PACFISH

TECHNICAL REPORT

Jennifer Mou || Cece Tsui

CS230: Final Project Phase II: Project Specifcation

18 May 2015

Table of Contents

A. THE ADTs USED

1) LINKEDLIST
    a. We used two LinkedLists in our program. One LinkedList stored all of the fish in the tank that PacFish has not eaten yet. The other LinkedList stores all of the sharks in the tank that PacFish must avoid. We used the LinkedList data structure because it allowed us to traverse through all of the fish and sharks and move them accordingly. It also allowed us to easily remove the fish that has been eaten from the LinkedList. This is much easier than storing it in an array where we would have to constantly re-adjust the array. Although using a Hashtable would be better when finding and removing elements, we chose not to use it because finding a unique key value to store the rest of the information of the fish was complicated. In using a LinkedList, all the information of each object was in one place and easily accessible. Additionally, we have a small number of objects that are placed in the LinkedList, and so the efficiency of traversing the LinkedList was not detrimental to the overall running time of our program.
2) STACK – LINKEDSTACK
    a. We used a Stack of type Fish to keep track of what the PacFish can and cannot eat. The Fish will be pushed onto the Stack in a sorted order so that the smallest size will be popped first. PacFish must eat five fish before leveling up. After eating five fish of the popped size, another Fish will be popped from the Stack, and when the PacFish tries to eat a fish, the fish's size will be compared to the size that is popped. If they match, PacFish can eat the fish and continue on. If not, the game is over. This will continue until no more fish can be eaten (aka the stack is empty) or until the game is over. We used a Stack because its push and pop methods allow us to store multiple values and only focus on the value that is popped. For example, if we were to use an array, we would need to traverse the array to find the next size, and then removing values from the array would require making another array and copying over values. We chose to use a LinkedStack vs another implementation of Stack because reassigning what the variable top is referring to is efficient.

B & C. LIST OF CLASSES AND METHODS

NOTE: We've included all the classes and methods we used. The first class utilizes the ADTs that we described earlier. The rest of the classes are either for the GUI or an object that represents the player and the opponents.

1) *<u>GamePanel Class</u>*
   a. *''The GamePanel class will create the game board on the GUI. It will create the game board using a grid of labels (a 2D array), as well as add necessary labels and buttons for the player to utilize. It will also initialize all the necessary components of the game, such as a LinkedList of all the fish the PacFish should eat and the Stack that keeps track of what the PacFish can eat. This class will be the one to move the PacFish dependent upon the user's keyboard input (only the arrow keys), change the size of the PacFish if it levels up, and handles the game if it is over. All other fish in the tank will move according to a timer.'''*
   b. *Instance Variables*
      i. int cellSize
         1. The size of the labels on the game board. This is used to ensure the all labels on the grid will have the same size.
      ii. int rowsAndColumns
         1. The size of the game board.
      iii. PacFish pacFish
         1. The player.
      iv. LinkedStack <Fish> whatCanBeEaten
         1. Keeps track of what size the PacFish can and cannot eat. (ADT)
      v. Fish prey
         1. The variable that keeps track of the type of fish that the PacFish can eat.
      vi. LinkedList<Fish> fishInTank, shark
         1. Stores all of the fish in the tank. fishInTank is the LinkedList of all the fish the PacFish has not eaten yet, while shark is all the sharks that PacFish must avoid. (ADT)
      vii. JLabel[][] game
         1. 2D array that stores all of the labels. It represents the game board (aka, the tank).
      viii. ImageIcon TITLE_IMAGE, FISHHOOK_IMAGE, CONGRATS_IMAGE, LOSE_IMAGE
         1. The images used for the GUI.
      ix. Timer timer
         1. The TimerListener that moves the fish every 200 milliseconds.
      x. boolean startTimer
         1. Keeps track of whether the timer is currently running.
      xi. JButton startOverButton, quitButton
      xii. JLabel countLabel, statusLabel, targetPicLabel, titleLabel, targetWordsLabel, avoidLabel
      xiii. JPanel gameBoard, information

xiv. DirectionListener movementListener
   1. An KeyListener that moves the PacFish based on the arrow keys.

c. *Constructor*
   i. Takes in one parameter, level – an integer that represents what level the user chooses.
   ii. Builds the GUI and creates the game based on the level (the game board gets smaller as the difficulty increases).

d. *moveFish*
   i. A helper method that takes in a Fish object, and integer values representing the change in row and column value of the Fish movement in the game board. It will move the Fish accordingly.

e. *moveAllFish*
   i. A method that takes in a LinkedList of Fish. It traverses the list. For each fish, it randomly generates a movement. It will then check if the new placement of the fish is within the game board and will not overlap another fish, shark, or fish hook. If so, it will call the helper method moveFish to move this individual fish. It will also check if the new placement is where PacFish is. If they do overlap and the size of the fish is bigger than the PacFish, we handle game over.

f. *movePacFish*
   i. A method that takes in integer values representing the change in row and column value. It will check if the new placement is within the game board. If so, it then checks if PacFish hit any fish, shark, or fish hook and whether it eats or dies. If it eats, it changes the countLabel and increments the count of number of fish eaten. If it dies, it handles game over.

g. *handleGameOver*
   i. Handles when the game is over. There are two cases: if the PacFish wins by eating all the fish in the tank, or if the PacFish loses by dying. It will remove PacFish from the tank, changes the labels accordingly, and stop all movement in the tank.

h. *handleSizeChange*
   i. If PacFish eats all of the fish it can eat at the time in the tank, it will level up. The status and count label changes to represent the new level. A new prey is selected, and at each level after level 1, sharks are added to the tank.

i. *randomlyPlaceComponent*
   i. *A helper methods that helps place the image randomly on the gameboard. It takes in an object (Fish) and the image to place randomly. If there is an object entered, meaning that the object corresponds to the image, then the object's placement will be changed to the random placement.*

j. *DirectionListener (Implements KeyListener)*
   i. A private class that moves PacFish when an arrow key is pressed using movePacFish. It also handles when the game is paused or unpaused. It also checks if the tank does not have any more fish. If so, it will handle game over.

k. *TimerListener (Implements ActionListener)*

- i. A private class that moves all other fish and sharks in the tank every 200 milliseconds.

l. *ButtonListener (Implements ActionListener)*
- i. A private class that reacts to when the Buttons startOver and quitButton are clicked.

2) <u>Fish Class</u>

a. ‘’’The Fish class will be a representation of the fish-computer players on the game board. It will keep track of where the fish is on the game board, its size, and the picture of the fish. This class will make it so that it all the information is stored easily within the class and can be retrieved simply with a method.’’’

b. Instance Variables
- i. int placement_r
  1. Indicates the row location the fish is on the game board.
- ii. int placement_c
  1. Indicates the column location the fish is on the game board.
- iii. int size
  1. The bigger the integer value, the bigger the fish is.
- iv. ImageIcon fishPic
- v. final String[] FISHES
  1. A final variable that will be an array of all the name of the fish images (as String values), ordered from smallest to largest.

c. Constructor #1
- i. This constructor takes in three parameters – an integer variable for size, and integer values indicating the row and column placement of the fish and initializes the Fish’s information accordingly.

d. *Contructor #2*
- i. This constructor takes in one parameter: an integer variable for size
- ii. The placement of the fish is set to be off the gamebaord and fishPic is initialized to be an image dependent on the given size.

e. *setImage*
- i. Takes in ImageIcon and sets fishPic to be it.

f. *getImage*
- i. Returns fishPic.

g. getPlacementR
- i. Returns the row placement of the fish on the game board.

h. setPlacementR
- *i.* Sets the new row placement of the fish on the game board.

i. getPlacementC
- i. Returns the column placement of the fish on the game board.

j. setPlacementC
- *i.* Sets the new column placement of the fish on the game board.

k. getSize
- *i.* Returns the size of the fish on the game board.

l. *setSize*
- i. Takes in new size value as integer and sets new size.

3) <u>*PacFish Class (extends Fish)*</u>
    a. *‘’PacFish class inherits from the Fish class. This will be a representation of the PacFish player (otherwise the main player) in the game. It will not only keep track of the placement and size, as in its parent class, but it will also keep track of the number of fish that it has eaten and be able to change its size (to show that it levels up).’’’*
    b. *Instance* Variables
        i. int numFishEaten
            1. Keeps track of the number of fish it has eaten.
        ii. final ImageIcon PACFISH_IMAGE
            1. Picture of pacFish player.
    c. *Constructor*
        i. Takes in integer values representing row and column placement.
        ii. Calls the constructor from parent class.
        iii. Sets image to the PacFish image.
        iv. Initialize numFishEaten to be 0.
    d. getNumFishEaten
        i. Returns numFishEaten.
    e. increaseNumFishEaten
        i. Increments value of numFishEaten.
    f. levelUp
        i. This method is called when pacFish player eats enough fish to level up
            1. Size is incremented and numFishEaten is reset to zero

4) <u>*PacFishStartupPanel Class (extends JPanel)*</u>
    a. ‘’This class creates the first GUI that appears when the game is launched. It shows a GUI that allows the user to choose the difficulty level they want to play. After pressing the difficulty, it launches another GUI that shows the game board (GameBoardPanel).’’’
    b. Instance Variables
        i. JButton easy, medium, hard, quitButton
    c. Constructor
        i. Builds the GUI with welcome message and necessary buttons for player to begin game.
    d. ButtonListener (Implements ActionListener)
        i. Private class that creates an instance of the GameBoardPanel according to the JButton pressed or exits the game if quitButton is pressed.

5) <u>*PacFishStartup Class*</u>
    a. *‘’This class runs the entire PacFish game. It launches the PacFishStartupPanel, and the user interacts with that.’’’*
    b. *Main Method*
        i. Launches the GUI.