce].
    block
        value: #GreetingCards
        value: dx * 9
        value: 11.0.
    block
```

```
        value: #PaperWrap
        value: dx * 9
        value: 22.0.
    block
        value: #Books
        value: dx * 9
        value: 33.0.
    block
        value: #SnackProducts
        value: dx * 9
        value: 44.0.
    block
        value: #PotatoChips
        value: dx * 9
        value: 55.0.
    block
        value: #Magazines
        value: dx * 9
        value: 66.0
```

createProduceAisle10

```
    | block aProduce |
    block := [:produceName :x :y |
            aProduce := ShopAppProduce
                            name: produceName
                            x: x
                            y: y.
            cProduce add: aProduce].
    block
        value: #Toys
        value: dx * 10
        value: 11.0.
    block
        value: #SeasonalAisle
        value: dx * 10
        value: 22.0.
    block
        value: #SnackNuts
        value: dx * 10
        value: 33.0.
    block
        value: #Candy
        value: dx * 10
        value: 44.0.
    block
        value: #Chocolate
        value: dx * 10
        value: 55.0.
```

createProduceAisle11

```
| block aProduce |
block := [:produceName :x :y |
        aProduce := ShopAppProduce
                        name: produceName
                        x: x
                        y: y.
        cProduce add: aProduce].
block
  value: #Cookies
  value: dx * 11
  value: 11.0.
block
  value: #Crackers
  value: dx * 11
  value: 22.0.
block
  value: #BabyFood
  value: dx * 11
  value: 33.0.
block
  value: #Diapers
  value: dx * 11
  value: 44.0.
block
  value: #BabyCare
  value: dx * 11
  value: 55.0
```

createProduceAisle12

```
| block aProduce |
block := [:produceName :x :y |
        aProduce := ShopAppProduce
                        name: produceName
                        x: x
                        y: y.
        cProduce add: aProduce].
block
  value: #FacialTissue
  value: dx * 12
  value: 11.0.
block
  value: #ToiletPaper
  value: dx * 12
  value: 22.0.
block
  value: #PaperTowels
```

```
        value: dx * 12
        value: 33.0.
    block
        value: #FeminineHygiene
        value: dx * 12
        value: 44.0.
    block
        value: #Charcoal
        value: dx * 12
        value: 55.0.
    block
        value: #WrapBags
        value: dx * 12
        value: 66.0
```

createProduceAisle13

```
    | block aProduce |
    block := [:produceName :x :y |
            aProduce := ShopAppProduce
                            name: produceName
                            x: x
                            y: y.
            cProduce add: aProduce].
    block
        value: #Cereal
        value: dx * 13
        value: 11.0.
    block
        value: #GranolaCereal
        value: dx * 13
        value: 22.0.
    block
        value: #FruitSnacks
        value: dx * 13
        value: 33.0.
    block
        value: #Tea
        value: dx * 13
        value: 44.0.
    block
        value: #Coffee
        value: dx * 13
        value: 55.0
```

createProduceAisle14

```
    | block aProduce |
    block := [:produceName :x :y |
            aProduce := ShopAppProduce
```

```
                                  name: produceName
                                  x: x
                                  y: y.
             cProduce add: aProduce].
      block
        value: #PastaSauce
        value: dx * 14
        value: 10.0.
      block
        value: #BoxDinner
        value: dx * 14
        value: 20.0.
      block
        value: #Rice
        value: dx * 14
        value: 30.0.
      block
        value: #Beans
        value: dx * 14
        value: 40.0.
      block
        value: #Candles
        value: dx * 14
        value: 50.0.
      block
        value: #GourmetSauce
        value: dx * 14
        value: 60.0.
      block
        value: #InternationalFood
        value: dx * 14
        value: 60.0.
```

createProduceAisle15

```
      | block aProduce |
      block := [:produceName :x :y |
             aProduce := ShopAppProduce
                                  name: produceName
                                  x: x
                                  y: y.
             cProduce add: aProduce].
      block
        value: #CannedVegetables
        value: dx * 15
        value: 11.0.
      block
        value: #CannedFruit
        value: dx * 15
```

```
        value: 22.0.
    block
        value: #MashedPotatoes
        value: dx * 15
        value: 33.0.
    block
        value: #CannedMeat
        value: dx * 15
        value: 44.0.
    block
        value: #CannedSoup
        value: dx * 15
        value: 55.0.
    block
        value: #SoupMix
        value: dx * 15
        value: 66.0.
```

createProduceAisle16

```
    | block aProduce |
    block := [:produceName :x :y |
            aProduce := ShopAppProduce
                        name: produceName
                        x: x
                        y: y.
            cProduce add: aProduce].
    block
        value: #Juices
        value: dx * 16
        value: 10.0.
    block
        value: #DrinkMix
        value: dx * 16
        value: 20.0.
    block
        value: #SportDrinks
        value: dx * 16
        value: 30.0.
    block
        value: #Jelly
        value: dx * 16
        value: 40.0.
    block
        value: #Jam
        value: dx * 16
        value: 50.0.
    block
        value: #PeanutButter
```

```
        value: dx * 16
        value: 60.0.
    block
        value: #BottleWater
        value: dx * 16
        value: 70.0
```

createProduceAisle17

```
    | block aProduce |
    block := [:produceName :x :y |
            aProduce := ShopAppProduce
                            name: produceName
                            x: x
                            y: y.
            cProduce add: aProduce].
    block
        value: #CakeMix
        value: dx * 17
        value: 11.0.
    block
        value: #Flour
        value: dx * 17
        value: 22.0.
    block
        value: #Sugar
        value: dx * 17
        value: 33.0.
    block
        value: #Oil
        value: dx * 17
        value: 47.0.
    block
        value: #KitchenGadgets
        value: dx * 17
        value: 55.0.
    block
        value: #PackageSpices
        value: dx * 17
        value: 66.0
```

createVegAisles1

```
    | block aProduce |
    block := [:produceName :x :y |
            aProduce := ShopAppProduce
                            name: produceName
                            x: x
                            y: y.
            cProduce add: aProduce].
```

```
block
   value: #YellowOnions
   value: (dx * 18) + (dxVeg / 5)
   value: 0.0.
block
   value: #RedPotatoes
   value: (dx * 18) + (2 * (dxVeg / 5))
   value: 0.0.
block
   value: #GrapeFruit
   value: (dx * 18) + (3 * (dxVeg / 5))
   value: 0.0.
block
   value: #Oranges
   value: (dx * 18) + (4 * (dxVeg / 5))
   value: 0.0.
block
   value: #RussetPotatoes
   value: (dx * 18) + (5 * (dxVeg / 5))
   value: 0.0.
```

createVegAisles2

```
| block aProduce |
block := [:produceName :x :y |
          aProduce := ShopAppProduce
                              name: produceName
                              x: x
                              y: y.
          cProduce add: aProduce].
block
   value: #RedOnions
   value: (dx * 18) + (dxVeg / 5)
   value: dyVeg.
block
   value: #Tomatoes
   value: (dx * 18) + (2 * (dxVeg / 5))
   value: dyVeg.
block
   value: #Yam
   value: (dx * 18) + (3 * (dxVeg / 5))
   value: dyVeg.
block
   value: #Avocado
   value: (dx * 18) + (4 * (dxVeg / 5))
   value: dyVeg.
block
   value: #Squash
   value: (dx * 18) + (5 * (dxVeg / 5))
```

```
   value: dyVeg.
```

createVegAisles3

```
| block aProduce |
block := [:produceName :x :y |
        aProduce := ShopAppProduce
                        name: produceName
                        x: x
                        y: y.
        cProduce add: aProduce].
block
  value: #Lime
  value: (dx * 18) + (dxVeg / 5)
  value: 2 * dyVeg.
block
  value: #Apples
  value: (dx * 18) + (2 * (dxVeg / 5))
  value: 2 * dyVeg.
block
  value: #Lemon
  value: (dx * 18) + (3 * (dxVeg / 5))
  value: 2 * dyVeg.
block
  value: #PineApple
  value: (dx * 18) + (4 * (dxVeg / 5))
  value: 2 * dyVeg.
block
  value: #Garlic
  value: (dx * 18) + (5 * (dxVeg / 5))
  value: 2 * dyVeg.
```

createVegAisles4

```
| block aProduce |
block := [:produceName :x :y |
        aProduce := ShopAppProduce
                        name: produceName
                        x: x
                        y: y.
        cProduce add: aProduce].
block
  value: #SweetOnions
  value: (dx * 18) + (dxVeg / 5)
  value: 3 * dyVeg.
block
  value: #Nuts
  value: (dx * 18) + (2 * (dxVeg / 5))
  value: 3 * dyVeg.
```

createVegAisles5

```
| block aProduce |
block := [:produceName :x :y |
        aProduce := ShopAppProduce
                        name: produceName
                        x: x
                        y: y.
        cProduce add: aProduce].
block
  value: #Cheese
  value: (dx * 18) + (dxVeg / 5)
  value: 6 * dyVeg.
block
  value: #Ham
  value: (dx * 18) + (2 * (dxVeg / 5))
  value: 6 * dyVeg.
```

createRightVegEdges1

```
| block aProduce |
block := [:produceName :x :y |
        aProduce := ShopAppProduce
                        name: produceName
                        x: x
                        y: y.
        cProduce add: aProduce].
block
  value: #Cabbage
  value: dx * 18 + dxVeg
  value: 1 * (dyVeg / 4).
block
  value: #Eggplant
  value: dx * 18 + dxVeg
  value: 3 * (dyVeg / 4).
block
  value: #BokChoy
  value: dx * 18 + dxVeg
  value: 3 * (dyVeg / 4)
```

createRightVegEdges2

```
| block aProduce |
block := [:produceName :x :y |
        aProduce := ShopAppProduce
                        name: produceName
                        x: x
                        y: y.
        cProduce add: aProduce].
block
  value: #Corn
```

```
        value: dx * 18 + dxVeg
        value: (1 * dyVeg) + (1 * (dyVeg / 4)).
    block
        value: #Broccoli
        value: dx * 18 + dxVeg
        value: (1 * dyVeg) + (2 * (dyVeg / 4)).
    block
        value: #Cauliflower
        value: dx * 18 + dxVeg
        value: (1 * dyVeg) + (3 * (dyVeg / 4)).
```

createRightVegEdges3

```
    | block aProduce |
    block := [:produceName :x :y |
            aProduce := ShopAppProduce
                            name: produceName
                            x: x
                            y: y.
            cProduce add: aProduce].
    block
        value: #BellPepper
        value: dx * 18 + dxVeg
        value: (2 * dyVeg) + (1 * (dyVeg / 4)).
    block
        value: #Cucumber
        value: dx * 18 + dxVeg
        value: (2 * dyVeg) + (2 * (dyVeg / 4)).
    block
        value: #Celery
        value: dx * 18 + dxVeg
        value: (2 * dyVeg) + (3 * (dyVeg / 4)).
```

createRightVegEdges4

```
    | block aProduce |
    block := [:produceName :x :y |
            aProduce := ShopAppProduce
                            name: produceName
                            x: x
                            y: y.
            cProduce add: aProduce].
    block
        value: #Spinach
        value: dx * 18 + dxVeg
        value: (3 * dyVeg) + (1 * (dyVeg / 5)).
    block
        value: #Cilantro
        value: dx * 18 + dxVeg
        value: (3 * dyVeg) + (2 * (dyVeg / 5)).
```

```
block
  value: #Parsley
  value: dx * 18 + dxVeg
  value: (3 * dyVeg) + (3 * (dyVeg / 5)).
block
  value: #Lettuce
  value: dx * 18 + dxVeg
  value: (3 * dyVeg) + (4 * (dyVeg / 5)).
```

createLeftVegEdges2

```
| block aProduce |
block := [:produceName :x :y |
        aProduce := ShopAppProduce
                        name: produceName
                        x: x
                        y: y.
        cProduce add: aProduce].
block
  value: #Bananas
  value: dx * 18
  value: (2 * dyVeg) - 5.
```

createLeftVegEdges3

```
| block aProduce |
block := [:produceName :x :y |
        aProduce := ShopAppProduce
                        name: produceName
                        x: x
                        y: y.
        cProduce add: aProduce].
block
  value: #Pickles
  value: dx * 18
  value: (3 * dyVeg) - 5.
```

createLeftVegEdges5

```
| block aProduce |
block := [:produceName :x :y |
        aProduce := ShopAppProduce
                        name: produceName
                        x: x
                        y: y.
        cProduce add: aProduce].
block
  value: #SaladDressing
  value: dx * 18
  value: (5 * dyVeg) - 5
```

createBackEdges4

```
| block aProduce |
block := [:produceName :x :y |
        aProduce := ShopAppProduce
                            name: produceName
                            x: x
                            y: y.
        cProduce add: aProduce].
block
  value: #Bacon
  value: dx * 4.5
  value: 0.0.
```

createBackEdges9

```
| block aProduce |
block := [:produceName :x :y |
        aProduce := ShopAppProduce
                            name: produceName
                            x: x
                            y: y.
        cProduce add: aProduce].
block
  value: #Eggs
  value: dx * 9.5
  value: 0.0.
```

createBackEdges14

```
| block aProduce |
block := [:produceName :x :y |
        aProduce := ShopAppProduce
                            name: produceName
                            x: x
                            y: y.
        cProduce add: aProduce].
block
  value: #Milk
  value: dx * 14.5
  value: 0.0.
```

drawPath

```
| produceList cPathEdge currentNode currentProduce currentEdge   fillForm
previousNode pathEdge produceEdge previousEdge |
produceList := shoppingList
            collect: [:name | aStore produceAt: name].
cPathEdge := OrderedCollection new.
currentNode := aStore aEntrance.
[currentProduce := self produce: produceList closestTo: currentNode.
previousEdge := nil.
```

```
[currentEdge := currentNode edgeTo: currentProduce except:          previousEdge.
currentEdge contains: currentProduce]
  whileFalse: [previousEdge := currentEdge.
          previousNode := currentNode.
          currentNode := currentNode isNode
                               ifTrue: [currentEdge nodeOtherThan:
                               currentNode]
                               ifFalse: [currentNode nodeCloserTo:
                               currentProduce].
          pathEdge := ShopAppEdge
                               name: #dummy
                               start: previousNode
                               end: currentNode.
          cPathEdge add: pathEdge].
produceEdge := ShopAppEdge
                 name: #dummy
                 start: currentNode
                 end: currentProduce.
cPathEdge add: produceEdge.
currentNode := currentProduce.
produceList remove: currentProduce.
produceList notEmpty] whileTrue.
fillForm := Form extent: 2 @ 2 depth: 8.
fillForm fillColor: Color red.
cPathEdge
  do: [:aEdge | aEdge displayOn: storeForm fillForm: fillForm]
```

cPathEdgeFor: produceList

```
| cPathEdge currentNode currentProduce currentEdge |
cPathEdge := OrderedCollection.
currentNode := self aNodeAt: cEntrance first.
[currentProduce := self produceClosestTo: currentNode.
[currentEdge := currentNode edgeTo: currentProduce.
cPathEdge add: currentEdge.
currentEdge contains: currentProduce]
  whileFalse: [currentNode := currentEdge nodeOtherThan:  currentNode].
currentNode := currentProduce.
cProduce remove: currentProduce name.
cProduce notEmpty] whileTrue.
^ cPathEdge
```

displayOn: aForm

```
| fillForm |
fillForm := Form extent: 2 @ 2 depth: 8.
fillForm fillColor: Color black.
cEdge
  valuesDo: [:aEdge | aEdge displayOn: aForm fillForm: fillForm]
```

produceClosestTo: aShopAppNode

```
| minSqDist sqDist answer targetPoint |
minSqDist := 1.0e100.
targetPoint := aShopAppNode point.
cProduce
  do: [:aProduce |
        sqDist := aProduce point squaredDistanceTo: targetPoint.
        sqDist < minSqDist
              ifTrue: [answer := aProduce.
                    minSqDist := sqDist]].
^ answer
```

doClearFrom: aPane

```
selectorList := Array new.
self changed: #selectorList.
messageList := Array new.
self changed: #messageList.
searchString := nil.
self changed: #searchString.
shoppingList := SortedCollection new.
self changed: #shoppingList.
self containingWindow setLabel: 'Shopping App'.
self changed: #containingWindow.
storeForm fillColor: Color lightGray lighter lighter lighter.
aSketchMorph layoutChanged.
```

cProduceMatching: searchString

```
| cFoundProduce |
cFoundProduce := cProduce
              select: [:a | '*' , searchString , '*' match: a name].
^ cFoundProduce
  collect: [:a | a name]
```

doFindFrom: aObject

```
| aProduce |
storeForm fillColor: Color lightGray lighter lighter lighter.
aStore displayOn: storeForm.
shoppingList
  do: [:name |
        aProduce := aStore produceAt: name.
        aProduce displayOn: storeForm].
self drawPath.
aSketchMorph layoutChanged
```

Bibliography:

1. Ducasse, Stéphane, Andrew P. Black. Oscar Nierstrasz. Damien Pollet. Squeak By Example. free download from SqueakByExample.org, 2007.

2. Shortest Path Problem: Dijkstra's algorithm. 3/21/10. Unknown. 3/24/10 http://en.wikipedia.org/wiki/Dijkstra's_algorithm

3. Squeak Home Page. 10/7/09. Unknown. 10/08/09 http://www.squeak.org/

4. Squeak Tutorial. 2009. Stephan B Wessels. 10/08/09 http://squeak.preeminent.org/tut2007/html/

5. Traveling Sales Man Problem. 2/27/10. Unknown. 10/02/09 http://en.wikipedia.org/wiki/Travelling_salesman_problem

6. TreoSystems' iPAL product locator. 2008. Unknown. 10/08/09 http://www.treosys.com/ipalsuite.htm