CS 6310 Project 2: Heated Earth

Design Description

Reece Karge (rkarge3), Subha Melapalayam (smelapalayam3), Kelly Nielsen (knielsen9), Rohan Parolkar (rparolkar3), Cedric Samuel (csamuel6)

Overview

Software engineering often involves making compromises between non-functional requirements and functional design. During this exercise we produced a program to simulate heat transfer across Earth's spherical surface given simplifying assumptions about the attributes of the Sun's radiation and Earth's conductivity. Through developing this program we considered implementing concurrency, initiative, and buffering mechanisms to meet the requirements specified. The program was divided into three main components: Simulation, Presentation and Graphical User Interface (GUI). This design enabled the components to be loosely-coupled which allowed the team members to work on them independently. In utilizing this approach we reduced the time wasted and were able to work more efficiently.

The most troubling aspect of this design was implementing the simulation. Given the time constraints and the complexity of the physics involved in representing a truly accurate scenario, designing the simulation proved to be extremely challenging. However, from the discussions on the class forum and subsequent updates to the domain model guidelines we decided to simplify the approach and focus on ensuring that the behavior of the model illustrated the sun's impact accurately i.e. cells within the sun's range heat up faster than they cool while those out of range cool.

Other approaches could have included:

CS 6310 Project 2: Heated Earth

Design Description

1. Working out the physics involved in creating a truly realistic model. However that would have increased the time spent designing and coding significantly leaving us with little time to prepare the other requirements.
2. Leveraging external libraries which already have implemented such logic. However, this would have defeated the purpose of the assignment which was to come up with our own design.

Program Characteristics

Table 1. Program Metrics

Component	Lines of code	Total size of class files	Number of classes	Avg number of operations	Avg number of attributes	# of inter-class dependencies
GUI	329	10 KB	1	6	17	0
Presentation	174	6 KB	1	14	20	2
Simulation	508	20 KB	4	20	7.5	1
Demo(main)	51	2 KB	1	0	2	0

UML Class Diagram

CS 6310 Project 2: Heated Earth

Design Description

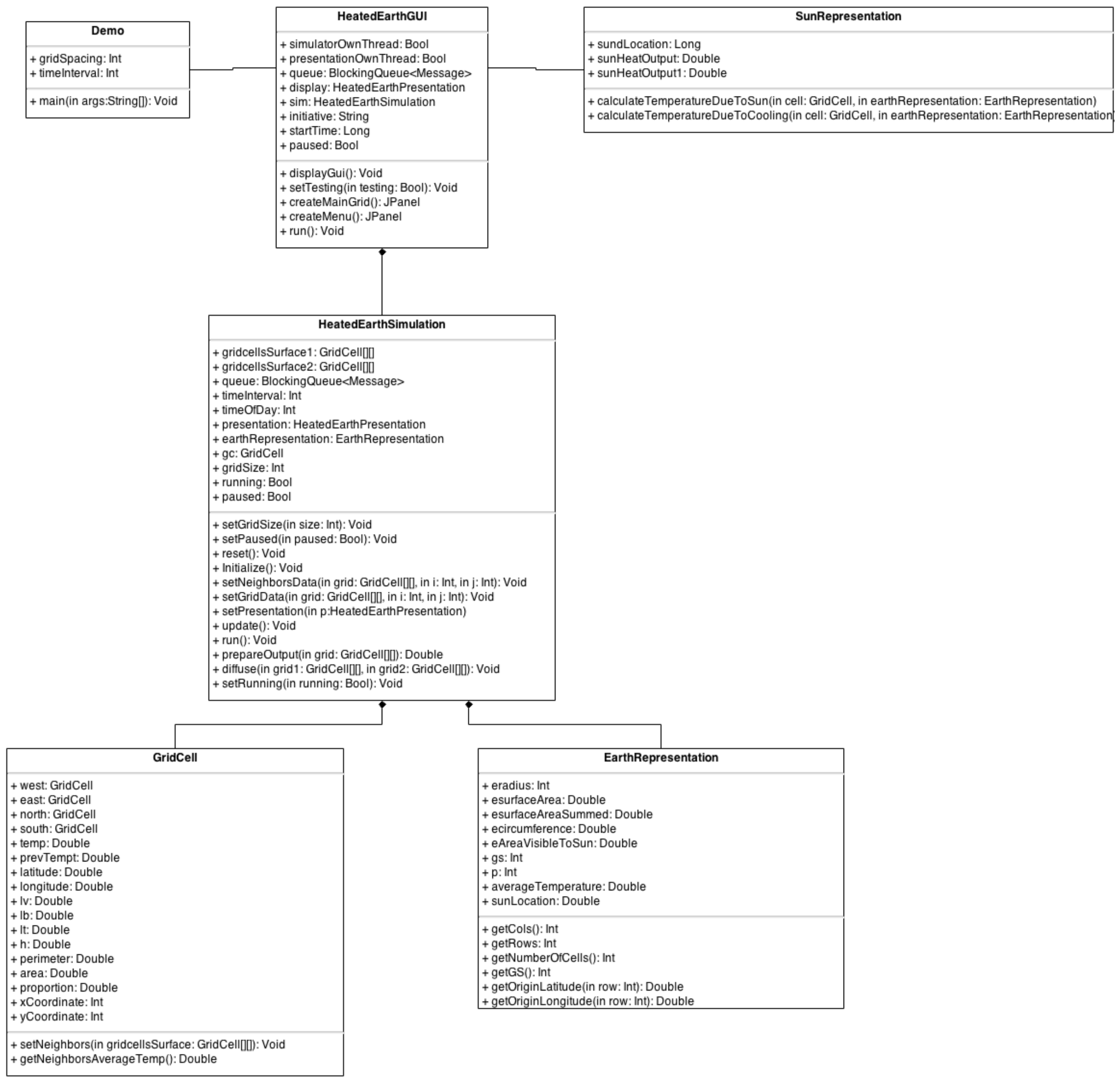
Demo
+ gridSpacing: Int
+ timeInterval: Int
+ main(in args:String[]): Void
HeatedEarthGUI
+ simulatorOwnThread: Bool
+ presentationOwnThread: Bool
+ queue: BlockingQueue<Message>
+ display: HeatedEarthPresentation
+ sim: HeatedEarthSimulation
+ initiative: String
+ startTime: Long
+ paused: Bool
+ displayGui(): Void
+ setTesting(in testing: Bool): Void
+ createMainGrid(): JPanel
+ createMenu(): JPanel
+ run(): Void
SunRepresentation
+ sundLocation: Long
+ sunHeatOutput: Double
+ sunHeatOutput1: Double
+ calculateTemperatureDueToSun(in cell: GridCell, in earthRepresentation: EarthRepresentation)
+ calculateTemperatureDueToCooling(in cell: GridCell, in earthRepresentation: EarthRepresentation
HeatedEarthSimulation
+ gridcellsSurface1: GridCell[][]
+ gridcellsSurface2: GridCell[][]
+ queue: BlockingQueue<Message>
+ timeInterval: Int
+ timeOfDay: Int
+ presentation: HeatedEarthPresentation
+ earthRepresentation: EarthRepresentation
+ gc: GridCell
+ gridSize: Int
+ running: Bool
+ paused: Bool
+ setGridSize(in size: Int): Void
+ setPaused(in paused: Bool): Void
+ reset(): Void
+ Initialize(): Void
+ setNeighborsData(in grid: GridCell[][], in i: Int, in j: Int): Void
+ setGridData(in grid: GridCell[][], in i: Int, in j: Int): Void
+ setPresentation(in p:HeatedEarthPresentation)
+ update(): Void
+ run(): Void
+ prepareOutput(in grid: GridCell[][]): Double
+ diffuse(in grid1: GridCell[][], in grid2: GridCell[][]): Void
+ setRunning(in running: Bool): Void
GridCell
+ west: GridCell
+ east: GridCell
+ north: GridCell
+ south: GridCell
+ temp: Double
+ prevTempt: Double
+ latitude: Double
+ longitude: Double
+ lv: Double
+ lb: Double
+ lt: Double
+ h: Double
+ perimeter: Double
+ area: Double
+ proportion: Double
+ xCoordinate: Int
+ yCoordinate: Int
+ setNeighbors(in gridcellsSurface: GridCell[][]): Void
+ getNeighborsAverageTemp(): Double
EarthRepresentation
+ eradius: Int
+ esurfaceArea: Double
+ esurfaceAreaSummed: Double
+ ecircumference: Double
+ eAreaVisibleToSun: Double
+ gs: Int
+ p: Int
+ averageTemperature: Double
+ sunLocation: Double
+ getCols(): Int
+ getRows: Int
+ getNumberOfCells(): Int
+ getGS(): Int
+ getOriginLatitude(in row: Int): Double
+ getOriginLongitude(in row: Int): Double

Figure 1. Class model diagram

Static Description

The design of the program as stated early can be divided into three separate components (Simulation, Presentation and GUI). These components are loosely-coupled and share only data between each other via messages on a queue. The major classes and their descriptions are listed in the table below.

CS 6310 Project 2: Heated Earth

Design Description

Table 2. Major classes and their descriptions

Class	**Description**
Demo	Accepts user-specified input and produces program window (HeatedEarthGUI)
EarthRepresentation	This class represents the Earth and performs the required calculations need to set the attributes t the cells which form the Earth's surface.
GridCell	Represents one section of the Earth. The Earth i split into a number of grid cells based on longitude and latitude.
HeatedEarthGUI	Specifies the properties and components of the program window, including labels, text fields, buttons, and the simulation view. Creates HeatedEarth Simulation.
SunRepresentation	Represents the sun and performs the calculation to determine the heat increase a particular cell would receive due to sunlight.
HeatedEarthSimulation	Initializes and runs the simulation. It instantiate the EarthRepresentation and set the initial values of the grid cells.
HeatedEarthPresentation	This class is responsible for the presentation of the results generated. It take in data from the simulation.
Message	This class represents the data that is transferred via the queue. Each message contains an array doubles representing the temperature of the various grid cells and a long value which represents the sun's location.
StandAloneSimulation	This class enabled easy testing and running of t simulation component without the GUI.

The following table displays the dependency relationships between major classes and supporting classes.

CS 6310 Project 2: Heated Earth

Design Description

Table 3. Major Classes and their dependencies

Class	Dependency Type	Dependent Class
Demo	«instantiates»	HeatedEarthGUI
EarthRepresentation	«call» parameter	SunRepresentation GridCell
GridCell		
HeatedEarthSimulation	«instantiates» «instantiates» «instantiates» returns as result	SunRepresentation EarthRepresentation GridCell Message
SunRepresentation	parameter	EarthRepresentation
HeatedEarthGUI	«instantiates» «instantiates»	HeatedEarthSimulation HeatedEarthPresentation
HeatedEarthPresentation	uses as a parameter	Message
Message		
StandAloneSimulation	«instantiates»	HeatedEarthSimulation

CS 6310 Project 2: Heated Earth

Design Description

Interaction Diagrams

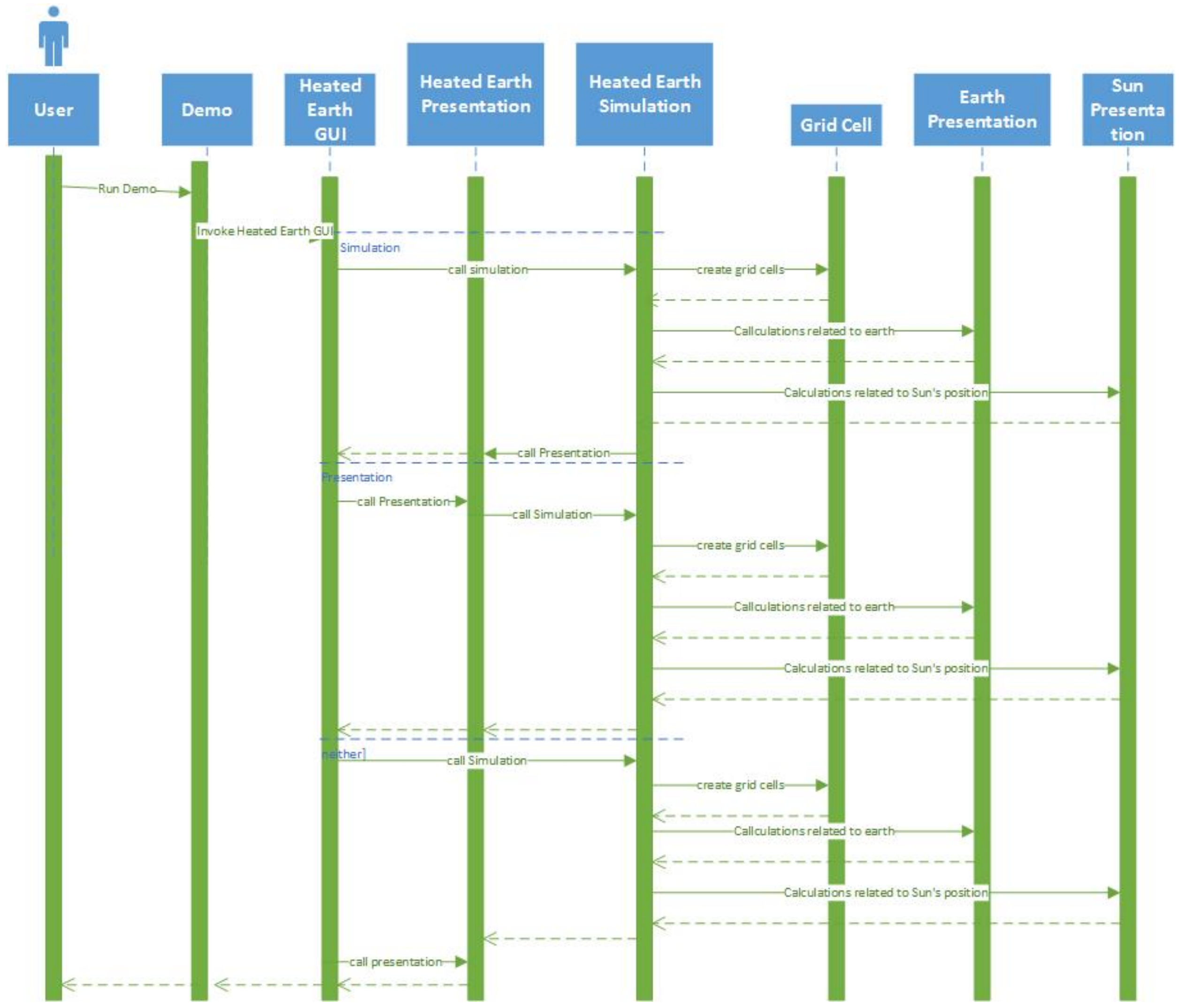

Figure 2. Sequence diagram

User invokes demo with options to indicate who is the producer, who is the consumer and also indicate if each of them should run in their own thread. If -t option is used HeatedEarthGUI will call simulation which produce the updated grid and call presentation. If -r option is used HeatedEarthGUI will call Presentation layer which in turn will call simulation to produce another updated grid. If neither options are used, HeatedEarthGUI will control simulation and presentation components by itself.

Architecture

CS 6310 Project 2: Heated Earth

Design Description

Components

The three main components of the Heated Earth Simulation are the User Interface (GUI), Presentation and the Simulation.

The GUI is the component responsible for allowing the user to control the simulation. Various options are available such as setting the grid space, time interval and display rate. In addition, the GUI also allows the user to pause/resume, stop and reset the simulation. The GUI also serves as the manager for the other two components allowing to the work together seamless and produce the desired the results.

The Simulation component is primarily the calculation engine. This component is responsible for calculation the results of the sun's effect of the Earth for the given parameters. The simulation takes in the grid spacing and simulation time step (the time between calculations), initializes the required objects based on the parameters specified and begins the calculations. After performing the calculations the simulation places the result on a queue which the Presentation component obtains and uses to display the updated heat allocation on the Earth. The simulation also manages the Earth's rotation around the sun. The message placed on the queue by the simulation contains the cells temperature and the sun's position relative to the Earth at that point in time.

The Presentation is responsible for displaying the temperature values obtained from the simulation into meaningful graphical display. The Presentation ensures that the cells have the appropriate color transitions based on their temperature. The temperature peaks and troughs are depicted by red and blue respectively. The Presentation component also ensured that the sun is displayed at the correct position relative to the Earth and the latitude and longitude lines were accurately drawn according to the grid spacing specified.

CS 6310 Project 2: Heated Earth

Design Description

Connectors

The primary connector used between the components was a queue. This queue allowed the Simulation and Presentation Components to communicate via messages on the queue. GUI communicated with the Presentation and Simulation via instantiation of the primary classes of these components. Given, that the GUI was the manager of the entire program it directed the behavior through Java calls based on the user input. HeatedEarthGUI component sends data of type Message which contains the location of sun and two dimensional array of doubles. The array of double represents the temperatures of the various cells comprising of the Earth's surface. It also communicates grid size, display rate and a boolean value indicating whether the simulation is in paused state or not.

HeatedEarthGUI calls HeatedEarthSimulation with Message object, grid size and time interval (1 to 1440).

When Simulation has the initiative it calls Presentation provides an array of doubles and the sun's location. Similarly when Presentation has the initiative it requests from the Simulation current location of the sun and a two dimensional array of doubles.

Configuration

The diagram below illustrates the communication between the major components of the Heated Earth Simulation. The Demo class starts the GUI which then gives the user the option to start the simulation with various parameters. The Simulation and Presentation components communicate via a message queue. The simulation places messages on the queue while the presentation obtains and parses these messages.

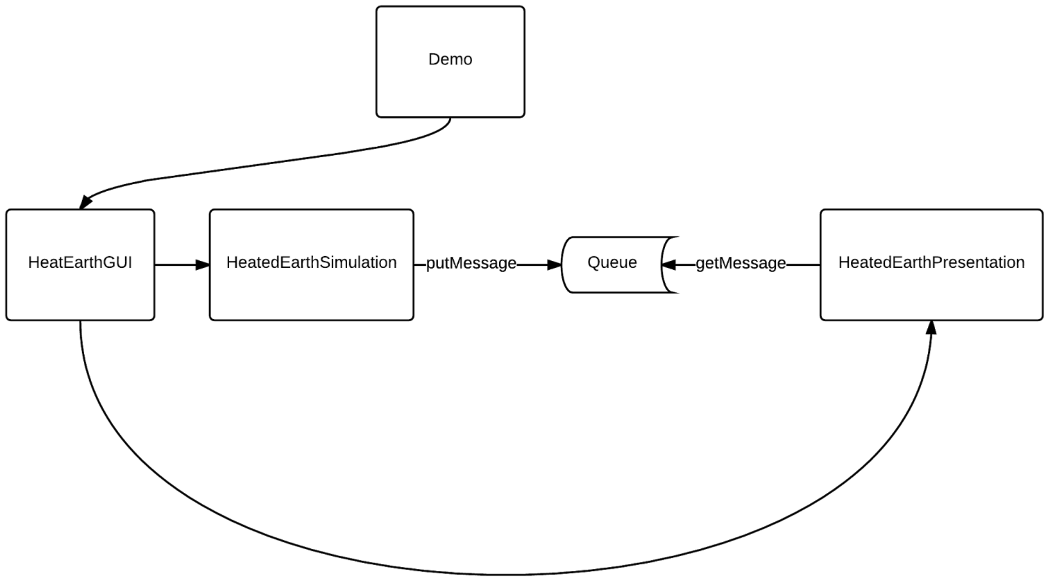

Figure 3. Communication diagram

Style

Our approach most resembles Object Oriented Architecture. This style defines responsibilities into independent reusable objects. Object Oriented Architecture is based on abstraction, composition, inheritance, encapsulation, polymorphism and decoupling. This architecture enabled the problem to be solved in a clear and concise manner and provided a solid foundation for code reuse. The various components required were modeled as objects and their responsibilities represented as methods on these objects.

CS 6310 Project 2: Heated Earth

Design Description

Our architecture in part implements the Pipes and Filters methodology because the main components (Simulation, Presentation and GUI) are designed to accept inputs from and produce outputs for each other independently from their identities. This is advantageous in that it allows for easier modification or replacement of individual components based on specifications rather than pre-existing sub-architectures.

At the highest level, we also utilized the commonly-recognized MVC architecture which allowed us the flexibility of writing very loosely-coupled code in a structured, repeatable way.

Design Considerations

During the design phase we decided to use an object to represent the cells on the grid primarily for the flexibility it provided. From the results in project one using an object was the least efficient implementation in terms of performance however we mitigated this by using arrays to store the objects and performing initial calculations only once during initialization. Such initial calculations include longitude, latitude, area and the cell's neighbors. In addition, the object array was converted to an array of doubles before being placed on the queue thus reducing the size of the data structure being passed to the GUI.

In implementing diffusion we decided to use the same algorithm used in project 1. This two plate approach ensured consistent results and allowed easy maintainability. In the approach used we ensured that the cells for the plates were only created at initialization and reused for each diffusion iteration. This approach minimized garbage collection and improved performance.

The program was divided into three separate components to provide the flexibility to ran each component on a separate thread. This also allowed the team to work independently on each module which enabled the final product to be delivered faster than having the modules tightly coupled.

Other design considerations are listed below:

Table 4. Design Considerations

Design Problem	**Solution**
Calculating the sun's effect	Created a Sun object which took in a grid cell and calculated the impact of the sun based on th cell's properties.
Illustrating the temperature movements with meaningful color transitions.	Ran the simulation with varying properties and came up with a consensus of "High" and "Low' temperatures. Using them we were able to develop meaningful color wheel.
Consistently placing the sun at the equator	Dynamically calculated the sun's position so as the user enters varying dimensions the sun's position remained at the equator.
Calculating the latitude and longitude positions	Ensure that we developed an algorithm that distributed cells on the grids in an even consiste manner.

Non-functional requirements

Table 5. Non-functional requirements

Non-functional Requirement	**Solution**
Stability	Stress tested the application to ensure that it wa able to handle extremes.
Independent Modules	Clearly defined object oriented design which encompassed the requirements for each module
Maintainability	Object oriented design using valid naming conventions and informative comments.
Performance	Utilized arrays to represent the grid, avoided duplicated calculations.
Portability	Avoided external libraries and utilised only Jav 1.6 libraries.

Testability	During the design testing was considered so modules could be tested independently. In addition test classes were designed to make it easier to identify bugs.

Graphical User Interface (GUI) Design

The graphical user interface (GUI) was designed to allow input of the configurable simulation variables and easy control of the running simulation. The GUI is a window which is mostly filled with a map of Earth's surface. On the left side of the map are text fields for the configurable properties: grid dimension, simulation time step, and simulation display rate. The run simulation button is located below the text fields. When the run button is pressed, the simulation begins. A grid appears on top of the map with the specified size and each grid cell is colored according to its simulation temperature. Simulation control buttons are located directly under the map with buttons to stop, resume, and restart the simulation.

The elements of the GUI were chosen because of their pertinence to the design objective, which is the simulation of Earth's temperature. Certain configurable options must be exposed to the user.

CS 6310 Project 2: Heated Earth

Design Description

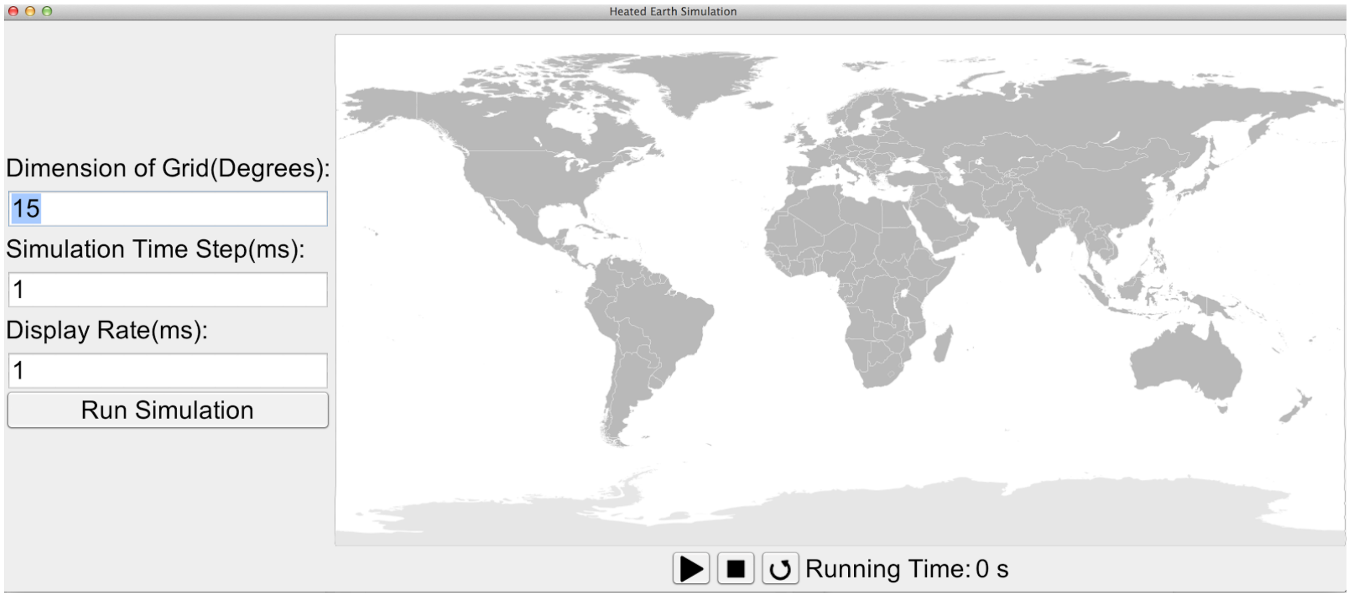

Figure 3. Graphical User Interface

CS 6310 Project 2: Heated Earth

Design Description

State Chart Diagram

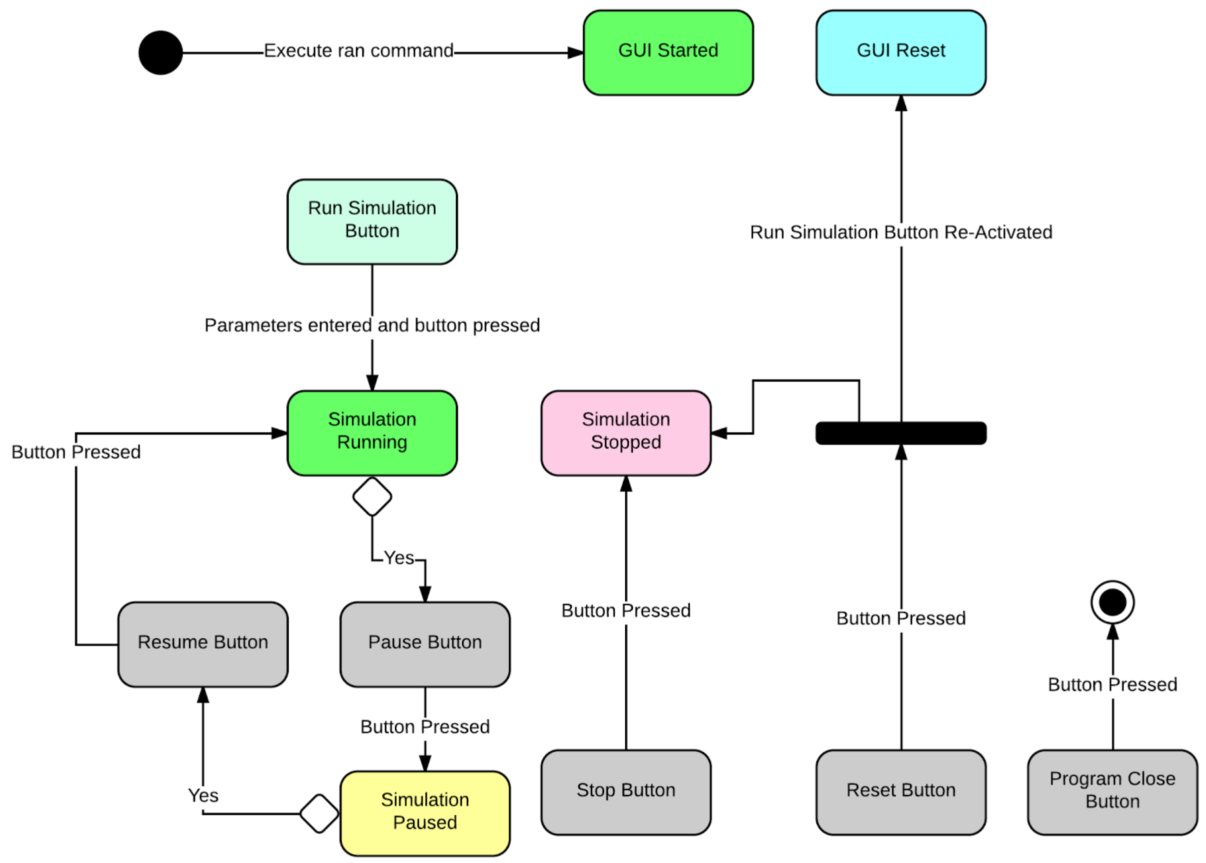

Figure 4. Statechart diagram

Code Reuse

For the most part, the code for the temperature simulation program was original and not re-used from the Heated Plate project. The object oriented portion of Project 1 was the code that was reused for this project. Diffusion between objects and their neighbors remained the same. Given the varied approaches used by the different teams and the significantly jump in complexity between Project 1 and 2, code reuse was not too useful. It was faster to create and implement a new design while utilizing the lessons learnt from the first project and avoiding repeating those mistakes. For example, given that array implementation of the grids had the best performance in the first project we decided to keep this implementation for project 2.

Reflection

The design of the project can definitely be improved from its current implementation, but given the time constraints, we decided to focus primarily on functionality, stability, maintainability. Since this was a relatively small program we decided not to make performance a major priority and focus on getting a reliable well-designed product built and then optimize. In most cases focusing on a good design will yield solid performance and this case was no different.

This project was definitely far more difficult than the first project. Understanding the problem was challenging and the frequent updates made it difficult to finalize an approach. Looking back we definitely should have made an earlier decision to stick with our current assumptions and design based off them. This would have given us a bit more time to work on the design and coding. Another challenge the team encountered with this project was the changing headcount within the team. It's more difficult to work effectively without consistency with resources. However, this was seen as a real world challenge and the remaining team members were able to re-allocate duties to complete the requirements.

While we did encounter many problems we also made a number of good decisions while completing this project. The decision to build the components individually was a great decision and this really enabled the team to get a prototype up and running quickly. By focusing on getting the initial prototype and going through iterations of testing and enhancing we were able to fine tune the application by identifying and solving defects as they were encountered.

CS 6310 Project 2: Heated Earth

Design Description

Though we faced a number of challenges we feel the final product certainly meets the requirements that were given.