PROJECT 2: HEATED EARTH

Design Description Document

Team 3
Faour, David (dfaour3)
Frank, Robin (rfrank8)
Poskevich, Robert (rposkevich3)
Sachdev, Gaurav (gsachdev3)
Slater, Joey (jslater3)

October 26th 2014
Georgia Institute of Technology
Atlanta, GA

Overview

The heated earth project is based on the scientific concept of diffusion and simulates the temperature of the earth as it is heated by sun. The design description document studies software architecture in the context of an application used to simulate the temperature at locations on the surface of the earth as it rotates about its axis. It is depicted with false-color animation representing the temperatures displayed on the earth, all within a graphical user control interface, which allows the user to control the simulation and presentation.

Program Characteristics

	Earth Simulation	Console Simulation
Total Lines of Code	845	420
Total Class Size (in bytes)	47 KB	16 KB
Number of Classes	15	5
Average number of non-static methods per class	5.07	5.60
Average number of static methods per class	0.07	0.80
Average number of total methods per class	5.13	6.40
Average number of non-static attributes per class	4.00	8.00
Average number of static attributes per class	1.2	0.80
Average number of total attributes per class	5.2	8.80
Number of inter-class dependencies	36	4

Source: This information was gathered using the Metrics plugin for Eclipse.

UML Class Model Diagram

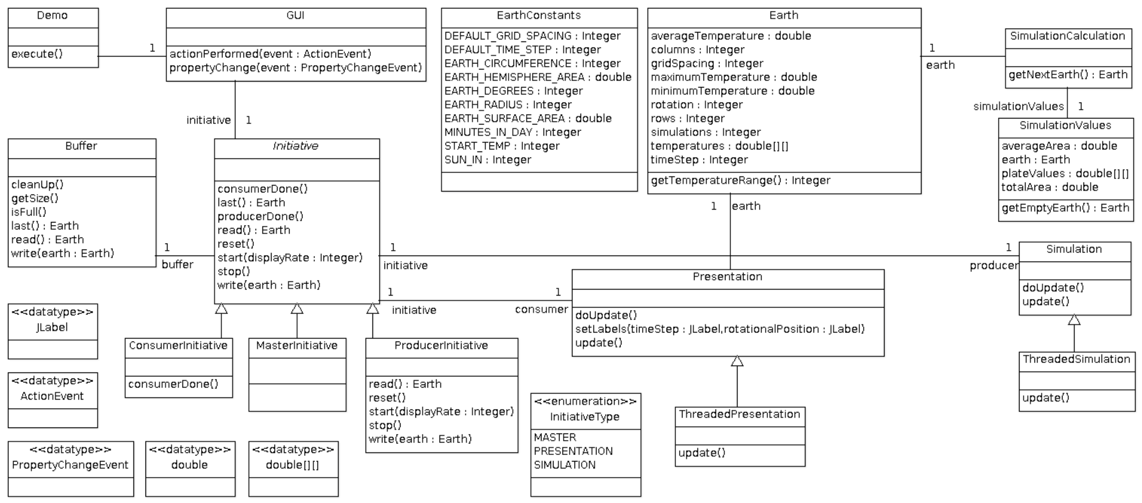


Figure 1.0: Heated Earth Class Diagram

Static Description

The major classes used in this program are listed in the the tables below. The tables contain the package name on the first row and the class name on the second row. The subsequent rows list the dependencies of each class. The Demo class in the EarthSim package is the point of entry to the application, while the MainScene and Presentation classes utilize Swing to provide the user interface. The Earth class is shared by a large number of components and models a snapshot of the planet at any given instance. It contains a matrix of double values representing the temperature across the Earth's surface as well as several other properties, such as the current simulation number, average temperature, etc.

To perform the heated earth simulation, the Simulation class receives external requests via its "update" routine and orchestrates the next calculated iteration by first initializing a SimulationValues object with the current Earth object. The SimulationValues class encapsulates many of the properties and the logic to initialize said properties required by the SimulationCalculation class. Afterwards, a SimulationCalculation object is created and its “getNextEarth” routine is invoked, which steps through the requisite calculations to model the Earth after one added iteration. The new Earth instance is returned to the Simulation class and written to the buffer. The Buffer class is shared between both Presentation and Simulation classes and stores an array of Earth objects corresponding in size to the provided buffer length argument. It utilizes the Object class “wait” and “notify” routines to, respectively, idle any thread attempting to write to a full buffer and to awaken said thread upon buffer slot availability.

The Initiative class is abstract and provides methods via which to control the producer-consumer cycle, e.g. start, stop and reset. The start routine initializes a Swing Timer object to be invoked at an interval equal to the display rate parameter, which requests an update to the Presentation class with every cycle. Upon completion of work for any given cycle both Presentation and Simulation objects invoke the Initiative “consumerDone” and “producerDone” routine, respectively, which are empty in the abstract Initiative. Therefore, it is through the implementation of said methods in the ConsumerInitiative and ProducerInitiative classes via which the current initiative is defined. For instance, in the ConsumerInitiative class, the “consumerDone” routine is implemented and simply invokes the Producer “update” routine.

However, this is not the case for the ProducerInitiative class, as it does not invoke the Consumer “update” routine. This is a result of the previously mentioned Timer object, which is attuned to the display rate and is responsible for invoking said routine. The ProducerInitiative class instead ensures the Simulation “update” routine is invoked continuously and sequentially and uses a Swing Timer object to ensure said invocation is not prohibitive to handling other asynchronous application events. This non-blocking requirement was also utilized therein in stashing a request to write an Earth object to the buffer that would otherwise result in process idling, particularly when the buffer is full. With the ProducerInitiative thus defined, the MasterInitiative was deemed redundant and was thereby not implemented.

Threading is provided to both Simulation and Presentation components through inheritance. Both ThreadedSimulation and ThreadedPresentation classes extend the Simulation and Presentation classes, respectively, and override its "update" routine. Upon initialization, these classes create a new Thread object, which invokes an internal "doUpdate" routine and then idles. Therefore, the functionality provided by the overridden "update" routine is simply to wake the idling thread.

EarthSim		
Buffer	**Demo**	**MainScene**
creates: Earth	creates: InitiativeType	calls: Initiative
instantiates: Earth	instantiates: ConsumerInitiative	creates: Initiative
returns as result: Earth	instantiates: InitiativeType	instantiates: Initiative
uses as a parameter: Earth	instantiates: Presentation	returns as result: Earth
	instantiates: ProducerInitiative	uses as a parameter: Initiative
Presentation	instantiates: Simulation	
creates: Earth	instantiates: ThreadedPresentation	**ThreadedPresentation**
instantiates: Earth	instantiates: ThreadedSimulations	uses as a parameter: Initiative
creates: Initiative	returns as result: Initiative	
instantiates: Initiative	returns as result: Presentation	
uses as a parameter: Initiative	returns as result: Simulation	
calls: Earth		

EarthSim.Initiative		
ConsumerInitiative	**Initiative**	**ProducerInitiative**
calls: producer	calls: Buffer	calls: Buffer
uses as a parameter: Buffer	calls: Presentation	calls: Simulation
	creates: Buffer	creates: Earth
	creates: Presentation	instantiates: Earth
	creates: Simulation	returns as result: Earth
	instantiates: Buffer	uses as a parameter: Buffer
	returns as result: Earth	uses as a parameter: Earth
	uses as a parameter: Buffer	
	uses as a parameter: Earth	

EarthSim.Simulation		
Simulation	**SimulationCalculation**	**SimulationValues**
calls: Earth	calls: Earth	calls: Earth
calls: Initiative	calls: SimulationValues	creates: Earth
calls: SimulationCalculation	creates: Earth	instantiates: Earth
creates: Initiative	creates: SimulationValues	uses as a parameter: Earth
instantiates: Earth	instantiates: SimulationValues	
instantiates: Initiative	returns as a result: Earth	**ThreadedSimulation**
instantiates: SimulationCalculation	uses as a parameter: SimulationValues	uses as a parameter: Initiative
uses as parameter: Initiative		

Sequence Diagram

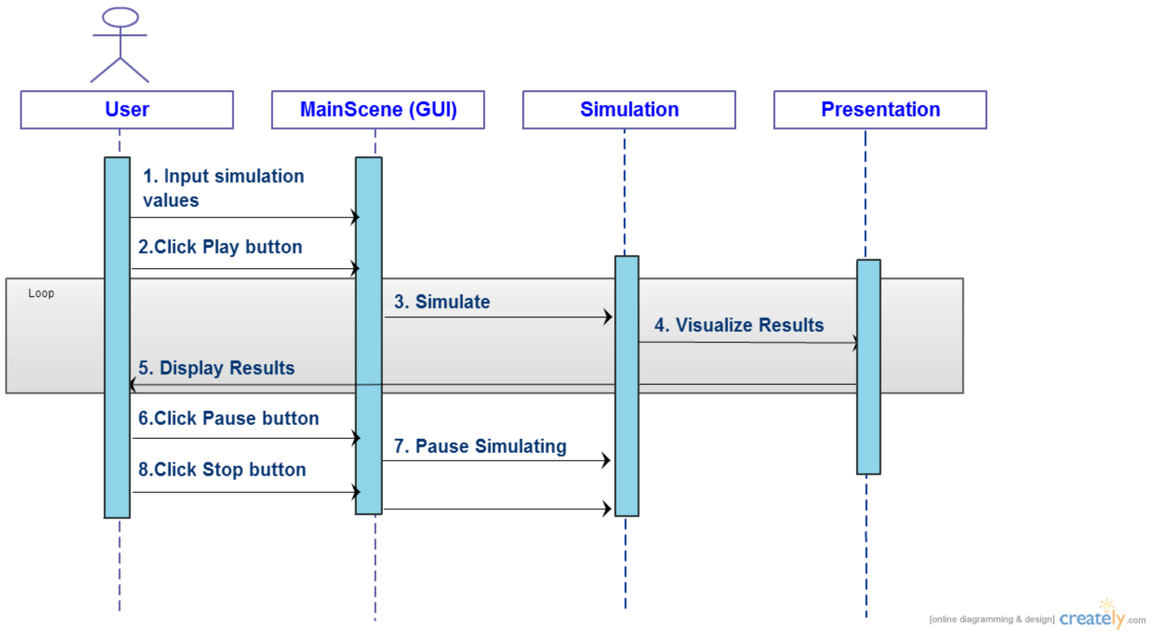


Figure 2.0: Execution of the GUI

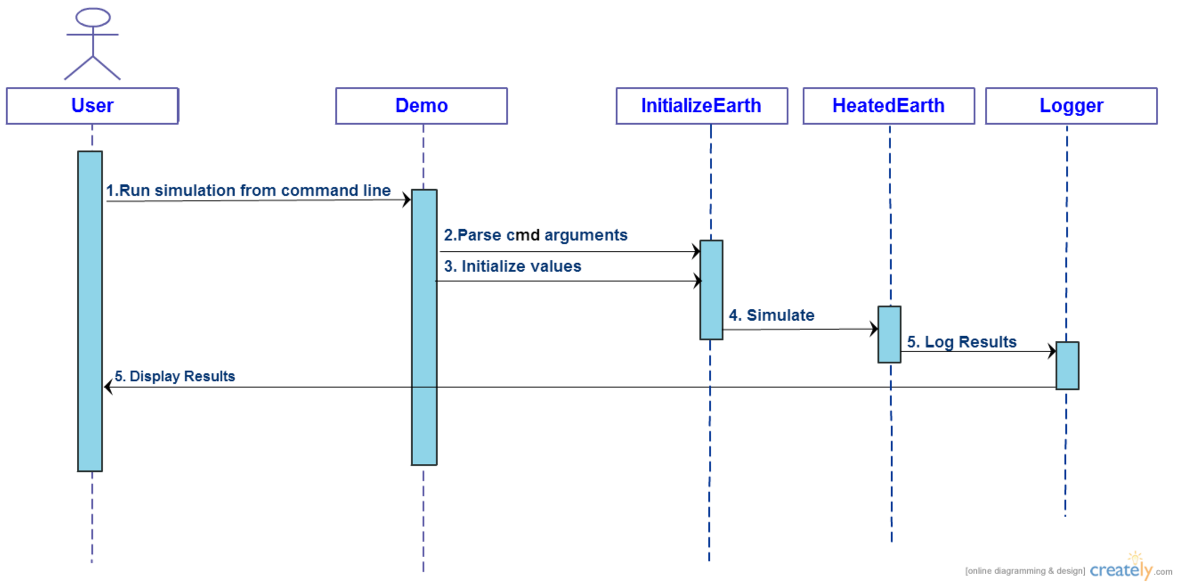


Figure 2.1: Execution of simulation and presentation from the command line

Architecture

Components

- Demo: Creates the Initiative, Simulation, Presentation, and MainScene (the latter three by passing an Initiative as a parameter). This class provides entry into the application and contains the main method.
- Buffer: Shared memory between Presentation and Simulation components.Used within the Initiative classes to store the Earth data model.
- Earth: Data model that contains the information regarding the planet during the simulation including: temperature values, number of rows, and number of columns, the grid spacing, current rotation and more. Used within the Simulation to store values and by Presentation to display the information.
- MainScene: Component housing the user inputs, information labels, and controls for the simulation. The class displays the Presentation and Earth's simulation

values. Uses the Initiative to pass information between the Presentations to the user interface.

- Presentation: Displays the Simulation in the user interface using Java Swing components. Uses the Initiative to get an Earth model to paint with graphics.
- ThreadedPresentation: Displays the Simulation in a separate thread using Java Swing components. Uses the Initiative to get an Earth model to paint with graphics.
- Simulation: Contains the actions for updating the simulation from the Initiative. Passes the Earth model from the Initiative into a SimulationValue that is then passed into SimulationCalculation to get the update iteration. This is then passed back into the Initiative.
- ThreadedSimulation: Contains the actions for updating the simulation from the Initiative in it's own thread. Passes the Earth model from the Initiative into a SimulationValue that is then passed into SimulationCalculation to get the update iteration. This is then passed back into the Initiative.
- SimulationCalculations: Contains the calculations for the heat transfer between Earth's double arrays of temperatures. Returns an Earth model containing the next rotated position values.
- SimulationValues: Component for calculating and holding the simulation values. Takes in an Earth model to create simulation values for aspects of the average temperature.

Connectors

- Initiative, ConsumerInitiative, ProducerInitiative: Mitigates interactions and sends the Earth model between the Simulation and the Presentation via read(), last(), reset(), start(). stop(), and more. It also sends the Earth model as a message between the Simulations to the Buffer. This is an event connectors as there is one callee and caller through call by name and are synchronous that pass information.

Configuration

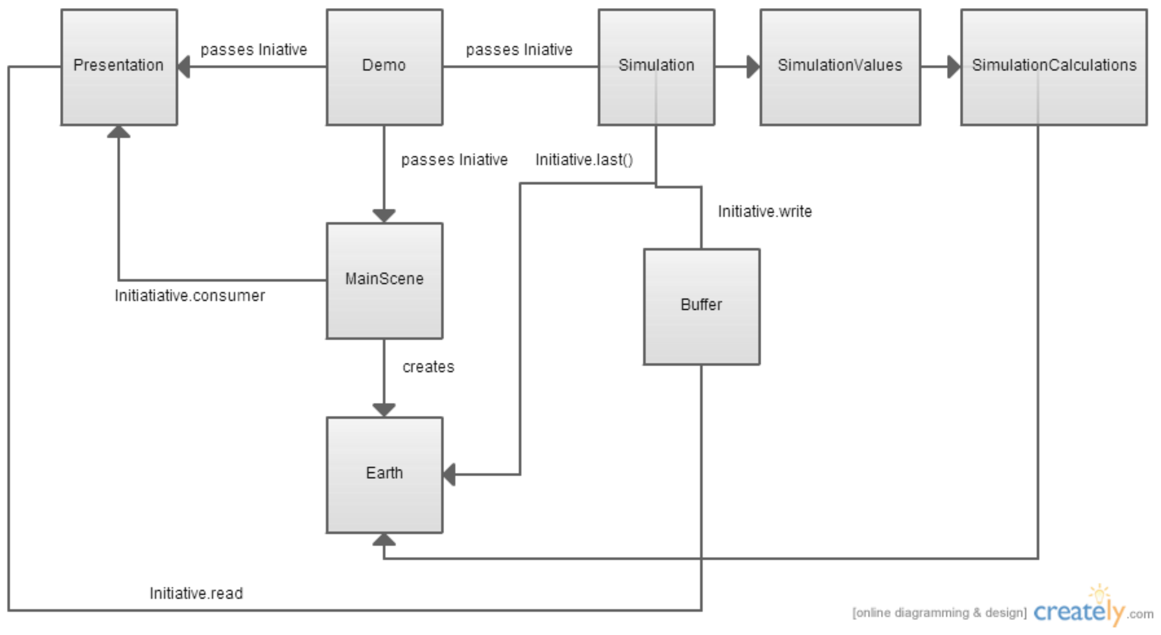


Figure 3.0: Heated Earth Configuration (Data Flow) Diagram

Architectural Style

There were multiple architectural styles implemented within the application design. For the Swing components in MainScene, the Event-based, Implicit Invocation architecture pattern is utilized extensively as the interface listens for action items, such as buttons presses, or property change events, as is the case for the grid spacing input. Furthermore, timers utilized in the Initiative and ProducerInitiative classes are indicative of this pattern. Once the timers are triggered, the respective producer or consumer then invokes its “update” method.

Object-oriented organization is apparent in the separation of responsibilities into different classes. This is most evident in the Initiative classes, which abstract and provide encapsulation to the producer and consumer invocation logic.

The Blackboard pattern was also exemplified in the shared buffer between the consumer and producer and the incremental approach taken to calculate Earth simulation values. For example, the buffer is first populated with an Earth model, which the producer then reads and uses to calculate the next Earth simulation iteration. The new Earth instance is then inserted back into the buffer, which the consumer reads and uses to update the display.

Low Level Design Considerations

Deliberation was given as to whether the plates of the Earth should be modeled in a Plate class or be stored as a property of the Earth class. While it was argued that the Plate class would provide encapsulation to some calculation methods and allow for shared usability by the Presentation and Simulation classes, it was deemed significantly less performant than would be the use of an internal instance variable and was thereby discarded. Additional performance considerations influenced the use of Swing Timer objects throughout the application, so as to avoid blocking on any single thread and potentially locking the GUI. Finally, the Java modifier final was used extensively and where appropriate, which can potentially lead to a gain in performance with the increased use of memorization.

Other design decisions relied heavily on the notion that runtime configuration options should not pervade the application. One example of this is the design choice to allow threading by extending the base Simulation and Presentation classes. Another example lies with the Initiative classes and the decision to invoke the member “producerDone” and “consumerDone” routines regardless of the bearer of initiative - allowing these methods to be implemented or not to meet the different criteria. Across both examples, only the Demo class need be aware of the runtime options, which it uses to initialize the appropriate classes and to then further launch the application.

Non-functional Requirement

	Evolvability
How requirement was dealt with	Design the program to be easily expanded by adding additional features.
Design decisions made	Separating out the calculations into a SimulationCalculation class. If needed, could easily be converted into an interface for different concrete calculation implementations.
Justification	Rather than placing calculations into the Simulation, the calculations were extracted into a separate class to adhere to the single responsibility principle. The Simulation now solely ‘updates’ while the SimulationCalculation class handles the actual calculations. Long term, this would allow future calculation implementations to be created without disrupting the Simulation class.

	Usability
How requirement was dealt with	1. Created text boxes for the user to input the required values. 2. Added a "Play/Resume", “Pause”, “Stop” button to operate the program. 3. Created a panel for visualization of the data - used to make it easier for the user to see the heat diffusion. 4. Grid is displayed as colored cells: A. Temperature at 0 - color is black. B. Temperature at 100 - color is red.
Design decisions made	1. Allow the user to enter any value in the text boxes A. Validate Grid Spacing to be an integer that’s evenly divisible into 180 B. Validate Time Step to be an integer between 1 and 1440 C. Validate Display Rate as an integer between 100 and the max integer value 2. Default valid values to quickly run application. 3. Color the cells based on the resulting temperature to display variable shades of red. 4. Disable certain buttons when simulation is running, paused, or stopped. 5. Disable input fields when simulation is running or paused.
Justification	There is no standard numerical input box in Java Swing, so the text input boxes were used that would check to determine if the input met

	restrictions. Default values were chosen to quickly run the application without having to place the values into the form each time. The “Play/Resume” button is disabled while the simulation is running as you can’t re-play the simulation if running. The “Pause” button is disabled while paused or stopped as there is no reason to pause if the simulation isn’t running. The “Stopped” button is disabled while stopped as there is no reason to stop the simulation if stopped. The input fields are disabled during the simulation as the user should not be able to change the properties mid simulation..

	Correctness
How requirement was dealt with	Additional calculations were made while developing the simulation to ensure that the total amount of heat gained equaled the amount of heat loss for any given iteration. Averages were also taken across rows and across columns to verify stabilization at which point values were examined and validated.
Design decisions made	The simulation calculation needed to be subdivided into several steps such that it would be clear where to add and to assert upon additional calculations.
Justification	Because there was no output provided to validate against, it was necessary to perform our own validation over the calculated temperature values.

State-chart Diagram

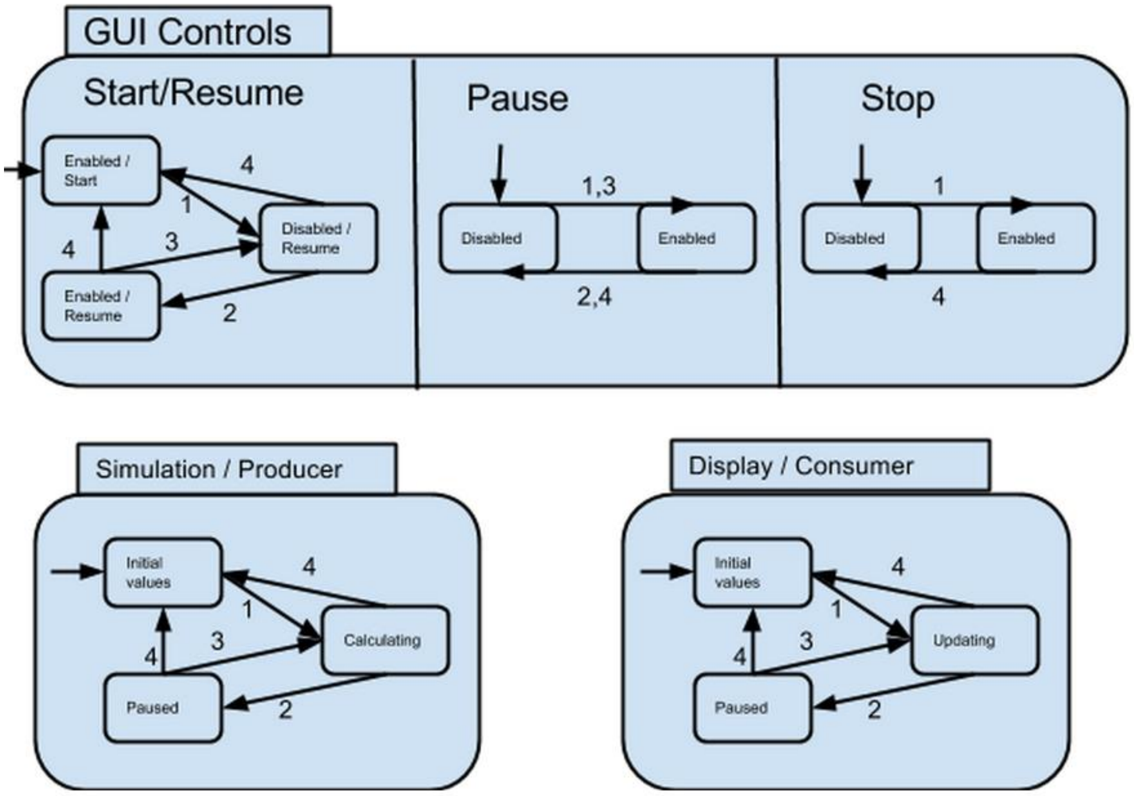


Figure 4.0: Heated Earth State-chart Diagram

Events:

1. User Presses Start Button
2. User Presses Pause Button
3. User Presses Resume Button
4. User Presses Stop Button

GUI Design

The most critical aspect of modeling a heated planet is attempting to visualize three-dimensions in a two-dimensional space, especially when the default Java Swing packages do not contain 3D modelling components. For presentation visualization, the team looked into displaying the information in a flattened cylinder model (like most maps) or flattening the sphere into two circles (one for each half sphere). The flattened cylindrical model made the most sense to view the temperature values of the entire planet at once in reference to the other parts of the planet, using colors to indicate the value.

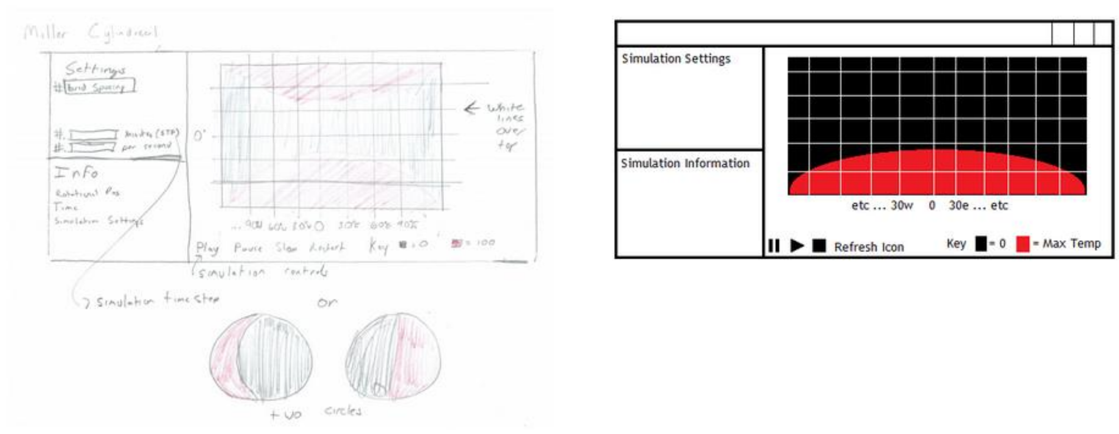
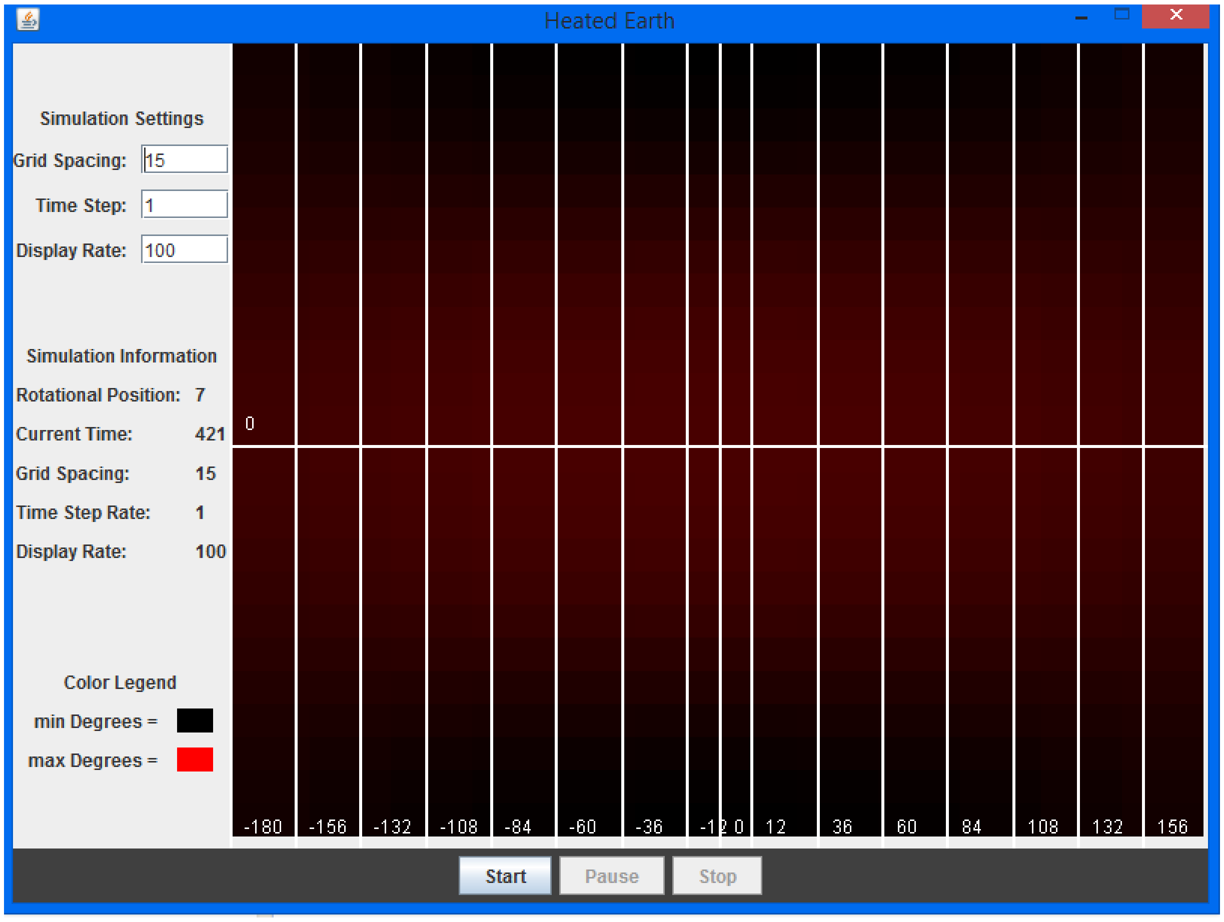


Figure 5.0: Heated Earth Initial Design Presentation

As shown in the sketches above, the simulation would contain the information and settings on the left hand side while the actual simulation would be displayed on the right with controls underneath. For displaying numerical information, such as current rotational position, the interface groups the values into the bottom right corner for quick accessibility without taking away from the simulation itself. For user inputs, the interface places the components in the top left corner as most applications read left to right. The controls are interactive, only enabling when a user is able to interact with them. The final product looks similar to the sketches with a few interface tweaks to take advantage of the Java Swing styling components.

Validation on user inputs is also performed by the GUI interface and is facilitated by the Swing component JFormattedTextField, which allows for the restriction of user inputs to integer values, as appropriate, with minimum and maximum values. In the case of the grid spacing input, further validation is performed to ensure that the input integer is evenly divisible into 180. This was accomplished by registering the MainClass as a listener to the text field's property change event, which then decrements the input value until it is evenly divisible.



Grid Spacing:	15
Time Step:	1
Display Rate:	100

Rotational Position:	7
Current Time:	421
Grid Spacing:	15
Time Step Rate:	1
Display Rate:	100

Figure 5.0: Heated Earth Final Design Presentation

Code Reuse

As the nature of the calculation changed, the structure of the code was reused more than specific implementations. For example, we maintained the concept of a constants class to hold invariant initial values. Within the EarthConstants class, the structure was reused, but most of the specific initialization constants were new and specific to heating Earth instead of a single plate.

Further reuse was had in modeling temperature values as a 2 dimensional array of doubles. However, the calculations to convert degree precisions into trapezoids modeled for the earth and to determine attenuation for various degrees of latitude, among others, are specific to this implementation. Within the simulation class, we were largely able to reuse the “heatNeighbors” method, which only required changes to accept the weight of each neighbor as opposed to an average. The swap concept to update to new temperature values was carried over, and specific implementation improved.

Some components from the Project 1 - Heated Plate, Team 8’s interface were reused and modified, including the layout and visualization color selection. The interface was built on Swing’s BorderLayout by adding more components to the layout to the information/controls on the left and the presentation to the right. The color creation method takes into consideration the Earth’s minimum temperature as well as negative temperatures.

Reflections

Upon completion of the design study and the intensive scrutiny that the design faced in the study, there were several takeaways on the overall quality of the design decisions. When creating the design key principles, e.g. performance, correctness, extensibility, and ease of use, were considered. However, while studying the design areas where extendibility could be improved were identified.

The design was overly tied to a 2 dimensional rectangular space. While we used interfaces to solve design challenges with the different types of heated earth calculations, they should have also been used to pass data to and from those calculations. Moreover, the implementation of the design was challenging, particularly to visualize three dimensional space in 2 dimensions. Though the design had several shortcomings for extendibility it worked well for the purposes for which it was originally intended.

***