Queueing Model Animation and Simulation System

A Thesis submitted to the faculty of
San Francisco State University
In partial fulfillment of
the requirements for
the Degree

Master of Science

In

Computer Science

by

Richard John Robinson
San Francisco, California

May 2021

Certification of Approval

I certify that I have read Queueing Model Animation and Simulation System by Richard John Robinson, and that in my opinion this work meets the criteria for approving a thesis submitted in partial fulfillment of the requirement for the degree Master of Science in Computer Science at San Francisco State University.

Jozo Dujmovic, Ph.D.
Professor, Advisor
Thesis Committee Chair

Arno Puder, Ph.D.
Department Chair

Queueing Model Animation and Simulation System

Richard John Robinson
San Francisco, California
2021

This work pertains to the development of a simulation and visualization program for queueing systems. The intention is the creation of a program for use in the instruction of queueing theory and the evaluation of accurate results. The program contains five queueing systems with user definable parameters. Each system has numerical results calculated using event-driven simulation, which are displayed to the user. Alongside the simulator there exists a visualizer. This visualizer renders the state and state transitions of queueing systems as an animation to provide insight on the internal workings of the given system. The speed of the visualizer may be increased, decreased, or paused and a single step mode exists to process one visual event at a time. The visualization can be forgone completely through a maximum speed mode to generate quick results. QAS is written as a web application to be utilized online through a browser.

Acknowledgements

I would like to thank Dr. Dujmovic for his support, time, and infinite patience.

I would also like to thank my friends for their positivity and encouragement, they know who they are.

TABLE OF CONTENTS

Table of Figures

1. Introduction

1.1 Statement of the Problem

Whenever a demand is made from a service there is an implication of a queueing system. At the most basic level a queueing system consists of three elements—a service facility, a demand for service, and a queueing mechanism[MEYE01]. These systems can be seen anywhere a service is requested and delivered. From an educational standpoint, queueing theory can be daunting and difficult to grasp and tools for facilitating this learning are invaluable. Texts and analytical models do indeed provide valuable knowledge yet students still need guidance and examples [CIBO92]. In order to achieve this goal, a visualization component is created. The flow within a system is presented in juxtaposition of the data, openly displaying the inner-workings and assisting in both the demonstration and instruction of queueing theory. Mathematical representations may be built for any queueing system, however not all queueing systems currently have proven analytic models. In this circumstance a well-crafted simulator can provide accurate results [BRAT87][MITRA82]. These results may then be utilized by researchers for the creation and testing of novel mathematical models. When the visualization is paired with the simulation a complete picture may be formed: the progression of the entire system is revealed. QAS aims to provide an useful tool for the visualization and simulation of queueing systems.

1.2 Related Work and Background

Educational tools are created with the goal of enhancing the learning experience and providing some utility that was not present before. QAS is an educational tool with the purpose of creating visualizations of queueing systems alongside accurate simulation written in HTML, CSS, and JavaScript.

A fundamental concept that must be understood before delving deeply into the simulation of queueing models is the evaluation of randomness as for accurate simulation to exist, the generated values must be representative of the chosen distributions. RAND is an educational tool used for developing an understanding of random number generation and the evaluation of the quality of randomness [LIU03]. The random number generator in JavaScript is xorshift+128, which has a maximum period of 2^{128}-1 and utilizes exclusive or operations alongside bit shifting [MARS03]. The only known tests that these generators have failed are binary ranking tests and linear complexity tests made specifically to detect linear generators; they pass all tests within Diehard and can be calculated quickly[VIGN14][BV18]. It is possible to achieve an even higher degree of randomness through the addition of a shuffler to this generator[DUJM08].

QAS is not the first education tool created for the queueing theory at San Francisco State University. For understanding core queueing system concepts there exists a self-study tool, QNS [SANK05]. Sankar provides a valuable tool for the introduction, progression, and assessment of knowledge regarding queueing theory. QNAS is a simulation and visualization tool for queueing networks, created as a Java applet with the purpose of developing an intuitive understanding of queueing phenomenon [PAN98]. Java applets were designed to be embedded within HTML documents and executed within web browsers. Java bytecode is linked to the HTML document and sent to an interpreter within the browser [LF15]. Java applets should not be mistaken as Java applications: they are two separate entities. Java applets are no longer supported and cannot run within modern browsers[ORAC20]. “When applets failed, JavaScript become the language of the Web by default”[DOUG08]. QAS expands upon what was available within QNAS through the addition of the parallel servers model and availability of multiple CPU cores within the central server model while providing a transition to more widespread internet languages-- JavaScript, HTML, and CSS. These languages have been chosen to be used without dependencies on external libraries as they are the current standard for Web development. HTML is the core of web content and defines its meaning and structure [MDN HTML 21]. Cascading Style Sheets (CSS) is a stylesheet language used to describe formatting, styles, design templates, and presentation of HTML or XML documents [W3C21][MDN CSS 21]. JavaScript is an interpreted programming language possessing first class functions and is currently utilized as the primary scripting language for the Web. JavaScript may be used to add dynamic behaviors and interactivity to a Web page, but is not only limited to Web pages and is used in Adobe Acrobat, Apache, and Node.js [MDN JS 21]. These languages were chosen for the Implementation of QAS as they can be used for the creation of an easily accessible application that will be usable within the foreseeable future.

1.3 Project Objective

QAS is an educational tool providing both simulation and visualization of various queueing systems to be used freely by students, professors, researchers, or any individual with the desire of learning queueing concepts. The primary goal is the provision of clear, accurate, and understandable representations of various queueing systems in an readily accessible fashion. The combination of the simulator and the visualizer unveils the inner workings of queueing systems -- every step and path a job takes the system progresses may be seen while accurate numerical results are provided. QAS is meant to be easily accessible as a web application created utilizing JavaScript, CSS, and HTML and runnable in any modern browser.

1.4 Outline and Organization

The first item to continue with is section two, Concepts and Design. This will contain a basic overview of queueing theory, descriptions of the structure of simulators, and how visualization may be obtained. Following this is chapter three, implementations. It is here where the internal structure of the program is described. Experimental results are afterwards and finally conclusions and extensions.

After the body of the thesis references are provided and appendices. These appendices contain the requirements to run the program, usage instructions, and the source code.

2. Concepts and Design of QAS

2.1 Overview of Queueing Systems

A queueing system can be described as a system with a service to be rendered and a demand for that service. To begin, a job needs to enter a system, this is the arrival. The job arrives when it requests to enter the first component of the system. The queueing mechanism is how enqueued jobs are distributed to the servers. Queuing mechanisms take and dispatch jobs in an ordered fashion, generally by the time of arrival. This queueing scheme comes in various arrangements: jobs may be pulled randomly by lottery, given priority levels, or be processed by time of arrival. The rules for the queueing methodology are defined by the system itself. The most simplistic queueing system is a single queue followed by a single server. In this system jobs arrive, are enqueued, dequeued, processed by the service facility, then depart. This is similar to a line for a single cashier at a grocery store. The study of queueing theory is of paramount importance to business operations. How long will it take for a customer to receive a product? What is the maximum number of jobs that can be serviced within a given time? Will a different arrangement of service facilities improve efficiency? All these questions may be answered through queuing theory.

As note previously, whenever a demand is made from a service there is an implication of a queueing system. While simple at a base level, complexity can very quickly become overwhelming. Arrival and processing rates are not uniform in most real-world situations. After initial departure a job may not be finished and must return to the system for further processing. Different jobs may follow varying paths through a given system. The service and arrival times are not always constant, arrivals may come in clusters and service times can vary from task to task.

As complexity of systems grow it can become mathematically challenging to determine each state within a given queueing system at a single point in time. Models can be created based off of real-world situations and be utilized to accurately predict behaviors of such a situations. The validity of the model created varies depending how completely the model matches the parameters of the components being described. Simple models may produce exact results, however other mathematical representations are estimations. This puts forth the importance of simulation and the development of simulator programs. Simulation may be used to grant useful information such as maximum load, utilization, and response times with confidence in the results when designed correctly. In addition to generating results simulators are also utilized to optimize system layouts, analyze theoretical improvements, and find the best of multiple proposed systems.

2.2 Simulation

2.2.1 Approaches for Simulation

The most intuitive approach for simulation is time driven simulation. This would be where operations mirror the outside world and as time passes events will occur. Utilizing this methodology events are generated and processed according to the time passed, whether it is real seconds or a scaled unit of time. Time driven simulation is useful for displaying a system in an understandable fashion. This has some troubles, however; the results would only be available as quickly as time passes and not immediately. Resources are used for the maintained threads. Calculations are not efficiently done and accumulate error over time. Much time is wasted waiting for an event to occur instead of processing the events. Results are not guaranteed to be replicable, and the actions taken are not necessarily representative of the queueing system. General computers do not possess dedicated and precise time slots in which to run programs; there is a scheduling and splitting of resources required to achieve this. Not all computers are equal and vary in internal clock rates and processing potential—the results generated from the same seeded values can and will be slightly different. This margin of error may not be large, but it is unnecessary. This does not mean that time driven simulation is without purpose, it could be fantastic for visualizations and presentations of queueing systems.

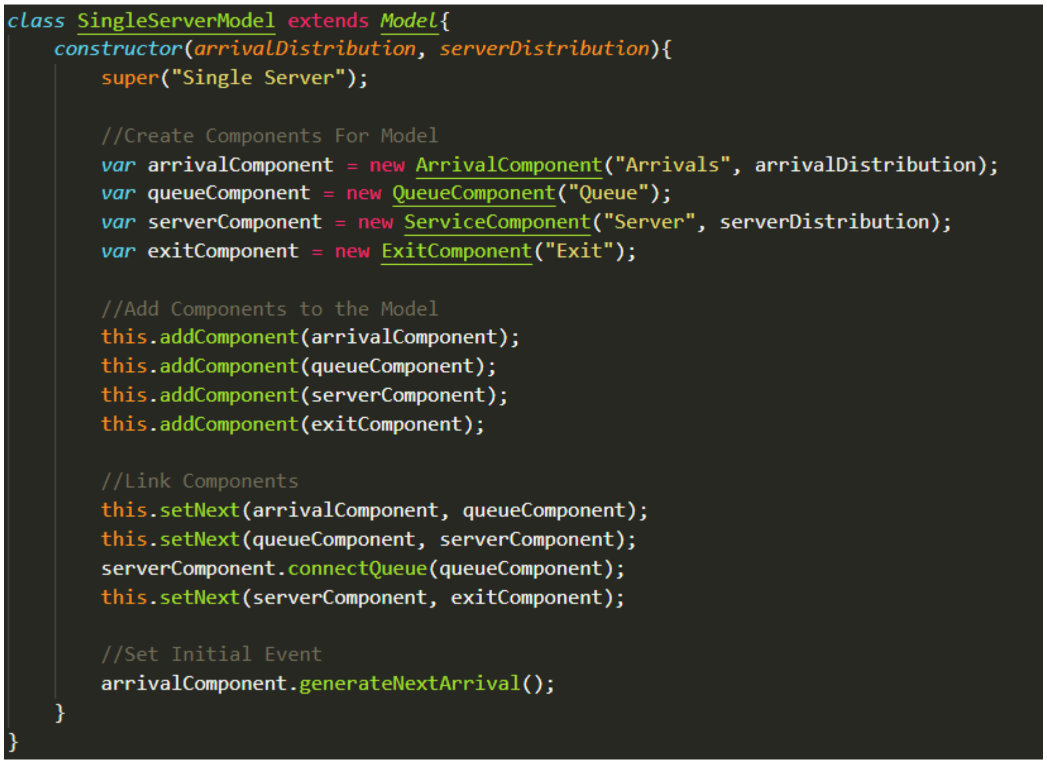
The second type of simulation to be discussed is event driven simulation. In an event driven simulator, the program advances through generating events and processing the next event to occur. To do this objects or indices must be created indicating the event to occur and the time at which the event is to be executed. The program time is initialized at 0. If there are no events present an event is generated applicable to the system or the running of the system is complete. The event closest in execution time to the current time is taken and the program's elapsed time is set to the time of the planned event. As events are processed new events along with their planned occurrence time are generated. No time is wasted waiting for real time to advance. The simulated time and events will match perfectly and are replicable and are independent of computer specifications assuming the framework is portable.

The simulator must also be capable of generating values based on the distribution rates of the internal components. Systems generally have variation between arrivals and within service times. The three distribution types used within QAS are uniform (evenly distributed between two values), constant (no variation), and Poisson (exponential). The internal random number generator for JavaScript will use xorshift128+ within most browsers. This has a high degree of randomness and is suitable for the generation of values.

2.2.2 Simulator Design and Organization

For this simulation program the event driven approach is used to minimize inaccuracies and maximize efficiency. A virtual time is recorded, initialized at zero. Events are organized, added to a sorted queue, and then processed according to the next time of occurrence. The foundation of the simulator lies within simulation objects. At the base level there is a class created named SimulationComponent which is expanded upon by all core elements of the simulation. The simulation component contains fields for type, its parent model, contained task, the time of the next event for that object, a flag for availability, and arrays for result data. The base functions of this class allow for the setting of the next and previously connected simulation components, a check for availability, result printing, updating output data, and a method to be later overridden for advancing tasks. This base class was developed to create a standardized template to be later expanded on for the various queueing network components. Components such as queues, service mechanisms, arrival and departure points, and containers for elements in parallel all expand from this base class. Designing the simulator in this fashion gives the ability to have a standardized base template for addition to models and allows reuse and ensures expected behavior.

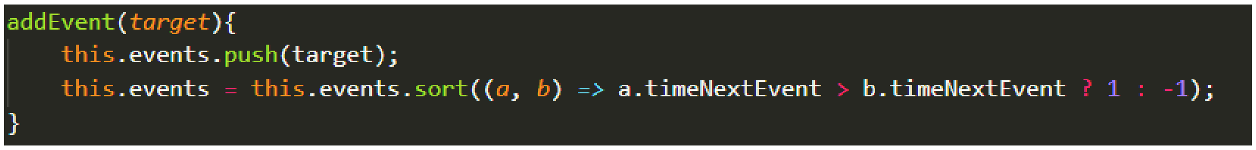
To create a queueing system the simulation components are added to an object that functions as a container: the simulation model. A sample of how a model is built is shown in the figure below.



```
class SingleServerModel extends Model{
    constructor(arrivalDistribution, serverDistribution){
        super("Single Server");

        //Create Components For Model
        var arrivalComponent = new ArrivalComponent("Arrivals", arrivalDistribution);
        var queueComponent = new QueueComponent("Queue");
        var serverComponent = new ServiceComponent("Server", serverDistribution);
        var exitComponent = new ExitComponent("Exit");

        //Add Components to the Model
        this.addComponent(arrivalComponent);
        this.addComponent(queueComponent);
        this.addComponent(serverComponent);
        this.addComponent(exitComponent);

        //Link Components
        this.setNext(arrivalComponent, queueComponent);
        this.setNext(queueComponent, serverComponent);
        serverComponent.connectQueue(queueComponent);
        this.setNext(serverComponent, exitComponent);

        //Set Initial Event
        arrivalComponent.generateNextArrival();
    }
}
```

Figure 2.1: Model Construction

The desired components are added and then the system may be run. The base Model class is a container that adds and links the internal components for the queueing system and holds universal metrics—number of events processed, if visualization is enabled, throughput of the system, response time of the system, and the number of events processed. The elapsed time for the simulation is not contained within the model but is instead of a global variable that can be easily accessed from other contexts without first accessing the model object. Class based design using JavaScript does not behave like classes within other languages such as C, Java, or Python. When "this" is called within an object the object being referred to depends on the context in which it is called. With timers, if a "this" function is called when inside of a class such as singleServerModel.sampleFunction() if that function contains a "this" keyword the "this" will refer to the context of the timer in the document instead of the model. There are ways around this, but it adds extra complexity. The model has events added to it through its contained simulation components and possesses the function to process a single event. The events are a reference to the object that contains the event and nothing more.



```
addEvent(target){
    this.events.push(target);
    this.events = this.events.sort((a, b) => a.timeNextEvent > b.timeNextEvent ? 1 : -1);
}
```

Figure 2.2: Simulation Event Creation

To process an event the event is popped from the event array and that object has its advance task function called to forward it through the system.

An overview of the design of the QAS simulator has been discussed, a more in-depth discussion of the implementation of the simulator is contained within section 3.3 Simulator Implementation. This contains more coding details and less concept discussion.

2.3 Visualization

Since QAS is also a visualization tool there must be some integration with the visualization component. If visualizations were done and updated purely using the event driven system, it would be incomprehensible since calculations will occur faster than human eyes can see. The visualizer must be time based and work with some form of scaled time to render understandable animations. The visualization is achieved through timer updates alongside associated drawing functions.

2.3.1 Visualizer Design and Organization

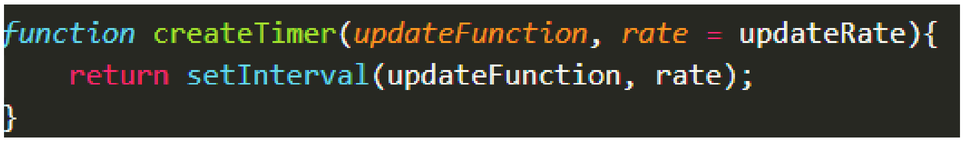
In order to first visualize queueing systems, the ability to draw onto the page must first be achieved. This is done through the HTML canvas element. The canvas, as the name suggests, is a component that may be drawn upon in the web page. Items could be drawn through other means using default components of CSS, HTML, and JavaScript but this allows less flexibility and control over what is being rendered. Support is given for the creation of shapes and animations, however there are some automatic assumptions made using these provided utilities and the behaviors are limited by being preset. For more control the canvas is utilized, which can draw on any specified pixel within the canvas.

Visualization objects are created in a similar fashion as the simulator has simulation components. The base class in this case is the DrawableObject. It possesses coordinates for starting and end points, flags for availability and drawing in reverse orientation. Default methods are resetting the visual, setting connects with the previous or next component, drawing, and accepting, receiving, and advancing jobs. The individual drawable objects extend from this in the same manner as the simulator expands upon simulation components. These are added to a visual model as well, with the visual model playing the role of a container for the placement and linking of system visual objects. Since all drawable objects have the draw method the model may render the entire model by chaining calls to its contained items. Events for the visualization are their own object instead of a reference to an object like the simulator. The visual events have the associated drawable object, the time at which the event is scheduled to occur, the type of event to occur, the task which is related to the event, and a reference to the simulation component. Visual events are required to be more complicated than the simulation events because they must sometimes wait for an animated job to arrive before forwarding it to the next element. The visualization cannot be instantaneous and immediately move on like the simulation—there are moments where the tasks are in transit.

For the visualizer to work correctly drawable objects have three job handling methods: accept, receive, and advance. Accepting a task means that the component has agreed to have the task delivered to it and will await its arrival. When the task arrives the receive function is called. When a task is forwarded to the next linked item the advance function is used. These are the three types of events associated with visualization. The visual events are created through communication with the simulator. The driver is created to establish the links behind the simulator and the visualizer. When a simulation event occurs, and if it requires visualization, a corresponding visual event will be added if a visual model is linked.

Animations require rules for movement and a methodology for updating the visualization. The movement is achieved through the task drawable object. When drawn, this task object renders a circle of a preset cycle of colors, at the coordinates of the task. The task may have a destination object set and will move at a rate configurable through the configuration file to the destination until it arrives. Upon arrival, the task will toggle a flag for visibility so that its draw method is not called upon to update and call the receive method of the destination object.

Movement is associated with the ticks of a timer created with an a configuration tick rate. The timer is a prebuilt function, of a simple form:



```
function createTimer(updateFunction, rate = updateRate){
    return setInterval(updateFunction, rate);
}
```

Figure 2-3: Timer Creation

A timer is created with a passed function to call every tick. By default, running the model will add the tick model method into this update function field. This update function will vary depending on the mode during runtime. The single step mode or maximum speed mode logic is different than this. A single tick of the visual model will increment the elapsed visualization time and redraw the system. The driver will call the process event function of the simulator if the elapsed visual time is greater than or equal to the time of the next scheduled event. Times must be different to maintain accurate results. A mapping of the visualization time to the simulator time is created. These are separate allowing the simulation results to remain accurate and preventing the visualizer from modifying or changing the simulator. As events occur and if the visualization mode is enabled relevant updates are sent to the visualizer. These visual events are not a reference to the simulation objects and are not created in a one-to-one ratio with simulation events as the visualizer does not need to know every processed event from the simulator, only what causes visual changes. The visualization time is incremented at static rates for updates, while the simulation time matches the scheduled event time completely. It should also be noted that the visualizer does not 100% mirror the simulator state at any given moment due to the need for tasks to move and be seen. Discussion of the code implementation may be found within section 3.2 Visualizer Implementation.

2.3.2 Visual Representations of Components

To visualize a queueing system each component within a given system must be distinguishable and easily understood. Simple shapes such as squares, rectangles, circles, and lines are used. Tasks are brightly colored and drawable objects containing tasks will match the color of the task inside. Queues will display the task it its circular form within the specific queue slot.

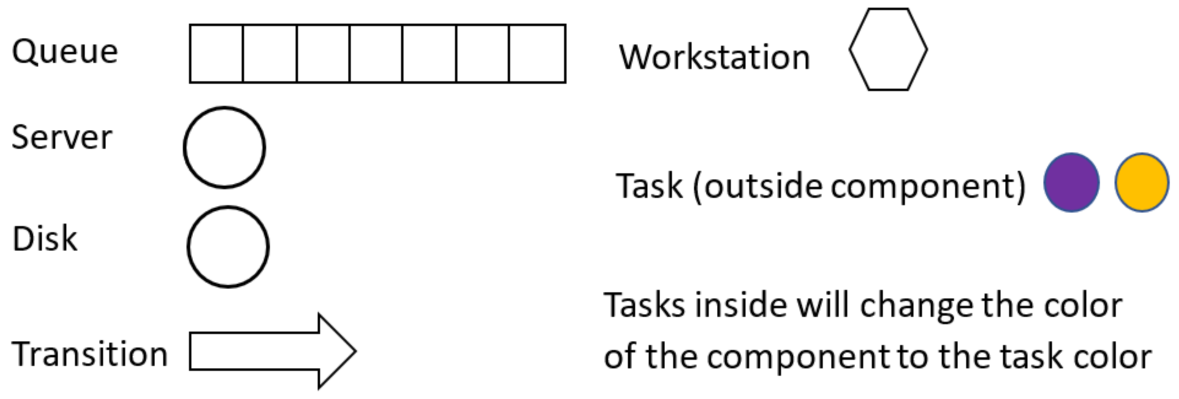

Queue
Server
Disk
Transition
Workstation
Task (outside component)
Tasks inside will change the color of the component to the task color

Figure 2-4: Model Key

For displaying a variety of queueing system concepts five models were chosen. These models demonstrate closed and open systems, splitting based on probability, and parallelization.

2.3.3 Single Server Model

The single server model is the simplest of models. It consists of a single queue and a single processor. Jobs arrive, are enqueued, are processed at the server and then exit the system. Adjustable parameters for this model are interarrival rate and distribution as well as server processing rate and distribution. This model could be representative of a drive through at a fast food restaurant: people arrive, enqueue, are serviced, and leave.

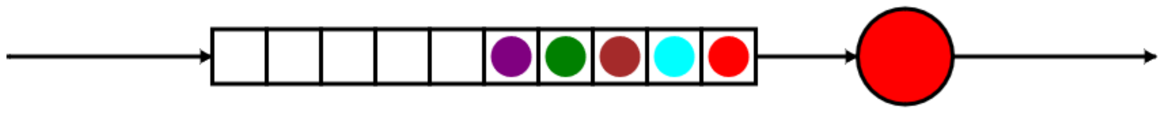


Figure 2-5: Single Server Model

2.3.4 Parallel Servers Model

The parallel servers model expands upon the single server model by adding additional servers in parallel. The distribution rates of each server within the parallel block are the same. Tasks from the queue are placed into the next available server for processing. If multiple servers are available, one is selected randomly. Adjustable parameters for this model include interarrival and service distribution rates and the number of servers in parallel . This type of model could be experienced at barbershops, a customer checks in (is enqueued) then gets service from one of many potential barbers.

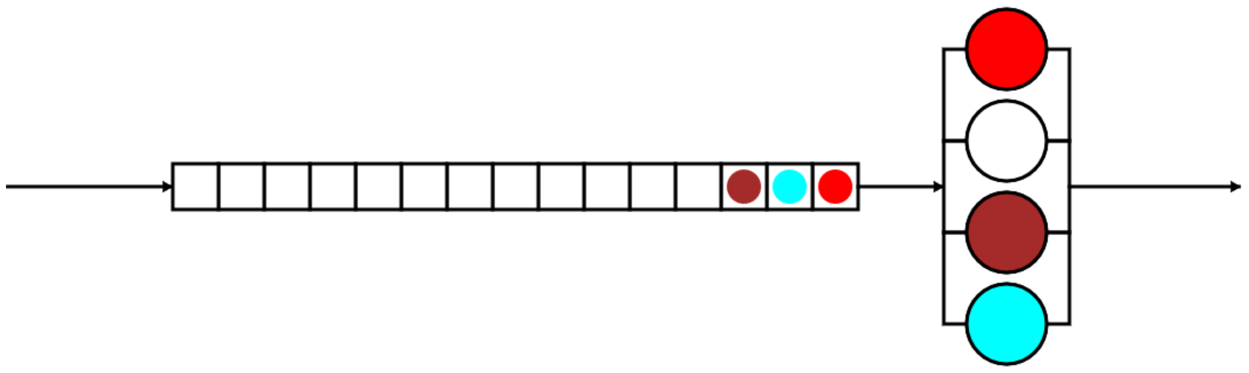


Figure 2-6: Parallel Servers Model

2.3.5 Single Server with Feedback Model

The single server with feedback model contains one server and one queue, just as the single server without feedback. However, this is a closed model, and allows for tasks to return into the system instead of leaving upon processing. The chance of returning to the system may be defined as a probability. The adjustable parameters for this model include arrival rate and distribution, server processing rate and distribution, and chance for departure. An interesting note on this model, is that the throughput is independent of the feedback probability if utilization is less than 1. The utilization, number of average elements in queue, and response times differ. This is understandable because every job that enters the system must be processed or else the system will overflow. It is also not possible to service jobs faster than they arrive—this makes the average throughput of this system remain the same even when feedback probability is changed and is achieved by putting additional strain on the server.

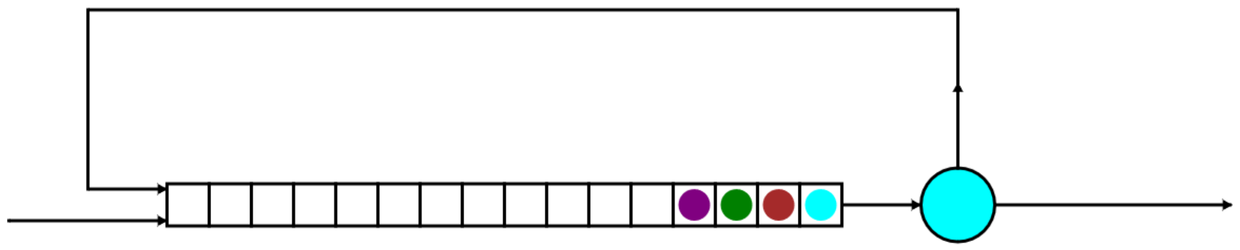


Figure 2-7: Single Server with Feedback Model

2.3.6 Interactive Workstations Model

The interactive workstation model possesses servers and workstations in parallel. A workstation will have a specified amount of thinking time and can be thought of as a component such as a person behind a computer filling out a form to submit online. The number of jobs in the system is equivalent to the number of workstations. Once a request is sent it is enqueued and awaits processing from an available server. This system has an adjustable arrival rate, thinking distribution, processing distribution, number of workstations and number of servers.

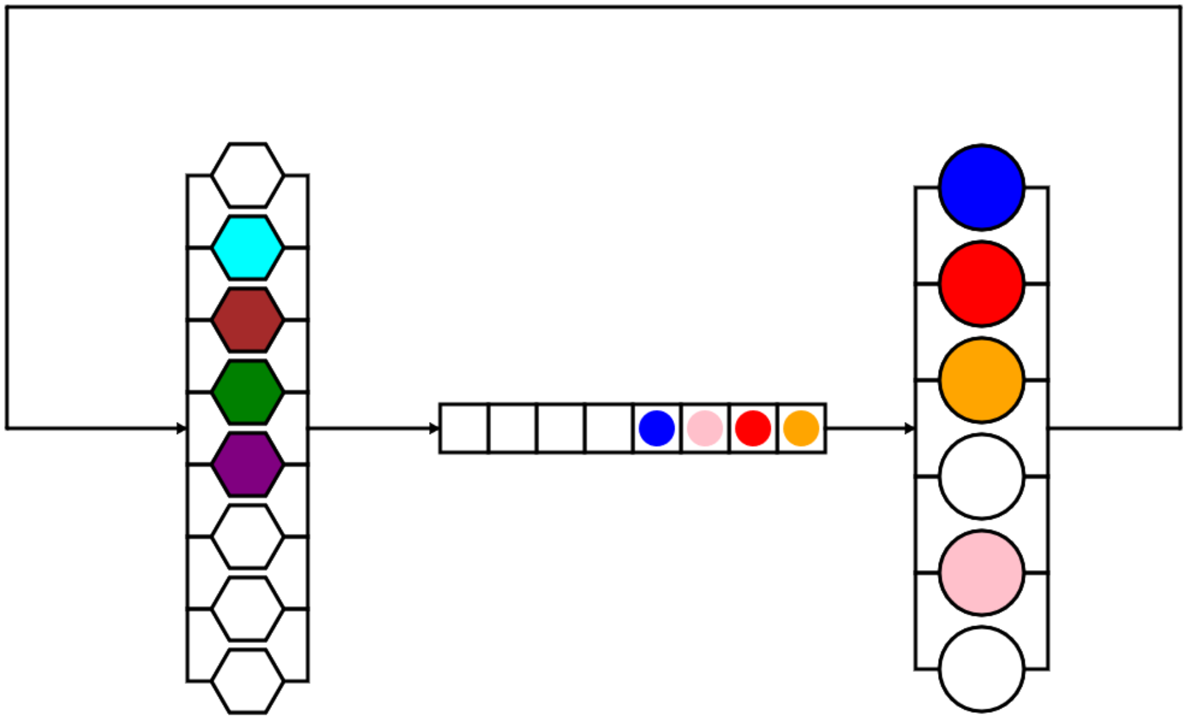


Figure 2-8: Interactive Workstations Model

2.3.7 Central Server Model

This central server model represents one central processor, a channel, and several disks in parallel. Each job may be processed by the CPU then be dispatched to a variety of disks from the channel. The CPU supports having a representation of multiple cores. The number of desired cores can be input, and it adds a parallel grouping with that number. The adjustable parameters for this model are the number of disks, probabilities for disk access, and distribution type and rates for the CPU, channel, and disks.

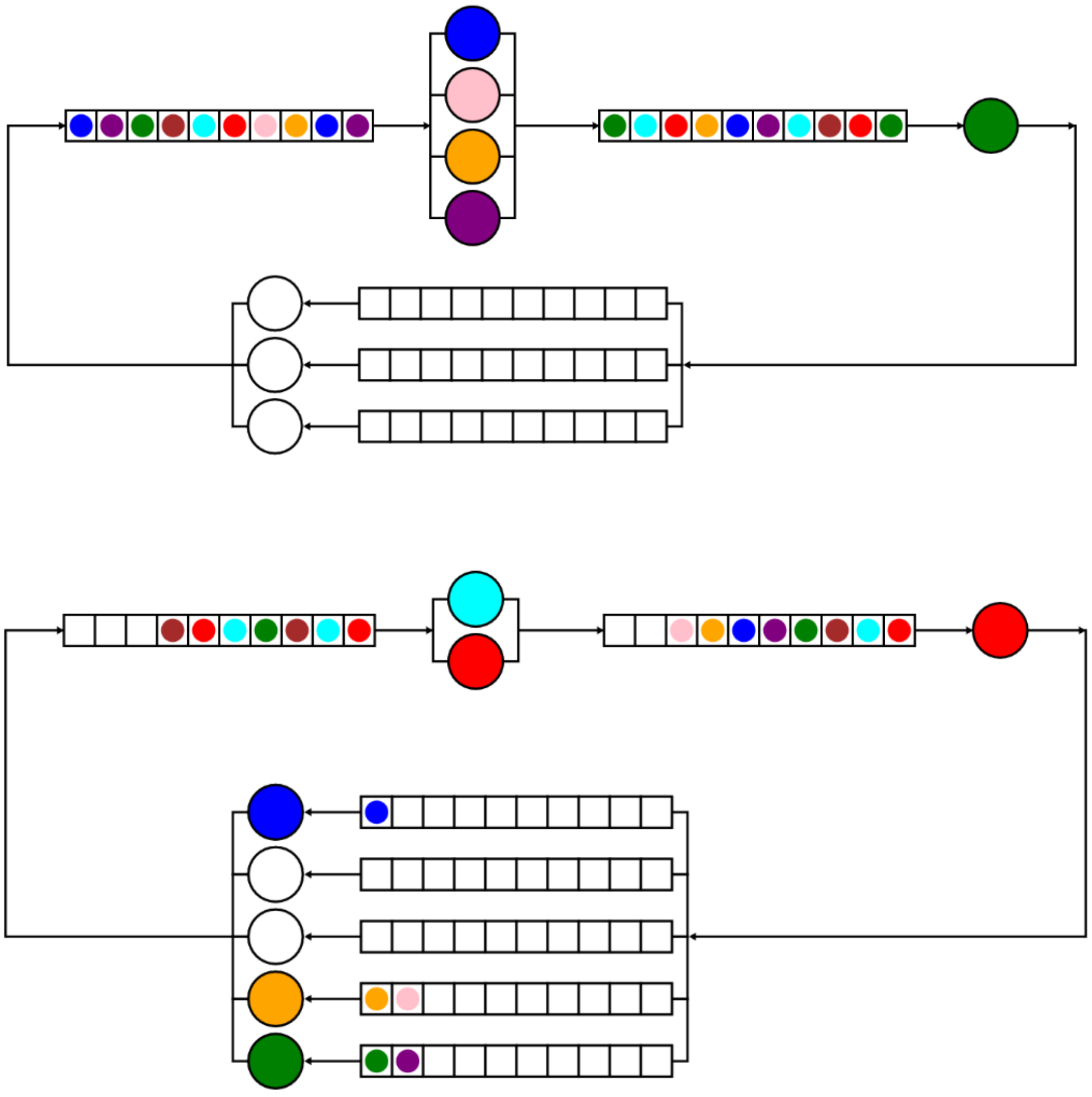


Figure 2-9: Central Server Model

3. QAS Implementation

3.1 Frameworks Utilized

QAS was created as a web application utilizing HTML, JavaScript, and CSS without external libraries. The decision to make the program utilizing these languages was to allow easy use and access of the program without external dependencies and additional installations. All data and calculates are handled on the client side preventing the needing of a database or server-side calculations. This also allows for continued support in the foreseeable future as these languages are the standard. Anyone with a computer and an internet browser should be capable of running QAS.

This program was created using primarily object-oriented programing. The reasoning behind this is to allow for reuse and easy expansion. Every model by default and each component has default functionality that may be expanded upon. Examples of this may be seen with all visual objects having coordinates, a draw function, and potential linking with other visual objects or with the simulation models being capable of having default functions for the linking of components and displaying of results. Base templates are expandable depending on the requirements. This not only allows reuse, but also brings some uniform behaviors for simplicity and understandability. Drawable objects may all be drawn, so they share the draw function. Systems may be started, stopped, stepped through, or have the visualization rate altered. Configuration inputs have listeners, organizational groups, and input-verification. The core logic can be implemented once, reused, and built upon.

The files in this project are organized by component type and location of use. The entry point of the program is labeled “index” and lies within the root of the folder. This is the landing page and where the user will first enter this program. The styling and formatting are handled by a CSS file within the “css” subfolder. HTML pages are stored within the “Pages” folder and are where the user will be directed towards for the use of the various parts of this program. The scripts or logic for these pages is written in JavaScript and held within the “Scripts” folder. Following this trend, the visual components are within the “Draw” folder and the simulation components are within the “Simulation” folder. The files were organized according to their purpose to keep the code manageable and structured.

3.2 Visualizer Implementation

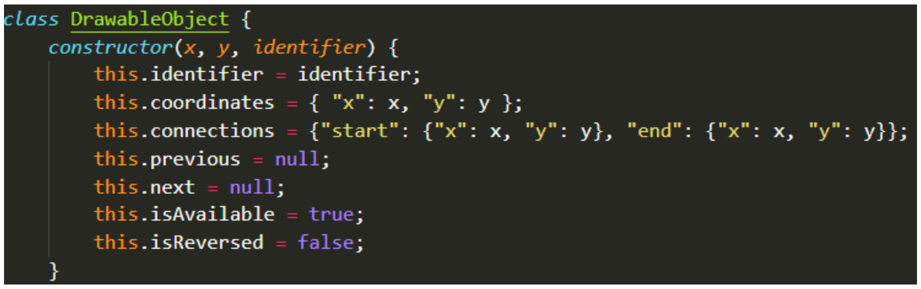
The visualizer consists of all drawable objects, the visual model, connections between drawable objects, and the rules for accepting and processing visual events. Adjustments for sizes and layouts can be found within the config.js file which was created to allow for quick adjustment and scaling of visual components. These variables include animation speeds, job movement rates, timer update interval, canvas size and scaling, line weight, colors, drawable object sizes, default spacings, and the number of queue slots to be drawn within a queue.

3.2.1 HTML Canvas

The canvas, as the name implies, is a component that may be presented and drawn upon. Initially, this is just empty and plain. The canvas is initialized with a location, height, and border color. It is created at the desired location and filled with a definable background color. This canvas is drawn upon to display the queueing systems and must be redrawn every update.

3.2.2 Drawable Objects

Drawable or visual objects are the core component of the visualizer. The DrawableObject is a base class that possesses core functionality needed by most, if not all, visual representations:

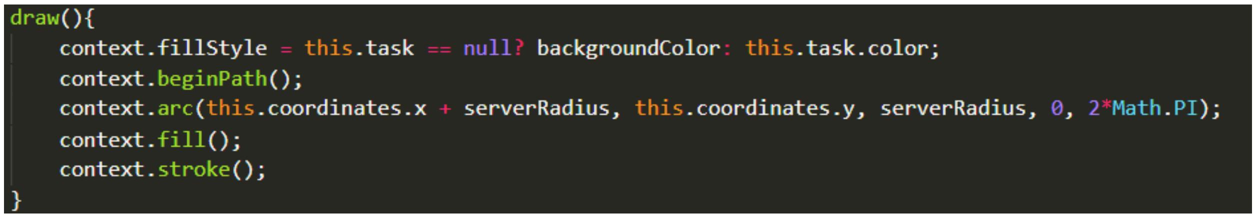


```
class DrawableObject {
    constructor(x, y, identifier) {
        this.identifier = identifier;
        this.coordinates = { "x": x, "y": y };
        this.connections = {"start": {"x": x, "y": y}, "end": {"x": x, "y": y}};
        this.previous = null;
        this.next = null;
        this.isAvailable = true;
        this.isReversed = false;
    }
```

Figure 3-1: Drawable Object

The base object contains an identifier for reference, coordinates for purposes of drawing, connections for ending and starting points, the previous and next linked objects, a flag to indicate availability, and a flag which notes that the object should be reversed. The junctions between drawable objects utilize the start and end coordinates to create connections. Drawable objects are rendered using a draw function, which is overridden by child classes. If the drawable object is reversed the rules for drawing may have to be altered depending on the object and the starting and ending connections must change.

To draw an object the canvas is used. Paths are created to draw through and color and line weights are set. The server visual's draw method is shown below to display how this is done.

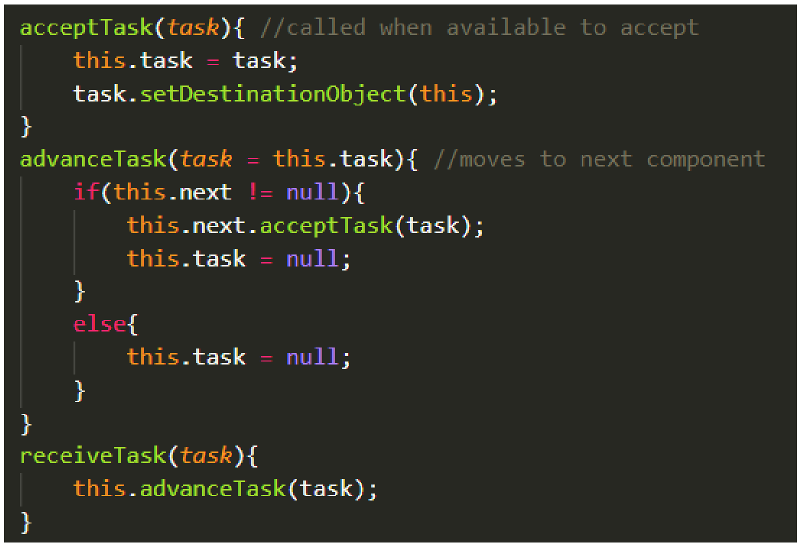


```
draw(){
    context.fillStyle = this.task == null? backgroundColor: this.task.color;
    context.beginPath();
    context.arc(this.coordinates.x + serverRadius, this.coordinates.y, serverRadius, 0, 2*Math.PI);
    context.fill();
    context.stroke();
}
```

Figure 3-2: Draw Method

The context refers to the canvas and the 2d context of the current page. The object is filled with the color of the task if present, otherwise with the canvas background color. The path is set then the stroke renders the item onto the canvas with the given path and settings. Some draw methods are more complicated than others, but the premise remains the same. Straight lines are created using the lineTo function and the position drawn from is specified with the moveTo function.

Drawable objects have base functionality for handling task events. These events are to accept, advance, and receive.



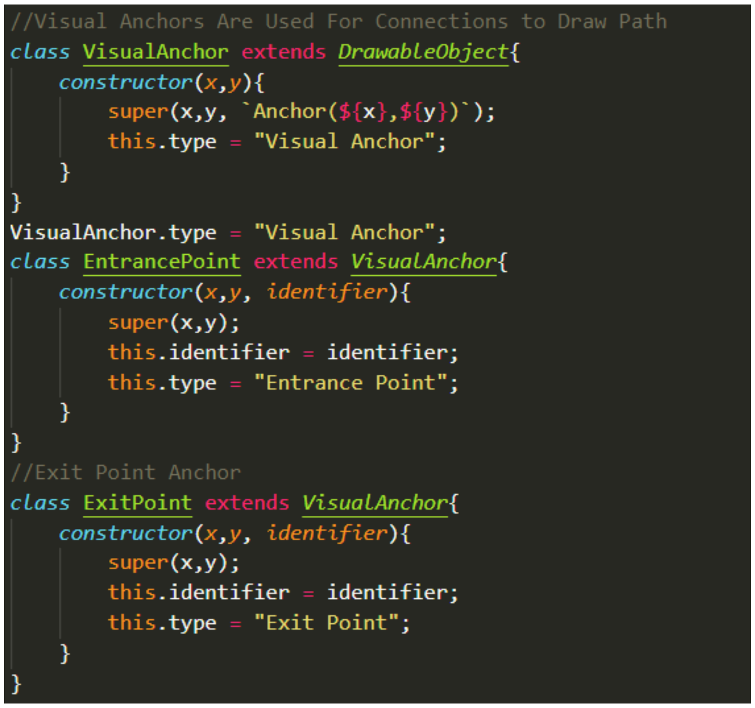
```
acceptTask(task){ //called when available to accept
    this.task = task;
    task.setDestinationObject(this);
}
advanceTask(task = this.task){ //moves to next component
    if(this.next != null){
        this.next.acceptTask(task);
        this.task = null;
    }
    else{
        this.task = null;
    }
}
receiveTask(task){
    this.advanceTask(task);
}
```

Figure 3-3: Visual Object Operations

Accepting means that the drawable object has intent to be given a task. The task will progress towards this object and upon arrival call the receive function. Advancing the task will forward it to the next drawable object. These primary functions will be overridden in visual objects that correlate with the simulation components. Without being overridden the task will move to the object then be immediately forwarded to the next connected element if it exists. If there is no next visual object, the task is deleted. JavaScript does automatic garbage removal when all references to an item are deleted.

Tasks are drawable objects. Tasks do not utilize the event processing components of drawable objects but use everything else. The task draw method will increment the x and y coordinates of the task according to the amount specified within the configuration file if the task is flagged as moving. Distance is never overshot as excess movement is pruned to prevent oscillation back and forth. Tasks begin moving through a set destination object function, which takes a drawable object as coordinates. This sets the destination x and destination y coordinates of the task to the start connection of the target object and registers the destination object then toggles the is moving flag to true. When the task arrives at the coordinates it will invoke the receive method on the destination object and set the is moving flag to false. If the task is not moving it will not be drawn, but still exist. This is because the drawable object has its own logic to change colors depending on the task within and these are not all the same size or shape of the circular task.

Connection points are called or labelled as “VisualAnchors” and derive from the DrawableObject class. These have blank draw methods and utilize the default accept, receive, and advance functions to forward tasks. Visual anchors can be thought of as intermediatory points between objects. Anchors exist wherever a path most be taken that is not a straight line between two points and their sole purpose is to aid in task navigation.



```
//Visual Anchors Are Used For Connections to Draw Path
class VisualAnchor extends DrawableObject{
    constructor(x,y){
        super(x,y, `Anchor(${x},${y})`);
        this.type = "Visual Anchor";
    }
}
VisualAnchor.type = "Visual Anchor";
class EntrancePoint extends VisualAnchor{
    constructor(x,y, identifier){
        super(x,y);
        this.identifier = identifier;
        this.type = "Entrance Point";
    }
}
//Exit Point Anchor
class ExitPoint extends VisualAnchor{
    constructor(x,y, identifier){
        super(x,y);
        this.identifier = identifier;
        this.type = "Exit Point";
    }
}
```

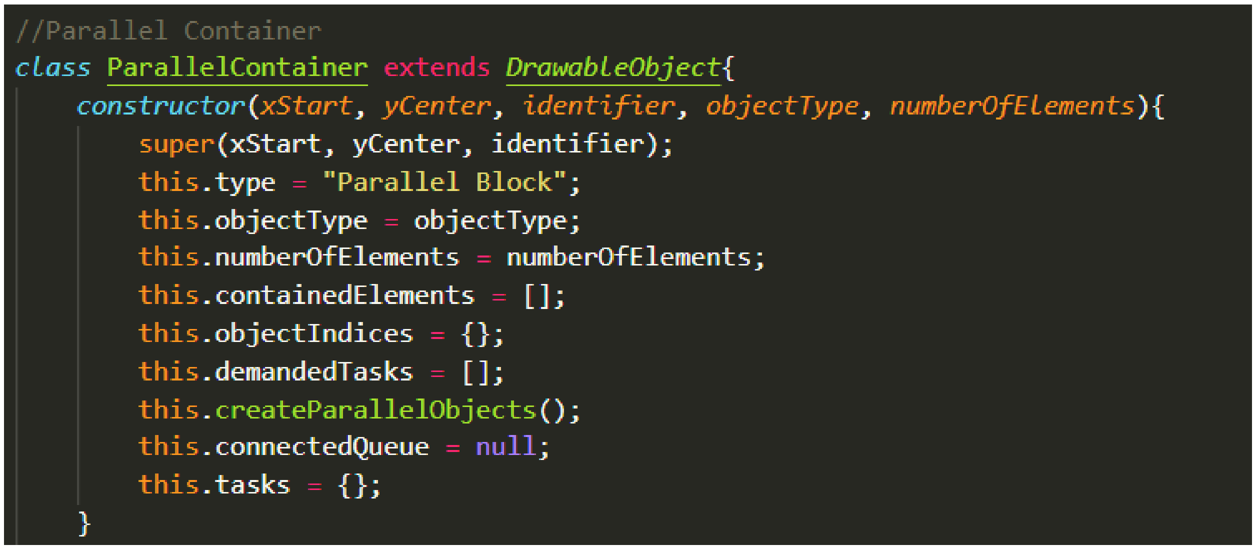
Figure 3- 4: Visual Anchors

There are two classes that are derived from visual anchors. Entrance points and exit points differ only from the default anchor by the label, but these labels are used to clarify where the queueing system begins and ends. The default visual anchor has its identifier automatically generated through the coordinates that it is created at.

Service mechanisms are labeled as ServerVisual and extend the base DrawableObject. These server visuals have the addition of a backlog and a flag to immediately process a task upon arrival. The backlog represents that an advance function was called, but the task has yet to arrive at the destination. This could be the case when a small service time is generated. The simulator would have already processed the task at the elapsed time, but the visualization is behind in this case. The backlog is created so that the visualized tasks do not teleport from the middle of its transition to the end connection of the server. It is a hash map that maps the identifier of the task to the task itself. In the case that a task needs to be added to the back log the immediately process flag is triggered and upon arrival the task will be advanced. Components can be connected to service mechanisms and this is done with queue visuals. When a server is connected to a queue visual it will upon call of its advance method make the queue advance its task as well due to the task being present within both a slot of the queue visual and the server visual. Advancing the task will remove the entry of that task from the back log and toggle the immediately process flag to false before calling the accept method of the next drawable object if present. If the service mechanism is contained within a parallel component the advance method will defer to the parallel container's advance method. The visual objects for disks, workstations, and feedback servers extend from this ServerVisual class. Disks and workstations simply have altered types and draw methodology. The feedback server allows for multiple destinations after advance is called. When the simulation component finds the path to take it is stored in a taskDestinations object that has a correlation between the task identifier and the destination to forward it to. These options are the exit point for leaving the system or the pathway to feed back into the system.

The queue visual object when initialized will add queue slots equal to the size indicated in configuration. These queue slot visuals only have a draw method to fill the colors of the squares. The queue visual has the current number of tasks within the queue, a mapping of task name to queue slot information, an array for tasks waiting for an available server that are within queue slots and must be drawn, and an array for tasks that lie outside the visible range. When the queue receives a task, it will check if the connected component is available and ready to receive the task. If this component is ready it will call the receive method of the correlated service mechanism. Else if the component is not ready the task will be added to the waiting array if there are queue spaces available or the overflow array if there is no space in the visualizer queue. Upon advancement the queue will remove the task from the queue then shift the queue slots over. Shifting the slots over will move tasks within the queue slots, check for advancement of waiting tasks if present, and move a task from overflow into the next available slot if that is not empty.

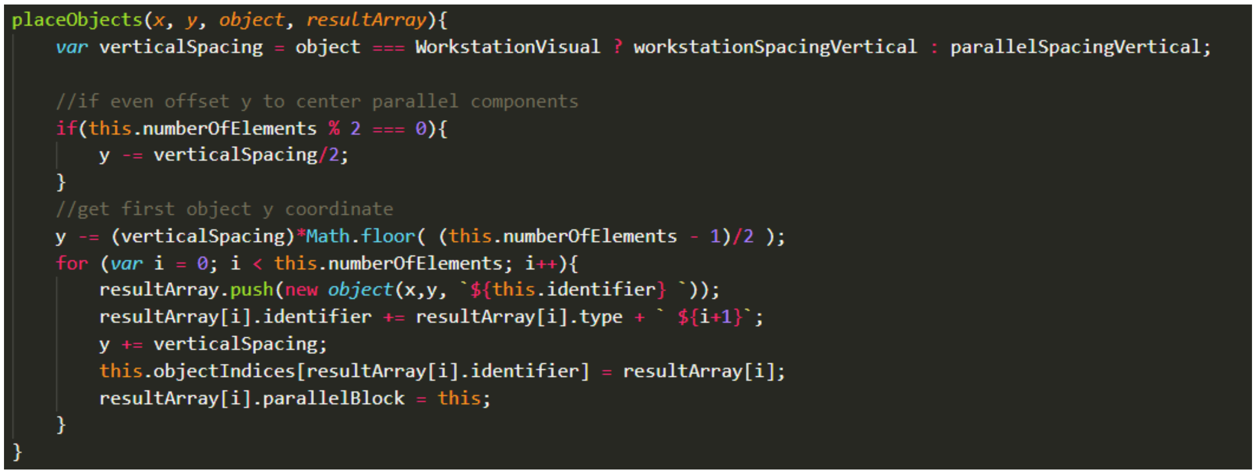
The last DrawableObject class is the ParallelContainer class:



```
//Parallel Container
class ParallelContainer extends DrawableObject{
    constructor(xStart, yCenter, identifier, objectType, numberOfElements){
        super(xStart, yCenter, identifier);
        this.type = "Parallel Block";
        this.objectType = objectType;
        this.numberOfElements = numberOfElements;
        this.containedElements = [];
        this.objectIndices = {};
        this.demandedTasks = [];
        this.createParallelObjects();
        this.connectedQueue = null;
        this.tasks = {};
    }
```

Figure 3-5: Parallel Container

The parallel container object contains multiple drawable objects within parallel and manages task flow to and from those objects. This works by taking a set of object types and the number parallel entries to make. These object types are not necessarily only one element: they may be multiple items, for example a set of queues leading into a disk. If there are multiple object types these will be linked together upon initialization. Placement of these objects is done through the place objects method.



```
placeObjects(x, y, object, resultArray){
    var verticalSpacing = object === WorkstationVisual ? workstationSpacingVertical : parallelSpacingVertical;

    //if even offset y to center parallel components
    if(this.numberOfElements % 2 === 0){
        y -= verticalSpacing/2;
    }
    //get first object y coordinate
    y -= (verticalSpacing)*Math.floor( (this.numberOfElements - 1)/2 );
    for (var i = 0; i < this.numberOfElements; i++){
        resultArray.push(new object(x,y, `${this.identifier} `));
        resultArray[i].identifier += resultArray[i].type + ` ${i+1}`;
        y += verticalSpacing;
        this.objectIndices[resultArray[i].identifier] = resultArray[i];
        resultArray[i].parallelBlock = this;
    }
}
```

Figure 3-6: Parallel Container Placing Objects

This will automatically create visual objects and add them to the passed location. The vertical placement is found depending on if an odd or even number of elements is present. Parallel containers must be centered as a whole and to do this the middle element needs to be in line with the middle of the container if odd and offset by the vertical spacing if even. The horizontal placement is set depending on the width of the interior objects. New objects are created at the calculated coordinates, given labels, and have that identifier mapped to the object for quick lookup. For task management when an event is done for a specific server it is added to a tasks hash map to link the task to the destination server. The previous component when checking availability will verify that this task correlates to a destination within the container before advancing. Tasks cannot be accepted if the server is not ready. Internal components work as they do normally, but the container provides the routing to and from the parallel block. Reversing a parallel container will reverse all items inside and adjust their spacings.

3.2.3 Connections Between Drawable Elements

Functions were created to perform basic tasks, such as drawing arrows, lines, and connecting parallel components more easily. These are simple functions that take either x,y starting and ending coordinates alongside a color, or an array of an indeterminate number of visual objects that will have connections drawn between them. These basic functions save considerable time and are utilized whenever model components are connected. Upon drawing a model will call these connection functions to easily establish clear visual representations of the junctions.

3.2.4 Development of Visual Models

Visual models are containers for visual objects and possess the elements necessary to draw the entire model. There is a unique model for each queuing system: single server, single server with feedback, parallel servers, interactive workstations, and central server. The base model class possesses containers for the interior components, events, connections, interval identifier (for timer), and a flag stating whether a timer is currently running. The visual models organize all visual objects and call their draw methods, as well as resetting the models when necessary. The visual models provide another very necessary service: to create, handle, receive visual events. Like the simulator, the visual model is also event-based. When linked to the simulation component the simulator will create and pass events to the visualizer to undergo actions. These visualization events are different than the simulation events—they pertain to the three main visual object functions: accept, receive, and advance. The processing of these events is what makes the program flow and give a representation of what is occurring in the simulator. The events have an associated time and are processed according to a tick method that updates the current time corresponding to the tick rate and runs the next visual event if the next event time is equal to or greater than the current elapsed time.

3.2.5 Animations and Timer Updates

Motion and animations are made possible using timers. These timer related functions are found within the driver.js file. Timers are created through the setInterval function, which takes a function to call on update and a rate for updating. For running the model this is passed the previously discussed tickOnce method present within the models. This will increment the elapsed time according to the update rate and process events accordingly when the standard run model function is called. Once the timer is started the rate cannot be changed, because of this it is necessary to keep track of the interval identifier to delete the timer later. Without this identifier the timer may not be deleted and if a new interval is made the old timer will still exist and cause unnecessary work. This interval is deleted within a stop model function. The model runtime needs to be stopped every time the speed of the model is modified. Then a new timer must be created with the updated rate. When using the step once mode the program will set the current elapsed time to the time of the next event, process the next simulator event, and set a timer including the tick once method. The simulation will only process a single event, but the visualization will continue to move. When the step mode is toggled off the time continues as normal from the time of the last event occurrence. For the simulation mode updates the timer interval is set to update a thousand times a second and the next event in the simulator is called each update—processing a thousand events every second. The visualizer is not updated while this mode is on. After simulation mode is left the state of the visualization will no longer match the internal state of the simulation because thousands of events have passed. The visual model is reset, and all tasks are cleared, and all components are set to how they would be upon initialization. After the reset the state of the simulation is copied to the visual side by iterating through each simulation object. When the copy is finished the states are now the same and the user may interact with the model as they desire.

3.3 Simulator Implementation

The simulator has the responsibility of the creating and running the various models and generating then displaying accurate results. It is a discrete event simulation that will continuously generate and process events according to the time at which the next event is scheduled to occur.

3.3.1 Simulator Objects

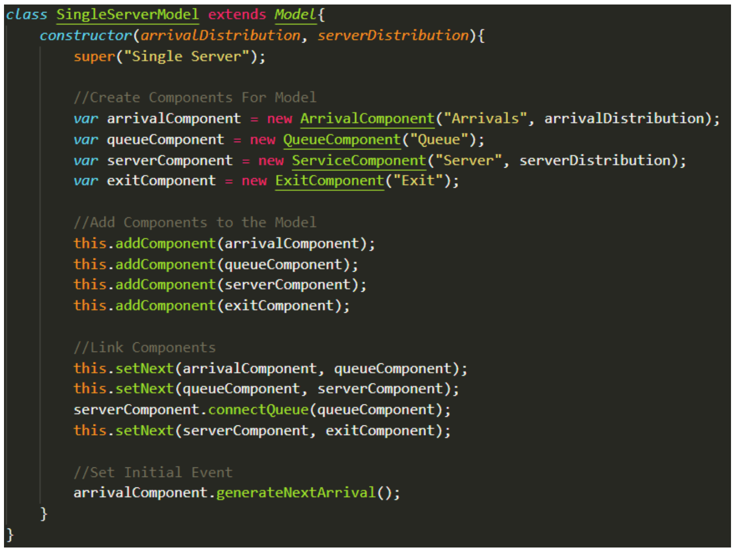
The foundation of the simulator objects is the SimulationComponent class. This contains basic logic for output of results, the type of object it is (for example server or disk), a flag for availability, the time at which the next event is to occur for that object (if any event is scheduled), and the task currently correlated with this object. The potential output is separated into headings and data. The headings are simply labelling for results, while the data are the values. Separation was created between the labels and the actual values. When placed in a table the data may be iterated through easily and displayed independently from the labels or alongside them. In tables with multiple components of the same type labels only need to be printed once while every value that corresponds to the labels needs to be output for each item within the table. Additionally, the data needs to be updated upon event processing before output is created for the user while the labels are static. There is a mapping between the headings and data through a hash map for quick look up and retrieval of the data. For the handling of events simulation objects possess the accept and advance functions. There is no receive function as the simulator does not need to wait for the visualization and is kept separate. Accept states the intent to accept a job, while advance forwards the job to the next linked component. Every simulation object also has a copy method to copy the state of the simulation to the visual objects upon exiting simulation only mode. Communication with the visual objects is necessary to link the simulator to the visualizer. As simulation events are created and processed corresponding visual events are forwarded to the visual models. These are not necessarily a one-to-one relationship—not every simulation event needs to be noted by the visualizer and not all visual events have a simulation counterpart. The simulation component base class is extended into QueueComponent, ServiceComponent, ArrivalComponent, ExitComponent, and ParallelComponent. The ServiceComponent is further extended into WorkstationComponent, DiskComponent, and FeedbackServer. Each of these simulation objects overrides and expands upon the base logic to undergo their specific roles.

Objects within the simulation that are not considered to be simulation components are tasks and distributions. The tasks are simplistic and do not need the base functionality present from the simulation component base class. Tasks only have an identifier and creation time. The creation time is used to calculate response time and throughput, while the unique identifier is utilized to keep track of the created tasks. Distributions contain the type of distribution and the values corresponding to that distribution. Uniform distributions have upper and lower bounds, exponential distributions have a mean, and constant distributions have a value. Alongside this base data the distributions possess a generate method which will generate a value corresponding to the distribution type and values.

The queueing systems at the core is only those three items: simulation components, distributions, and tasks. Simulation models are created to hold and establish relationships between these components.

3.3.2 Simulator Models

Simulation models are used to group and define specific models. The Model class keeps track of all simulation components within that model and links the components together. An example of this can be shown below:

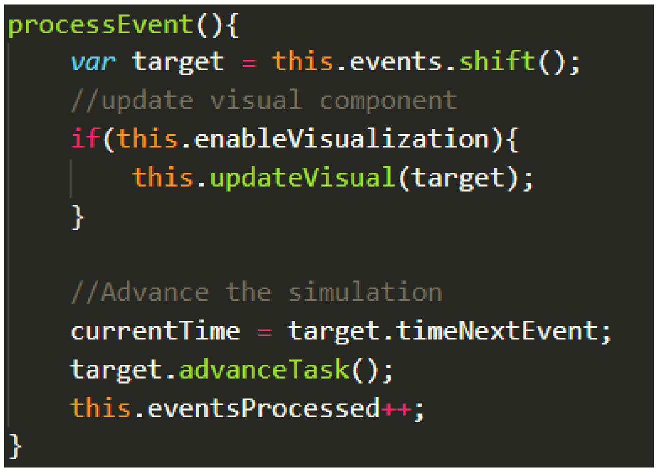


```
class SingleServerModel extends Model{
    constructor(arrivalDistribution, serverDistribution){
        super("Single Server");

        //Create Components For Model
        var arrivalComponent = new ArrivalComponent("Arrivals", arrivalDistribution);
        var queueComponent = new QueueComponent("Queue");
        var serverComponent = new ServiceComponent("Server", serverDistribution);
        var exitComponent = new ExitComponent("Exit");

        //Add Components to the Model
        this.addComponent(arrivalComponent);
        this.addComponent(queueComponent);
        this.addComponent(serverComponent);
        this.addComponent(exitComponent);

        //Link Components
        this.setNext(arrivalComponent, queueComponent);
        this.setNext(queueComponent, serverComponent);
        serverComponent.connectQueue(queueComponent);
        this.setNext(serverComponent, exitComponent);

        //Set Initial Event
        arrivalComponent.generateNextArrival();
    }
}
```

Figure 3-7: Simulation Model

Components can be easily added and linked to diverse and unique queueing systems. If a new model wants to be built simply create and add the corresponding simulation components and it should be runnable assuming it is a valid model.

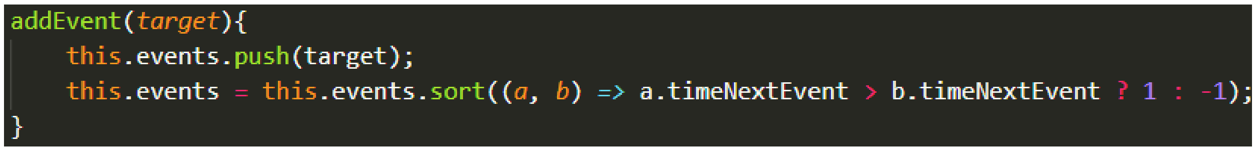
Models contain the events and execute the next event to occur through a process event function. This is a simple method that advances the target object of that event and updates the visual representation if visualization is enabled.



```
processEvent(){
    var target = this.events.shift();
    //update visual component
    if(this.enableVisualization){
        this.updateVisual(target);
    }

    //Advance the simulation
    currentTime = target.timeNextEvent;
    target.advanceTask();
    this.eventsProcessed++;
}
```

Figure 3-8: Simulator Event Processing

The events themselves are not objects, but instead pointers to the element which needs to be advanced.



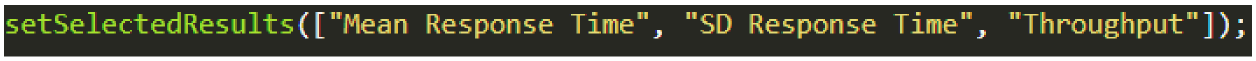
```
addEvent(target){
    this.events.push(target);
    this.events = this.events.sort((a, b) => a.timeNextEvent > b.timeNextEvent ? 1 : -1);
}
```

Figure 3-9: Simulator Events

The model does not need to know how the details of how the simulation component processes the events, they are designed to work independently so that components can be easily added. One event will be executed with each event call indefinitely. This may be utilized and called according to the needs. In simulation mode the process event function is called a thousand times a second. For testing purposes, a set number of events may want to be processed or jobs handled.

3.3.3 Result Selection and Output

Desired results may be chosen and configured within the simulation models. By default, all components will have their fields and labels printed but specific results can be chosen and organized for output selection and formatting. The model base class does this through categorizing components by type to bring some sort of structure to the output. This default output scheme is suitable for some models such as the single server models, but does not work with models with large amounts of internal components such as central server and interactive workstations as the resulting output has too much to be displayed cleanly. In these cases output needs to be set or chosen and tables must be made to group more concisely. This is done by creating custom tables through functions which generate table headings and table data by rows. Selected results may be set on specific objects, such as:

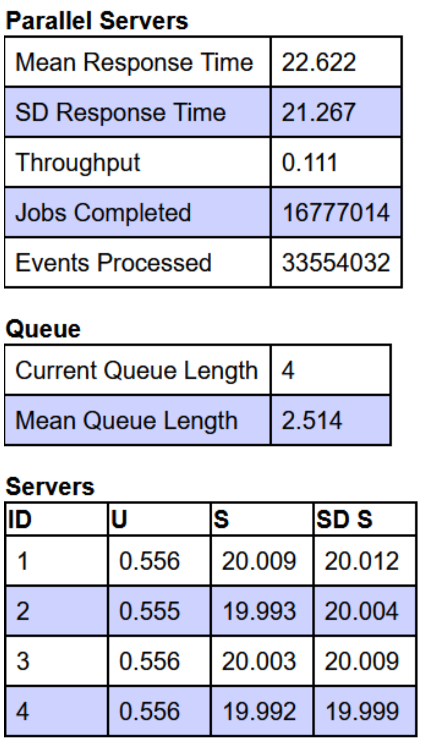


```
setSelectedResults(["Mean Response Time", "SD Response Time", "Throughput"]);
```

Figure 3-10: Result Selection

These selected results will be displayed automatically on output. If more specific rules must be set the displayResults function within the model itself may be overridden and have all printed elements configured. This is done with parallel components to group results so labels may only need to be printed once.

The results are rounded to two decimal places and converted into HTML code. Tables are created, headers are added, and data are populated. The string of HTML code is added to the results output section of the page and formatted according to the style rules set in the the CSS. Standard output tables have a label for the component named followed by label + data pairs. The formatting of odd and even rows background color varies to increase legibility and the spacing rules for label and data are different. The more condensed tables have different formatting than the standard output. The labels are at the top for data fields and the rows contain the element identifier followed by the data values of that specific element.



Parallel Servers

Mean Response Time	22.622
SD Response Time	21.267
Throughput	0.111
Jobs Completed	16777014
Events Processed	33554032

Queue

Current Queue Length	4
Mean Queue Length	2.514

Servers

ID	U	S	SD S
1	0.556	20.009	20.012
2	0.555	19.993	20.004
3	0.556	20.003	20.009
4	0.556	19.992	19.999

Figure 3-11: Sample Results

Results are returned as a string of HTML code when the displayResults function is called on a specific component or model. This function is called on every tick of the timer in all modes except for the single step mode. It is never called by the simulator or visualizer and is instead called by the driver that links the two together and deals with the calls to run or stop the models.

3.4 Page Implementations

HTML pages are what is seen and rendered in the browser. These are organized by role, with each page having a specific purpose. Pages were made so that by default they are essentially blank templates with organizational regions that will be built upon through the calling of JavaScript code as the need arises. The content for this program cannot by its nature be static content as it contains animations and the selected configuration or model to be displayed correctly. Adaptability is a requirement, and it is not sufficient to simply show the same content for every model and it is not efficient to create unique pages for every single presented model. JavaScript code and styles for the content being displayed are linked within the pages and the pages are filled and made interactive through these linked scripts.

3.4.1 Obtaining Configurations

To obtain user input for model configurations user data is stored within cookies. If cookies are disabled, the program will not run correctly. The user selects one of the available models within the model selection page and this choice is saved as a cookie. The model configuration page is then rendered utilizing the chosen model. If no choice is made the program will return to the index instead of displaying an empty page.

Inputs were perhaps made more complicated than they had to be, but each field is modular and inputs may be easily added and removed. To begin with, default parameters and validations for possible ranges of these values are stored within hash maps. Every model contains key value pairs of all possible inputs which are used for automatic filling if no user-submitted parameter currently exists for that input. If an input was saved previously that will be filled instead. For range validations there is another hash map created with minimums and maximums of various fields: queue size, feedback probabilities, number of initial jobs, and number of parallel components have assigned ranges.

As with the simulation and visualization components the configuration was done using a general base class that is later expanded upon. In this case it is the InputField, which consists of an identifier, type, description, name, and value with a function for later overriding to generate HTML representative of that input field. The two main categories of input used are radio buttons and text fields. Radio buttons provide a mutually-exclusive set of parameters and text fields take character inputs. Without formatting radio buttons look like this:

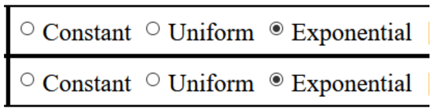

○ Constant ○ Uniform ◉ Exponential
○ Constant ○ Uniform ◉ Exponential

Figure 3 -12: Default Radio Buttons

To make choices clear the CSS formatting removed the button portion completely through overriding the position, size, and visibility of the button and modified the input so that a square is drawn and filled around the selected choice that is interactable like a button.

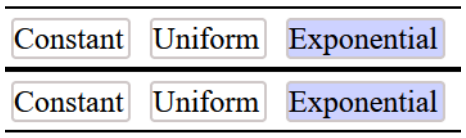


Figure 3-13: Modified Radio Buttons

This was done because the divisions were unclear using the standard radio button format and this makes the current choice very apparent. The text input validates as the user is typing and where it is possible invalid text will not be rendered using regular expressions and input listeners. If an invalid input is placed, it will automatically remove the input, lowering possible user error. Checks exist for ranges, floating point inputs, and whole number inputs. It is not possible to apply these rules to queue size visible slot range checks until submission where the user will be alerted of an invalid input. This is not possible because range checks cannot be made with multiple possible digits if the first number is greater than 1: visible queue slots cannot be this small and the lower endpoint is 3. Numbers such as 12 or 15 would not be possible if the automatic correction rules were applied.

The inputs cannot just exist as fields or buttons to press. They require listener functions that undergo logic to process the input. For dynamic items the division needs to have its HTML code rewritten, this is demonstratable in the central server model where each disk can have its individual distribution rate adjusted. Inputs will be added varying on the number of disks chosen. When a section is overridden in HTML the on-input functions are removed, causing that element to become unresponsive. To remedy this listener functions are used. Listener functions are called by the configuration group for that specific model upon section update or state change. The old functions are removed, and the proper ones are substituted. An example of this in action would be changing the distribution type:

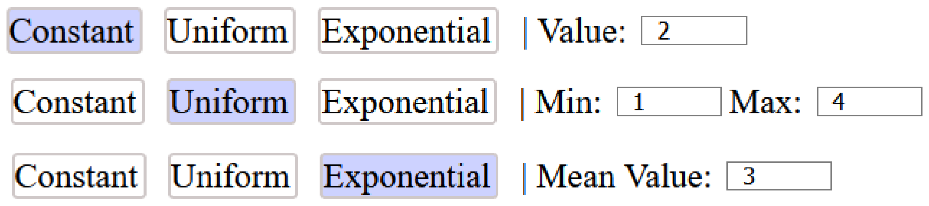
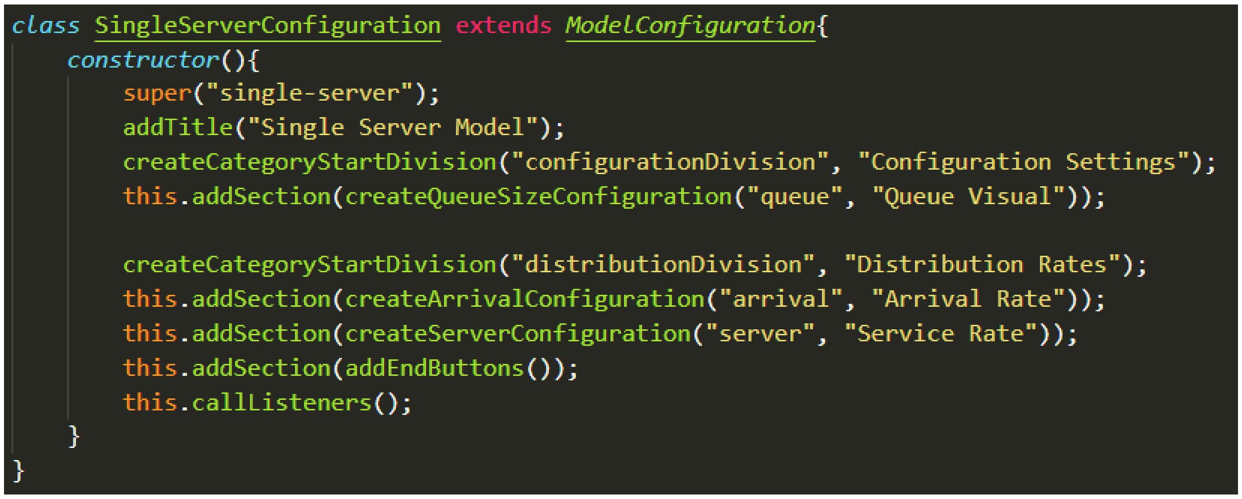

Constant Uniform Exponential | Value: 2
Constant Uniform Exponential | Min: 1 Max: 4
Constant Uniform Exponential | Mean Value: 3

Figure 3-14: Distribution Inputs

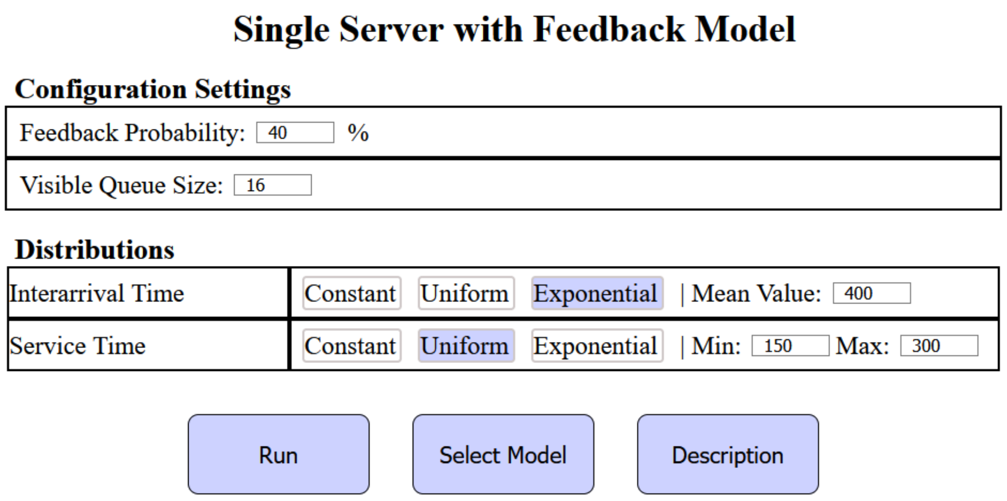
Immediately, on distribution choice, the proper text input fields are created. The distribution group consists of a combination of the radio buttons and dynamically created input fields through these listener functions. These input fields that are created also have their own listener functions to support updating. On input change the change is saved into key value pairs of the parameter and its corresponding value(s). Upon submission these inputs are written into cookies for later retrieval during runtime. Every component can exist independently without concern for the other input sections and added to create a finished configuration:



```
class SingleServerConfiguration extends ModelConfiguration{
    constructor(){
        super("single-server");
        addTitle("Single Server Model");
        createCategoryStartDivision("configurationDivision", "Configuration Settings");
        this.addSection(createQueueSizeConfiguration("queue", "Queue Visual"));

        createCategoryStartDivision("distributionDivision", "Distribution Rates");
        this.addSection(createArrivalConfiguration("arrival", "Arrival Rate"));
        this.addSection(createServerConfiguration("server", "Service Rate"));
        this.addSection(addEndButtons());
        this.callListeners();
    }
}
```

Figure 3-15: Configuration Example

This allows for reusability of sections within various models and automatic updating, validations, and saving of inputs. All components may function alone and to create a configuration for a model the relative sections simply must be added. The base ModelConfiguration class contains functions for calling the listener functions, adding sections, and writing the content to the HTML page. When these elements are placed together a configuration page may be created.


Single Server with Feedback Model

Configuration Settings

Feedback Probability: 40 %

Visible Queue Size: 16

Distributions

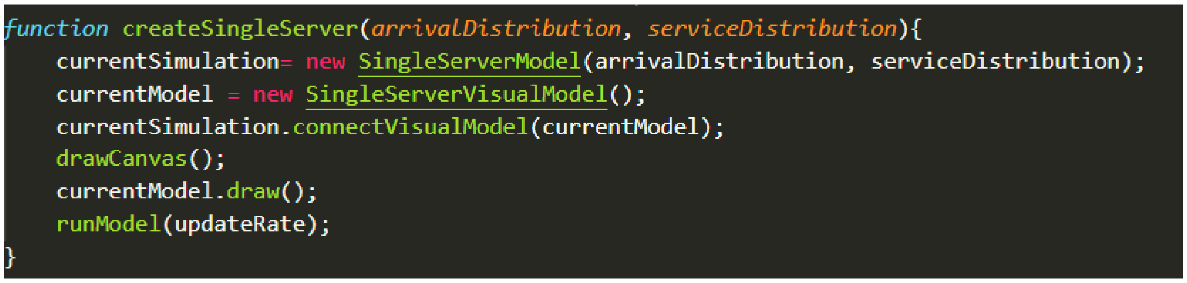
Interarrival Time	Constant Uniform [✓ Exponential] | Mean Value: 400
Service Time	Constant [✓ Uniform] Exponential | Min: 150 Max: 300

Figure 3-16: Complete Configuration Page

The inputs are styled in a way so that the size on the page is standardized for each input and adjustable using variables in the CSS class. To organize and group elements each row can be considered its own independent division that is added. Within the row there are subdivisions unique to the properties of that specific input type. This must be organized in this fashion with divisions within divisions to display the model configurations correctly. Every division is labeled using an identifier matching the specific division and the automatic writing and retrieval of parameters is handled using that identifier or extension of the identifier. Examples of extensions are seen within distribution rates. These contain the identifier followed by what is being described through the input. Names must be unique as cookies are key value pairs and have a one-to-one ratio. If an element is labeled as “server” it will come along with “server-distribution-type” and the specific values for the parameters associated with the chosen distribution such as “server-uniform-lower” or “server-uniform-upper”. The application has no access to user files outside of the web browser so persistent data must be handled according to the rules set forth within that framework. A database could be implemented to circumvent this, but there is no purpose for retrievable data specific to a particular user for this program. After writing this data it can be utilized by the program to create the model simulations and visualizations.

3.4.2 Simulator and Visualizer Interactions

The HTML page loads the scripts for the simulator as well as the visualization. These scripts are linked together through a driver component titled driver.js. This is where the desired model has the visualization and simulation elements linked together and initialized. The objects are simply added and the runModel function is called.

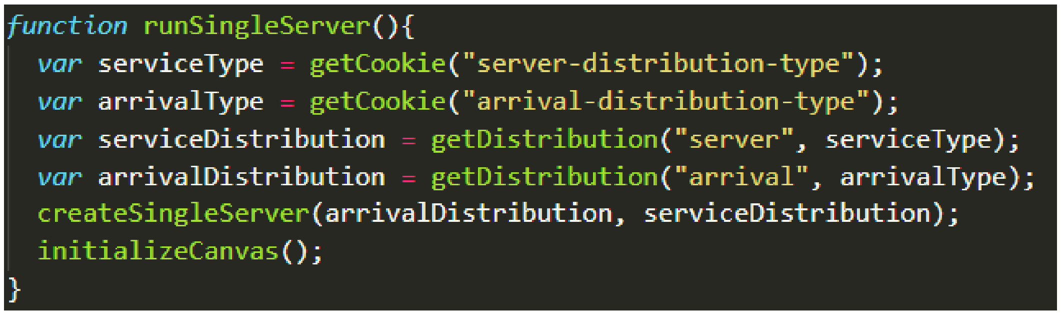


```
function createSingleServer(arrivalDistribution, serviceDistribution){
    currentSimulation= new SingleServerModel(arrivalDistribution, serviceDistribution);
    currentModel = new SingleServerVisualModel();
    currentSimulation.connectVisualModel(currentModel);
    drawCanvas();
    currentModel.draw();
    runModel(updateRate);
}
```

Figure 3-17: Creating the Runtime Environment

There is a creation function for each model and the desired function is called according to the model chosen previously through a switch statement. The driver does more than link the visualization and simulation: it also contains the functions for speed changes and mode switching. These functions were placed within the driver because some affect both the simulation and visualization. When step once mode is run only one event can be processed and the visualization must still run, when maximum speed mode is started the visualization needs to be halted, and when the system is paused the timer must be halted.

The file runTime.js is what maps the driver to the user interface and updates the UI accordingly. When specific buttons are pressed text and functions must be changed and the corresponding methods within the driver need to be called. The configuration settings and cookies are read through the configurationReader.js file. Cookies are read and models are initialized based on the user-choices.



```
function runSingleServer(){
  var serviceType = getCookie("server-distribution-type");
  var arrivalType = getCookie("arrival-distribution-type");
  var serviceDistribution = getDistribution("server", serviceType);
  var arrivalDistribution = getDistribution("arrival", arrivalType);
  createSingleServer(arrivalDistribution, serviceDistribution);
  initializeCanvas();
}
```

Figure 3-18: Configuration Input Retrieval

This merely creates the template for the model and calls the corresponding create method within the driver. It does not do anything except read the configuration options and apply them.

4. Experimental Results

In this section the results of QAS will be compared analytically with those of mathematical models.

Distribution Types:

M = Markov (Poisson, exponential)

D = Deterministic (constant)

GI = General Independent

G = General

Mathematical Representations:

λ = average arrival rate μ = average service rate U = utilization X = throughput

S = average service time A = average interarrival time

4.1 Single Server Model

M/M/1 (exponential arrival rate, exponential service rate, one service mechanism)

Throughput:

$X = \lambda$

Average Utilization:

$S = 1/\mu$

$U = \lambda / \mu = SX \quad | \quad U \in [0,1]$

Response Time:

$R = \frac{S}{1-U}$

If the average arrival rate surpasses the average service rate the number of jobs in the system will accumulate indefinitely and the response time will approach infinity.

Average Number of Jobs Within the System:

$J = \frac{U}{1-U}$

100,000,000 jobs were completed with QAS for comparison. Results were rounded after three decimal places. If the program output 5.999504159 it would be rounded to 6 for the purpose of result comparison.

Three points of testing were completed for M/M/1:

	A	***S***	***U***	***R***	***X***	***Q***
Calculations	*2*	*1*	*0.5*	*2*	*0.5*	*1*
Simulation	*2*	*1*	*0.5*	*2*	*0.5*	*1*
% Difference	*0*	*0*	*0*	*0*	*0*	*0*

	A	**S**	**U**	**R**	**X**	**Q**
Calculation	5	3	0.6	7.5	0.2	1.5
Simulation	5	3	0.6	7.5	0.2	1.5
% Difference	0	0	0	0	0	0

	A	**S**	**U**	**R**	**X**	**Q**
Calculation	12	4	0.333	6	0.083	0.5
Simulation	12	4	0.333	5.999	0.083	0.5
% Difference	0	0	0	0.017	0	0

The results provided through QAS were virtually identical to the calculated expectations for the M/M/1 model. There were slight differences, but only a fraction of a percent. This validates the QAS simulator implementation of the single server model.

4.2 Single Server with Feedback Model

For the feedback loop the single server feeds back into itself. This makes the system load dependent. The jobs do not always leave, but also have the potential of re-entering the system. This is an addition to the arrivals from outside.

The throughput, X of the system can be found by one of two ways. The first is intuitive—for the system to not overflow the output rate must match the input rate, otherwise jobs will accumulate and overflow the queue.

$$X = \lambda$$

We can verify this by finding the average arrival rate as an infinite number of jobs arrive.

P = probability of returning λ = the initial arrival rate

Let λ_n represent the arrival rate of jobs after n jobs have entered the system.

$\lambda_0 = \lambda$

$\lambda_1 = \lambda + PX_0 = \lambda(1 + P)$

$\lambda_2 = \lambda + PX_1 = \lambda + P\lambda(1 + P) = \lambda(1 + P + P^2)$

$\lambda_3 = \lambda + PX_2 = \lambda + P(\lambda(1 + P + P^2) = \lambda(1 + P + P^2 + P^3)$

$\lambda_n = \lambda(1 + P + P^2 + P^3 + \ldots P^n)$

$P\lambda_n = \lambda(P + P^2 + P^3 + \ldots P^{n+1})$

$\lambda_n - P\lambda_n = \lambda_n(1 - P) = \lambda(1 + P + P^2 + P^3 + \ldots P^n) - \lambda(P + P^2 + P^3 + \ldots P^{n+1}) = \lambda(1 - P^{n+1})$

$\lambda_n(1-P) = \lambda(1 - P^{n+1})$

$\lambda_n = \frac{\lambda(1 - P^{n+1})}{1-P}$

As n approaches infinity P^{n+1} approaches 0 where $0 < P < 1$

$$\therefore \lambda_n \text{ converges to } \frac{\lambda}{1-P}$$

If we use our solution $\frac{\lambda}{1-P}$ for the average of total incoming traffic we can find X through

$$X = \frac{\lambda}{1-P} \cdot (1 - P) = \lambda$$

This confirms our earlier assessment that the arrival rate must match the departure rate to form a steady system. It is not possible to process jobs faster than they arrive, and if the throughput is less than the arrival rate the system will fall behind and continuously accumulate more jobs.

Utilization:

Let λ_F = the average arrival rate into the queue including feedback.

$$U = \frac{\lambda_F}{\mu}$$

Response Time:

For a single server

$$R = \frac{S}{1-U}$$

We must also factor in that there is a probability of leaving the system for the feedback model.

$$R = \frac{S}{1-U} \cdot \frac{1}{1-P}$$

Average Number of Jobs Within the System:

$$J = \frac{U}{1-U}$$

An interesting observation may be made. The throughput remains the same as the single server model even though there is a chance of feeding back into the system, but the response times for jobs and utilization are not the same.

The non-inclusive upper bound for interarrival rate from outside of the system may be solved for by finding where utilization reaches one.

$$U = \frac{\lambda_F}{\mu} = 1$$

Utilizing the equations derived above an analytical comparison may be created to QAS. Over 10,000,000 jobs were completed for each comparison.

A = average interarrival time	S = average service time	P = feedback probability
U = average utilization	R = average response time	X = throughput
Q = average queue length		

The feedback probabilities were chosen between 0.1 up to 0.4 for the values A = 2, S =2. After the utilization approaches one the system is no longer stable and response time queue size will begin to approach infinity. Exponential distribution types were used for the service time as well as the interarrival rate.

	A	S	P	U	R	X	Q
Calculations	2	1	0.1	0.555	2.5	0.5	1.25
Simulation	2.002	1	0.1	0.555	2.495	0.5	1.245
% Difference	0.1	0	0	0	0.2	0	0.4

	A	S	P	U	R	X	Q
Calculations	2	1	0.2	0.625	3.333	0.5	1.666
Simulation	2	1	0.2	0.624	3.322	0.5	1.66
% Difference	0	0	0	0.16	0.330	0	0.360

	A	S	P	U	R	X	Q
Calculations	2	1	0.3	0.714	5	0.5	2.5
Simulation	2	1.001	0.3	0.717	4.965	0.5	2.482
% Difference	0	0.1	0	0.42017	0.7	0	0.72

	A	S	P	U	R	X	Q
Calculations	2	1	0.4	0.833333	10	0.5	5
Simulation	2	1	0.4	0.833	9.978	0.5	4.986
% Difference	0	0	0	0.04	0.22	0	0.28

Once again, the percentage difference is extremely low in all cases between QAS and the analytical models. The results were graphed in order to better demonstrate the impact of feedback probability on the model behaviors.

Graphs for Experiment One

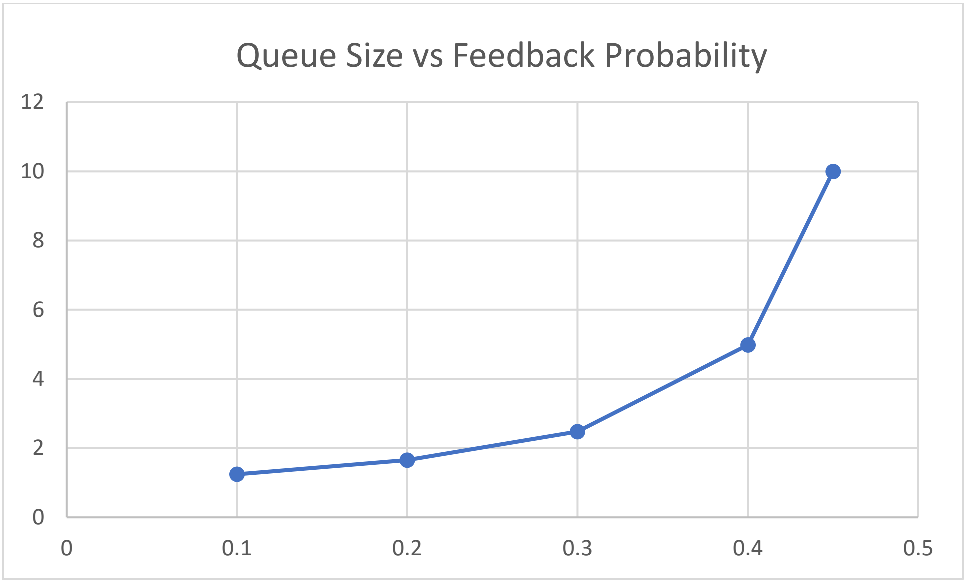


Figure 4-1: Experiment One, Q vs Feedback Probability

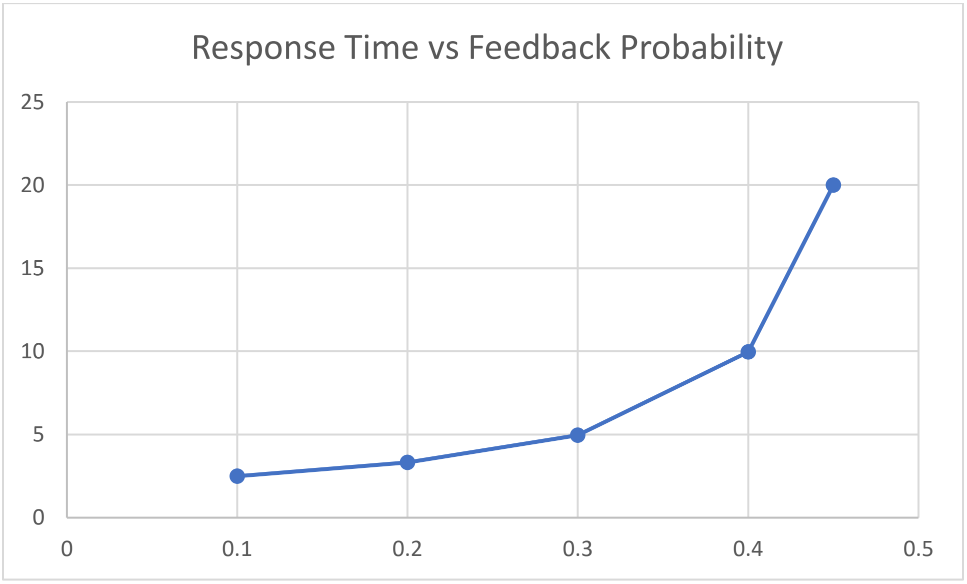


Figure 4-2: Experiment One, R vs Feedback Probability

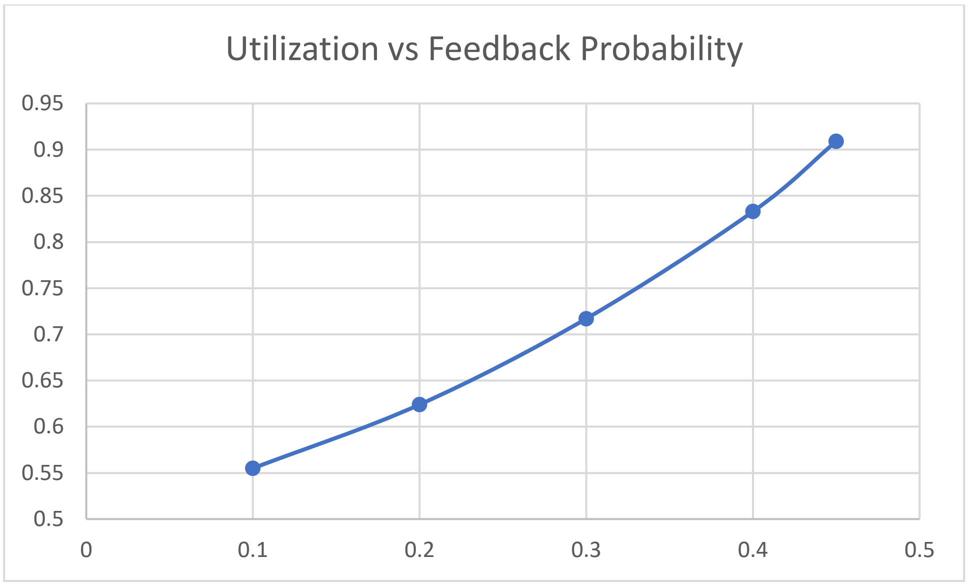


Figure 4-3: Experiment One, U vs Feedback Probability

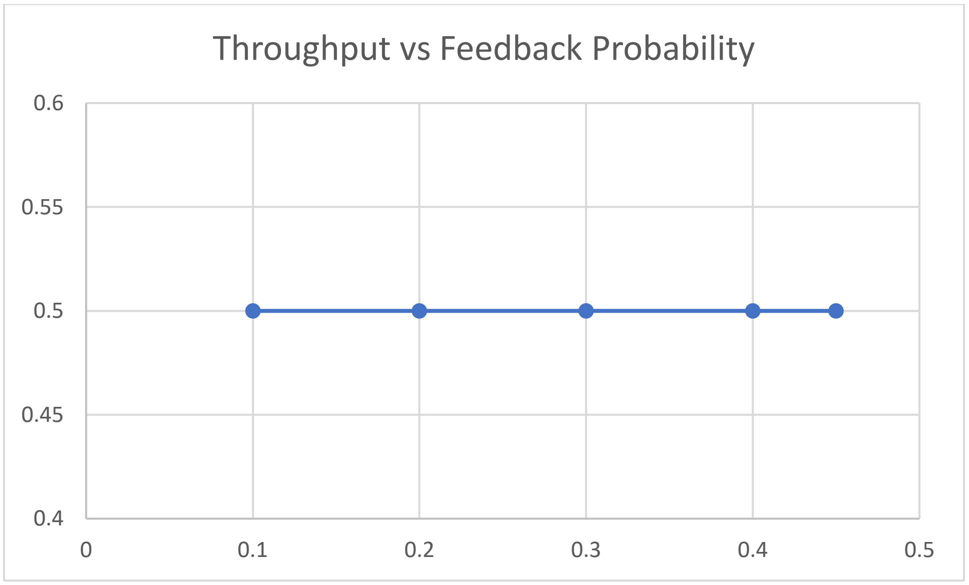


Figure 4-4: Experiment One, X vs Feedback Probability

From these results, what was discussed before is easily seen. The throughput is constant, despite the average response time, queue size, and utilization increasing alongside the feedback probability.

A second set of interarrival and service rates was chosen for verification. This time a larger disparity between service rate and interarrival rate was chosen to further demonstrate the behavior of feedback models. 10% increments in feedback probability were done from 10% to 70% and one at 75%.

	A	S	P	U	R	X	Q
Calculations	9	2	0.1	0.246914	2.95082	0.111111	0.327869
Simulation	8.998	2.001	0.1	0.247	2.952	0.111	0.328
% Difference	0.022	0.05	0	0.035	0.04	0.1	0.04

	A	S	P	U	R	X	Q
Calculations	9	2	0.2	0.277778	3.461538	0.111111	0.384615
Simulation	8.998	1.999	0.2	0.278	3.46	0.111	0.384
% Difference	0.022	0.05	0	0.08	0.044	0.1	0.16

	A	S	P	U	R	X	Q
Calculations	9	2	0.3	0.317	4.186	0.111	0.465
Simulation	9.009	2	0.3	0.317	4.186	0.111	0.465
% Difference	0.1	0	0	0.145	0.001	0	0.025

	A	S	P	U	R	X	Q
Calculations	9	2	0.4	0.370	5.294	0.111	0.588
Simulation	8.994	2	0.4	0.371	5.297	0.111	0.589
% Difference	0.066	0	0	0.17	0.05444	0.1	0.13

	A	S	P	U	R	X	Q
Calculations	9	2	0.5	0.444	7.2	0.111	0.8
Simulation	8.986	2	0.5	0.444	7.188	0.111	0.8
% Difference	0.156	0	0	0	0.166	0.1	0

	A	S	P	U	R	X	Q
Calculations	9	2	0.6	0.555556	11.25	0.111111	1.25
Simulation	8.994	2.003	0.6	0.557	11.272	0.111	1.253
% Difference	0.067	0.15	0	0.26	0.196	0.1	0.24

	A	S	P	U	R	X	Q
Calculations	9	2	0.7	0.740	25.714	0.111	2.857
Simulation	8.992	2	0.7	0.741	25.731	0.111	2.862
% Difference	0.089	0	0	0.035	0.065	0.1	0.17

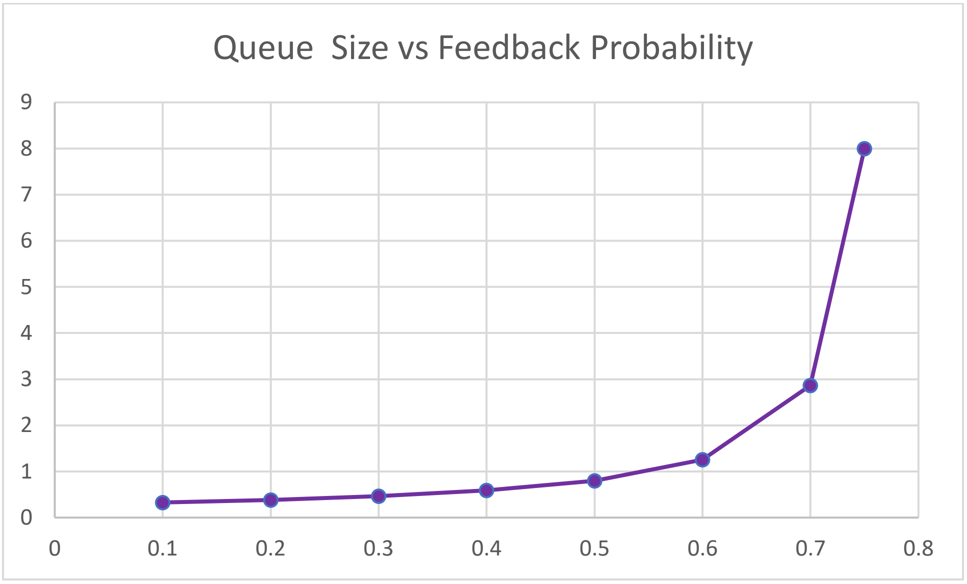


Figure 4-5: Experiment Two, Q vs Feedback Probability

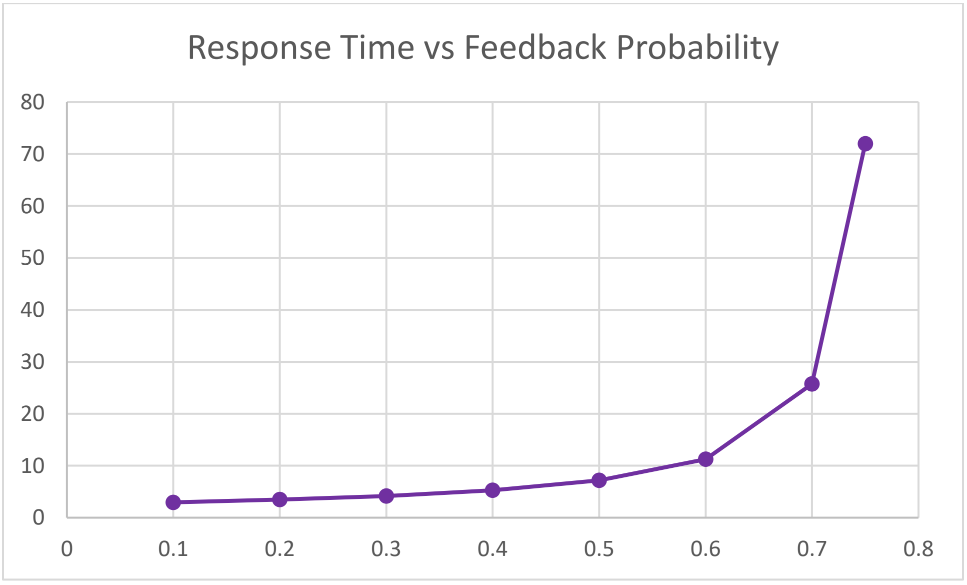


Figure 4-6: Experiment Two, R vs Feedback Probability

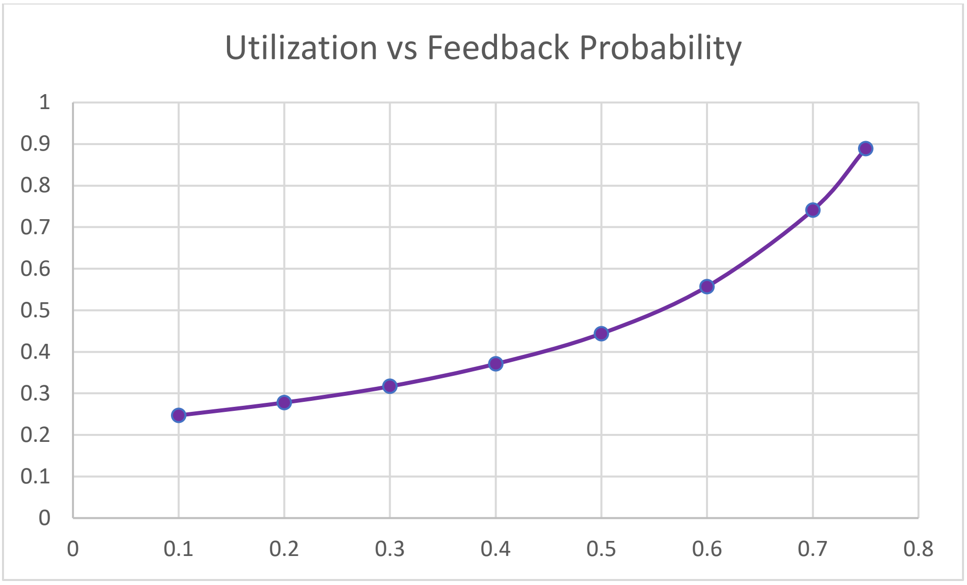


Figure 4-7: Utilization vs Feedback Probability

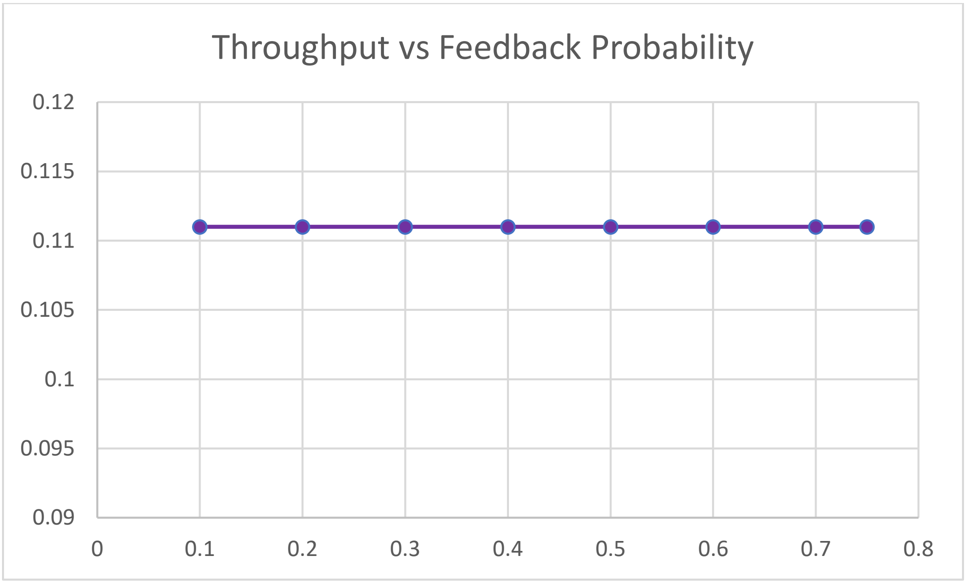


Figure 4-8: Throughput vs Feedback Probability

Despite the changes in the service and arrival distributions the relationship between feedback probability and the average queue size, response time, utilization, and throughput remain the same. This is consistent with what is anticipated. As the service and arrival rates vary the maximum feedback probability will change alongside it, leaving asymptotes at different points. The core trends created by approaching this maximum will remain the same.

4.3 Parallel Servers Model

The parallel servers model contains one queue with multiple servers in parallel. When a job arrives, it is assigned randomly to an available server if present.

Let k = number of servers in parallel

$U_n = \frac{P_n \lambda}{\mu}$, where P_n is the probability of being assigned server n.

In this case, each parallel server is equally likely to receive a job, therefore $P_1 = P_2 = P_3 \ldots = P_k$

$$\sum_{n=1}^{k} P_n = 1 \rightarrow P_n = \frac{1}{k}$$

Utilization:

$$U_n = \frac{P_n \lambda}{\mu} = \frac{\lambda}{k\mu}$$

$$U_1 = SX = \frac{\lambda}{k\mu}$$

$$U_k = U_1 / k$$

Response Time:

$$R_k \approx S\left[1 + \frac{U_1^k}{k!k(1-U_k)^2 \left(1 + U_1 + \frac{U_1^{k-1}}{(k-1)!} + \frac{U_1^{k-1}}{k!(1-U_k)}\right)}\right] = S + W_k$$

The response time calculation is an approximation, but it will be compared to QAS at treated as the basis for comparison.

These parameters were utilized for the analysis of parallel systems.

k = 2..7 A = 15 S = 20

k = 1 was not possible with these service times and interarrival times as one server would not be capable of handling that load.

k	U	R	X
2	0.667	36.13	0.067
3	0.444	22.151	0.067
4	0.333	20.37	0.067
5	0.267	20.063	0.067
6	0.222	20.012	0.067
7	0.191	20.007	0.067

QAS – Results

k	U	R	X
2	0.667	40.425	0.067
3	0.444	24.690	0.067
4	0.333	21.227	0.067
5	0.267	20.637	0.067
6	0.222	20.202	0.067
7	0.190	20.102	0.067

Mathematical Results

k	% Difference U	% Difference R	% Difference X
2	0	10.626	0
3	0	10.284	0
4	0	4.035	0
5	0	2.783	0
6	0	0.939	0
7	0.05	0.471	0

Parallel Server Comparison

The utilization amounts and throughput were the same, but the response time differed by up to 10%. This is likely due to an approximation being utilized in the calculations of response time, but a 10% difference is still sizable. The QAS program had the same utilizations for the given number of service mechanisms in parallel and the throughput was equivalent to the calculated throughput. With some degree of confidence QAS has successfully modeled this parallel system.

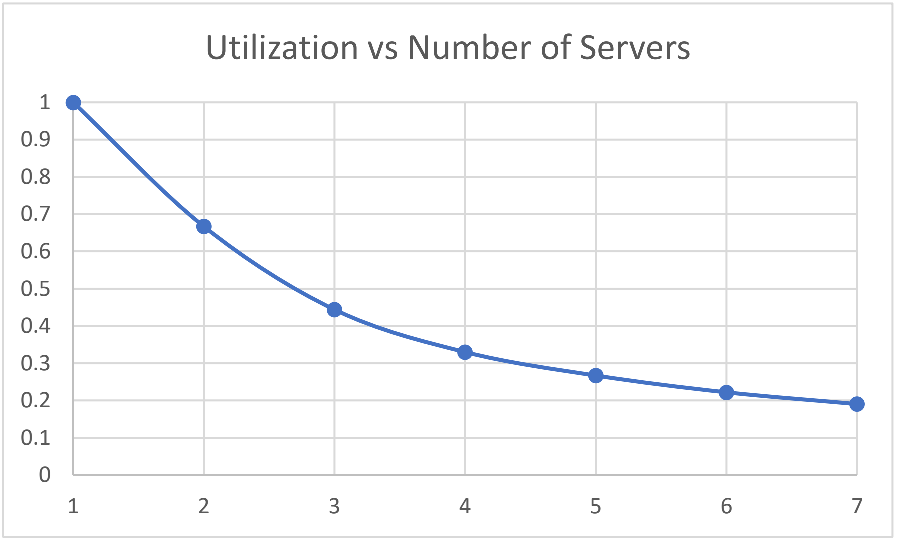


Figure 4-9: Parallel Servers U vs k

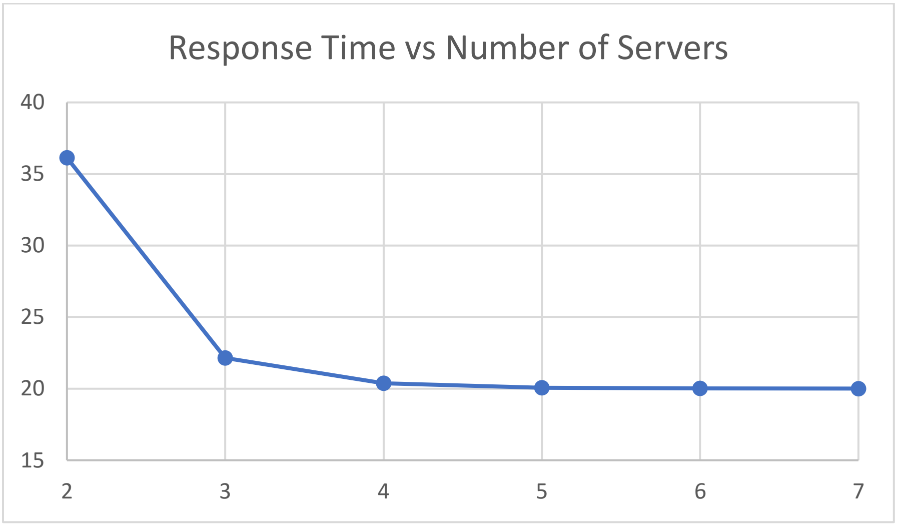


Figure 4-10: Parallel Servers R vs k

The utilization goes down proportionally to the increase in service mechanisms. The response time does the same, rapidly approaching a lower limit: the service time average from the service mechanisms. The average response time cannot possibly be less than the average service time. When sufficient servers are added that every arrival be immediately be assigned to a server the response time becomes incredibly close to the service time. The benefit of adding additional servers beyond this is minimal regarding the average response time.

4.4 Interactive Workstations Model

n = number of workstations Z = Think time

$X = \frac{n}{Z+R} = \frac{U_P}{S}$ $U_p = \frac{nS}{Z+R}$ $R = \frac{nS}{U_P} - Z$

For the analysis of QAS a model with six workstations and two processors will be shown. We construct a flow diagram in order to find the probabilities of each state. P_0 represents both processors being idle, all workstations full. P_1 represents a single processor idle, 5 workstations full. This continues until the flow of the system is mapped out. The flow diagram and the visual representation are shown below.

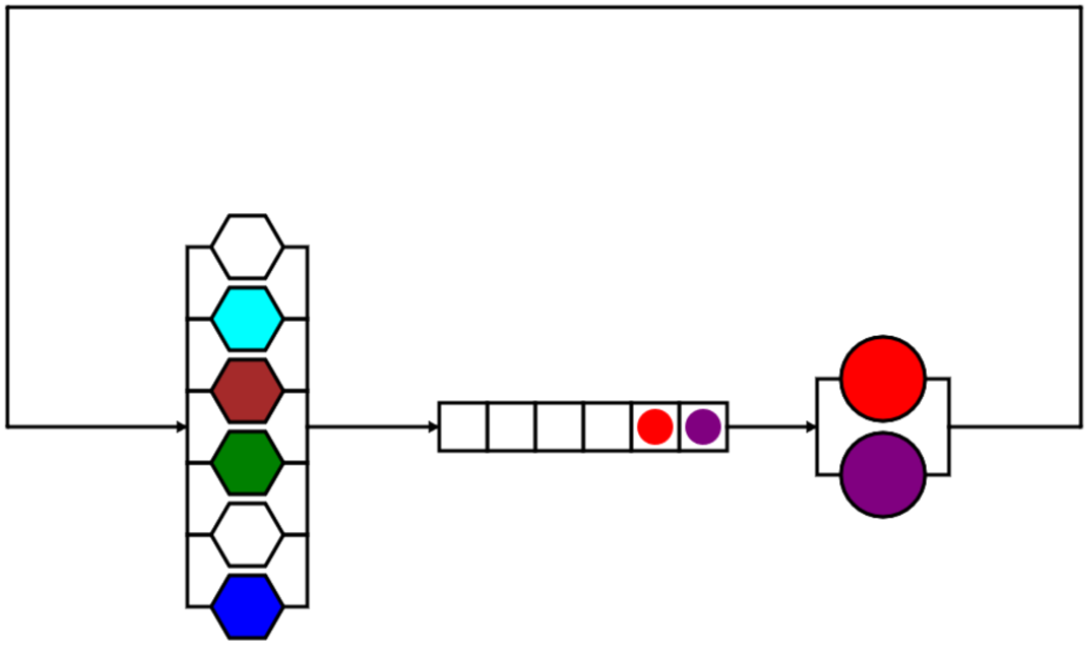


Figure 4-11: Interactive Servers Example

$$\begin{tikzcd} P_0 \arrow[r, bend left, "6\lambda"] & P_1 \arrow[l, bend left, "\mu"] \arrow[r, bend left, "5\lambda"] & P_2 \arrow[l, bend left, "2\mu"] \arrow[r, bend left, "4\lambda"] & P_3 \arrow[l, bend left, "2\mu"] \arrow[r, bend left, "3\lambda"] & P_4 \arrow[l, bend left, "2\mu"] \arrow[r, bend left, "2\lambda"] & P_5 \arrow[l, bend left, "2\mu"] \arrow[r, bend left, "1\lambda"] & P_6 \arrow[l, bend left, "2\mu"] \end{tikzcd}$$

Figure 4-12: Interactive Flow Diagram

$P_1 = 6\rho\ P_0$

$P_2 = \frac{5}{2}\rho\ P_1 = 15\rho^2 P_0$

$P_3 = \frac{4}{2}\rho\ P_2 = 30\rho^3 P_0$

$P_4 = \frac{3}{2}\rho\ P_3 = 45\rho^4 P_0$

$P_5 = \frac{2}{2}\rho\ P_4 = 45\rho^5 P_0$

$P_6 = \frac{1}{2}\rho\ P_5 = \frac{45}{2}\rho^6 P_0$

Sum of all probabilities must add up to one.

$1/P_0 = 6\rho + 15\rho^2 + 30\rho^3 + 45\rho^4 + 45\rho^5 + \frac{45}{2}\rho^6$

$\rho = \lambda / \mu$

$U = 1 - P_0 - (P_1 / 2)$

With:

$S = 3 \quad Z = 8$

$\rho = \lambda/\mu = 3/8$

$P_0 = 0.375 \quad P_1 = 0.273$

$U = 0.742 \quad R = 4.134 \quad X = 0.495$

Compared to QAS these are the results:

	U	**R**	**X**
Calculated	0.742	4.134	0.495
QAS	0.742	4.13	0.495
% Difference	0	0.096759	0

There is very little difference between the calculated solution and the QAS run.
It should be noted that QAS currently shows the total response time including the thinking time. If the thinking time is subtracted from this than it would give what is shown above. An update will be added to show the server response time separately alongside the entire lifeline of a job from the point when the user begins.

4.5 Central Server Model

This model is a closed system. The limiting factor of this system will be equivalent to the bottlenecking section. The rate of job flow cannot exceed the slowest element. First this system will be discussed at full saturation, where the bottleneck is constantly in use. Afterwards it will be explained how to calculate system metrics with n number of jobs within the system.

The first results will be found using these values:

Cores: 4	Distribution: Exponential, mean = 6
Disks: 2	Distribution: Exponential, mean = 2
Channel	Distribution: Exponential, mean = 1

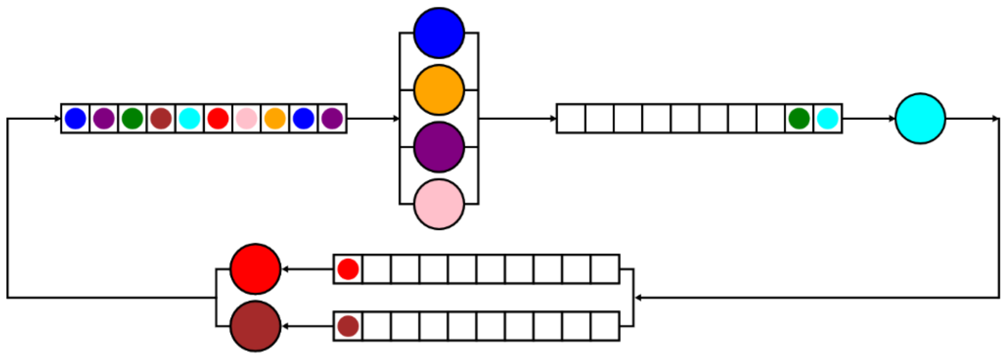


Figure 4-13: Central Server Experiment 1

We find the maximum flows out of the sections.

CPU section:

Each core can output one job in 6 units of time. This is 1/6 jobs per unit.

4 * 1/6 = 4/6 = 2/3 jobs per unit of time

Disks:

2 * ½ = 1 job per unit of time

Channel:

1 job per unit

Here the maximum possible throughput is 2/3 jobs per unit.

$X_{max} = 2/3 \approx 0.667$

Every section can be thought of as connected in series; each section will have $\lambda = X$ at the beginning and end of the section. The utilization of the bottleneck will be expected to be one, as this is the limited factor.

CPU:

Subscript p refers to the processor cores.

$$U_{p1} = U_{p2} = U_{p3} = U_{p4}$$

There is an equivalent probability of going into each core. The average demand or incoming flow into each core will be the same.

$$D_{p1} = D_{p2} = D_{p3} = D_{p4} = \lambda /4$$

The utilization for each core:

$$U = \frac{\lambda}{\mu}$$

$$U_{p1} = \frac{\lambda}{4\mu} = \frac{2/3}{4 \cdot 1/6} = 1 = U_{p2} = U_{p3} = U_{p4}$$

This makes sense because the CPU is the bottleneck in this scenario.

Channel:

$$U = \lambda / \mu = SX = 1 \cdot 2/3 = 2/3$$

Disks:

$$U = \frac{\lambda}{2\mu} = \frac{2/3}{1} = 2/3$$

These results are representative of the server at maximum capacity. If it is over capacity, the system will still respond, but the throughput will not increase from the X_{max}. The utilization of the bottleneck will remain one, but the response times and queue lengths will increase proportional to the number of jobs.

These are in fact the same results we obtain from QAS simulation when the system is at full capacity.

UTILIZATIONS AT MAXIMUM CAPACITY

	Core 1	Core 2	Core 3	Core 4	Channel	Disk 1	Disk 2
Calculations	1	1	1	1	0.667	0.667	0.667
Simulation	1	1	1	1	0.667	0.667	0.667
% Difference	0	0	0	0	0	0	0

The maximum throughput is X_{max} = 2/3 Jobs per unit of time.

Mean Value Analysis:

Now what would happen if we had smaller number of jobs in the system? The throughput will not be the same. The utilization of the bottleneck is not constantly one. There are potential situations where the bottleneck will always be used completely, but the response time will become greater and greater as more jobs are added.

What if the system is not at capacity? The problem must be approached in a different way.

A process known as mean value analysis may be used to approximate the state of our system. This is an iterative method that can find the state of the system with N number of jobs.

M = total number of components within the system

$D_m = S_m \cdot V_m$ -- demand of a single job

R_m = the response time for the component

$X = \frac{n}{\sum_{m=1}^{M} r_m}$ $U_m = X{\cdot}D_m$

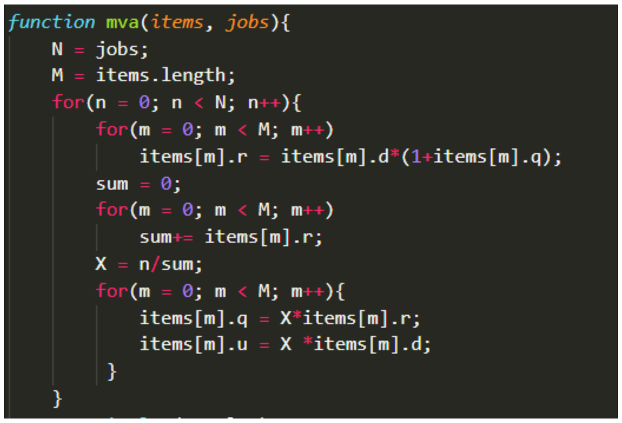
N = maximum number of jobs within the system

V_m = visits for item m

$r_m = R_m \cdot V_m$ -- residence time for component m

The residence time can be less than the component m service time. This is possible because components within the system can have a visit amount less than 1. This is experienced in parallel or branching systems. If this were an open model, the visit ratio of the outside world would be set as the baseline of comparison. If the relative proportion of the visits remains the same the utilization will not change, but the response time and throughput will differ. If a job requires multiple visits to one component that is acceptable, a disk may have to be read from many times before completion. If a disk had to be read 8 times for each incoming job it would have a visit value of 8.

It is a rather short implementation for the algorithm, it is shown below. Items is simply an array of components within the system that have been assigned the visits and service time.



```
function mva(items, jobs){
    N = jobs;
    M = items.length;
    for(n = 0; n < N; n++){
        for(m = 0; m < M; m++)
            items[m].r = items[m].d*(1+items[m].q);
        sum = 0;
        for(m = 0; m < M; m++)
            sum+= items[m].r;
        X = n/sum;
        for(m = 0; m < M; m++){
            items[m].q = X*items[m].r;
            items[m].u = X *items[m].d;
        }
    }
```

Figure 4-14: MVA Program

The MVA evaluation will be undergone for a sample model and compared with experimental results from QAS.

The model has a single core CPU, two disks, and a channel with the following service times:

$S_{cpu} = 3$ $S_{channel} = 2$ $S_{disk1} = 5$ $S_{disk2} = 5$

Each disk has equal likelihood to be accessed. That is $V_{d1} = V_{d2}$.
The visit amount represented by this system:

$V_{cpu} = 1$ $V_{channel} = 1$ $V_{disk1} = ½$ $V_{disk2} = ½$

These visit amounts mean that for every job the CPU should be accessed once, the channel once, and one of the two disks should be accessed.

The experimental results are compared between MVA (left) and QAS (right) up to nine jobs.

n	CPU	channel	disk 1	disk 1	X
1	0.300	0.200	0.250	0.250	0.100
2	0.478	0.319	0.398	0.398	0.159
3	0.595	0.397	0.496	0.496	0.198
4	0.678	0.452	0.565	0.565	0.226
5	0.738	0.492	0.615	0.615	0.246
6	0.785	0.523	0.654	0.654	0.262
7	0.821	0.547	0.684	0.684	0.274
8	0.850	0.566	0.708	0.708	0.283
9	0.873	0.582	0.727	0.727	0.291

Figure 4-15: MVA Results

n	CPU	Channel	Disk 1	Disk 2
1	0.3	0.2	0.25	0.25
2	0.479	0.319	0.398	0.398
3	0.595	0.398	0.495	0.497
4	0.677	0.452	0.565	0.563
5	0.739	0.492	0.616	0.615
6	0.785	0.524	0.654	0.654
7	0.821	0.547	0.684	0.684
8	0.85	0.566	0.707	0.707
9	0.871	0.581	0.729	0.727

Figure 4-16: QAS Central Results

The differences for utilizations were calculated as a percentage.

n	CPU	Channel	Disk 1	Disk 2
1	0	0	0	0
2	0.209	0	0	0
3	0	0.252	0.202	0.202
4	0.147	0	0	0.354
5	0.136	0	0.166	0
6	0	0.191	0	0
7	0	0	0	0
8	0	0	0.141	0.141
9	0.229	0.172	0.275	0

Figure 4-17: Central Server Comparison

The experimental results obtained from QNAS match very well with the results generated from the MVA algorithm. The relationship between the number of jobs within the closed central server system and the utilization of the CPU, channel, and disks is demonstrated below.

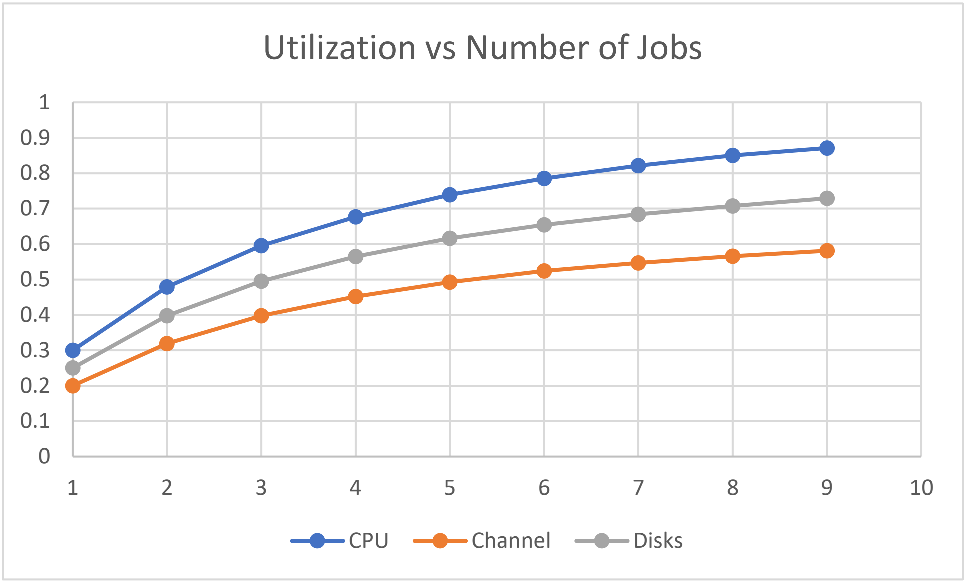


Figure 4-18: Central Server, Utilization vs Jobs

This clearly demonstrates that the utilization increases as the number of jobs within the system increases. The utilization will approach one. When this occurs the throughput of the system will become limited by the bottleneck (the CPU in this example).

5. Conclusions and Future Work

5.1 Conclusions

QAS is an effective tool regarding simulation and will provide the generation of accurate results. It allows for easy experimentation and a quick set up of configurations. Additionally, QAS provides visual representations of the available systems. The flow and state of the system can be observed, and relationships understood.   Yet there is room for expansion and improvements. New models may be created, and the visualization may can be further refined.

5.2 Extensions

This program could be expanded upon to add the capability of the graphing of results and job flows between components. This is of particular interest for the feedback model to help demonstrate the disruption of Poison flow brought forth by the feedback loop. A potential to graph other data within models can assist in the explanation of trends, patterns, and grant a better understanding of the presented models in general.

It is possible for new models to be added with relative ease due to components being easily added and linked. It would be interesting to have more branching systems, where one of many potential routes can be taken.

Researchers may utilize the simulator to assist in the development novel mathematical formulae relating to queueing systems. There are distribution types and systems that lack analytical models. The dynamic nature of the simulator would also allow for adjustment or expansion according to the researchers' needs.

Dynamic assessments may be added for the purposes of examination or self-assessment. An example of this could be asking the user what the utilization is expected to be for a given a model with program generated parameters. The student could predict what the result is and then observe QAS and receive feedback based on the correctness of their response. This would add an additional layer of interactivity, encourage engagement, and could even store results within a remote database for academic feedback. The professor would be able to see where students are struggling and what concepts need further elaboration if any and this feature would be a fantastic addition.

6. References

BRAT87	Paul Bratley. *A Guide to Simulation*, Springer Verlag, 1987
BV18	Blackman, David. Vigna, Sebastiano. *Scrambled Linear Pseudorandom Number Generators,* 2018
CIBO92	Ciborowski, J. *Textbooks and the Students Who Cannot Read Them: A Guide to Teaching Content*. Brookline Books, 1992
DOUG15	Crockford, Douglas. *JavaScript: the Good Parts*. OReilly, 2015.
DUJM08	Dujmović, Jozo. *Computer Performance Evaluation,* SFSU, 2008.
LF15	Lewis, John. Loftus, William. *Java Software Solutions: Foundations of Program Design, Eighth Edition, 2015*
LIU03	Liu, Weili. *RAND-Random Number Generation and Testing***.** *San Francisco State University, 2013*
MARS03	Marsaglia, George. *XorshiftRNGs*, The Florida State University, 2003
MDN HTML 21	*MDN Web Docs. HTML: HyperText Markup Language.* Mozilla, 2021
MDN CSS 21	*MDN Web Docs.CSS: Cascading Style Sheets.* Mozilla, 2021
MDN JS 21	*MDN Web Docs. JavaScript.* Mozilla, 2021
MEYE01	Meyers, Robert. Encyclopedia of Physical Science and Technology, Third Edition, 2001
MITRA82	I. Mitrani. Simulation Techniques for Discrete Event Systems (Cambridge Computer Science Text, 14), University Press, Cambridge, 1982
ORAC20	Oracle. *Java Client Roadmap Update: An update of timelines for Java Deployment and Java UI technologies*. Oracle, 2020

PAN98 Pan, Fang. Dujmović, Jozo. *A Java System For Simulation and Visualization of Queueing Networks*. San Francisco State University, 1998

SANK05 Sankar, Hemamalini. Dujmović, Jozo. *QNS – An Online System For The Study of Queueing Models*. San Francisco State University, 2005

VIGN14 Vigna, Sebastiano. *An Experimental Exploration of Marsaglia's xorshift Generators, Scrambled*, Universit`a degli Studi di Milano, Italy, 2014

W3C21 World Wide Web Consortium. *Cascading Style Sheets home page,* 2021

7. Appendices

A. Program Requirements

The requirements to run this program are to have a computer with a modern browser. QAS may be accessed from anywhere online and does not require the user to install additional tools. The data for configuration is stored as cookies, so cookies must be enabled. QAS is meant for access via computer and was not created with mobile scaling in mind. Some elements were made a specific size to ensure consistency and will not automatically scale.

B. Program Features and Usage

QAS is a simulation and visualization tool for queueing theory. It can run various models and relaying accurate results to the user while providing visualizations of the model. When starting QAS the user is greeted with the home or index page.

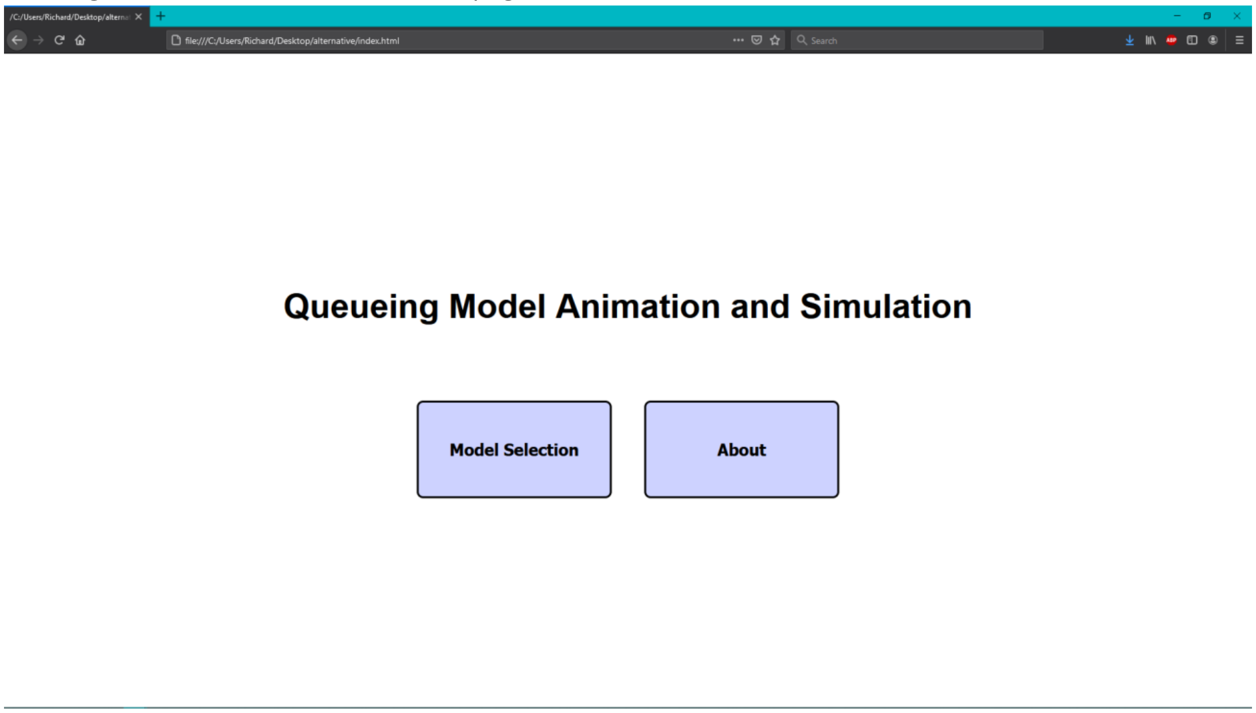
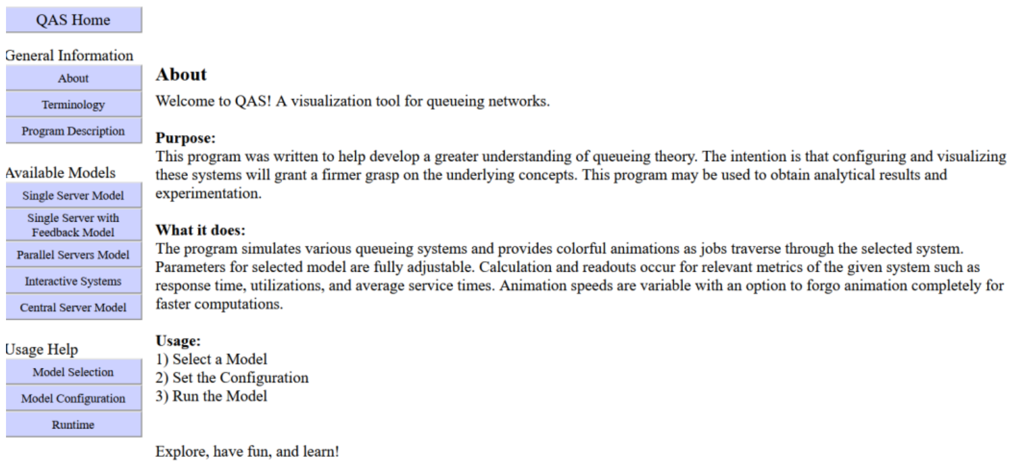

file:///C:/Users/Richard/Desktop/alternative/index.html
Queueing Model Animation and Simulation
Model Selection
About

Figure 7-5: Home Page

From here there are two options, “Model Selection” and “About”. Model selection will allow the user to select one of the available models and begin their journey in this interactive system. The about button will redirect the user to a help page with descriptions and usage instructions.

Lets explore the about page first.



About

Welcome to QAS! A visualization tool for queueing networks.

Purpose:
This program was written to help develop a greater understanding of queueing theory. The intention is that configuring and visualizing these systems will grant a firmer grasp on the underlying concepts. This program may be used to obtain analytical results and experimentation.

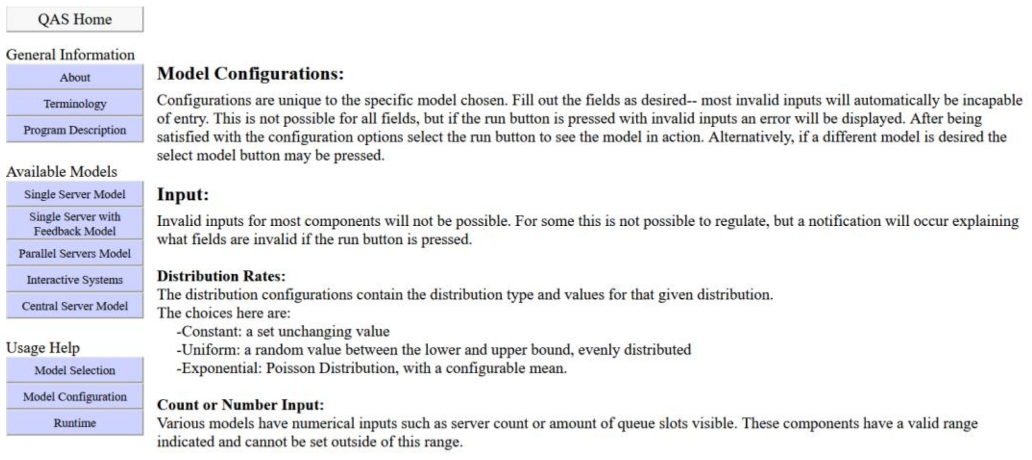
What it does:
The program simulates various queueing systems and provides colorful animations as jobs traverse through the selected system. Parameters for selected model are fully adjustable. Calculation and readouts occur for relevant metrics of the given system such as response time, utilizations, and average service times. Animation speeds are variable with an option to forgo animation completely for faster computations.

Usage:
1) Select a Model
2) Set the Configuration
3) Run the Model

Explore, have fun, and learn!

Figure 7 -6: About Page

The about page has options on the left, that when pressed will render information regarding that topic. Allow an example to be shown: the "Runtime" button under Usage Help is pressed to learn more about the features present while the simulation is running.



Model Configurations:

Configurations are unique to the specific model chosen. Fill out the fields as desired-- most invalid inputs will automatically be incapable of entry. This is not possible for all fields, but if the run button is pressed with invalid inputs an error will be displayed. After being satisfied with the configuration options select the run button to see the model in action. Alternatively, if a different model is desired the select model button may be pressed.

Input:

Invalid inputs for most components will not be possible. For some this is not possible to regulate, but a notification will occur explaining what fields are invalid if the run button is pressed.

Distribution Rates:
The distribution configurations contain the distribution type and values for that given distribution.
The choices here are:
-Constant: a set unchanging value
-Uniform: a random value between the lower and upper bound, evenly distributed
-Exponential: Poisson Distribution, with a configurable mean.

Count or Number Input:
Various models have numerical inputs such as server count or amount of queue slots visible. These components have a valid range indicated and cannot be set outside of this range.

Figure 7-7: About Page Navigation

To return to the index page at any moment click the QAS in the top left corner.

The model selection page displays the five available models.

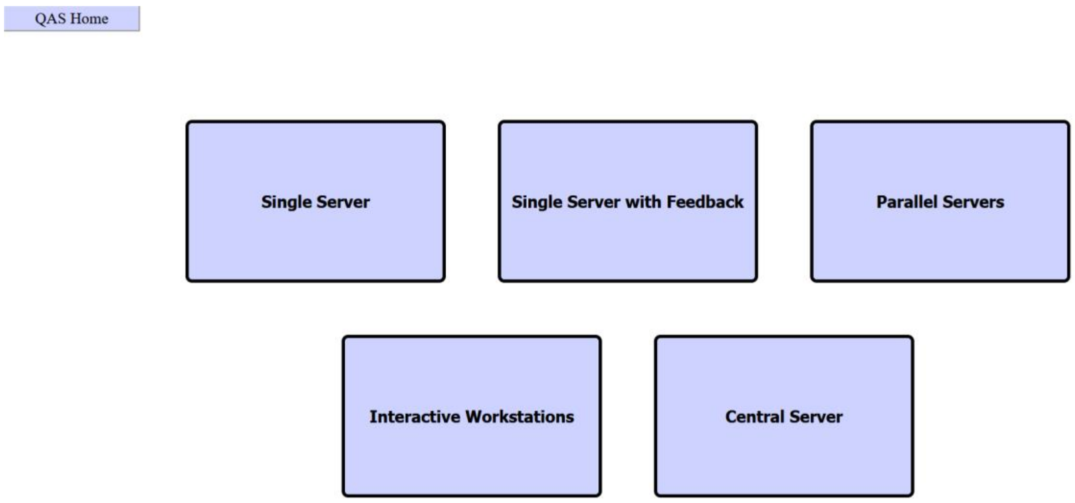
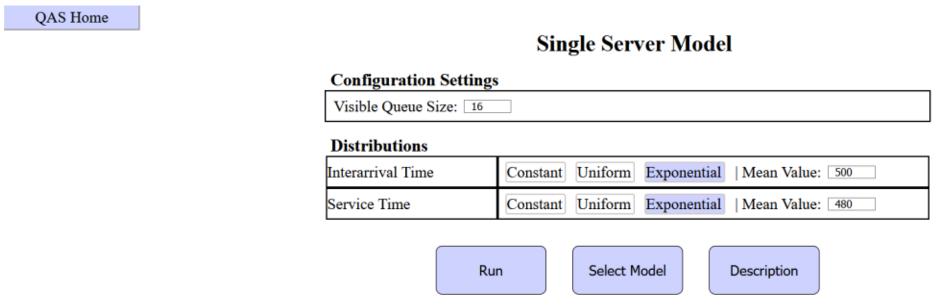


Figure 7 - 8: Model Selection Page

When a model is selected a configuration page will be displayed pertaining to that model.



Single Server Model

Configuration Settings

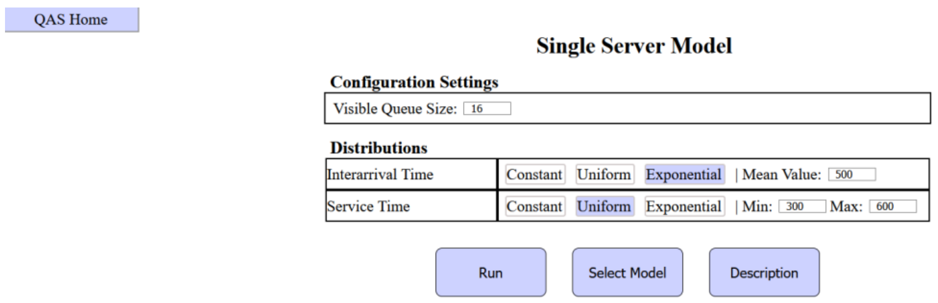
Visible Queue Size: 16

Distributions

Interarrival Time	Constant Uniform [✓ Exponential]	Mean Value: 500
Service Time	Constant Uniform [✓ Exponential]	Mean Value: 480

Figure 7 - 9: Configuration Page

These fields may be filled out according to the user's needs. If a different distribution type is desired click one of the non-highlighted types.


QAS Home

Single Server Model

Configuration Settings

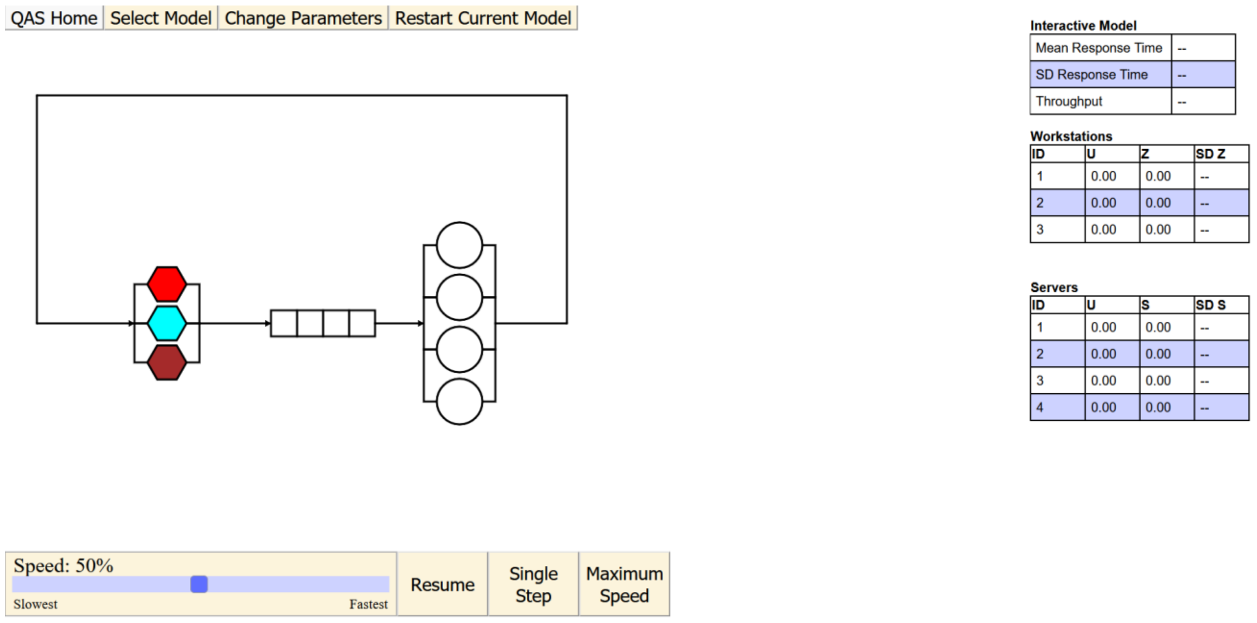
Visible Queue Size:	16

Distributions

Interarrival Time	Constant Uniform Exponential | Mean Value: 500
Service Time	Constant Uniform Exponential | Min: 300 Max: 600

Figure 7 - 10: Modifying Distribution Type

The fields are updated, and desired values can be input. If an invalid input is detected it will not update the field. An example of this would be attempting to write a letter instead of a numerical value. When the model configuration is set hit the "Run" button.



Interactive Model

Mean Response Time	--
SD Response Time	--
Throughput	--

Workstations

ID	U	Z	SD Z
1	0.00	0.00	--
2	0.00	0.00	--
3	0.00	0.00	--

Servers

ID	U	S	SD S
1	0.00	0.00	--
2	0.00	0.00	--
3	0.00	0.00	--
4	0.00	0.00	--

Speed: 50%

Figure 7 - 11: Runtime Page

This leads to the runtime page. There is a navigation menu at the top if a different model wants to be selected, or if the current model should have a change in configuration or be restarted.

The middle of the page is the canvas and where the visualizations are rendered. At the bottom of the page there are options for speed control and they do what is labeled. The rightmost button is a pause/resume button and will update according to if the system is running or not. “Maximum Speed” enters simulation mode, where results are calculated extremely quickly. Clicking this pauses the visualization, but will progress the simulation quickly without visualization. When leaving the maximum speed mode the current state of the simulator is copied into the visualization and running may continue. The right of the page displays results relevant to the given model. This concludes the brief walkthrough for utilizing QAS.

C. Source Code

index.html

```
<!DOCTYPE html>
<html>
<head>
  <link rel="stylesheet" href="./css/styles.css">
</head>

<body>
  <div class="index-container">
    <div class="index-introduction">
        <h1>Queueing Model Animation and Simulation </h1>

      <div>
        <div class="index-nav-buttons">
            <button id='buttonModelSelect' class="index-button"> Model Selection </button>
            <button id='buttonHelp'class="index-button"> About </button>
        </div>
    </div>
  </div>
</body>
<script src="Scripts/pageLayout.js"> </script>
</html>
```

help.html

```
<!DOCTYPE html>
<html>
<head>
  <link rel="stylesheet" href="../css/styles.css">
</head>
```

```
<body>
<script src="../Scripts/header.js"></script>
<div class="help-main-block">
  <div class= "help-navigation">
    <div id= "general-button-title" class= "help-button-title"> General Information </div>
    <button id= "about-button" class= "help-navigation-button"> About </button>
    <button id= "terminology-button" class= "help-navigation-button"> Terminology </button>
    <button id= "program-description-button" class= "help-navigation-button"> Program Description
</button>

    <div class="help-navigation-spacing"> </div>
    <div id= "server-button-title" class= "help-button-title"> Available Models </div>
    <button id= "single-server" class= "help-navigation-button">Single Server</button>
    <button id= "single-feedback" class= "help-navigation-button">Single Feedback</button>
    <button id= "parallel-servers" class= "help-navigation-button">Parallel Servers</button>
    <button id= "interactive-workstations" class= "help-navigation-button">Interactive
Workstations</button>
    <button id= "central-server" class= "help-navigation-button">Central Server</button>

    <div class="help-navigation-spacing"> </div>
    <div id= "help-title" class= "help-button-title"> Usage Help </div>
    <button id= "model-selection" class= "help-navigation-button">Model Selection</button>
    <button id= "model-configuration" class= "help-navigation-button">Model Configuration</button>
    <button id= "model-runtime" class= "help-navigation-button">Runtime</button>
  </div>

  <div id="description-block" class = "help-description"></div>
</div>

<script src="../Scripts/help.js"></script>
</body>
```

```
</html>
```

modelConfiguration.html

```
<!DOCTYPE html>
<html>
<head>
  <link rel="stylesheet" href="../css/styles.css">
</head>

<body>
  <script src="../Scripts/header.js"></script>
  <div id = "model-title" class="model-title"> </div>
  <div id="configuration-body" class="configuration-body"> </div>
  <script src="../Scripts/modelConfiguration.js"></script>
</body>
</html>
```

modelDescriptions.html

```
</div class= "server-descriptions">
<button class="server-descriptions">Server Descriptions</button>
<div>
<div class="description-block">

  <div>
    <b>Single Server Model: </b> <br>
    The single server model is the simplest of models. It consists of a single queue and a single processor. Jobs arrive, are enqueued, processed at the server then exit the system. Adjustable parameters for this model are interarrival rate and distribution as well as server processing rate and distribution. This model could be representative of a drive through at a fast food restaurant: people arrive, enqueue, are serviced, and leave.
  </div>
  <div>

    <br><br>

    <b>Two Servers Model:</b> <br>

    The two servers model expands upon the single server model by adding an additional server in
parallel. This model consists of a single queue with two servers. Tasks from the queue are placed into
the next available server for processing. Adjustable parameters for this model are interarrival rate and
distribution as well as server processing rate and distribution for each of the two servers. This type of
model could be experienced at barbershops, a customer checks in (is enqueued) then gets service from
one of many potential barbers


  </div>

  <button class="description-button">Return to Model Selection</button>

</div>


</div>
```

modelSelection.html

```
<!DOCTYPE html>

<html>

<head>

  <link rel="stylesheet" href="../css/styles.css">

</head>


<body>

  <script src="../Scripts/header.js"></script>

<div class="selection-container">

  <div class="model-selection-row">

    <button id="single-server" class="model-selection-button">Single Server</button>

    <button id="single-feedback" class="model-selection-button">Single Server with Feedback</button>

    <button id = "parallel-servers" class="model-selection-button">Parallel Servers</button>
  </div>
  <div class = "model-selection-row">
    <button id="workstations" class="model-selection-button">Interactive Workstations</button>
    <button id="central-server" class="model-selection-button">Central Server</button>
  </div>

<script src="../Scripts/modelSelection.js"></script>
</body>
</html>
```

runTime.html

```
<!DOCTYPE html>
<html>
  <head>
    <link rel="stylesheet" href="../css/styles.css">
  </head>

  <body>
    <script src="../Scripts/header.js"></script>
    <div id="run-time-main-block" class="run-time-main-block">
      <div id="left-block">
        <div id="navigationMenu" class="navigation-menu-update">Menu</div>
        <canvas id="window" class="run-time-canvas-block"></canvas>
        <div id="speedMenu" class="speed-menu"></div>
      </div>

      <div id="resultBlock" class="run-time-results-block">
        <div id="titleOutput">
          <table id="titleTable"> </table>
        </div>
          <div id="resultsOutput">
          <div id="resultsTable"> </div>
        </div>
      </div>
      <div id = "simulation-mode-menu" class="simulation-menu"></div>
    </div>
    <script src="../Scripts/runTime.js"></script>
    <script src="../Draw/config.js"></script>
    <script src="../Draw/draw.js"></script>
    <script src="../Draw/drawable.js"></script>
    <script src="../Draw/visualModels.js"></script>
    <script src="../Simulation/simulation.js"></script>
    <script src="../Simulation/simulationModels.js"></script>
    <script src="../Drivers/driver.js"></script>
    <script src="../Scripts/configurationReader.js"></script>
  </body>
</html>
```

styles.css

```
:root{
  --primary-background-color: #ffffff;
  --primary-text-color: black;
  --primary-button-color: #cdd2ff;
  --primary-button-border-color: black;
  --primary-button-text-color:black;

  /* HEADER */
  --header-color: #ffffff;
  --header-text-color:#000000;
```

```
 /* INDEX ELEMENTS */
 --index-background-color: #ffffff;
 --index-button-color: #cdd2ff;

 /* ABOUT */
 --help-border-color:white;
 --help-background-color: white;

 /* MODEL SELECTION */
 --model-select-border-color: #000000;
 --model-select-background-color: #cdd2ff;

 /* RUNTIME ELEMENTS*/
 --table-primary-color: #cdd2ff;
 --table-secondary-color: #ffffff;
 --run-time-background-color: #ffffff;
 --speed-menu-background-color: #fcf4dc;
 --table-border-color: black;
 --navigation-menu-color:  #ffffff;
 --navigation-button-color: #fcf4dc;
 --speed-button-background-color: #fcf4dc;
 --runtime-hover-color: #f8f8f8;

/* CONFIGURATION ELEMENTS */
--description-block-background: rgb(202, 202, 202);
--description-block-border-color: #cdd2ff;
--configuration-description-block-border-color: #ffffff;
--configuration-description-block-background-color:#ffffff;
```

```
--description-button-background-color: rgb(65, 188, 197);
--configuration-description-background-color: #cdd2ff;
--configuration-body-color: #ffffff;
--configuration-group-background: #ffffff;
--configuration-group-big-background: #ffffff;
--configuration-button-background-color: #fcf4dc;
--configuration-button-border-color: black;
--run-button-background-color: #cdd2ff;
--select-button-background-color: #cdd2ff;


--configuration-big-title-font-size: 26pt;
--configuration-height: 45px;
--configuration-font-size: 18pt;
--configuration-title-width: 260px;
--configuration-input-width: 650px;
--configuration-input-font-size: 12pt;
--configuration-end-button-height: 75px;
--configuration-end-button-width: 170px;
--configuration-end-button-font-size: 16pt;
--configuration-entire-width: 914px;
/*HELP SCREEEN */
--help-button-color: #cdd2ff;
--help-hover-color: #fcf4dc;
--help-background-color: #ffffff;
--help-button-width: 210px;
--help-button-height: 40px;
--help-button-font-size: 14pt;
--help-button-font-type: 'Times New Roman', Times, serif;
--help-button-title-font-size: 18pt;
```

```
 --help-title-height: fit-content;
 --help-title-background-color: #ffffff;
 --help-navigation-spacing: 30px;
 --help-description-background-color: #ffffff;
 --help-inner-font-size: 18pt;
 --help-inner-font-type: 'Times New Roman', Times, serif;
 --help-title-font-size: 22pt;
 --help-title-font-type: 'Times New Roman', Times, serif;

}
.help-navigation-spacing{
  width: var(--help-button-width);
  min-width: var(--help-button-width);
  height: var(--help-navigation-spacing);
  min-height: var(--help-navigation-spacing);
}
.help-button-title{
  display: flex;
  width: var(--help-button-width);
  height: var(--help-title-height);
  font-size: var(--help-button-title-font-size);
  font-family: var(--help-button-font-type);
  background-color: var(--help-title-background-color);
  align-content: end;
  align-content: end;
}
.help-main-block{
  margin-top: 20px;
  height: 100%;
  display: grid;
  grid-template-columns: max-content auto;
  width: auto;
  background-color: var(--help-background-color);
  flex: 1 1 0;
  justify-items: end;
  min-width: 800px;
  padding-bottom: 10px;
}
.help-navigation{
  display: grid;
  justify-self: start;
  justify-content:  start;
  align-content:flex-start;
  height:100%;
  max-height: 100%;
}
.help-navigation-button{
  width:auto;
  height: var(--help-button-height);
  max-height: var(--help-button-height);
  width: var(--help-button-width);
  max-width: var(--help-button-width);
  font-size: var(--help-button-font-size);
  background-color: var(--help-button-color);
  font-weight: normal;
  font-family: var(--help-button-font-type);
}
.help-navigation-button:hover{
  background-color: var(--runtime-hover-color);
}
.help-description{
  height: 100%;
  width: 80%;
  justify-self: start;
  justify-content: start;
  justify-items: start;
  background-color: var(--help-description-background-color);
}
.run-time-canvas-block{
  width: 100%;
}

.help-description-inner-body{
  font-size: var(--help-inner-font-size);
  font-family: var(--help-inner-font-type);
  margin-top: 25px;
  margin-left: 20px;
}
.help-description-title{
  margin-top: 0px;
  font-size: var(--help-title-font-size);
  font-family: var(--help-title-font-type);
  font-weight: bold;
  margin-bottom: 10px;
}

/* e9e9e9 */
```

```
.navigation-menu-update{
  background-color: var(--navigation-menu-color);
  display: flex;


  max-height: 40px;
  /* top: 40px; */
}


.navigation-menu-button-update{
  width:auto;
  max-height:fit-content;
  font-size: 30px;
  background-color: var(--navigation-button-color);
  font-weight: normal;
}
.navigation-menu-button-update:hover{
  background-color: var(--runtime-hover-color);
}
.run-time-main-block{
  display: grid;
  grid-template-columns: 3fr 1fr;
  width: auto;
  background-color: var(--run-time-background-color);
  flex: 1 1 0;
  justify-items: end;
  min-width: 800px;
  padding-bottom: 10px;
}
#left-block{
  display: grid;
  justify-self: start;
  justify-content:  start;
  align-content:flex-start;
  height:80%;
  max-height: 80%;
}
.run-time-canvas-block{
  width: 100%;
}
.run-time-results-block{
  overflow: auto;
}

/*
  --primary-button-color: rgb(202, 202, 202);
  --primary-button-border-color: #264653;
  --primary-button-text-color: black;
  --primary-background-color: #d9d9d9;
  --header-color: #264653;
  --header-text-color: #d9d9d9;
  --model-select-border-color: rgb(102, 99, 99);
  --model-select-background-color: #d9d9d9;
  --text-based-background-color: #264653;
  --text-dominant-section-color: #d9d9d9;

  --result-table-bg-color: blue;
  --result-table-row-color: #d9d9d9;
  --result-table-off-row-color: orange;
  --help-border-color: #264653;
  --help-background-color: rgb(202, 202, 202);

  --configuration-group-background: darkseagreen;
  --configuration-group-big-background: darkseagreen;
  --description-block-background: rgb(202, 202, 202);
  --description-block-border-color: #264653;
  --configuration-description-block-border-color: #264653;
  --configuration-description-block-background-color: rgb(202, 202, 202);
  --description-button-background-color: rgb(65, 188, 197);
  --configuration-description-background-color: darkseagreen;
  --table-primary-color: #FFFFFF;
  --table-secondary-color: #a87300;
`
*/
.speed-menu{
  background-color: var(--speed-menu-background-color);
  margin-top: 0px;
  width: 1200;
  display: flex;

  max-height: 50px;
}
.simulation-menu{
  background-color: var(--speed-menu-background-color);
  height: 0px;
  margin-top: 0px;
  width: 1200;
  display: flex;
```

```
}
.simulation-button{
  min-height: 100px;
  font-size: 28px;
  background-color: var(--speed-button-background-color);
  display: flex;
  flex: 1 1 0px;
  align-items: center;
  justify-content: center;
  margin-top: 50px;
  max-width: 300px;
  margin-right: 20px;
}
.speed-button{
  min-height: 100px;
  font-size: 28px;
  background-color:var(--speed-button-background-color);
  display: flex;
  flex: 1 1 0px;
  align-items: center;
  justify-content: center;
  max-height: 50px;
}
.speed-button:hover{
  background-color: var(--runtime-hover-color);
}

.html{
   height:100%;
   width:100%;
   color:rgb(58, 130, 139);
}
body {
   height: 100%;
   width: 100%;
   background-color: var(--primary-background-color);
   color: var(--primary-text-color);
   margin: 0;
   padding: 0;
}
html{
 height: 100%;
 width: 100%;
}
input[type="radio"] {
  opacity: 0;
  position: fixed;
  width: 0;
}
input[type="radio"] + label{
 border: solid;
 border-color: rgb(207, 200, 200);
 border-width: 2px;
 padding-right: 4px;
 border-radius: 4px;
}
input[type="radio"]:checked + label {
  background-color: var(--primary-button-color);


}
input[type="radio"]:hover + label {
  background-color: var(--primary-button-color);
  border-color: var(--primary-button-color);
}

.header {
    background-color: var(--header-color);
    border:none;
    grid-area: header;
    height: 60px;
    width: 180px;
    font-size: 30pt;
    align-items: center;
    text-align: left;
    color: var(--header-text-color);
  }
  .model-title {
    background-color: var(--configuration-group-big-background);
    font-size: var(--configuration-big-title-font-size);
    font-weight: bold;
    margin-left:20px;
    align-self: center;
    text-align: center;
  }
  /**********************************************************
  //Landing Page
```

```
**********************************************************/
.index-header{
  background-color: var(--header-color);
  border:none;
  grid-area: header;
  height: 100px;
  width: 100%;
  font-size: 30pt;
  align-items: center;
  text-align: center;
  color: var(--header-text-color);
}

.index-introduction{
  display: flexbox;
  align-self: center;
  justify-content: center;
  justify-items: center;
  justify-self: center;
  font-size: 26px;
  margin-left: 15%;
  margin-right: 15%;
  font-weight: bold;
  font-family: Arial, Helvetica, sans-serif;
  vertical-align: middle;
  text-align: center;
}
.index-nav-buttons {
    width: 100%;
  grid-area: navButtons;
  display: grid;
  grid-template-columns: auto auto;
  grid-template-rows: 1fr;
  justify-content: center;
  grid-gap: 50px;
 }

 .index-button{
    border-radius: 10px;
    border-width: medium;
    width: 300px;
    height: 150px;
  border-color: var(--primary-button-border-color);
  color: var(--primary-button-text-color);
  border-style: solid;
  background-color: var(--index-button-color);
  font-size: 25px;
  font-weight: bold;
  margin-top: 80px;
 }
 .index-button:hover{
  background-color: rgb(152, 160, 226);
 }

.index-container {
 background-color: var(--index-background-color);
 position: fixed;
 top: 0;
```

```
  left: 0;
  bottom: 0;
  right: 0;
  overflow: auto;
  vertical-align: middle;
  display: flex;
  align-items: center;
  justify-content: center
}
.index-grid-container  {
  text-align: center;
  padding: 5px 0;
  font-size: 30px;
}
/**********************************************************
//Model Selection
**********************************************************/
.model-selection-row {
  display: flex;
  justify-content: center;
}

.model-selection-button{
  border-radius: 10px;
      border-width: thick;
      width: 400px;
      height: 250px;
    border-color: var(--model-select-border-color);
    border-style: solid;
```

```
   background-color: var(--model-select-background-color);
   font-size: 25px;
   font-weight: bold;
   margin: 40px;
}
.model-selection-button:hover{
  background-color: rgb(153, 214, 238);
}
.selection-container{
  display: flexbox;
  align-items: center;
  justify-content: center;
  margin-top: 5%;


}
.selection-button{
  border-radius: 4px;
  width: 300px;
  height: 150px;
}

/**********************************************************
//Run Time
**********************************************************/
.navigation-menu{
  text-align: center;
  background-color: var(--navigation-menu-color);
  width: fit-content;
  min-height: fit-content;
```

```
}
.navigation-button{
  width: fit-content;
  min-width: fit-content;
  max-height: 40px;
  font-size: 28px;
  overflow-y: auto;
  background-color: var(--navigation-button-color);
}
.navigation-button:hover{
  background-color: var(--runtime-hover-color);
}
.runtime-grid{
  padding: 0px;
  margin: 0px;
  display: grid;
  grid-template-columns: auto auto;
  grid-template-rows: auto auto;
  margin-top: 0px;
  grid-gap: 0px;
  padding: 0px;
}
.screen{
  min-width: 1200px;
  min-height: 800px;
  background-color: var(--primary-background-color);
}
.results{
  grid-area: results;
  min-height: 200px;
  background-color: var(--text-based-background-color);
  color: var(--text-dominant-section-color);
  font-size: 50px;
  text-align: center;
}
.runtime-block{
  display: grid;
  justify-content: center;
  grid-template-columns: 1fr auto;
  width: 100%;
}

/*************************************************************
//Configuration
*************************************************************/
.configuration-group{
  background-color: var(--configuration-group-background);
  display: flex;
  font-size: var(--configuration-font-size);
  justify-content: center;
  justify-self: center;
}
.configuration-body{
  background-color: var(--configuration-body-color);
  display: flexbox;
  margin-top: 0%;
}
.configuration-selection-button{
  border-radius: 0px;
  border-width: thin;
  width: 202px;
  height: 40;
  border-color: var(--configuration-button-border-color);
  background-color: var(--configuration-button-background-color);
  border-style: solid;
  font-size:var(--configuration-font-size);
  justify-content: center;
  justify-self: center;
}
.configuration-category-start-division{
  font-family: 'Times New Roman', Times, serif;
  font-size: var(--configuration-font-size);
  font-size: 20pt;
  font-weight: bold;
  width: var(--configuration-entire-width);
  min-width: var(--configuration-entire-width);
  display:flex;
  padding-left: 0px;
  padding-top: 20px;
}
.configuration-category-end-division{
  width: var(--configuration-entire-width);
  min-width: var(--configuration-entire-entire);
  height: var(--configuration-height);
  min-height: var(--configuration-height);
  display:flex;
}
```

```
.configuration-labelless-division{
  width: var(--configuration-entire-width);
  min-width: var(--configuration-entire-width);
  display:inline-flex;
  border-style: solid;
  border-width: 2px;
  height: var(--configuration-height);
  padding-left: 7px;
  padding-right: 7px;
  align-items: center;
  align-self: center;
  text-align: center;
  justify-self: center;
  font-family: 'Times New Roman', Times, serif;
}
.distribution-group{
  padding: 5px;
  width: var(--configuration-input-width);
  min-width: var(--configuration-input-width);
  display:flex;
  border-style: solid;
  border-width: 2px;
}
.feedback-input{
  width: var(--configuration-input-width);
  min-width: var(--configuration-input-width);
  display:inline-flex;
  border-style: solid;
  border-width: 2px;
  height: var(--configuration-height);
  padding-left: 5px;
  padding-right: 5px;
  align-items: center;
  align-self: center;
  text-align: center;
  justify-self: center;
  font-family: 'Times New Roman', Times, serif;
}
.disk-configuration-group{
  background-color: var(--configuration-group-background);
  display: flex;
  font-size: var(--configuration-font-size);
  justify-content: center;
  justify-self: center;
}
.disk-distribution-group{
  padding: 5px;
  width: var(--configuration-input-width);
  min-width: var(--configuration-input-width);
  display:flex;
  border-style: solid;
  border-width: 2px;
  border-bottom:none;
}
.disk-weight-input{
  width: var(--configuration-input-width);
  min-width: var(--configuration-input-width);
  display:inline-flex;
  border-style: solid;
  border-width: 2px;
  height: var(--configuration-height);
  padding-left: 5px;
  padding-right: 5px;
  align-items: center;
  align-self: center;
  text-align: center;
  justify-self: center;
  font-family: 'Times New Roman', Times, serif;
  border-top: none;
}
.disk-distribution-title{
  display:inline-flex;
  /* font-weight: bold; */
  width: var(--configuration-title-width);
  min-width: var(--configuration-title-width);
  height: var(--configuration-height);
  border-style: solid;
  border-width: 2px;
  align-items: end;
  border-bottom: none;
}
.disk-weight-title{
  display:inline-flex;
  /* font-weight: bold; */
  width: var(--configuration-title-width);
  min-width: var(--configuration-title-width);
  height: var(--configuration-height);
  border-style: solid;
  border-width: 2px;
  align-items: center;
  border-top: none;
}
.disk-weight-group{
  background-color: var(--configuration-group-background);
  display: flex;
  font-size: var(--configuration-font-size);
  justify-content: center;
  justify-self: center;
  border:none;
}
/* Label For Where Text is Input*/
.configuration-label{
  display: inline-flex;
  align-items: center;
  margin-left: 5px;
  margin-right: 10px;
  align-self: center;
  text-align: center;
  justify-self: center;
  font-family: 'Times New Roman', Times, serif;
}
.configuration-component-title{
  display:inline-flex;
  /* font-weight: bold; */
  width: (--configuration-title-width);
  min-width: var(--configuration-title-width);
  max-width: var(--configuration-title-width);
  height: var(--configuration-height);
  border-style: solid;
  border-width: 2px;
  align-items: center;
  font-size: var(--configuration-font-size);
}
/* Where Text is Input*/
.configuration-text-input{
  display: inline-block;
  vertical-align: middle;
  height: var(--configuration-input-font-size);
  padding-left: 10px;
  width: 60px;
  font-size: var(--configuration-input-font-size)
}
.configuration-title-big{
  padding-top: 40px;
  background-color: var(--configuration-group-big-background);
  font-size: var(--configuration-big-title-font-size);
  font-weight: bold;
  margin-left:20px;
  align-self: center;
  text-align: center;
}
.configuration-subdivision{
  align-self: center;
  text-align: center;
  justify-self: center;
```

```
}
.configuration-radio{
  border-style: solid;
  height: 16pt;
}
.server-descriptions{
  background-color: var(--header-color);
  border:none;
  grid-area: header;
  height: 100px;
  width: 100%;
  font-size: 30pt;
  align-items: center;
  color: var(--header-text-color);
}
#end-buttons{
  margin-top: 40px;
}
#run-button{
  border-radius: 10px;
  background-color: var(--run-button-background-color);
  width: var(--configuration-end-button-width);
  height: var(--configuration-end-button-height);
  font-size: var(--configuration-end-button-font-size);
  margin-right: 40px;

  justify-self: center;
  justify-content: center;
  align-self: center;
  text-align: center;
}
#run-button:hover{
  background-color: rgb(153, 214, 238);
}
#select-button{
  background-color: var(--select-button-background-color);
  border-radius: 10px;
  width: var(--configuration-end-button-width);
  height: var(--configuration-end-button-height);
  font-size: var(--configuration-end-button-font-size);
  align-self: center;
  text-align: center;
  justify-self: center;
}
#select-button:hover{
  background-color: rgb(153, 214, 238);
}
.description-block{
  margin-left: 20px;
  margin-right: 20px;
  margin-top: 40px;
  border-style: solid;
  border-radius: 4px;
  border-color: var(--description-block-border-color);
  border-width: 10px;
  font-size: 30px;
  text-align: left;
  background-color: var(--description-block-background);
```

```
}
.configuration-description-block{
  position: fixed;
  left: 950px;
  top: 120px;
  margin-left: 20px;
  margin-right: 20px;
  margin-top: 40px;
  border-style: solid;
  border-radius: 4px;
  border-color: var(--configuration-description-block-border-color);
  border-width: 10px;
  font-size: 30px;
  text-align: left;
  background-color: var(--configuration-description-block-background-color);
}
.configuration-description-block-2{
  position: fixed;
  left: 0px;
  top: 300px;
  margin-left: 20px;
  margin-right: 20px;
  margin-top: 40px;
  border-style: solid;
  border-radius: 4px;
  border-color: var(--configuration-description-block-border-color);
  border-width: 10px;
  font-size: 30px;
  text-align: left;
  background-color: var(--configuration-description-block-background-color);
 }
 .description-button{
  border-radius: 0px;
  border-width: thick;
  width: 300px;
  height: 100px;
  border-color: var(--primary-button-border-color);
  color: var(--primary-button-text-color);
  border-style: solid;
  font-size: 25px;
  font-weight: bold;
  background-color: var(--description-button-background-color);
 }

 .distribution-input{
  display:inline-flex;
  align-items: center;
 }

 .configuration-description{
  background-color: var(--configuration-description-background-color);
  display: flex;
  flex-wrap: wrap;
  font-size: 16pt;
 }
.simulationOutputTest{
 margin-left: 20px;
 margin-right: 20px;
```

```
  margin-top: 40px;
  border-style: solid;
  border-radius: 4px;
  border-color: #264653;
  border-width: 10px;
  font-size: 30px;
  text-align: left;
  background-color: rgb(202, 202, 202);
}

table {
  font-family: arial, sans-serif;
  border-collapse: collapse;
  font-size: 20px;
  margin-bottom: 20px;
  margin-right: 5px;
}
td {
  border: 2px solid var(--table-border-color);
  text-align: left;
  padding: 8px;
  width: 200px;
  font-size: 20px;
}
td:nth-child(even){
  width: 80px;
}
th{
  text-align: left;
  width: 200px;
  font-size: 20px;
  background-color: white;
}
tr:nth-child(odd){
  background-color: var(--table-primary-color);
}
tr:nth-child(even) {
  background-color: var(--table-secondary-color);
}
.tiny-table-data{
  border: 2px solid var(--table-border-color);
  text-align: left;
  width: 60px;
  max-width: 50px;
  font-size: 20px;
}
.tiny-table{
  font-family: arial, sans-serif;
  border-collapse: collapse;
  font-size: 5px;
  margin-bottom: 0px;
  margin-right: 5px;
}
.tiny-table-heading{
  border: 2px solid var(--table-border-color);
  text-align: left;
  width: 80px;
  max-width: 80px;
  font-size: 20px;
}
.title-table{
  margin-bottom: 0px;
}
.tiny-information{
  margin-bottom: 30px;
  font-size: 16pt;
}
```

config.js

```
//ANIMATION SPEED
let taskMovementRate = 10;
var taskSteps = 1;
//let visualTimeScaling = 100;
let defaultUpdateRate = 100;
let updateRate = defaultUpdateRate;

var timeScaling = 1;
let timePerTick = 1*timeScaling;

//let timeConversionMultiplier = visualTimeScaling*1000*defaultUpdateRate ;
let speeds = `
    Speed1 : 2
    Speed2: 3
    `;

//speed 3: 3
```

```
//CANVAS COMPONENTS
let canvasWidth = 1400;
let canvasHeight = 800;
let canvasScaling = 1; //scaling for resize
let canvasScalingX = 1;
let canvasScalingY = 1;

//LINE SIZE
let lineWidth = 4;
let canvasLineWidth = 4; //canvas border
let arrowThickness = 4;
let arrowHeadLength = 10;
let connectionThickness = 5;

//COLORS
let canvasBorderColor = "white";
let background = "white";
let strokeColor = "black";
let backgroundColor = "white";
let connectionColor = "black";

//SIZES
let serverRadius = 35;
let workStationSize = 30; //Length is two times this, simular to radius
let taskRadius = 15;
let diskRadius = 35;
let diskWidth = 30;
let diskHeight = 30;
let queueSlotSize = 40;
```

```
//SPACINGS
let horizontalSpacing = 75; //horizontal spacing
let verticalSpacing = 150; //vertical spacing
let parallelSpacingHorizontal = 20;
let parallelSpacingVertical = 80;
let workstationSpacingVertical = 60;
let parallelSpacingWorkstation = 60;

//QUEUE SLOT COUNT
let numberSlotsInQueue = 10;
let maxSlots = 20;

//color choices:
//paleturquoise
// #f4a261 // orangeish
// #e9c46a // yellowish
// #e76f51 // darker orange
// #2a9d8f //greenish
// #264653//darker green//
```

draw.js

```
//Draws components onto the canvas
let canvas = document.getElementById("window");
let context = canvas.getContext("2d");

//******************** */
//Canvas Functions
```

```
//******************** */
function initializeCanvas(){
   canvas.width = canvasWidth;
   canvas.height = canvasHeight;
   context.lineWidth = canvasLineWidth;
   drawCanvas();
}

function drawCanvas(){
   context.fillStyle = background;
   context.fillRect(0,0,canvasWidth,canvasHeight);
   context.strokeStyle = canvasBorderColor;
   context.strokeRect(0, 0, canvasWidth, canvasHeight);
   context.strokeStyle = strokeColor;
}

//*********************** */
//  Connection Functions
//*********************** */

//Draws a line from (x1,y1) to (x2,y2)
function drawLine(x1, y1, x2, y2, color){
   context.lineWidth = lineWidth;
   context.fillStyle = color;
   context.strokeStyle = color;
   context.beginPath();
   context.moveTo(x1, y1);
   context.lineTo(x2, y2);
   context.fillRect(x1- lineWidth/2, y1-lineWidth/2, lineWidth, lineWidth);
   context.stroke();
}


//Draws the head of an arrow oriented based on the given coordinates of the line
function drawArrowHead(x1,y1,x2,y2,color){
   context.fillStyle = color;
   let angle = Math.atan2(y2-y1, x2-x1);
   context.beginPath();
   context.moveTo(x2,y2);
   context.lineTo(x2 - arrowHeadLength * Math.cos(angle - Math.PI / 6), y2 - arrowHeadLength *
Math.sin(angle - Math.PI / 6));
   context.lineTo(x2 - arrowHeadLength * Math.cos(angle + Math.PI / 6), y2 - arrowHeadLength *
Math.sin(angle + Math.PI / 6));
   context.lineTo(x2,y2);
   context.lineTo(x2 - arrowHeadLength * Math.cos(angle - Math.PI / 6), y2 - arrowHeadLength *
Math.sin(angle - Math.PI / 6));
   context.stroke();
}


//Draws Arrow Between Two Coordinates
function drawArrow(x1, y1, x2, y2, color){
   context.lineWidth = connectionThickness;
   drawLine(x1, y1, x2 - lineWidth, y2, color);
   drawArrowHead(x1, y1, x2 - lineWidth, y2, color);
 }


function connectArrowReverse(){
   try{
      for (var i=0; i < arguments.length-1; i++) {
         drawArrow(arguments[i].connections.end.x, arguments[i].connections.end.y,
            arguments[i+1].connections.start.x + arrowHeadLength, arguments[i+1].connections.start.y,
connectionColor);
        }
    } catch(err){
        console.log("ERROR IN CONNECTION");
        console.error();
    };
}

//Draws Arrow Connection Between Arguments
function connectArrow(){
    try{
        for (var i=0; i < arguments.length-1; i++) {
            drawArrow(arguments[i].connections.end.x, arguments[i].connections.end.y,
                arguments[i+1].connections.start.x, arguments[i+1].connections.start.y, connectionColor);
        }
    } catch(err){
        console.log("ERROR IN CONNECTION");
        console.error();
    };
}

//Draws Line Connection Between Arguments
function connectLine(){
    try{
        if(arguments.length < 2) return;
        context.beginPath();
        context.moveTo(arguments[0].connections.start.x, arguments[0].connections.end.y);
        for (var i=0; i < arguments.length-1; i++) {
            drawLine(arguments[i].connections.end.x, arguments[i].connections.end.y,
                arguments[i+1].connections.start.x, arguments[i+1].connections.start.y, connectionColor);
        }
    } catch(err){
        console.log("ERROR IN CONNECTION");
        console.error();
    }
}


//Connects Parallel Components
function drawParallelstart(){
    context.beginPath();
    context.moveTo(arguments[0].connections.start.x, arguments[0].connections.start.y);
    for (var i=0; i < arguments.length - 1; i++) {
        context.lineTo(arguments[i].coordinates.x - parallelSpacingHorizontal,
arguments[i].connections.start.y);
        context.lineTo(arguments[i].coordinates.x - parallelSpacingHorizontal, arguments[i +
1].connections.start.y);
        context.lineTo(arguments[i + 1].coordinates.x , arguments[i + 1].connections.start.y);
    }
    context.stroke();
}
function drawParallelStartArrowed(){
    context.beginPath();
    drawArrow()
    context.moveTo(arguments[0].connections.start.x, arguments[0].connections.start.y);
    for (var i=0; i < arguments.length - 1; i++) {
        context.lineTo(arguments[i].connections.start.x - parallelSpacingHorizontal,
arguments[i].connections.start.y);
```

```
        context.lineTo(arguments[i].connections.start.x - parallelSpacingHorizontal,
arguments[i].connections.start.y + parallelSpacingVertical);

        context.lineTo(arguments[i + 1].connections.start.x, arguments[i + 1].connections.start.y);

        drawArrow(arguments[i].connections.start.x, arguments[i].connections.start.y,
arguments[i].connections.start.x + parallelSpacingHorizontal, arguments[0].start.y, connectionColor);

    }

    context.stroke();


}

//coordinates.x + arguments[i].constructor.width

function drawParallelRight(){

    context.beginPath();

    context.moveTo(arguments[0].connections.end.x, arguments[0].connections.end.y);

    for (var i=0; i < arguments.length - 1; i++) {

        context.lineTo(arguments[i].coordinates.x + arguments[i].constructor.width +
parallelSpacingHorizontal, arguments[i].connections.end.y);

        context.lineTo(arguments[i].coordinates.x + arguments[i].constructor.width +
parallelSpacingHorizontal,  arguments[i + 1].connections.end.y);

        context.lineTo(arguments[i + 1].coordinates.x + arguments[i].constructor.width, arguments[i +
1].connections.end.y);

    }

    context.stroke();

}


function drawParallelConnection(){

    context.strokeStyle = connectionColor;

    try{

        drawParallelstart.apply(null, arguments);

        drawParallelRight.apply(null, arguments);

    }catch(err){
        console.log("ERROR IN PARALLEL CONNECTION");
        console.error();
    }
}

function drawParallelConnectionArrowed(){
    context.strokeStyle = connectionColor;
    try{
        drawParallelStartArrowed.apply(null, arguments);
        drawParallelRight.apply(null, arguments);
    }catch(err){
        console.log("ERROR IN PARALLEL CONNECTION");
        console.error();
    }
}
```

drawable.js

```
/************************************************************
* Contains Components Needed To Draw Objects Onto The Canvas
************************************************************/

//Base drawable object
class DrawableObject {
    constructor(x, y, identifier) {
        this.identifier = identifier;
        this.coordinates = { "x": x, "y": y };
        this.connections = {"start": {"x": x, "y": y}, "end": {"x": x, "y": y}};
        this.previous = null;
        this.next = null;
        this.isAvailable = true;
        this.isReversed = false;
    }
    reset(){
        this.isAvailable = true;
        this.task = null;
    }
    calculateWidth(){
        Math.abs(this.connections.end.x - this.connections.start.x);
    }
    calculateHeight(){
        return undefined;
    }
    setNext(drawableObject){
        this.next = drawableObject;
        if (drawableObject != null)
            this.next.previous = drawableObject;
    }
    setPrevious(drawableObject){
        this.previous = drawableObject;
        this.previous.next = drawableObject;
    }
    draw (){}; //will be overidden with specific draw function

    receiveMessage(message){
        var type = message.type;
        switch(type){
            case "accept":
                this.acceptTask(message);
            case "advance":
                this.advanceTask(message);
            case "receive":
                this.receiveTask(message);
        }
    }

    acceptTask(task){ //called when available to accept
        this.task = task;
        task.setDestinationObject(this);
    }
    advanceTask(task = this.task){ //moves to next component
        if(this.next != null){
            this.next.acceptTask(task);
            this.task = null;
        }
        else{
            this.task = null;
        }
    }
    receiveTask(task){
        this.advanceTask(task);
    }
    reverseDrawing(){
        var temp = this.coordinates.x;
        this.connections.start.x = this.connections.end.x;
        this.connections.end.x = temp;
        this.isReversed = true;
    }
```

```
   checkAvailability(task){
      return this.isAvailable;
   }
}
DrawableObject.width = undefined;
DrawableObject.height = undefined;

/*********************************************************
 * Model Components
 *********************************************************/

 //Visual Element For Tasks
class TaskVisual extends DrawableObject{
   constructor(x,y,identifier, color){
      super(x,y,identifier);
      this.color = color;
      this.visible = true;
      this.destination = {"x": x, "y": y};
      this.movementRate = taskMovementRate;
      this.destinationObject = {};
      this.isMoving = false;
   }
   setDestination(coordinates){
      this.destination.x = coordinates.x;
      this.destination.y = coordinates.y;
      this.isMoving = true;
   }
   move(){
      if (this.destination != this.coordinates && this.isMoving){
```

```
this.isDrawingSelf = true;
for(var i = 0; i < taskSteps; i++){
    var dy = this.destination.y - this.coordinates.y;
    var yDirection = dy == 0 ? 0 : dy > 0 ? 1 : -1;
    this.coordinates.y += yDirection*this.movementRate;

    dy = this.destination.y - this.coordinates.y;
    if (dy * yDirection < 0){
        this.coordinates.y = this.destination.y;
    }

    if(dy==0){
        var dx = this.destination.x - this.coordinates.x;
        var xDirection = dx == 0 ? 0 : dx > 0 ? 1 : -1;
        this.coordinates.x += xDirection*this.movementRate;

        dx = this.destination.x - this.coordinates.x;
        if (dx * xDirection < 0){
            this.coordinates.x = this.destination.x;
        }
    }

    if(dx == 0 && dy == 0){
        this.isMoving = false;
        this.arrive();
    }
    /*

    currentModel.drawnConnections.forEach(connection => {
```

```
                currentModel.callConnectionFunction(connection);
            });

            context.fillStyle = this.color;
            context.beginPath();
            context.arc(this.coordinates.x, this.coordinates.y, taskRadius, 0, 2*Math.PI);
            context.fill();
            */
        }
        this.isDrawingSelf = false;
    }
}

draw(){
    if(this.isMoving ){//&& this.isDrawingSelf != true){
        this.move();
    }

    if (this.isMoving){
        context.fillStyle = this.color;
        context.beginPath();
        context.arc(this.coordinates.x, this.coordinates.y, taskRadius, 0, 2*Math.PI);
        context.fill();
    }
}
setDestinationObject(destinationObject){
    if(destinationObject != null){
        this.destinationObject = destinationObject;
        this.setDestination(destinationObject.connections.start);
        }
    }
    arrive(){this.destinationObject.receiveTask(this);}
    setCoordinates(coordinates){
        this.coordinates.x = coordinates.x;
        this.coordinates.y = coordinates.y;
    }
    //moveTo(visualElement){visualElement.acceptTask();}
    makeInvisible(){this.visible = false;}
    makeVisible(){this.visible = true;}
}


//Visual Element for Queues
class QueueVisual extends DrawableObject{
    constructor(x,y,identifier){
        super(x,y,identifier);
        this.connections.start = {"x": x, "y": y}; //start connection point
        this.connections.end = {"x": x + queueSlotSize * numberSlotsInQueue, "y": y}; //right connection
point
        this.addQueueSlotVisuals();
        this.numberTasks = 0;
        this.tasks = {};
        this.taskOverflow = []; //used to contain tasks greater than amount of slots
        this.tasksWaiting = []; //used to contain tasks that cannot be processed by next component yet
        this.type = "Queue";
    }
    addQueueSlotVisuals(){ //adds queue slots to queue and labels based on position
        this.queueSlots = [];
        for(let i = numberSlotsInQueue - 1; i >= 0; i--){
            this.queueSlots.push(new QueueSlotVisual(this.coordinates.x + i * queueSlotSize,
this.coordinates.y, `${this.identifier}_${i}`));
        }
    };
    reset(){
        this.numberTasks = 0;
        this.taskOverflow = [];
        this.tasksWaiting = [];
        this.tasks = {};


        for(var i = 0; i < this.queueSlots.length; i++){
            this.queueSlots[i].task = null;
        }
    }
    getTaskByIdentifier(taskIdentifier){
        if(this.tasks[taskIdentifier] == undefined){
            return undefined;
        }
        return (this.tasks[taskIdentifier].task);
    }
    receiveTask(task){
        if(this.tasks[task.identifier] !== undefined){
          stopModel();
          alert("TASK ARRIVING THAT IS ADDED ALREADY");
          return;
        }
        this.tasks[task.identifier] = {"assignedSlot": null, "task": task, "server": null};
    if (this.numberTasks  < numberSlotsInQueue) {
      this.queueSlots[this.numberTasks].acceptTask(task);
      this.tasks[task.identifier].assignedSlot = this.numberTasks;


      if(this.next.checkAvailability(task))
        this.next.acceptTask(task);
      else
        this.tasksWaiting.push(task);
    }
    else {  //Number of tasks is greater than displayed slots
      this.taskOverflow.push(task);
    }
    this.numberTasks++;
    task.setCoordinates(this.next.connections.start);
  }
  advanceTask(task){ //moves task to next component if present, shifts visual for task in queue slots
    if(this.tasks[task.identifier] == undefined) {
      stopModel();
      return undefined;
    }
    this.shiftQueueSlots(task);
    delete this.tasks[task.identifier];
  }


  shiftQueueSlots(task){
    var currentTaskData = this.tasks[task.identifier];


    //Remove Shifted Task
    var assignedSlot = currentTaskData.assignedSlot;
    this.queueSlots[assignedSlot].task = null;
    this.numberTasks --;

    //Move the tasks in the queue over
    this.moveTasksInQueue(assignedSlot);

    //Connected component now has an opening, add waiting task to it if available
    if(this.tasksWaiting.length > 0 && this.next.checkAvailability(this.tasksWaiting[0])){
        var shiftedTask = this.tasksWaiting.shift();
        this.next.acceptTask(shiftedTask);
    }

    //Handle tasks that cannot fit into the queue
    this.processOverflow();
}
moveTasksInQueue(index){
    //Move Items in Queue Over
    for(var i = index; i < numberSlotsInQueue - 1; i++){
        if(this.queueSlots[i+1].task != null)
            this.queueSlots[i].task = this.queueSlots[i+1].task;
        else this.queueSlots[i].task = null;
        this.queueSlots[i+1].task = null;
        if(this.queueSlots[i].task != null){
            this.tasks[this.queueSlots[i].task.identifier].assignedSlot = i;
        }
    }
}
processOverflow(){
    //If any items in overflow add to last slot.
```

```
        if(this.taskOverflow.length > 0){
            var overflowTask = this.taskOverflow.shift();
            this.tasks[overflowTask.identifier] = {"assignedSlot": numberSlotsInQueue - 1, "task":
overflowTask, "server": null};
            this.queueSlots[numberSlotsInQueue - 1].acceptTask(overflowTask);


            if(this.next.checkAvailability(overflowTask))
                this.next.acceptTask(overflowTask);
            else this.tasksWaiting.push(overflowTask);
        }
    }
    updateQueueSlotLocations(){
        var xCoordinate = this.coordinates.x;
        for (let i = numberSlotsInQueue-1; i >= 0; i--){
            this.queueSlots[i].coordinates.x = xCoordinate;
            this.queueSlots[i].connections.start.x = xCoordinate;
            xCoordinate += queueSlotSize;
            this.queueSlots[i].connections.end.x = xCoordinate;
        }
    }
    draw(){
        for (let i = 0; i < numberSlotsInQueue; i++){
            this.queueSlots[i].draw()
        }
    }
    reverseDrawing(){
        var temp = this.connections.start.x;
        this.connections.start.x = this.connections.end.x;
        this.connections.end.x = temp;
        var tempCoordinates;
        var tempConnections;
        for (let i = 0; i < numberSlotsInQueue/2; i++){
            tempCoordinates = this.queueSlots[i].coordinates;
            this.queueSlots[i].coordinates = this.queueSlots[numberSlotsInQueue - 1 - i].coordinates;
            this.queueSlots[numberSlotsInQueue - 1 - i].coordinates = tempCoordinates;

            tempConnections = this.queueSlots[i].connections;
            this.queueSlots[i].connections = this.queueSlots[numberSlotsInQueue - 1 - i].connections;
            this.queueSlots[numberSlotsInQueue - 1 - i].connections = tempConnections;
        }
        this.isReversed = true;
    }
}

QueueVisual.width = queueSlotSize * numberSlotsInQueue;
QueueVisual.height = QueueVisual.width;
QueueVisual.type = "Queue";

//Visual Element for Queue Slots
class QueueSlotVisual extends DrawableObject{
    constructor(x,y,identifier){
        super(x,y,identifier);
        this.connections.start = {"x": x, "y": y};
        this.connections.end = {"x": x + queueSlotSize, "y": y};
        this.task = null;
        this.type = "Queue Slot";
    }
    acceptTask(task){
        if (task == undefined){
            this.task = null
        }
        this.task = task;
    }

    draw(){
        context.fillStyle = this.task == null? backgroundColor : this.task.color;
        context.fillRect(this.coordinates.x, this.coordinates.y - queueSlotSize/2, queueSlotSize,
queueSlotSize);
        context.strokeRect(this.coordinates.x, this.coordinates.y - queueSlotSize/2, queueSlotSize,
queueSlotSize);
    }
}
QueueSlotVisual.width = queueSlotSize;
QueueSlotVisual.height = queueSlotSize;
QueueSlotVisual.type = "Queue Slot";

//Visual Element for Servers
class ServerVisual extends DrawableObject{
    constructor(x,y, identifier){
        super(x,y,identifier);
        this.connections.start = {"x": x, "y": y};
        this.connections.end = {"x": x + serverRadius*2, "y": y};
        this.task = null;
        this.connectedComponent = null;
        this.type = "Server";
        this.parallelBlock = null;
        this.immediatelyProcess = false;
        this.backLog = {};
    }
    reset(){
        this.task = null;
        this.immediatelyProcess = false;
        this.backLog = {};
        this.isAvailable = true;
    }
    setConnection(drawableObject){
        this.connectedComponent = drawableObject;
    }
    acceptTask(task){
        this.isAvailable = false;
        this.task = task;
        if(this.backLog[task.identifier] !== undefined){
            if(this.connectedComponent != null && this.connectedComponent.tasks[task.identifier] !=
undefined)
                this.advanceTask(task);
            delete this.backLog[task.identifier];
        }
        this.draw();
    }
    receiveTask(task){};
    advanceTask(task){
        this.isAvailable = true;
        if(this.task !== task)
            this.backLog[task.identifier] = task;
        else{
            task.setCoordinates(this.connections.end);


            if(this.parallelBlock !== null){
               this.parallelBlock.advanceTask(task);
               task.setCoordinates(this.parallelBlock.connections.end);
            }
            else{
               task.setCoordinates(this.connections.end);
               if(this.next.isAvailable){
                  this.next.acceptTask(this.task);
               }
            }


            task.makeVisible();
            if(this.connectedComponent != null)
               this.connectedComponent.advanceTask(task);
            this.task = null;
         }
      }
      draw(){
         context.fillStyle = this.task == null? backgroundColor: this.task.color;
         context.beginPath();
         context.arc(this.coordinates.x + serverRadius, this.coordinates.y, serverRadius, 0, 2*Math.PI);
         context.fill();
         context.stroke();
      }
   }
   ServerVisual.width = serverRadius*2;
   ServerVisual.height = serverRadius*2;
```

```
ServerVisual.type = "Server";


class FeedbackServerVisual extends ServerVisual{
  constructor(x,y,identifier){
    super(x,y,identifier);
    this.taskDestinations = {};
  }
  reset(){
    this.task = null;
    this.taskDestinations = {};
    this.immediatelyProcess = false;
    this.backLog = {};
    this.isAvailable = true;
  }
  connectExit(exitPoint){
    this.exitPoint = exitPoint;
  }
  setTaskDestination(task, destination){
    this.taskDestinations[task.identifier] = destination;
  }
  getTaskDestination(task){
    return this.taskDestinations[task.identifier];
  }
  advanceTask(task){
    if(this.task !== task)
      this.backLog[task.identifier] = task;


    else{
      var destination = this.getTaskDestination(task);
```

```
            if(destination == "Exit"){
               this.exitPoint.acceptTask(task);
            }
            else{
               this.next.acceptTask(task);
            }
            task.setCoordinates(this.connections.end);

            this.task = null;
            this.isAvailable = true;
            task.makeVisible();
            if(this.connectedComponent != null)
               this.connectedComponent.advanceTask(task);

            delete this.taskDestinations[task.identifier];
         }
      }
   }
//Workstation
class WorkstationVisual extends ServerVisual{
   constructor(x,y, identifier){
      super(x,y, identifier);
      this.connections.end.x = this.coordinates.x + workStationSize*2;
      this.type = "Workstation";
   }

   draw(){
      context.fillStyle = this.task == null? backgroundColor: this.task.color;
      context.beginPath();
```

```
        context.moveTo(this.coordinates.x + workStationSize + workStationSize * Math.cos(0),
this.coordinates.y + workStationSize * Math.sin(0));
        for (var side = 0; side <= 6; side++) {
            context.lineTo(this.coordinates.x + workStationSize + workStationSize * Math.cos(side * 2 *
Math.PI / 6), this.coordinates.y + workStationSize * Math.sin(side * 2 * Math.PI / 6));
        }
        context.lineTo(this.coordinates.x + workStationSize*2, this.coordinates.y + 1);
        context.fill();
        context.stroke();
    }
}
WorkstationVisual.width = workStationSize*2;
WorkstationVisual.height = workStationSize*2;
WorkstationVisual.type = "Workstation";

//Disk
class DiskVisual extends ServerVisual{
    constructor(x,y, identifier){
        super(x,y,identifier);
        this.connections.start = {"x": x, "y": y};
        this.connections.end = {"x": x + diskRadius*2, "y": y};
        this.task = null;
        this.type = "Disk";
    }
    draw(){
        context.fillStyle = this.task == null? backgroundColor: this.task.color;
        context.draw
        context.beginPath();
        context.arc(this.coordinates.x + diskRadius, this.coordinates.y, diskRadius, 0, 2*Math.PI);
        context.fill();

        context.stroke();

    }

}

DiskVisual.width = diskRadius*2;

DiskVisual.height = diskRadius*2;

DiskVisual.type = "Disk";


//Visual Anchors Are Used For Connections to Draw Path

class VisualAnchor extends DrawableObject{

    constructor(x,y){

        super(x,y, `Anchor(${x},${y})`);

        this.type = "Visual Anchor";

    }

}

VisualAnchor.type = "Visual Anchor";

class EntrancePoint extends VisualAnchor{

    constructor(x,y, identifier){

        super(x,y);

        this.identifier = identifier;

        this.type = "Entrance Point";

    }

}

//Exit Point Anchor

class ExitPoint extends VisualAnchor{

    constructor(x,y, identifier){

        super(x,y);

        this.identifier = identifier;

        this.type = "Exit Point";
    }
}



//Parallel Container
class ParallelContainer extends DrawableObject{
    constructor(xStart, yCenter, identifier, objectType, numberOfElements){
        super(xStart, yCenter, identifier);
        this.type = "Parallel Block";
        this.objectType = objectType;
        this.numberOfElements = numberOfElements;
        this.containedElements = [];
        this.objectIndices = {};
        this.demandedTasks = {};
        this.createParallelObjects();
        this.connectedQueue = null;
        this.tasks = {};
    }
    reset(){
        this.demandedTasks = {};
        this.tasks = {};
    }
    checkTask(taskIdentifier){
        if(this.tasks[taskIdentifier] !== undefined && this.tasks[taskIdentifier].isAvailable)
            return true;
        else {
            return false;
        }
    }
```

```
connectQueue(queue){
   this.connectedQueue = queue;
   for(var i = 0; i < this.containedElements.length; i++){
      this.containedElements[i].setConnection(this.connectedQueue);
   }
}
connectInteriorObjects(){
   for(var i = 0; i < this.containedElements.length - 1; i++){
      for(var j = 0; j < this.containedElements[i].length; j++){
         connectArrow(this.containedElements[i][j], this.containedElements[i+1][j]);
      }
   }
}
connectInteriorObjectsReversed(){
   for(var i = 0; i < this.containedElements.length - 1; i++){
      for(var j = 0; j < this.containedElements[i].length; j++){
         connectArrowReverse(this.containedElements[i][j], this.containedElements[i+1][j]);
      }
   }
}
createParallelObjects(){
   var objectWidth = this.findWidth(this.objectType);
   var objectHeight = this.objectType.width;
   this.connections.end.x = this.coordinates.x + 2*parallelSpacingHorizontal + objectWidth;
   this.calculateWidth();
   this.placeAllObjects();
}
placeAllObjects(){
   var x = this.coordinates.x + parallelSpacingHorizontal;
```

```
    var y = this.coordinates.y;


    var newElements = [];


    if(!Array.isArray(this.objectType)){
        this.placeObjects(x,y, this.objectType, this.containedElements);
    }
    else {
        for(var i = 0; i < this.objectType.length; i++){
            if(this.containedElements[i] === undefined){
                this.containedElements.push([]);
            }
            this.placeObjects(x, y, this.objectType[i], this.containedElements[i]);
            x += this.objectType[i].width + horizontalSpacing;
        }


        for(var i = 0; i < this.numberOfElements; i++){
            for(var j = 1; j < this.objectType.length; j++){
                this.containedElements[j-1][i].setNext(this.containedElements[j][i]);
                if(this.containedElements[j-1][i].type == "Queue"){
                    this.containedElements[j][i].setConnection(this.containedElements[j-1][i]);
                }
            }
        }
    }


}
placeObjects(x, y, object, resultArray){
        var verticalSpacing = object === WorkstationVisual ? workstationSpacingVertical :
parallelSpacingVertical;


        //if even offset y to center parallel components
        if(this.numberOfElements % 2 === 0){
          y -= verticalSpacing/2;
        }
        //get first object y coordinate
        y -= (verticalSpacing)*Math.floor( (this.numberOfElements - 1)/2 );
        for (var i = 0; i < this.numberOfElements; i++){
          resultArray.push(new object(x,y, `${this.identifier} `));
          resultArray[i].identifier += resultArray[i].type + ` ${i+1}`;
          y += verticalSpacing;
          this.objectIndices[resultArray[i].identifier] = resultArray[i];
          resultArray[i].parallelBlock = this;
        }
      }
      setNext(drawableObject){
        this.next = drawableObject;
        if( drawableObject != null && !Array.isArray(this.containedElements[0])){
          for(var i = 0; i < this.containedElements.length; i++){
            this.containedElements[i].setNext(this.next);
          }
          this.next.previous = this;
        }
        else if(drawableObject != null){
          var lastIndex = this.containedElements.length - 1;
          if(drawableObject != null){
            for(var i = 0; i < this.containedElements[lastIndex].length; i++){
                this.containedElements[lastIndex][i].setNext(this.next);
            }
            this.next.previous = this;
        }
    }
}
findWidth(){
    if (Array.isArray(this.objectType)){
        var width = 0;
        this.objectType.forEach(element => {
            width += element.width;
        });
        return width + (this.objectType.length - 1) * horizontalSpacing;
    }
    else return this.objectType.width;
}
draw(){
    if(!this.isReversed){
        if(Array.isArray(this.objectType)){
            this.connectInteriorObjects();
            drawParallelstart(...this.containedElements[0]);
            this.containedElements.forEach(type => {
                type.forEach(element => {
                    element.draw();
                });
            });
            drawParallelRight(...this.containedElements[this.containedElements.length-1]);
        }
        else{
            drawParallelConnection(...this.containedElements);
            this.containedElements.forEach(element => {
                element.draw();
            });
        }
    }
    else{
        if(Array.isArray(this.objectType)){
            this.connectInteriorObjectsReversed();
            drawParallelRight(...this.containedElements[0]);
            this.containedElements.forEach(type => {
                type.forEach(element => {
                    element.draw();
                });
            });
            drawParallelstart(...this.containedElements[this.containedElements.length-1]);
        }
        else{
            drawParallelConnection(...this.containedElements);
            this.containedElements.forEach(element => {
                element.draw();
            });
        }
    }
}
placeTask(taskIdentifier, serverIdentifier){
    var destination = this.objectIndices[serverIdentifier]
    this.tasks[taskIdentifier] = destination;
}
```

```
acceptTask(task){
   if(selectedModel == "centralServer")
     task.setDestinationObject(this);
   else
      this.tasks[task.identifier].acceptTask(task);
}
receiveTask(task){
   var destination = this.tasks[task.identifier];
   destination.acceptTask(task);
}
advanceTask(task){
   this.next.acceptTask(task);
   if(selectedModel !== "workstations") //workstations has assigned tasks for specific stations
      delete this.tasks[task.identifier];
}
checkAvailability(task){
   return this.checkTask(task.identifier);
}
reverseDrawing(){
   var temp = this.coordinates.x;
   this.connections.start.x = this.connections.end.x;
   this.connections.end.x = temp;
   this.isReversed = true;

   for(var key in this.objectIndices){
      var currentObject = this.objectIndices[key];

      if(currentObject.next !== this.next){
         var nextObject = currentObject.next;
```

```
            currentObject.coordinates.x = nextObject.connections.end.x - currentObject.constructor.width;

            currentObject.connections.start.x = currentObject.coordinates.x;

            currentObject.connections.end.x = currentObject.connections.start.x +
currentObject.constructor.width;



            nextObject.coordinates.x = currentObject.coordinates.x - nextObject.constructor.width -
horizontalSpacing;

            nextObject.connections.start.x = nextObject.coordinates.x;

            nextObject.connections.end.x = nextObject.connections.start.x + nextObject.constructor.width;


            if(nextObject.type == "Queue"){

              nextObject.updateQueueSlotLocations();

            }

            if(nextObject.isReversed !== true)

              nextObject.reverseDrawing();

          }

          if(currentObject.type == "Queue"){

            currentObject.updateQueueSlotLocations();

          }

          if(currentObject.isReversed !== true)

            currentObject.reverseDrawing();

        }

    }

}


var taskColorChoices = [

    "red",
    "cyan",
    "brown",
    "green",
    "purple",
    "blue",
    "orange",
    "pink"
]
var colorIndex = 0;
function generateColor(){
    var colorChoice = taskColorChoices[colorIndex];
    colorIndex = ( (colorIndex + 1) % taskColorChoices.length);
    return colorChoice;
}
function createVisualTask(x,y, identifier){
    return new TaskVisual(x, y, identifier, generateColor());
}
```

visualModels.js

```
//Timer For Drawing Update
var timeElapsed = 0;
var currentModel = null;

function createTimer(updateFunction, rate = updateRate){
    return setInterval(updateFunction, rate);
}

function runModel(rate = updateRate){
    currentModel.timerRunning = true;
    currentModel.intervalID = createTimer(tickModel, rate);
}
function tickModel(){
    currentModel.tickOnce();
}
/******************************************************************
 *  Contains Functions to Create the Visual Component of Models
 ******************************************************************/
class ModelVisual {
    constructor(identifier){
        currentModel = this;
        this.identifier = identifier;
        this.components = {};
        this.tasks = {};
        this.events = [];
        this.drawnConnections = [];
        this.intervalID = null;
        this.timerRunning = false;
    }
    reset(){
        this.events = [];
        this.tasks = {};
        for(const key in this.components){
            this.components[key].reset();
        }
        timeElapsed = currentTime;
    }

    startTimer(){
        if(this.intervalID == null)
            this.intervalID = runModel(this);
    }
    stopTimer(){
        if(this.intervalID !== null){
            clearInterval(this.intervalID);
        }
        this.intervalID = null;
    }
    addComponent(visualObject){
        this.components[visualObject.identifier] = visualObject;
        if(visualObject.containedElements !== undefined){
            for(var i = 0; i < visualObject.containedElements.length; i++ ){
                if(Array.isArray(visualObject.containedElements[i])){
                    for(var j = 0; j < visualObject.containedElements[i].length; j++)
                        this.components[visualObject.containedElements[i][j].identifier] =
visualObject.containedElements[i][j];
                }
                else{
                    this.components[visualObject.containedElements[i].identifier] =
visualObject.containedElements[i];
                }
            }
        }
        if(visualObject.containedElements !== undefined){
            for(var i = 0; i < visualObject.containedElements.length; i++ ){


            }
        }
```

```
}
setNext(first, second){
  first.setNext(second);
}
connectQueue(object, queue){
  object.setConnection(queue);
}
addEvent(event){
  this.events.push(event);
}
addTask(task){
  this.tasks[task.identifier] = task;
}
addDrawnConnection(itemOne, itemTwo, connectFunction){
  this.drawnConnections.push({"arguments": [itemOne, itemTwo], "function": connectFunction});
}
callConnectionFunction(connection){
  connection.function(...connection.arguments);
}
removeTask(task){
  delete this.tasks[task.identifier];
}
processEvent(){
  if(this.events.length > 0){
    if(this.events[0].type == "accept"){
      this.events[0].object.acceptTask(this.events[0].task);
      this.events.shift();
    }
    else if (this.events[0].type == "advance"){
            this.events[0].object.advanceTask(this.events[0].task);
            this.events.shift();
        }
    }
}
tickOnce(){
    timeElapsed += timePerTick;
    if(this.events.length > 0 && this.events[0].time <= timeElapsed){
            if(this.events[0].type == "accept"){
                this.events[0].object.acceptTask(this.events[0].task);
                this.events.shift();
            }
            else if (this.events[0].type == "advance"){
                this.events[0].object.advanceTask(this.events[0].task);
                this.events.shift();
            }
    }
    this.draw();
}
createEvent(simulationObject, time, eventType, taskIdentifier = null){
    if(simulationMode !== true){
        var object, time, task = taskIdentifier;
        object = this.components[simulationObject.identifier];
        time = time;
        eventType = eventType;
        if(taskIdentifier != null){
            if(this.tasks[taskIdentifier] != undefined){
                task = this.tasks[taskIdentifier];
            }
            else {
                this.createVisualTask(taskIdentifier, object);
                task = this.tasks[taskIdentifier];
            }
        }
        var event = {
            "object": object,
            "time": time,
            "type": eventType,
            "task": task,
            "simulationObject": simulationObject
        }
        this.addEvent(event);
    }


}
createVisualTask(taskIdentifier, initialObject){
    var x = initialObject.connections.start.x;
    var y = initialObject.connections.start.y;
    this.addTask(createVisualTask(x,y, taskIdentifier));
    return this.tasks[taskIdentifier];
}
draw(){
    drawCanvas();
    //Draw Base Components
    for(const key in this.components){
        if(this.components.hasOwnProperty(key)){
            this.components[key].draw();
        }
        }
        //Draw Connections
        this.drawnConnections.forEach(connection => {
            this.callConnectionFunction(connection);
        });

        //Draw Tasks
        for (const key in this.tasks){
            if(this.tasks.hasOwnProperty(key)){
                this.tasks[key].draw();
            }
        }
    }
}
var xCenter = canvasWidth/2;
var yCenter = canvasHeight/2;

class SingleServerVisualModel extends ModelVisual{
    constructor(){
        super("Single Server");

        //Adjust Size
        if(numberSlotsInQueue > 20){
            canvasScalingX = (canvasWidth + queueSlotSize*(numberSlotsInQueue - 20))/canvasWidth;
            canvasWidth = canvasWidth * canvasScalingX;
        }

        let x = (xCenter -(QueueVisual.width + ServerVisual.width)/2) / 2;
        let y = canvasHeight/2 - ServerVisual.height/2;
        //Set Up Base Components
        let queue = new QueueVisual(x + horizontalSpacing,y,"Queue");
        let server = new ServerVisual(x + QueueVisual.width + 2*horizontalSpacing, y, "Server");
        let entryPoint = new EntrancePoint(x - horizontalSpacing, y, "Arrivals");
        let exitPoint = new ExitPoint(server.connections.end.x + 2*horizontalSpacing, y);

        //Add Components to the Model
        this.addComponent(entryPoint);
        this.addComponent(queue);
        this.addComponent(server);
        this.addComponent(exitPoint);

        //Link Components Together
        this.setNext(entryPoint, queue);
        this.setNext(queue, server);
        this.setNext(server, exitPoint);
        this.setNext(exitPoint, null);
        server.setConnection(queue);

        //Drawn Paths
        this.addDrawnConnection(entryPoint, queue, connectArrow);
        this.addDrawnConnection(queue, server, connectArrow);
        this.addDrawnConnection(server, exitPoint, connectArrow);
    }
}

class SingleFeedbackVisualModel extends ModelVisual{
    constructor(){
```

```
    super("Single Server With Feedback");


    //Adjust Size
    if(numberSlotsInQueue > 20){
      canvasScalingX = (canvasWidth + queueSlotSize*(numberSlotsInQueue - 20))/canvasWidth;
      canvasWidth = canvasWidth * canvasScalingX;
    }


    let drawX = (xCenter -(QueueVisual.width + ServerVisual.width)/2) / 2 + horizontalSpacing;
    let drawY = canvasHeight/2 + verticalSpacing/2;


    var startX =  (xCenter -(QueueVisual.width + ServerVisual.width)/2) / 2;
    var offset = 15;


    var serverOneQueue = new QueueVisual(drawX, drawY, "Queue");
    drawX += horizontalSpacing + QueueVisual.width;
    var serverOne = new FeedbackServerVisual(drawX,drawY, "Server");


    var boundaries = {
      "bottomRight" : new VisualAnchor( serverOne.connections.end.x - ServerVisual.width/2,
serverOne.connections.end.y - ServerVisual.height/2),
      "topRight" : new VisualAnchor(serverOne.connections.end.x - ServerVisual.width/2,
serverOne.connections.end.y - ServerVisual.height/2 - verticalSpacing),
      "topLeft" : new VisualAnchor(startX, serverOne.connections.end.y - verticalSpacing -
ServerVisual.height/2),
      "bottomLeft" : new VisualAnchor(startX, serverOneQueue.connections.start.y - offset),
      "exitConnection" : new ExitPoint(serverOne.connections.end.x + horizontalSpacing*3,
serverOne.connections.end.y)
    };
```

```
	var arrowUp = new VisualAnchor(boundaries.bottomRight.coordinates.x + 4,
serverOne.connections.end.y - verticalSpacing/2 - ServerVisual.height/2);
	var mergeBottom = new VisualAnchor(serverOneQueue.connections.start.x, drawY + offset);
	var mergeTop = new VisualAnchor(serverOneQueue.connections.start.x, drawY - offset);
	var entrance = new EntrancePoint(startX - horizontalSpacing, drawY + offset, "Arrivals");

	//Add Components
	this.addComponent(serverOneQueue);
	this.addComponent(serverOne);
	this.addComponent(entrance);
	this.addComponent(boundaries.exitConnection);
	serverOne.setConnection(serverOneQueue);

	//Set Next
	this.setNext(entrance, mergeBottom);
	this.setNext(mergeBottom, serverOneQueue);
	this.setNext(serverOneQueue, serverOne);
	this.setNext(serverOne, boundaries.bottomRight);
	serverOne.connectExit(boundaries.exitConnection);

	this.setNext(boundaries.bottomRight, boundaries.topRight);
	this.setNext(boundaries.topRight, boundaries.topLeft);
	this.setNext(boundaries.topLeft, boundaries.bottomLeft);
	this.setNext(boundaries.bottomLeft, mergeTop);
	this.setNext(mergeTop, serverOneQueue);

	//Drawn Connections
	this.addDrawnConnection(boundaries["bottomRight"], arrowUp, connectArrow);
	this.addDrawnConnection(boundaries["bottomLeft"], mergeTop, connectArrow);
```

```
        this.addDrawnConnection(entrance, mergeBottom, connectArrow);
        this.addDrawnConnection(serverOneQueue, serverOne, connectArrow);
        this.addDrawnConnection(boundaries["bottomRight"], boundaries["topRight"], connectLine);
        this.addDrawnConnection(boundaries["topRight"], boundaries["topLeft"], connectLine);
        this.addDrawnConnection(boundaries["topLeft"], boundaries["bottomLeft"], connectLine);
        this.addDrawnConnection(serverOne, boundaries["exitConnection"], connectArrow);
    }
}
class ParallelServersVisualModel extends ModelVisual{
    constructor(serverCount){
        //Adjust Size
        if(numberSlotsInQueue > 20){
            canvasScalingX = (canvasWidth + queueSlotSize*(numberSlotsInQueue - 20))/canvasWidth;
            canvasWidth = canvasWidth * canvasScalingX;
        }
        super("Parallel Servers");
        let x1 = 200;
        let y1 = 400;

        //Set up Components
        let queue = new QueueVisual(x1, y1, "Queue");
        let parallelServers = new ParallelContainer(x1 + horizontalSpacing + QueueVisual.width, y1,
"Parallel", ServerVisual, serverCount);
        let entryPoint = new EntrancePoint(queue.connections.start.x - horizontalSpacing*2, y1, "Arrivals");
        let exitPoint = new ExitPoint(parallelServers.connections.end.x + horizontalSpacing*2, y1);

        //Add Components to Model
        this.addComponent(entryPoint);
        this.addComponent(queue);
        this.addComponent(parallelServers);
        this.addComponent(exitPoint);

        //Link Components
        this.setNext(entryPoint, queue);
        this.setNext(queue, parallelServers);
        this.setNext(parallelServers, exitPoint);
        //this.setNext(exitPoint, null);
        parallelServers.connectQueue(queue);

        //Drawn Paths
        this.addDrawnConnection(entryPoint, queue, connectArrow);
        this.addDrawnConnection(queue, parallelServers, connectArrow);
        this.addDrawnConnection(parallelServers, exitPoint, connectArrow);

        //Adjust Size
    }

}
class TwoServersVisualModel extends ModelVisual{
    constructor(){
        super("Two Servers");
        let x1 = 400;
        let y1 = 400;

        //Set up Components
        let queue = new QueueVisual(x1, y1, "Queue");
        let twoServers = new ParallelContainer(x1 + horizontalSpacing + QueueVisual.width, y1, "Parallel",
ServerVisual, 2);
```

```
        let entryPoint = new EntrancePoint(queue.connections.start.x - horizontalSpacing*2, y1, "Arrivals");
        let exitPoint = new ExitPoint(twoServers.connections.end.x + horizontalSpacing*2, y1);

        //Add Components to Model
        this.addComponent(entryPoint);
        this.addComponent(queue);
        this.addComponent(twoServers);
        this.addComponent(exitPoint);

        //Link Components
        this.setNext(entryPoint, queue);
        this.setNext(queue, twoServers);
        this.setNext(twoServers, exitPoint);
        //this.setNext(exitPoint, null);
        twoServers.connectQueue(queue);

        //Drawn Paths
        this.addDrawnConnection(entryPoint, queue, connectArrow);
        this.addDrawnConnection(queue, twoServers, connectArrow);
        this.addDrawnConnection(twoServers, exitPoint, connectArrow);

        //Adjust Size
    }
}

class InteractiveVisualModel extends ModelVisual{
    constructor(numberWorkstations, numberServers){
        super("Interactive Workstations");
        //Coordinates
        var yMain = 450;
        var xStart = 50;
        var yTop = 100;

        var stationCoord = {"x": 200, "y": yMain};
        var queueCoord = {"x": stationCoord.x + WorkstationVisual.width + horizontalSpacing*2, "y":
yMain};
        var serverCoord = {"x": queueCoord.x + QueueVisual.width + horizontalSpacing, "y": yMain};
        var xEnd = serverCoord.x + horizontalSpacing*2 + ServerVisual.width;
        var visualAnchors = {};
        var anchorCoords = [
          { "name": "botLeft", "x": xStart, "y": yMain},
          { "name": "botRight", "x": xEnd, "y": yMain},
          { "name": "topRight", "x": xEnd, "y": yTop},
          { "name": "topLeft", "x": xStart, "y": yTop}
        ];

        anchorCoords.forEach(element => {
          var name = element.name;
          visualAnchors[name] = new VisualAnchor(element.x, element.y);
        });

        //Main Components
        var workstationsParallel = new ParallelContainer(stationCoord.x, stationCoord.y, "Parallel",
WorkstationVisual, numberWorkstations);
        var serversQueue = new QueueVisual(queueCoord.x, queueCoord.y, "Queue");
        var serversParallel = new ParallelContainer(serverCoord.x, serverCoord.y, "Servers", ServerVisual,
numberServers);
```

```
        //Add Components
        this.addComponent(workstationsParallel);
        this.addComponent(serversQueue);
        this.addComponent(serversParallel);

        //Link Components
        this.setNext(visualAnchors["botLeft"], workstationsParallel);
        this.setNext(workstationsParallel, serversQueue);
        this.setNext(serversQueue, serversParallel);
        this.setNext(serversParallel, visualAnchors["botRight"]);
        this.setNext(visualAnchors["botRight"], visualAnchors["topRight"]);
        this.setNext(visualAnchors["topRight"], visualAnchors["topLeft"]);
        this.setNext(visualAnchors["topLeft"], visualAnchors["botLeft"]);
        serversParallel.connectQueue(serversQueue);

        //Add Drawn Paths
        this.addDrawnConnection(workstationsParallel, serversQueue, connectArrow);
        this.addDrawnConnection(serversQueue, serversParallel, connectArrow);
        this.addDrawnConnection(visualAnchors["botLeft"], workstationsParallel, connectArrow);
        this.addDrawnConnection(serversParallel, visualAnchors["botRight"], connectLine);
        this.addDrawnConnection(visualAnchors["botRight"], visualAnchors["topRight"], connectLine);
        this.addDrawnConnection(visualAnchors["topRight"], visualAnchors["topLeft"], connectLine);
        this.addDrawnConnection(visualAnchors["topLeft"], visualAnchors["botLeft"], connectLine);

    }
}

class CentralServerVisualModel extends ModelVisual{
    constructor(numberDisks){
        super("Central Server");
        var coreCount = 4;
        //Adjust Size
        if(numberSlotsInQueue > 10 || true){
          canvasScalingX = (canvasWidth + 2*queueSlotSize*(numberSlotsInQueue - 10))/canvasWidth;
          canvasWidth = canvasWidth * canvasScalingX + horizontalSpacing;
        }
        var cpuHeight = (parallelSpacingVertical/2+ serverRadius)*coreCount;
        canvasWidth = canvasWidth + 500;
        var diskHeight = numberDisks * (parallelSpacingVertical) ;
        var diskWidth = QueueVisual.width + horizontalSpacing + 2*parallelSpacingHorizontal +
ServerVisual.width;
        var verticalGap = verticalSpacing/2;
        var modelHeight = diskHeight + verticalGap + cpuHeight;
        var modelWidth = 2*QueueVisual.width + 6*horizontalSpacing  + 2*ServerVisual.width;

        //Coordinates
        var middleY = canvasHeight/2;
        var middleX = canvasWidth/2;
        var xStart = middleX - modelWidth/2;
        var xEnd = middleX + modelWidth/2;
        var yTop = middleY - modelHeight/2 + cpuHeight/2;
        var yBot = middleY + modelHeight/4;

        //DISK
        var disksY = yBot;
        var disksX = xStart + diskWidth/2;
        //Channel
        var channelX = xEnd - horizontalSpacing - ServerVisual.width;
        var channelY = yTop;
        var channelQueueX = channelX - horizontalSpacing - QueueVisual.width;
        var channelQueueY = yTop;


        //CPU
        var centralQueueX = xStart + horizontalSpacing;
        var centralQueueY = yTop;
        var centralServerX = xStart + horizontalSpacing*2 + QueueVisual.width;
        var centralServerY = yTop;


        //Anchors
        var topRight = new VisualAnchor(xEnd, yTop);
        var bottomRight = new VisualAnchor(xEnd, yBot);
        var topLeft = new VisualAnchor(xStart, yTop);
        var bottomLeft = new VisualAnchor(xStart, yBot);
        var bottomArrow = new VisualAnchor( disksX  + diskWidth +(xEnd - disksX - diskWidth)/2, yBot);
        //Components
        var disksParallel = new ParallelContainer(disksX, disksY, "Parallel", [QueueVisual, DiskVisual],
numberDisks);
        var centralQueue = new QueueVisual(centralQueueX, centralQueueY, "Processor Queue");


        var centralProcessor = new ParallelContainer(centralServerX, centralServerY, "Central Processor",
ServerVisual, 4)
        //var centralProcessor = new ServerVisual(centralServerX, centralServerY, "Central Processor");
        var channelQueue = new QueueVisual(channelQueueX, channelQueueY, "Channel Queue");
        var channel = new ServerVisual(channelX, channelY, "Channel");
```

```
//Reverse Top Components
disksParallel.reverseDrawing();

//Add Components to Model
this.addComponent(disksParallel);
this.addComponent(centralQueue);
this.addComponent(centralProcessor);
this.addComponent(channelQueue);
this.addComponent(channel);

//Set Next
this.setNext(topLeft, centralQueue);
this.setNext(centralQueue, centralProcessor);
this.setNext(centralProcessor, channelQueue);
this.setNext(channelQueue, channel);
this.setNext(channel, topRight);
this.setNext(topRight, bottomRight);
this.setNext(bottomRight, disksParallel);
this.setNext(disksParallel, bottomLeft);
this.setNext(bottomLeft, topLeft);

//Connect Queues
centralProcessor.connectQueue(centralQueue);

channel.setConnection(channelQueue);

//Drawn Connections
this.addDrawnConnection(topLeft, centralQueue, connectArrow);
this.addDrawnConnection(centralQueue, centralProcessor, connectArrow);
```

```
        this.addDrawnConnection(centralProcessor, channelQueue, connectArrow);
        this.addDrawnConnection(channelQueue, channel, connectArrow);
        this.addDrawnConnection(channel, topRight, connectArrow);
        this.addDrawnConnection(topRight, bottomRight, connectLine);
        this.addDrawnConnection(bottomRight, disksParallel, connectArrowReverse);
        this.addDrawnConnection(disksParallel, bottomLeft, connectLine);
        this.addDrawnConnection(bottomLeft, topLeft, connectLine);
    }
}
```

driver.html

```
<!DOCTYPE html>
<html>
<head>
  <link rel="stylesheet" href="../css/styles.css">
</head>
<body>
  <button class="header">Queueing Network Animation System</button>

  <div class="runtime-block">
    <canvas id="window" class="screen"></canvas>
    <div id="navigationMenu" class="navigation-menu">Menu</div>
  </div>

  <div id="resultBlock" class="results">
    <div id="titleOutput">
      <table id="titleTable"> </table>
    </div>
    <div id="resultsOutput">
      <table id="resultsTable"> </table>
    </div>
  </div>



</body>
  <script src="../Draw/config.js"></script>
  <script src="../Draw/draw.js"></script>
  <script src="../Draw/drawable.js"></script>
  <script src="../Draw/visualModels.js"></script>
  <script src="../Simulation/simulation.js"></script>
  <script src="../Simulation/simulationModels.js"></script>
  <script src="driver.js"></script>
  <script src="runtime.js"></script>
  <script> initializeCanvas(); </script>
</html>
```

driver.js

```
/////////////////////////////////////////////////////////////////
// driver.js
// The interface to run the simulation and update the visual
/////////////////////////////////////////////////////////////////

/************************************************
* TIMER AND RUNNING COMPONENTS
************************************************/
var currentModel = null;
var currentSimulation = null;
var lock = false;
var timeNextEvent;
var tempRate = 1;
var speedModifier = 1;
var stepMode = false;
var simulationMode = false;
function displayError(){
   stopModel();
   alert("Error: " + error);
}
function increaseSpeed(){
   speedModifier += .1;
   updateRate = defaultUpdateRate * speedModifier;
   stopModel();
   runModel();
}
function decreaseSpeed(){
   if(speedModifier >= .2){
      speedModifier = speedModifier - .1;
   }
   updateRate = defaultUpdateRate * speedModifier;
   stopModel();
   runModel();
}
function resetModel(){
   location.reload();
}

function stopModel(){
   clearInterval(currentModel.intervalID);
   currentModel.timerRunning = false;
```

```
    currentModel.intervalID = null;

}

function createTimer(updateFunction, rate = updateRate){

    return currentModel.intervalID = setInterval(updateFunction, tempRate);

}

function singleEvent(){

    currentSimulation.processEvent();

}

function runModel(rate = updateRate){

    stopModel();

    stepMode = false;

    simulationMode = false;

    currentModel.timerRunning = true;

    currentModel.intervalID = createTimer(tickModel, rate);

    printResults();

}

function tickModel(){

    timeNextEvent = currentSimulation.getNextEventTime();

    if(timeNextEvent !== null && timeElapsed >= timeNextEvent){

        currentSimulation.processEvent();

        //console.log(currentModel.events[0].object.identifier + "|" +
currentModel.events[0].task.identifier + "|" + currentModel.events[0].type);

        printResults();

    }

    currentModel.tickOnce();


    if(error) displayError();

}

function startStepOnceMode(){
    stepMode = true;
    stopModel();
    currentSimulation.copyAllTasks();

    while(currentModel.events.length < 1)
        currentSimulation.processEvent();
    currentModel.processEvent();
    timeNextEvent = currentSimulation.getNextEventTime();
    elapsedTime = timeNextEvent;
    currentModel.intervalID = createTimer(stepOnce, updateRate);
}
function stepOnce(){
    currentModel.tickOnce();
    printResults();
    if(error) displayError();
}
function stopStepOnceMode(){
    stepMode = false;
    stopModel();
    currentSimulation.copyAllTasks();
    timeElapsed = currentSimulation.getNextEventTime();
}
function beginSimulationMode(){
    stopModel();
    simulationMode = true;
    currentModel.intervalID = createTimer(simulate, 1/1000);
}
function turboMode(){
    stopModel();
    currentModel.intervalID = createTimer(simulate, updateRate);
}
function simulate(){
    for(var i = 0; i < 1000; i++){
        timeNextEvent = currentSimulation.getNextEventTime();
        currentSimulation.processEvent();
    }
    timeElapsed = currentSimulation.getNextEventTime();
    printResults();
}
/*
function endSimulationMode(){
    stopModel();
    currentSimulation.copyAllTasks();
    simulationMode = false;
}
*/
/***********************************************
* OUTPUT PRINTING
***********************************************/
var titleLocation, resultsLocation;
function setPrintOutputLocation(titleLocationID, resultLocationID){
    titleLocation = document.getElementById(titleLocationID);
    resultsLocation = document.getElementById(resultLocationID);

    if(titleLocation === undefined)
        console.log("Invalid title output location");

    if(resultsLocation === undefined)
```

```
        console.log("Invalid results output location");

}
function printDebugResults(){
    //titleLocation.innerHTML = currentSimulation.toTable();
    resultsLocation.innerHTML = currentSimulation.componentsToTable();
}
function printResults(){
    resultsLocation.innerHTML = currentSimulation.displayResults();
}

/**********************************************
* Creation methods
**********************************************/
function createSingleServer(arrivalDistribution, serviceDistribution){
    currentSimulation= new SingleServerModel(arrivalDistribution, serviceDistribution);
    currentModel = new SingleServerVisualModel();
    currentSimulation.connectVisualModel(currentModel);
    drawCanvas();
    currentModel.draw();
    runModel(updateRate);
}
function createSingleFeedback(arrivalDistribution, serviceDistribution, feedbackRate){
    currentSimulation= new SingleFeedbackModel(arrivalDistribution, serviceDistribution, feedbackRate);
    currentModel = new SingleFeedbackVisualModel();
    currentSimulation.connectVisualModel(currentModel);
    drawCanvas();
    currentModel.draw();
    runModel(updateRate);
```

```
}

function createTwoServers(arrivalDistribution, serviceDistributions){
   currentSimulation= new TwoServersModel(arrivalDistribution, serviceDistributions);
   currentModel = new TwoServersVisualModel();
   currentSimulation.connectVisualModel(currentModel);
   drawCanvas();
   currentModel.draw();
   runModel(updateRate);
}

function createParallelServers(arrivalDistribution, serviceDistributions, serverCount){
   currentSimulation = new ParallelServersModel(arrivalDistribution, serviceDistributions, serverCount);
   correntModel = new ParallelServersVisualModel(serverCount);
   currentSimulation.connectVisualModel(currentModel);
   drawCanvas();
   currentModel.draw();
   runModel(updateRate);
}

var workstationCount = 6;
var serverCount = 6;

var w1 = new Distribution("uniform", {"lowerBound": 6, "upperBound": 24});
var w2 = new Distribution("uniform", {"lowerBound": 10, "upperBound":15});

function createWorkstations(workstationDistribution, workstationCount, serverDistribution,
serverCount){
   currentSimulation= new InteractiveModel(workstationDistribution, workstationCount,
serverDistribution, serverCount);
   currentModel = new InteractiveVisualModel(workstationCount, serverCount);
   currentSimulation.connectVisualModel(currentModel);
   currentSimulation.addInitialVisualEvents();
   drawCanvas();
   currentModel.draw();
   runModel(updateRate);
}

function createCentralServer( centralDistribution, channelDistribution, diskDistributions, diskCount,
jobCount, diskWeights){
   currentSimulation= new CentralServerModel(centralDistribution, channelDistribution,
diskDistributions, diskCount, jobCount, diskWeights);
   currentModel = new CentralServerVisualModel(diskCount);
   currentSimulation.connectVisualModel(currentModel);
   currentSimulation.addInitialEvents();
   drawCanvas();
   currentModel.draw();
   runModel(updateRate);
}

function draw(){
   currentModel.draw();
}

function onClickPause(){

}
function onClickResume(){
}
function onClickSpeedUp(){

}
function onClickSpeedDown(){

}

//Distribution Templates
var lowerBound = 0;
var upperBound = 1;
var constantValue = 1;
var exponentialMean = 5;

var uniformDistribution = new Distribution("uniform", {"lowerBound": lowerBound, "upperBound":
upperBound});
var constantDistribution = new Distribution("constant", {"value": constantValue});
var exponentialDistribution = new Distribution("exponential", {"mean": exponentialMean});

/////////////////////////////////////
// TWO SERVERS
////////////////////////////////////
var twoServersUniformDistribution = new Distribution("uniform", {"lowerBound": lowerBound,
"upperBound": upperBound});
var twoServersConstantDistribution = new Distribution("constant", {"value": constantValue});
var twoServersExponentialDistribution = new Distribution("exponential", {"mean": exponentialMean});

var twoServersArrivalDistribution = twoServersUniformDistribution;
var twoServersServerDistribution = twoServersUniformDistribution;


////////////////////////////////////
//SINGLE FEEDBACK
////////////////////////////////////
var feedbackLowerBound = 0;
var feedbackUpperBound = 5;
var feedbackConstantValue = 5;
var feedbackExponentialMean = 5;
var feedbackUniformDistribution = new Distribution("uniform", {"lowerBound": 0, "upperBound": 5});
var feedbackConstantDistribution = new Distribution("constant", {"value": constantValue});
var feedbackExponentialDistribution = new Distribution("exponential", {"mean": exponentialMean});


var singleFeedbackServerDistribution = new Distribution("uniform", {"lowerBound": 0, "upperBound":
4});
var singleFeedbackArrivalDistribution = feedbackUniformDistribution;
var feedbackReturnProbability = 0.3;


//////////////////////////////
// Workstations
////////////////////////////////
var workstationCount = 5;
var workstationDistribution = new Distribution("uniform", {"lowerBound": 0, "upperBound": 5});


var serverCount = 3;
var serverDistribution = new Distribution("uniform", {"lowerBound": 0, "upperBound": 4});



////////////////////////////////
```

```
// Central Server
//////////////////////////////
var centralDistribution = new Distribution("uniform", {"lowerBound": 0, "upperBound": 4});
var channelDistribution = new Distribution("uniform", {"lowerBound": 0, "upperBound": 4});
var diskDistributions = [
    new Distribution("uniform", {"lowerBound": 0, "upperBound": 4}),
    new Distribution("uniform", {"lowerBound": 0, "upperBound": 4}),
    new Distribution("uniform", {"lowerBound": 0, "upperBound": 4}),
    new Distribution("uniform", {"lowerBound": 0, "upperBound": 4})
];
var diskCount = 4;
```

help.html

```
<!DOCTYPE html>
<html>
<head>
  <link rel="stylesheet" href="../css/styles.css">
</head>


<body>
<script src="../Scripts/header.js"></script>
<div class="help-main-block">
  <div class= "help-navigation">
    <div id= "general-button-title" class= "help-button-title"> General Information </div>
    <button id= "about-button" class= "help-navigation-button"> About </button>
    <button id= "terminology-button" class= "help-navigation-button"> Terminology </button>
    <button id= "program-description-button" class= "help-navigation-button"> Program Description
</button>
```

```
    <div class="help-navigation-spacing"> </div>

    <div id= "server-button-title" class= "help-button-title"> Available Models </div>

    <button id= "single-server" class= "help-navigation-button">Single Server</button>

    <button id= "single-feedback" class= "help-navigation-button">Single Feedback</button>

    <button id= "parallel-servers" class= "help-navigation-button">Parallel Servers</button>

    <button id= "interactive-workstations" class= "help-navigation-button">Interactive
Workstations</button>

    <button id= "central-server" class= "help-navigation-button">Central Server</button>


    <div class="help-navigation-spacing"> </div>

    <div id= "help-title" class= "help-button-title"> Usage Help </div>

    <button id= "model-selection" class= "help-navigation-button">Model Selection</button>

    <button id= "model-configuration" class= "help-navigation-button">Model Configuration</button>

    <button id= "model-runtime" class= "help-navigation-button">Runtime</button>

  </div>


  <div id="description-block" class = "help-description"></div>

</div>


<script src="../Scripts/help.js"></script>

</body>

</html>
```

modelConfiguration.html

```
<!DOCTYPE html>

<html>

<head>

  <link rel="stylesheet" href="../css/styles.css">

</head>
```

```
<body>
  <script src="../Scripts/header.js"></script>
  <div id = "model-title" class="model-title"> </div>
  <div id="configuration-body" class="configuration-body"> </div>
  <script src="../Scripts/modelConfiguration.js"></script>
</body>
</html>
```

configurationReader.js

```
//Starting point of program
setPrintOutputLocation("titleTable","resultsTable");

var selectedModel = getCookie("selectedModel");
if(selectedModel == "" || selectedModel == undefined || selectedModel == null){
  window.location.href = "../Pages/modelSelection.html";
}

numberSlotsInQueue = getCookie("queue-size");
if(selectedModel == "workstations"){
  numberSlotsInQueue = getCookie("workstation-count");
  if(numberSlotsInQueue < 4) numberSlotsInQueue = 4;
}
QueueVisual.width = queueSlotSize * numberSlotsInQueue;
function getCookie(key) {
    var name = key + "=";
    var decodedCookie = decodeURIComponent(document.cookie);
    var cookieArray = decodedCookie.split(';');
    for(var i = 0; i < cookieArray.length; i++) {
      var cookie = cookieArray[i];
      while (cookie.charAt(0) == ' ') {
        cookie = cookie.substring(1);
      }
      if (cookie.indexOf(name) == 0) {
        return cookie.substring(name.length, cookie.length);
      }
    }
    return "";
}
function getFloatCookie(key){
  var name = key + "=";
  var decodedCookie = decodeURIComponent(document.cookie);
  var cookieArray = decodedCookie.split(';');
  for(var i = 0; i < cookieArray.length; i++) {
    var cookie = cookieArray[i];
    while (cookie.charAt(0) == ' ') {
      cookie = cookie.substring(1);
    }
    if (cookie.indexOf(name) == 0) {
      return parseFloat(cookie.substring(name.length, cookie.length));
    }
  }
  return "";

}
function getDistribution(componentName, distributionType){
  var parameters = {};
  switch(distributionType){
    case "constant":
      parameters.value = getFloatCookie(`${componentName}-constant-input`);
      break;
    case "uniform":
      parameters.lowerBound = getFloatCookie(`${componentName}-uniform-lower`);
      parameters.upperBound = getFloatCookie(`${componentName}-uniform-upper`);
      break;
    case "exponential":
      parameters.mean = getFloatCookie(`${componentName}-exponential-input`);
      break;
  }
  return new Distribution(distributionType, parameters);
}
function runSingleServer(){
  var serviceType = getCookie("server-distribution-type");
  var arrivalType = getCookie("arrival-distribution-type");
  var serviceDistribution = getDistribution("server", serviceType);
  var arrivalDistribution = getDistribution("arrival", arrivalType);
  createSingleServer(arrivalDistribution, serviceDistribution);
  initializeCanvas();
}
function runParallelServers(){
  var arrivalType = getCookie("arrival-distribution-type");
  var arrivalDistribution = getDistribution("arrival", arrivalType);
  var serverCount = getFloatCookie("server-count");
  var serviceType = getCookie("parallel-servers-distribution-type");
  var serviceDistributions = getDistribution("parallel-servers",serviceType);
  createParallelServers(arrivalDistribution, serviceDistributions, serverCount);
  initializeCanvas();
}
```

```
function runSingleFeedback(){
  var serviceType = getCookie("server-distribution-type");
  var arrivalType = getCookie("arrival-distribution-type");
  var serviceDistribution = getDistribution("server", serviceType);
  var arrivalDistribution = getDistribution("arrival", arrivalType);
  var feedbackRate = getFloatCookie("feedback-return-rate");
  createSingleFeedback(arrivalDistribution, serviceDistribution, feedbackRate);
  initializeCanvas();
}
function runWorkstations(){
  var workstationCount = getCookie("workstation-count");
  var workstationType = getCookie("workstation-distribution-type");
  var workstationDistribution = getDistribution("workstation", workstationType);

  var serverCount = getCookie("server-count");
  var serverType = getCookie("processor-distribution-type");
  var serverDistribution = getDistribution("processor", serverType);

  createWorkstations(workstationDistribution, workstationCount, serverDistribution, serverCount);
  initializeCanvas();
}
function runCentral(){
  var centralType = getCookie("cpu-distribution-type");
  var centralDistribution = getDistribution("cpu", centralType);

  var channelType = getCookie("channel-distribution-type");
  var channelDistribution = getDistribution("channel", channelType);

  var diskCount = getCookie("disk-count");
  var diskDistributions = [];
  var diskWeights = [];
  var jobCount = getCookie("job-count");


  for(var i = 1; i <= diskCount; i++){
    var diskType = getCookie(`disk-${i}-distribution-type`);
    diskDistributions.push(getDistribution(`disk-${i}`, diskType));


    var diskWeight = getCookie(`disk-${i}-weight`);
    diskWeights.push(diskWeight);
  }
  createCentralServer(centralDistribution, channelDistribution, diskDistributions, diskCount, jobCount,
diskWeights);
  initializeCanvas();
  console.log(currentModel);
  console.log(currentSimulation);
}
switch(selectedModel){
    case "singleServer":
        runSingleServer();
        break;
    case "parallelServers":
        runParallelServers();
        break;
    case "singleFeedback":
        runSingleFeedback();
        break;
    case "workstations":
        runWorkstations();
        break;
    case "centralServer":
        runCentral();
        break;
}
```

header.js

```
document.body.innerHTML += `<button id="title" class="header" >QAS</button>`;


var titleButton = document.getElementById("title");
titleButton.onclick = function(){
    window.location.href = "../index.html";
}
```

help.js

```
var descriptionDivision = document.getElementById("description-block");


function placeNavigationButtons(){
    var destination = document.getElementById("help-navigation");
}


function writeDescription(identifier, descriptionText){
    document.getElementById(identifier).onclick = function(){
        descriptionDivision.innerHTML = `<div class = "help-description-inner-body"> ${descriptionText}
</div>`;
    };
}


const aboutDescription =`
    <div class = "help-description-title"> About </div>
```

```
   Welcome to QAS! A visualization tool for queueing networks.


   </br></br>
   <b> Purpose: </b> </br>
   This program was written to help develop a greater understanding of queueing theory.
   The intention is that configuring and visualizing these systems will grant a firmer grasp on the
   underlying concepts. This program may be used to obtain analytical results and experimentation.


   </br></br>
   <b> What it does: </b> </br>
   The program simulates various queueing systems and provides colorful animations as jobs traverse
through
   the selected system. Parameters for selected model are fully adjustable.
   Calculation and readouts occur for relevant metrics of the given system such as response time,
   utilizations, and average service times. Animation speeds are variable with an option to forgo
animation
   completely for faster computations.


   </br></br>
   <b> Usage: </b>
   </br> 1) Select a Model
   </br> 2) Set the Configuration
   </br> 3) Run the Model



   </br> </br> </br>
   Explore, have fun, and learn!
`;
```

```
const programDescription = `

    <div class = "help-description-title"> Simulation: </div>

    This program is an event-driven simulator. The simulator portion will repeatedly process the next
event to occur

    and does not directly correspond to real passing time. This eliminates errors created by system
scheduling or machine

    differences.


    </br> </br>

    There is a simulation time initialized at zero. The system is initialized corresponding to the user-set
parameters

    and events for the various elements in the system are organized by time of occurrence.

    As events occur in the system the time is forwarded to the time of the next event and any new events
that must be created as a response

    are then pushed onto the event queue. This process is repeated, and results are calculated as the
events are processed.


    </br> </br> </br>

    <div class = "help-description-title"> Visualization: </div>

    Visualization occurs based on real time. The update rate set changes the speed that items occur on
screen-- this can be thought of as

    scaled time. When the time reaches the time the next simulation event is set to occur it the
visualization component will message the

    simulator and tell it to advance the event and then retrieve the time of the next event to occur. The
simulation does not need to do

    any work until that time is reached. This time is modular and may be slowed down, sped up, paused,
or even forgone completely in favor of pure simulation for faster result

    calculations.


    </br></br></br>

    <div class = "help-description-title"> Interactions: </div>
    Both the simulation and visualization components contain representations of the models broken into
their components such as arrivals,

    service components, jobs, queues, and anything else relevant to the given system. These
representations are not the same.

    The simulation component is responsible for calculations and the construction and processing of
events. The visualization component

    knows nothing of this, only the results of event that requires visualization. Each element is drawn and
updated as it moves throughout

    the system. Likewise, the simulator does not know the on screen position of the visual
representations. These components do communicate

    with each other, but the simulation component is not dependent on the visualization for core
functionality.
`

const terminologyDescription =`

    <div class = "help-description-title"> Terminology: </div>


    <b> Queue: </b> A container for jobs be processed stored in a retrievable and definite order. </br>

    <b> Job / Task: </b> A singular component requesting service. </br>

    <b> Server: </b> An element which processes a job. </br>

    <b> Arrival Time: </b> The time at which a job enters the starting point of a system. </br>

    <b> Interarrival Time: </b> The time between two consecutive arrivals. </br>

    <b> Response Time: </b> The time it takes for a job completes one cycle of processing within a
system. </br>

    <b> Server Utilization: </b> The fraction of time a service element is busy over total time. </br>

    <b> Throughput: </b> Average time for a job to complete one cycle per unit of time. </br>

    <b> Open System: </b> A system allowing for arrivals into the system and departures with no chance
of

        returning after a jov is processed. </br>

    <b> Closed System: </b> A system where jobs may return for further processing after departure.
</br>

    <b> Distribution: </b> The spread of a component based on mathematical formulae. </br>

    <b> Distribution Types: </b>
```

```
        </br> \u00a0\u00a0\u00a0\u00a0 M- Markov (Poisson, exponential)
        </br> \u00a0\u00a0\u00a0\u00a0 D- Deterministic (Constant)
        </br> \u00a0\u00a0\u00a0\u00a0 U- Uniform (Evenly spread between two points)
        </br> \u00a0\u00a0\u00a0\u00a0 G- General
        </br> \u00a0\u00a0\u00a0\u00a0 GI- General Independent
`;

const singleServerDescription = `
    <div class = "help-description-title"> Single Server Model: </div>
    The single server model is the simplest of models. It consists of a single queue and a
    single processor. Jobs arrive, are enqueued, processed at the server then exit the system.

    </br></br>
    <b> Adjustable Parameters: </b> </br>
    -Arrival Distribution Rate
    </br>-Service Distribution Rate
    </br>-Number of Visible Queue Slots
`;

const singleFeedbackDescription = `
    <div class = "help-description-title"> Single Server with Feedback Model: </div>
    The single server with feedback model contains arrivals, a single queue, and a single server.
    This model has a feedback loop. Jobs are allowed to return back into the system instead of leaving
    for further processing. The chances of returning to the system may be defined as a configurable
percentage.

    </br></br>
    <b> Adjustable Parameters: </b> </br>
    -Arrival Distribution Rate
    </br>-Service Distribution Rate
    </br>-Feedback Percentage
    </br>-Number of Visible Queue Slots
`;
const parallelServersDescription = `
    <div class = "help-description-title"> Parallel Servers Model: </div>
    The parallel servers model expands upon the single server model by adding additional
    servers in parallel. This model consists of a single queue with multiple servers. Tasks from the queue
    are placed into one of the available servers for processing.

    </br></br>
    <b> Adjustable Parameters: </b> </br>
    -Number of Servers
    </br>-Arrival Distribution Rate
    </br>-Service Distribution Rate
    </br>-Number of Visible Queue Slots
`;
const interactiveWorkstationsDescription = `
    <div class = "help-description-title"> Interactive Workstations Model: </div>
    The interactive workstation model possesses serveral workstations in parallel, which are then routed through a queue
    into a number of parallel servers for processing. Each workstation job is unique to that specific workstation and will
    return there once processed. This can be thought of as a user submitting data or interacting-- the user needs think time
    to fill out or prepare the submission, which upon processing is returned back to the user who can then do further interaction.

    </br></br>
    <b> Adjustable Parameters: </b> </br>
    -Number of Servers
    </br>-Number of Workstations
    </br>-Workstation Distribution Rates
    </br>-Processor Distribution Rates
`;
const centralServerDescription = `
    <div class = "help-description-title"> Central Server Model: </div>
    The central server model contains one central processor and a number of jobs already present within
the system. The jobs
    are forwarded from the processor queue into the processor and then into a channel queue and
channel. The channel directs
    the flow of jobs to various servers in parallel. Each of these contains their own corresponding queue,
distributions, and may
    be given a weight representitive of the odds of being directed to that particular element.


    </br></br>
    <b> Adjustable Parameters: </b> </br>
    -Number of Disks / Servers
    </br>-Number of Initial Jobs
    </br>-Central Processor Distribution Rate
    </br>-Channel Distribution Rate
    </br>-Distributions rates of the Individual Disks / Servers
    </br>-Number of Queue Slots
`;


const modelSelectionDescription = `
    <div class = "help-description-title"> Model Selection: </div>
    In the model selection page it is possible to choose between one of five models: Single Server, Single
Server with Feedback,
    Parallel Servers, Interactive Workstations, or Central Server. To move on and begin configuration
simply click on one of the models.

    To return to the main menu the header "QAS" in the top left corner may be clicked.


`;

const modelConfigurationDescription = `

    <div class = "help-description-title"> Model Configurations: </div>

    Configurations are unique to the specific model chosen. Fill out the fields as desired-- most invalid
inputs will automatically be incapable of

    entry. This is not possible for all fields, but if the run button is pressed with invalid inputs an error will
be displayed. After being

    satisfied with the configuration options select the run button to see the model in action. Alternatively,
if a different model is desired

    the select model button may be pressed.


    </br></br>

    <div class = "help-description-title"> Input: </div>

    Invalid inputs for most components will not be possible. For some this is not possible to regulate, but
a notification will occur

    explaining what fields are invalid if the run button is pressed.

    </br></br>


    <b>Distribution Rates:</b>

    </br> The distribution configurations contain the distribution type and values for that given
distribution.

    </br>The choices here are:

    </br>\u00a0\u00a0\u00a0\u00a0 -Constant: a set unchanging value

    </br>\u00a0\u00a0\u00a0\u00a0 -Uniform: a random value between the lower and upper bound,
evenly distributed

    </br>\u00a0\u00a0\u00a0\u00a0 -Exponential: Poisson Distribution, with a configurable mean.
    </br></br><b> Count or Number Input: </b>

    </br>  Various models have numerical inputs such as server count or amount of queue slots visible.
These components have a valid

    range indicated and cannot be set outside of this range.

    </br></br>




`;

const modelRuntimeDescription = `

    <div class = "help-description-title"> Runtime: </div>


    <b>Navigation Buttons:</b>

    </br> The navigation buttons may be seen in the top of the page above the model. They are "Select
Model", "Change Parameters", and

    "Restart Current Model". The select model button returns to the model selection page. The change
parameters button will take you back

    to the configuration page for the model that is currently displayed. The restart model button will clear
the model, results, and begin

    freshly with the same parameters


    </br></br>

    <b>Speed Controls:</b>

    </br>The Speed controls can be found at the bottom of the page below the shown model. The choices
here are related to the speed at which the

    model will run. Increase speed, pause, and decrease speed behave as one would expect and modify
the running speed or momentarily pause

    the simulation. The maximum speed button forgoes animations completely in order to run
significantly faster calculations. The single step

    option will pause the model then process a single event every time the button is pressed.


    </br></br>
    <b>Results:</b>
    </br>Results for the shown model will appear on the right-hand side of the screen. These are
organized by components and show
    relevant results.
`;


writeDescription("about-button", aboutDescription);
writeDescription("terminology-button", terminologyDescription);
writeDescription("program-description-button", programDescription);
writeDescription("single-server", singleServerDescription);
writeDescription("single-feedback", singleFeedbackDescription);
writeDescription("parallel-servers", parallelServersDescription);
writeDescription("interactive-workstations", interactiveWorkstationsDescription);
writeDescription("central-server", centralServerDescription);
writeDescription("model-selection", modelSelectionDescription);
writeDescription("model-configuration", modelConfigurationDescription);
writeDescription("model-runtime", modelRuntimeDescription);



descriptionDivision.innerHTML = `<div class = "help-description-inner-body"> ${aboutDescription}
</div>`;
```

modelConfiguration.js

```
var body = document.getElementById("configuration-body");
var selectedModel = getCookie("selectedModel");
if(selectedModel == "" || selectedModel == undefined || selectedModel == null){
    window.location.href = "../Pages/modelSelection.html";
}
var inputs = {};
var distributionSectionNames = {};
```

```
//Default Values
var defaultConstantDistributionValue = 200;
var defaultUniformLower = 100;
var defaultUniformUpper = 400;
var defaultExponentialValue = 300;
var defaultInteractiveWorkstations = 4;
var defaultInteractiveServers = 3;
var defaultInteractiveServers = 3;
var defaultInteractiveWorkstations = 4;

//Validation Boundaries
var validations = {
  "minServerCount": 1,
  "maxServerCount": 8,
  "minWorkstationCount": 1,
  "maxWorkstationCount": 9,
  "maxDiskCount": 8,
  "minDiskCount": 1,
  "minQueueSize": 3,
  "maxQueueSize": 30,
  "minFeedbackRate": 0,
  "maxFeedbackRate": 100,
  "minJobCount": 1,
  "maxJobCount": Infinity
};

//Detailed Default Parameters For Specific Models
var defaultParameters = {
```

```
"singleServer": {
   "queue-size": 16,
   "server-distribution-type": "exponential",
   "server-exponential-input": 100,
   "arrival-distribution-type": "exponential",
   "arrival-exponential-input": 200
},
"parallelServers": {
   "queue-size": 15,
   "parallel-servers-distribution-type": "exponential",
   "parallel-servers-exponential-input": 100,
   "arrival-distribution-type": "exponential",
   "arrival-exponential-input": 200,
   "parallel-servers-count": 4,
   "server-count": 4
},
"workstations": {
   "queue-size": 4,
   "server-count": 3,
   "workstation-count": 4,
   "max-workstations": 9,
   "max-servers": 8,
   "workstation-distribution-type": "exponential",
   "workstation-exponential-input": 400,
   "processor-distribution-type": "exponential",
   "processor-exponential-input": 200,
},
"singleFeedback":{
   "queue-size": 16,
        "feedback-return-rate": 40,
        "server-distribution-type": "exponential",
        "server-exponential-input": 200,
        "arrival-distribution-type": "exponential",
        "arrival-exponential-input": 600
    },
    "centralServer":{
        "queue-size": 10,
        "disk-count": 3,
        "job-count": 50,
        "core-count": 4,
        "cpu-distibution-type": "exponential",
        "cpu-exponential-input": 300,
        "channel-distribution-type": "exponential",
        "channel-exponential-input": 200,
        "disk-weight": 1,
        "disk-distribution-type": "constant",
        "disk-constant-input" : 200
    }
}
function setDefaultParameters(modelIdentifier){
    for( const key in defaultParameters[modelIdentifier]){
        if(getCookie(key) == undefined || getCookie(key) == "")
            inputs[key] = defaultParameters[modelIdentifier][key];
        else inputs[key] = getCookie(key);
    }


}
function setDefaultDistributionValues(identifier, textInputs){
    if( inputs[`${identifier}-constant-input`] == undefined)
        inputs[`${identifier}-constant-input`] = defaultConstantDistributionValue;
    textInputs.constant.value.value = inputs[`${identifier}-constant-input`];

    if(inputs[`${identifier}-uniform-lower`] == undefined)
        inputs[`${identifier}-uniform-lower`] = defaultUniformLower;
    textInputs.uniform[`uniform-lower`].value = inputs[`${identifier}-uniform-lower`];

    if(inputs[`${identifier}-uniform-upper`] == undefined)
        inputs[`${identifier}-uniform-upper`] = defaultUniformUpper;
    textInputs.uniform[`uniform-upper`].value = inputs[`${identifier}-uniform-upper`];

    if(inputs[`${identifier}-exponential-input`] == undefined)
        inputs[`${identifier}-exponential-input`] = defaultExponentialValue;
    textInputs.exponential.value.value = inputs[`${identifier}-exponential-input`];

}
function setDefaultFeedbackValue(feedbackInput){
    if(inputs[`feedback-return-rate`] == undefined)
        inputs[`feedback-return-rate`] = defaultFeedbackRate;
    feedbackInput.value = inputs[`feedback-return-rate`];
}
function setDefaultInputValue(inputName, inputValue, inputField){
    if(inputs[inputName] == undefined)
        inputs[inputName] = inputValue;
    inputField.value = inputs[inputName];
}
function setDefaultValue(inputName, value, inputComponent){
    if(inputs[inputName] == undefined)
```

```
        inputs[inputName] = value;
    inputComponent.value = inputs[inputName];
}
function setDefaultQueueSize(queueInput){
    var model = getCookie(`selectedModel`);
    inputs['queue-size'] = defaultParameters[model]["queue-size"];
    queueInput.value = defaultParameters[model]["queue-size"];
}

/////////////////////////////////////////////////////////////////////
//CLASSES
////////////////////////////////////////////////////////////////////

//Field Classes
class InputField{
    constructor(type, identifier, name, description, value){
        this.identifier = identifier;
        this.type = type;
        this.description = description;
        this.value = value;
        this.name = name;
    }
    generateHTML(){}
}
class TextField extends InputField{
    constructor(identifier, name, description){
        super("text", identifier, name, description, '');
    }
    generateHTML(){
        var label = `<label for="${this.identifier}" class="configuration-label">${this.description}</label>`;
        var input = `<input type="${this.type}" id="${this.identifier}" name="${this.name}"
value="${this.value}" class="configuration-text-input">`;
        return label+input;
    }
}
class RadioButton extends InputField{
    constructor(identifier, name, description, value){
        super("radio", identifier, name, description, value);
    }
    generateHTML(){
        var label = `<label for="${this.identifier}" class="configuration-label">${this.description}</label>`;
        var input = `<input type="${this.type}" id="${this.identifier}" name="${this.name}"
class="configuration-radio" value="${this.value}">`;
        return input + label;
    }
}
class Button extends InputField{
    constructor(identifier, description){
        super("button", identifier, name, description, null);
    }
    generateHTML(){
        var button = `<button id="${this.identifier}" class="configuration-selection-
button">${this.description}</button>`;
        return button;
    }
}

//Section for a Single Component
class ConfigurationSection{
    constructor(identifier, fields, listenerFunction = null){
        this.identifier = identifier
        this.fields = fields;
        this.listenerFunction = listenerFunction;
    }
    //calls the listener function this must be done after the initial section creations
    callListener(){
        if(this.listenerFunction !== null){
            var outputFields = this.listenerFunction(this.identifier);
            if(typeof outputFields == "object"){
                this.fields = {...this.fields, ...outputFields};
            }
        }
    }
    generateHTML(location){
        this.fields.forEach(field => {
            location.innerHTML += field.generateHTML();
        });
        location.innerHTML += '<br>';
    }
}

//Model Configurations
class ModelConfiguration{
    constructor(identifier){
        this.identifier = identifier;
        this.sections = [];
    }
    callListeners(){
        for(var i = 0; i < this.sections.length; i++){
            this.sections[i].callListener();
        }
    }
    addSection(section){
        this.sections.push(section);
    }
    addSectionArray(sectionArray){
        for(var i = 0; i < sectionArray.length; i++){
            this.sections.push(sectionArray[i]);
        }
    }
    addResponsiveSection(responsiveSection){
        this.responsiveSections.push(responsiveSection);
    }
    writeContent(){
        this.sections.forEach(section => {
            this.location.innerHTML += section.generateHTML;
        });
    }
}
class SingleServerConfiguration extends ModelConfiguration{
    constructor(){
        super("single-server");
        addTitle("Single Server Model");
        createCategoryStartDivision("configurationDivision", "Configuration Settings");
        this.addSection(createQueueSizeConfiguration("queue", "Queue Visual"));

        createCategoryStartDivision("distributionDivision", "Distribution Rates");
        this.addSection(createArrivalConfiguration("arrival", "Arrival Rate"));
        this.addSection(createServerConfiguration("server", "Service Rate"));
        this.addSection(addEndButtons());
        this.callListeners();
    }
}
class ParallelServersConfiguration extends ModelConfiguration{
    constructor(){
        super("parallel-servers");
        addTitle("Parallel Servers Model");
        createCategoryStartDivision("configurationDivision", "Configuration Settings");
        this.addSection(createServerCountInput("server-count", "Server Count "));
        this.addSection(createQueueSizeConfiguration("queue", "Queue Visual"));

        createCategoryStartDivision("distributionDivision", "Distribution Rates");
        this.addSection(createArrivalConfiguration("arrival", "Arrival Rate"));
        this.addSection(createServerConfiguration("parallel-servers", "Server Rate"));
        this.addSection(addEndButtons());
        this.callListeners();
        setDefaultValue("server-count", inputs["server-count"], document.getElementById("server-
count"));
    }
}
class SingleFeedbackConfiguration extends ModelConfiguration{
    constructor(){
        super("single-server-feedback");
        addTitle("Single Server with Feedback Model");
        createCategoryStartDivision("configurationDivision", "Configuration Settings");
        this.addSection(createFeedbackConfiguration("feedback", "Feedback"));
        this.addSection(createQueueSizeConfiguration("queue", "Queue Visual"));


        createCategoryStartDivision("distributionDivision", "Distribution Rates");
        this.addSection(createArrivalConfiguration("arrival", "Arrival Rate"));
        this.addSection(createServerConfiguration("server", "Server Rate"));
        this.addSection(addEndButtons());
        this.callListeners();
    }
}
class WorkstationConfiguration extends ModelConfiguration{
    constructor(){
        super("workstations");
        addTitle("Interactive Workstations Model");
        createCategoryStartDivision("configurationDivision", "Configuration Settings");
        this.addSection(createWorkstationSelections());


        createCategoryStartDivision("distributionDivision", "Distribution Rates");
        this.addSection(createServerConfiguration("workstation", "Workstations"));
        this.addSection(createServerConfiguration("processor", "Processors"));
        this.addSection(addEndButtons());
        this.callListeners();
        setDefaultValue("server-count", inputs["server-count"], document.getElementById("server-
count"));
        setDefaultValue("workstation-count", inputs["workstation-count"],
document.getElementById("workstation-count"));
    }
}
class CentralServerConfiguration extends ModelConfiguration{
    constructor(){
        super("central-server");
        addTitle("Central Server Model");
        createCategoryStartDivision("configurationDivision", "Configuration Settings");
        this.addSection(createCentralCountInputs());
        this.addSection(createQueueSizeConfiguration("queue", "Queue Visual:"));

        createCategoryStartDivision("distributionDivision", "Distribution Rates");
        this.addSection(createServerConfiguration("cpu", "CPU"));
        this.addSection(createServerConfiguration("channel", "Channel"));
        createParallelDiskInput(validations["maxDiskCount"]);
        clearDiskInputs();
        this.addSection(addEndButtons());
        drawDisks({"value": inputs["disk-count"]});
        this.callListeners();
    }
}

////////////////////////////////////////////////////////////////////////////////
//Functions to Add Components to Model Configurations
////////////////////////////////////////////////////////////////////////////////
function createConfigurationGroup(identifier, location, className="configuration-group"){
    location.innerHTML += `<div id="${identifier}" class="${className}"></div>`;
    return document.getElementById(`${identifier}`);
}
function createSubDivision(identifier, location, className = "configuration-subdivision"){
    location.innerHTML += `<div id="${identifier}" class="${className}"></div>`;
    return document.getElementById(`${identifier}`);
}
function createSectionTitle(identifier, title, destination, className="configuration-component-title"){
    var section = createSubDivision(`${identifier}-title`, destination, className);
    section.innerHTML = title;
    return section;
}
function addTitle(title){
    //var titleBig = createSubDivision("title-big-sub", body, "configuration-title-big" );
    document.getElementById("model-title").innerHTML = title;
    //titleBig.innerHTML = title;
}
function createRadioButton(identifier, name, description, value){
    var label = `<label for="${identifier}">${description}</label>`;
    var input = `<input type="radio" id="${identifier}" name="${name}" value="${value}">`;
    return label+input;
}
function createtextField(identifier, name, description, value){
    var label = `<label for="${identifier}">${description}</label>`;
    var input = `<input type="text" id="${identifier}" name="${name}" value="${value}
class="configuration-text-input">`;
    return label+input;
}
function createInputField(identifier, name){
    var configurationGroup = createConfigurationGroup(`${identifier}-configuration`, body);
    //createSectionTitle(`${identifier}-title`, name, configurationGroup);
    var inputDivision = createSubDivision(`${identifier}-input`,configurationGroup, "configuration-
labelless-division");
    var componentInput = {identifier : new TextField(`${identifier}`, `${identifier}`, `${name}`)};
    writeHTML(componentInput, inputDivision);
    return new ConfigurationSection(`${identifier}`, componentInput, addUpdateListener);
}
```

```
function createServerCountInput(identifier, name){
    var configurationGroup = createConfigurationGroup(`${identifier}-configuration`, body);
    //createSectionTitle(`${identifier}-title`, name, configurationGroup);
    var inputDivision = createSubDivision(`${identifier}-input`,configurationGroup, "configuration-
labelless-division");
    var componentInput = {identifier : new TextField(`${identifier}`, `${identifier}`, `${name} [1,9]: `)};
    writeHTML(componentInput, inputDivision);
    return new ConfigurationSection(`${identifier}`, componentInput, serverCountListener);
}
function createDiskWeightConfiguration(identifier){
    var weightIdentifier = `${identifier}-weight`;
    var configurationGroup = createConfigurationGroup(weightIdentifier, body, "disk-weight-group");
    createSectionTitle(`weightIdentifier-title`, "", configurationGroup, "disk-weight-title");
    var inputDivision = createSubDivision(`${weightIdentifier}-input`,configurationGroup, "disk-weight-
input");
    var componentInput = {weightIdentifier : new TextField(`${weightIdentifier}`, `${weightIdentifier}`,
`${name}`)};
    writeHTML(componentInput, inputDivision);
    return new ConfigurationSection(`${weightIdentifier}-input`, componentInput, diskWeightListener);
}
function createQueueSizeConfiguration(identifier, name){
    var configurationGroup = createConfigurationGroup(`queue-group`, body);
    //createSectionTitle(identifier, name, configurationGroup);
    var inputDivision = createSubDivision(`queue-input`,configurationGroup, "configuration-labelless-
division");
    var queueInput = {"queueInput" : new TextField(`queue-size`, `queue-size`, "Visible Queue Size:")};
    setDefaultQueueSize(queueInput.queueInput);
    writeHTML(queueInput, inputDivision);
    return new ConfigurationSection(`queue-size`, queueInput, queueListener);
}
```

```
function createFeedbackConfiguration(identifier, name){
   var configurationGroup = createConfigurationGroup(identifier, body);
   //createSectionTitle(identifier, name, configurationGroup);
   var inputDivision = createSubDivision(`${identifier}-feedback-input`,configurationGroup,
"configuration-labelless-division");
   var feedbackInput = {"feedbackInput" : new TextField(`${identifier}-return-rate`, `${identifier}-return-
rate`, "Return Rate:")};
   writeHTML(feedbackInput, inputDivision);
   inputDivision.innerHTML += "\xa0\xa0%";
   return new ConfigurationSection(`${identifier}-return-rate`, feedbackInput, feedbackListener);
}
function addInputListener(identifier, passedFunction){
   var component = document.getElementById(identifier);
   component.oninput = function(){
      passedFunction(component);
   }
}
function updateInputs(component){
   inputs[component.id] = component.value;
}
function addUpdateListener(identifier){
   var component = document.getElementById(identifier);
   component.oninput = function(){
      updateInputs(component);
   }
}
function updateNumber(component){
   if (validateNumber(component))
      updateInputs(component);
```

```
}
function serverCountListener(component){
    if (validateRange(validations["minServerCount"], validations["maxServerCount"], component))
        updateInputs(component);
}


//Components With Distributions
function createServerConfiguration(identifier, title){
    var configurationGroup = createConfigurationGroup(identifier, body);
    createSectionTitle( identifier, title, configurationGroup);
    var buttons = createDistributionButtons(identifier, configurationGroup);
    var listener = createDistributionInputs;
    return new ConfigurationSection(identifier, buttons, listener);
}
function createArrivalConfiguration(identifier, title){
    var configurationGroup = createConfigurationGroup(identifier, body);
    createSectionTitle( identifier, title, configurationGroup);
    var arrivalOptions = createDistributionButtons(identifier, configurationGroup);
    return new ConfigurationSection(identifier, arrivalOptions, createDistributionInputs);
};
function createDistributionButtons(identifier, destination, className = "distribution-group"){
    var distributionLocation = createSubDivision(`${identifier}-distribution`, destination, `${className}`);
    var buttonLocation = createSubDivision(`${identifier}-distribution-buttons`,distributionLocation);
    var addedButtons = {
        "constant": new RadioButton(`${identifier}-distribution-constant`, `${identifier}-distribution-
buttons`, 'Constant', 'constant'),
        "uniform": new RadioButton(`${identifier}-distribution-uniform`, `${identifier}-distribution-buttons`,
'Uniform', 'uniform'),
        "exponential": new RadioButton(`${identifier}-distribution-exponential`, `${identifier}-distribution-
buttons`, 'Exponential', 'exponential')
```

```
    }
    writeHTML(addedButtons, buttonLocation);
    var textInputLocation = createSubDivision(`${identifier}-distribution-input`, distributionLocation,
"distribution-input");
    return addedButtons;
}
function createDistributionInputs(identifier){
    var destination = document.getElementById(`${identifier}-distribution-input`);
    distributionSectionNames[`${identifier}`] = identifier;


    var textInputs = {
        "constant": {
            "value": new TextField(`${identifier}-constant-input`,`${identifier}-constant-input`, '| Value:') },
        "uniform": {
            "uniform-lower": new TextField(`${identifier}-uniform-lower`,`${identifier}-uniform-lower`, '|
Min:'),
            "uniform-upper": new TextField(`${identifier}-uniform-upper`,`${identifier}-uniform-upper`,
'Max:')},
        "exponential": {
            "value": new TextField(`${identifier}-exponential-input`,`${identifier}-exponential-input`, '|  Mean
Value:')}
    }
    setDefaultDistributionValues(identifier, textInputs);
    var inputType = inputs[`${identifier}-distribution-type`];
    if(inputType == "" || inputType == undefined){
        inputs[`${identifier}-distribution-type`] = "constant";
        inputType = inputs[`${identifier}-distribution-type`];
    }
    document.getElementById(`${identifier}-distribution-${inputType}`).checked = true;
    writeHTML(textInputs[inputType], destination);
```

```
switch(inputType){
    case "constant":
        addInputListener(`${identifier}-constant-input`, distributionListener);
        break;
    case "uniform":
        addInputListener(`${identifier}-uniform-lower`, distributionListener);
        addInputListener(`${identifier}-uniform-upper`, distributionListener);
        break;
    case "exponential":
        addInputListener(`${identifier}-exponential-input`, distributionListener);
        break;
}
document.getElementById(`${identifier}-distribution-constant`).onclick = function() {
    writeHTML(textInputs.constant, destination);
    addInputListener(`${identifier}-constant-input`, distributionListener);
    inputs[`${identifier}-distribution-type`] = 'constant';
    fillDistributionField(identifier, "constant");
};
document.getElementById(`${identifier}-distribution-uniform`).onclick = function() {
    writeHTML(textInputs.uniform, destination);
    addInputListener(`${identifier}-uniform-lower`, distributionListener);
    addInputListener(`${identifier}-uniform-upper`, distributionListener);
    inputs[`${identifier}-distribution-type`] = 'uniform';
    fillDistributionField(identifier, "uniform");


};
document.getElementById(`${identifier}-distribution-exponential`).onclick = function() {
    writeHTML(textInputs.exponential, destination)
    addInputListener(`${identifier}-exponential-input`, distributionListener);
```

```
        inputs[`${identifier}-distribution-type`] = 'exponential';
        fillDistributionField(identifier, "exponential");
    };
    return textInputs;
}
function fillDistributionField(identifier, distribution){
    //FILL IN FORM DATA
    switch(distribution){
        case "constant":
            var constantInput = document.getElementById(`${identifier}-constant-input`);
            if(inputs[constantInput.id] != undefined && inputs[constantInput.id] != null){
                constantInput.value = inputs[constantInput.id];
            }
            break;
        case "uniform":
            var lower = document.getElementById(`${identifier}-uniform-lower`);
            var upper = document.getElementById(`${identifier}-uniform-upper`);
            if(inputs[lower.id] != undefined && inputs[lower.id] != null){
                 lower.value = inputs[lower.id];
            }
            if(inputs[upper.id] != undefined && inputs[upper.id] != null){
                upper.value = inputs[upper.id];
            }
            break;
        case "exponential":
            var exponentialInput = document.getElementById(`${identifier}-exponential-input`);
            if(inputs[exponentialInput.id] != undefined && inputs[exponentialInput.id] != null){
                exponentialInput.value = inputs[exponentialInput.id];
            }
        break;
    }


}
function fillServerInput(identifier){
    var distribution = inputs[`${identifier}-distribution-type`];
    var checkedButton = document.getElementById(`${identifier}-distribution-${distribution}`);
    checkedButton.checked = true;
    var event = document.createEvent('HTMLEvents');
    event.initEvent('click', false, true);
    checkedButton.dispatchEvent(event);
    fillDistributionField(identifier, distribution);
}
//Single Server with Feedback Specific
function createFeedbackConfiguration(identifier, name){
    var configurationGroup = createConfigurationGroup(identifier, body);
    //createSectionTitle(identifier, name, configurationGroup);
    var inputDivision = createSubDivision(`${identifier}-feedback-input`,configurationGroup,
"configuration-labelless-division");
    var feedbackInput = {"feedbackInput" : new TextField(`${identifier}-return-rate`, `${identifier}-return-
rate`, "Feedback Rate:")};
    setDefaultFeedbackValue(feedbackInput.feedbackInput);
    writeHTML(feedbackInput, inputDivision);
    inputDivision.innerHTML += "\xa0\xa0%";
    return new ConfigurationSection(`${identifier}-return-rate`, feedbackInput, feedbackListener);
}
//Interactive Workstations Specific
function createWorkstationSelections(){
    var configurationGroup = createConfigurationGroup("component-counts", body);
```

```
   //createSectionTitle("component-counts", "", configurationGroup);
   var inputDivision = createSubDivision(`componentCount-input`,configurationGroup, "configuration-
labelless-division");


   var inputs = {
      "serverCount": new TextField("server-count", "server-count", "Number of Servers[1,8]:"),
      "workstationCount": new TextField("workstation-count", "workstation-count", "|\xa0Number of
Workstations[1,9]: ")
   };
   writeHTML(inputs, inputDivision);
   return new ConfigurationSection("componentCounts", inputs, countListener);
}
//Interactive Workstations Specific
function createWorkstationServerCount(){
   var configurationGroup = createConfigurationGroup("server-count-group", body);
   //createSectionTitle("server-count", "Server Count", configurationGroup);
   var inputDivision = createSubDivision(`server-count`,configurationGroup, "configuration-labelless-
division");
   var serverCount = {"serverCount": new TextField("server-count", "server-count", "Quantity[1,9]:")};
   writeHTML(serverCount, inputDivision);
   return new ConfigurationSection("server-count", serverCount, serverCountListener);
}
function createWorkstationCount(){
   var configurationGroup = createConfigurationGroup("workstation-count-group", body);
   //createSectionTitle("workstation-count", "Workstation Count", configurationGroup);
   var inputDivision = createSubDivision(`workstation-count`,configurationGroup, "configuration-
labelless-division");
   var workstationCount = {"workstationCount": new TextField("workstation-count", "workstation-
count", "Quantity[1,9]:")};
   writeHTML(workstationCount, inputDivision);
```

```
    return new ConfigurationSection("workstation-count", workstationCount, workstationCountListener);
}
//Central Server Specific
function createCentralCountInputs(){
    var configurationGroup = createConfigurationGroup("component-counts", body);
    //createSectionTitle("component-counts", "", configurationGroup);
    var inputDivision = createSubDivision(`countInputs`,configurationGroup, "configuration-labelless-
division");
    var diskInputs = {
        "disk-count": new TextField("disk-count", "disk-count", "Number of Disks[1,9]:"),
        "core-count": new TextField("core-count", "core-count", "|  Number of Cores[1,8]:"),
        "job-count": new TextField("job-count", "job-count", "|  Number of Jobs:")
    };
    diskInputs["disk-count"].value = inputs["disk-count"];
    diskInputs["job-count"].value = inputs["job-count"];
    writeHTML(diskInputs, inputDivision);
    return new ConfigurationSection("component-counts", diskInputs, centralCountListener);
}
//Central Server Specific
function createDiskDistributionConfiguration(identifier, title){
    var configurationGroup = createConfigurationGroup(identifier, body, "disk-configuration-group");
    createSectionTitle( identifier, title, configurationGroup);
    var buttons = createDiskDistributionButtons(identifier, configurationGroup, "disk-distribution-group");
    var listener = createDistributionInputs;
    return new ConfigurationSection(identifier, buttons, listener);
}
function createDiskDistributionButtons(identifier, destination, className = "disk-distribution-group"){
    var distributionLocation = createSubDivision(`${identifier}-distribution`, destination, `${className}`);
```

```
   var buttonLocation = createSubDivision(`${identifier}-distribution-buttons`,distributionLocation, "disk-
distribution-group");
   var addedButtons = {
       "constant": new RadioButton(`${identifier}-distribution-constant`, `${identifier}-distribution-
buttons`, 'Constant', 'constant'),
       "uniform": new RadioButton(`${identifier}-distribution-uniform`, `${identifier}-distribution-buttons`,
'Uniform', 'uniform'),
       "exponential": new RadioButton(`${identifier}-distribution-exponential`, `${identifier}-distribution-
buttons`, 'Exponential', 'exponential')
   }
   writeHTML(addedButtons, buttonLocation);
   var textInputLocation = createSubDivision(`${identifier}-distribution-input`, distributionLocation,
"disk-distribution-input");
   return addedButtons;
}
//Category Divisions
function createCategoryStartDivision(identifier, title){
   var configurationGroup = createConfigurationGroup(identifier, body);
   createSectionTitle( identifier, title, configurationGroup, "configuration-category-start-division");
}
function createCategoryEndDivision(identifier){
   var configurationGroup = createConfigurationGroup(identifier, body);
   createSectionTitle( identifier, title, configurationGroup, "configuration-category-end-division");
}
function createCategoryEndDivision(identifier){


}
///////////////////////////////////////////////////////
// Input Handling and Validations
///////////////////////////////////////////////////////
```

```
function getCookie(key) {
  var name = key + "=";
  var decodedCookie = decodeURIComponent(document.cookie);
  var cookieArray = decodedCookie.split(';');
  for(var i = 0; i < cookieArray.length; i++) {
   var cookie = cookieArray[i];
   while (cookie.charAt(0) == ' ') {
    cookie = cookie.substring(1);
   }
   if (cookie.indexOf(name) == 0) {
    return cookie.substring(name.length, cookie.length);
   }
  }
  return "";
}
function writeCookies(){
  for(var i = 0; keys = Object.keys(inputs), i < keys.length; i++){
    document.cookie = `${keys[i]}=${inputs[keys[i]]}`;
  }
}
function writeCookie(name, value){
  document.cookie = `${name} = ${value}`;
}
function clearCookies(){
    var cookies = document.cookie.split("; ");
    for (var c = 0; c < cookies.length; c++) {
      var d = window.location.hostname.split(".");
      while (d.length > 0) {
            var cookieBase = encodeURIComponent(cookies[c].split(";")[0].split("=")[0]) + '=; expires=Thu,
01-Jan-1970 00:00:01 GMT; domain=' + d.join('.') + ' ;path=';
            var p = location.pathname.split('/');
            document.cookie = cookieBase + '/';
            while (p.length > 0) {
              document.cookie = cookieBase + p.join('/');
              p.pop();
            };
            d.shift();
          }
      }
}
function validateCountInput(component){
  var result = component.value.replace(/\D|^0/, '');//.substr(0, component.value.length - 1);
  if (result == component.value) return true;
  else{
    if(component.value.length == 0)
      component.value = "";
    else
      component.value = result;
    return false;
  }
}
function validateNumber(userInput){
  var value = userInput.value;
  if( isNumeric(value) ){
    return true;
  }
  else{
        if(value.length != 0)
            userInput.value = value.substr(0, value.length - 1);
        else
            userInput.value = "";
        return false;
    }
}
function isNumeric(numberToCheck){
    return !isNaN(numberToCheck - parseFloat(numberToCheck));
}
function validateWholeNumber(userInput){
    var result = userInput.value.replace(/^0|\D/, '');
    if (result == userInput.value) return true;
    else{
        userInput.value = result;
        return false;
    }
}
function validateSingleServer(){
    return (
        validateDistribution("arrival") &&
        validateDistribution("server") &&
        validateQueueSize()
    );
}
function validateParallelServer(){
    return (
        validateDistribution("arrival") &&
        validateDistribution("parallel-servers") &&
        checkRange(minServerCount, maxServerCount, inputs["server-count"]) &&
        validateQueueSize()
    );
}
function validateSingleFeedback(){
    return (
        validateDistrubtion("arrival") &&
        validateDistribution("server") &&
        checkRange(inputs["feedback-return-rate"]) &&
        validateQueueSize()
    );
}
function validateInteractive(){
    return (
        checkRange(minServerCount, maxServerCount, inputs["server-count"]) &&
        checkRange(minWorkstationCount, maxWorkstationCount, inputs["workstation-count"]) &&
        validateDistribution("workstation") &&
        validateDistribution("processor")
    );
}
function validateRange(min, max, component){
    var numericalValue = parseFloat(component.value);
    if ( isNumeric(component.value) && (min <= numericalValue) && (numericalValue <= max) )
        return true;
    else{
        if(component.value.length != 0)
            component.value = component.value.substring(0, component.value.length-1);
    }
    return false;
```

```
}
function validateWholeNumberRange(min, max, component){
   var numericalValue = parseFloat(component.value);
   if ( validateWholeNumber(component) )
      return validateRange(min,max,component);
   return false;
}


/////////////////////////////////////////////////////
// Listeners
/////////////////////////////////////////////////////
function writeHTML(inputText, location){
   location.innerHTML = "";
   for(var key in inputText){
      location.innerHTML += inputText[key].generateHTML();
  }
}
function distributionListener(component){
   if (validateRange(0, Infinity, component)){
      updateInputs(component);
   }
}
function countListener(identifier){
   addInputListener("server-count", serverCountUpdate);
   addInputListener("workstation-count", workstationUpdate);
}
function serverCountListener(identifier){
   addInputListener(identifier, serverCountUpdate);
```

```
}
function workstationCountListener(identifier){
   addInputListener(identifier, workstationCountListener);
}
function serverCountUpdate(component){
   if(validateWholeNumberRange(validations["minServerCount"], validations["maxServerCount"],
component))
       updateInputs(component);
}
function workstationUpdate(component){
   if(validateWholeNumberRange(validations["minWorkstationCount"],
validations["maxWorkstationCount"], component))
       updateInputs(component);
}
function centralCountListener(identifier){
   addInputListener("job-count", jobUpdate);
   addInputListener("disk-count", diskUpdate);
}
function diskWeightListener(component){
   var inputIdentifier = component.id.replace("-input", "");
   if(validateNumber(component))
       inputs[`${inputIdentifier}`] = component.value;
}
function diskWeightUpdate(component){
   if(validateNumber(component.value))
       updateInputs(component);
}
function jobUpdate(component){
   if(validateWholeNumberRange(validations["minJobCount"], validations["maxJobCount"],
component))
```

```
        updateInputs(component);
}
function diskUpdate(component){
    if(validateWholeNumberRange(validations["minDiskCount"], validations["maxDiskCount"],
component))
        updateInputs(component);
    drawDisks(component);
}
function drawDisks(component){
    clearDiskInputs();
    for(var i = 0; i < component.value; i++){
        var identifier = `disk-${i+1}`;
        var configurationGroup = document.getElementById(identifier);
        createSectionTitle( identifier, `Disk ${i+1}`, configurationGroup, "disk-distribution-title");
        createDistributionButtons(identifier, configurationGroup, "disk-distribution-group");
        createDistributionInputs(identifier);
        fillServerInput(identifier);
        drawDiskWeightInput(identifier);
    }
}
function drawDiskWeightInput(identifier){
    var weightIdentifier = `${identifier}-weight`;
    var configurationGroup = document.getElementById(`${weightIdentifier}`);


    createSectionTitle(`${weightIdentifier}-title`, "", configurationGroup, "disk-weight-title");
    var inputDivision = createSubDivision(`${weightIdentifier}-division`,configurationGroup, "disk-weight-
input");
    var componentInput = {"textInput" : new TextField(`${weightIdentifier}-input`, `${weightIdentifier}`,
`Access Probability / Weight: `)};
```

```
    if(inputs[`${weightIdentifier}`] == null || inputs[`${weightIdentifier}`] == undefined)
        inputs[`${weightIdentifier}`] = defaultParameters["centralServer"]["disk-weight"];

    componentInput["textInput"].value = inputs[`${weightIdentifier}`];
    writeHTML(componentInput, inputDivision);
    addInputListener(`${weightIdentifier}-input`, diskWeightListener);
}
function clearDiskInputs(){
    for(var i = 0; i < validations["maxDiskCount"]; i++){
        var section = document.getElementById(`disk-${i+1}`)
        if(section != null)
            section.innerHTML ='';
        var weightSection = document.getElementById(`disk-${i+1}-weight`);
        if(weightSection != null)
            weightSection.innerHTML = ``;
    }
}
function createParallelDiskInput(count){
    var diskIdentifiers = [];
    var diskSections = [];
    for(var i = 0; i < count; i++){
        var identifier = `disk-${i+1}`;
        diskSections.push(createDiskDistributionConfiguration(identifier, `Disk ${i+1}`));
        diskIdentifiers.push(identifier);
        createDistributionInputs(identifier);
        diskSections.push(createDiskWeightConfiguration(identifier));
    }
    return diskSections;
}
```

```
function feedbackListener(identifier){
    var component = document.getElementById(identifier);
    component.oninput = function(){
        feedbackUpdate(component);
    }
}
function feedbackUpdate(component){
    validateRange(validations["minFeedbackRate"], validations["maxFeedbackRate"], component);
}
function queueListener(identifier){
    var component = document.getElementById(identifier);
    component.oninput = function(){
        if(queueUpdate(component)) inputs["queue-size"] = component.value;
        console.log(inputs["queue-size"]);
    }
}
function queueUpdate(component){
    if(component.value.length >2){
        component.value = component.value.substr(0, component.value.length-1);
    }
    console.log(component.value);
    return validateCountInput(component);
}
//////////////////////////////////////////////////
// Creating the Configuration Sections
//////////////////////////////////////////////////
function createConfiguration(){
    var configuration = undefined;
    setDefaultParameters(selectedModel);
    switch(selectedModel){
        case "singleServer":
            configuration = new SingleServerConfiguration();
            break;
        case "parallelServers":
            configuration = new ParallelServersConfiguration();
            break;
        case "singleFeedback":
            configuration = new SingleFeedbackConfiguration();
            break;
        case "workstations":
            configuration = new WorkstationConfiguration();
            break;
        case "centralServer":
            configuration = new CentralServerConfiguration();
            break;
    }
    clearCookies();
    writeCookie("selectedModel", selectedModel);
    return configuration;
}

//End Buttons Are The Run / Select Model Buttons
function addEndButtons(){
    var buttons = {
        "select-button": new Button("select-button", "Select Model"),
        "run-button": new Button("run-button", "Run")
    }
    var endButtonGroup = createConfigurationGroup("end-buttons", body);
    endButtonGroup.innerHTML += buttons[`run-button`].generateHTML();
    endButtonGroup.innerHTML += buttons[`select-button`].generateHTML();

    return new ConfigurationSection("endButtons", buttons, endButtonListener);
}
function endButtonListener(){
    var selectButton = document.getElementById("select-button");
    var runButton = document.getElementById("run-button");
    selectButton.onclick = function(){
        window.location.href = "../Pages/modelSelection.html";
    }
    runButton.onclick = function(){
        var displayedErrors = "";
        if(!checkQueueSize())
            displayedErrors += "Invalid Queue Size ";
        if(!checkDistributions())
            displayedErrors += "Invalid Distribution ";

        if(displayedErrors == ""){
            writeCookies();
            window.location.href = "../Pages/runTime.html";
        }
        else alert(displayedErrors);
    }
}
function checkQueueSize(){
    var queueSection = document.getElementById("queue-size");
    if (queueSection == null) return true;
```

```
  var currentSize = queueSection.value;
  if(currentSize < validations["minQueueSize"] || currentSize > validations["maxQueueSize"])
    return false;
  return true;
}
function checkDistributions(){
  for( var name in distributionSectionNames ){
    var distributionType = inputs[`${name}-distribution-type`];
    switch(distributionType){
      case "uniform":
        if(parseFloat(inputs[`${name}-uniform-lower`]) == 0 || parseFloat(inputs[`${name}-uniform-
upper` == 0])){
          return false;
        }
        if ( parseFloat(inputs[`${name}-uniform-lower`]) > parseFloat(inputs[`${name}-uniform-upper`])
){
          if(document.getElementById(`${name}-distribution-input`) !== undefined)
            return false;
        }
        break;
      case "exponential":
        console.log(inputs[`${name}-exponential-input`]);
        if(parseFloat(inputs[`${name}-exponential-input`]) == 0)
          return false;
        break;
      case "constant":
        if(parseFloat(inputs[`${name}-constant-value`]) == 0 )
          return false;
        break;
        }


    }


    return true;
}
createConfiguration();
```

modelDescriptions.js

```
function createDescriptionBlock(location){
    var destination = document.getElementById(location);
    destination.innerHTML += `<div id="description-block" class="description-block"></div>`;
    return document.getElementById("description-block");
}
function addDescriptionTitle(content){
    document.getElementById("description-block").innerHTML += `<br></br><b>${content}</b>`;
}
function addDescriptionBlock(content){
    document.getElementById("description-block").innerHTML += content;
}
var singleTitle = "Single Server Model";
var singleDescription = "The single server model is the simplest of models. It consists of a single queue
and a single processor.  Jobs arrive, are enqueued, processed at the server then exit the system.
Adjustable parameters for this model are interarrival rate and distribution as well as server processing
rate and distribution. This model could be representative of a drive through at a fast food restaurant:
people arrive, enqueue, are serviced, and leave.";


var twoTitle = "Two Servers Model";
var twoDescription = "The two servers model expands upon the single server model by adding an
additional server in parallel. This model consists of a single queue with two servers. Tasks from the
queue are placed into the next available server for processing. Adjustable parameters for this model are
```

```
interarrival rate and distribution as well as server processing rate and distribution for each of the two
servers. This type of model could be experienced at barbershops, a customer checks in (is enqueued)
then gets service from one of many potential barbers.";


var feedbackTitle = "Single Server with Feedback Model";
var feedbackDescription = "FEEDBACK DESCRIPTION GOES HERE";


var workstationTitle = "Interactive Workstations Model";
var workstationDescription = "WORKSTATION DESCRIPTION GOES HERE";


var centralTitle = "Central Server with Disk Access Model";
var centralDescription = "CENTRALDESCRIPTION GOES HERE";


var selectedModel = getCookie("selectedModel");
function writeDescription(){
  switch(selectedModel){
    case "singleServer":
      addDescriptionTitle(singleTitle);
      addDescriptionBlock(singleDescription);
      break;
    case "twoServers":
      addDescriptionTitle(twoTitle);
      addDescriptionBlock(twoDescription);
      break;
    case "singleFeedback":
      addDescriptionTitle(feedbackTitle);
      addDescriptionBlock(feedbackDescription);
      break;
    case "workstations":
      addDescriptionTitle(workstationTitle);
      addDescriptionBlock(workstationDescription);
      break;
    case "centralServer":
      addDescriptionTitle(centralTitle);
      addDescriptionBlock(centralDescription);
      break;
  }
}
```

modelSelection.js

```
var single = document.getElementById("single-server");
var feedback = document.getElementById("single-feedback");
var parallel = document.getElementById("parallel-servers");
var workstations = document.getElementById("workstations");
var center = document.getElementById("central-server");

single.onclick = function(){
    document.cookie = `selectedModel = singleServer;`;
    window.location.href = "../Pages/modelConfiguration.html";
}

feedback.onclick = function() {
    document.cookie = `selectedModel = singleFeedback;`;
    window.location.href = "../Pages/modelConfiguration.html";
}
parallel.onclick = function() {
    document.cookie = `selectedModel = parallelServers;`;
    window.location.href = "../Pages/modelConfiguration.html";
}
workstations.onclick = function() {
   document.cookie = `selectedModel = workstations;`;
   window.location.href = "../Pages/modelConfiguration.html";
}
center.onclick = function() {
   document.cookie = `selectedModel = centralServer;`;
   window.location.href = "../Pages/modelConfiguration.html";
}
```

pageLayout.js

```
var modelSelectButton = document.getElementById("buttonModelSelect");
modelSelectButton.onclick = function(){
   window.location.href = "Pages/modelSelection.html";
}

var helpButton = document.getElementById("buttonHelp");
helpButton.onclick = function(){
   window.location.href = "Pages/help.html";
}
```

runTime.js

```
var navigationMenu = document.getElementById("navigationMenu");
navigationMenu.innerHTML = `
<button id="modelSelectionButton" class="navigation-button"> Select Model </button>
<button id="parameterButton" class="navigation-button"> Change Parameters </button>
<button id="resetButton" class="navigation-button"> Restart Current Model </button>`;
```

```
var speedMenu = document.getElementById("speedMenu");
speedMenu.innerHTML = `
<button id="increaseSpeedButton" class="speed-button"> Increase Speed </button>
<button id="decreaseSpeedButton" class="speed-button"> Decrease Speed </button>
<button id="changeModeButton" class="speed-button"> Maximum Speed </button>
<button id="stepOnceButton" class="speed-button"> Single Step </button>
<button id="stopResumeButton" class="speed-button"> Pause </button>`;

//BUTTONS!!!!
var stopResumeButton = document.getElementById("stopResumeButton");
var modelSelectButton = document.getElementById("modelSelectionButton");
var helpButton = document.getElementById("parameterButton");
var resetButton = document.getElementById("resetButton");
var increaseSpeedButton = document.getElementById("increaseSpeedButton");
var decreaseSpeedButton = document.getElementById("decreaseSpeedButton");
var stepOnceButton = document.getElementById("stepOnceButton");
var changeModesButton = document.getElementById("changeModeButton");
/////////////////////////////////////////////////////////////

function setDefaultButtons(){
  stopResumeButton.onclick = function(){
    stopModel();
    updateSpeedMenu();
  }
  modelSelectButton.onclick = function(){
    window.location.href = "../Pages/modelSelection.html";
  }
  helpButton.onclick = function(){
    window.location.href = "../Pages/modelConfiguration.html";
    }
    resetButton.onclick = function(){
        stopModel();
        resetModel();
        updateSpeedMenu();
    }
    increaseSpeedButton.onclick = function(){
        increaseSpeed();
    }
    decreaseSpeedButton.onclick = function(){
        decreaseSpeed();
    }
    stepOnceButton.onclick = function(){
        startStepOnceMode();
        updateSpeedMenu();
    }
    changeModesButton.onclick = function(){
        beginSimulationMode();
        updateSpeedMenu();
    }

}

function compareServer(identifier){
    if( currentModel.components[identifier].task == null){
        if( currentSimulation.components[identifier].task != null ){
            alert(`${identifier}` + "MODEL NULL, SIM NOT");
            stopModel();
        }
        return;
    }else if( currentSimulation.components[identifier].task == null){
        if( currentModel.components[identifier].task != null ){
            alert(`${identifier}` + "SIM NULL, MODEL NOT");
            stopModel();
        }
        return;
    }else if(currentModel.components[identifier].task.identifier  != 
currentSimulation.components[identifier].task.identifier){
        alert(`${identifier}` + "SIM: " + currentSimulation.components[identifier].task.identifier + " MODEL: 
" + currentModel.components[identifier].task.identifier );
        return;
    }
}


function updateSpeedMenu(){
    if(simulationMode == true){
        changeModesButton.innerHTML = "Normal Speed";
        changeModesButton.onclick = function(){
            console.log("CLICKED NORMAL MODE");
            setDefaultButtons();
            currentSimulation.copyAllTasks();
            currentModel.draw();


            simulationMode = false;


            //stopModel();
            runModel();
            updateSpeedMenu();
```

```
}
stopResumeButton.innerHTML = "Pause";
stopResumeButton.onclick = function(){
   currentSimulation.copyAllTasks();
   currentModel.draw();
   simulationMode = false;
   stopModel();
   updateSpeedMenu();
}
increaseSpeedButton.onclick = function(){
   currentSimulation.copyAllTasks();
   currentModel.draw();
   simulationMode = false;
   runModel();
   increaseSpeed();
}
decreaseSpeedButton.onclick = function(){
   currentSimulation.copyAllTasks();
   currentModel.draw();
   simulationMode = false;
   runModel();
   decreaseSpeed();
}
stepOnceButton.onclick = function(){
   currentSimulation.copyAllTasks();
   currentModel.draw();
   simulationMode = false;
   startStepOnceMode();
   updateSpeedMenu();
```

```
        }
    }
    else if (currentModel.timerRunning == false || stepMode == true){
        stopResumeButton.innerHTML = "Resume";
        stopResumeButton.onclick = function(){
            runModel();
            updateSpeedMenu();
        }
    }
    else{
        stopResumeButton.innerHTML = "Pause";
        stopResumeButton.onclick = function(){
            stopModel();
            updateSpeedMenu();
        }
    }
    if(simulationMode == false){
        changeModesButton.innerHTML = "Maximum Speed";
        simulationMode = false;
        changeModesButton.onclick = function(){
            beginSimulationMode();
            updateSpeedMenu();
        }
    }
}
/////////////////////////////////////

setDefaultButtons();
```

simulation.js

```
var error = null;


function outputNumber(number){
   if(isNaN(number) )
      return "--";
   else return number;
}
//Generates headings from array or map
function generateTableHeadings(tableHeadings){
   var output = "<tr>";
   if(Array.isArray(tableHeadings)){
      tableHeadings.forEach(element => {
         output += `<th>${element}</th>`;
      });
   }
   else{
      for(var key in tableHeadings){
         output += `<th>${outputNumber(tableHeadings[key])}</th>`;
      }
   }
   return output += "</tr>"
}


//Generates rows from array or map
function generateTableData(tableData){
   var output = "<tr>"
   if(Array.isArray(tableData)){
      tableData.forEach(element => {
            output += `<td>${element}</td>`;
        });
    }
    else{
        for(var key in tableData){
            output += `<td>${outputNumber(tableData[key])}</td>`;
        }
    }
    return output += "</tr>";
}
//Generates headings followed by rows from headings / data
function generateTable(tableHeadings, tableData){
    var output = generateTableHeadings(tableHeadings);
    output += generateTableData(tableData);
    return output;
}

function generateLabeledRow(label, data){
    var output = "";
    output += `<tr>`
    output += `<td>${label}</td>`;
    output += `<td>${outputNumber(data)}</td>`;
    output += `</tr>`;
    return output;
}
function generateOutputTables(tableHeadings, tableData){
    var output = "";
    output+= `<th>${tableData[0]}</th> `;
    for(var i = 1; i < tableHeadings.length; i++){
        output += `<tr>`
        output += `<td>${tableHeadings[i]}</td>`;
        output += `<td>${outputNumber(tableData[i])}</td>`;
        output += `</tr>`;
    }
    return output;
}
function generateTinyTable(tableData){
    var output = "<tr>"
    if(Array.isArray(tableData)){
        tableData.forEach(element => {
            output += `<td class="tiny-table-data">${outputNumber(element)}</td>`;
        });
    }
    else{
        for(var key in tableData){
            output += `<td class="tiny-table-data">${outputNumber(tableData[key])}</td>`;
        }
    }
    return output += "</tr>";
}
function generateTinyHeadings(tableHeadings){
    var output = "<tr>"
    if(Array.isArray(tableHeadings)){
        tableHeadings.forEach(element => {
            output += `<th class="tiny-table-heading">${element}</th>`;
        });
    }
    else{
        for(var key in tableHeadings){
            output += `<th class="tiny-table-heading">${outputNumber(tableHeadings[key])}</th>`;
        }
    }
    return output += "</tr>";
}
var currentTime = 0;
var tasksCompleted = 0;
var cumulativeResponseTime = 0;
var responseTimeCumulativeDeviation = 0;
var responseTimeDeviation = 0;

function simulationToTable(){
    var generalHeadings = ["Current Time", "Jobs Completed", "Response Time"];
    var generalData = [currentTime, tasksCompleted, cumulativeResponseTime / tasksCompleted];
    var output = "<tr>";
    generalHeadings.forEach(element => {
        output += `<th>${element}</th>`;
    });
    output += "</tr><tr>"
    generalData.forEach(element => {
        output += `<td>${element}</td>`;
    });
    output += "</tr>";
    return output;
}
function setTime(time){
    currentTime = time;
```

```
}

class Task{
    constructor(identifier, creationTime){
        this.identifier = identifier;
        this.creationTime = creationTime;
    }
    getRunTime(){
        return currentTime - this.creationTime;
    }
    toString(){
        return this.identifier;
    }
}

class Distribution{
    constructor(type, values){
        this.type = type;
        this.values = values;
    }
    generate(){
        switch(this.type){
            case "uniform":
                var upperBound = this.values.upperBound;
                var lowerBound = this.values.lowerBound;
                return Math.random() * ( (upperBound - lowerBound) ) + lowerBound;
            case "exponential":
                var mean = this.values.mean;
                return -1.0 * mean * Math.log(1.0 - Math.random());
            case "constant":
                return this.values.value;
        }
    }
    toString(){
        var distributionComponents;
        switch(this.type){
            case "uniform":
                distributionComponents = "Lower: " + this.values.lowerBound + "| Upper: "+
this.values.upperBound;
                break;
            case "exponential":
                distributionComponents = this.values.mean;
                break;
            case "constant":
                distributionComponents = this.values.value;
                break;
        }
        return "Distribution: " + this.type + " | " + distributionComponents;
    }
}

class SimulationComponent {
    constructor(identifier, type){
        this.identifier = identifier;
        this.type = type; //the type of object, ex. server
        this.model = null;
        this.available = true;
        this.task = null;
        this.timeNextEvent = null; //time at which next event is scheduled to occur for this object


        //user output
        this.userHeadings;
        this.userData;
        this.resultIndices = {};
        this.selectedResults = [];
    }
    copyTask(){
        if(this.task !== undefined && this.task !== null && this.visual !== undefined & this.visual !== null)
            this.model.copyTask(this.task.identifier, this.visualComponent);
    }
    updateUserData(){
        console.log("Update undefined for: " + this.identifier);
    }
    setResultIndices(){
        for(var i = 1; i < this.userHeadings.length; i++){
            this.resultIndices[this.userHeadings[i]] = i;
        }
    }
    setSelectedResults(selectedResults){
        this.setResultIndices();
        for(var i = 0; i < selectedResults.length; i++){
            this.selectedResults.push(selectedResults[i]);
            if(this.getResult(selectedResults[i]) == undefined){
                console.log("Invalid selected result! " + selectedResults[i]);
            }
        }
    }
```

```
getSelectedResults(){
   this.updateUserData();
   var output = "";
   var label;
   var data;
   for(var i = 0; i < this.selectedResults.length; i++){
      label = this.selectedResults[i];
      data = this.getResult(label);
      output += generateLabeledRow(label, data);
   }
   return output;
}
getDataResults(resultNames){
   var output = "";
   var results = [];
   this.updateUserData();
   for(var i = 0; i < resultNames.length; i++){
      results.push(this.getResult(resultNames[i]));
   }
   return generateTableData(results);
}
getResultValues(resultNames){
   var results = [];
   this.updateUserData();
   for(var i = 0; i < resultNames.length; i++){
      results.push(this.getResult(resultNames[i]));
   }
   return results;
}
```

```
getTinyTable(resultNames){
    var output = "";
    var results = [];
    this.updateUserData();
    for(var i = 0; i < resultNames.length; i++){
        results.push(this.getResult(resultNames[i]));
    }
    return generateTinyTable(results);
}
getHeadings(headingNames){
    return generateTableHeadings(headingNames);
}
getResult(resultName){
    return this.userData[ this.resultIndices[resultName] ];
}
userHeadingsToTable(){
    if(this.userHeadings !== undefined)
        return generateTableHeadings(this.userHeadings);
    return "";
}
userDataToTable(){
    this.updateUserData();
    if(this.userData !== undefined)
        return generateTableData(this.userData);
    return "";
}
userOutputTable(){
    this.updateUserData();
    if(this.userData !== undefined)
```

```
        return generateOutputTables(this.userHeadings, this.userData);
    }
    setNext(component){
        this.next = component;
    }
    setPrevious(component){
        this.previous = component;
    }
    acceptTask(task){
        this.task = task;
    }
    advanceTask(task){
        this.next.acceptTask(task);
    }
    //to be overridden in decendants for relevant output
    toString(){
        var output;
        output = this.identifier + "| Task: " + this.task;
    }
    checkAvailability(){
        return this.available;
    }
    resetVisual(){
        this.visual.reset();
    }
}

class QueueComponent extends SimulationComponent{
    constructor(identifier){
```

```
    super(identifier, "Queue");
    this.tasks = {};
    this.tasksWaiting = [];
    this.numberTasks = 0;

    //INFORMATION FOR RESULTS
    this.averageQueueLength = 0;
    this.weightedQueueSize = 0;
    this.timeLastAdvance = 0;
    this.timeLastUpdate = 0;
    this.userHeadings = ["Identifier", "Current Queue Length", "Mean Queue Length"];
    this.userData = [this.identifier, this.numberTasks, this.averageQueueLength];
    this.setResultIndices();
    //this.forwardedTasks = {};
}
copyTask(){
    /*
    for(var key in this.forwardedTasks){
        var createdTask = currentModel.createVisualTask(key, this.visual);
        this.visual.receiveTask(createdTask);
    }
    */
    for(var i = 0; i < this.tasksWaiting.length; i++){
        var taskIdentifier = this.tasksWaiting[i].identifier;
        var createdTask = currentModel.createVisualTask(taskIdentifier, this.visual);
        this.visual.receiveTask(createdTask);
    }
}
updateUserData(){
    this.averageQueueLength = this.weightedQueueSize / this.timeLastUpdate;
    this.userData = [this.identifier, this.numberTasks, this.averageQueueLength.toFixed(2)];
}
acceptTask(task){ //Receiving a new task
    if(this.numberTasks > maxQueueSize){
        error = "MAX QUEUE SIZE EXCEEDED";
    }
    this.tasks[task.identifier] = task;
    this.weightedQueueSize += (currentTime-this.timeLastUpdate)*this.numberTasks;
    this.timeLastUpdate = currentTime;
    this.numberTasks++;

    //if not available push to waiting
    if(!this.next.checkAvailability()){
        this.tasksWaiting.push(task);
    }
    else {
        if(this.tasksWaiting.length > 0){
            this.forwardTask(this.tasksWaiting.shift());
            this.tasksWaiting.push(task);
        }
        else{
            this.forwardTask(task);
        }
    }
    this.updateUserData();
}
forwardTask(task){
    //this.forwardedTasks[`${task.identifier}`] = task;
    this.next.acceptTask(task);
  }
  advanceTask(task){ //Called by component that queue is linked to
    //this.weightedQueueSize += this.numberTasks * (currentTime - this.timeLastAdvance);
    this.timeLastAdvance = currentTime;
    this.weightedQueueSize += (currentTime-this.timeLastUpdate)*this.numberTasks;
    this.timeLastUpdate = currentTime;
    this.removeTask(task);

    if(this.tasksWaiting.length > 0 && this.next.checkAvailability()){
      this.next.acceptTask(this.tasksWaiting.shift());
    }

    this.updateUserData();
  }
//TODO verify removed
  removeTask(task){
   // delete this.forwardedTasks[task.identifier];
    delete this.tasks[task.identifier];
    this.numberTasks--;
  }
  toString(){
    var output = this.identifier + ": <br/>Contained Tasks: ";
    for (const key in this.tasks) {
      if (this.tasks.hasOwnProperty(key))
        output  += this.tasks[key].toString() + " | ";
    }
    return output;
  }
```

```
    toTable(){
        var tableHeadings = ["Identifier", "Contained Tasks"];
        var tasks = "";
        for (const key in this.tasks) {
            if (this.tasks.hasOwnProperty(key))
            tasks  += this.tasks[key].toString() + " | ";
        }
        var tableData = [this.identifier, tasks];
        return generateTable(tableHeadings, tableData);
    }
}

class ServiceComponent extends SimulationComponent{
    constructor(identifier, distribution){
        super(identifier, "Server");
        this.distribution = distribution;
        this.connectedQueue = null;
        this.available = true;

        //Metrics to Evaluate
        this.jobsServiced = 0;
        this.timeUtilized = 0;
        this.utilization = 0;

        this.totalServiceTime = 0;
        this.averageServiceTime = 0;
        this.serviceCumulativeDeviation = 0;
        this.serviceTimeDeviation = 0;
        this.lastGeneratedServiceTime = 0;

        this.timeLastUpdate = 0;

        //User Output

        this.userHeadings = ["Identifier", "Utilization", "Mean Service Time", "SD Service Time", "Jobs
Serviced"];

        this.userData = [this.identifier, this.utilization, this.averageServiceTime, this.serviceTimeDeviation,
this.jobsServiced];

        this.setResultIndices();

    }

    copyTask(){

        this.visual.reset();

        this.visual = this.model.visualModel.components[this.identifier];

        if(this.task != undefined && this.task != null ){

            if( this.connectedQueue == null){

                var assignedTask = currentModel.createVisualTask(this.task.identifier, this.visual);

                this.visual.task = assignedTask;

                this.visual.acceptTask(assignedTask);

            }

            else{

                var assignedTask = currentModel.createVisualTask(this.task.identifier,
this.connectedQueue.visual);

                this.connectedQueue.visual.receiveTask(assignedTask);

                this.visual.acceptTask(assignedTask);

            }


        }

    }

    updateUserData(){

        this.serviceTimeDeviation = Math.sqrt(this.serviceCumulativeDeviation/(this.jobsServiced) -
this.averageServiceTime*this.averageServiceTime)
```

```
        this.userData = [this.identifier, this.utilization.toFixed(2), this.averageServiceTime.toFixed(2),
    this.serviceTimeDeviation.toFixed(2), this.jobsServiced];
    }
    connectQueue(queue){
        this.connectedQueue = queue;
    }
    acceptTask(task){
        this.task = task;
        //Generate Random Service Time Based on Distribution
        this.lastGeneratedServiceTime = this.distribution.generate();
        this.timeNextEvent = this.lastGeneratedServiceTime + currentTime;
        this.available = false;
        this.model.addEvent(this);

        //Return Information Regarding Next Event for This Object
        return {"component": this.identifier, "time": this.timeNextEvent, "task": task};
    }
    updateMetrics(){
        this.totalServiceTime += this.lastGeneratedServiceTime;
        this.averageServiceTime = this.totalServiceTime / (this.jobsServiced);
        this.serviceCumulativeDeviation += this.lastGeneratedServiceTime*this.lastGeneratedServiceTime
        this.utilization = this.totalServiceTime / currentTime;
        this.averageResponseTime = this.totalServiceTime / this.jobsServiced;
    }
    advanceTask(){
        //Move Task to Next Component
        var task = JSON.parse(JSON.stringify(this.task));
        if(this.next != null && this.next.available)
            this.next.acceptTask(this.task);
        this.jobsServiced++;
        this.updateMetrics();

        this.available = true;
        this.timeNextEvent = null;
        this.task = null;

        //If there is a connected queue, notify that server is available
        if(this.connectedQueue != null)
            this.connectedQueue.advanceTask(task);
    }
    toString(){
        var serverInformation = this.identifier + ": ";
        serverInformation += "<br>Jobs: " + this.jobsServiced + " | Response: " + this.averageResponseTime
+ " | Utilization: " + this.utilization;
        serverInformation += "<br>" + this.distribution.toString();
        serverInformation += "<br>Active Task: " + this.task;
        serverInformation += "<br/>Next Event: " + this.timeNextEvent;
        return serverInformation;
    }
    getOutput(){
        return [
            {"Jobs" : this.jobsServiced},
            {"Service Time" : this.jobsServiced},
            {"Active Task" : this.task},
            {"Next Event" : this.timeNextEvent}
        ];
    }
```

```
  toTable(){
    var tableHeadings = ["Identifier", "Jobs", "Service Time", "Active Task", "Next Event", "Utilization"];
    var tableData = [this.identifier,this.jobsServiced, this.totalServiceTime/this.jobsServiced, this.task,
this.timeNextEvent, this.utilization];
    return generateTable(tableHeadings, tableData);
  }
}
class DiskComponent extends ServiceComponent{
  constructor(identifier, distribution){
    super(identifier, distribution);
    this.type = "Disk";
  }
}
class ArrivalComponent extends SimulationComponent{
  constructor(identifier, distribution){
    super(identifier, "arrival");
    this.identifier = identifier;
    this.distribution = distribution;
    this.jobsArrived = 0;

    //metrics needed
    this.interArrivalTime = 0;

    this.totalInterArrivalTime = 0;
    this.averageInterArrivalTime = 0;
    this.interArrivalCumulativeDeviation = 0;
    this.interArrivalTimeDeviation = 0;

    //User Output
```

```
        this.userHeadings = ["Identifier", "Mean Interarrival Time", "SD Interarrival Time", "Jobs Arrived"];
        this.userData = [this.identifier, this.averageInterArrivalTime, this.averageInterArrivalTimeDeviation,
this.jobsArrived];
        this.setResultIndices();
    }
    copyTask(){


    }
    updateUserData(){
        this.interArrivalTimeDeviation = Math.sqrt(this.interArrivalCumulativeDeviation/(this.jobsArrived) -
this.averageInterArrivalTime*this.averageInterArrivalTime);
        this.userData = [this.identifier, this.averageInterArrivalTime.toFixed(2),
this.interArrivalTimeDeviation.toFixed(2), this.jobsArrived-1];
    }
    generateNextArrival(){
        var generatedTime = this.distribution.generate();
        this.timeNextEvent += generatedTime;
        this.jobsArrived += 1;
        this.totalInterArrivalTime += generatedTime;
        this.averageInterArrivalTime = this.totalInterArrivalTime/(this.jobsArrived);
        this.interArrivalCumulativeDeviation += generatedTime*generatedTime;
        this.task = new Task(`Task ${this.jobsArrived}`, this.timeNextEvent);
        this.model.addEvent(this);
    }

    advanceTask(){
        this.next.acceptTask(new Task(`Task ${this.jobsArrived}`, this.timeNextEvent));
        this.generateNextArrival();
    }
    toString(){
        var arrivalInformation = this.identifier + ": "
        arrivalInformation += "<br/>Jobs: " + this.jobsArrived;
        arrivalInformation += "<br/>" + this.distribution.toString();
        arrivalInformation += "<br/>Active Task: " + this.task;
        arrivalInformation += "<br/>Next Event: " + this.timeNextEvent;
        return arrivalInformation;
    }
    toTable(){
        var tableHeadings = ["Identifier", "Jobs", "Inter-Arrival Time", "Active Task", "Next Event"];
        var tableData = [this.identifier,this.jobsArrived, this.totalInterArrivalTime/this.jobsArrived, this.task,
this.timeNextEvent];
        return generateTable(tableHeadings, tableData);
    }
}

class ExitComponent extends SimulationComponent{
    constructor(identifier){
        super(identifier, "Exit");
    }
    acceptTask(task){
        this.task = task;
        this.advanceTask(task);
    }
    advanceTask(task){
        cumulativeResponseTime += task.getRunTime();
        responseTimeCumulativeDeviation += task.getRunTime() * task.getRunTime();
        tasksCompleted++;
        this.task = null;
    }
    updateUserData(){
        return "";
    }
    updateUserHeadings(){
        return "";
    }
}

class FeedbackServer extends ServiceComponent{
    constructor(identifier, distribution, feedbackProbability){
        super(identifier, distribution);
        this.feedbackProbability = feedbackProbability/100;
    }
    connectExit(exitPoint){
        this.exitPoint = exitPoint;
    }
    advanceTask(){
        var task = JSON.parse(JSON.stringify(this.task));

        if(this.next != null){
            var randomValue = Math.random();
            randomValue < this.feedbackProbability ? this.choseBelow++ : this.choseAbove++;
            if(randomValue < this.feedbackProbability){
                this.visual.setTaskDestination(this.task, "Loop");
                this.next.acceptTask(this.task);
                this.tasksReturned++;
            }
            else{
                this.visual.setTaskDestination(this.task, "Exit");
            this.exitPoint.acceptTask(this.task);
            this.tasksExitted++;
          }
        }
        this.jobsServiced++;
        this.updateMetrics();

        this.available = true;
        this.timeNextEvent = null;
        this.task = null;

        //If there is a connected queue, notify that server is available
        if(this.connectedQueue != null)
          this.connectedQueue.advanceTask(task);
    }
}

class ParallelComponent extends SimulationComponent{
    constructor(identifier, objectType, distribution, numberOfElements, model){
        super(identifier, "parallel block");
        this.objectType = objectType;
        this.distribution = distribution;
        this.numberOfElements = numberOfElements;
        this.containedElements = [];
        this.objectIndices = {};
        this.demandedTasks = [];
        this.connectedQueue = null;
        this.model = model;
        this.placeAllObjects();
```

```
}
copyTask(){
   /*
   this.visual.reset();
   for(var key in this.objectIndices){
      var assignedComponent = this.objectIndices[key];
      var assignedTask = assignedComponent.task;
      if(assignedTask !== null)
         this.visual.placeTask(assignedTask.identifier,  key);
   }
   */


}
updateUserData(){


}
userHeadingsToTable(){
   return "";//this.containedElements[0].userHeadingsToTable();
}
userDataToTable(){
   var output = "";
   output += this.containedElements[0].userHeadingsToTable();
   for(const key in this.objectIndices){
     output += this.objectIndices[key].userDataToTable();
   }
   return output;
}
connectQueue(queue){
    this.connectedQueue = queue;
    for(var i = 0; i < this.containedElements.length; i++){
        this.containedElements[i].connectQueue(this.connectedQueue);
    }
}
connectInteriorObjects(){
    for(var i = 0; i < this.containedElements.length - 1; i++){
        for(var j = 0; j < this.containedElements[i].length; j++){
            connectArrow(this.containedElements[i][j], this.containedElements[i+1][j]);
        }
    }
}
placeAllObjects(){
    if(!Array.isArray(this.objectType)){
        this.placeObjects(this.objectType, this.containedElements);
    }
    else {
        for(var i = 0; i < this.objectType.length; i++){
            if(this.containedElements[i] === undefined){
                this.containedElements.push([]);
            }
            this.placeObjects(this.objectType[i], this.containedElements[i]);
        }

        for(var i = 0; i < this.numberOfElements; i++){
            for(var j = 1; j < this.objectType.length; j++){
                this.containedElements[j-1][i].setNext(this.containedElements[j][i]);
                if(this.containedElements[j-1][i].type == "Queue"){
                    this.containedElements[j][i].connectQueue(this.containedElements[j-1][i]);
            }
          }
        }

        var tempArray = [];
        for (var i = 0; i < this.numberOfElements; i++){
          for( var j = 0; j < this.objectType.length; j++)
            tempArray.push(this.containedElements[j][i]);
        }
        this.containedElements = tempArray;
      }
    }

    placeObjects(object, resultArray){
      if(Array.isArray(this.distribution)){
        for (var i = 0; i < this.numberOfElements; i++){
          resultArray.push(new object(`${this.identifier} `, this.distribution[i]));
          resultArray[i].identifier += resultArray[i].type + ` ${i+1}`;
          resultArray[i].model = this.model;
          this.objectIndices[resultArray[i].identifier] = resultArray[i];
          this.model.addComponent(resultArray[i]);
        }
      }
      else{
        for (var i = 0; i < this.numberOfElements; i++){
          resultArray.push(new object(`${this.identifier} `, this.distribution));
          resultArray[i].identifier += resultArray[i].type + ` ${i+1}`;
          resultArray[i].model = this.model;
          this.objectIndices[resultArray[i].identifier] = resultArray[i];
```

```
            this.model.addComponent(resultArray[i]);
        }
    }
}
setNext(drawableObject){
    this.next = drawableObject;
    if( drawableObject != undefined){
        for(var i = 0; i < this.containedElements.length; i++){
            if(this.containedElements[i].next == undefined)
                this.containedElements[i].setNext(this.next);
        }
        this.next.previous = this;
    }
}
acceptTask(task){
    var success = false;
    var availableComponents = [];
    if(!Array.isArray(this.objectType)){
        for(var i = 0; i < this.containedElements.length; i++){
            if(this.containedElements[i].checkAvailability()){
                availableComponents.push(this.containedElements[i]);
                success = true;
            }
        }
    }
    else{
        for(var i = 0; i < this.containedElements.length; i++){
            if(this.containedElements[i].type == "Queue"){
                availableComponents.push(this.containedElements[i]);
                success = true;
            }
        }
    }
    if(success){
        var numberAvailable = availableComponents.length;
        var randomValue =  Math.floor(Math.random() * numberAvailable);
        availableComponents[randomValue].acceptTask(task);
        var assignedComponent = this.objectIndices[availableComponents[randomValue].identifier];
        assignedComponent.task = task;
        this.visual.placeTask(task.identifier, assignedComponent.identifier);
    }
    if (!success){
        this.demandedTasks.push(task);
    }
}
toTable(){
    var output = "";
    this.containedElements.forEach(element => {
        output += element.toTable();
    });
    return output;
}
checkAvailability(){
    var success = false;
    for(var i = 0; i < this.containedElements.length; i++){
        if(this.containedElements[i].checkAvailability()){
            success = true;
            i = this.containedElements.length;
            }
        }
        return success;
    }
}

class WorkstationComponent extends ServiceComponent{
    constructor(identifier, distribution){
        super(identifier, distribution);
        var workstationTask = new Task(identifier + "|task", currentTime);
        this.assignedTask  = workstationTask;
        this.task = workstationTask;
        this.type = "Workstation";
        this.lastGeneratedServiceTime = 0;
        this.userHeadings=["Identifier", "Utilization", "Think Time", "SD Think Time", "Jobs Submitted"];
        this.setResultIndices();
    }
    acceptTask(task){
        this.task = task;
        //Generate Random Service Time Based on Distribution

        this.lastGeneratedServiceTime = this.distribution.generate();
        this.timeNextEvent = this.lastGeneratedServiceTime + currentTime;
        this.available = false;
        this.model.addEvent(this);

        //UPDATE GENERAL INFORMATION
        if(currentTime != 0){
            cumulativeResponseTime += task.getRunTime();
            responseTimeCumulativeDeviation += task.getRunTime() * task.getRunTime();
            tasksCompleted++;
            task.creationTime = currentTime;
        }
        //Return Information Regarding Next Event for This Object
        return {"component": this.identifier, "time": this.timeNextEvent, "task": task};
    }
}

class ParallelWorkstationComponent extends ParallelComponent{
    constructor(identifier, distribution, numberOfElements, model){
        super(identifier, WorkstationComponent, distribution, numberOfElements, model);
        this.assignedTaskMapping = {};
        this.setAssignedTasks();
    }
    setAssignedTasks(){
        this.containedElements.forEach(element => {
            element.assignedTask.identifier = element.identifier + "|task";
            this.assignedTaskMapping[element.task.identifier] = element;
            element.acceptTask(element.assignedTask);
        });
    }
    getServerFromTask(task){
        return this.assignedTaskMapping[task.identifier];
    }
    checkAvailability(){
        return true;
    }
    acceptTask(task){
        var workstation = this.getServerFromTask(task);
        workstation.acceptTask(task);
        this.visual.placeTask(task.identifier, workstation.identifier);
    }
}

class ParallelDiskComponent extends ParallelComponent{
    constructor(identifier, objectType, diskDistributions, diskCount, diskWeights, model){
        super(identifier, objectType, diskDistributions, diskCount, model);
        this.disks = [];
        this.queues = [];
        this.availableComponents = [];
        this.setAdjustedWeights(diskWeights);
    }

    setAdjustedWeights(diskWeights){
        for(var i = 0; i < this.containedElements.length; i++){
            if(this.containedElements[i].type == "Disk"){
                this.disks.push(this.containedElements[i]);
            }
            else{
                this.queues.push(this.containedElements[i]);
                this.availableComponents.push(this.containedElements[i]);
            }
        }



        var sumOfWeights = 0;
        for(var i = 0; i < diskWeights.length; i++){
            this.disks[i].weight = parseFloat(diskWeights[i]);
            sumOfWeights += this.disks[i].weight;
        }

        var weightWatcher = 0;
        for(var i = 0; i < this.disks.length; i++){
            if(this.disks[i].weight == 0){
                this.disks[i].adjustedWeight = 0;
                this.disks[i].randomBoundary = -1;
            }
            else{
                this.disks[i].adjustedWeight = this.disks[i].weight / sumOfWeights;
                weightWatcher += this.disks[i].adjustedWeight;
                this.disks[i].randomBoundary = weightWatcher;
            }
        }
    }
    acceptTask(task){
        var randomValue =  Math.random();
        var assignedComponent = null;
        for(var i = 0; i < this.disks.length && assignedComponent == null; i++){
            if(randomValue < this.disks[i].randomBoundary){
                assignedComponent = this.disks[i];
            }
        }
        assignedComponent.connectedQueue.acceptTask(task);
        this.visual.placeTask(task.identifier, assignedComponent.connectedQueue.identifier);
    }
}
```

simulationModels.js

```
var maxTasks = 1000;

var maxEvents = 10000;

var maxQueueSize = 500;


class Simulation{

   constructor(identifier, model, maximumEvents, maximumTasks){

      this.identifier = identifier;

      this.maximumEvents = maximumEvents;

      this.maximumTasks = maximumTasks;

      this.model = model;

      this.tableOutputs = [];

   }

   processEvent(){

      this.model.processEvent();

   }

   runNumberOfEvents(eventCount){

      for(var i = 0; i < eventCount; i++){

         this.processEvent();

      }

   }

   runSimulation(){

      while(this.model.eventsProcessed < eventsToRun && tasksCompleted < maximumTasks){

         this.processEvent();

      }

   }

}
```

```
class Model{
  constructor(identifier){
      this.identifier = identifier;
      this.components = {};
      this.events = [];
      this.eventsProcessed = 0;
      this.visualModel = null;
      this.enableVisualization = true;
      //Results
      this.throughput = 0;
      this.throughputDeviation = 0;
      this.throughputCumulativeDeviation = 0;

      this.responseTime = 0;
      this.responseTimeDeviation = 0;
      this.responseTimeCumulativeDeviation = 0;

      this.userHeadings = ["Identifier", "Mean Response Time",  "SD Response Time", "Throughput", "Jobs
Completed", "Events Processed"];
      this.userData = [this.identifier, this.responseTime, this.responseTimeDeviation, this.throughput,
tasksCompleted, this.eventsProcessed];
      //Organization for user output
      this.compByType = {};

      this.resultIndices = {};
      this.selectedResults = [];
  }
  setResultIndices(){
      for(var i = 1; i < this.userHeadings.length; i++){
          this.resultIndices[this.userHeadings[i]] = i;
      }
  }
  setSelectedResults(selectedResults){
      for(var i = 0; i < selectedResults.length; i++){
          this.selectedResults.push(selectedResults[i]);
          if(this.getResult(selectedResults[i]) == undefined){
              console.log("Invalid selected result! " + selectedResults[i]);
          }
      }
  }
  getSelectedResults(){
      this.updateUserData();
      var output = "";
      var label;
      var data;
      for(var i = 0; i < this.selectedResults.length; i++){
          label = this.selectedResults[i];
          data = outputNumber(this.getResult(label));
          output += generateLabeledRow(label, data);
      }
      return output;
  }
  getSelectedTable(){
      var output = "";
      var label;
      var data;
      for(var i = 0; i < selectedResults.length; i++){
          label = selectedResults[i];
          data = this.getResult(label);
          output += generateLabeledRow(label, data);
      }
      return output;
  }
  getDataResults(resultNames){
      var output = "";
      var results = [];
      for(var i = 0; i < resultNames.length; i++){
          results.push(this.getResult(resultNames[i]));
      }
      return generateTableData(results);
  }
  getHeadings(headingNames){
      return generateTableHeadings(headingNames);
  }

  getResult(resultName){
      return this.userData[this.resultIndices[resultName]];
  }
  copyAllTasks(){
      this.visualModel.reset();

      for(var key in this.compByType){
          if(key != "Queue"){
              var typeArray = this.compByType[key];
              for(var i = 0; i < typeArray.length; i++){
                  typeArray[i].copyTask();
              }
          }
      }
      for(var i = 0; i < this.compByType["Queue"].length; i++){
           this.compByType["Queue"][i].copyTask();
       }

  }
  copyTask(taskIdentifier, destination){
      this.visualModel.createVisualTask(taskIdentifier, destination);
  }
  displayResults(){
      return this.displayResultsByType();
  }
  displayResultsByType(){
      var output = `<table>`;
      output += this.userOutputTable();
      output += `</table>`;
      for(const key in this.compByType){
          for(var i = 0; i < this.compByType[key].length; i++){
              output+= `<table>`
              var temp = this.compByType[key][i].userOutputTable();
              if (temp != undefined)
                  output += `<table> ${temp} </table>`;
          }

      }
      return output + `</table>` + `<div></div>`;
  }
```

```
    userOutputTable(){
        this.updateUserData();
        if(this.userData !== undefined)
            return generateOutputTables(this.userHeadings, this.userData);
    }
    userHeadingsToTable(){
        return generateTableHeadings(this.userHeadings);
    }
    userDataToTable(){
        this.updateUserData();
        if(this.userData !== undefined)
            return generateTableData(this.userData);
        else
            console.log("User data for: " + this.identifier + " is undefined");
        return "";
    }
    updateUserData(){
        this.responseTime = cumulativeResponseTime / tasksCompleted;
        this.throughput = tasksCompleted / currentTime;
        this.responseTimeDeviation = Math.sqrt(responseTimeCumulativeDeviation/tasksCompleted -
this.responseTime*this.responseTime);
        this.userData = [this.identifier, this.responseTime.toFixed(2),
this.responseTimeDeviation.toFixed(2), this.throughput.toFixed(2), tasksCompleted,
this.eventsProcessed];
    }
    addComponent(component){
        this.components[component.identifier] = component;
        component.model = this;
    if(this.compByType[component.type] === undefined){
        this.compByType[component.type] = [];
    }
    this.compByType[component.type].push(component);
}
setNext(first, second){
    first.setNext(second);
}
linkComponent(first, second){
    second.setConnection(first);
}
//adds event then sorts according to time, running time for sorted list should not take long enough
//to need creation of nodes and insertion sort
addEvent(target){
    this.events.push(target);
    this.events = this.events.sort((a, b) => a.timeNextEvent > b.timeNextEvent ? 1 : -1);
}
updateEvent(target){
    var index = this.events.findIndex(target);
    if(index == -1)
        this.addEvent(target);
    else{
        this.events[index].timeNextEvent = target.timeNextEvent;
        this.events = this.events.sort((a, b) => a.timeNextEvent > b.timeNextEvent ? 1 : -1);
    }
}
//runs the event
processEvent(){
    var target = this.events.shift();
    //update visual component
    if(this.enableVisualization){
        this.updateVisual(target);
    }

    //Advance the simulation
    currentTime = target.timeNextEvent;
    target.advanceTask();
    this.eventsProcessed++;
}
getNextEventTime(){
    if(this.events.length > 0)
        return this.events[0].timeNextEvent;
    else return null;
}
toTable(){
    var tableHeadings = ["Identifier", "Contained Events"];

    var events = "";

    this.events.forEach(element => {
        events += element.identifier + ": " + element.timeNextEvent + " | ";
    });

    var tableData = [this.identifier, events];

    var output = "<tr>";
    tableHeadings.forEach(element => {
        output += `<th>${element}</th>`;
    });
    output += "</tr><tr>"
    tableData.forEach(element => {
        output += `<td>${element}</td>`;
    });
    output += "</tr>";

    return output;
}
componentsToTable(){
    var output = "";
    for(const key in this.components){
        output += this.components[key].toTable();
    }
    return output;
}
getUserData(){
    var output = "";
    for(const key in this.components){
        output += this.components[key].userDataToTable();
    }
    return output;
}
getTitle(){
    return this.toTable();
}
getResults(){
    return this.componentsToTable();
}
```

```
  updateVisual(simulationObject){
    var object, time, type, task;
    //object = simulationObject.identifier;
    time = simulationObject.timeNextEvent;
    type = simulationObject.type == "arrival" ? "accept" : "advance";
    if(simulationObject.task == null){
      console.log("TASK IS NULL FOR: " + simulationObject.identifier);
    }
    task = simulationObject.task.identifier;
    this.visualModel.createEvent(simulationObject,  time, type, task);
  }

  connectVisualModel(visualModel){
    this.visualModel = visualModel;
    for (const key in this.components){
      var component = this.components[key];
      var visualComponent = visualModel.components[component.identifier];
      if(visualComponent != undefined)
        component.visual = visualComponent;
    }
  }

}
class SingleServerModel extends Model{
  constructor(arrivalDistribution, serverDistribution){
    super("Single Server");

    //Create Components For Model
    var arrivalComponent = new ArrivalComponent("Arrivals", arrivalDistribution);
```

```
        var queueComponent = new QueueComponent("Queue");
        var serverComponent = new ServiceComponent("Server", serverDistribution);
        var exitComponent = new ExitComponent("Exit");

        //Add Components to the Model
        this.addComponent(arrivalComponent);
        this.addComponent(queueComponent);
        this.addComponent(serverComponent);
        this.addComponent(exitComponent);

        //Link Components
        this.setNext(arrivalComponent, queueComponent);
        this.setNext(queueComponent, serverComponent);
        serverComponent.connectQueue(queueComponent);
        this.setNext(serverComponent, exitComponent);

        //Set Initial Event
        arrivalComponent.generateNextArrival();
    }
}
class ParallelServersModel extends Model{
    constructor(arrivalDistribution, serverDistributions, serverCount){
        super("Parallel Servers");

        //Create Model Components
        var arrivalComponent = new ArrivalComponent("Arrivals", arrivalDistribution);
        var queueComponent = new QueueComponent("Queue");
        var parallelServers = new ParallelComponent("Parallel", ServiceComponent, serverDistributions,
serverCount, this);
```

```
    var exitPoint = new ExitComponent("Exit");
    //Add to Model
    this.addComponent(arrivalComponent);
    this.addComponent(queueComponent);
    this.addComponent(parallelServers);
    this.addComponent(exitPoint);

    //Link Components
    this.setNext(arrivalComponent, queueComponent);
    this.setNext(queueComponent, parallelServers);
    parallelServers.connectQueue(queueComponent);
    this.setNext(parallelServers, exitPoint);

    //Initialize Model
    arrivalComponent.generateNextArrival();

}
displayResults(){
    var results = [];
    var parallelServersOutput = `<table>`;
   // parallelServersOutput += generateTableHeadings(["Parallel Servers Model"]);
    parallelServersOutput += this.userOutputTable();
    parallelServersOutput += `</table>`;

    //OUTPUTS
    var serverOutput =`<table class="title-table"> ${generateTableHeadings(["Servers"])} </table>`;
    serverOutput += `<table class="tiny-table">`;
    serverOutput += generateTinyHeadings(["ID", "U", "S", "SD S"]);
    for(var i = 0; i < this.compByType["Server"].length; i++){
            results = this.compByType["Server"][i].getResultValues(["Utilization", "Mean Service Time", "SD
Service Time"]);
            results.unshift(i + 1);
            serverOutput += generateTinyTable(results);
        }
        serverOutput += "</table>";

        var queueOutput = "<table>" + this.components["Queue"].userOutputTable() + "</table>";
        var outputs = [
            parallelServersOutput,
            queueOutput,
            serverOutput
        ]

        var output = "";
        outputs.forEach(element => {
            output += element;
        });
        return output;
    }
}
class TwoServersModel extends Model{
    constructor(arrivalDistribution, serverDistributions){
        super("Two Servers");

        //Create Model Components
        var arrivalComponent = new ArrivalComponent("Arrivals", arrivalDistribution);
        var queueComponent = new QueueComponent("Queue");
        var parallelServers = new ParallelComponent("Parallel", ServiceComponent, serverDistributions, 2,
this);
        var exitPoint = new ExitComponent("Exit");
        //Add to Model
        this.addComponent(arrivalComponent);
        this.addComponent(queueComponent);
        this.addComponent(parallelServers);
        this.addComponent(exitPoint);

        //Link Components
        this.setNext(arrivalComponent, queueComponent);
        this.setNext(queueComponent, parallelServers);
        parallelServers.connectQueue(queueComponent);
        this.setNext(parallelServers, exitPoint);

        //Initialize Model
        arrivalComponent.generateNextArrival();
    }
}

class SingleFeedbackModel extends Model{
    constructor(arrivalDistribution, serverDistribution, feedbackProbability){
       super("Single Server (Feedback)");

       //Create Model Components
       var arrivalComponent = new ArrivalComponent("Arrivals", arrivalDistribution);
       var queueComponent = new QueueComponent("Queue");
       var serverComponent = new FeedbackServer("Server", serverDistribution, feedbackProbability);
       var exitPoint = new ExitComponent("Exit");
```

```
        //Add to Model
        this.addComponent(arrivalComponent);
        this.addComponent(queueComponent);
        this.addComponent(serverComponent);
        this.addComponent(exitPoint);

        //Link Components
        this.setNext(arrivalComponent, queueComponent);
        this.setNext(queueComponent, serverComponent);
        serverComponent.connectQueue(queueComponent);
        this.setNext(serverComponent, queueComponent);
        serverComponent.connectExit(exitPoint);

        //Initialize
        arrivalComponent.generateNextArrival();
    }
}

class InteractiveModel extends Model{
    constructor( workstationDistribution, workstationCount, serverDistribution, serverCount){
        super("Interactive Model");
        var taskColors = {};
        //Create Model Components
        var parallelWorkstations = new ParallelWorkstationComponent("Parallel", workstationDistribution,
workstationCount, this);
        var parallelServers = new ParallelComponent("Servers", ServiceComponent, serverDistribution,
serverCount, this);
        var queueComponent = new QueueComponent("Queue");
        //Add to Model
        this.addComponent(parallelWorkstations);
        this.addComponent(parallelServers);
        this.addComponent(queueComponent);

        //Link Components
        this.setNext(parallelWorkstations, queueComponent);
        this.setNext(queueComponent, parallelServers);
        this.setNext(parallelServers, parallelWorkstations);
        parallelServers.connectQueue(queueComponent);

        this.setResultIndices();
        this.setSelectedResults(["Mean Response Time", "SD Response Time", "Throughput"]);

        createVisualTask = function(x,y, identifier){
            if(taskColors[identifier] == undefined){
                var task = new TaskVisual(x, y, identifier, generateColor());
                taskColors[identifier] = task.color;
                return task;
            }
            else{
                return new TaskVisual(x, y, identifier, taskColors[identifier]);
            }
        }
    }

    addInitialVisualEvents(){
        if(this.visualModel === null || this.visualModel === undefined){
            console.log("no visual model connected");
        }
        else{
            var workstations = this.components["Parallel"].containedElements;

            for(var i = 0; i < workstations.length; i++){
                this.visualModel.createVisualTask(workstations[i].task.identifier,
this.visualModel.components[workstations[i].identifier]);
                var task = this.visualModel.tasks[workstations[i].task.identifier];
                this.visualModel.components[workstations[i].identifier].acceptTask(task);
            }
        }
    }
    displayResults(){
        var results = [];
        var interactiveOutput = `<table>`;
        interactiveOutput += generateTableHeadings(["Interactive Model"]);
        interactiveOutput += this.getSelectedResults();
        interactiveOutput += `</table>`;

        //OUTPUTS
        var workstationOutput =`<table class="title-table"> ${generateTableHeadings(["Workstations"])}
</table>`;
        workstationOutput += `<table class="tiny-table">`;
        workstationOutput += generateTinyHeadings(["ID", "U", "T", "SD T"]);
        for(var i = 0; i < this.compByType["Workstation"].length; i++){
            results = this.compByType["Workstation"][i].getResultValues(["Utilization", "Think Time", "SD
Think Time"]);
            results.unshift(i + 1);
            workstationOutput += generateTinyTable(results);
        }
        workstationOutput += "</table>";
        workstationOutput += `<div class="tiny-information"> U = Utilization </br> T = Think Time</div>`;


        var serverOutput =`<table class="title-table"> ${generateTableHeadings(["Servers"])} </table>`;
        serverOutput += `<table class="tiny-table">`;
        serverOutput += generateTinyHeadings(["ID", "U", "S", "SD S"]);
        for(var i = 0; i < this.compByType["Server"].length; i++){
            results = this.compByType["Server"][i].getResultValues(["Utilization", "Mean Service Time", "SD
Service Time"]);
            results.unshift(i + 1);
            serverOutput += generateTinyTable(results);
        }
        serverOutput += "</table>";
        serverOutput += `<div class="tiny-information"> U = Utilization, S = Service Time </div>`;


        var outputs = [
            interactiveOutput,
            workstationOutput,
            serverOutput
        ]


        var output = "";
        outputs.forEach(element => {
            output += element;
        });
        return output;
    }
}
class CentralServerModel extends Model{
   constructor( centralDistribution, channelDistribution, diskDistributions, diskCount, jobCount, 
diskWeights){

        super("Central Server");

        this.jobCount = jobCount;


        //Create Components

        var centralProcessorQueue = new QueueComponent("Processor Queue");

        //var centralProcessor = new ServiceComponent("Central Processor", centralDistribution);

        var centralProcessor = new ParallelComponent("Central Processor", ServiceComponent, 
centralDistribution, 4, this);


        var channelQueue = new QueueComponent("Channel Queue");

        var channel = new ServiceComponent("Channel", channelDistribution);

        var parallelDisks = new ParallelDiskComponent("Parallel", [QueueComponent, DiskComponent], 
diskDistributions, diskCount, diskWeights, this);


        //Add to Model

        this.addComponent(centralProcessorQueue);

        this.addComponent(centralProcessor);

        this.addComponent(channelQueue);

        this.addComponent(channel);

        this.addComponent(parallelDisks);


        //Link Components

        this.setNext(centralProcessorQueue, centralProcessor);

        this.setNext(centralProcessor, channelQueue);

        this.setNext(channelQueue, channel);

        this.setNext(channel, parallelDisks);

        this.setNext(parallelDisks, centralProcessorQueue);
```

```
    centralProcessor.connectQueue(centralProcessorQueue);
    channel.connectQueue(channelQueue);

    //for output
    for(var i = 1; i <= parallelDisks.numberOfElements; i++){
      parallelDisks.objectIndices[`Parallel Queue ${i}`].setSelectedResults( ["Current Queue Length"] );
      parallelDisks.objectIndices[`Parallel Disk ${i}`].setSelectedResults( ["Utilization", "Mean Service
Time"] );
    }

    const desiredResults = {
      /*
      "Central Processor": [
        "Utilization",
        "Mean Service Time",
        "SD Service Time"
      ],
      */
      "Processor Queue": [
        "Current Queue Length"
      ],
      "Channel": [
        "Utilization",
        "Mean Service Time",
        "SD Service Time"
      ],
      "Channel Queue": [
        "Current Queue Length"
      ]
    }


    for(var key in desiredResults){
        this.components[key].setSelectedResults( desiredResults[key] );
    }


    this.setResultIndices();
    this.setSelectedResults(["Mean Response Time",  "SD Response Time", "Throughput"]);
}
addInitialEvents(){
    var processorQueue = this.components["Processor Queue"];
    var centralProcessor = this.components["Central Processor"];
    for(var i = 0; i < this.jobCount; i++){
        processorQueue.acceptTask(new Task(`Task ${i}`, 0));



        //var visualTask = createVisualTask(0,0, `Task ${i}`);
        //this.visualModel.addTask(visualTask);
        //visualTask.visible = false;
        //this.visualModel.components["Processor Queue"].receiveTask(visualTask);


    }
    for(var i = 0; i < centralProcessor.containedElements.length; i++){
        centralProcessor.containedElements[i].copyTask();
    }


    processorQueue.copyTask();


    console.log(centralProcessor);
```

```
        console.log(processorQueue);
    }
    displayResults(){
        //OUTPUTS
        var centralOutput = "<table>";
        centralOutput += generateTableHeadings(["Central Processor"]);
        centralOutput += this.components["Central Processor"].getSelectedResults();
        centralOutput += this.components["Processor Queue"].getSelectedResults();
        centralOutput += "</table>";

        var channelOutput = "<table>";
        channelOutput += generateTableHeadings(["Channel"]);
        channelOutput += this.components["Channel"].getSelectedResults();
        channelOutput += this.components["Channel Queue"].getSelectedResults();
        channelOutput += "</table>";



        var parallelDisks = this.components["Parallel"];



        var parallelOutputs = "";


         //OUTPUTS
        var parallelOutputs =`<table class="title-table"> ${generateTableHeadings(["Workstations"])}
</table>`;
        parallelOutputs += `<table class="tiny-table">`;
        parallelOutputs += generateTinyHeadings(["ID", "U", "S", "Q"]);
        var results;
        for(var i = 1; i <= parallelDisks.numberOfElements; i++){
            results = parallelDisks.objectIndices[`Parallel Disk ${i}`].getResultValues(["Utilization", "Mean
Service Time"]);
            results.unshift(i + 1);
            results.push(parallelDisks.objectIndices[`Parallel Queue ${i}`].getResultValues(["Current Queue
Length"]));
            parallelOutputs += generateTinyTable(results);
        }
        parallelOutputs += "</table>";


        /*
        for(var i = 1; i <= parallelDisks.numberOfElements; i++){
            parallelOutputs += "<table>";
            parallelOutputs += generateTableHeadings([`Disk ${i}`]);
            parallelOutputs += parallelDisks.objectIndices[`Parallel Disk ${i}`].getSelectedResults();
            parallelOutputs += parallelDisks.objectIndices[`Parallel Queue ${i}`].getSelectedResults();
            parallelOutputs += "</table>";
        }
        */
        var outputs = [
            centralOutput,
            channelOutput,
            parallelOutputs
        ]


        var output = "";
        outputs.forEach(element => {
            output += element;
        });
        return output;
```

```
    }
}

//for output
function addTableRow(tableIdentifier, textResults){
    var table = document.getElementById(tableIdentifier);
    var output = "<tr>";
    textResults.forEach(element => {
        output += `<td>${element}</td>`;
    });
    output += "</tr>";
    table.innerHTML += output;
}
function addTableHeading(tableIdentifier, textResults){
    var table = document.getElementById(tableIdentifier);
    var output = "<tr>";
    textResults.forEach(element => {
        output += `<th>${element}</th>`;
    });
    output += "</tr>";
    table.innerHTML += output;
}
function addTable(identifier, location){
    var destination = document.getElementById(location);
    destination.innerHTML += `<table id="${identifier}"></table>`;
}
function createTableSection(tableIdentifier){
    document.body.innerHTML += `<div id="${tableIdentifier}Div"></div>`;
    addTable(`${tableIdentifier}`, `${tableIdentifier}Div`);
```

```
   return document.getElementById(tableIdentifier);
}
```