

Final Report

Beach Simulation

Group 15:

Daniel Davis

Yang

Steven Fisher

Gegi

Jose Villamor-Delgado

Tanzina Farzana

Table Of Contents

Summary of Changes	
1. Customer Statement of Requirements	4
a. Problem Statement	6
2. Glossary	8
3. System Requirements	9
a. Enumerated Functional Requirements	9
b. Non-Functional Requirements	10
c. On Screen Appearance Requirements	12
4. Functional Requirements Specification	14
a. Stakeholders, Actors, and Goals	14
b. Actors and Goals	14
c. Use Cases	15
i. Use Cases Casual Description:	15
ii. Use Case Diagram:	16
iii. Traceability Matrix	16
iv. Fully-Dressed Description:	17
d. System State Diagram	19
5. Effort Estimation Using Use Case Points	22
6. Domain Analysis	23
a. Domain Model	26
b. System Operation Contacts	28
c. Mathematical Model	29
7. Interaction Diagrams	31
a. UC-1: Simulate	31
b. UC-2: Registration	32
c. UC-3: Login	33
d. UC-4: Modify Simulation Settings	34
e. UC-5a: Export	35
f. UC-5b: Export Alternate Scenario	36
8. Class Diagram and Interface Specification	37
a. Class Diagram	37
b. Data Types and Operation Signatures	37
c. Traceability Matrix	38
d. OCL Contract Specification	39
9. System Architecture and System Design	42
a. Architectural Styles	42

b. Identifying Subsystems	42
c. Mapping Subsystems to Hardware	43
d. Persistent Data Storage	43
e. Network Protocol	44
f. Global Control Flow	44
g. Hardware Requirements	44
10. Algorithms and Data Structures	45
a. Algorithms	45
b. Data Structures	45
11. User Interface Design and Implementation	46
a. Preliminary Design	46
12. Design of Tests	51
13. Project Management and Plan of Work	52
a. Merging and Contributions from Individual Team Members	52
b. Project Coordination and Progress Report	52
c. History of Work	53
d. Future Work	53
e. Breakdown of Responsibilities	54
14. References	56

Summary of Changes:

1. Problem Statement

After reviewing our project we realized some of the terms in the problem statement weren't relevant to the project at hand. There were also more requirements that were vaguely expressed in the problem statement. We expanded upon some requirements to improve the clarity of the problem at hand. Otherwise, there were some general changes for consistency and better clarification.

3. System of Requirements

Altered to reflect changes to project and problem statement. Switch of platform and problem statement alteration changed some of the requirements and were changed or

5. Effort Estimation Using Use Case Point

Updated to reflect the new totals with a new UI implementation.

6. Domain Analysis

c. Mathematical Model

Updated the mathematical model to represent the current model that the application is using. Added better descriptions and more in depth examples.

8. Class Diagrams and Interface Specification Not finished for this handin**

a. Class Diagram

Edited class diagram application to Log in, Account to Create Account and deleted Account setting and Export.

Edited to reflect the merger of the application with the controller in our application.

b. Data Types and Operation Signature

Edited the operation types to reflect the change from web application to desktop application. The controller has been merged with the application and each class is now isolated within their own individual structures.

c. Traceability Matrix

Edited the traceability matrix to reflect the merger of the application and the controller.

d. OCL contracts

Added OCL contracts.

9. System Architecture and System Design

a. Architectural Style

Now a mixture of client/server model, event driven model and implemented using Object Oriented design. The client server model part is the application check for permission to load via the account database online.

b. Subsystems

Updated the subsystems to reflect the changed design of the application.

c. Subsystems and Hardware

The system now needs a windows machine that has .NET 3.5 installed in order to run in addition to an internet connection.

d. Persistent Data Storage

Persistent data storage now exists in the form of one simulation at a time, what this means is that each new simulation overwrites the old simulation file. However, the server does contain a database storing user settings perpetually as well as user data.

e. Network Protocol

Client is no longer on the internet so the server requests will instead be serviced in the form of Visual Basic Requests instead of PHP requests

f. Global Flow Control

Added event driven description for simulation and export cases.

Changed time dependant events to match new plan of work.

g. Hardware Requirements

Added recommended 1024*768 to fit application window size.

Added operating system requirement of windows with .NET 3.5 installed due to changes in the application design.

Added minimum hard drive space now that application must be stored on the local drive.

10. Algorithms and Data Structures

a. Algorithms

Edited to reflect the current algorithm implemented in the program i.e. curves reflecting agent preferences and the adjustments that agents will make pending the results of the simulations.

b. Data Structures

SQL database and local array storage.

1. Customer Statement of Requirements

Problem Statement

As a coastal business owner, running a profiting beachfront shop is already difficult. Running our business includes inventory, staffing, and many other factors that all depend upon the general population that is passing through our store. Like all businesses, the factors of profit are sales and costs. In order to keep costs down, we must be able to predict demand and adjust accordingly. If too many staff members are on hand, or too many supplies are purchased and not used/sold, it cuts into profit. It is an important skill to be able to predict the amount of customers and their spending habits. However, unlike a mall or mainland store, coastal markets are difficult to predict because tourism relies heavily on various factors including weather, timing (dates and seasons), water temperature, and many other factors.

Because this is an integral part of our business, we need a software that will simulate a large amount of these factors. By simulating the population, our business will be able to better predict the weeks' customer flow. This can help us cut unnecessary costs like overstaffing and overstocking.

One variable we have seen that greatly affects the tourism population, is the dates. On holidays, like the Fourth of July, there is a considerably greater attendance than a non-holiday day. Also, the day of the week changes as well, weekends are definitely more populated than a regular week day. Finally, season, May time starts to have a more populated beach, however August is generally fuller than May. So the date should be considered when modeling the population.

Weather is an important factor in a customer's decision to visit the beach. If it is raining, we have seen that this negatively impacts our stores amount of customers. This makes sense, since most tourists don't want to walk the boardwalk or be on the beach in rainy weather. Therefore, this simulation should take into account raining and cloud cover to help the accuracy of the prediction.

In addition to weather, customers are also affected by their past experiences. A beachgoer may not want to visit the beach if it was too crowded. The beach being crowded makes it un-enjoyable. On the contrary, if one hasn't been going to the beach, they might "settle" on not as favorable conditions because they still want to go. This program should simulate the actions of others; if a beachgoer predicts that the beach is going to be too crowded, he/she may choose to stay home. On the opposite side of the spectrum, a beach can also be deserted, displeasing others (i.e people-watchers). The simulation should take both these possible outcomes into account. Because for various simulated people, their experiences affect their decision each time, the simulation should actually simulate multiple times. Therefore, when you simulate the settings, it is like simulating one thousand days with the same settings. This will then eventually end at a equilibrium of some agents knowing not to go, while others will. This will make the results a lot more accurate.

The simulation should also take into account the demographics of beachgoers. The population can be split into both gender and age groups. Each demographic has subtle preferences that affect their experience. Split into age groups, children may be more affected by water temperature because they plan to swim more than adults. A gender may factor their preferred gender's population into their decision. The demographics for the simulation should be able to be set individually. Meaning we should be able to apply the simulation's demographic settings to see how it affects the population curve. For example, if we are looking for a particular target audience of young adults, we can create a greater percentage of that age group. Then the resulting curve will reflect the changes for those new settings. For business purposes it is important to know what age groups affect the customer flow. When marketing or releasing new products, having some analytics will help our business efforts achieve a successful launch. Therefore, this simulation should incorporate these settings.

By inputting a location, the simulation should automatically update with that particular location's corresponding population data. The program should also have default and saved settings. When no settings have been saved the program will default the location to the current zip code and default the day to current day. If settings have been saved, this will default the setting's location to the setting, date will stay defaulted to current day. This will let us simulate multiple business locations, and in turn decrease the time it takes to simulate the population. Also allowing us to incorporate population statistics and information without having to research and input them ourselves.

When simulating the population, there should be a way to save any settings previously set. Most simulations will be run with the same basic settings like location and population statistics or demographic settings. There should be a way to save our custom settings that will then load as default. By allowing custom settings to be saved, it will reduce the amount of time to run a simulation straight from start-up.

In the simulation, this program will take into account all the factors discussed above. It will then result in a graph representing the approximate population that will come to the beach. This will then help us predict our clientele's statistical information and enhance our business practices accordingly.

2. Glossary

Population: Amount of people residing in a certain area. Can be based upon, city, county, state, etc. Also can be used in explaining an amount of people.

Agents: A simulated person that makes a decision based on their personal strategy, and contributes to the simulated population that the results are based upon.

Win: Each agent takes into account all the variables, and population results and determines if they would have an enjoyable time at the beach.

Strategy: Each agent will have set criteria on how they will base their decision. They can then alter their strategy if they aren't "winning" with their current strategy.

Self-Optimization: The process where agents will vary their strategy choices dependent on if they win.

Simulator: Running thousands of virtual days with the same settings in the simulator to end with an approximate answer of how many people would go to the beach. Resulting in an accurate representation of the projected population for the day.

Business User: The user who will be using the user interface. This is the main customer of our product. They will be able to simulate with variable control to gain information and data.

Variables: Different settings that are able to change to directly affect the population simulation. In this case, air temperature, water temperature, humidity, chance of precipitation and location are all variable in our program.

Simulated Population Percentage: This is the result of the simulation. After all the agents have repeatedly made a decision, there will be a stabilized average of what percent of the population would go according to the set variables.

Population to Agent Ratio: The number of people one agent represents. If a population is 500,000 people, instead of simulating 500,000 agents, we can define one agent to every 1,000 people. Resulting in only 500 agents.

3. System Requirements

3. a. Enumerated Functional Requirements

Identifier	Priority Weight (Low 1- 5 High)	Requirement Description
REQ-1	5	Agents decide if they go to the beach or not. The majority loses
REQ-2	5	Each agent has a personal strategy.
REQ-3	5	Agents with winning strategies repeat the winning strategies. Agents with losing strategies may alter them.
REQ-4	5	The more recent an Agents win/loss is, the more likely the agent will consider it in its strategy.
REQ-5	5	The system initial conditions have a default setting.
REQ-6	3	The system's default is the current day, unless previously set.
REQ-7	3	The simulation should take into account rain and cloud cover.
REQ-8	3	Agent action depends heavily on the weather conditions of the day.
REQ-9	5	The system updates graphs when pressing simulate, displaying changed variables.
REQ-10	4	Gender and Age Groups should have settings that are independent of each other.
REQ-11	4	The system runs the simulation multiple times so a good data set can be made.
REQ-12	1	Our result can be used for business logic functions.
REQ-13	1	Actual area demographics are used to give a more accurate prediction.
REQ-14	2	Agent strategies should account for friend, family, and neighbor agents strategies.
REQ-15	4	Agents should factor in weather and water temperature into their strategies.
REQ-16	2	Agents should account for what day of the week it is.
REQ-17	2	An agent's strategy depends on personal agent factors, such as age.
REQ-18	2	Agents should favor dates that are more convenient for their personal factors (such as various holidays, spring break for students, etc.)
REQ-I	5	The system will allow the user to run the software on different computers
REQ-II	4	The system shall allow the user to negotiate the software parameters
REQ-III	3	The system shall allow users to register with unique identifiers

REQ-IV	2	The system shall allow users to export simulation results into sep files.
--------	---	---

REQ-(1-11) Defines what the consumer believes are the most important variables that need to be included in the application to run the simulation based off of.

REQ-(13-19) Defines the expected AI interaction within that region that the user requested.

REQ-I Is intended for better compatibility which will allow the user to access their subscription from anywhere.

REQ-II Is implemented so that if the user ever changes his preferences or the regions parameters have shifted, the user can adjust them as needed.

REQ-III This requirement is meant to allow separate users to keep their own individual settings, and allow the user to be more productive as a result.

REQ-IV This requirement is meant to let the user carry results as a portable file.

3. b. Non-Functional Requirements

Non-functional requirements are requirements that describe how the system/system environment should be, rather than what the system needs to be able to do. A method of categorizing those requirements is FURPS+. FURPS+ stands for Functionality, Usability, Reliability, Performance, and Supportability. The + in FURPS+ stands for additional requirements such as design constraints, implementation requirements, interface requirements, and physical requirements. Since we are focusing on the non-functional requirements, only the URPS+ will be taken into consideration.

Usability includes the ease of use, aesthetics, and documentation. For usability, we will focus on implementing the application on a web browser through the use of javascript. The purpose of implementing it online is to allow users to access their saved simulations from anywhere. In terms of aesthetics, the interface should be pleasing and simple allowing for ease of use. Documentation is a major part of usability as it gives the user the information necessary to navigate the application.

Reliability is defined by the system's uptime and ability to recover from errors. In order to have a high level of reliability, the system should address as many errors as possible that might occur. In addition, should any errors occur, the system should keep a log of such errors and fail gracefully and inform the user of the error.

Performance is relative to how fast the application runs. If the application takes too long to simulate, user demand for the application will drop. The application must have a low response time, low resource time, and be fast in performing calculations. To test performance, while programming the application, time counters should be implemented for all modules in order to find areas of delay.

Supportability includes compatibility which should be addressed to allow users to connect from many different browsers. Modularity to allow us, the programmer to make a system that has easy maintainability, and adaptability, as well as, making the program serviceable. On top of those requirements, the application should be configurable and allow the programmer to adapt the application to the needs of the user.

On top of the main requirements the implementation of the system should have readable code, and set limits on operation to prevent overflow. In addition, the system should be formatted so that all calculations/variables use the same system of measurement.

Identifier	Priority Weight (Low 1- 5 High)	Requirement Description
REQ-21	4	Help option should be available to aid the user (documentation).
REQ-22	5	The user should be able to access their account data from any location.
REQ-23	5	System should account for any possible errors and set up a system to report the errors and fail gracefully.
REQ-24	3	System should implement runtime counters in order to allow the developer to find areas of delay.
REQ-25	5	System should ensure that there are no memory leaks to prevent needless resource usage.
REQ-26	5	System should be modular in order to allow easy maintenance and adaptation.
REQ-27	2	System should be able to be configured on the admin end for specific users and their needs.
REQ-28	5	System should use good coding practices such as readable code.
REQ-29	5	System should use one standard set of measurements to ensure interoperability.
REQ-30	4	System should have default values based on the specific use.
REQ-31	4	The user should be able to navigate the application using a maximum of three clicks from the root location.
REQ-32	4	The interface should be user friendly, and allow the user to see without squinting.

These URPS+ requirements are meant to make navigation simpler for the user, and allow for the system manager to have easier access to altering the application without greatly impacting the users. The requirements should define a system with minimal downtime, and good customer satisfaction. The objective of the requirements is to make a system so pleasing to use, that there are no discouraging factors of interaction.

3. c. Onscreen Appearance Requirement

Identifier	Priority	
-------------------	-----------------	--

	Weight (Low 1-5 High)	Requirement Description
REQ-33	5	The user interface takes default values such as population size, weather, etc. based on the location given.
REQ-34	3	The user interface able to check the graph based on data input.
REQ-35	2	The user interface would be able to save data and graph, which th would get from the simulation.
REQ-36	1	Visual Demographics, the data we get from the location GPS.



4. Functional Requirements Specification

4. a. Stakeholders, Actors, and Goals

Stakeholders:

1. Beach business owner/investor
2. Township (Beach Owner)

The beach is a natural resource that can be used for both relaxation and profit. Local businesses count on the beach's popularity for providing customers. Local municipalities are concerned with the tax revenues and the upkeep of the beach itself.

Local businesses have incentive to maximize profit and minimize loss. This means maximizing not only the number of customers, but also the amount spent per customer. Customers are more likely to spend money if they are having a good time. Also, beachgoers will stay longer at the beach if they are having fun. If a visitor stays long at the beach, they are more likely to make food and other purchases. In turn, the business has incentive to keep prices low to attract customers to the general area. Local businesses will use the simulation to evaluate the effect of pricing schemes and marketing on the consumer demand. Businesses will also use the simulation to predict the need for extra staff and other purchases. Since beach use patterns are highly affected by weather patterns, businesses may be hurt by a "bad season". It is useful to be able to simulate custom weather patterns to get an estimate of how the business will perform in the best and worst case scenarios.

Simulating the beach-going population is useful for event planning and tax purposes. Extra lifeguards must be stationed when the beach is more crowded. This also means more supplies, equipment, and training. In addition, the township is responsible for keeping the beach clean with the use of beach combing equipment and sanitation workers. As such, the township needs taxes to fund these operations. Therefore, the township will use the simulation to gauge operation costs and tax revenue. Some municipalities charge for parking, and the simulation can be used to show how changes in pricing affect beach use patterns and the tax revenue needed to be directed towards upkeep. The township has incentive to increase tax revenue and support local businesses. In addition, the township has incentive to attract new businesses (and therefore, jobs).

4. b. Actors and Goals

Actor 1: User [UC-1][UC-2][UC-3][UC-4][UC-5]

[Roles]: Provides data to the system and memory to be interpreted

[Type]: Initiating

[Goals]:

-To register and login to the system, run the simulation, adjust simulation parameters, adjust simulation algorithms

Side: System [UC 1-5]

[Roles]: Provides data to the system and memory to be interpreted

[Type]: Participating

4. c. i. Use Cases Casual Description:

The summary use-cases are as follows:

UC-1: Simulate -- Allow the user to run the simulation using the default settings.

Derived from REQ I-IV. *Only available to registered users.

UC-2: Register -- Allows a user to fill out a registration form to obtain access to the software.

Derived from REQ III.

UC-3: Login -- Allows a user to login to access their saved settings, and simulations.

Derived from REQ III.

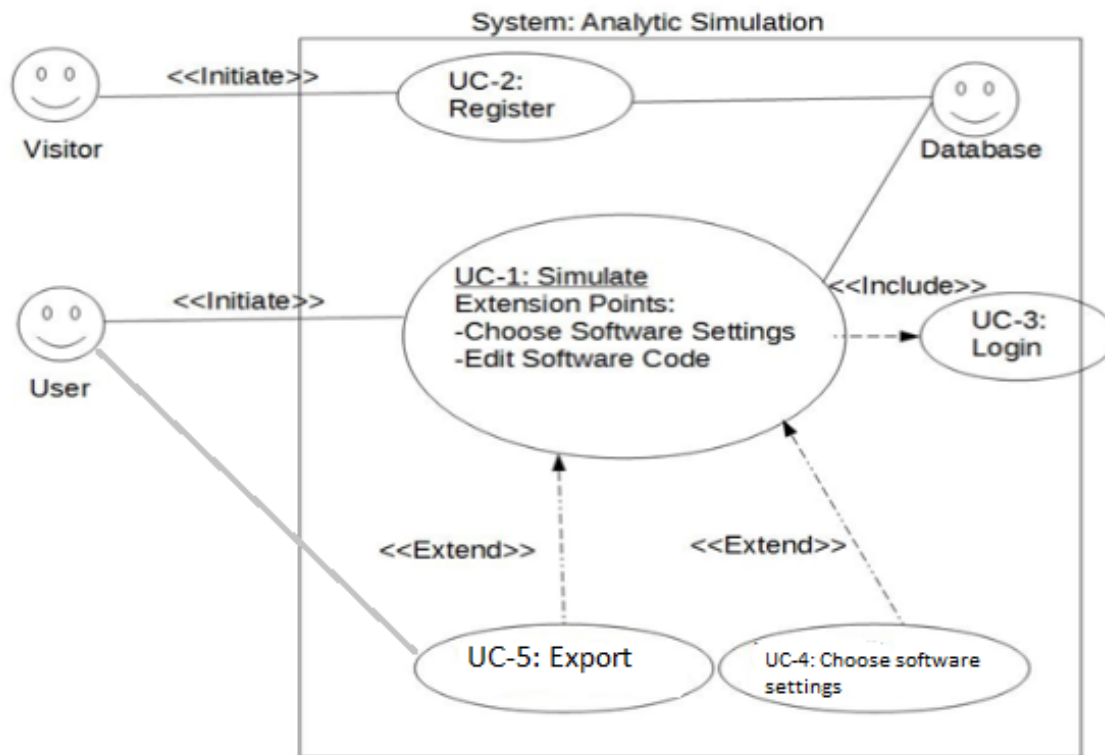
UC-4: ChooseSoftwareSettings -- Allows the user to edit the default settings to create new variations in parameters for the simulation to run.

Derived from REQ II.

UC-5: Export -- Allows the user to get the results of the simulation in a separate file, in the format of their choosing.

Derived from REQ IV.

4. c. ii. Use Case Diagram:



4. c. iii. Traceability Matrix

The traceability matrix shows the relationship between the use cases and requirements.

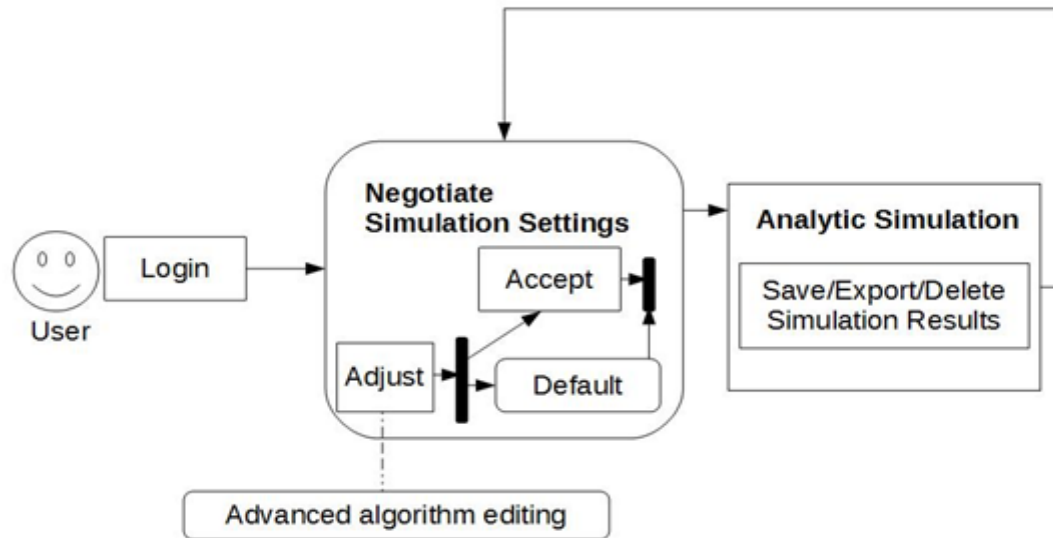
Req't	PW	UC1	UC2	UC3	UC4	UC5
REQ I	5	X				
REQ II	4	X			X	
REQ III	3	X	X	X		
REQ IV	2	X				X
Max PW		5	3	3	4	2
Total PW		14	3	3	4	2

Looking at the priority weights:

UC1>UC4>UC3,UC2>UC5

Using this weighting we will choose the use cases with the highest priority and implement them first.

4. c. iv. Fully-Dressed Description:



*Operational model for the analytical simulation software

Use Case UC-1:	Simulate
Related Requirements: REQ I-REQ IV	
Initiating Actor:	User
Actor's Goal:	To simulate the expected business
Participating Actors:	Database
Preconditions:	User is registered
Success End Condition:	If the simulation is completed, user is given a choice to save sim.
Failed End Condition:	If inputted parameters are outside of range, sim is not run.
Extension Points:	Choose Software Settings (UC-4) in step 2, Export (UC-5) in step 4
Flow of Events for Main Success Scenario:	

include::Login (UC-3)

- ← 1. **System** displays the default software parameters, as well as, an advanced option button
- 2. **User** accepts the default conditions
- ← 3. **System** simulates the results based upon the parameters and displays the results
- 4. **User** can then choose to save or export the simulation results
- If user chooses to simulate again based on default settings, steps 2 to 4 are repeated ---
- 4. a. **User** chooses to save/export
- ← 5. a. **System** saves the results/exports them
- Steps 2 → 5 are repeated until the user exits ---

Flow of Events for Alternate Scenario:

- 2. a. **User** chooses to modify default conditions, and must accept them
- ← 3. a. **System** checks modified parameters for legality and simulates if legal
- User is informed if values are not acceptable / Simulation is not run ---
- 4. a. **User** can then choose to save or export the simulation results

Use Case UC-4: Choose Simulation Settings

Related Requirements: REQ II

Initiating Actor: User

Actor's Goal: To adjust simulation settings

Participating Actors: Database

Preconditions: User must be logged in

Success End Condition: If parameters are acceptable, settings can be saved

Failed End Condition: If inputted parameters are outside of range, must be reentered

Extension Points:

Flow of Events for Main Success Scenario:

- ← 1. **System** displays the default software parameters, as well as, an advanced option button
- 2. **User** chooses the advanced option button
- ← 3. **System** displays all the implemented variables that may be edited
- 4. **User** can then choose to alter the variables to suit their own needs
- ← 5. **System** checks the variables to see if they are legal and informs the user if they are not
- If variables are not legal, user is returned to step 4 ---
- ← 6. **System** prompts user with option to save their preferences
- 7. **User** chooses to save
- ← 8. **System** stores preferences in the database

Flow of Events for Extensions:

Use Case UC-5: Export

Related Requirements: REQ IV, REQ I

Initiating Actor: User

Actor's Goal: To export simulation result as chosen file

Participating Actors: Database

Preconditions: User must be logged in

Success End Condition: If parameters are acceptable, a file will be given to user.

Failed End Condition: If inputted parameters are outside of range, must be reentered

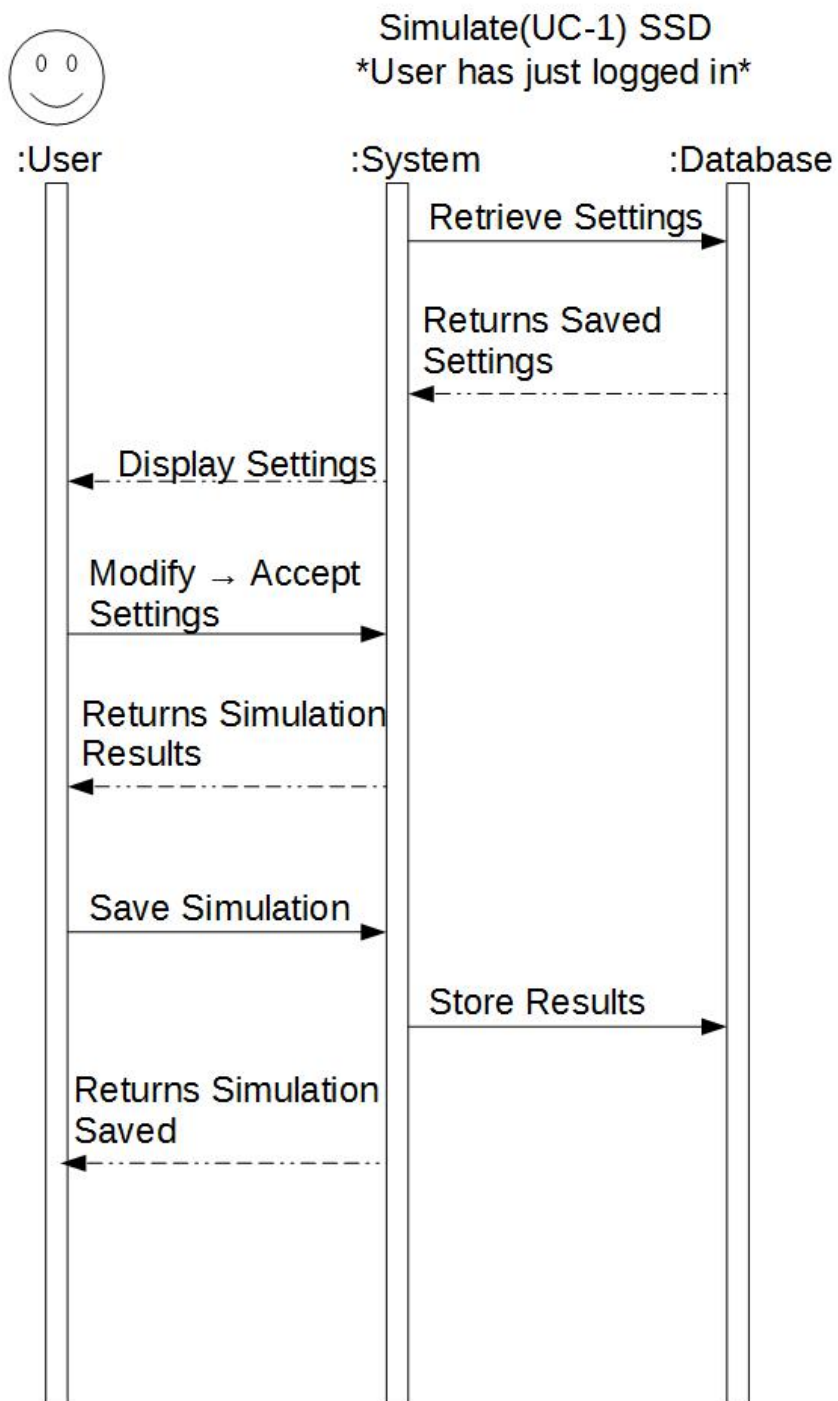
Extension Points:

Flow of Events for Main Success Scenario:

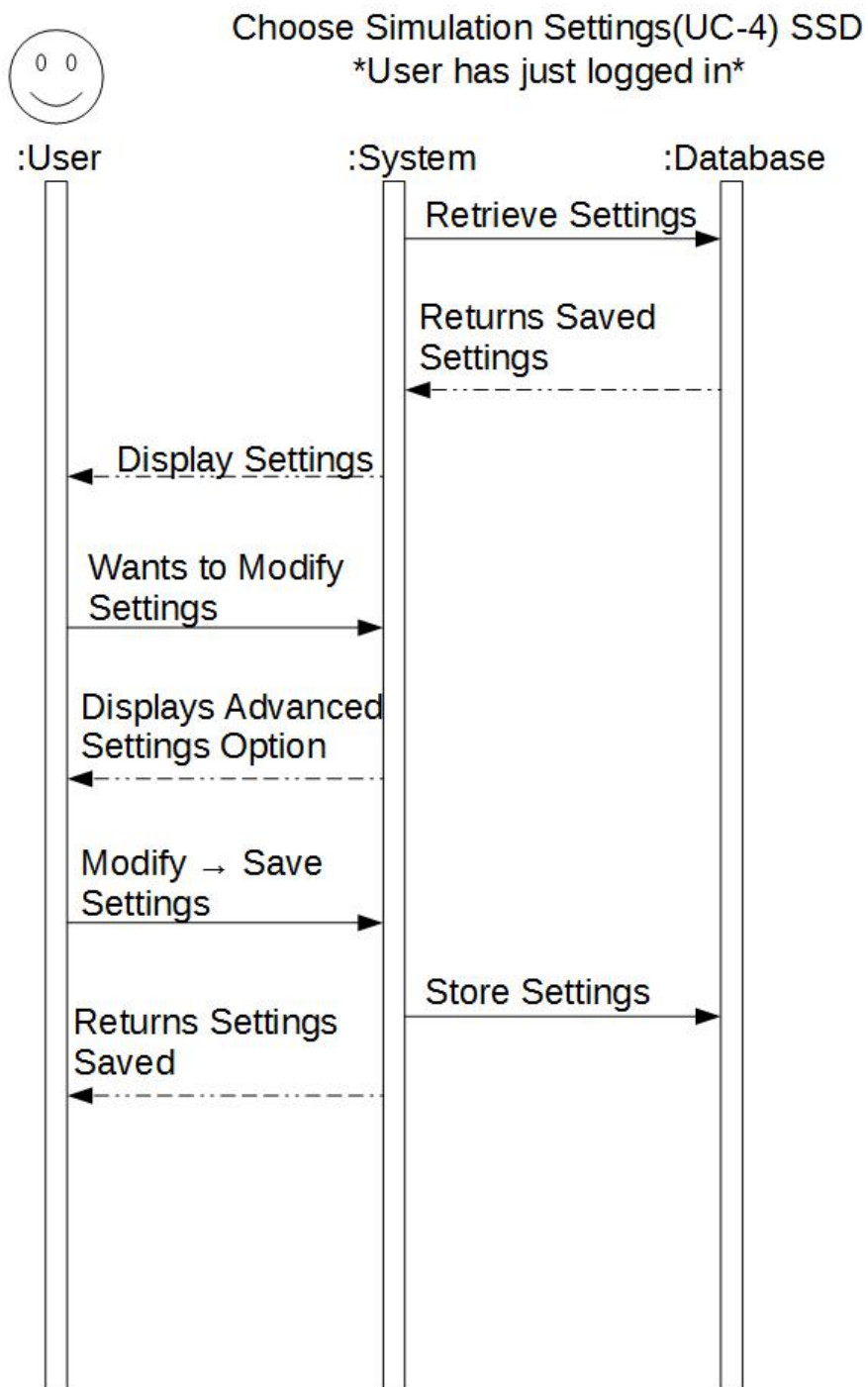
- 1. **User** Chooses the export button
- ← 2. **System** Displays the export options.
- 3. **User** selects the options that suit their own needs.
- 4. **User** can then choose to alter the variables to suit their own needs
- ← 5. **System** checks the variables to see if they are legal and informs the user if they are not
--- If variables are not legal, user is returned to step 4 ---
- ← 6. **System** formats the simulation output and produces a file for the user

Flow of Events for Extensions:

4. d. System State Diagram



*



5. Effort Estimation Using Use Case Points

Non-Registered User:

It will take 11 Clicks to Register to the web page for a user. For a new user it could take 15/16 click to register, even to get familiar with the website. So new user needs average 14 clicks for the registered to the website. Includes typing in the initial information in the sign-on screen and then being directed to the registration page.

- A user needs 2 clicks to log in, for a new user may need 4 clicks to log in.
- A user needs 1 click to simulate, for a new user may need 2 or 3 clicks to get the correct simulation.
- A user needs 14 clicks minimum to register to the web page, log in to the account and get the simulation. A new user may need 18/20 clicks maximum to get the simulation for the beach.

Registered User:

Registered user only needs 3 clicks minimum to get the simulation from the web page and can make decision whether he/she wants to go to the beach or not.

- It will take a registered user 2 clicks to enter the program.
- It will take 1 click to get simulation from the program.
- It will take 3 clicks minimum for a registered user but it may take more than 3 clicks, if user enter wrong email address or password, then they need more than 3 clicks to get the result.

Running Advanced options:**Non- Register User:**

A new user would take 14 clicks plus 14 clicks to enter and alter all the demographics and simulate, and 14 clicks plus 8, total 22 clicks for default and simulate. It may need more than 22 clicks for new user to get correct demographics and desire simulation.

Registered User:

- A registered user would take 3 clicks plus the 14, total 17 clicks to set and simulate a determined simulation. User may need more than 17 clicks if they enter wrong email add or password.
- and 3 clicks plus 8, total 11 clicks for defaults and simulate. Registered user may need more than 11 clicks if they enter wrong email address or password or user wants to change the default value to compare the simulation.

General:

Most clicks in this program are to select a data and run the simulation. Only a few clicks are designated to input data. Especially when a user need to register their account.

6. Domain Analysis

We derive the domain model concept starting from the responsibilities mentioned in the detailed use cases. Following table lists the responsibilities and the assigned concept.

Concept Definitions

The concepts and their responsibilities are discussed below.

Deriving concepts from responsibilities identified in the detailed use cases .

Responsibility Description	Type	Concept Name
Chooses whether to go to the beach or not depending on various conditions set by the user	K	Agent
Decides whether to change decision in the next round based on previous outcomes	K	Agent
The medium that the user uses to access the simulation online	D	Website
A simple and effective display for the user to interact and obtain information from the simulation	D	Interface
Shows default values to the user	D	Interface
Allows the user to select advanced options using a button interface	D	Interface
Displays all of the advanced options available to be modified by the user	D	Interface
Displays an error if the changes are not legal	D	Interface
If user modifies default conditions, checks if modified conditions are legal	D	Simulation
Runs the game once user succeeds in inputting correct parameters detailed in UC-1 Main Success Scenario	D	Simulation
Checks if the changes made to the advanced menu are legal	D	Simulation
Registers Visitor into database and the registered Visitor is now considered a User	K	Database
Stores past simulation results, user registration information and default values	K	Database
Keeps track of how many agents went to the beach for the current round	K	Beach

Keeps track of a user's login information, status, and simulation data	D	User Profile
Allows the user to get the simulation outputs as a portable file	D	Exporter

Association Definitions

Deriving the association of concepts listed in concept definitions table.

Concept Pair	Association Description	Association Name
InterfaceSimulator	The Interface sends the input ,given by the user, to the simulator	Inputs
SimulatorInterface	The Simulator sends simulation data to the interface to be displayed to the user	Simulator Updates
Simulator Database	The Simulator sends data to the Database where all the data about each simulation is stored for future use and reference	Store data
DatabaseSimulator	The Simulator may use past simulation data stored in the Database to influence current simulations	Data Lookup
SimulatorAgent	Simulator creates a number of agents based on population data	Agent Creation
AgentBeach	Agent sends its decision to the Beach whether they are going or not	Agent Decision
BeachSimulator	The Beach gives the data of how many went to the beach and how many did not go to the beach	Beach Tally (Outcome)
InterfaceUser Profile	The interface sends login information to the user profile	Send login
SimulatorUser profile	Updates the information in the account after each simulation	Update
Simulator Exporter	Sends the simulation output to the exporter	Send output

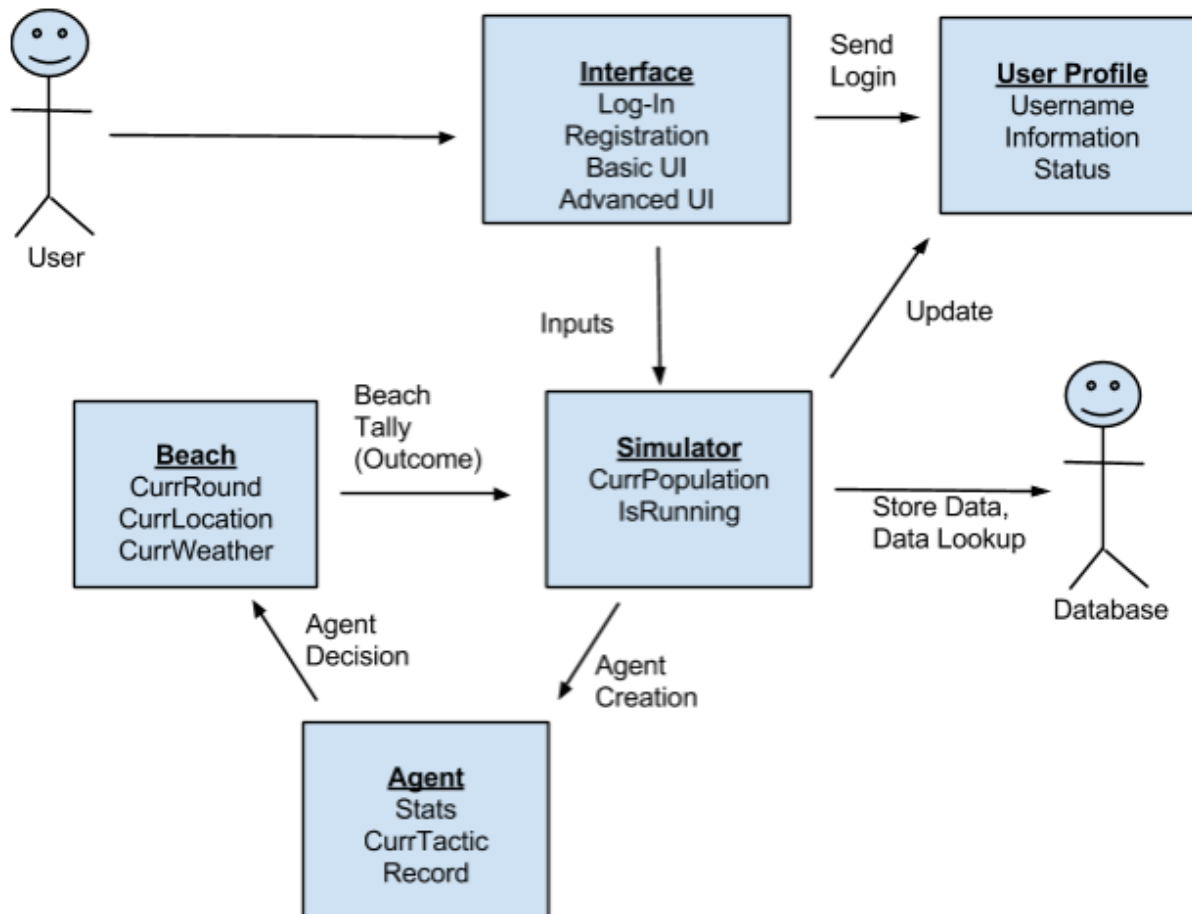
Attribute Definitions

Deriving the attributes of concepts in concept definitions table from responsibilities identified in detailed use cases.

Concept	Attributes	Attribute Description
Interface	Log-In	Allows users to log in to the website
	Registration	Allows visitors to register for an account
	Basic UI	Contains basic input interface for users that are looking for a quick and easy way to get simulation data. Also contains the Advanced UI button.
	Advanced UI	Contains the advanced input interface for users looking for more in-depth control of the simulation.
Beach	CurrRound	Keeps track of the current round of simulation
	CurrLocation	Keeps track of the location of the beach
	CurrWeather	Keeps track of the weather at the beach's location
Agent	Stats	Each agent keeps track of their own gender, age group, etc.
	CurrTactic	Keeps track of the agent's current strategy
	Record	Keeps track of the agent's win/loss record
Simulator	CurrPopulation	Keeps track of the current population based on the location given. The number of active agents is based on the population
	IsRunning	Keeps track of whether the game is running or not
User Profile	Username	Keeps track of a user's login information
	Information	Keeps a record of a user's activity and past simulations
	Status	Keeps track of whether a user is logged in and/or whether they are running a simulation or not

Exporter	Input	The input to be formatted
----------	-------	---------------------------

6. a. Domain Model Diagram



Traceability Matrix

			Domain Model			
		Interface	Beach	Agent	Simulator	User Profile
Use Case	PW					

UC1	5	X	X	X	X	X
UC2	3	X				X
UC3	3	X				X
UC4	4	X			X	
UC5	3	X			X	
Max PW		5	5	5	5	5
Total PW		18	5	5	12	11

6. b. System Operation Contacts

Operation	Name of operation and parameter
Cross reference	UC-1
Precondition	User login Input the default values based on the location given.
Postcondition	Data valid, simulate the system. Save the simulation. Store the result.

Operation	Name of operation and parameter
Cross reference	UC-1
Precondition	User login Input the default values based on the location give.
Postcondition	If the inputted parameter are outside of range, Data invalid. Simulate does not run.

Operation	Name of the operation and parameter
Cross reference	UC-4

Precondition	User log in. User can modify inputted default values based on location. User can choose the advance option button.
Postcondition	Data valid, simulate the system. Analysis graphs and visual Demographics. Save the simulation. Store the result..

6. c. Mathematical model

The foundation for our mathematical model for this application is based within game theory. The game theory that our model resembles is the El-Faro Bar minority game. The purpose of the El-Faro Bar minority game was to create a statistical model in which you could use to predict the satisfaction of visiting a bar. When the occupants of a bar exceed 60% that satisfaction disappears and the agent would much rather stay at home. However, while the bar occupancy is under 60% the user would have an enjoyable experience at the bar. However, in addition to this version of the minority game, we added our own little tweaks. First and foremost, instead of simulating a bar's experience, we simulate a beachgoers experience. For example, going to the beach is fun and all, but going to the beach too many days in a row would result in decreased satisfaction and would be a monotonous discouragement in which the agent would rather take a break from going to the beach. This also applies in the opposite scenario. What if the agent hasn't visited the beach in a week, clearly the agent's strategy is not a winning strategy so the agent might reconsider its conditions to visiting that beach.

Variables that determine the desirability of visiting the beach on a given day include the temperature, the humidity, the weather, the temperature, and other various climate effects. These climate effects sum up to form the desirability index for the given day that we simulate. The climate effects are represented through gaussian curves that represent the desirability that each demographic will feel in general in regards to the conditions presented. For example, rainy weather would decrease the desirability levels for most demographics, and sunny weather will increase the desirability level. We run this simulation on the same invariable conditions 100 times to get the general consensus of how many people we would expect to visit the beach in that situation. The agents are initialized with the following information, their own individual desirability index, and their own personal demographic desirability level in which to compare their index to.

The choice in whether or not the agent chooses to go to the beach lies in their individual desirability index in comparison to their demographic desirability level. The following is a binary comparison in which if the demographic desirability index is out of bounds in comparison to their individual desirability index the agent would choose not to go to the beach and if it is within the bounds, the agent would proceed to head to the beach. After determining whether the agent will go to the beach on a given day, the agent's decision for that day is marked down. If a great number of people visited the beach on that given day the user would not be satisfied with their decision, and would choose to lower their expectations of going to the beach agent in the same

conditions. Otherwise, if the user had a good experience going to the beach on that day, the user would keep their decision making rather stable, but would still account for the factor that the user may not care too much for always going to the beach. This factors into a minor drop in the individual desirability index in comparison to the much larger drop brought upon by a bad decision in which too many people went to the beach, or too few people went to the beach. The translation of these factors are in the form of: Satisfied = $15\% < \text{beachgoers} < 60\%$, Dissatisfied = All other cases. If the agent has been to the beach consecutively for multiple instances, the agent will essentially excuse themselves from heading to the beach for a period of time equivalent to their reevaluation period. During this evaluation period the agent will delay and not alter their strategy. However, once the reevaluation period ends, the agent will reinitialize their individual desirability index in order to once again allow itself the chance to enjoy the beach environment. Through these factors the relative attendance to the beach will stabilize and the enjoyment factor will increase for all parties involved. Each agent is initialized with their desirability index range(DIR) for negative values between $-50 \rightarrow 0$ and for positive values between $0 \rightarrow 50$.

Climate variables have their values pulled from the table for their specific demographic. For example(not using exact measurements here), Sunny would be +5 on the desirability index while Rainy would be -5.

These values are then summed up for each individual demographic and forms that demographics desirability index.

i.e. Sunny + 25C + 50% humidity = DI

If $(\text{min DIR}) < \text{DI} < (\text{max DIR})$

The agent will go to the beach.

Otherwise the agent will not. If the agent does not go to the beach the evaluation period begins, and the time spent in the evaluation period is incremented by 1.

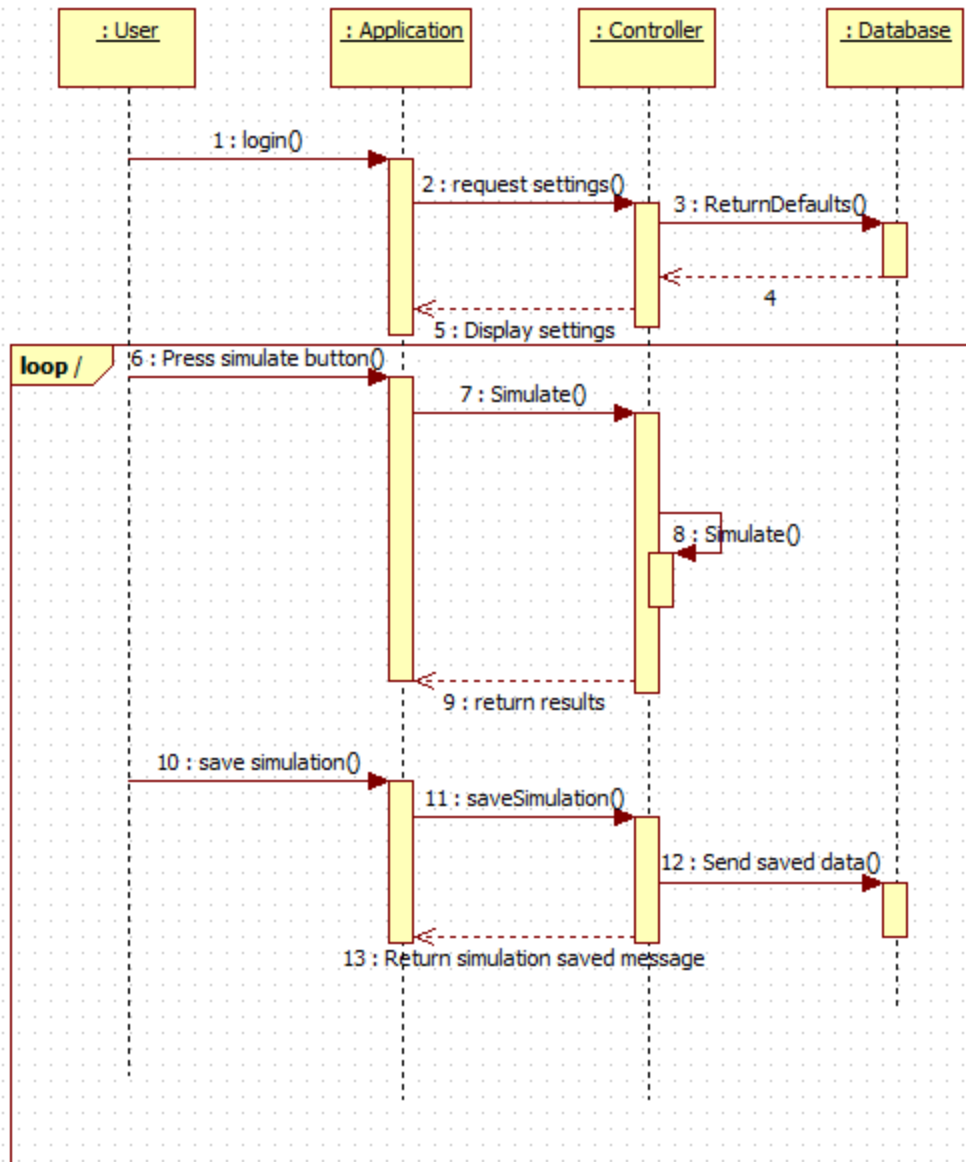
Once the evaluation period time == the agents maximum evaluation period the agent resets their DIR into new randomized values.

If the agent is satisfied on a particular day, their DIR index will be reduced minimally to represent the resulting change from spending lots of time at the beach. However, if the agent is not satisfied, the DIR index will be reduced by a much larger amount.

Using these conditions, the agents undergo the simulation with the given conditions eventually leading to a rather steady line in which the agents are on average satisfied with their outcome, and choice, as well as their time spent on the beach.

7. Interaction Diagrams

UC-1: Simulate

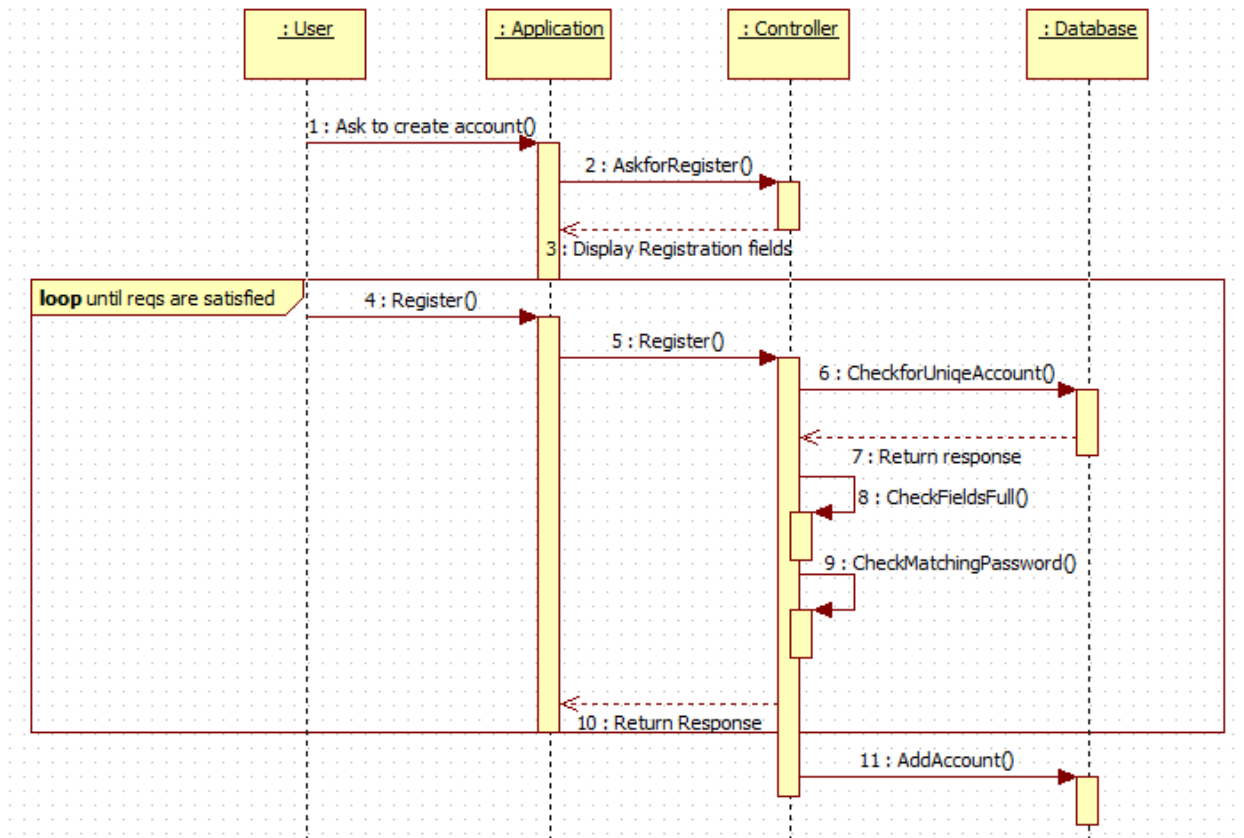


Description:

The simulator use case simply for settings and then runs the simulation based on settings. The defaults for the specific login are stored in the database. After a login, the defaults are returned. When the user presses the simulate button, the controller starts the simulation and when it's done, returns the results. The simulation can be saved into the database for later access.

The controller contains the main program, which handles simulations and stores temporary data. The application is the main GUI that the user sees. The database is an online MySQL database that stores accounts and data associated with them.

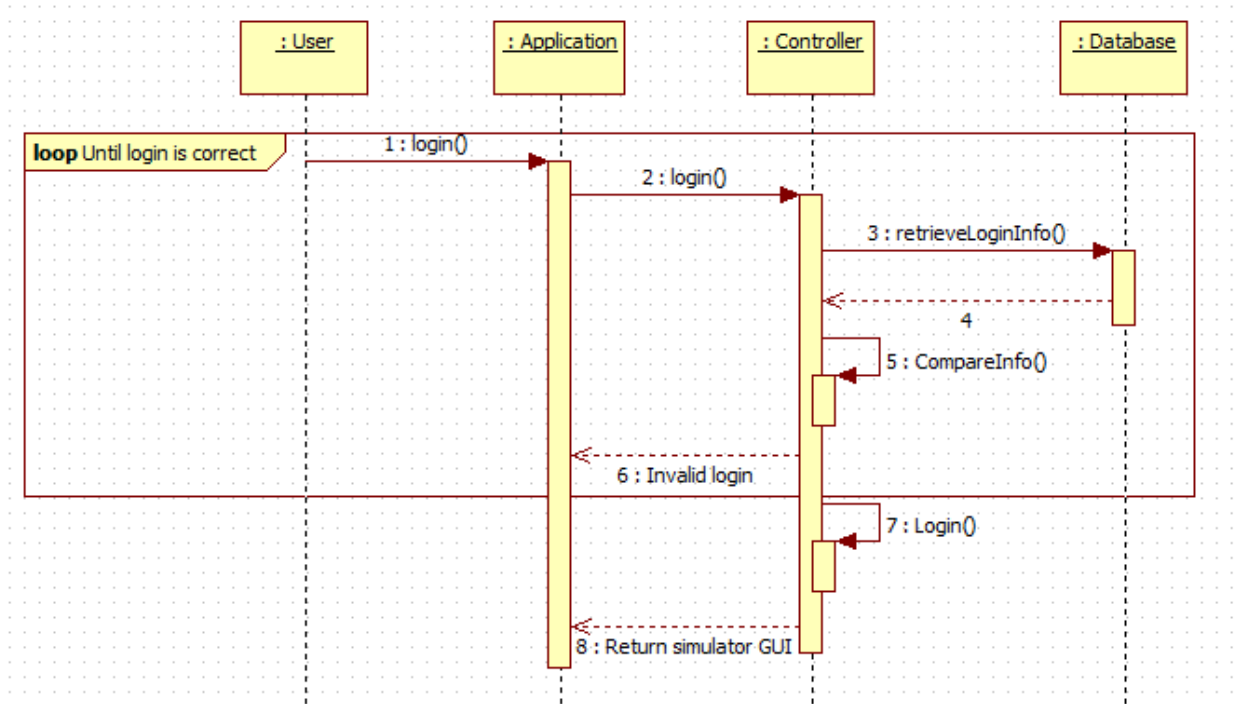
UC-2: Registration



Description:

First the user asks to register. The controller will bring up the form for registration. When the User registers, the data is sent to the controller, who makes sure the registration is valid. First it asks the database if the account is unique (by Email). Then it checks if all the required fields are full. Finally it checks if the password and the confirm password fields are the same. If any of these are not fulfilled, it returns an error and doesn't register. If all requirements are fulfilled, it returns a success text and saves the information to the database.

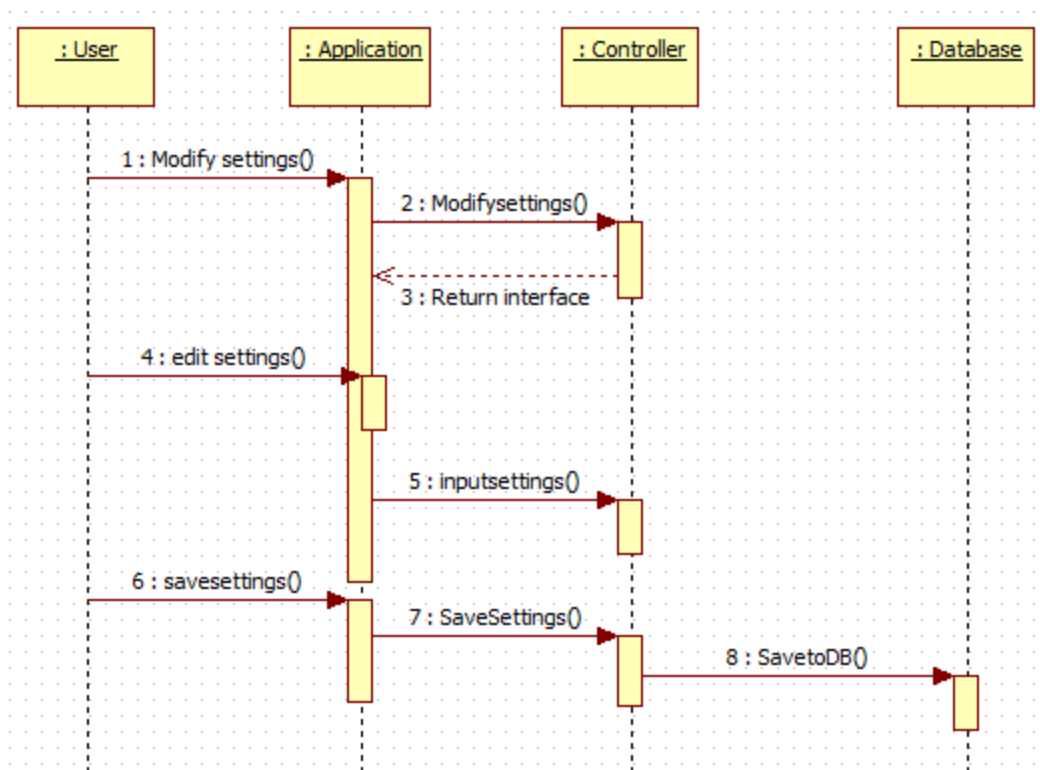
UC-3: Login



Description:

The user attempts to login with user-entered credentials. The application accepts the login info and sends it to the controller. The controller retrieves the password related to the Email and compares it with the password the user inputted. If they don't match, it doesn't log in. If they do match, the user is logged in and the simulator window is shown.

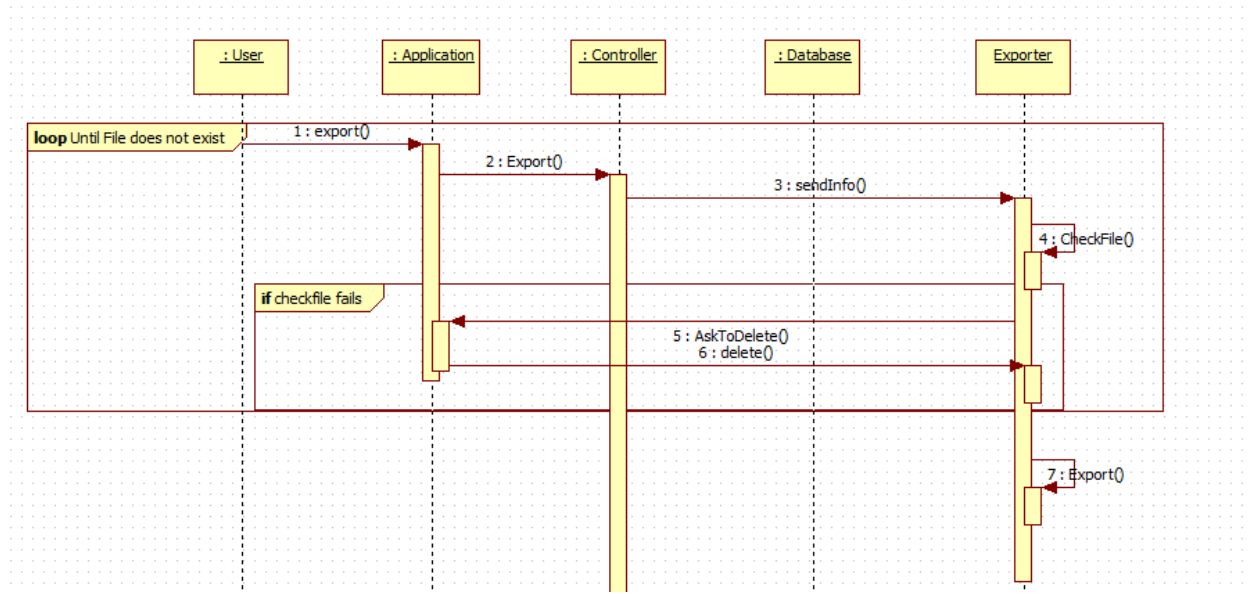
UC- 4: Modify Simulation Settings



Description:

Use case 4 allows the user to modify the settings for the simulation. First the user requests to modify the settings. The controller returns the modify settings display. When the user edits the settings, the controller will accept them. The User can also save the settings. The Controller will accept the settings and store them in the database. These settings will become default for that user.

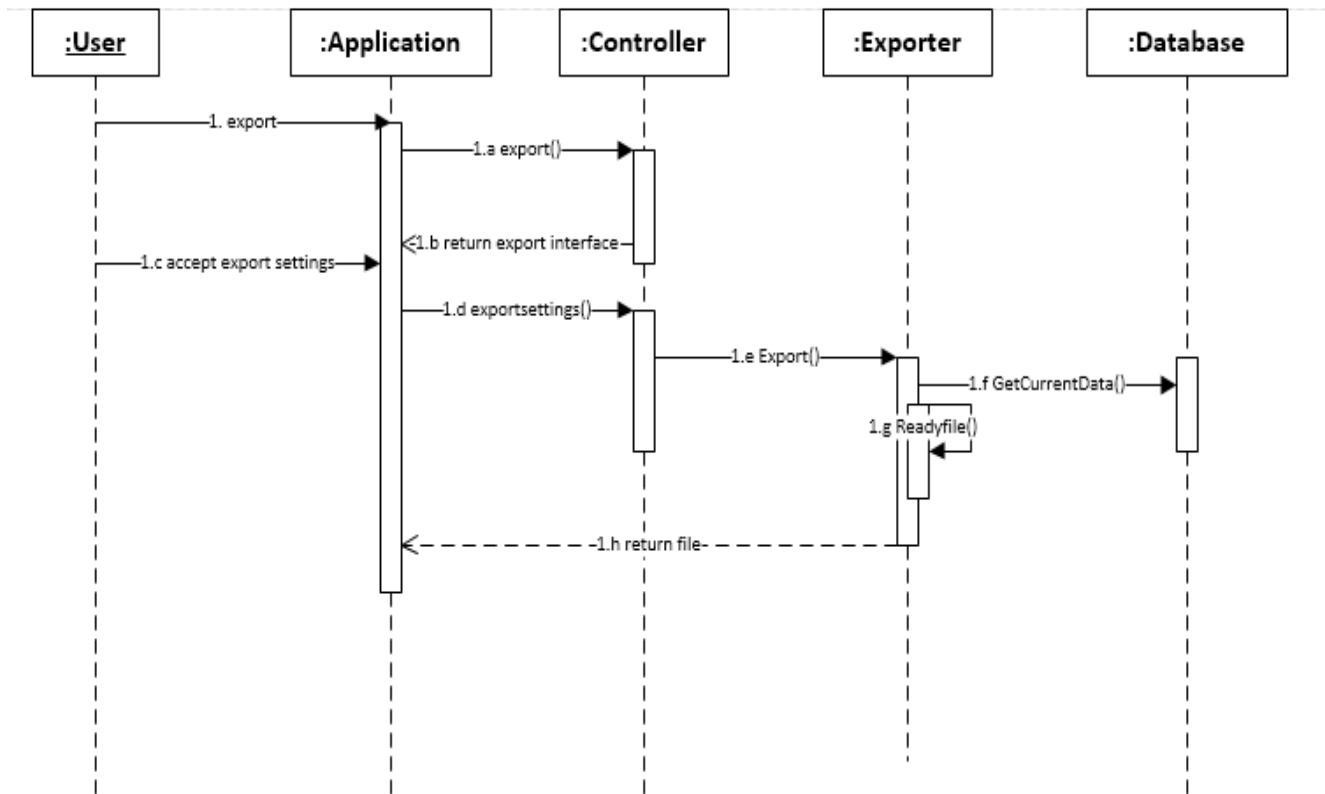
UC-5: Export



Description:

This use case starts after the user finishes the simulation and asks to export. The Controller sends a request to the exporter, along with the data to be exported. The exporter checks if there already is a file in the same place with the same name as the file about to be exported. If there is, the exporter asks to delete the file. If they don't delete, the exporter can't export. If they do, the exporter sends the file as a text file.

UC-5: Export Alternate Scenario

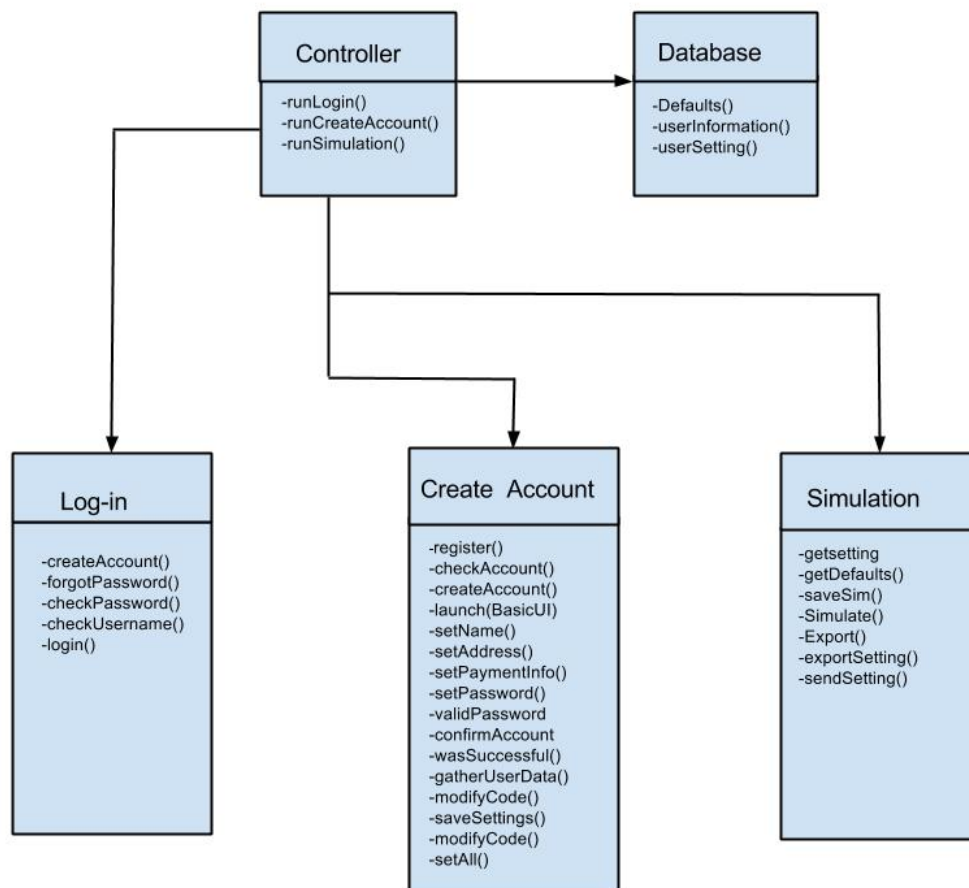


Description:

In this alternate solution for UC-6 the control does not access the data directly. The signal to export is send directly to the exporter, then the exporting options are selected. The exporter takes the simulation data from the database itself, formats it, and sends it to the user.

8. Class Diagrams and Interface Specifications

8. a. Class Diagram



8b. Data Types and Operation Signatures

The Application/Controller Class is given the responsibility of creating and managing the Accounts and Account List. The Application Class contains the attributes checkAccount() and createAccount(). checkAccount() will search the Account List for a similar account returning a Boolean response. Once this response is true, createAccount() will start to build a new Account structure. The application will then call its series of functions

to input the various inputs into their corresponding values in the new account node. Once all these are inputted, addAccount() will then place the new structure into a new record in the our database.

The Controller class allows the user to control and retrieve settings from the Database. As such, it has attributes such as: changeUserSettings(), getSettings(), and saveSettings() to achieve its purpose. The Controller is also responsible for controlling the forms that are visible, and link in the database. When the program first starts, the controller will call the login class and form. This class will run and occasionally check back with the controller when needing to verify data in the database. Once successful the controller will move onto the Create Account or Simulation class and form depending on the Login. Here it will pull default data from the database and push it to the Simulation form and class.

The Database class is a structure that holds important information used by the Simulator. It contains the Defaults of each setting of the simulation which allows users to quickly run a simulation without fiddling with the different settings. If the user does decide to edit and save some settings, the Database will save this within UserSettings. SimData contains the results of previous simulations and AI contains the currently used AI algorithm. UserSettings, SimData and AI are all account specific.

The Simulator class is responsible for the user's simulation session. First, the Simulator calls login() when the user first starts up the program. The first thing the Simulator does once the user logs in is it calls getDefaults() to create defaults for every setting of the simulation. Once the defaults are retrieved from the Database and the user accepts the defaults, the simulation is run using Simulate(). The user can then save the results, which uses saveSim(), simulate again, or Export() the results. If the user decides to Export() the results.

8. c. Traceability Matrix

Domain Concept Vs. Classes	Class	Class	Class	Class	Class
Concept	Controller	Login	Database	Simulator	Create Account
Registration	x		x		x
Login	x	x	x		
Simulation	x		x	x	
Export	x		x	x	

Modify Simulation Settings	x		x	x	
---	---	--	---	---	--

The Login and Registration concepts work together to allow users to create and access their very own personal accounts within our database. These concepts were created to give each user a personalised experience and be able to save their preferred simulation settings in the database to be pulled from later.

The Simulation concept is in charge of the entire simulation. It creates and manages the agents and any other variables within the simulation. It does this by pulling the simulation settings from the GUI and parses it to make sure that the simulation settings are valid and if the simulation settings are valid the simulation runs. It also saves simulator results into a temporary array for the user. If the user should choose to export their simulation that array will be written to a txt file for the user. If the user does not export, and closes the application, or runs another simulation, the data in the array is lost as it is a temporary storage solution.

The Simulation Settings concepts allow users to edit the simulation variables. They also check whether the new user defined settings and algorithm are valid within the system parameters and, will save them into the database for future easy access if the user so desires.

The Export concept is simply a concept to allow users to export their results in a txt file from the resulting simulation array.

8. d. OCL Contract Specification

Registration

```

context Controller::registration(b: Button) : boolean pre:
    inputs.text.length() > 0
context Controller::registration(b: Button) : Boolean
    --postcondition (Poc1')
post: let allValidKeys : Set = self.checker.validKeys()
    if allValidKeys.exists (vk | k = vk) then
        if account.exists() then
            invalidaccountname()
        else
            account.passwordset()
    else
        invalidsubmission()

```

Login

```

context Controller::login(b: Button) : boolean pre:
    inputs.text.length() > 0
context Controller::login(b: Button) : Boolean

```

```

--postcondition (Poc1')
post: let allValidKeys : Set = self.checker.validKeys()
      if allValidKeys.exists (vk | k = vk) then
        if account.exists() then
          if password.correct() then
            application.simulation()
          else
            wronginfo()
        else
          wronginfo()
      else
        invalidsubmission()

```

Simulate Button

```

context Controller::simulate(b: Button) : boolean pre:
  input.text.length() > 0
context Controller::simulate(b: Button) : boolean pre:
  sumofdemographics() > 0
context Controller::simulate(b: Button) : boolean pre:
  comboboxselection() != NULL
context Controller::simulate(b: Button) : Boolean
  --postcondition (Poc1')
post: let allValidInputs : Set = self.checker.validInputs()
      if allValidInputs.exists (vi | i = vi) then
        runsimulation() implies
          simulationisrunning() = 1
        simulationisrunning() = 0
        simulationhasrun() = 1
      else
        invalidfields()

```

Export

```

context Controller::export(b: Button) : boolean inv:
  simulationhasrun() > 0
context Controller::export(b: Button) : boolean pre:
  simulationisrunning() == 0
context Controller::export(b: Button) : Boolean
  --postcondition (Poc1')
post: if simulationhasrun() <= 0 then
      export.disabled()
    else

```

```

        if simulationisrunning() != 0 then
            export.disabled()
        else
            export()

```

Save Settings

```

context Controller::saveSettings(b: Button) : boolean pre:
    input.text.length() > 0

```

```

context Controller::saveSettings(b: Button) : boolean pre:
    sumofdemographics() > 0

```

```

context Controller::saveSettings(b: Button) : boolean pre:
    comboBoxselection() != NULL

```

```

context Controller::saveSettings(b: Button) : Boolean
    --postcondition (Poc1')
    post: let allValidInputs : Set = self.checker.validInputs()
        if allValidInputs.exists (vi | i = vi) then
            saveSimulationSettings()
        else
            invalidFields()

```

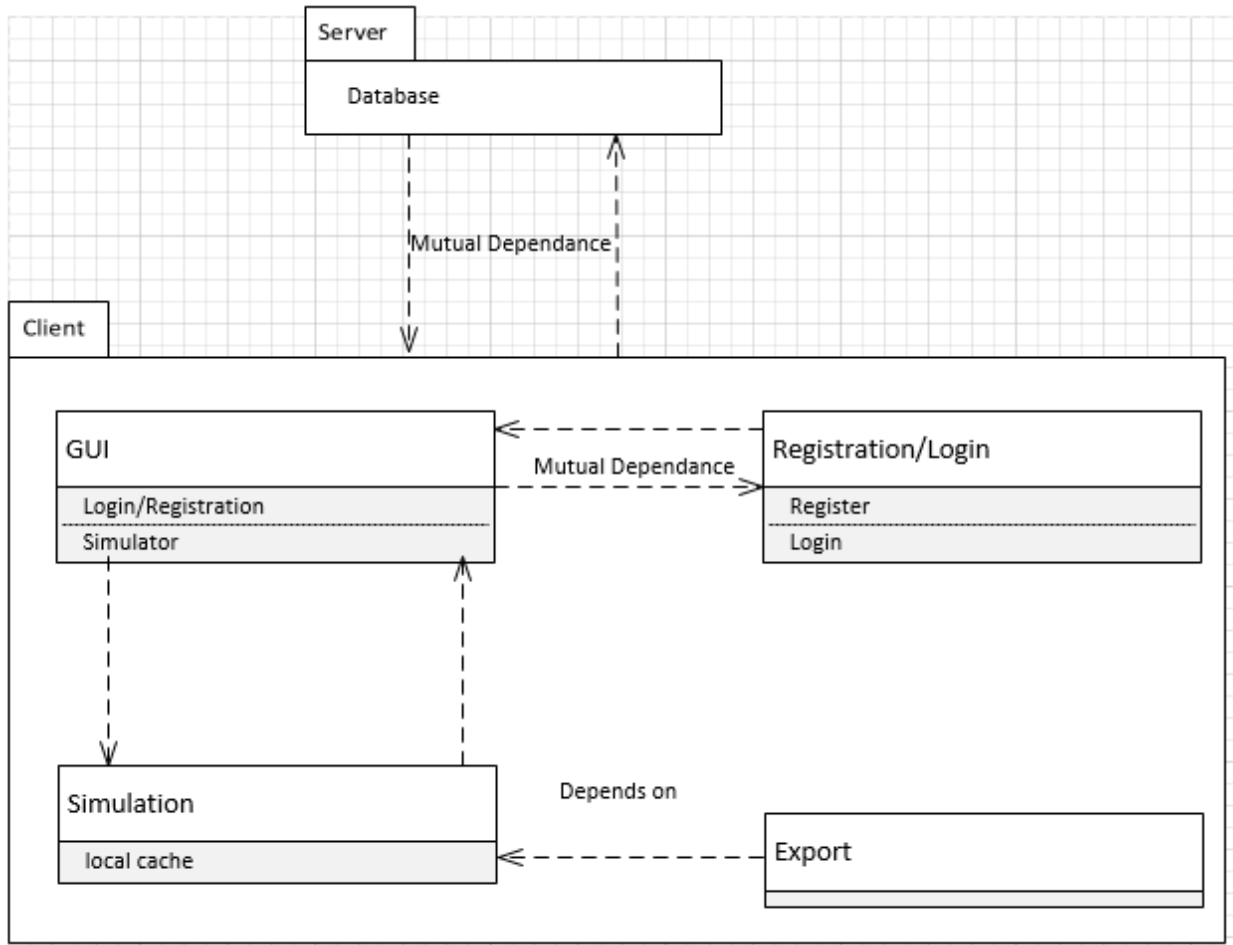

9. System Architecture and System Design

9. a. Architectural Style

The architectural styles that are used in this application are a mixture of the following: the client/server model, the event driven model, and implemented with a touch of object oriented design. How we use the client/server model in our application is through the application acting as a client and the server being the database in which we store our consumer information. The client requests the server for the validity of the user account, and the server responds with whether or not the user account exists. If the user account exists the server will respond accordingly and then the client may unlock the application for the user to use. In addition the client/server model is used once again to determine the user preferences once the application is loaded. The settings that were stored on the server are preloaded into the client, and upon request to restore settings/save settings the client will call the server with the user information and either retrieve the settings or upload the new preferred settings. The event driven model comes into play once the user is logged into the application. The event driven model is implemented in our case through user selection of the actions they wish to drive. These actions themselves are programmed with object oriented design in mind. This means that the actions are modular and self-sufficient while running. The actions require no interaction with each other and contain their own data and objects with the exception of the export use case which relies on the simulation data to exist. This simulation data can only exist once the simulation has run its course. The user interface is also filled with masked text boxes to help in ensuring that the user does not attempt to fill in any faulty data that may corrupt the simulation and crash it. This data is pulled off of the interface when the user attempts to drive the simulation data. If any data is missing the user will then be informed of an attempt to run the application with insufficient information. Otherwise, the simulation will run with storage of the simulation results into a temporary local cache. It is at this point that the simulation results can be exported in a form that the user prefers.

9. b. Subsystems

The major subsystems of our application are the client (which includes the simulation module and the export module) and the server (which holds our database information that the client must interact with in order to run the application). The server contains the registration data and the user settings and the client contains the original shell that masks the full application until the user submits correct information that will allow the user to login. The simulation subsystem is fully functional without any dependencies except user data.



9. c. Subsystems and Hardware

The server will exist online in the form of a SQL database. The client will, however, require a computer running a windows operating system that includes .NET 3.5.

9. d. Persistent Data Storage

Persistent data storage exists only in the form of the online database which stores our user information and their settings. Local storage (the local storage is contained in an array) will handle each individual simulation, and if the user wants the information from the simulation to keep, the user must download that information through the export function. This information in the local storage is overwritten on each new simulation run so if the user does not export the simulation results, the results will be lost as there is no persistent storage in regards to the actual simulation. As for the user information, the SQL database contains columns which designate the specific user information, upon login or registration, the client calls these tables and either writes to them for registration, or reads through them for login calls. The database then returns with the values requested by the client and the user can continue running the application.

9. e. Network Protocol

The system requires internet connection in order for the client to communicate with the server database. To communicate with the server, the application with use SQL calls, through Visual Basic commands, to access the database.

9. f. Global Flow Control

i. Execution orderliness

The program execution is event driven and requires the user to submit the requests before the server will process the information. In every case, the user must save their settings if they wish for the new settings to become the default for the simulation. The execution order in the instance of clicking the simulation button, is that the application generates the agent table and loops for 100 simulations using the mathematical model described beforehand, all the while, storing the data in a table and writing the data series to the graph. The export use case takes the data from the table generated by the simulate button, and writes that table data onto the output stream in the form of a text file.

ii. Time dependency

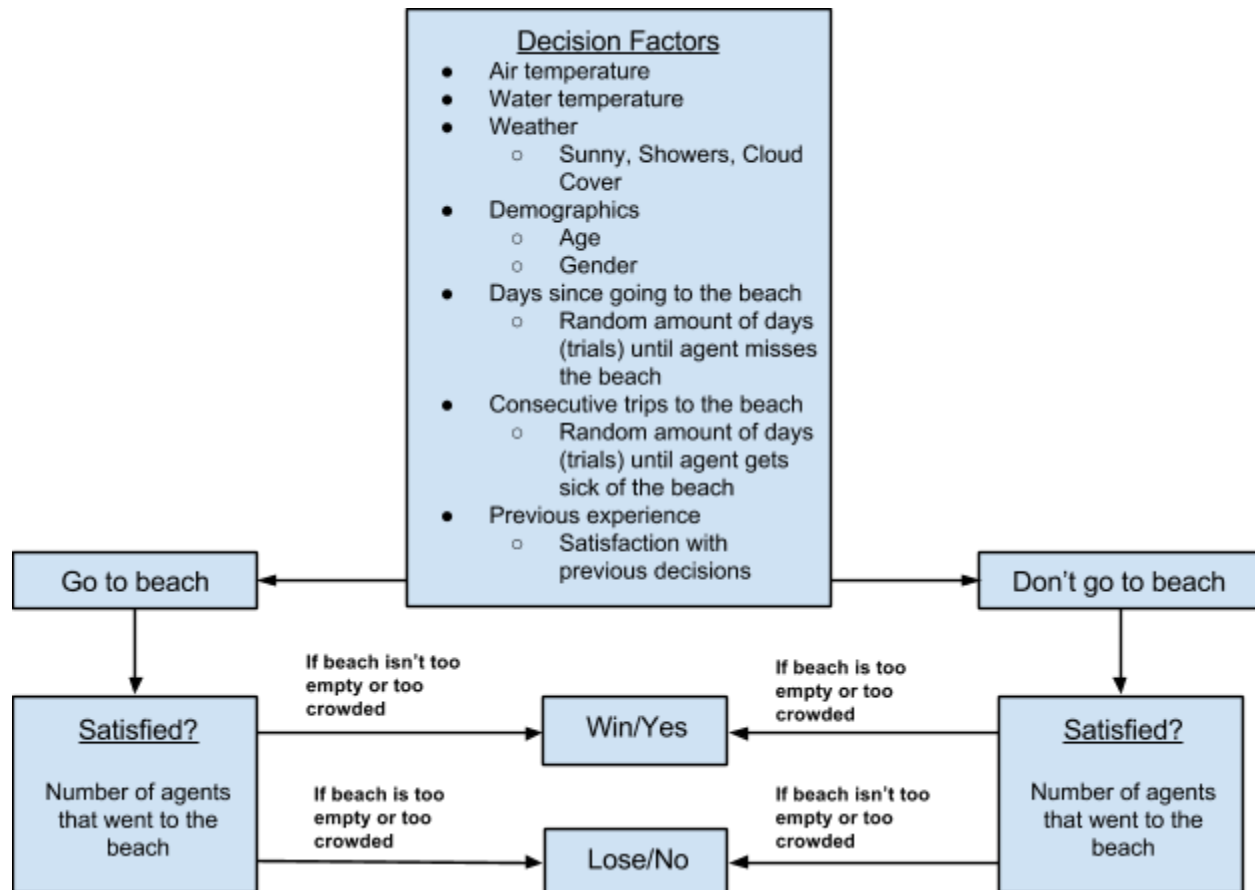
There are no timers on the system currently. The planned system changes will, however, take the different times and instances of the year into account when simulating the events that are expected to take place. The export use case does require that simulation be run before any data can be exported accordingly.

9. g. Hardware Requirements

	Minimum	Recommended
Display Resolution	640x480 color display	1024x768
Browser	N/A	N/A
Operating System	Windows (with .NET 3.5)	-----
Network	56Kbps	1Mbps
HD Space	100MB	-----

10. Algorithms and Data Structures

10. a. Algorithms



10. b. Data Structures

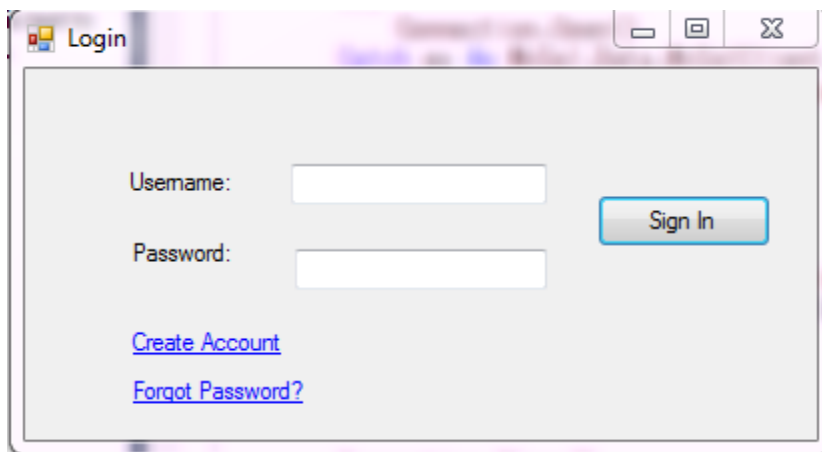
The application will be using a SQL database to store the settings, and users rather than linked lists or arrays. The reason why a database was chosen was for the ease of use, and performance, as well as the scalability of the database. However, for the simulation itself, an array was chosen due to the speed and ease in which the final product could be calculated for each individual agent. This array stores the agent's individual preferences and information relevant to their initialization. The array itself is a temporary storage array which is reinitialized upon each press of the simulation button with new parameters and new agents. The agent array is used

comparatively with the climate arrays which are static and constant due to demographic preferences remaining relatively constant. The climate arrays, unlike the simulation array are not reinitialized with each iteration of the application and are not modified through the course of the application's runtime.

11. User Interface Design and Implementation

11. a. Preliminary Design

Initial Screen:



Just like a standard log-in, the user will enter the username (his e-mail) and the password. There are three buttons to click on this initial screen.

1. Sign-In: Will then launch the basic UI Screen (See below)
2. Create Account: Which will launch the Registration UI Screen (see below)
3. Forgot Password: Which will launch standard password recovery (Not shown below)

Registration UI Screen:

On this screen user will enter required data to start his account. Most fields are just plain text boxes, however there are a few check boxes and drop downs for state. You can then click create which will enter you right into the program's Basic UI Screen.

Create Account

Personal Information

* First Name: * Last Name:

Business:

* Address:

* City: * State: * Zip:

Login Information

* Email:

* Password:

* Reenter Password:

Payment Information

*Card Number: *CVV:

*Card Type: *Exp Date: *Name on Card:

Billing Address

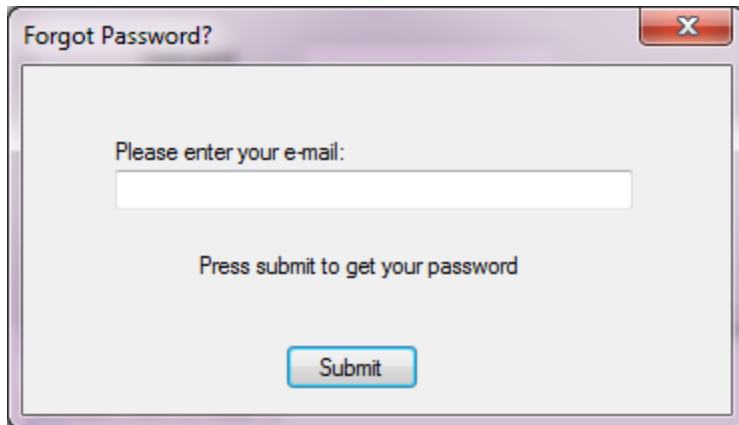
Address1:

Address2:

City: State: Zip:

Submit

Forgot Password:



Forgot Password?

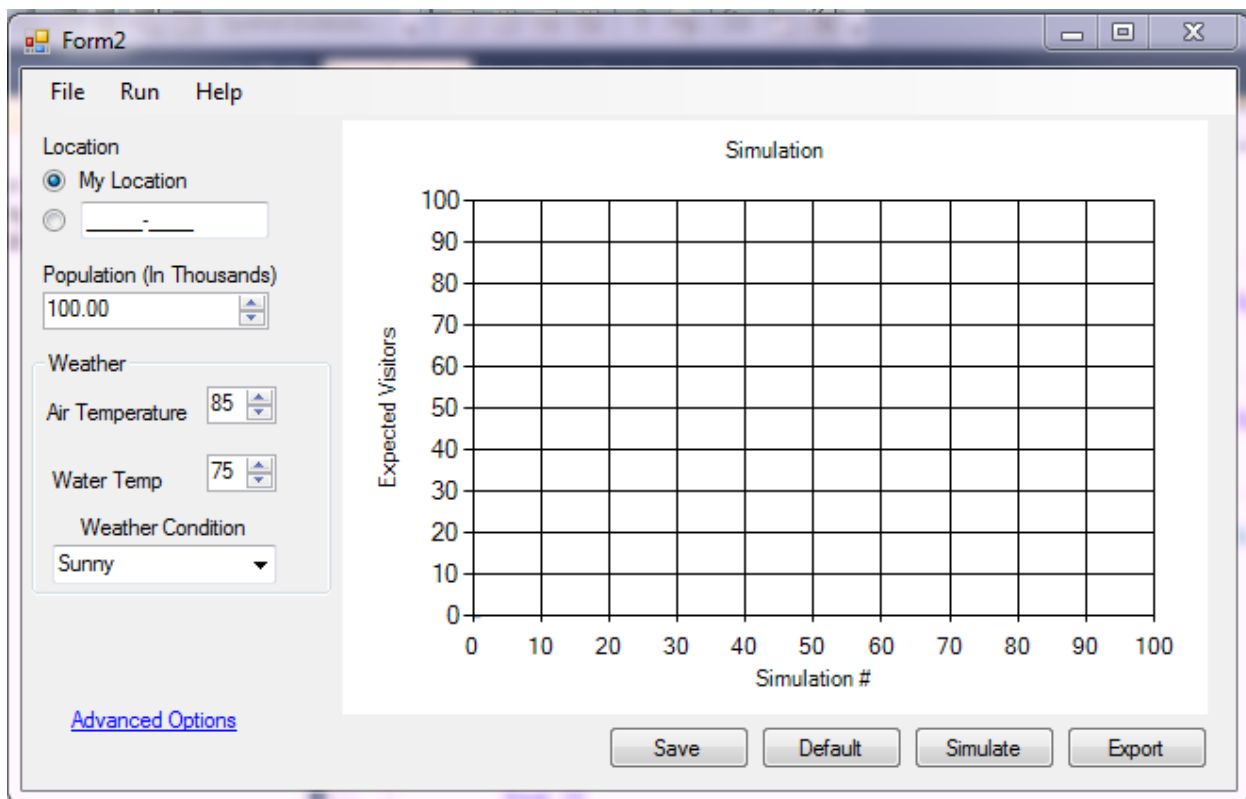
Please enter your e-mail:

Press submit to get your password

Submit

When forgot password is clicked, this pop-up will assist in resetting your password. Your accounts password will be reset and you will receive an email with your new password. This also cross checks to make sure you get an account.

Basic UI:



Form2

File Run Help

Location

☒ My Location

☐

Population (In Thousands)

100.00

Weather

Air Temperature 85

Water Temp 75

Weather Condition

Sunny

[Advanced Options](#)

Simulation

Expected Visitors

Simulation #

Save Default Simulate Export

The basic UI will look like like this. On the left you will have your basic variables you can alter, and on the right will be your simulation graph report. The variables will be always previously set to default values, and are

represented as checkbox groups, therefore something will always be selected. This also cuts down on the minimum number of clicks necessary to run the program. Default values are based upon current location or saved settings of the user.

On this you have three action clicks.

Action clicks:

1. Save Settings: This option will take current settings and set them as the default. Once the user logs on they will be automatically selected. If this button is never clicked, default settings will be set as the programs default settings.
2. Simulate: This button refreshes the graph by re-running the simulation inputting the new settings selected. If there is no change it will just re-run the simulation with the same settings selected.
3. Advanced Options: This hyperlinked phrase, will expand the basic UI view to the Advanced UI (see below).

Advanced UI:

Form2

File Run Help

Location

☒ My Location

☐ _____

Population (In Thousands)

100.00

Weather

Air Temperature 85

Water Temp 75

Weather Condition

Sunny

[Advanced Options](#)

Save Default Simulate Export

Demographics

Gender

Male: 50%

Female: 50%

Age Group

0-9 20.00

10-19 20.00

20-39 20.00

40-59 20.00

60+ 20.00

Simulation

Expected Visitors

Simulation #

Amount Distribution

child

teen

young adult

adult

elderly

The change in this UI is the addition of population demographics. These are originally set at a default relative to the inputted location. Once this section is opened you are presented with two slider groups and another graphic. The two slider groups will always equal 100%. When sliding the men's slider lower, the women's slider will increase. Same relative to the age groups. Same with the basic, the simulation and the pie chart graphics will update corresponding to the inputted values.

One implementation that is being added, is the auto-fill of values like Location, based on users current location. This, as well as an auto-fill for remembered user, will help with user ease-of-use, being able to sign in with two clicks instead of three. Default values derived from user's current/selected location will allow for fast and easy simulations without requiring the user to enter his/her own values. But, the option to change those values are still within the user's power.

Another implementation that is being added is the ability for the user to export their simulation results into a format of their choosing. This export function will help users to easily extract their new found information into something that they can share and present to others.

12. Design of Tests

These are test cases for determining the correctness of implemented structure in our program:

Simulation Button

Unit to test: Simulation Button/Function

Assumptions: Valid data values for simulation fields have been input into the GUI.

Test data: Inputted data

Steps to be executed:

(1) Press the Test Button

Expected result: A Graph will display right side of the screen for valid inputted data.

Pass/Fail: Passes if system generate a graph right side of the screen, and the dataset pulled off the chart matches the values passed into the chart. Fails if anything else occurs.

Comments: This test makes sure that the data will be visually accessible to the user. The test also makes sure that the simulation is actually running, and the data being loaded onto the screen is not previous simulation data.

Save Settings

Unit to test: Save Settings Button

Assumptions: Valid data values for simulation fields have been input into the GUI.

Test data: Inputted data

Steps to be executed:

(1) Change the Settings

(2) Press the Test Button

Expected result: The settings should be saved under your account.

Pass/Fail: The test is passed if the comparison of the current settings matches with the settings stored in the database.

Load Settings

Unit to test: Load Settings Button

Assumptions: Valid data values for simulation fields have been input into the GUI.

Test data: Inputted data

Steps to be executed:

(1) Press the Test Button

Expected result: The settings should be the same as the ones saved under your account.

Pass/Fail: The test is passed if the comparison of the current settings matches with the settings stored in the database.

Export Button***

Unit to test: Export Button/Function

Test data: N/A

Steps to be executed:

(1) Press the Test Button

Expected result: A file has been created.

Pass/Fail: Passes if system generate a test.txt file within the applications directory.

Comments: This test makes sure that the user can retrieve their simulation results after exporting them.

13. Project Management and Plan of Work

13. a. Merging and Contributions from Individual Team Members

--mathematical model added to 5.c. because simulation relies on relationships between random variables

--removed u.c. 5 advanced sim settings and replaced it with export

13 b. Project Coordination and Progress Report

None of the use cases have been fully implemented as of now, however, we have been discussing the algorithms that we will be implementing. The groups have been split up according to the use cases that each group has done their diagrams for. Our group will be working on those use cases individually. While our core use case is the simulation so that is our main priority to get a working simulation running for the first presentation and then for future presentations we hope to get a simple settings up and running without saving into the account database. Then we will work on getting the registration working as well as a logon system. After the login and registration system is fully functional will be try merging the login system with saved settings. The goal for now is to fully implement a working simulation application. The other modules are optional for the first demonstration and if implemented will be simple without the flexibility that the final implementation should offer.

13. c. History of Work

One of the biggest changes was the changes in program platform. We started this program as a web page simulator. While attempting to code everything and reflect our plans onto the web-page we encountered many issues. The hardest part was the inexperience with web-design. We originally started with this approach, since one of our early team members had experience. Since his inactivity in the group, we were left to try and learn everything by ourselves. Even though the ambition and will was there, without even basic knowledge, realistically we were not going to come close to completing our intended project. We then redesigned the application, to still adhere to our problem statement, and still be accessible through multiple computers via a wireless database. This way we could work with easier technology to complete our project goals efficiently and on-time.

13. d. Future Work

One plan for future work, would be to move the program application to a website platform. Also, by going to website extending it to mobile devices as well. Businesses are starting to implement mobile technology in their day to day functions, and to be a useful business application, supporting the platforms a business chooses is vital. Especially since, the actual simulation and project is complete, it is more of re-implementing it.

Two of the most important things we would like to add to our software if we had more time would be to increase our statistical data, and to add a few more factors to our simulation. Right now our statistical data is based upon a very small poll of data collected. For a more diverse statistical model, it would be best to poll a varied group of people to have a more enhanced model to base our predictions off of. Once this is done, we will be ready to pair with a few companies. Our simulation will have a general prediction tweaking the model with actual data we will be able to optimize our solution to the perfect model, better than just a guess.

The most important factor to add in is dates. Special dates like holidays, or even just a time of the month where kids are still in school affect the whole simulation. These have many outcomes and would take a while to implement, however that would make our implementation a lot more encompassing.

One variable that we initially were working with was tidal patterns. For instance, currents and forces (like undertows). However, through the stages of our project we found this to have very little effect on a general beach-goer's decision of going to the beach. With that being said, shops that cater to swimmers like surf shops are greatly affected by these tidal variables, so future work would be to implement an altered simulation equation based on type of store/ specified clientele.

Also, one requirement we were unable to finish was pulling API data directly into the project. We found a source and knew how to complete the task, but it would take a lot of parsing text, that we did not have enough time to complete. That would be a huge improvement for the program. With a little more time and skill, that feature could be implemented to make a huge change.

13. e. Breakdown of Responsibilities

Product Ownership - Updated with Point Totals

Member #	1 Danny Davis	2 Tanzina Farzana	3 Steven Fisher	4 Yang Ren	5 Gegi Oniani	6 Sarid Shinwari	7 Jose Delgado
Project management (13 points)	16.6%	16.6%	16.6%	16.6%	16.6%	0%	16.6%
Summary of Changes (5 points)	16.6%	16.6%	16.6%	16.6%	16.6%	0%	16.6%
Sec. 1: Customer Statement of Requirements (6 points)	16.6%	16.6%	16.6%	16.6%	16.6%	0%	16.6%
Sec. 2: Glossary of Terms (4 points)	16.6%	16.6%	16.6%	16.6%	16.6%	0%	16.6%
Sec. 3: System Requirements (6 points)	16.6%	16.6%	16.6%	16.6%	16.6%	0%	16.6%
Sec. 4: Functional Requirements (30 points)	16.6%	16.6%	16.6%	16.6%	16.6%	0%	16.6%

Sec. 5: Effort Analysis (4 points)	16.6%	16.6%	16.6%	16.6%	16.6%	0%	16.6%
Sec. 6: Domain Analysis (25 points)	16.6%	16.6%	16.6%	16.6%	16.6%	0%	16.6%
Sec. 7: Interaction Diagram (40 points)	16.6%	16.6%	16.6%	16.6%	16.6%	0%	16.6%
Sec 8: Class Diagram and Interface Specification (20 points)	16.6%	16.6%	16.6%	16.6%	16.6%	0%	16.6%
Sec 9: System Architecture and System Design (15 points)	16.6%	16.6%	16.6%	16.6%	16.6%	0%	16.6%
Sec 10: Algorithm & Data Structure (4 points)	16.6%	16.6%	16.6%	16.6%	16.6%	0%	16.6%
Sec 11: User Interface Design & Implementation (11 points)	16.6%	16.6%	16.6%	16.6%	16.6%	0%	16.6%
Sec. 12: Design of Test (12 points)	16.6%	16.6%	16.6%	16.6%	16.6%	0%	16.6%
Sec. 13: History of Work, Current	16.6%	16.6%	16.6%	16.6%	16.6%	0%	16.6%

Status and Future Work (5 points)							
Sec. 14: References	16.6%	16.6%	16.6%	16.6%	16.6%	0%	16.6%
Total Contribution	33.3%	33.3%	33.3%	33.3%	33.3%	0.0%	33.3%

15. References

1. Software Engineering by Ivan Marsic
2. Textbook of Software Engineering by Professor Ivan Marsic
3. Project #3, group #7, spring 2012
4. El Farol Bar Problem and the Minority game Project Description
5. Survey Data Polled from about 52 people.