Jonathan Chen (jc953)
&& Ishaan Jhaveri (iaj8)

Assignment 5: **Interpretation and Simulation**

Entry Class: **Assignment-5.a5.Console.java**

Summary

The most challenging aspect of the assignment was writing an interpreter compatible with the AST implementation provided to us. We needed to use the provided AST because we made the decision to go ahead with the new AST rather than spend time debugging our own AST implementation. We have decided to create a CritterWorld class along with a Hex class to represent each Hex in the CritterWorld. The Hex class contains information pertaining to the amount of food on it, whether it is a rock or not and the Critter that may or may not inhabit it. Critter is a separate class coordinating the state and actions of each individual Critter.

At first an issue arose about how to represent the Hexes because there is no data structure that intuitively represents an array of hexagonal tiles with rows and columns such as the one given to us. So we decided to use a 2d array and have the code modify how it indexes the rows and columns.

We have no known problems with our implementation.

Specification

Package ast holds the classes that represent the Abstract Syntax Tree (AST). Package parse handles parsing a file and creating the corresponding AST. Package a5 interprets the AST for a particular Critter and simulates the corresponding critter, initializing it with rules and a state. All the Critters thus initialized make up the CritterWorld, which is represented through the Hex class that keeps tracks of all other aspects of the critter world like rocks.

Jonathan Chen (jc953)
&& Ishaan Jhaveri (iaj8)

Design Implementation

Classes and Architecture:

Console: This is the entry class of the program. It allows users to create a customized CritterWorld simulate through steps and receive information about the World's state.

Constants: This contains information on all the constants used in CritterWorld. It parses a file provided in the program with the constants inside it, adding their values to the constants of CritterWorld.

Critter: This is an individual critter. It handles a critter's state, position, actions and executes the time steps.

CritterWorld: This is the world. It keeps track of all the Hexes in the world, the critters that exist and the total number of steps taken. It contains information about the world's state.

Hex: This represents each Hex space in the CritterWorld. Contains information about possible food, whether it is a rock and the possible critter in this space

The classes in parse and ast have been documented earlier.

Code Design.

We interpret the AST recursively.

We use 2d Arrays and ArrayLists to represent the Hexes and Critters.

Programming.

We have implemented a ‘depends-on’ relationship between our classes. We use a Bottom-Up approach so we can test each class independently of each other.

Our main challenge was how to represent the Hexes using the Data structures at our disposal. We also were slightly tripped up on deciding how to order the Critters. We finally modified how we index a 2d array and use an ArrayList for ordering.

We use the given AST and parsing code rather that re-using our own implementation from the previous assignment.

Jonathan Chen wrote CritterWorld, Hex and Console. Ishaan Jhaveri wrote Critter and Constants. We then edited what each other wrote. We worked together to edit ast and ParserImpl. We worked together on a5test.

Testing

We created a sample CritterWorld, to check that randomly implementing rocks works, the food field works and critters interact properly with the world. We create a critter or a combination of critters to test each and every action that a critter can perform. Our tests are representative of different condition outcomes too.

Known Problems

No known problems.

Jonathan Chen (jc953)
&& Ishaan Jhaveri (iaj8)

Comments

This is was easier than the previous assignment because we could see the concrete effects of what we were implementing. We are able to constantly test the correctness of our classes independently of each other.

Time: Roughly 25 hours

Advice: Plan ahead how you want to store the Hex information.

Surprising: Layout.

Hard: Layout.

Like: Being able to see the effect of what we code.