Chief snow Engineer (Project Manager)
Staff snow Scientist (Language Guru)
snow plow (Systems Integrator)
snow man (Systems Architect)
Icebreaker (Tester)

Jonathan Bell
Willi Ballenthin
Cesar Aguilar
Vinay Bettadapura
Sharadh Bhaskharan
What is snow?

- [snoh]
- *n.* A precipitation in the form of ice crystals, mainly of intricately branched, hexagonal form and often agglomerated into snowflakes, formed directly from the freezing of the water vapor in the air
No, that’s snow, not *sn*w

• *sn*w (like snow) is clear, simple, and pure
• *sn*w is a declarative, event-based language leveraging the power of genetic algorithms
• So, what’s a genetic algorithm?
  • They make hard problems easy!
Genetic Optimizations

• Start with initial solutions
• “Mutate” these solutions
• Keep the best solutions
• Rinse, lather, repeat (as necessary)
Genetic Optimizations

- Travel Salesman
- Knapsack Problem
- Timetabling
- Code-breaking
- N-Queens
The hard way and the **sn*w** way

- 700 lines of C or 10 Java classes that don’t even get CLOSE to solving a genetic algorithm-based problem...

- ...or... **sn*w**
The snow way

Program Initialization → Organism Construction (to construct:) → Evaluate Fitness (to evaluateFitness:) → Selection (automatic)

Mutation (to mutate:) → Program Termination

Termination criteria not met

Termination criteria met
Overall Structure

• Override default methods and type definitions for the GA run loop
• Can provide event listeners - blocks of code that are executed before or after key events in the simulation

```
<table>
<thead>
<tr>
<th>BaseSnowProgram</th>
</tr>
</thead>
<tbody>
<tr>
<td>Default constructors</td>
</tr>
<tr>
<td>Default block methods</td>
</tr>
<tr>
<td>...</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>SnowProgramImpl</th>
</tr>
</thead>
<tbody>
<tr>
<td>Overridden base methods</td>
</tr>
<tr>
<td>Custom methods &amp; types</td>
</tr>
<tr>
<td>...</td>
</tr>
</tbody>
</table>
```
A to mutate: organismA
  nth: (rand: 0 8), organism.chromosome = rand: 0 8
  return organismA
end

B define chromosome:
  8 genes
end

to evaluateFitness: checkOrganism
  var curfit = 28
  var i,j, gene1, gene2
  for i from 0 to 6
    gene1 = nth:i, checkOrganism.chromosome
    for j from 1 to 7
      gene2 = nth:j, checkOrganism.chromosome
      if gene1 = gene2 then
        curfit = curfit - 1
      elseif ((gene1.num - gene2.num) / (i - j)) = 1 then
        curfit = curfit - 1
      end
    end
  end
  return 28 - curfit
end

to construct: newOrganism
  foreach gene in newOrganism.chromosome as gene1
    gene1.num = rand: 0 8
  end
  return newOrganism
end

C to construct: newOrganism
  foreach gene in newOrganism.chromosome as gene1
    gene1.num = rand: 0 8
  end
  return newOrganism
end

to evaluateFitness: checkOrganism
  var curfit = 28
  var i,j, gene1, gene2
  for i from 0 to 6
    gene1 = nth:i, checkOrganism.chromosome
    for j from 1 to 7
      gene2 = nth:j, checkOrganism.chromosome
      if gene1 = gene2 then
        curfit = curfit - 1
      elseif ((gene1.num - gene2.num) / (i - j)) = 1 then
        curfit = curfit - 1
      end
    end
  end
  return 28 - curfit
end

E after termination:
  print: "The best solution found was:
  foreach organism in ~population as org1
    if (evaluateFitness: organism) = ~endFitness
      then
        for i from 0 to 7
          print: "queen at" + i + "," + (nth: i, organism.chromosome.gene) + "\n"
        end
      end
  end
end

F ~endGenerations= 800
~endFitness = 28
~topParentPool = 0.1
~mutationRate = (rand: 1 5)/10
A to mutate: organismA
  nth: (rand: 0 8), organism.chromosome = rand: 0 8
  return organismA
end

B define chromosome:
  8 genes
end

to construct: newOrganism
  foreach gene in newOrganism.chromosome as gene1
    gene1.num = rand: 0 8
  end
  return newOrganism
end

to evaluateFitness: checkOrganism
  var curfit = 28
  var i,j,gene1,gene2
  for i from 0 to 6
    gene1 = nth:i, checkOrganism.chromosome
    for j from 1 to 7
      gene2 = nth:j, checkOrganism.chromosome
      if gene1 = gene2 then
        curfit = curfit - 1
      elseif ((gene1.num - gene2.num) / (i - j)) = 1 then
        curfit = curfit - 1
      end
    end
  end
  return 28 - curfit
end

E after termination:
  print: "The best solution found was:
  foreach organism in ~population as org1
    if (evaluateFitness: organism) = ~endFitness
      for i from 0 to 7
        print: "queen at" + i + "," + (nth: i, organism.chromosome.gene) + 
      end
    end
  end
end

F ~endGenerations= 800
~endFitness     = 28
~topParentPool  = 0.1
~mutationRate   = (rand: 1 5)/10
Syntactic Constructs

- **Declarative**
  
  ```
  A to mutate: organismA
  nth: (rand: 0 8), organism.chromosome = rand: 0 8
  return organismA
  end
  ```

- **Loosely Typed**
  
  ```
  B define chromosome:
  8 genes
  end
  ```

![Diagram showing relationships between SnowType, SnowList, and SnowAtom]
A to mutate: organismA
  nth: (rand: 0 8), organism.chromosome = rand: 0 8
  return organismA
end

B define chromosome:
  8 genes
end

to evaluateFitness: checkOrganism
  var curfit = 28
  var i,j,gene1,gene2
  for i from 0 to 6
    gene1 = nth:i, checkOrganism.chromosome
    for j from 1 to 7
      gene2 = nth:j, checkOrganism.chromosome
      if gene1 = gene2 then
        curfit = curfit - 1
      elseif ((gene1.num - gene2.num) / (i - j)) = 1 then
        curfit = curfit - 1
      end
    end
  end
  return 28 - curfit
end

to construct: newOrganism
  foreach gene in newOrganism.chromosome as gene1
    gene1.num = rand: 0 8
  end
  return newOrganism
end

to evaluateFitness: checkOrganism
  var curfit = 28
  var i,j,gene1,gene2
  for i from 0 to 6
    gene1 = nth:i, checkOrganism.chromosome
    for j from 1 to 7
      gene2 = nth:j, checkOrganism.chromosome
      if gene1 = gene2 then
        curfit = curfit - 1
      elseif ((gene1.num - gene2.num) / (i - j)) = 1 then
        curfit = curfit - 1
      end
    end
  end
  return 28 - curfit
end

to evaluateFitness: checkOrganism
  var curfit = 28
  var i,j,gene1,gene2
  for i from 0 to 6
    gene1 = nth:i, checkOrganism.chromosome
    for j from 1 to 7
      gene2 = nth:j, checkOrganism.chromosome
      if gene1 = gene2 then
        curfit = curfit - 1
      elseif ((gene1.num - gene2.num) / (i - j)) = 1 then
        curfit = curfit - 1
      end
    end
  end
  return 28 - curfit
end

to evaluateFitness: checkOrganism
  var curfit = 28
  var i,j,gene1,gene2
  for i from 0 to 6
    gene1 = nth:i, checkOrganism.chromosome
    for j from 1 to 7
      gene2 = nth:j, checkOrganism.chromosome
      if gene1 = gene2 then
        curfit = curfit - 1
      elseif ((gene1.num - gene2.num) / (i - j)) = 1 then
        curfit = curfit - 1
      end
    end
  end
  return 28 - curfit
end

E after termination:
  print: "The best solution found was:
  foreach organism in ~population as org1
    if (evaluateFitness: organism) = ~endFitness
      for i from 0 to 7
        print: "queen at" + i + "," + (nth: i, organism.chromosome.gene) + 
      end
    end
  end
end

F ~endGenerations= 800
~endFitness     = 28
~topParentPool  = 0.1
~mutationRate   = (rand: 1 5)/10
Syntactic Constructs

• Very **natural** declaration
• **List** oriented

```c
C to construct: newOrganism
    foreach gene in newOrganism.chromosome as gene1
        gene1.num = rand: 0 8
    end
return newOrganism
end
```
Translator Architecture

JFLex → Byacc/J → Translated Code (java)

BaseSnowParser
Construct translation methods

SnowParser
Generated by BYACC/J
A to mutate: organismA
   nth: (rand: 0 8), organism.chromosome = rand: 0 8
   return organismA
end

B define chromosome:
   8 genes
end

C to construct: newOrganism
   foreach gene in newOrganism.chromosome as gene1
   gene1.num = rand: 0 8
   return newOrganism
end

D to evaluateFitness: checkOrganism
   var curfit = 28
   var i,j,gene1,gene2
   for i from 0 to 6
      gene1 = nth:i, checkOrganism.chromosome
      for j from 1 to 7
         gene2 = nth:j, checkOrganism.chromosome
         if gene1 = gene2 then
            curfit = curfit - 1
         elseif ((gene1.num - gene2.num) / (i - j)) = 1 then
            curfit = curfit - 1
         end
      end
   end
   return 28 - curfit
end

E after termination:
   print: "The best solution found was:
   foreach organism in ~population as org1
      if (evaluateFitness: organism) = ~endFitness
         then
            for i from 0 to 7
               print: "queen at" + i + "," + (nth: i, organism.chromosome.gene) + "\n"
         end
      end
   end

F ~endGenerations= 800
~endFitness     = 28
~topParentPool  = 0.1
~mutationRate   = (rand: 1 5)/10
Take a sleigh ride down the sn*w slide

```java
public void dbg_afterTERMINATION(){
    snw_print(new SnowAtom("The best solution found was:\n"));
    for(SnowType org1 : symbols.get("~population")){
        if ((snow_evaluateFitness(org1)).
            equals(symbols.get("~endFitness"))){
            for(SnowAtom i = new SnowAtom(0);
                i.nequals(new SnowAtom(7));i.plus(new
                SnowAtom(1))){
                snw_print( new SnowAtom("queen at").plus(i).plus(new SnowAtom("
                ",
                ")).plus((
                snow_nth(i,org1.getField("gene")))
                ).plus(new SnowAtom("\n")));
            }
        }
    }
}
```
A to mutate: organismA
   nth: (rand: 0 8), organism.chromosome = rand: 0 8
   return organismA
end

B define chromosome:
  8 genes
end

C to construct: newOrganism
   foreach gene in newOrganism.chromosome as gene1
      gene1.num = rand: 0 8
   end
   return newOrganism
end

D to evaluateFitness: checkOrganism
   var curfit = 28
   var i,j,gene1,gene2
   for i from 0 to 6
      gene1 = nth:i, checkOrganism.chromosome
   end
   for j from 1 to 7
      gene2 = nth:j, checkOrganism.chromosome
      if gene1 = gene2 then
         curfit = curfit - 1
      elseif ((gene1.num - gene2.num) / (i - j)) = 1 then
         curfit = curfit - 1
      end
   end
   return 28 - curfit
end

E after termination:
   print: "The best solution found was:
   foreach organism in ~population as org1
      if (evaluateFitness: organism) = ~endFitness
         for i from 0 to 7
            print: "queen at" + i + "," + (nth: i, organism.chromosome.gene) + "\n" 
         end
      end
   end
end

F ~endGenerations= 800
~endFitness = 28
~topParentPool = 0.1
~mutationRate = (rand: 1 5)/10
Development Environment

• Mixed Mac/Linux/Windows environment

• Eclipse, Google Code (SVN + wiki), JFlex, Byacc/JFlex

• Java 6
Test Strategy

- Create black-box tests
- Incremental & regression testing
- Ensure that there is complete coverage of grammar in test cases.
- Make sure each grammar production was hit

160 Productions → 160+ Test Cases → 3000+ Lines of Testing Code
Concluding Remarks

• Come up with a logo, then a name, then a domain
• We don’t know C as well as we thought
• Java is handy, but gets in the way sometimes
• Google Code + SVN are awesome
• Decide on the grammar and stick to it!
  • (We did*)

*We stuck to the grammar except for adding “return”
Natural
Portable
Powerful

snow