BLUESIM PRODUCT PROPOSAL

V 1.1



Aaron Jackson

Ajit Sutreja

Arthur Collé

David Grant

George Menjivar

Gonzalo Martinez

Lawrence Niu

Mark Tulchinskiy

Mehran Mizani

Nicolas Homble

Sam Edwards

Timothy Bohannon

Contents

[1 Introduction 2](#_Toc417320464)

[1.1 Determining the Problem Space 2](#_Toc417320465)

[1.2 Goals 2](#_Toc417320466)

[1.3 Audience 3](#_Toc417320467)

[1.4 Definitions 3](#_Toc417320468)

[2 Project Overview 4](#_Toc417320469)

[3.1 Requirements 5](#_Toc417320470)

[3.3 Workflow 7](#_Toc417320471)

[4 Delivery Manifest 10](#_Toc417320472)

[4.1 Deliverables 10](#_Toc417320473)

[4.3 Verification 10](#_Toc417320474)

[4.4 Validation 11](#_Toc417320475)

[4.5 Targeted Dates 16](#_Toc417320476)

[4.6 Cost Estimate 17](#_Toc417320477)

[4.7 Risks 18](#_Toc417320478)

[5 Documentation 18](#_Toc417320479)

[6 Intellectual Property Statement 19](#_Toc417320480)

[7 Summary 19](#_Toc417320481)

[8 Sources 19](#_Toc417320482)

# 

# 1 Introduction

BLUESIM is a practical simulation tool that supports a wide variety of potential simulations. Its limited scope enables it to excel in the creation of quick, efficient simulations with an easily manageable and gentle learning curve, backed by strong documentation and an architecture that lends itself to future enhancements.

## 1.1 Determining the Problem Space

Simulation software is extremely useful in its ability to model large and complex systems. When working with simulation software, years of training and experience are accepted as a natural necessity because of the ultimate value in a simulation’s data gathering capabilities. Learning how to create and gather information from simulations is a crucial skill for many technical professions. However, learning still requires significant time, practice, and familiarity with various programming languages. To this end, there is no niche simulation software for users looking for model simple problems in a relatively short amount of time; users interested in simulating students transferring between colleges, users who need to show the flow of data from point A to point B, or any other number of quick simulation requirements.

Users interested in simply creating small-scale simulations are largely discouraged by the steep learning curve and time investment required by professional software. Currently, users are left with clunky tools such as Microsoft PowerPoint and Prezi that allow users to easily generate a few animations but leave much to be desired in the terms of actual computation. This type of software is considered to be more of an articulation tool than a simulation tool. On the other hand, there are several simulation libraries publically available such as sim.js and SimPy, JavaScript and Python2 implementations of discrete event simulators. These libraries open up simulation software to amateur developers, as a user with coding experience could build up their own simulation from the ground up. However, this is not viable for a great majority of our audience considering the requirement to have preexisting knowledge of JavaScript and/or Python2, and a significant time investment to program a simulation from scratch. There is a vital necessity for a tool that can cater to non-technically inclined users who want to build something more informative than an extravagant slideshow, and there needs to be a tool that would allow technical users to create their own simulations quickly. To this end, we present BLUESIM.

## 1.2 Goals

BLUESIM is a web application that allows users with little to no programming skills to create custom simulations that model the flow of data within a system with corresponding graphical representations of that simulation. The primary goals of our product are ease of use, ease of customization, and ease of sharing.

BLUESIM utilizes a simple drag and drop UI as its method of customization. This interface provides a great visibility to the simulation software, where the individual pieces of the simulation, known as entities, are easily viewed in the main HTML5 canvas. BLUESIM functions as both an articulation tool and a simulation tool that allows users to see their edits and modifications in real-time. Users will also be able to move around pieces of their simulation and make dynamic changes to their simulation almost instantaneously. If the user requires more advanced customization, however, our product allows custom input.

BLUESIM also allows for the generation of saved simulations. A simulation may be saved as a text JSON file, which allows a simulation to saved, shared, and distributed as necessary. Storage for the saved text files is client-side, as BLUESIM does not save states on any server. Since BLUESIM is hosted online, users can easily share their simulations between each other and/or multiple compatible devices. The product can even be downloaded and hosted locally to allow the simulation to be as portable as the user requires.

## 1.3 Audience

The BLUESIM simulation tool is designed to be accessible and intuitive to users ranging from beginner simulation users to those experienced with coding and simulation systems. Targeted users are interested in a gentle learning curve with quick results. BLUESIM is modeled heavily against popular drag and drop software like Alice and Scratch, which are great introductions in computer programming by minimizing coding requirements and focusing on actual problem solving. Current simulation software is typically too robust for the simple simulations this audience is trying to accomplish.

BLUESIM makes it easy for power users to write simulations quickly using more advanced features. The drag and drop feature can be bypassed by those users familiar with JSON that want to write their simulations quickly in a text file (or generate the file in their own program). Our tool also serves as a nice introduction to the backend API sim.js that is a complete discrete event simulation library. Our targeted advanced users are interested in a tool that allows for simple simulations but at a fraction of the time of other professional packages.

## 1.4 Definitions

* Entity - Object that represents a splitter, source, sink, queue, or function splitter.
* GUI - Graphical user interface.
* API - Application Programming Interface.
* User Defined Object (UDO) - A representation of real world objects with user-defined attributes.
* Attributes - Parameters associated with a UDO.
* Canvas - Sandbox on webpage in which the user creates simulations with user defined entities and UDOs.
* Paths - Connections between entities; used by UDOs to travel between entities; represented by a line.
* Splitter - An entity that separates input data from a single path and outputting it to two possible divergent paths.
* Sink - An entity through which UDOs exit the simulation. Upon running the simulation, each sink entity will display simulation data in a table.
* Source - An entity that generates the UDOs that flow through the system.
* Queue - An entity that contains a finite series of UDOs that must exit in the same order they come in. Upon running the simulation, each queue entity will display simulation data in a table.
* D3 - is a JavaScript library for manipulating documents based on data. D3 helps you bring data to life using HTML, SVG, and CSS.
* Raphael - Raphael is a small JavaScript library that simplifies your work with vector graphics on the web.
* Sim.js - An event-based discrete-event simulation library based on standard JavaScript. The library was written in order to enable simulation within standard browsers by utilizing web technology.
* JSON - Human-readable text that represents objects.
* UX - User experience.
* UI - User Interface.

# 2 Project Overview

BLUESIM is an interactive in-browser simulation tool giving users the option to choose from any number of UDOs and link these entities in order to simulate the environment of their choice. Custom JSOn input entities give users a set of library functions to choose from in the event that the available basic tools do not fulfill the user’s simulation requirements. BLUESIM will receive that code and, when the data reaches a function block, it will execute the library functions, creating more advanced simulations.

BLUESIM will generate statistics from the UDOs when the simulation is run. The resulting statistics are then displayed both visually and numerically to the user. BLUESIM utilizes a JavaScript library that will provide a complete simulation library and expands upon this library with an intuitive GUI to allow users to interact with the existing API. This product also allows for the sharing of simulations between users and devices through the generation of seeds.

The project control plan involves distributing the workload between two primary teams, the “Frontend” and “Backend” teams.

**Frontend**

The team will be responsible for the development and support of code related to the interface between the user and the backend. Their primary responsibilities will be updating the UI, experimenting with UX enhancements, and creating animations to enable the user to better understand the simulation they have designed.

**Backend**

The team’s responsibility includes: improving the data access layer, updating “JSONify” and data reporting, and implementing custom object creation and manipulation. Both teams will work to prepare examples and demos for the finished product as well as full documentation.

**Targeted Platform and Constraints**

The BLUESIM web application offers significant flexibility. The primary requirement to access BLUESIM is an active Internet connection; while the product may be stored locally, a user must connect to the Internet to initially access the product. In terms of hardware, the application is limited by the user’s ability to interact with the HTML5 canvas element. Generally, this does mean that tablets and smartphones may have a difficult time clicking and dragging the objects. BLUESIM also uses several JavaScript libraries that limits the user’s choice of Internet browser, depending on the age of the browser. The application uses the D3 graphics library that is supported on almost all web browsers except for Internet Explorer 8 and below. The Raphael library also supports all modern Internet browsers (Firefox 3.0+, Safari 3.0+, Chrome 5.0+, Opera 9.5+ and Internet Explorer 6.0+).

## 3.1 Requirements

Requirements for BLUESIM are based on vetting ideas and a prototype to the client. Using feedback from these sessions, we determined that the following are essential requirements of the product.

1. UDOs can be created dynamically and added to pre-loaded or newly created simulations.
   1. *Each entity created will populate an on-screen container filled with user-defined entities.*
   2. *Fields and functions for UDOs are user-specified or set to a default value*
2. Allow user to load driving examples that sample the functionality provided in the simulation.
   1. *Users can modify existing general systems to better fit their simulation requirements.   These examples will include a bank queue, student pathways and a few other examples that demonstrate the capabilities of the simulator.  They are accessible via a drop-down menu.*
   2. *Loading an example will result in auto-populating the canvas with pre-defined entities.*
3. User can drag and drop entities into the simulation to include Queues, Sources, Splitters and Sinks
   1. *Users drag and drop entities from a pre-populated container to place in the canvas and have the ability to assign values to entities to control simulation flow-paths*
   2. *Entities can be added, edited and removed from canvas by double-clicking on them. In addition, entities have the following parameters to modify*
      1. *Queue* 
         1. *Average service rate*
         2. *Max Queue length*
      2. *Splitter*
         1. *Fraction of traffic that flows through channel*
      3. *Source*
         1. *Distributions*
   3. *Flow paths are created by connecting entities via a draggable arrow image originating from Splitters, Sources, UDOs and Queues.*
4. The user will be able to pull simulation models, as well as simulation data, from the software.
   1. *The user will be able to translate their created simulation into JSON format for easy referencing.  This can be added to the canvas via a load field.*
   2. *Data derived from running the simulation will also be accessible to user and stored via a text file.*
5. The software will have a graphical/animated representation of simulation results.
   1. *Entities on the canvas will graphically reflect statistical feedback computed by the simulator via animation.*
   2. *Data produced by the simulation will be displayed in tables as well as graphs for Queues and Sinks as follows.*
   3. *Queue* – *Mean time and Standard deviation (where applicable) for:*
      1. *Arrivals/Dropped*
      2. *Server Utilization*
      3. *Time Spent*
      4. *Queue Length*
      5. *Customers in System*
   4. *Sink* 
      1. *Departures*
      2. *Population*
      3. *Stay Duration*
6. The simulation tool is generic enough to handle a user of any experience type.
   1. *The software will have driving examples to help the user understand the use of the simulation.*
   2. *The software will also include simple instructions as to how to start using the simulation tool.*
   3. *Complex components are abstracted to the user, allowing for users with little or no experience with simulation tools to fully interact with the software.*
7. The software is open and accessible.
   1. *The software is built on a web-based platform using generic web languages that allow it to be run on any browser application.*
8. The user can change the run-time for simulations, specifying the time unit in hours, minutes or seconds.
   1. *The simulation has a few control options to include:*
      1. *Time running - the user can specify the time for which the simulation will run.*
      2. *Pause/Resume - a running simulation can be paused if the simulation run-time is set to a long enough time.  The software will then return data for the simulation up to the time paused.  User has option to resume running the simulation.*
      3. *Stop - The user can stop the simulation from running in the simulation run-time is set to a long enough time.*
9. The software will give options for creating a new canvas, exporting and loading simulations.
   1. *The simulator will have the following options:*
      1. *New - the simulation prompts the user to reset the screen or cancel otherwise.*
      2. *Load - triggers a pop-up window with a textbox in which the user can insert JSON files that are created when a user saves a simulation.*
      3. *Export - creates a JSON file representation of the current simulation in the canvas.*

## 

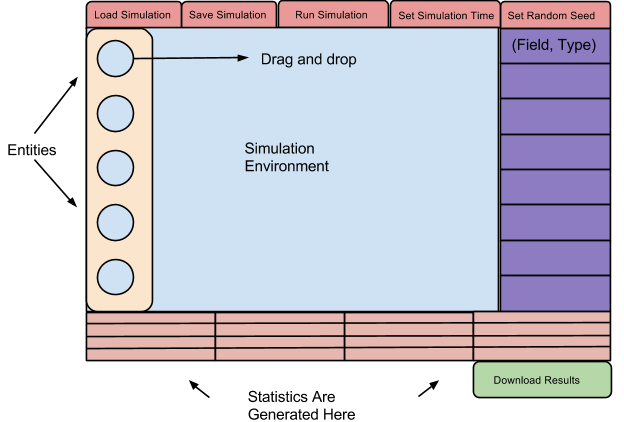
## 3.3 Workflow

The goal of this product is to expose the discrete event simulator library, sim.js, in a practical way. The user can access BLUESIM in one of any of the supported browsers (see Targeted Platforms). The website will be hosted at vernacular.cs.umd.edu. The product can be launched on a local browser and will run normally as long as the user’s machine is connected to the Internet. A user familiar with hosting a server could also host our work on their own machine like any other webpage in this way.

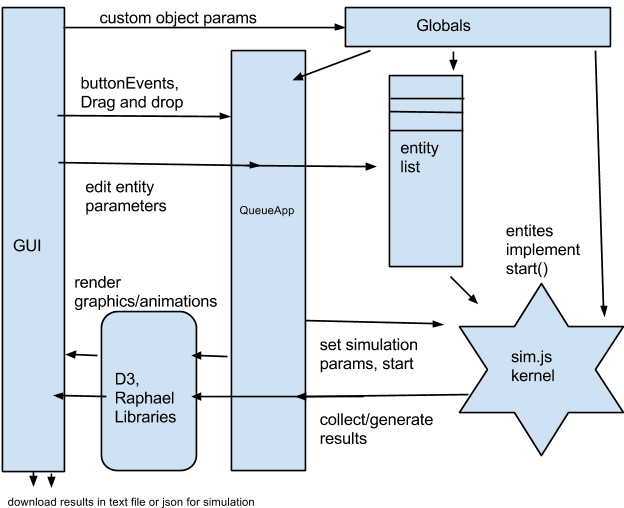
On the website, the user will be presented with a link to the documentation explaining how to use the tool as well as a list of pre-loaded simulations that provide users with working examples of simulations. A large HTML5 canvas houses the simulation workspace. Creation of a simulation involves dragging and dropping a set of pre-defined entities: a generator, a function block, a queue, a branch, and a sink. This subset of sim.js entities can construct a large number of different simulations that are perfect for the average user’s requirements. Furthermore, the user will have the ability to define custom objects in the simulation. The user may declare the fields and types of the data and the simulation will adapt accordingly and generate statistics from the fields.

**Code Architecture**

The HTML5 canvas is the first layer of the whole package where onclick events are triggered. All events call functions in a user-configurable object called QueueApp that allows the user to change the state of the sim.js simulation. QueueApp encapsulates the entire simulation and gives the user the ability to add entities, execute, and gather the statistics from the simulation. The user starts the simulation by pressing the main “Simulate” button. The simulation is constructed by dragging the corresponding icons from a side list onto the simulation grid. Each drag and drop causes the QueueApp to instantiate an entity with modifiable fields. To use the sim.js library, every participating node implements a start() function. Entity objects include: a generator, function block, queue, branch, and sink. Each entity has a set of output entities (except for the sink entity). Entities communicate by passing anonymous objects to one another via sim.js’s message system. These anonymous objects are custom objects that the user can define which give a greater level of complexity to our simulations. Only one object class is ever chosen to be instantiated. The user describes the names of the fields and their types. In the generator entities, the user can specify the frequency of their instantiation and the values that the object can obtain. These objects are loaded into the sim.js simulation object that abstracts the actual simulation. QueueApp posts statistics gathered from the simulation in a <div> element on the website. QueueApp allows the user to download this data and the corresponding JSON representation.



###### Fig. 1: UI Design



###### Fig. 2 BLUESIM’s System Architecture

# 

# 4 Delivery Manifest

## 4.1 Deliverables

* URL to host server where the web-application resides.
* User documentation page, reference guide in the application, installation guide (Detailed in section 5).
* User documentation: Front-end interaction guide for users
* Reference Guide: Glossy of entities
* Installation Guide: Installing BLUESIM locally
* Source code of the application as a signed zip file
* Includes client-side (frontend and backend), graphical assets

The above outline includes the software components that we plan to deliver upon completion of the product. The URL address will link back to the SVN server and house the web-application and will be runnable on any popular browser. Though the BLUESIM application is hosted online, it can be run locally without a server as long as the user has an Internet connection and has downloaded the source files. The user documentation will be embedded in the software and will be viewable upon accessing the software. The reference guide will be packaged with the initial product submission along with an installation guide. These are further explained in Section 5. The source code will also be uploaded onto the Bluesim SVN repository and submitted along with the product. Generated simulations are saved as text in the form of a JSON string that must be manually saved by the user for future use. Alternatively, a “Save” button will be present to save the JSON string as a local file. Output from the simulations is available as both a downloadable text file and a JSON file that also must be saved client-side.

## 4.3 Verification

The verification of our product will be executed through working example simulations that will be provided in our extended documentation. These listed simulations will provide example cases that highlight the features and entities of our tool. Each simulation scenario will have an expected input and output that will be then run through the BLUESIM simulator to verify that the simulator’s output matches the anticipated output. The customer will likewise be able to verify that BLUESIM meets their requirements by matching their requested features (as featured in the “Requirements” section of the document) to the realized product that can be tested by the customer with a wide breadth of simulations. Specifically, the user will create their input, view the animations of the simulation running, and review the resulting statistics from the simulation. Additionally, to verify that our simulation handles JSON inputs correctly, we will implement the Bank Teller simulation through JSON input only. The resulting simulation is expected to match the example Bank Teller simulation. Finally, to verify that our product works on multiple devices, we will have two virtual machines download the source files and run the simulation locally.

Example simulations:

* Bank Teller
  + - * Task: Simulation to compare a multi-queue vs. single-queue system for customers being served by bank tellers. Details are the same as the Grocery Store simulation explained in section 4.4.
      * Expected Results: A single-queue system has a shorter run time than a multi-queue system.
* College Pathways
  + - * Task: Students are generated with two parameters: SAT and loopsUntilFail. SAT is randomly set from 1 to 10, while loopsUntilFail is initialized as 12. From there, students are put into either community college (SAT <5) or University (SAT >= 5). Two sinks are set: “Employed” and “Unemployed”. Each student passes through eight “loops” of either community college or University. If a student completes 8 loops with loopsUntilFail > 0, then they are “Employed.” Otherwise, they are “Unemployed.” The probability of success for a loop at community college is (2\*SAT / 10). If after 4 loops, a student at a community college has at loopsUntilFail >= 8, then they are transferred to University. The probability of success for a loop at University is (SAT/10).
      * Expected Results: Students who start their loops at University have a higher employment rate, on average, than students who transfer into University.
* Predator-Prey
  + - * Task: This simulation models the ratio of predators to prey within an ecosystem. As the number of prey grows, the number of predators grows as well. When the predator numbers grow, the prey population falls; this causes the predator population to subsequently fall. A lower predator population causes the prey population to rise, which repeats the cycle. This simulation will run to gather long-term data about the distribution of animal population over time.
      * Expected Results: Over an exponential time, the system will reach equilibrium if the system is stable as per the Volterra Equation.

## 4.4 Validation

Once we have internally verified our product, we will sample various students at the University of Maryland to test our software. We will divide our test groups into three categories: students who are familiar with simulation software, students with programming background but without a simulation background, and students with no programming background nor simulation background. We are targeting these groups in order to validate different areas of our project. By asking students with simulation experience, we can confirm that we have indeed created a simulation tool. By asking students with only programming experience, we will validate that our project is usable by technical user. By asking users with no experience, we can test the usability of our tool. We plan to ask at least 5 students in each category to go through our validation exercise so that we have a total of 15 tests. This test will validate the installation process on new machines. Each group will be given the same task: to create 2 given simulations, and save, export, and load 1 simulation. The first simulation will be to simulate lines at a grocery store and the second will ask the participants to simulate integer distributions. In both simulations, we will time the user with a limit of fifteen minutes to conduct the simulation. The simulations listed below contain “Example Steps” that showcase how a user may solve the problem. These “Example Steps” are not given to the user; the user only has access to the task description, BLUESIM, example cases, and software documentation. They will be allowed to ask as many questions as needed during the test.

**Grocery Store Simulation**

* **Task**
  + Users will be asked to create two simulations side-by-side. These simulations represent customers queueing up to one of two cashiers at a grocery store. The first simulation is a multi-queue system, where customers are sent to one of two queues in an alternating fashion (i.e. first customer sent to first queue, second customer to second queue, third customer to first queue, fourth customer to second queue, etc.). The second simulation is a single-queue system where customers are put into a single queue and choose an open cashier using a first in, first out system. Users are suggested to look at existing example simulations for help.
  + **Parameters**
    - Customers are created by an exponential function with lambda = 1
    - Each customer has the following probability for the time required to check out:
      * Uniform distribution over [1,5] seconds
    - Simulation is set to run for 5 minutes
* **Example Steps**
  + 1 – User reviews the “Bank Teller” example simulation
  + 2 – User create 2 generators for customers
  + 3 – User update each generator with the probability check out time
  + 4 – User attach a splitter to the first generator
  + 5 – User specifies the splitter to alternate between customers 50/50
  + 6 – User adds a queue to each end of the splitter
  + 7 – User updates the queues from step (6) to have 1 “cashier”
  + 8 – User points each queue from step (7) to a new sink
  + 9 – User attaches a queue to the second generator
  + 10 – User points the queue from step (8) to a new sink
  + 11 – User updates the queue from step (8) to have 2 “cashiers”
  + 12 – User runs the simulation for 50 customers
  + 13 – User reviews collected data and determines which queue type processes customers the fastest
* **Validation**
  + Upon reviewing the results, the user should determine that a single queue system works more effectively than a multi-queue system.
* **Goal** 
  + The goal of this simulation is to verify that the example simulations are clear and concise for new users. Users should be able to replicate the bank teller problem as the grocery store problem is a basic re-wording of the bank teller problem.

**Number Distribution Simulation**

* **Task**
  + Users will be asked to generate and examine the distribution of positive and negative numbers from a uniform probability distribution between [-2, 1]. The simulation’s output data is expected to show that the distribution of positive numbers converges to 1/2.
* **Definitions**
  + Uniform Probability Distribution: Distribution of numbers that have constant probability
* **Example Steps**
  + 1 – Users place an integer field into the simulation’s source as a generator.
  + 2 – Users specify the integer range to be from -2 to 1.
  + 3 – Users create a splitter into two sinks, attach it to the generator.
  + 4 – Users specify the splitter to split based on an integer value >=0, <0.
* **Validation**
  + If the distribution of integers generated by the simulation results shows that 1/2 of the generated numbers were positive, then the user has completed the task accurately.
* **Goal**
  + The goal of this simulation is to show that, even without example simulations to work off of, users can create simple simulations in a short amount of time.

**Save, Export, and Load**

* **Task**
  + Users will be asked to save the Number Distribution Simulation previously completed. They will then open a new instance of BLUESIM and import their Number Distribution Simulation as a new simulation. Users will then run their simulation again to verify the integrity of the save/load feature.
* **Example Steps**
  + 1 – Users click “Save” on their Number Distribution Simulation and copy the JSON text to their clipboard
  + 2 – Users refresh the webpage
  + 3 – Users click “Load” and paste the JSON code into the corresponding field.
  + 4 – Users run the simulation and verify similar output data.
* **Validation**
  + If the new output data matches the older, then we can consider this test a success.
* **Goal**
  + The goal of this simulation is to show that users may save, export, and load their simulations accurately and effectively.

**After the Tests are Concluded**

Once we are finished with the tests, we will evaluate the number of students that completed the simulation and the amount of time it took to complete them each. We will consider their backgrounds in our review process, but we expect most of the simulations to be correctly generated in the allotted time. If at least 75% of the students completed their task successfully within the allotted time, then we can consider our product successful. Users will also be asked to fill out a survey hosted at the following URL: <https://www.surveymonkey.com/s/KZ3NC3H>. We will consider the user experience aspect of BLUESIM successful if each question within the survey receives a “3” or better score as an average amongst all students tested. The scoring methodology is outlined below.

**Survey Details**

Users are asked to answer a quick survey after the completion of the validation tests. After reviewing basic user information, users are asked to answer four questions related to the BLUESIM experience. Responses are based on a 5-point scale, ranging from Strongly Disagree (1) to Strongly Agree (5). Finally, users are asked to answer two final questions related to the complexity of using the BLUESIM software. Responses in this section are also based on a 5-point scale.

**Survey: BLUESIM experience**

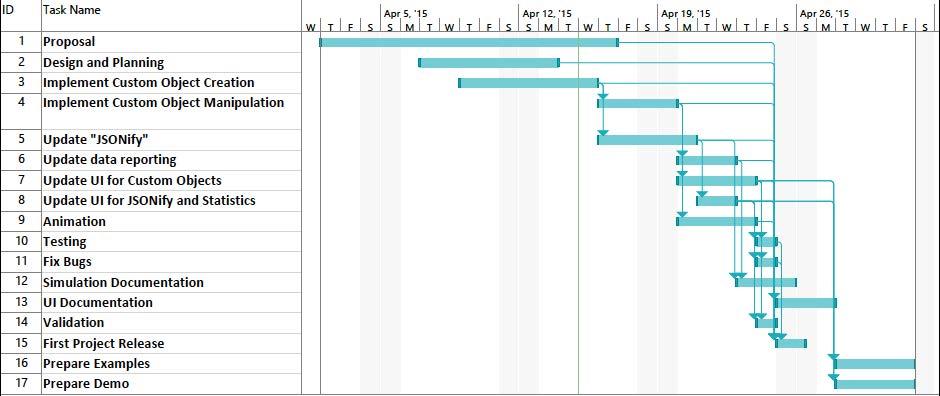
* The example simulations provided were clear and understandable (1 – 5 Scale)
  + This question examines whether the provided example simulations were intuitive for new users to understand. The provided examples are designed to introduce users to basic BLUESIM simulations. If the majority of responses are (3) or higher, we can determine that the simulations were easy for new users to understand.
* The BLUESIM entities (generator, splitter, etc.) was intuitive to use in the simulation (1 -5 Scale)
  + If the majority of the users respond with a (3) or higher, we can determine that our drag and drop interface was intuitive for new users. An average score of (2) or lower indicates that creating and linking entities within BLUESIM is difficult for new users and will require an improvement of the interface.
* BLUESIM output is easily accessible (1-5 Scale)
  + Since BLUESIM data is outputted as both raw text and a JSON file, we want to test whether users can access the output files effectively. New users are not expected to interpret two JSON files within the 15 minute time limit, so this test verifies that users may access the data for further examination.
* Creating a new canvas, exporting, and loading simulations was intuitive and simple (1-5 Scale)
  + This tests determines whether the saving and loading of simulations was intuitive for new users. This question reviews the loading process (where a user saves a simulation in one canvas, creates a new canvas, and imports their old simulation) and verifies that novice users may save their simulations for future use.
* If I was stuck or confused, the documentation provided answered my questions effectively (1-5 Scale)
  + The user documentation provided for BLUESIM is intended to help introduce new users to the simulation software. While we conduct a basic introduction at the beginning of our validation tests, we expect that some users will still have questions during the completion of their tests. If their questions are not answered by the provided documentation, we will know that our documentation is not sufficiently comprehensive and requires updating.

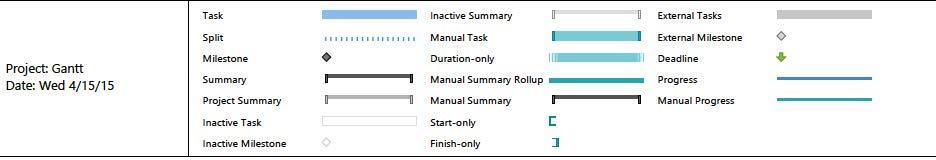
**Survey: Complexity Rating**

* Did you feel that the time spent building the simulations took too long, too little, or just right? (1, 3, 5 Scale)
  + In this question, points are distributed as follow: Too Long (1), Too Little (5), Just Right (3). Users who report that our simulation software emulated their simulations in “too little” time indicate that the ease of use within our simulation exceeded their expectations. If the average score is below (3), then we can determine that our simulation interface is overly complex for an average user’s needs and requires improvement.
* Do you feel that BLUESIM would satisfy your potential simulation requirements? (1, 3, 5 Scale)
  + In this question, points are distributed as follow: No (1), Yes (5), Neutral (3). We do not expect all users to have the same simulation requirements. However, as our software is intended for an average user, we expect that our three targeted testing groups will have similar potential simulation requirements. If the majority of users answer “No,” then we will have found that our simulation does not satisfy our targeted audience’s requirements and requires further improvement.

## 4.5 Targeted Dates

Our proposal includes an anticipated schedule for the development of this simulator. At each “end date”, as detailed within the “Targeted Dates” section of our proposal, teams will meet to review the completion of their tasks in accordance to the requirements set at the beginning of the development cycle. Furthermore, all teams meet twice a week outside of class to review the current development progress and adherence to the development schedule.





###### Fig. 3 Project Schedule

## 4.6 Cost Estimate

The cost estimate table below was generated based on the project schedule. The total amount of hours invested in the project represents the number of work hours as a team.

|  |  |
| --- | --- |
| **Task** | **Estimated Time (hr)** |
| Design and Planning | 80 |
| Proposal | 120 |
| Implement custom object creation | 110 |
| Update “JSONify” | 85 |
| Update data reporting | 75 |
| Implement custom object manipulation | 72 |
| Update UI for custom objects | 67 |
| Update UI for JSONify and statistics | 55 |
| Animation | 90 |
| UI Documentation | 27 |
| Simulation and documentation | 35 |
| Testing | 45 |
| Fix Bugs | 40 |
| Validation | 35 |
| First project release | 30 |
| Prepare examples | 20 |
| Prepare demo | 10 |
| **Totals** | 996 |
| **Schedule (workdays)** | 30 |

## 

## 4.7 Risks

In efforts to maximize BLUESIM’s ease of use, we have minimized the space of simulations that can be realized with our tool. We have made multiple design decisions, such as hiding certain features in the sim.js library, limiting the diversity of data in the simulation, and restricting mutability of the data. Removing features such as the sim.js store and timer entities reduces the complexity of the simulations without terribly minimizing our use cases. A simpler simulation range creates a simpler user experience that allows us to create optimal documentation for a gradual learning curve approach to the simulation tool. The risk to our product is improper balance between the minimization of the learning curve and the simulation potential of our final deliverable. Advanced simulation tools are already available on the market, and covering a maximal amount of simulation possibilities is not the goal of the BLUESIM product. Our niche is to write a tool for the layman; however, no one is interested in a tool that only makes trivial simulations. In order to minimize our risk, we have scheduled demos and reviews with our primary customer, who can provide input as to their expected level of simplicity towards the product. With persistent feedback from our customer, we anticipate that the risk of improper balance between user-friendly design and simulation capabilities will be minimal.

# 5 Documentation

A reference guide similar to the sim.js guide will be provided that explains each of the entities in our simulation, their functions, and their uses within the simulation. The reference guide serves as both a glossary and a guidebook by explaining how each entity may interact with other entities with the simulation. In addition, a second guide will be provided detailing how to navigate our builder website so that users will know how to manipulate objects within the site. It will also feature a “Getting Started” section that provides users with a basic tutorial towards creating their first simulation. Furthermore, we will provide links to demo simulations that will highlight different features and functions of our tool. The user will be able to load these given examples and manipulate them on the spot. This will be extremely helpful for our audience as they can tailor our preset demos to suit their needs with drastically lower effort than if they were to build from scratch. The product will also be packaged with a simple installation guide. Since BLUESIM is developed as a web application, a user will only need the source files (downloadable from the SVN and/or host website), a compatible browser, and an Internet connection in order to launch BLUESIM locally. The source files must be stored in the same location in order for BLUESIM to launch via the main HTML file. Lastly, the source code itself will be stored on an online SVN, and engineering logs will be kept to verify the history of development of our product by means of documented recorded information.

# 6 Intellectual Property Statement

BLUESIM is free, open source, and licensed under LPGL.

**Libraries in use and their licenses**

* Raphael
  + MIT License
  + http://raphaeljs.com/license.html
* D3
  + BSD License
  + http://opensource.org/licenses/BSD-3-Clause
* Sim.js:
  + LGPL
* Sim.js demo:
  + Declared LGPL after contacting the author about the code’s license

# 7 Summary

A vast variety of simulation tools currently exist on the market. While these tools are available, they are either severely limited in simulation capabilities or require a steep learning curve to utilize. No simulation tool available on the market offers a middle ground that allows users to quickly and effectively create custom simulations without a significant time investment. Our proposed product, BLUESIM, is a niche software solution that offers moderate simulation capabilities paired with a gentle learning curve. BLUESIM allows a user to create fully modifiable simulations, complete with custom input, design, and easily viewable outputs. Easily accessible both off and online, BLUESIM is an optimal solution to a clear and prevalent gap in the market.

# 8 Sources

* <http://d3js.org/>
* <http://raphaeljs.com/>
* <http://simjs.com/>
* [http://faculty.mu.edu.sa](http://faculty.mu.edu.sa/public/uploads/1422182346.2949%D9%83%D8%AA%D8%A7%D8%A8%20%D9%87%D9%86%D8%AF%D8%B3%D8%A9%20%D8%A7%D9%84%D8%A8%D8%B1%D9%85%D8%AC%D9%8A%D8%A7%D8%AA%20%D8%A7%D9%84%D8%B7%D8%A8%D8%B9%D8%A9%20%D8%A7%D9%84%D8%AA%D8%A7%D8%B3%D8%B9%D8%A9E%20%281%29.pdf) -Ian Sommerville’s Software Engineer book