**Jacob Alspaw**

**Software Craftsmanship**

**Pseudocode Homework**

Data structure description:

The structure for this assignment will consist of a two-dimensional linked lists forming a completely connected graph of warrior relations. The outer linked list will be each warrior. The inner linked list will define a certain warrior’s relationship to every other warrior. The data structure will remember the first warrior added to the collection of nodes. Each node will have four fields: warrior data, a next warrior, a next relationship, and a relationship type. Below is a visual representation.

W -> x -> y -> z

|

x -> y -> z -> w

|

y -> z -> w -> x

|

z -> w -> x -> y

INFERENCES:

* The example will have warriors w, x, y, and z. We will infer that a warrior can oppose no one, and that all oppositions are a two-way relationship such that if x opposes y, then y must oppose x.
* There are three kinds of relationships, known enemies and known allies, and unknown.
* A node can be constructed with parameters (Warrior info, next warrior, next relation, relation type).
* There is an equals method on the warrior node class that will compare “this” object to two other objects. It will return true if either objects represent “this”. This means that the data is the same (the same warrior), but the relation or next pointers may differ.

**METHOD**: CREATE(X)

**INPUT**: An object X (warrior)

**OUTPUT**: A boolean to signal success of adding object

**IMPLEMENTATION**

Let the two-dimensional linked list W be a field of the data structure.

**If** W is empty //empty list will make X the first

W.firstWarrior 🡨 newNode(X.data, null, null, NEUTRAL)

currentWarrior 🡨 W.firstWarrior //checking if warrior already exists

**While** currentWarrior is not equal to null

**If** currentWarrior is equal to X then

Return false // already been added to the set

currentWarrior 🡨 currentWarrior.nextWarrior

currentWarrior.nextWarrior 🡨 X

currentWarrior 🡨 W.firstWarrior //ensuring completely connected graph

Initialize new linked list xRelations

**While** currentWarrior has nextWarrior

tempNode 🡨 currentWarrior.nextRelation

currentWarrior.nextRelation 🡨 newNode(X.data, null, tempNode, NEUTRAL)

Add newNode(currentWarrior.data, null, xRelations.firstNode, NEUTRAL) to xRelations

currentWarrior 🡨 currentWarrior.nextWarrior

X.nextRelation 🡨 xRelations.firstNode

Return true

**COMPLEXITY:** 4

**RUNTIME:** O(N) where N is the number of distinct warriors

**METHOD**: OPPOSE(X,Y)

**INPUT**: An object X and Y (two warriors)

**OUTPUT**: A boolean to signal success of adding opposition

**IMPLEMENTATION** O(N) where N is the number of distinct warriors

Let the two-dimensional linked list W be a field of the data structure.

CREATE(X)

CREATE(Y)

currentWarrior 🡨 W.firstWarrior //make X and Y oppose

**While** currentWarrior is not null

**If** currentWarrior.equals(X,Y) then

currentRelation 🡨 currentWarrior.nextRelation

**While** not currentRelation.equals(X,Y) //go through the relations

currentRelation 🡨 currentRelation.nextRelation

**If** currentRelation is allied

Return false

Else

currentRelation.nextRelation 🡨 ENEMY

currentWarrior 🡨 currentWarrior.nextWarrior

Return true

**COMPLEXITY:** 4

**RUNTIME:** O(N) where N is the number of distinct warriors

**METHOD**: OPPONENTS(X,Y)

**INPUT**: An object X and Y (two warriors)

**OUTPUT**: Whether x and y must belong to different sides

**IMPLEMENTATION** O(N) where N is the number of distinct warriors

Let the two-dimensional linked list W be a field of the data structure.

CREATE(X)

CREATE(Y)

currentWarrior 🡨 W.firstWarrior

**While** currentWarrior is not equal to null

**If** currentWarrior.equals(X,Y)

currentRelation 🡨 currentWarrior.nextRelation

**While** not currentRelation.equals(X,Y) //go through the relations

currentRelation 🡨 currentRelation.nextRelation

Return currentRelation is an enemy or not

currentWarrior 🡨 currentWarrior.nextWarrior

Return false

**COMPLEXITY:** 3

**RUNTIME:** O(N) where N is the number of distinct warriors