CSCI165 Computing Fundamentals I
Final Project
Spring 2020
Hobbits vs Nazgul

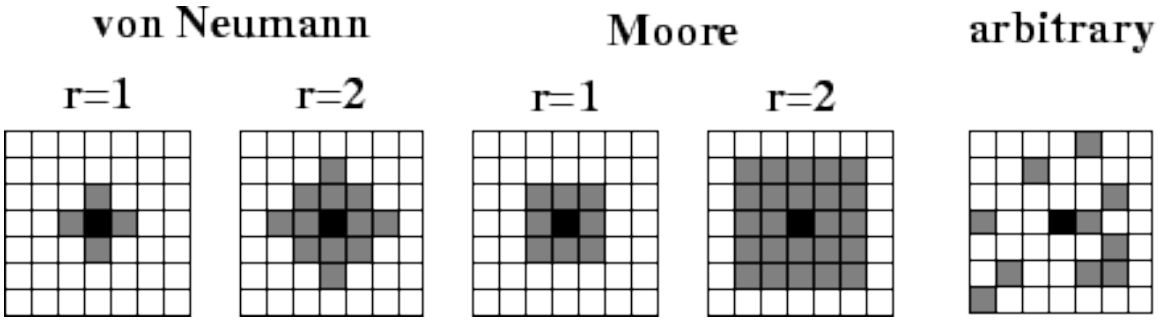
Worth 100 points

The goal for this programming project is to create a simple 2D predator–prey simulation implemented as a cellular automaton.

Cellular Automaton

A cellular automaton consists of a regular grid of *cells*, each in one of a finite number of *states*, such as *on* and *off*. The grid can be in any finite number of dimensions. For each cell, a set of cells called its *neighborhood* is defined relative to the specified cell.

The image below shows the von Neumann and Moore neighborhoods with sample radius. Any type of neighborhood can be defined as long as it is uniform and finite.



An initial state (time $t = 0$) is selected by assigning a state for each cell. A new *generation* is created (advancing t by 1), according to some fixed *rule* (generally, a mathematical function) that determines the new state of each cell in terms of the current state of the cell and the states of the cells in its neighborhood. Typically, the rule for updating the state of cells is the same for each cell and does not change over time and is applied to the whole grid simultaneously.

See Conway's Game of Life for a neat example of a cellular automaton:

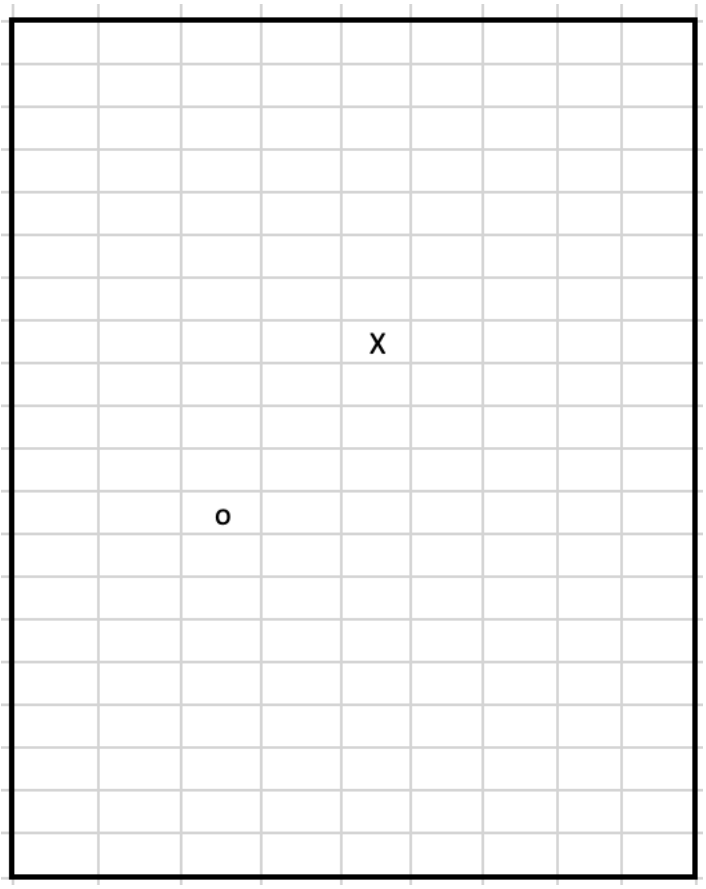
https://playgameoflife.com/

In this simulation we will model the interaction of two types of creatures: predator and prey; where part of the movement motivation of the prey is to move *away* from the predator and vice versa. An object moves by changing cell locations. An object can only move to one of the cells in its neighborhood with a radius of 1.

In our simulation, the prey are Hobbits, and the predators are Nazgul. These beings live in a fictional 2D world composed of a 50 X 50 (or whatever size you choose) grid of cells (think two-dimensional array). This grid of cells represents the surroundings and terrain of the world. There may be trees, rocks, buildings . . . etc. Only one being may occupy a

cell at a time and the Hobbits are constantly on the run from the voracious Nazgul.

The **X** could represent a Nazgul and the **o** could represent a hobbit. Depending on the radius you implement for your characters, the Nazgul may or may not be able to sense the direction of the Hobbit and vice versa. Realize that large ***r*** values could drastically effect performance.



Around the edge of this world there is an ominous containing wall. It is over emphasized in the following image. Due to the casting of some arcane magic, the Nazgul are not capable of crossing this wall. They are essentially repelled and must change direction. On the other hand, the Hobbits, who appeased the witches with hand crafted ales and excellent long bottom leaf are capable of permeating these walls and teleporting to the other side of town. When a Hobbit approaches a particular edge, it can wrap around to the opposite edge and appear there, similar to the mythological beast Paccus Mannus.

As this world is a zero-player automaton, it does not require any human interaction. One interacts with the game by creating an initial configuration and observing how it evolves. In this simulation you should observe patterns of Nazgul pursuing Hobbits around the grid, with the Hobbits in turn, fleeing from the Nazgul.

Time

Time in this world is simulated in discreet steps which can be implemented as a loop iteration. A single iteration over the entire world is equivalent to an earth minute. Each being performs some action every time step and the *state* of the creature changes. For example, a Hobbit can only go so many time steps without eating or it withers and dies.

First breakfast, second breakfast, elevenzies, lunch, brunch, brinner, dinner, supper, snack, second snack, midnight snack . . . these milestones are important to the Hobbit life cycle. The more time steps a Hobbit takes without sustenance, the closer it gets to dying from starvation. This can be implemented with a simple integer counter and a series of colors to represent the various stages of hunger that a Hobbit can exhibit.

In this simulation our Nazgul, being undead, will gain sustenance from the stabbing of a Hobbit with a morgul blade. The longer a Nazgul goes without this, the closer they get to perishing. This can be implemented with a simple integer counter and a series of colors to represent the various stages of withering that a Nazgul can exhibit.

Items

The creatures in this world can possess an inventory of items. The items are of a variety, with each type having a particular ability. Items can be magical, can be of medicinal use, weapons or defense. Again, you have the ability to be as creative as you like. A Creature can have an ArrayList of Items in their possession. Maybe only certain types of Creatures are able to pick up and use certain types of items

An example of an item could be a form of armor (such as mithril from LotR) that is particularly resistant to being breached by a sword. If a hobbit is donning mithril the damage caused by a morgul blade would be lessened and the benefit of the stabbing to the Nazgul would also be diminished. If a human is holding a shield, an attack with a sword would be lessened.

Another cool example could be a magical compass that increases a characters ***r*** value for its neighborhood, allowing it to sense others from a larger distance.

You could equip your creatures with items during the seeding of the world, or you could plant items throughout the grid as a form of treasure. When a creature enters a cell with an Item the creature could add it to the inventory, if it is able. There are no strict rules on how this should happen.

Are all Creatures equip-able?

Shared Behaviors

- Each creature should have definitions of the following methods. The arguments for these methods can be adapted to your particular situation. More behaviors can be added. The **minimum requirement** is for these behaviors to be called polymorphically without checking for type.
- **void move():** Called by the simulator when a creature moves.
- **void attack(Creature c):** Called by the simulator when a creature attacks another Creature. This method will affect the state of the Creature argument by diminishing its health if this is the result.
- **Creature replicate():** Called by the simulator when a creature replicates.
- **void stay():** Called by the simulator when a creature replicates. This method may affect the internal state of **this** Creature by diminishing health
- **chooseAction(Map<Direction, Occupant> neighbors):** Chooses from the above four actions based on the neighborhood.
- **Color color():** Provides a color for the creature (usually used to indicate health).

The Creatures will proceed through a series of colors to communicate their health. Choose colors to make this obvious. Perhaps Nazgul proceed through a series of Black to Grey to White to indicate their withering. Hobbits could proceed through a series of Green to Yellow to Red to indicate starvation. Each time step in the world will affect the internal state of the Creature in some meaningful way.

The methods are intended to make changes only to the internal state of each creature, and not the outside world. Creatures can act on other Creatures via argument passing.

While each species is of the same genetic lineage and share the same basic behaviors, the implementation is vastly different. The two basic behaviors are **Move** and **Replicate.** Your task is to create an inheritance hierarchy that allows for these behaviors to be triggered ***polymorphically.***

The Hobbits behave according to the following model:

- **Move:** Each time step the Hobbit will scan the neighborhood radius and
    - Move in the exact opposite direction from any Nazgul if one is present. If there are multiple Nazgul you will need to make a decision. Perhaps the possession of a certain golden circular band can allow clever behaviors.
    - If no Nazgul are within the radius, the Hobbit will move toward fellow Hobbits
    - If there are no living beings within the radius, the Hobbit will randomly choose a direction.
    - Hobbits can cross the barrier of the magical wall in all four directions
- **Replicate:** If a Hobbit survives for three time steps, then at the end of the third time step (i.e., after moving), the Hobbit will replicate. This is simulated by creating a new Hobbit in an adjacent neighborhood cell that is empty. If there is no empty cell available, no breeding occurs. Once an offspring is produced, that particular Hobbit cannot produce an offspring until three more time steps have elapsed.

The Nazgul behave according to the following model:

- **Move:** Every time step, if there is an adjacent neighborhood cell occupied by a Hobbit, then the Nazgul will move to that cell and attempt to stab the Hobbit with its Morgul Blade. Otherwise, the Nazgul moves according to the same rules as the Hobbit. Note that Nazgul **must** move toward Hobbits.
- **Breed:** If a Nazgul survives for eight time steps, then at the end of the time step, it will spawn off a new Nazgul in the same manner as the Hobbits.

During one turn, each being should move in the order they appear on the grid. The move and breed behaviors should be called without checking the class type. Checking the class type defeats the purpose of polymorphism and should only be done if you need to invoke a behavior that is not shared among the siblings. In order to achieve this here you can have the polymorphic method definition call the unique behavior.

I am purposefully leaving the design open ended, with you making the ultimate decisions. I will expect an accompanying write-up that explains your design decisions.

Temporary Grid

In order to maintain consistency in the grid you may need a temporary grid to record the moves. For example, . . . if a creature moves down from row 3 to row 4 that creature shouldn't be called on to move again once that row is processed, but that Creature **can** have an effect on a separate Creature in its neighborhood.

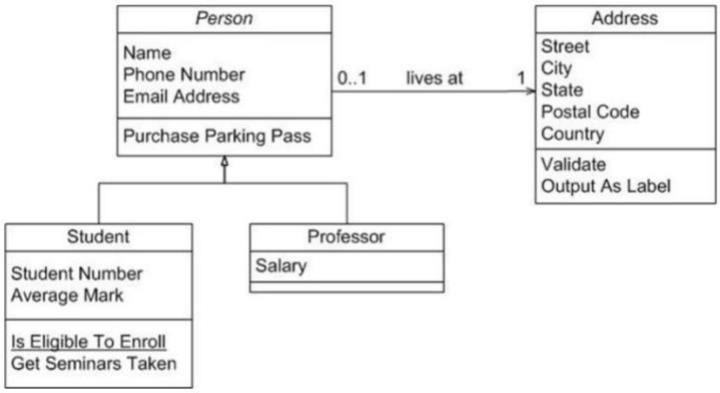
Application

Write a program to implement this simulation and draw the world using Java2D Graphics.

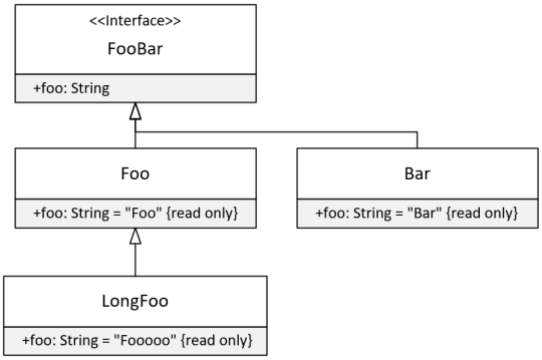
First Deliverable: Inheritance Hierarchies (or interface implementation) with accompanying design write-up.

Due: Monday May 4, 2020 by 9:00 am

- Develop the Hobbit and Nazgul Creature inheritance hierarchy (or interface implementation, you decide). The Top of this hierarchy should be some form of abstraction that contains the shared methods that all classes should implement. Again, the design decisions are up to you. You could use abstract classes or a series of interfaces. Just be prepared to explain your choices.
- Add two additional Creature sub classes of your design. Provide descriptive details of their behaviors. The code does not have to implemented but I do want the ideas flushed out.
- Develop the Item inheritance hierarchy (or interface implementation, you decide). Group related behaviors and characteristics at the top of the hierarchy.
- The methods can be stubs and be as simple as printing a message stating that the method was called. The minimum requirement for this stage is to demonstrate that the methods can be called polymorphically. Define an array of ***some type*** and fill it with Creatures. Write a simple ***for each*** and demonstrate that the methods can be called polymorphically.
- Develop a complete UML diagram, clearly showing all inheritance and composition design details. Here is an example



Be sure to correctly illustrate any interfaces you may be using. Here is a UML example with interfaces



What to expect with part 2 of this project?

1. **Intense unit testing:** These types of programs are quite difficult to verify simply by watching rectangles beep and boop on the screen. Be expected to write numerous unit tests to verify correctness.
2. **World Seeding:** How will you seed your world? Randomly? Config Files? User Input? Think about this.
3. **Implementation:** Full, error free implementation is expected for full credit.
4. **Exception Handling:** maybe . . .
5. **On final exam day we will share our work with each other on Discord**