**Project One Pseudocode Andy Nguyen**

**CS – 2336.003**

**Smith**

**Main Class**

**Main**

* Ask, reads, and stores user for input initial grid filename
* Ask, reads, and stores user for character to represent ant in console output of turns
* Ask, reads, and stores user for character to represent beetle in console output of turns
* Ask, reads, and stores user for the number of turns
* Declare and initialize 2D array of creature class object in 10 x 10 size
* **Process Initial Input Grid File**
* For 0 : number of turns
  + For 0 : creature 2D array’s column size // traversal by column first then row
    - For 0 : creature 2D array row Size
      * If creature at index is instance of beetle object
        + **Move Beetle in Grid**
      * If creature at index is instance of ant object
        + **Ant Can’t Move Illegally**
      * If **Beetle Starve** is equal to true
        + Beetle object at creature 2d array indexes set to null
      * If **Ant Breed** is equal to true
        + **Ant Breed Create**
      * If **Beetle Breed** is equal to true
        + **Beetles Breed Create**
      * **Increment Number of Turns for Beetle**
      * **Increment Number of Turns for Ant**
  + **Print Grid**

**Print Grid**

**Parameters:** creature 2D array, character to represent ant, character to represent beetle

**Return:** Nothing

* Print out “Turn “ with the number at which the number of turns it is at
* For 0 : creature 2D array row Size
  + For 0 : creature 2D array’s column size
    - If creature at index is null
      * Print Space
    - If creature at index is instance of beetle
      * Print character to represent beetle
    - If creature at index is instance of ant
      * Print character to represent ant
  + Print new line
* Print new line

**Populate Creature Array**

**Parameters:** input initial file grid filename, creature 2D array

**Returns:** Nothing

* Open input grid file
* While not end of file
  + Read file line, store file line into temporary string
  + For 0: Length of file line string
    - If character at index of string matches ‘B’
      * Declare and Initialize Beetle object
      * Store ‘x’ and ‘y’ positions (‘x’ as in the row beetle was on and ‘y’ as in the index of file line string) in the newly declared object
      * Store Beetle object in the ‘x’ and ‘y’ position of the creature 2D array
    - Else if the character at index of string matches ‘a’
      * Declare and Initialize Ant object
      * Store ‘x’ and ‘y’ positions (‘x’ as in the row beetle was on and ‘y’ as in the index of file line string) in the newly declared object
      * Store Ant object in the ‘x’ and ‘y’ position of the creature 2D array
* Close input grid file

**Move Beetle in Grid**

**Parameters:** beetle object

**Return:** Nothing

* If **Beetle Move** is equal to “North”
  + If Creature 2D Array index of North Orthogonal is not instance of beetle // checks if there is not already a beetle there
    - If Creature 2D Array index of North Orthogonal is instance of ant
      * **Beetle Eaten**
    - Creature 2D Array index of North Orthogonal is equal to beetle object
    - Creature 2D Array index of original position is set to null;
* Else if is equal to “East”
  + If Creature 2D Array index of East Orthogonal is not instance of beetle
    - If Creature 2D Array index of North Orthogonal is instance of ant
      * **Beetle Eaten**
    - Creature 2D Array index of South Orthogonal is equal to beetle object
    - Creature 2D Array index of original position is set to null;
* Else if is equal to “South”
  + If Creature 2D Array index of South Orthogonal is not instance of beetle
    - If Creature 2D Array index of North Orthogonal is instance of ant
      * **Beetle Eaten**
    - Creature 2D Array index of South Orthogonal is equal to beetle object
    - Creature 2D Array index of original position is set to null;
* Else if is equal to “West”
  + If Creature 2D Array index of South Orthogonal is not instance of beetle
    - If Creature 2D Array index of North Orthogonal is instance of ant
      * **Beetle Eaten**
    - Creature 2D Array index of South Orthogonal is equal to beetle object
    - Creature 2D Array index of original position is set to null;

// Movement of the Beetle

**Find Closest Orthogonal Ant**

**Parameters:** beetle object, creature 2D array

**Return:** Return integer array holding distances of orthogonal ants

* Create integer array holding distances of orthogonal ants and each index of size 4 array representing North, South, East, and West respectively.
* For loop which traverses creature indexes (North) above the beetle position until edge
  + If creature at indexes above beetle is an instance of ant object
    - Store the distance from beetle to the northern ant
    - Break from the loop
* For loop which traverses creature indexes (East) left of the beetle position until edge
  + If creature at indexes left of the beetle is an instance of ant object
    - Store the distance from beetle to the left ant
    - Break from the loop
* For loop which traverses creature indexes (South) below the beetle position until edge
  + If creature at indexes below beetle is an instance of ant object
    - Store the distance from beetle to the southern ant
    - Break from the loop
* For loop which traverses creature indexes (West) right of the beetle position until edge
  + If creature at indexes right of the beetle is an instance of ant object
    - Store the distance from beetle to the right ant
    - Break from the loop

**Find Ant Neighbors**

* **Parameter:** beetle object, creature 2D array, integer array holding distances of orthogonal ants and each index of size 4 array representing North, South, East, and West respectively.

**Return:** Return integer array holding number of ant neighbors of orthogonal ants

* Create integer array holding amount of ant neighbors of orthogonal ants
* For 0 : integer array size of distances of orthogonal ants
  + Checks adjacent spots of orthogonal ants for instances of
  + Store the amount of neighbor ants for each orthogonal ant

**Find Nearest Edge**

**Parameter:** beetle object, creature 2D array

**Return:** Return integer array holding the distances of the edges from the beetle

* Create integer array holding the distances of the edges from the beetle and each index of size 4 array representing North, South, East, and West respectively.
* For loop which traverses creature indexes (North) above the beetle position until edge
  + Count distance until edge for North
* For loop which traverses creature indexes (East) left of the beetle position until edge
  + Count distance until edge for East
* For loop which traverses creature indexes (South) below the beetle position until edge
  + Count distance until edge for South
* For loop which traverses creature indexes (West) right of the beetle position until edge
  + Count distance until edge for West
* Store distances of cardinal directions into the array

// Breeding of the Beetle

**Beetles Breed Create**

**Parameters:** object beetle, creature 2D array

* If north orthogonal of object beetle is empty
  + Create new beetle object and store its x and y position
* Else if east orthogonal of object beetle is empty
  + Create new beetle object and store its x and y position
* Else if south orthogonal of object beetle is empty
  + Create new beetle object and store its x and y position
* Else if west orthogonal of object beetle is empty
  + Create new beetle object and store its x and y position

// Movement of Ants

**Find Nearest Orthogonal Beetle**

**Parameters:** object ant, creature 2d array

**Return:** Return integer array holding distances of orthogonal beetles

* Create integer array holding distances of orthogonal beetles and each index of size 4 array representing North, South, East, and West respectively.
* For loop which traverses creature indexes (North) above the ant position until edge
  + If creature at indexes above beetle is an instance of ant object
    - Store the distance from ant to the northern beetle
    - Break from the loop
* For loop which traverses creature indexes (East) left of the ant position until edge
  + If creature at indexes left of the ant is an instance of beetle object
    - Store the distance from ant to the left beetle
    - Break from the loop
* For loop which traverses creature indexes (South) below the ant position until edge
  + If creature at indexes below ant is an instance of beetle object
    - Store the distance from ant to the southern beetle
    - Break from the loop
* For loop which traverses creature indexes (West) right of the beetle position until edge
  + If creature at indexes right of the ant is an instance of beetle object
    - Store the distance from ant to the right beetle
    - Break from the loop

**Ant Can’t Move Illegally**

**Parameters:** object ant, creature 2d array

**Return:** String object which contains direction

* If **Ant Move** direction of Ant would result in positive position and less than max size of grid // makes sure ant within edges of grid
  + If direction of ant ( possible creature 2d array new position of ant) is not instance of creature
    - Return direction
* Return “None”

// Breeding of the Ant

**Ants Breed Create**

**Parameters:** object ant, creature 2D array

* If north orthogonal of object ant is empty
  + Create new ant object and store its x and y position
* Else if east orthogonal of object ant is empty
  + Create new ant object and store its x and y position
* Else if south orthogonal of object ant is empty
  + Create new ant object and store its x and y position
* Else if west orthogonal of object ant is empty
  + Create new ant object and store its x and y position

**Beetle Class**

**Move**

**Parameter:** integer array holding distances of orthogonal ants, integer array holding number of ant neighbors of orthogonal ants, integer array holding the distances of the edges from the beetle

**Return:** Direction the beetle will move

* Create single element array holding index of furthest edge
* If **Check Beetle for Zero Orthogonal Ants is false**
  + Create single element array holding index of closest orthogonal ant
  + If **Check Beetle for Ties on Distance** is true
    - If **Check Beetle for Ties on Neighbors** is equal to 0
      * Return “North”
    - Else if it is equal to 1
      * Return “East”
    - Else if it is equal to 2
      * Return “South”
    - Else if it is equal to 3
      * Return “West”
  + Else // executes if no ties when finding closest orthogonal ant
    - If single element array holding index of closest orthogonal ant is equal to 0
      * Return “North”
    - Else if it is equal to 1
      * Return “East”
    - Else if it is equal to 2
      * Return “South”
    - Else if it is equal to 3
      * Return “West”
* Else // executes if no orthogonal ants
  + If single element array holding index of furthest edge is equal to 0
    - Return “North”
  + Else if it is equal to 1
    - Return “East”
  + Else if it is equal to 2
    - Return “South”
  + Else if it is equal to 3
    - Return “West”

**Check Beetle for Ties on Distance (Beetle Move)**

**Parameter:** integer array holding distances of orthogonal ants, single element array holding index of closest orthogonal ant

**Return**: (Boolean) Check for Ties on Neighbor?

* For 1 : integer array size of distances of orthogonal ants // Finds closest ant from orthogonal
  + If integer at index is lower
    - Store the index of lower distance
* For 0 : integer array size distances of orthogonal ants // Finds ties of closest orthogonal ant
  + If index is not equal to lowest distance & integer at index is the same
    - Return true
* Single element array is set equal to lowest distance found // using array so I can “return” multiple values
* Return false

**Check Beetle for Ties on Neighbors (Beetle Move)**

**Parameter:** integer array holding number of ant neighbors of orthogonal ants

**Return:**

* For 1: integer array size of ant neighbors of orthogonal ants // Finds greatest amount of neighbors
  + If integer at index is higher
    - Store the index of higher neighbors
    - Store the value of higher neighbors
* Return index of highest number of neighbors // Why return index? The index of the arrays represents the cardinal directions in the NESW priority. If it is still tied, the if statements in the move method will take care of the NESW priority.

**Check Beetle for Zero Orthogonal Ants**

**Parameter:** integer array holding distances of orthogonal ants, single element array holding index of furthest edge

**Return:** boolean value which determines whether the movement matrix should go on

* Integer which holds the number of zero orthogonal ants
* For int 0 : integer array size of distances of orthogonal ants
  + If integer at index is equal to zero
    - Zero orthogonal ants plus one
* If zero orthogonal ants is equal to 0, 1 ,2 , or 3
  + Return false
* If zero orthogonal ants is equal to 4 // 4 as in all cardinal directions have zero orthogonal ants
  + For 1: integer array size of distances of the edges from the beetle
    - If integer at index is higher
      * Store the index of higher integer
* Since NESW priority established with integer array order, return index of farthest edge using single element array
* Return true, meaning we had to find the furthest edge

**Breed**

**Parameters:** Nothing

**Return:** Boolean value of whether or not to breed

* If number of turns for Breeding is equal to 8 // Number of turns is a class variable in beetle
  + Set number of turns for Breeding equal to 0
  + Return true
* Else
  + Return false

**Starve**

**Parameters:** Nothing

**Return:** Boolean value whether or not to starve

* If number of turns for Starvation is equal to 5 // Number of turns is a class variable in beetle
  + Return true // to make beetle “die” or delete beetle
* Else
  + Return false

**Eaten**

**Parameter:** Nothing

**Return:** Nothing

* Number of turns for starvation is set to 0

**Increment Number of Turns for Beetle**

**Parameters:** Nothing

**Return:** Nothing

* Increment number of turns for Breeding by 1 // Number of turns for Breeding is a class variable in beetle
* Increment number of turns for Starvation by 1 // Number of turns for Starvation is a class variable in beetle

**Get Number of Turns for Starving Beetle**

**Parameters:** Nothing

Return: Number of turns for Starvation

**Get Number of Turns for Breeding Beetle**

**Parameters:** Nothing

**Return:** Number of turns for Breeding // Number of turns is a class variable in beetle

**Ant Class**

**Move**

**Parameters:** integer array holding distances of orthogonal beetles

**Return:**

* If **Check Ant for Zero Orthogonal Beetles** is equal to true
  + return “None”
* Else
  + Create single element array holding index of closest orthogonal ant
  + If **Check Ant for Ties on Distance** is true
    - Create single element array holding direction with no beetle
    - If **No Beetle in Direction** is false
      * If single element array holding index of direction with no beetle
        + Return “North”
      * Else if it is equal to 1
        + Return “East”
      * Else if it is equal to 2
        + Return “South”
      * Else if it is equal to 3
        + Return “West”
  + Else
    - If single element array holding index of closest orthogonal beetle is equal to 0
      * Return “South”
    - Else if it is equal to 1
      * Return “West”
    - Else if it is equal to 2
      * Return “North”
    - Else if it is equal to 3
      * Return “East”

**Check Ant for Zero Orthogonal Beetles**

**Parameters:** integer array holding distances of orthogonal beetles

**Return:** Boolean value which determines whether the movement matrix should go on

* Integer which holds the number of zero orthogonal beetles
* For int 0 : integer array size of distances of orthogonal beetles
  + If integer at index is equal to zero
    - Zero orthogonal beetles
* If zero orthogonal beetles is equal to 0, 1, 2, or 3
  + Return false
* Return true

**Check Ant for Ties on Distance**

**Parameter:** integer array holding distances of orthogonal beetles, single element array holding index of closest orthogonal beetle

**Return**: Boolean which determines whether the movement matrix should go on

* For 1 : integer array size of distances of orthogonal ants // Finds closest beetle from orthogonal
  + If integer at index is lower
    - Store the index of lower distance
* For 0 : integer array size distances of orthogonal beetles // Finds ties of closest orthogonal beetle
  + If index is not equal to lowest distance & integer at index is the same
    - Return true
* Single element array is set equal to lowest distance found // using array so I can “return” multiple values
* Return false

**No Beetle in Direction**

**Parameter:** integer array holding distances of orthogonal beetles, single element array holding index no beetle in direction

**Return**: Boolean which determines whether the movement matrix should go on

* For int 0 : integer array size of distances of orthogonal beetles
  + If integer at index is equal to zero
    - Return false
* Return true // return true meaning ant is surrounded and needs to move towards farthest beetle

**Breed**

**Parameters:** Nothing

**Return:** Boolean value of whether or not to breed

* If number of turns for Breeding is equal to 3 // Number of turns is a class variable in beetle
  + Set number of turns for Breeding equal to 0
  + Return true
* Else
  + Return false

**Increment Number of Turns for Ant**

**Parameters:** Nothing

**Return:** Nothing

* Increment number of turns for Breeding by 1 // Number of turns for Breeding is a class variable in ant

**Test Cases**

1. Checking ant movement that is at the edge of the grid but is also surrounded by fellow ants at all its orthogonals
2. Checking ant movement when ants are the northern and southern orthogonal
3. Checking beetle movement when surrounded by ants
4. Checking beetle movement when two ants are equidistant on east and west orthogonal but the west orthogonal has more ant neighbors
5. Check ant movement when at edge while beetles are at its orthogonals
6. Check ant breeding when surrounded by adjacent ants
7. Check beetle breeding when surrounded by adjacent beetles
8. Check beetle starvation when beetle is unable to eat due to distance of ants
9. Check ant movement when north, south, and west beetles are equidistant
10. Checking ant movement when in the corner of the grid and beetles are surrounding the open orthogonals