REPORT 2: Final

Beach Simulation

Group 15:
Daniel Davis
Yang
Steven Fisher
Gegi
Jose Villamor-Delgado
Tanzina Farzana

Table Of Contents

1. Customer Statement of Requirements

1. a. 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 due to the variability of seasons, tides, and 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.

Weather and tides are an important factor in a customer's decision to visit the beach. If it is raining, little to no visitors will come to the beach that day. Rough tides will also detract beachgoers that come for water sports, i.e. surfers.

In addition to weather, customers are also affected by their past experiences. A beachgoer may not visit the same beach again if it was too crowded to be enjoyable on a previous visit. This is a game theory problem that involves predicting 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 this into account.

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 rough tides than adults. Male young adults may prefer to go to beaches where young female adults frequent. The demographics for the simulation should be able to be set individually. For business purposes it is important to know what age groups affect the customer flow. Expanding our target audience to include a certain age group, as well as seeing how that age group effects the simulated crowd, will help with our marketing efforts.

Locations that will display real time population statistics as a default is another feature necessary for this software. This will let us simulate our multiple business locations, and in turn decrease the time it takes to simulate the population. We would like to avoid having to look up our population statistics to input.

When simulating the population, there should be a way to save any settings previously set. By allowing custom settings to be saved, it will reduce the amount of time to run a specific simulation multiple times.

In the simulation, agents (beachgoers) should aim to "win" by having an enjoyable experience at the beach. Their strategy should be based on past wins and future predictions. If an agent continues to lose, it should change its strategy. The simulation will return the simulated population percentage that chose to go to the beach on a given day.

1. b. 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 pick a new strategy if they aren't winning with one.
Self-Optimization: The process where agents will vary their strategy choices dependent on if they win.
Simulator: Running thousands of virtual days of simulations 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, weather, tidal weather, time, date, 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.

2. System Requirements

2. 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 majorit loses
REQ-2	5	Each agent has a personal strategy.
REQ-3	5	Agents with winning strategies repeat the winning strateg Agents with losing strategies may change them.
REQ-4	5	The more recent an Agents win/loss is, the more likely th agent is to consider it in its strategy.
REQ-5	5	The system initial conditions have a default setting.
REQ-6	3	The system default to the current day.
REQ-7	3	Weather forecasts used from the system are taken off of a website's 10 day forecast.
REQ-8	3	Agent action depends heavily on the weather conditions the day.
REQ-9	5	The system updates graphs in real time depending on the users chosen actions.
REQ-10	3	If an area is selected with multiple beaches, it is possible there to be multiple winning beaches.
REQ-11	4	The system runs the simulation multiple times so a good set can be made.
REQ-12	1	Business analytics are included in the system to help beachside businesses
REQ-13	1	Actual area demographics are used to give a more accura prediction.
REQ-14	2	Agent strategies should account for friend,family, and neighbor agents strategies.
REQ-15	2	Some agents should account for tidal forecast.
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, su

		as age.
REQ-18	2	Agents should favor dates that are more convenient for th personal factors (such as various holidays, spring break f students, etc..)
REQ-19	2	Some agents should have the ability to get sickness, preventing them from going to the beach.
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 separate 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.

2. 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 locatio
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	3	System should be compatible with multiple browsers.
REQ-28	2	System should be able to be configured on the admin end for specific users and their needs.
REQ-29	5	System should use good coding practices such as readable code.
REQ-30	5	System should use one standard set of measurements to ensure interoperability.
REQ-31	4	System should have have default values based on the specific user.

REQ-32	4	The user should be able to navigate the application using a maximun three clicks from the root location.
REQ-33	4	The interface should be user friendly, and allow the user to see witho 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.

2. c. Onscreen Appearance Requirement

Identifier	**Priority Weight (Low 1- 5 High)**	**Requirement Description**
REQ-34	5	The user interface takes default values such as population size, weather, etc. based on the location given.
REQ-35	3	The user interface able to check the graph based on data input.
REQ-36	2	The user interface would be able to save data and graph , which the would get from the simulation.
REQ-37	1	Visual Demographics, the data we get from the location GPS.


MONTH:
SUNDAY
MONDAY
TUESDAY
WEDNESDAY
THURSDAY
FRIDAY
SATURDAY
Quartet
NOTES
Factors.
Location:
Beach!
Demographics.
Weather:
Updates w/ any factor changes. or Button.
1/100 people/agent.
Buy more icecream.

3. Functional Requirements Specification

3a. 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).

3b. 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

3.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.

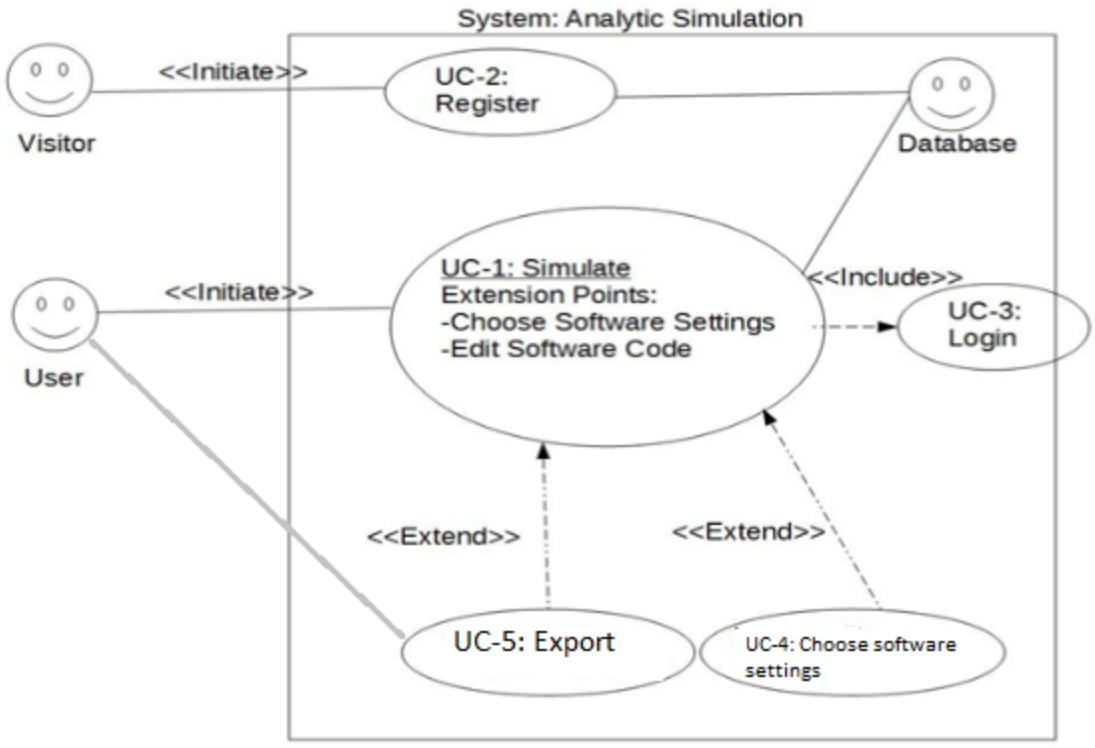
UC-2: Register -- Allows a user to fill out a registration form to obtain access to the software.
Derived from REQ III.

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 seperate file, in the format of their choosing.
Derived from REQ IV.

3.c.ii. Use Case Diagram:



3.c.iii. Traceability Matrix

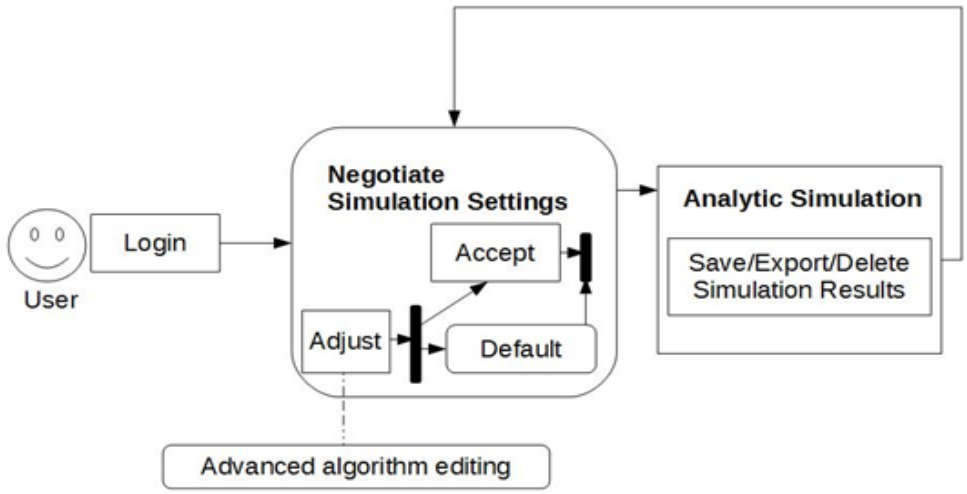
The traceability matrix shows the relationship between the use cases and requirements.

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.

3.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

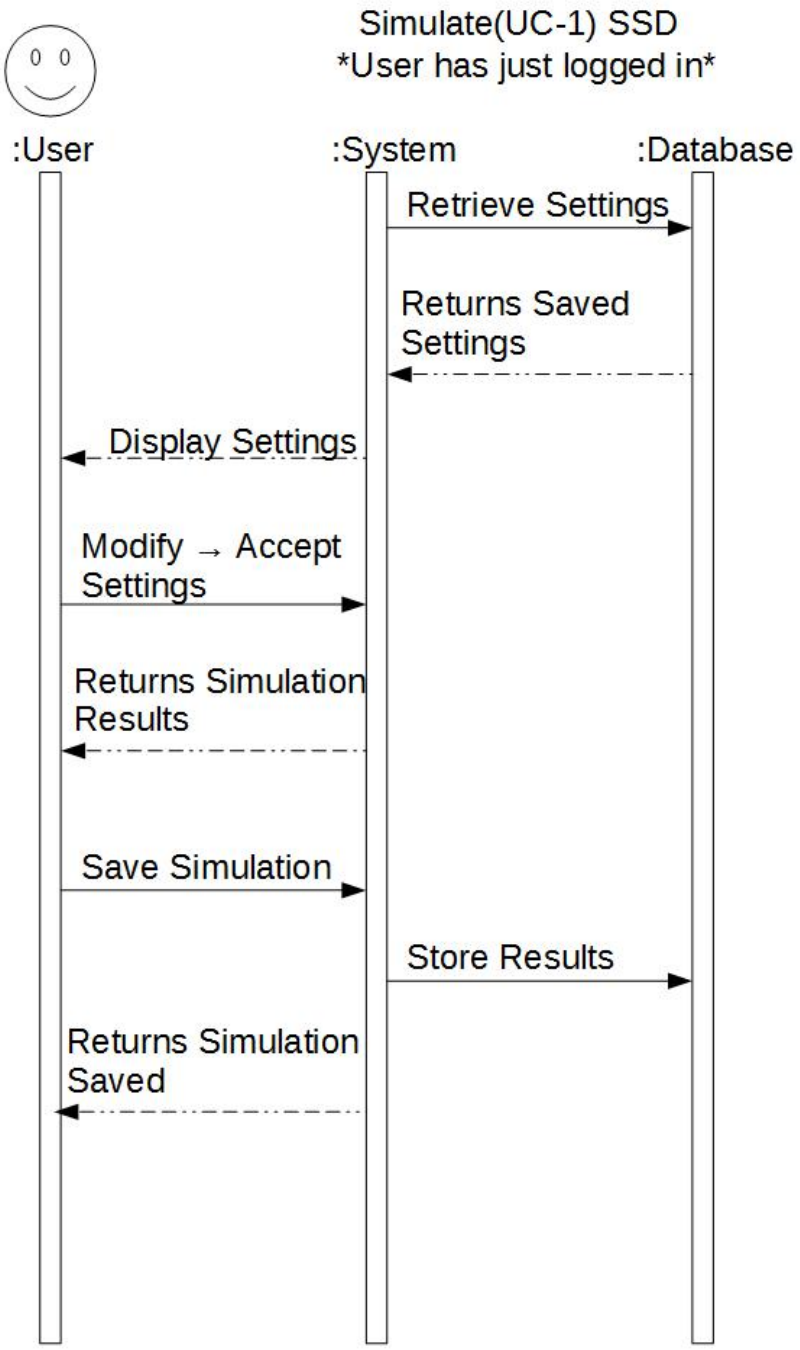
Related Requirements: REQ II
Initiating Actor: User
Actor's Goal: To adjust simulation settings
Participating Actors: Database
Preconditions: User must be logged in
Success End Condition: If parameters are acceptable, settings can be saved
Failed End Condition: If inputted parameters are outside of range, 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:

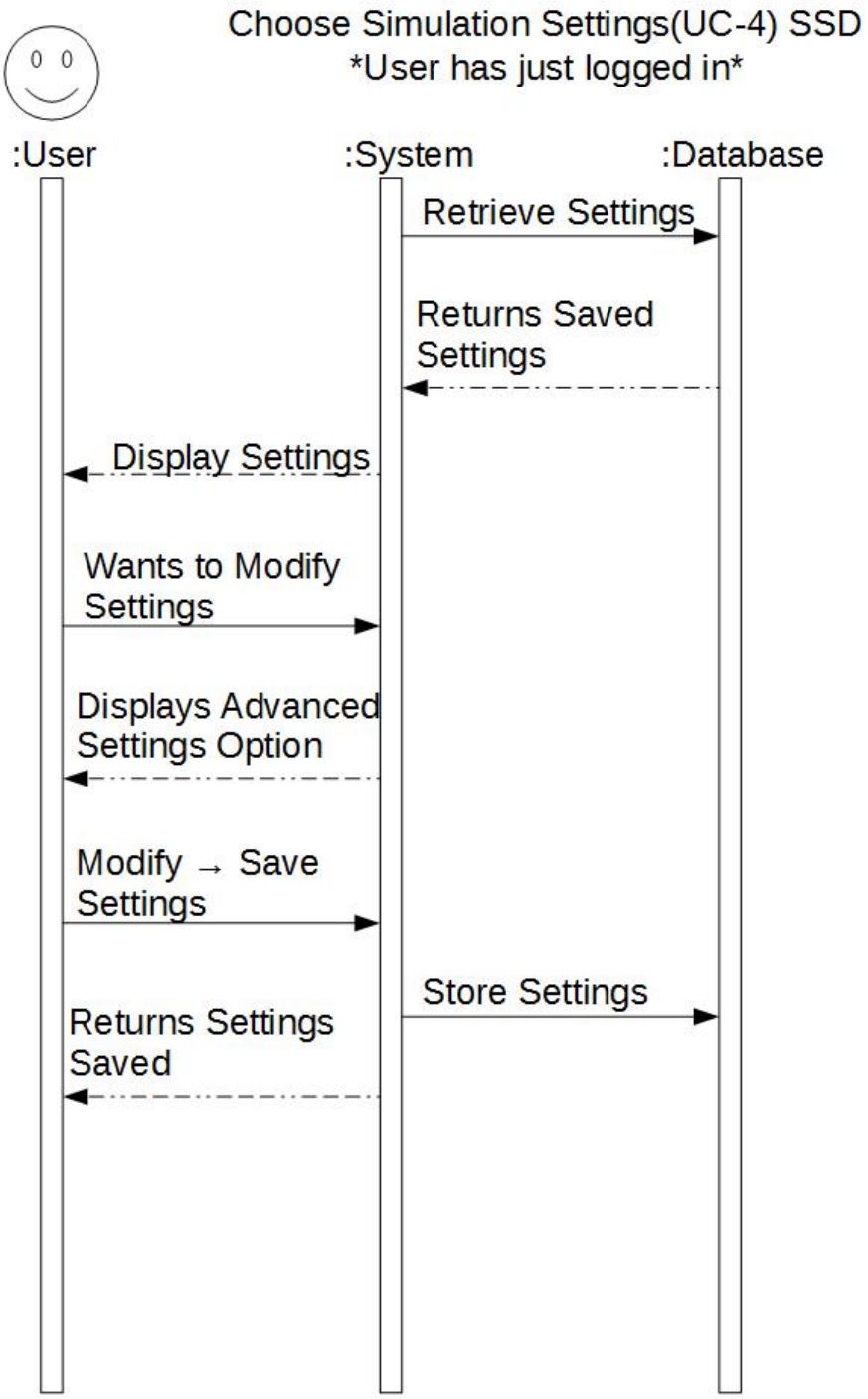
Use Case UC-5: Export

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:

3.d. System State Diagram



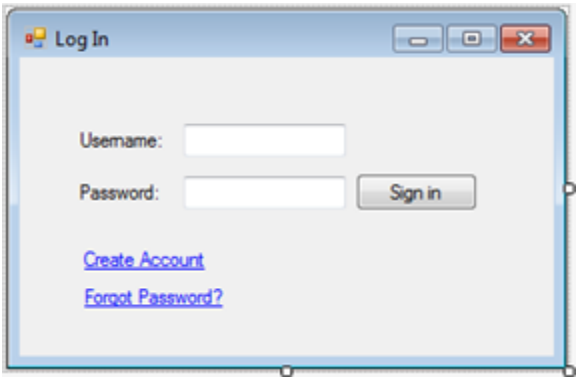
*


Choose Simulation Settings(UC-4) SSD
User has just logged in
:User
:System
:Database
Retrieve Settings
Returns Saved Settings
Display Settings
Wants to Modify Settings
Displays Advanced Settings Option
Modify → Save Settings
Store Settings
Returns Settings Saved

4. User Interface Specification

4. a. Preliminary Design

Initial Screen:

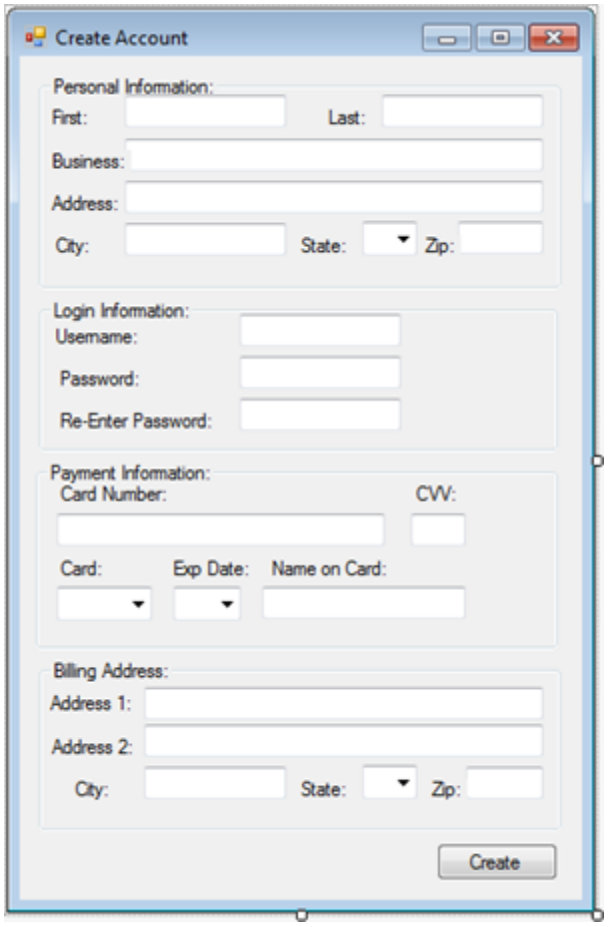


Just like a standard log-in, the user will enter the username (his e-mail) and the password. There are three buttons to click on this initial screen.

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.

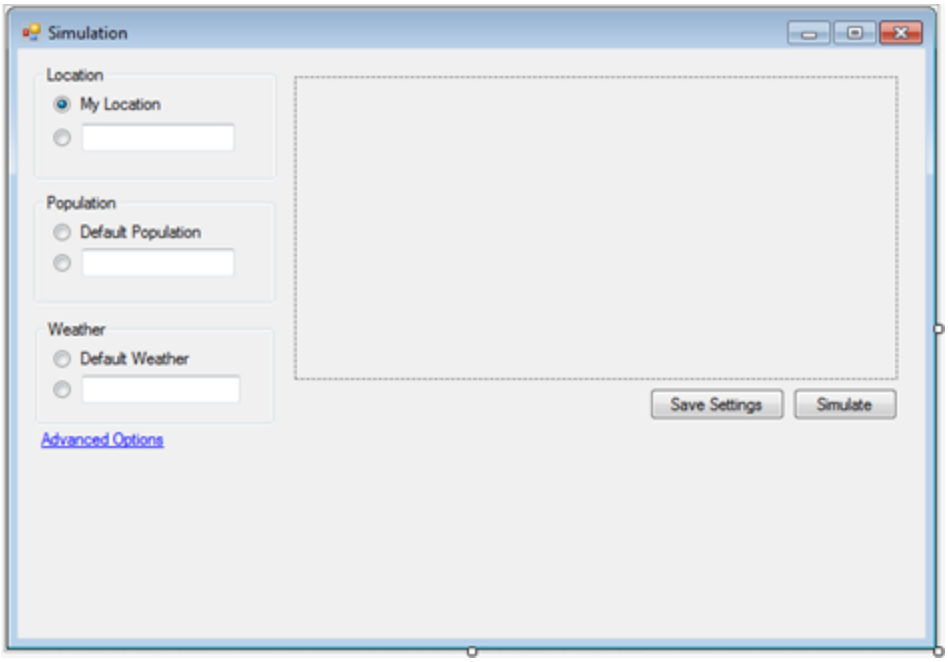


Basic UI:

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.

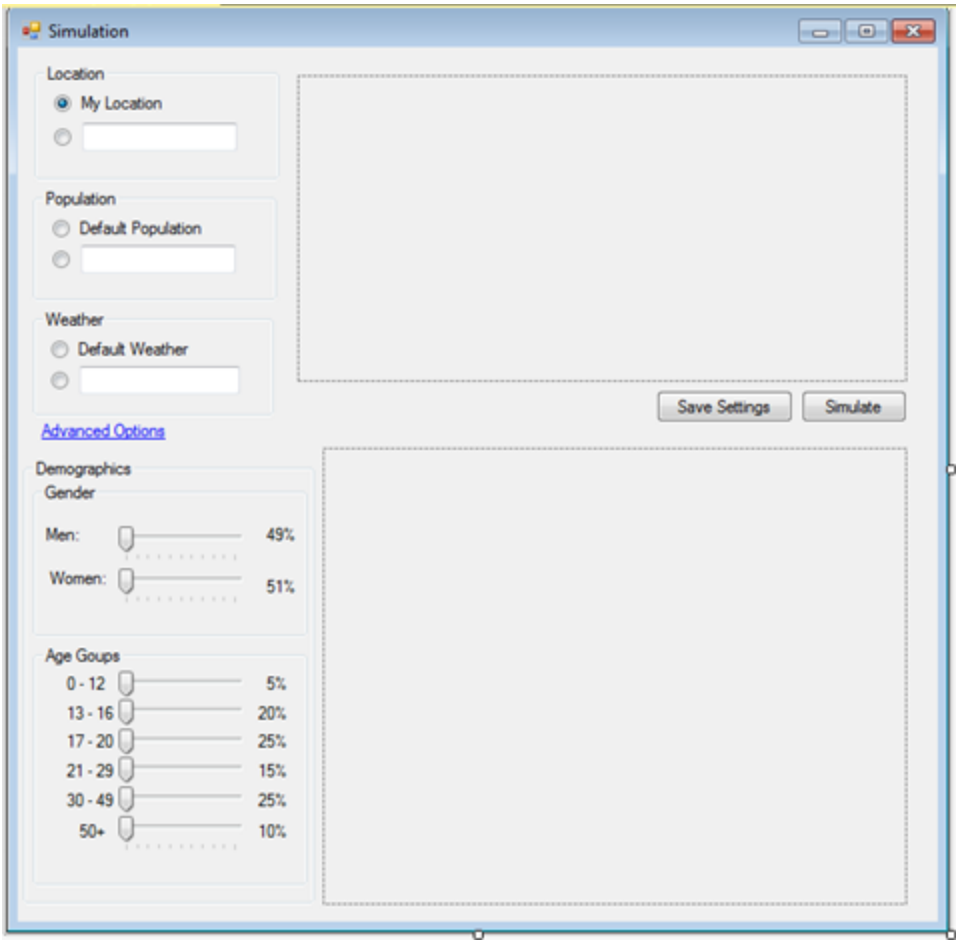
Action clicks:

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:

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.



4b. User Effort Estimation

Non-Registered User:

25 Clicks to Register. Includes typing in the initial information in the sign-on screen and then being directed to the registration page.

Registered User:
It will take a registered user 3 clicks to enter the program.

First time basic simulation:

This will take a user 25 clicks to register and enter, and one click to simulate the basic settings. This would go down to 4 clicks if the user has been previously registered.

Running Advanced options:

A new user would take 25 clicks clicks plus 7 clicks to enter and alter all the demographics, and simulate. A registered user would take 3 clicks plus the 7 to set and simulate a determined 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.

5. 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 discuss below.

Deriving concepts from responsibilities identified in the detailed use cases .

Responsibility Description	Type	Concept Name
Runs the game once user succeeds in inputting correct paramet detailed in UC-1 Main Success Scenario	D	Simulator
If user modifies default conditions, checks if modified conditio are legal	D	Simulator
A simple and effective display for the user to interact and obtai information from the simulation	D	Interface
Registers Visitor into database and the registered Visitor is nov considered a User	K	Database
Stores past simulation results, user registration information and default values	K	Database
Chooses whether to go to the beach or not depending on variou conditions set by the user	K	Agents
Decides whether to change decision in the next round based on previous outcomes	K	Agents
Keeps track of how many agents went to the beach for the current round	K	Beach
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 b the user	D	Interface
Checks if the changes made to the advanced menu are legal	D	Simulator

Displays an error if the changes are not legal	D	Interface
The medium that the user uses to access the simulation online	D	Website
Keeps track of a user's login information, status, and simulatio 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 beac	Beach Tally (Outcome)
InterfaceUser Profile	The interface sends login information to the use profile	Send login
SimulatorUser profile	Updates the information in the account after eac simulation	Update
Simulator Exporter	Sends the simulation output to the exporter	Send output

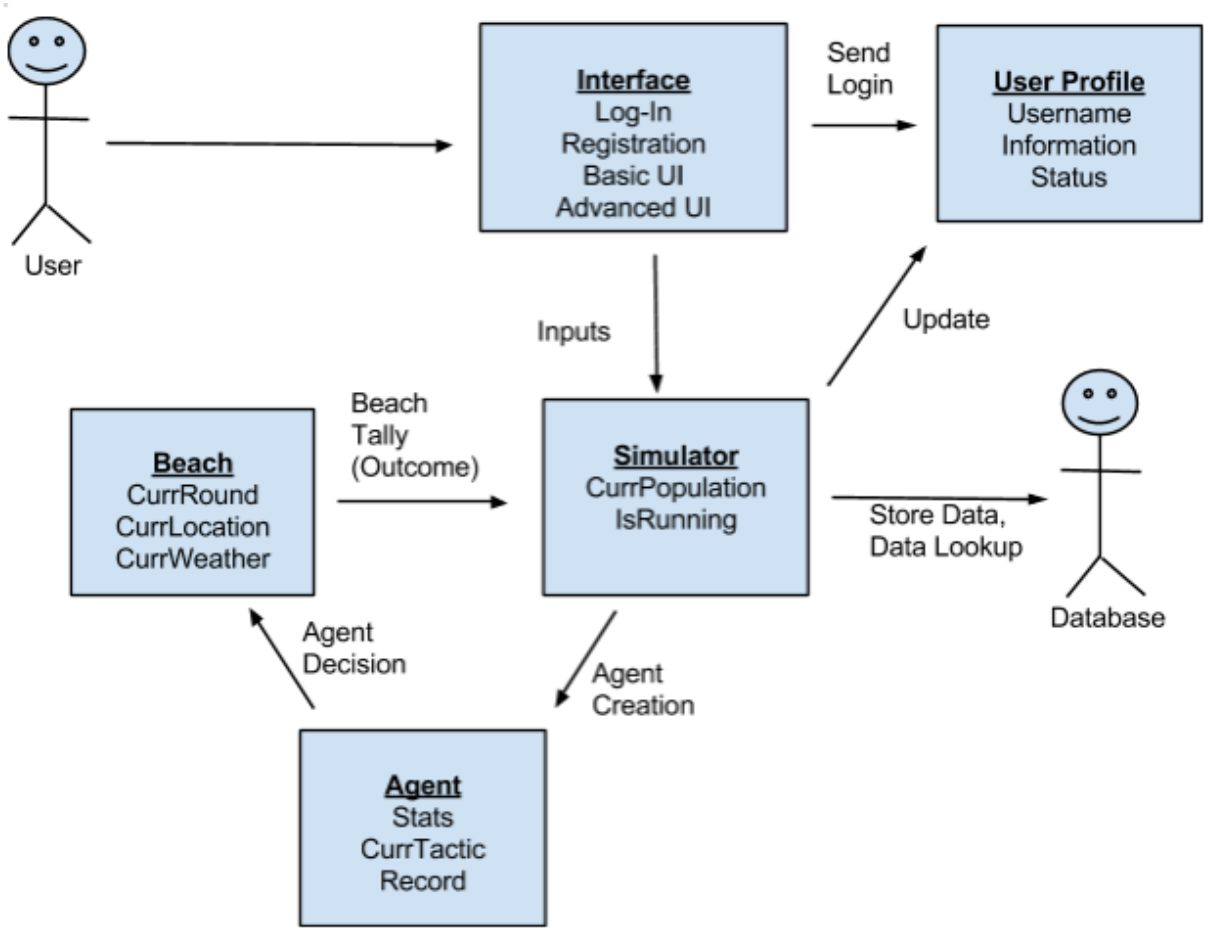
Attribute Definitions

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 simulatic data. Also contains the Advanced UI button.
	Advanced UI	Contains the advanced input interface for users looking for more in-depth control of the simulatic
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 locatio
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 th location given. The number of active agents is based on the population
	IsRunning	Keeps track of whether the game is running or n
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/o

		whether they are running a simulation or not
Exporter	Input	The input to be formatted

5. 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

5. 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.

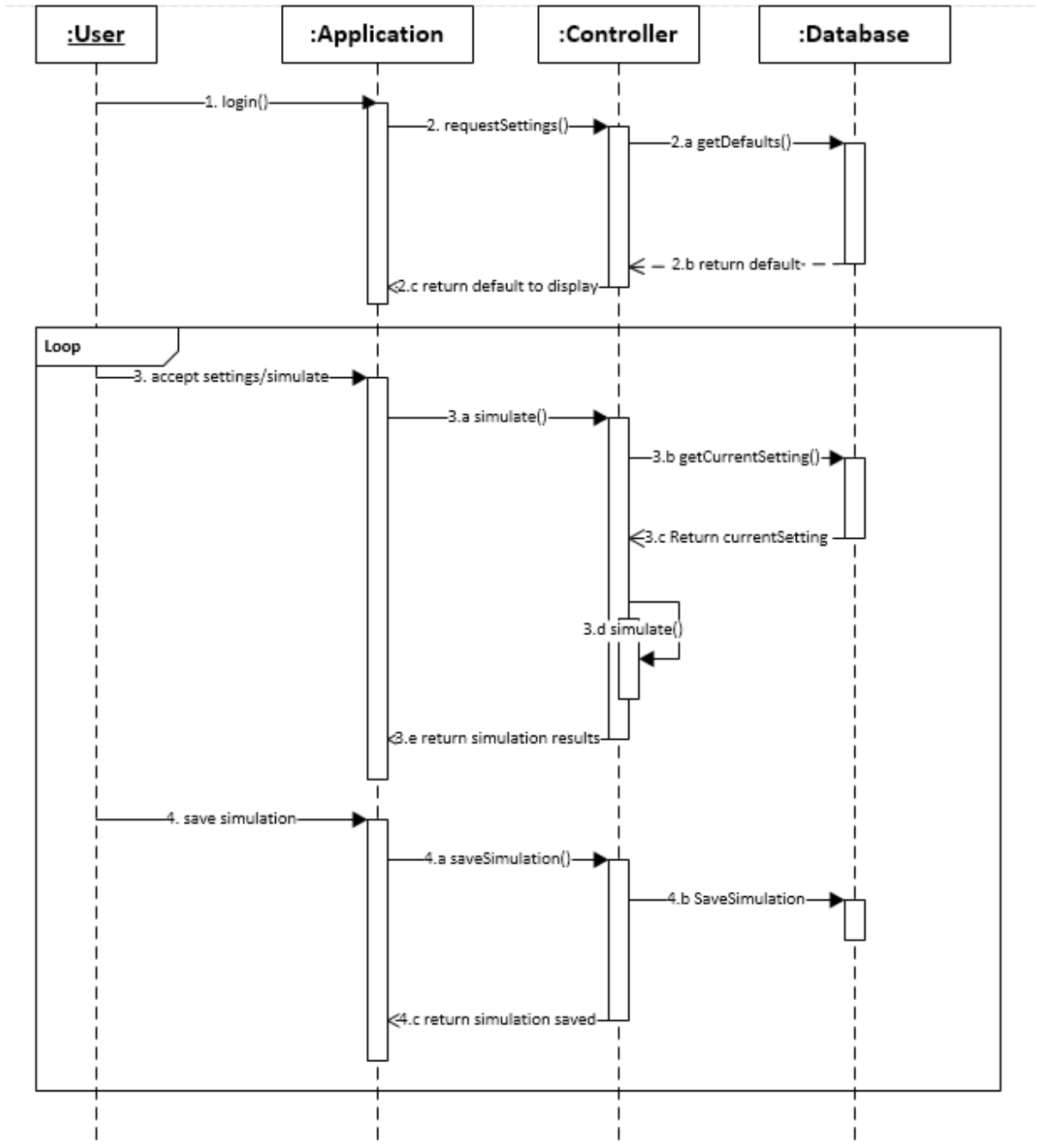
Operation	**Name of the operation and parameter**
Cross reference	UC-4
Precondition	User log in. User can modify inputted default values based on location. User can choose the advance option button.
Postcondition	Data valid, simulate the system. Analysis graphs and visual Demographics. Save the simulation. Store the result..

5.c Mathematical model

A statistical model must be used in order to simulate the business predicted for the day based on the consumer preferences and outside variables as selected by the user. The consumer prefers better weather, sunny, cloudy, over colder weather, rainy, snowy. However, if too many people went to the beach the day before, the consumer will be deterred by the overcrowdedness of the previous day and not be as likely to visit again the next day. The location where the beach is located matters as well, if the beach is too far from a populated area, it is less likely to be crowded all of the time, so the enjoyability factor will go up for the consumers and they will be more likely to visit consistently. The agents represent the consumers, and they each represent a portion of the population whether the agent chooses to go or not is determined by factors including whether a nearby agent is also going. The agent could be influential or very naive and either listen to its neighbors, or make its own decisions.

6. Interaction Diagrams

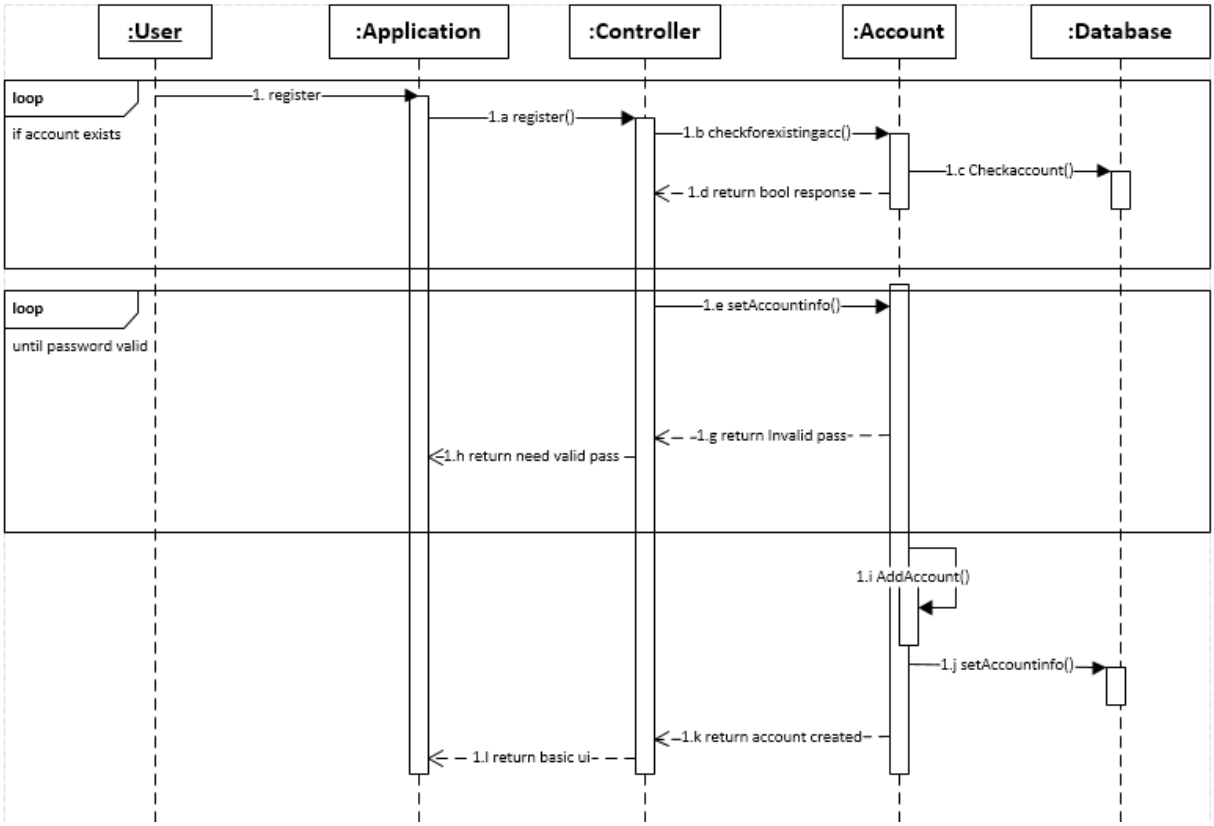
UC-1: Simulate



Description:

The simulator use case simply for settings and then runs the simulation based on settings. The database provides default settings while the user can change them. The simulator runs the simulation, and then sends the user to a menu where they can either simulate again, or save the simulation. They'll also be able to export the simulation, which leads to another use case.

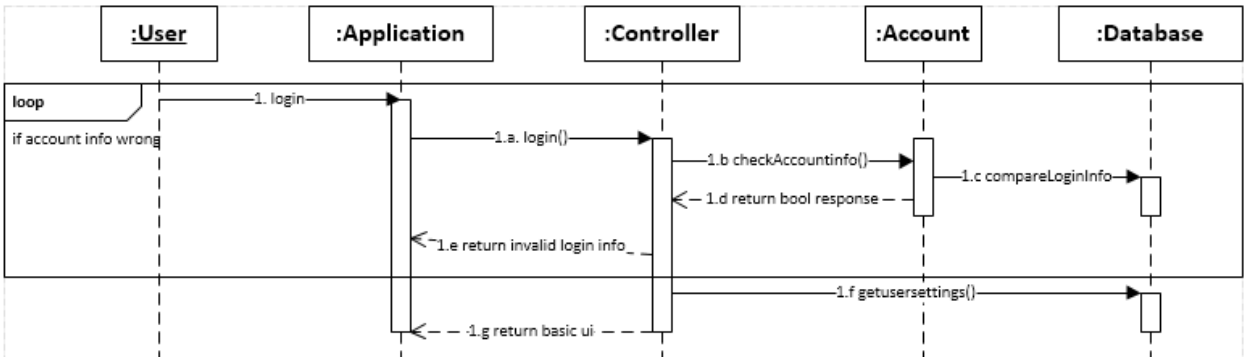
UC-2: Registration



Description:

The user will fill out all information in the form. When they click Register, they will call the register function. This will then send a check to the account list to see if there is a corresponding account already made. If there is there will be an error message sent back to the user, and will loop until all errors have been corrected. If the Boolean response is true, it will then go into making and account list. It will create the structure, and call individul group messages to input data. When setting the password is called, it will then check the validity and loop to the user until the issue is fixed. Then the system will add the account to the account list. The program will then launch the UI to the user.

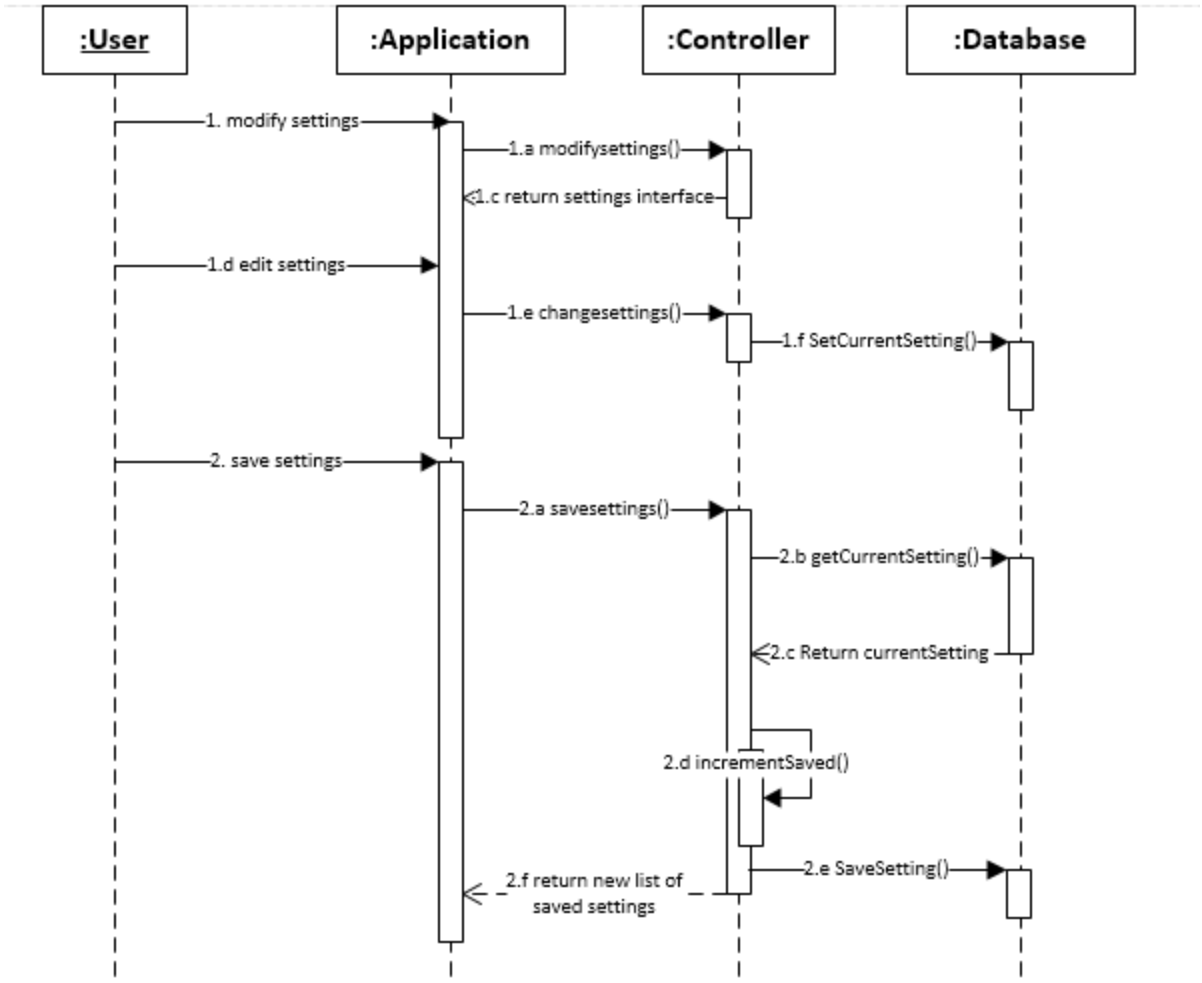
UC-3: Login



Description:

The user attempts to login with user-entered credentials. The application will confirm the login credentials and launch the simulator with the user's stored data. If the the credentials are incorrect, the user is prompted about the failed login. The user can choose to alter the credentials and perform another attempt, or choose to register a new account (covered in UC-2).

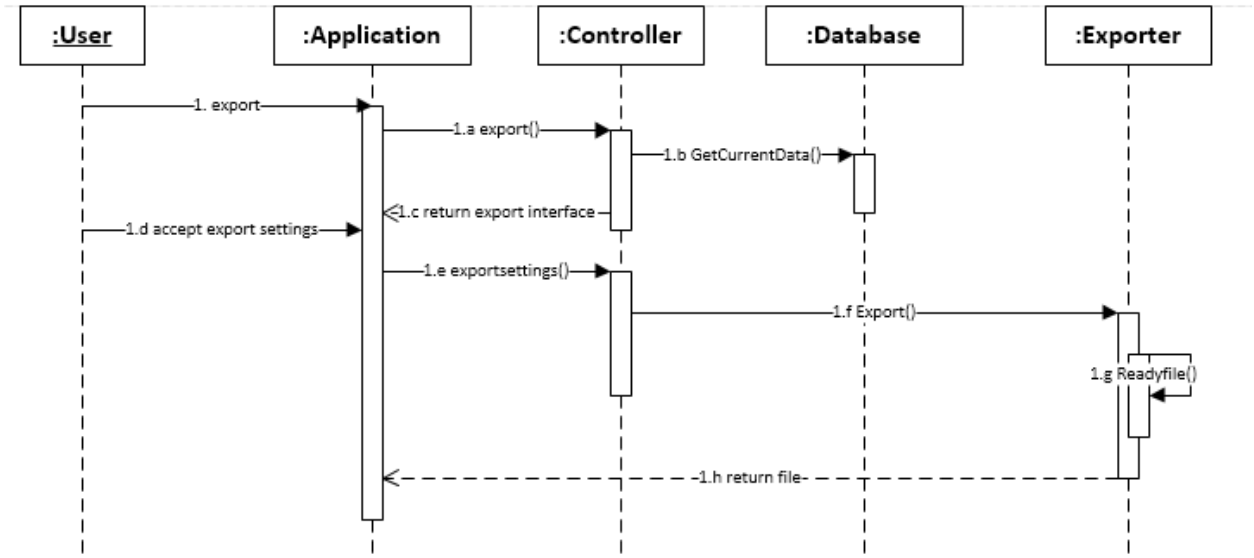
UC- 4: Modify Simulation Settings



Description:

Use case 4 allows the user to modify the settings for the simulation. First the user requests to modify the settings, upon receiving this request the system retrieves the current settings and displays them to the user. At this point, the user can modify the settings. Once the settings are modified, the user must save the settings before the simulation can be run. Saving the settings, gets the current list of settings increments the list of settings, and saves the new setting in the slot. Then the controller returns the new list of settings to the application. Upon which the user can choose their settings from, or run their simulation.

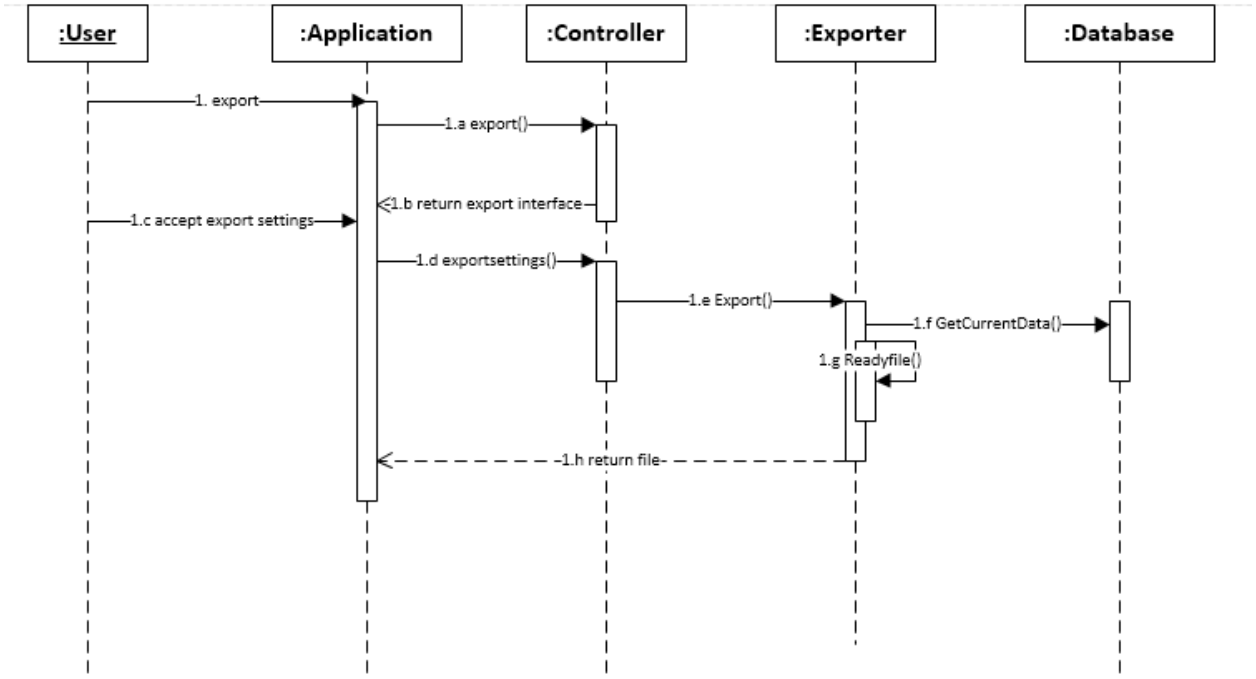
UC-5: Export



Description:

This use case starts after the user finishes the simulation and asks to export. The controller sends export options to the user. Then the user fills the options the controller sends those options to the exporter, along with the simulation data, to be formatted. The exporter then returns the formatted file to the user.

UC-5: Export Alternate Scenario

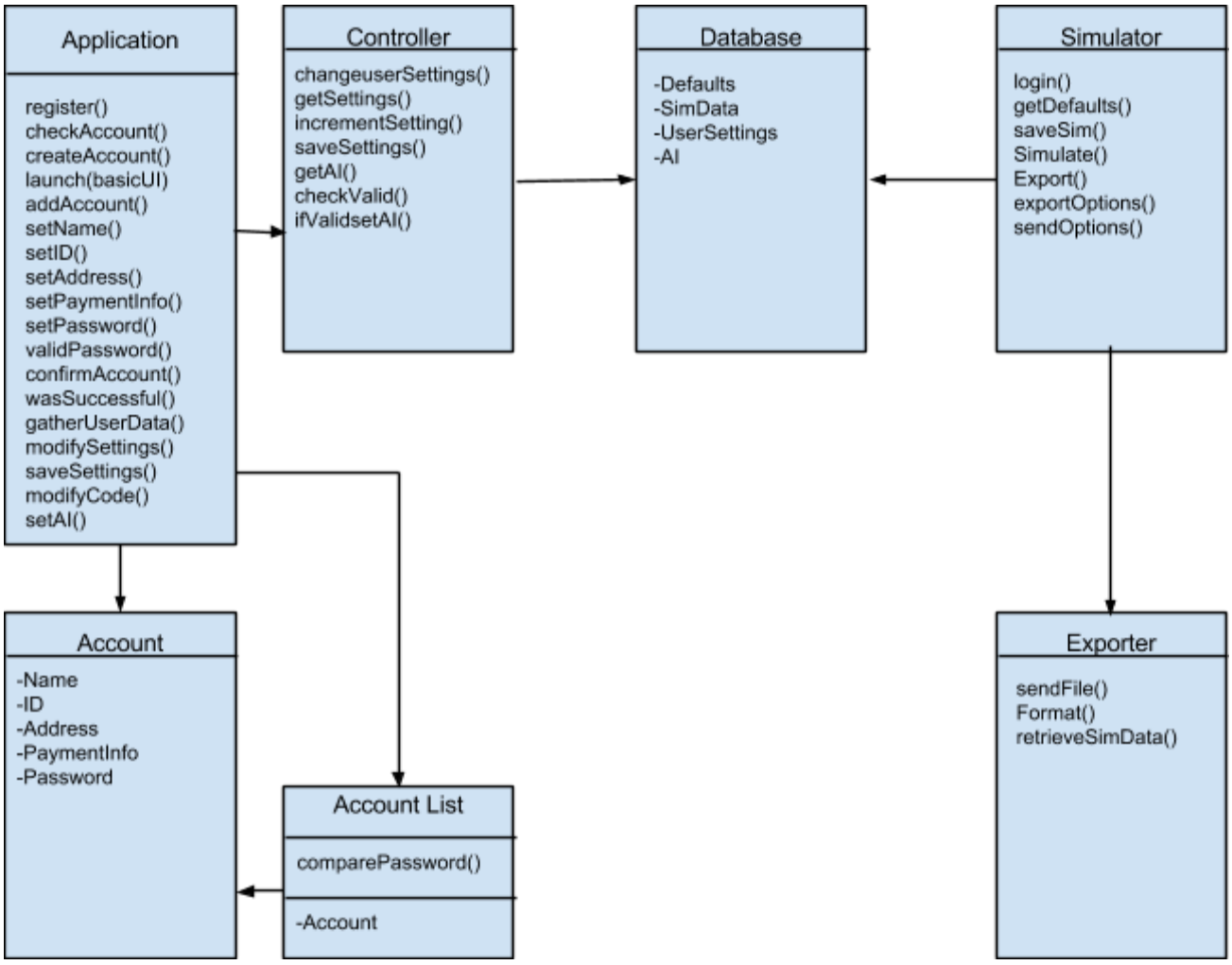


Description:

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.

7. Class Diagrams and Interface Specifications

7a. Class Diagram



7b. Data Types and Operation Signatures

The Application 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 node. 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 node into its sorted spot in the Account List's linked list.

The Account List class is a linked list consisting of account nodes. It also has the attribute comparePassword() which is a second check on the validity and the double input of the password.

The Account class is simply a structure containing one classes information including Name, Address, Password, Email, and Payment Info which are all strings, and ID which is an integer.

The Controller class allows the user to control and retrieve settings from the Database. As such, it has attributes such as: changeuserSettings(), getSettings(), incrementSettings() and saveSettings() to achieve its purpose. The Controller is also responsible for retrieving the AI algorithm that is used in the simulation. Whenever Application calls modifyCode(), the Controller calls getAI() which returns the current AI algorithm. If the user edits this algorithm, Application will call setAI() which attempts to save the new AI to the Database, but before this is allowed, checkValid() is called to make sure the new algorithm is acceptable. Once an AI algorithm is accepted and validated, ifValidsetAI() saves the new algorithm to the database.

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, the Simulator calls exportOptions(). exportOptions() allows the user to specify the format and file type of the return file. Once the user has finished with the options, sendOptions() is called to send the request for a return file to the Exporter class.

The Exporter class is responsible for creating a return file of the simulation data. In addition to the above scenario, the user can, alternatively, access the Exporter class directly. In this case, after exportOptions() has finished, Exporter calls retrieveSimData() to retrieve the current results of the simulation from the Simulator. The Exporter then calls Format() to format the data into the specifications that the user specified in exportOptions(). Finally, Exporter calls sendFile() to output the file to the user. retrieveSimData(), Format() and sendFile() are also used in the scenario described within the Simulator paragraph above.

7c. Traceability Matrix

Domain Concept Vs. Classes	Class	Class	Class	Class	Class	Class
Concept	Application	Account	Controller	Database	Simulator	Exporter
Registration	x	x		x		
Login	x	x		x		
Simulation				x	x	
Mod Simulation Settings	x		x	x		
Export					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 own simulation results within the database.

The Simulation concept is in charge of the entire simulation. It creates (or takes defaults from the database) and manages the agents, towns, beaches and any other variables within the simulation. It also saves simulator results into the database for accounts to access later.

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

The Export concept is simply a concept to allow users to export their results in a format of their choosing directly from the simulation or the database.

8. System Architecture and System Design

---

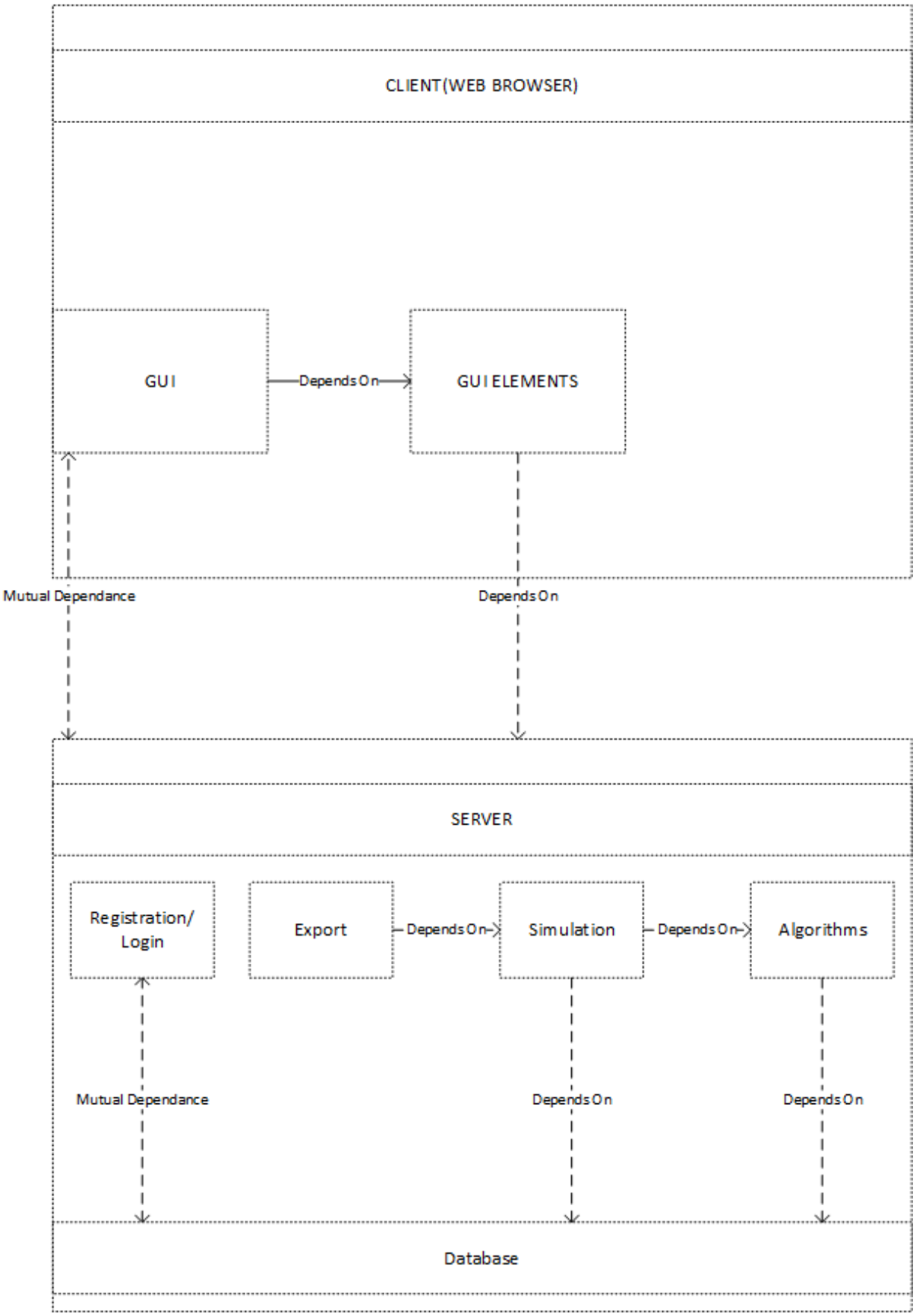
8. a. Architectural Style

The architectural styles that we will be using are the client server model and the event driven model. The client server model is a model where the client interacts with an interface that then processes the information from the client and sends that information back to the server to run. At which point the server sends the results of the simulation back to the client. Our architecture is event driven as well because the user must interact with the client (the web browser) in order to drive the events (simulation)

that interact with the server. An example of this is, the user clicking simulate triggers an event, the simulation, however, the simulation runs on the server, so the event trigger notifies the server that it must run the simulation. At which point the server pulls up the settings and runs the simulation, then saves the results of the simulation and sends the results back to the client (browser) for the user. Another example is the registration/login use case for our system. In this scenario, the user clicks the option to register, this triggers the registration event which pulls the registration page from the server, and sends it to the client. This registration form must then be filled out by the user, when the user is done with the form, the user clicks register, at this time, the client sends the filled out form to the server. The server must then compare the registration email with the current database for any duplicates. If duplicates exist the registration must be refused, and the corresponding information will be sent from the server back to the client.

8. b. Subsystems

The major subsystems of our application is the client (the web browser) which includes the GUI. The client communicates with the server which houses our other subsystems. The server contains the simulation and registration/login subsystems which must interact with the database subsystem in order to run. The export subsystem relies on the simulation results, and therefore simulation. The database subsystem houses all of the information that is stored from the other subsystems for future reference by those subsystems. The simulation subsystem interacts with the algorithm subsystem in order to run the simulation subsystem.


CLIENT(WEB BROWSER)
GUI
Depends On
GUI ELEMENTS
Mutual Dependance
Depends On
SERVER
Registration/ Login
Export
Depends On
Simulation
Depends On
Algorithms
Mutual Dependance
Depends On
Depends On
Database

8. c. Subsystems and Hardware

Since our system is browser based, the client subsystem should be able to run on any user machine through their web browser. The server subsystem, however, will exist on the network servers online.

8. d. Persistent Data Storage

Persistent data storage exists through the database. Each simulation is saved in the database as a text file, which can be loaded onto the client GUI. This data once it reaches the screen is processed into graphs, etc. At this point, the user can choose to either export that data directly using the text file or export it graphically as in with a pdf. The user base is saved in the system database and is used as a comparator whenever someone tries to login and register.

8. e. Network Protocol

For our system, we plan on using just plain http for the protocol, the requests should be sent in a text file to the server. In this case the major changes to the settings are what requires the sending of the text files. This includes the algorithms, registration form entries, login form, and the slider settings. For the simulation itself, what will be sent is just the request for the simulation to be run. We choose this format because the system will be browser based, and we don't plan on running through a platform such as java.

8. f. Global Flow Control

i. Execution orderness

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 run with the next instance of the simulation.

ii. Time dependency

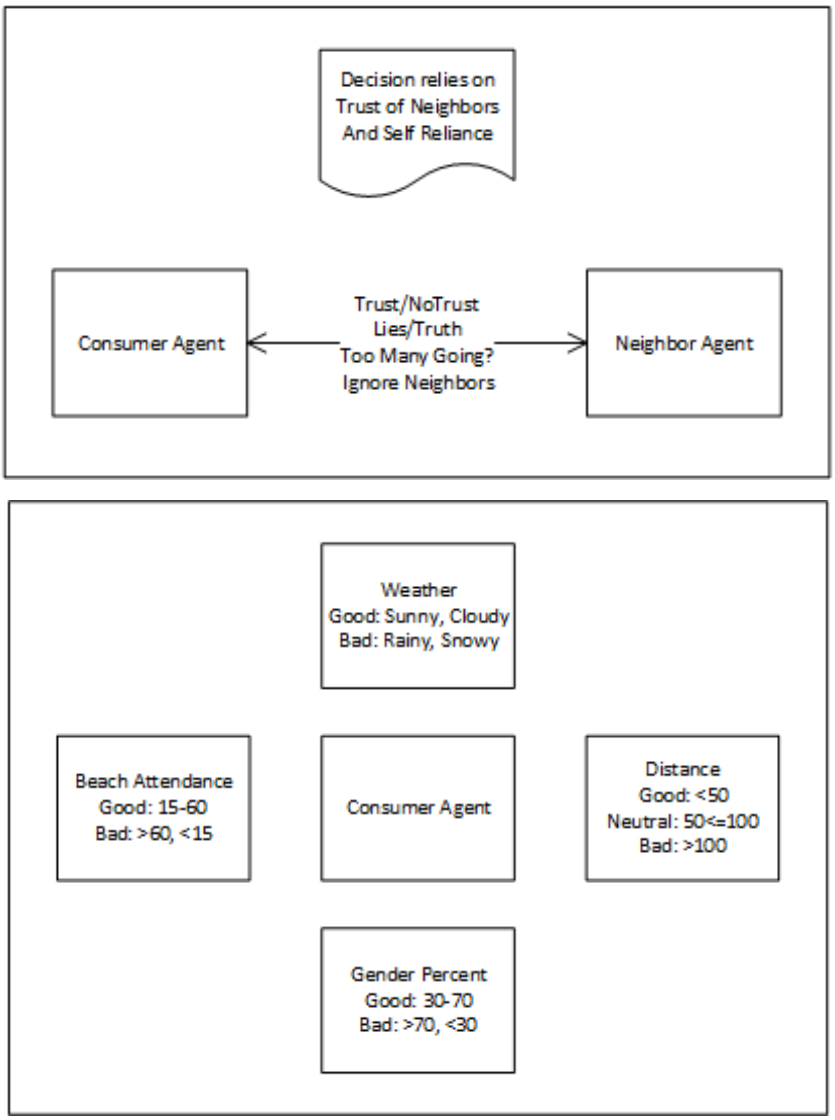
There are no timers on the system. The system does however, take the different times and instances of the year into account when simulating the events that are expected to take place.

8. g. Hardware Requirements

	Minimum	Recommended
Display Resolution	640x480 color display	----------
Browser	Supports latest javascript implementations	----------
Operating System	Any	----------
Network	56Kbps	1Mbps

9. Algorithms and Data Structures

9. a. Algorithms



9. b. Data Structures

The application will be using a 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. The type of database chosen is essentially a collection of tables associatively linked to each other in order to allow fast access and search.

10. User Interface Design and Implementation

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.

11. Design of Tests

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

Initialization Error Messages
Unit to test: GUI Input
Assumptions: The program has displayed the input screen and is waiting for user action.
Test data: Invalid data values in each field
Steps to be executed:
(1)Input invalid values into each test field
(2)Check to see that a red exclamation point shows up next to the field
Expected result: For any invalid value in a field, a red exclamation point should appear next to said field and when cursor is hovered a bubble describing error appears
Pass/Fail: Passes if all fields return an error message. Fails if any fields accept invalid input or doesn't have an exclamation point next to them.
Comments: This test is to make sure the GUI interaction of the user handles errors well.

Run 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) Check to make sure no exclamation points exist next to input fields
(2)Press the Simulate 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. Fails if anything else occurs.
Comments: This test makes sure that the data will be visually accessible to the user.

Chart List
Unit to test: Chart drop down menu
Assumptions: The new window popped up from clicking Simulate button
Test data: Items in drop down
Steps to be executed:
(1) Click the right drop down and select a chart.
(2) Click the left drop down and select a chart.
Expected result: New charts visibly appear and are updated in real time.

Pass/Fail: Passes if new chart appears and is updated in real time. Fails if anything else occurs.
Comments: This test makes sure that the chart functionality works.

Start Simulation Button
Unit to test: Run Simulation Button/Function
Assumptions: The new window popped up from clicking the Simulate button
Test data: Charts, Run simulation button, Advance option slider
Steps to be executed:
(1) Press Run simulation button
(2) Change charts using drop down
(3) Move Advance option slider to an arbitrary amount to the right
Expected result: Charts are visible and are updating in real time. When a new chart is selected, a new chart appears. When the slider is moved to the right, the rate of demo graph for the charts slows down
Pass/Fail: Passes if expected results are met. Fails if anything else occurs.
Comments: This test makes sure that our data will be visually accessible to the user as well as test the backend. This test case is the most important, as it encompasses all test cases and the backend

Start/Stop Simulation Button
Unit to test: GUI Stop/Resume Input
Assumptions: The simulation is currently running with valid data having been input into the program.
Test data: Mouse Click
Steps to be executed:
(1) Click on the pause button on the GUI
(2) Check to make sure the simulation has ceased running
(3) Click on the same button again
(4) Check to make sure the simulation has resumed
Expected result: The simulation should cease running and all data creation should halt. The simulation should then resume where it left off.
Pass/Fail: Passes if the system exits its run functions and stops updating graphs, then the system starts its run functions and updates graphs. Fails if system never stops updating graphs or never resumes updating graphs.
Comments: This test should be rather easy to satisfy because of the ease in which a computer can be asked to exit a loop. This logic is therefore simple and the test should only fail when somehow the button press is disassociated from its responding function in the code.

Change Advance option Slider

Unit to test: GUI Slider Button

Assumptions: The simulation is currently running with valid data having been input into the program

Test data: Results from a successful simulation

Steps to be executed:

(1)Move the slider one way or the other depending upon its current position

1a. One should notice the simulation slow down if one has moved the slider to the left

1b. One should notice the simulation speed up if one has moved the slider to the right

(2) Move the slider back to its original position

(3) One should notice the return of the simulation to the same execution speed as before step 1.

Expected result: The speed at which the simulation executes should change according to the direction in which it is slid.

Pass/Fail: The test is passed if moving the slider to the left results in the slowing down of the execution of the program and moving the slider to the right results in the speeding up. If any other result occurs, the test is failed.

Comments: This adds a user friendly option to the interface in that it allows the user to slow down the simulation and watch as the data is generated right before their eyes. This may allow the user to pick up on certain patterns that might otherwise be hard to see when looking at the complete data set all together

Data Retention

Unit to test: Output Data function

Assumptions: A simulation has been run and data is ready to be written/recorded.

Test data: The text file that should be returned by the output function in the program

Steps to be executed:

(1) Enter a name in the Output File Name text box.

(2) Run Simulation

(3) Check to see if a file exists with entered name from step 1 in the directory of the running program

(4) Open the text file and verify data inside

Expected result: The file that was generated should be stored and be readable

Pass/Fail: This test is passed if the data exists inside the output file and is human readable. This test is failed if the data isn't readable or there is no data inside the file.

Comments: This test is important in that it assures the user's time has not been wasted in running the simulation and assuring the retention and preservation of the data generated

Strategy

Unit to test: Strategy method
Assumptions: Data has been inputted and is valid.
Test data: Inputted user data
Steps to be executed:
(1) Run test code
(2) Read cout statements and verify its correctness
Expected result: The outputted data is correct within its context
Pass/Fail: Passes is outputted data is valid, fails if outputted data does not make sense or is nonexistent.
Comments: This test solely tests the strategy function in the backend. See unit testing for code.

Location

Unit to test: Town method
Assumptions: Data has been inputted and is valid.
Test data: Inputted user data
Steps to be executed:
(1) Run test code
(2) Read cout statements and verify its correctness
Expected result: The outputted data is correct within its context
Pass/Fail: Passes is outputted data is valid, fails if outputted data does not make sense or is nonexistent.
Comments: This is the most important test because it contains all other classes. See unit testing for code.

Weather

Unit to test: Weather Condition
Assumptions: Data has been inputted and is valid.
Test data: Inputted user data
Steps to be executed:
(1) Run test code
(2) Read cout statements and verify its correctness
Expected result: The outputted data is correct within its context
Pass/Fail: Passes is outputted data is valid, fails if outputted data does not make sense or is nonexistent.
Comments: This is the most important test because it contains all other classes. See unit testing for code.

12. Project Management and Plan of Work

12. 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

12 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.

12. c. Plan of Work

For the project the entire group decided to split up work so we could each do our parts in parallel. The whole team was divided into 3 subgroups (Yang+Sarid, Danny+Steven, Gegi+Jose+Tanzina) and during our meeting we decided on what each subgroups should do. The subgroups split up their work individually. There were some problems though. Sarid could not be reached. He did very little for part 1 and did not contribute at all for part 2. There were also plenty of times where subgroups helped each other out, as we ran into each other doing the projects at the same time in the online file, so contributions got a little blurred. In the end everyone except Sarid did their fair share of the work.

For this report we decided to make a change to our use cases. We removed the old use case #5, which allowed the user to change the internal algorithms of our simulation. We felt that it was a useless use case and that option should not be available to the user. Instead we replaced it with a new use case, the export use case, which let us give the user a portable file of the output.

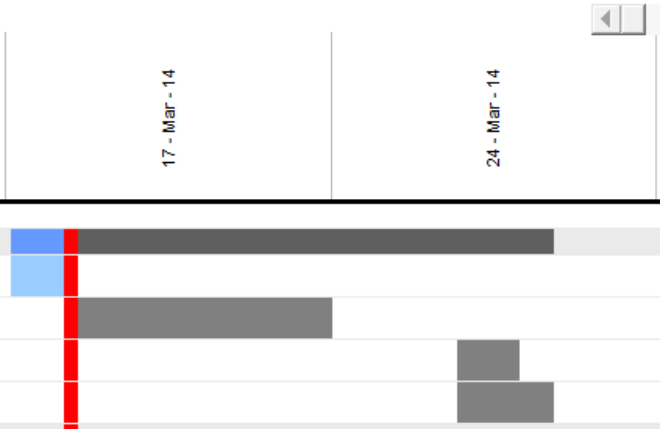
The next project section due is the demo, which requires us to code the first part of the project.

Ideally the group will be split to handle different modules of the code. If Sarid does not decide to help with the demo, then Yang, Danny, and Steven will work on getting the main simulator working, while Gegi, Jose, and Tanzina will work on the other smaller parts of the project, such as the login system, exporter, and database. What each individual person will do will be decided by the subgroups. The plan is a little uncertain as Sarid's contribution is unlikely based on his past performance.

A basic representation of our schedules for the first demo via grant chart:

Project Lead: Daniel Davis
Today's Date: 3/19/2014 (Wed) (vertical red line)
Start Date: 3/19/2014 (Wed) *First Day of Week (Sun=1):* 2

WBS	Tasks	Task Lead	Start	End	Duration (Days)	% Complete	Working Days	Days Complete	Days Remaining
1	**Write the demo**	Daniel Davi	3/18/14	3/28/14	10	17%	9	1	9
1.1	Split work into groups		3/18/14	3/20/14	2	100%	3	2	0
1.2	Start work on project		3/19/14	3/24/14	5	0%	4	0	5
1.3	Edit project and fix bugs		3/25/14	3/27/14	2	0%	3	0	2
1.4	Prepare presentation		3/25/14	3/28/14	3	0%	4	0	3



12. d. 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 (18 points)	**26%**	**20%**	**14%**		**20%**		**20%**
Sec 1: Interaction Diagram (30 points)	**16.7%**	**16.7%**	**16.7%**	**16.5%**	**16.7%**		**16.7%**
Sec 2: Class Diagram and Interface Specification (10 points)		**33.3%**			**33.3%**		**33.4%**
Sec 3: System Architecture and System Design (11 points)				**100%**			
Sec 4: Algorithm & Data Structure (2 points)				**100%**			
Sec 5: User Interface Design & Implementation (11 points)	**75%**		**25%**				
Sec 6: Design of Test (18 points)		**33.4%**			**33.3%**		**33.3%**
Total Contribution	**17.94%**	**17.94%**	**10.28%**	**17.95%**	**17.94%**		**17.94%**

*Our Internal Group Split

*Group 1: Yang, Sarid

*Group 2: Danny, Steven

*Group 3: Tanzina, Jose, Gegi

13. 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