Department of Computer Science, Mathematics & Physics

COMP2232 – Object-oriented Programming Concepts
Semester 1, 2018-19: Assignment #2

Hungry Hunters

Read this entire document before starting your assignment!

For this assignment, you are to **complete the following** in order to continue work on the framework for the **Hungry Hunters** game. You will be continuing work on the files from Assignment #1.

1. The structure of the following class has been updated. Make the appropriate changes to your file(s).

[Abstract Class] Animal (inherits from **GamePiece**)	
Field	**Description**
`int trackingID`	Contains the ID used to track this animal on the board.
`int points`	Contains the points associated with this animal.
`boolean isActive`	Contains **true** is animal is active (alive) and **false**, otherwise.
`int[] newPos`	Contains the coordinate pair (row, col) for the animal's destination.
Method	**Description**
`Animal()`	Constructor used to initialize `type` to "Animal", and `symbol` to '**?**'.
`void setActive(boolean status)`	Updates the animal's **isActive** variable.
`boolean isActive()`	Returns whether an animal is active or not (true or false).
`public abstract int[] move()`	Abstract method.
Mutators & Accessors	Accessor and mutator methods must be created for any new data members.

2. **`Hunter`:** You have been given this file and must update it to implement the **StraightMovement** and **DiagonalMovement** interfaces along with the additional method below:
   a. **int[] move(int direction):** This method is used to get the row&col position the hunter wants to move to. **It does NOT update the hunter's actual position.** The parameter **direction** indicates direction for movement such that 1 = north, 2 = south, 3 = east, 4 = west, 5 = northeast, 6 = northwest, 7 = southeast, 8 = southwest. This method will call the appropriate method (i.e. moveNorth, moveSouth, etc..) This method returns the new position the hunter wishes to move to (i.e. **newPos** array holding row & col)
   b. **void moveNorth():** This method is implemented from the **StraightMovement** interface. It updates **newPos** data member with a new row/col pair which would allow for movement north.

c. **void moveSouth():** Similar to the **moveNorth()** method but updates **newPos** with a pair which allows for southward movement.

d. **Ensure that you implement all other methods in the StraightMovement and DiagonalMovement interfaces in a similar fashion to moveNorth() & moveSouth().**

3. **`Slider:`** You have been given this file and must update it to implement the **StraightMovement** interface along with the additional method below:

   a. **int[] move():** This method is used to get the row&col position the animal wants to move to. **It does NOT update the animal's actual position.** The method must randomly choose a direction from the ones **valid for this type of animal.** (Note: 1 = north, 2 = south, 3 = east, 4 = west, 5 = northeast, 6 = northwest, 7 = southeast, 8 = southwest.) This method will call the appropriate method (i.e. moveNorth, moveSouth, etc..) This method returns the new position the animal wishes to move to (i.e. **newPos** array holding row & col)

      This method also implements the creeper's hiding action following game rules:

      - can **hide** (i.e. go into a hole and cannot be eaten) randomly;
      - cannot hide for two successive rounds

   b. **void moveNorth():** This method is implemented from the **StraightMovement** interface. It updates **newPos** data member with a new row/col pair which would allow for movement north.

   c. **void moveSouth():** Similar to the **moveNorth()** method but updates **newPos** with a pair which allows for southward movement.

   d. **Ensure that you implement all other methods in the StraightMovement interface in a similar fashion to moveNorth() & moveSouth().**

4. **`Skipper:`** You have been given this file and must update it to implement the **StraightMovement** interface along with the additional method below:

   a. **int[] move():** This method is used to get the row&col position the animal wants to move to. **It does NOT update the animal's actual position.** The method must randomly choose a direction from the ones **valid for this type of animal.** (Note: 1 = north, 2 = south, 3 = east, 4 = west, 5 = northeast, 6 = northwest, 7 = southeast, 8 = southwest.) This method will call the appropriate method (i.e. moveNorth, moveSouth, etc..) This method returns the new position the animal wishes to move to (i.e. **newPos** array holding row & col)

      This method also implements the skipper's skipping action following game rules:

      - can **skip** across **two** cells in a **single direction** (N, S, E, W)
      - have a movement pattern that **alternates** between a one cell movement and a skip.

   b. **void moveNorth():** This method is implemented from the **StraightMovement** interface. It updates **newPos** data member with a new row/col pair which would allow for movement north.

   c. **void moveSouth():** Similar to the **moveNorth()** method but updates **newPos** with a pair which allows for southward movement.

d. Ensure that you implement all other methods in the StraightMovement interface in a similar fashion to moveNorth() & moveSouth().

5. **`Spook:`** You have been given this file and must update it to implement the **StraightMovement** interface along with the additional method below:

   a. **int[] move():** This method is used to get the row&col position the animal wants to move to. **It does NOT update the animal's actual position.** The method must randomly choose a direction from the ones **valid for this type of animal.** (Note: 1 = north, 2 = south, 3 = east, 4 = west, 5 = northeast, 6 = northwest, 7 = southeast, 8 = southwest.) This method will call the appropriate method (i.e. moveNorth, moveSouth, etc..) This method returns the new position the animal wishes to move to (i.e. **newPos** array holding row & col)

   This method also implements the spook's crazy skipping action following game rules:

   - can move **one** cell at a time (N, S, E, W)
   - can **skip** across **two** cells in **any combination of direction** (N-N, N-E, N-W, E-N, E-E, S-S, S-W, etc…) except a combination that results in no movement (e.g. N-S, W-E, etc…)
   - have no movement pattern (i.e. randomly move one cell or skip); however, **cannot** skip for two successive rounds

   NB: You may also implement the **DiagonalMovement** interface in this class if you require it.

   b. **void moveNorth():** This method is implemented from the **StraightMovement** interface. It updates **newPos** data member with a new row/col pair which would allow for movement north.

   c. void moveSouth(): Similar to the **moveNorth()** method but updates **newPos** with a pair which allows for southward movement.

   d. Ensure that you implement all other methods in the StraightMovement (& DiagonalMovement if needed) interface(s) in a similar fashion to moveNorth() & moveSouth().

6. **`GameBoard:`** You have been given this file and must update it with the additional methods below:

   a. **int getNumHunters():** Accessor method for numHunters.

   b. **int getNumAnimals():** Accessor method for numAnimals.

   c. **boolean validateHunterMove(int[] destination):** This method is used to validate the parameter **destination**. (Note: data in array is stored [0]-row, [1]-column). Valid destinations are empty board positions or ones which contain an animal; they return **true**. Invalid destinations will return **false**.

   d. void moveHunters(): Method used to implement the movement of hunters. This method will allow the user to move all of their hunters.

   - For each hunter:

- the user must enter the direction (Note: 1 = north, 2 = south, 3 = east, 4 = west, 5 = northeast, 6 = northwest, 7 = southeast, 8 = southwest)
- Based on the user input, destination position is determined using the hunter's **move** method.
- The destination must be validated using **validateHunterMove** method. An invalid destination will require the user to re-enter.
- Decrement the hunter's energy.
- If an animal is at the destination, it will be eaten i.e. update hunter's energy based on game rules and remove animal from board.
- Relocate hunter to destination position.

- If all hunters have expired, the board becomes inactive using the class' **isBoardActive** data member.
- If all animals have been eaten, the board becomes inactive using the class' **isBoardActive** data member.

e. **boolean validateAnimalMove(Animal animal, int[] destination):** This method is used to validate the parameter **destination**. (Note: data in array is stored [0]-row, [1]-column). Valid destinations are empty board positions other than caves; they return **true**. Invalid destinations will return **false**.

f. void autoMoveAnimal(Animal currAnimal): Method used to automatically move animals (without user intervention).

i. Determine animal's destination using its **move** method.

ii. The destination must be validated using **validateAnimalMove** method. A valid destination will require the animal to be relocated to that new position; an invalid destination results in the animal being left in its original position.

g. **public void moveAnimals():** Method used to implement the movement of animals. This method will move each active animal in the **animalTracker** by calling **autoMoveAnimal** method.

7. **`HungryHunters`** which contain **`main`** and does the following:

a. Welcomes the user to the game.
b. Creates an instance of **`GameBoard`**.
c. Sets up the game board.
d. Displays the game board.
e. Allows user to engage in gameplay as long as the board is active (i.e. at least one hunter and one animal are active/alive) and the maximum number of rounds of 30 has not been reached.
   i. Displays the board
   ii. Moves hunters
   iii. Moves animals
f. Message will be displayed indicating if player lost or won game. Player wins if they make it past the last round with at least one hunter alive.

8. **For this assignment, you will need to implement the phase (light/dark) and cycle features of the game, keeping track of them and implementing the ability of animals to eat hunters during the correct cycles.**

DO NOT CHANGE THE FIELD OR METHOD NAMES OR DEFINITIONS. YOU ARE REQUIRED TO USE EACH OF THE FIELDS AND IMPLEMENT EACH OF THE METHODS AS GIVEN. YOU MAY CREATE ADDITIONAL FIELDS AND METHODS, IF NECESSARY, TO COMPLETE YOUR PROGRAM.

Deliverables

1. **Assignment 2 is due for submission on Friday 16th November 2018 by 11:50pm via the Moodle/eLearning submission tool or via tessa.king-inniss@cavehill.uwi.edu (only if the tool does not work).**
2. **Only ZIP files will be accepted.** No other compression types should be used.
3. You must submit a **Plagiarism Declaration Form** with this assignment.
4. This assignment is worth **10**% of your final course mark.

PLEASE NOTE: The specifications for this assignment are subject to change. You will be notified if any such changes were to occur.