A Game of Thrones

1 Introduction

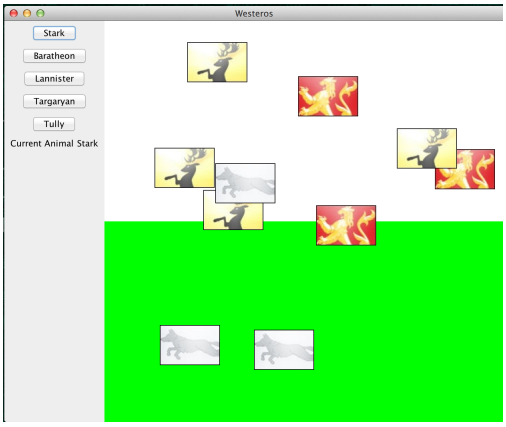
Welcome to the world of Westeros, where you either play the Game of Thrones or die. (Not really though.)

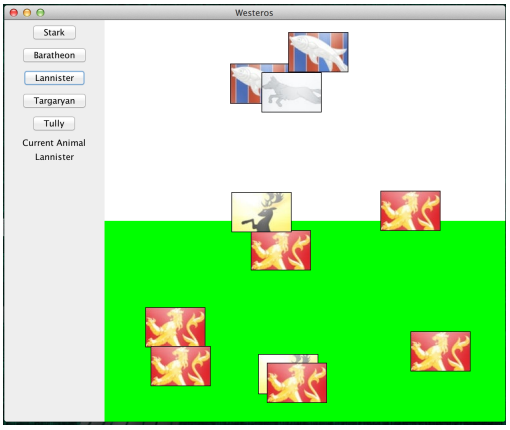
Note: You don't need to know anything about the show/books in order to understand this assignment. We are purely trying to put an interesting theme on a classic homework problem.

You will be given a GUI that represents the world, both the snowy north, and the grassy south. Within this world you will build classes that represent six Houses (families) from the show. You will then form logic to make people (instances) from those Houses interact with other Houses by either reproducing or fighting an instance of the other House.

Game of Thrones is complicated and the interactions between Houses are also complex. You should be reading through this document slowly and taking notes as you go along so that you truly understand what each House is supposed to do when they collide with another House.

Once you've read through everything, you should write up a class hierarchy along with the methods you'll need to implement. We still expect top notch OOP design!





2 Westeros

The world in which Game of Thrones takes place is called Westeros. As such, we have provided you with three files that represent your GUI: `GameOfThrones.java, Westeros.java`, and `ControlPanel.java`. Also, we have included `House.java` with one method filled out for you.

- `GameOfThrones.java` is the driver for your program.
- `Westeros.java` is the panel which your various House objects will be running around in. Each House is represented by a sigil, which is what you will see moving around in your GUI.
- `ControlPanel.java` is a panel with buttons that will control which House gets instantiated next onto the GUI
- `House.java` is the abstract parent class for your other Houses of Westeros. We are giving you the shell for this class with the `draw(Graphics g)` method already written for you.

Because we are giving you a GUI structure to work with, we will require you to name some methods and classes in a specific way. If you see the name of a class or method that you need to write mentioned in this document, you should name it as such.

You will need to change lines 44 and 45 in ControlPanel.java as well as lines 175 and 176 in Westeros.java to accommodate your custom house. More to be explained below.

3 House Hierarchy

The main purpose of this assignment is to implement the hierarchy of Houses within the world of Westeros. Remember, you will be making instances from these houses that run around in your GUI and interact with other instances of other houses (you can think of each House object as a person from that House if you'd like). We will be giving you a little bit of guidance on specific methods to write, but a large part of this assignment will be you designing classes on your own!

3.1 `House Class`

The top level of your hierarchy should be an abstract class called `House`. We have provided you with a stub for this class that includes the draw(Graphics g) method implemented. This class will have certain instance variables and methods that all Houses will have in common.

Remember what the purpose of an **abstract method** is and when to use them. We won't tell you which of these methods should be abstract, but you should make good design decisions and figure it out on your own.

Here are the methods that the GUI requires from `House`. You *must* implement these methods for the GUI to work. You may add more methods if you'd like, but always remember to use good OOP design.

- `move()` will move the instance of a House in a random yet effective manner (i.e. it doesn't move in circles). Each time the instance moves, its health should decrease and its age should increase.
- `collidesWithHouse(House otherHouse)` returns whether or not this current instance of a House is colliding with another given instance of a House. This should be determined by the instance's location and the dimensions of its image (hint - check the API for this).
- `canReproduceWithHouse(House otherHouse)` returns whether or not the two instances can reproduce. This will be detailed in the specific House descriptions below.
- `reproduceWithHouse(House otherHouse)` if the two Houses are able to reproduce (as determined by `canReproduceWithHouse()`) this method returns a new House of the same type and in the same location. If for some reason, re-production does not happen, null should be returned. Even if one house *can* reproduce with another, you should still limit the chance of reproduction somehow such that you don't infinitely reproduce. Think about giving reproduction some random chance of occurring if the houses are colliding with each other.
- `isOld()` returns whether or not an instance of a House has surpassed some determined maximum age.
- `canHarmHouse(House otherHouse)` determines whether or not the current instance of a House can harm an instance of the other house through mortal combat.
- `harmHouse(House otherHouse)` if the current house can harm the other house (as determined by `canHarmHouse()`) then it decreases the other House's health by some amount. How lethal Houses are is up to you!
- `die()` called when the instance dies for one reason or another
- `isDead()` returns whether or not the instance is dead (ie health = 0)

Houses in Westeros will be divided geometrically. Two will be from the North, two will be from the South, and one does not have a home (think about where that one needs to fit in your hierarchy). Being from the North or from the South grants you a special ability when you are roaming in your own lands.

3.2 `NorthHouse`

Northerners are hardy, strong fellows. As such, they get a health bonus when roaming in their own lands. When a NorthHouse moves in the northern half of your GUI, you should increase its health by some value.

3.3 `SouthHouse`

Southerners thrive on the grasslands that populate southern Westeros. As such, they get a speed boost when roaming in their own lands. When a SouthHouse moves in the southern half of your GUI, you should find some way to make them move faster than when they are in the north (perhaps some speed instance data?).

4 Specific Houses

And now to introduce the Family Houses of Westeros that you will be building for the simulation. Remember to go through this part slowly as each house has specific ways of interacting with each other house. When naming these classes, leave out the word "House" i.e. name them Stark, Tully, Lannister, etc...

4.1 House `Stark`

The cold Stark family from the north is represented by the fierce Direwolf on a gray field. Their words are: "Winter is Coming." Their image can be found in the provided files and is named `direwolf.png`.

- Members of House `Stark` only reproduce with members of House `Tully`
- When colliding with a member of House `Lannister`, House `Stark` has a 40% chance of wounding them.
- Members of House `Stark` live to an older age than anyone other than House `Tully`

4.2 House `Tully`

The honorable House Tully from the north is represented by a Trout on a field of blue and red. Their words are: "Family, Duty, Honor." Their image can be found in the provided files and is named `trout.png`.

- Members of House `Tully` only reproduce with members of House `Stark`
- When colliding with a member of House `Lannister`, House `Tully` has a 20% chance of wounding them.
- Members of House `Tully` live to an older age than any other house

4.3 House `Lannister`

The wealthy House `Lannister` from the south is represented by a gold Lion on a crimson field. Their words are: "Hear me roar!" Their image can be found in the provided files and is named `lion.png`.

- Instances of House `Lannister` reproduce with instances of House `Baratheon` *or* House `Lannister`
- When colliding with a member of House `Stark`, House `Lannister` has a 60% chance of wounding them.
- When colliding with a member of House `Tully`, House `Lannister` has an 80% chance of wounding them.
- Members of House `Lannister` live to the smallest age of all of the houses.

4.4 House `Baratheon`

The mighty House `Baratheon` from the south is represented by a black Stag on a gold field. Their words are: "Ours is the fury". Their image can be found in the provided files and is named `stag.png`.

- Instances of House `Baratheon` reproduce with instances of House `Lannister`.
- House `Baratheon` can wound members of House `Targaryan` but no other house.
- Members of House `Baratheon` live to an older age than House `Lannister`, but no other house.

4.5 House `Targaryan`

House `Targaryan` has no homeland and is represented by a red Dragon on a black field. Their words are: "Fire and blood". Their image can be found in the provided files and is named `dragon.png`.

- Instances of House `Targaryan` only reproduce with other instances of House `Targaryan`
- House `Targaryan` can wound members of all other houses, other than House `Baratheon`
- Members of House `Targaryan` live forever (unless they die in the glory of combat).

4.6 Make Up Your Own!

You should add one more house to the realm of Westeros. Feel free to name it whatever you want and interact with the other houses however you want! If you know the books, feel free to add another house that's not here (House Martell is a personal favorite :D). You can make your own symbol at this website: `http://www.jointherealm.com/` or just use any old picture.

The only requirements for this house is that it fits somewhere logical in your hierarchy, and it has some uniqueness to it that is different than the houses that already exist (Do they move faster? Destroy different things? Reproduce differently? etc...)

In order to add your house to the GUI, we need you to make 4 changes to our GUI code. Don't worry, it's not hard! Simply change the words CUSTOM_HOUSE in lines 44 and 45 in ControlPanel.java as well as lines 175 and 176 in Westeros.java to the name of your custom house file to make things work. After doing this, remove the TODO comments near these lines to fix Checkstyle.

5 General Tips

Here are some general tips for this assignment!

- Re-read this entire document. It's long, it's complicated, but it's meant to be like that. Oftentimes you will deal with complicated problem specs that may not make much sense the first time you read it. Take notes, draw pictures, and make sure you know *exactly* what's going on **before** you start coding.
- Remember, a large part of this assignment is a good hierarchy and good OOP design, so really put a lot of time and effort into designing this in a smart way.
- We will be grading largely on functionality, although we will of course be looking at your code too. Make sure we can actually *see* the interactions we've described here between the houses! If your reproduction rate is too high or too low, we might not be able to tell if your program is doing what it's supposed to be.
- It's ok if on a collision, both houses kill one another, or both houses reproduce with the same partner.

6 Javadocs

We are going to have you do Javadocs for this assignment (and for all assignments here on out). Javadocs are a clean and useful way to document your code's functionality. For more information on what Javadocs are and why they are awesome, the online documentation for them is very detailed and helpful. The relevant tags that you need to have are `@author`, `@version`, `@param`, and `@return`. Here is an example of a properly Javadoc'd class:

```
import java.util.Scanner;

/**
* This class represents a Dog object.
* @author Ethan Shernan
* @version 1.0
*/
public class Dog {

    /**
     * This method takes in two ints and returns their sum
     * @param a, b
     * @return their sum
     */
    public int add(int a, int b)){
        ...
    }
}
```

7 Checkstyle

Just in case you forget how to do Checkstyle, here are the instructions again!

You have been provided with a Checkstyle Jar file and the CS 1331 Checkstyle configuration file (`cs1331-checkstyle.xml`). We do this to make sure you are following the proper style guidelines as posted in the Style Guide under T-Square resources. Please refer to these guidelines for any help as you are writing your programs.

We will run checkstyle on all the Java source files you submit. You can run Checkstyle with the command below (in the directory containing all your Java source files):

```
$ java -jar checkstyle-5.6-all.jar -c cs1331-checkstyle.xml *.java
Starting audit...
Audit done.
```

The message above means there were no Checkstyle errors. You can easily count Checkstyle errors by using the command below and subtracting 2 from the number printed (which is how we will deduct points). For example:

```
$ java -jar checkstyle-5.6-all.jar -c cs1331-checkstyle.xml *.java | wc -l
2
```

Alternatively, on Windows:

```
C:\> java -jar checkstyle-5.6-all.jar -c cs1331-checkstyle.xml *.java |
findstr /v "Starting audit..." | findstr /v "Audit done" | find /c /v "hashcode()"
0
```

These both mean there are no errors.

If you have any questions about a Checkstyle error, please let a TA know! They should be pretty self-explanatory and you should be able to fix them easily. We will be taking points off for checkstyle on this assignment, so please make sure you run it! The checkstyle cap for this homework is **30** points.

8 Turn-in Procedure

Submit all of the Java source files and images your program requires to run to T-Square. Do not submit any compiled bytecode (`.class` files). When you're ready, double-check that you have submitted and not just saved a draft.

Please remember to run your code with Checkstyle!

Verify the Success of Your Submission to T-Square
Practice safe submission! Verify that your HW files were truly submitted correctly, the upload was successful, and that the files compile and run. It is solely your responsibility to turn in your homework and practice this safe submission safeguard.

1. After uploading the files to T-Square you should receive an email from T-Square listing the names of the files that were uploaded and received. If you do not get the confirmation email almost immediately, something is wrong with your HW submission and/or your email. Even receiving the email does not guarantee that you turned in exactly what you intended.

2. After submitting the files to T-Square, return to the Assignment menu option and this homework. It should show the submitted files.

3. Download copies of your submitted files from the T-Square Assignment page placing them in a new folder.

4. Recompile and test those exact files.

5. This helps guard against a few things.

   (a) It helps insure that you turn in the correct files.

   (b) It helps you realize if you omit a file or files.[1] (If you do discover that you omitted a file, submit all of your files again, not just the missing one.)

   (c) Helps find last minute causes of files not compiling and/or running.

[1] Missing files will not be given any credit, and non-compiling homework solutions will receive few to zero points. Also recall that late homework will not be accepted regardless of excuse. Treat the due date with respect. The real due date is midnight. Do not wait until the last minute!